Randomized methods for computing the Singular Value Decomposition (SVD) of very large matrices

Gunnar Martinsson
The University of Colorado at Boulder

Students:

Adrianna Gillman

Nathan Halko

Sijia Hao

Patrick Young

Collaborators:

Edo Liberty (Yahoo research)

Vladimir Rokhlin (Yale)

Yoel Shkolnisky (Tel Aviv University)

Joel Tropp (Caltech)

Mark Tygert (Courant)

Franco Woolfe (Goldman Sachs)

REFERENCE: *Finding structure with randomness: Stochastic algorithms for constructing approximate matrix decompositions* N. Halko, P.G. Martinsson, J. Tropp. Sep. 2009.

Goal: Given an $m \times n$ matrix A, for large m, n (say $m, n \sim 10^6$, or larger), we seek to compute a rank-k approximation, with $k \ll n$ (say $k \sim 10^2$ or 10^3),

$$\begin{array}{ccccccc} \mathsf{A} & \approx & \mathsf{U} & \mathsf{\Sigma} & \mathsf{V}^* & = & \displaystyle\sum_{j=1}^{k} \sigma_j \, \boldsymbol{u}_j \, \boldsymbol{v}_j^*, \\ m \times n & & m \times k & k \times k & k \times n & & \end{array}$$

where

$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k \geq 0$ are the (approximate) singular values of A
$\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_k$ are orthonormal, the (approximate) *left singular vectors* of A, and
$\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k$ are orthonormal, the (approximate) *right singular vectors* of A.

The methods presented are capable of solving other closely related problems:

- Eigenvalue decomposition $\mathsf{A} \approx \mathsf{V} \mathsf{\Lambda} \mathsf{V}^*$.
- Finding spanning columns or rows. $\mathsf{A} \approx \mathsf{C} \, \mathsf{U} \, \mathsf{R}$.
  C consists of k columns of A, and R consists of k rows of A.
- *etc.*

Applications:

- *Principal Component Analysis:* Form an empirical covariance matrix from some collection of statistical data. By computing the singular value decomposition of the matrix, you find the directions of maximal variance.

- *Finding spanning columns or rows:* Collect statistical data in a large matrix. By finding a set of spanning columns, you can identify some variables that "explain" the data. (Say a small collection of genes among a set of recorded genomes, or a small number of stocks in a portfolio.)

- *Relaxed solutions to k-means clustering:* Partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean. Relaxed solutions can be found via the singular value decomposition.

- *Eigenfaces:* Form a matrix whose columns represent normalized grayscale images of faces. The "eigenfaces" are the left singular vectors.

- *PageRank:* Form a matrix representing the link structure of the internet. Determine its leading eigenvectors. Related algorithms for graph analysis.

There is no claim that linear algebra is a panacea! There of course are many environments where other methods (graphs, combinatorics,...) perform better.

One reason that linear approximation problems frequently arise is that it is one of the large-scale global operations that we *can* do.

Many non-linear algorithms can only run on small data sets, or operate locally on large data sets. Iterative methods are common, and these often involve linear problems.

Another approach is to recast a non-linear problem as a linear one via a transform or reformulation (as is done, *e.g.*, in the *diffusion geometry* approach).

Excellent algorithms for computing SVDs exist, but many of them are not well suited for an emerging computational environment where *communication* is the bottleneck. Complications include:

- *Multi-processor computing.*
  CPU speed is growing slowly, but processors get cheaper all the time.

- *Communication speeds improve only slowly.*
  Communication between different levels in memory hierarchy, latency in hard drives, inter-processor communication, *etc.*

- *The size of data sets is growing very rapidly.*
  The cost of slow memory is falling rapidly, and information is gathered at an ever faster pace — automatic DNA sequencing, sensor networks, *etc.*

  – From a numerical linear algebra perspective, an additional problem resulting from increasing matrix sizes is that noise in the data, and propagation of rounding errors, become increasingly problematic.

The power of randomization in the modern context has been observed before:

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 (2006a, 2006b, 2006c, 2006d, *etc*)

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)

K. Clarkson, D. Woodruff (2009)

For details, see review (Halko/Martinsson/Tropp 2009)

Goal (restated):

Given an $m \times n$ matrix A we seek to compute a rank-k approximation

$$\begin{array}{ccccccc} \mathsf{A} & \approx & \mathsf{U} & \mathsf{\Sigma} & \mathsf{V}^* & = & \displaystyle\sum_{j=1}^{k} \sigma_j \, \boldsymbol{u}_j \, \boldsymbol{v}_j^*, \\ m \times n & & m \times k & k \times k & k \times n & & \end{array}$$

where U and V are orthogonal matrices holding the left and right (approximate) *singular vectors* of A, respectively, and where $\mathsf{\Sigma}$ is a diagonal matrix holding the (approximate) *singular values* $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k \geq 0$.

Recall:

Once you have the SVD, "any" other standard factorization is easily obtained.

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

Algorithm:

1. Draw an $n \times k$ Gaussian random matrix Ω.

2. Form the $m \times k$ sample matrix $\mathsf{Y} = \mathsf{A}\,\Omega$.

3. Form an $m \times k$ orthonormal matrix Q such that $\mathsf{Y} = \mathsf{Q}\,\mathsf{R}$.

4. Form the $k \times n$ matrix $\mathsf{B} = \mathsf{Q}^*\,\mathsf{A}$.

5. Compute the SVD of the small matrix B: $\mathsf{B} = \hat{\mathsf{U}}\,\Sigma\,\mathsf{V}^*$.

6. Form the matrix $\mathsf{U} = \mathsf{Q}\,\hat{\mathsf{U}}$.

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

Algorithm:

1. Draw an $n \times k$ Gaussian random matrix Ω.

   ```
   Omega = randn(n,k)
   ```

2. Form the $m \times k$ sample matrix $\mathsf{Y} = \mathsf{A}\,\Omega$.

   ```
   Y = A * Omega
   ```

3. Form an $m \times k$ orthonormal matrix Q such that $\mathsf{Y} = \mathsf{Q}\,\mathsf{R}$.

   ```
   [Q, R] = qr(Y)
   ```

4. Form the $k \times n$ matrix $\mathsf{B} = \mathsf{Q}^*\,\mathsf{A}$.

   ```
   B = Q' * A
   ```

5. Compute the SVD of the small matrix B: $\mathsf{B} = \hat{\mathsf{U}}\,\Sigma\,\mathsf{V}^*$.

   ```
   [Uhat, Sigma, V] = svd(B)
   ```

6. Form the matrix $\mathsf{U} = \mathsf{Q}\,\hat{\mathsf{U}}$.

   ```
   U = Q * Uhat
   ```

Single pass algorithms:

A is symmetric:
Generate a random matrix Ω.
Compute a sample matrix Y.
Find an ON matrix Q such that $\mathsf{Y} = \mathsf{Q}\,\mathsf{Q}^*\,\mathsf{Y}$.
Solve for T the linear system $\mathsf{Q}^*\,\mathsf{Y} = \mathsf{T}\,(\mathsf{Q}^*\,\Omega)$.
Factor T so that $\mathsf{T} = \hat{\mathsf{U}}\,\Lambda\,\hat{\mathsf{U}}^*$.
Form $\mathsf{U} = \mathsf{Q}\,\hat{\mathsf{U}}$.
Output: $\mathsf{A} \approx \mathsf{U}\Lambda\mathsf{U}^*$

Note: With B as on the previous slide we have $\mathsf{T} \approx \mathsf{B}\,\mathsf{Q}$ (sym. case) and $\mathsf{T} \approx \mathsf{B}\,\mathsf{W}$ (nonsym. case).

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

Single pass algorithms:

A is symmetric:	**A is not symmetric:**
Generate a random matrix Ω.	Generate random matrices Ω and Ψ.
Compute a sample matrix Y.	Compute sample matrices $Y = A\,\Omega$ and $Z = A^*\,\Psi$.
Find an ON matrix Q such that $Y = Q\,Q^*\,Y$.	Find ON matrices Q and W such that $Y = Q\,Q^*\,Y$ and $Z = W\,W^*\,Z$.
Solve for T the linear system $Q^*\,Y = T\,(Q^*\,\Omega)$.	Solve for T the linear systems $Q^*\,Y = T\,(W^*\,\Omega)$ and $W^*\,Z = T^*\,(Q^*\,\Psi)$.
Factor T so that $T = \hat{U}\,\Lambda\,\hat{U}^*$.	Factor T so that $T = \hat{U}\,\Sigma\,\hat{V}^*$.
Form $U = Q\,\hat{U}$.	Form $U = Q\,\hat{U}$ and $V = W\,\hat{V}$.
Output: $A \approx U\Lambda U^*$	Output: $A \approx U\Sigma V^*$

Note: With B as on the previous slide we have $T \approx B\,Q$ (sym. case) and $T \approx B\,W$ (nonsym. case).

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

Question: What if the rank is not known in advance?

Answer: It is possible to incorporate error estimators.

- Almost free in terms of flop count.
- Use of an error estimator may increase the pass count.
- Even though they are barely mentioned in this presentation, error estimators are very important! In addition to allowing adaptive rank determination, they greatly improve on the robustness and reliability of randomized methods.

Input: An $m \times n$ matrix A and a target rank k.	
Output: Rank-k factors U, Σ, and V in an approximate SVD $\mathsf{A} \approx \mathsf{U}\Sigma\mathsf{V}^*$.	
(1) Draw an $n \times k$ random matrix Ω.	(4) Form the small matrix $\mathsf{B} = \mathsf{Q}^* \mathsf{A}$.
(2) Form the $n \times k$ sample matrix $\mathsf{Y} = \mathsf{A}\Omega$.	(5) Factor the small matrix $\mathsf{B} = \hat{\mathsf{U}}\Sigma\mathsf{V}^*$.
(3) Compute an ON matrix Q s.t. $\mathsf{Y} = \mathsf{Q}\mathsf{Q}^*\mathsf{Y}$.	(6) Form $\mathsf{U} = \mathsf{Q}\hat{\mathsf{U}}$.

Question:

How much does the computation above cost?

Answer:

Cost of steps 2 and 4: Application of A and A^* to k vectors.

Cost of steps 3,5,6: Dense operations on matrices of sizes $m \times k$ and $k \times n$.

We will consider three proto-typical environments.

Input: An $m \times n$ matrix A and a target rank k.	
Output: Rank-k factors U, Σ, and V in an approximate SVD $\mathsf{A} \approx \mathsf{U}\Sigma\mathsf{V}^*$.	
(1) Draw an $n \times k$ random matrix Ω.	(4) Form the small matrix $\mathsf{B} = \mathsf{Q}^* \mathsf{A}$.
(2) Form the $n \times k$ sample matrix $\mathsf{Y} = \mathsf{A}\Omega$.	(5) Factor the small matrix $\mathsf{B} = \hat{\mathsf{U}}\Sigma\mathsf{V}^*$.
(3) Compute an ON matrix Q s.t. $\mathsf{Y} = \mathsf{Q}\mathsf{Q}^*\mathsf{Y}$.	(6) Form $\mathsf{U} = \mathsf{Q}\hat{\mathsf{U}}$.

Cost of steps 2 and 4: Application of A and A^* to k vectors.

Cost of steps 3,5,6: Dense operations on matrices of sizes $m \times k$ and $k \times n$.

Case 1: A is presented as an array of real numbers in RAM.

Cost is dominated by the $2\,m\,n\,k$ flops required for steps 2 and 4.

The $O(m\,n\,k)$ flop count is the same as that of standard methods such as Golub-Businger. However, the algorithm above requires no random access to the matrix A — the data is retrieved *in two passes.* (One pass in the modified version.)

Remark 1: Improved data access leads to a moderate gain in execution time.

Remark 2: Using a special structured random sampling matrix (for instance, a "subsampled random Fourier transform"), substantial gain in execution time, and asymptotic complexity of $O(m\,n\,\log(k))$ can be achieved.

Input: An $m \times n$ matrix A and a target rank k.	
Output: Rank-k factors U, Σ, and V in an approximate SVD $\mathsf{A} \approx \mathsf{U}\Sigma\mathsf{V}^*$.	
(1) Draw an $n \times k$ random matrix Ω.	(4) Form the small matrix $\mathsf{B} = \mathsf{Q}^* \mathsf{A}$.
(2) Form the $n \times k$ sample matrix $\mathsf{Y} = \mathsf{A}\Omega$.	(5) Factor the small matrix $\mathsf{B} = \hat{\mathsf{U}}\Sigma\mathsf{V}^*$.
(3) Compute an ON matrix Q s.t. $\mathsf{Y} = \mathsf{Q}\mathsf{Q}^*\mathsf{Y}$.	(6) Form $\mathsf{U} = \mathsf{Q}\hat{\mathsf{U}}$.

Cost of steps 2 and 4: Application of A and A^* to k vectors.

Cost of steps 3,5,6: Dense operations on matrices of sizes $m \times k$ and $k \times n$.

Case 2: A is presented as an array of real numbers in slow memory — on "disk".

In this case, standard methods such as Golub-Businger become prohibitively slow due to the random access requirements on the data.

However, the method described above works just fine.

Limitation 1: Matrices of size $m \times k$ and $k \times n$ must fit in RAM.

Limitation 2: For matrices whose singular values decay slowly (as is typical in the data-analysis environment), the method above is typically not accurate enough. We will shortly discuss the accuracy of the method in detail, and describe a much more accurate variation.

Input: An $m \times n$ matrix A and a target rank k.	
Output: Rank-k factors U, Σ, and V in an approximate SVD $\mathsf{A} \approx \mathsf{U}\Sigma\mathsf{V}^*$.	
(1) Draw an $n \times k$ random matrix Ω.	(4) Form the small matrix $\mathsf{B} = \mathsf{Q}^* \mathsf{A}$.
(2) Form the $n \times k$ sample matrix $\mathsf{Y} = \mathsf{A}\Omega$.	(5) Factor the small matrix $\mathsf{B} = \hat{\mathsf{U}}\Sigma\mathsf{V}^*$.
(3) Compute an ON matrix Q s.t. $\mathsf{Y} = \mathsf{Q}\mathsf{Q}^*\mathsf{Y}$.	(6) Form $\mathsf{U} = \mathsf{Q}\hat{\mathsf{U}}$.

Cost of steps 2 and 4: Application of A and A^* to k vectors.

Cost of steps 3,5,6: Dense operations on matrices of sizes $m \times k$ and $k \times n$.

Case 3: A and A^* admit fast matrix-vector multiplication.

In this case, the "standard" method is some variation of Krylov methods such as Lanczos (or Arnoldi for non-symmetric matrices) whose cost T_{Krylov} satisfy:

$$T_{\text{Krylov}} \sim k\, T_{\text{mat-vec-mult}} + O(k^2\,(m+n)).$$

The asymptotic cost of the randomized scheme is the same;
its advantage is again in how the data is accessed — the k matrix-vector multiplies can be executed in parallel.

The method above is in important environments less accurate than Arnoldi, but this can be fixed while compromising only slightly on the pass-efficiency.

Input: An $m \times n$ matrix A and a target rank k.	
Output: Rank-k factors U, Σ, and V in an approximate SVD $\mathsf{A} \approx \mathsf{U}\Sigma\mathsf{V}^*$.	
(1) Draw an $n \times k$ random matrix Ω.	(4) Form the small matrix $\mathsf{B} = \mathsf{Q}^* \mathsf{A}$.
(2) Form the $n \times k$ sample matrix $\mathsf{Y} = \mathsf{A}\Omega$.	(5) Factor the small matrix $\mathsf{B} = \hat{\mathsf{U}}\Sigma\mathsf{V}^*$.
(3) Compute an ON matrix Q s.t. $\mathsf{Y} = \mathsf{Q}\mathsf{Q}^*\mathsf{Y}$.	(6) Form $\mathsf{U} = \mathsf{Q}\hat{\mathsf{U}}$.

How accurate is the proposed method?

Let $\mathsf{A}_k^{\text{computed}}$ denote the output of the method above,

$$\mathsf{A}_k^{\text{computed}} = \mathsf{U}\,\Sigma\,\mathsf{V}^*.$$

We are interested in the error

$$e_k = ||\mathsf{A} - \mathsf{A}_k^{\text{computed}}||.$$

The error e_k should be compared to the theoretically minimal error

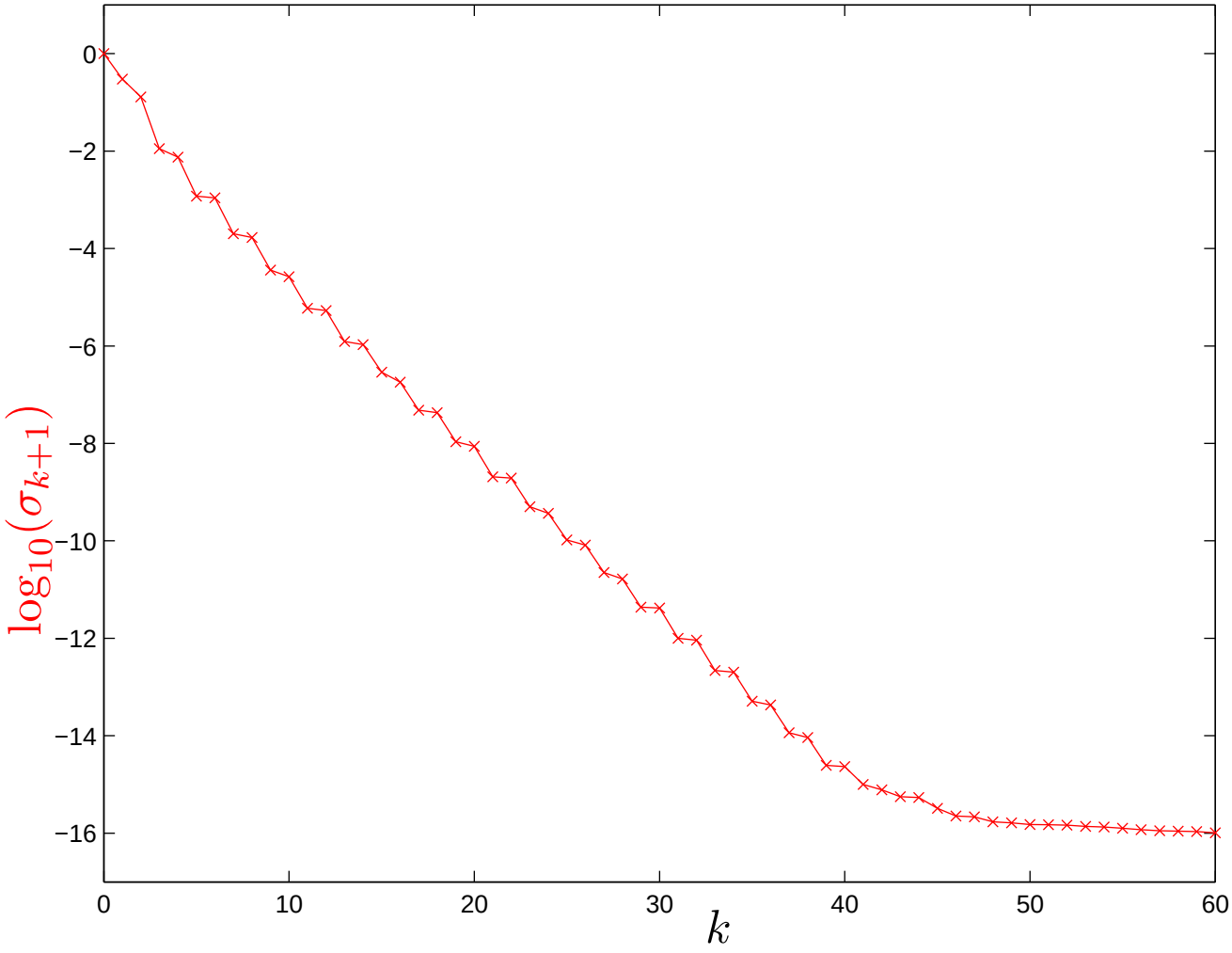
$$\sigma_{k+1} = \min\{||\mathsf{A} - \mathsf{A}_k|| : \mathsf{A}_k \text{ has rank } k\},$$

where σ_k is the (exact) k'th singular value of A.

(We criminally switched notation here — σ_j used to be the j'th computed *approximate* singular value of A — now it is the exact singular value.)

Example 1 (neither massive nor modern):

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

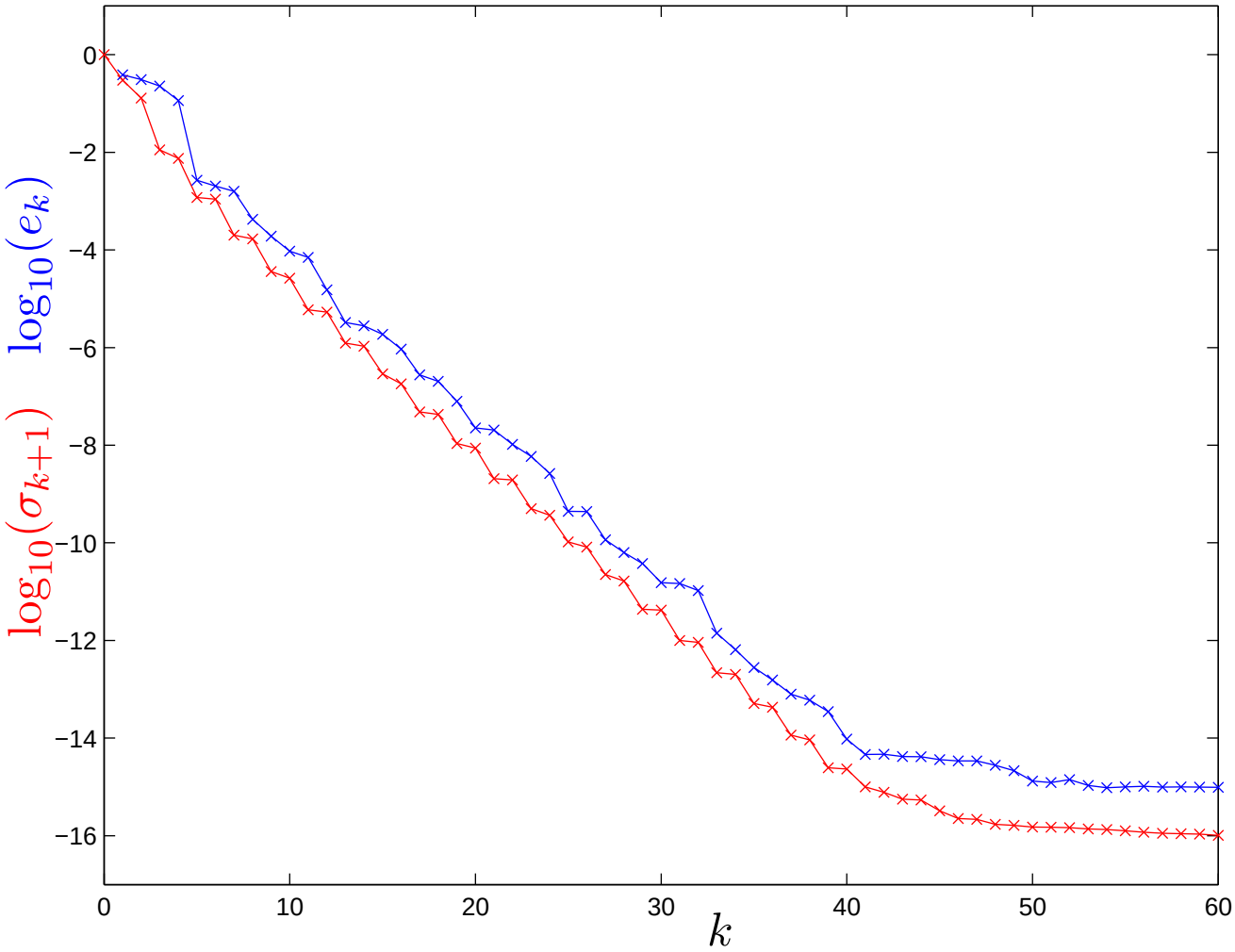


The red line indicates the singular values σ_{k+1} of A. These indicate the theoretically minimal approximation error.

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

Example 1 (neither massive nor modern):

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



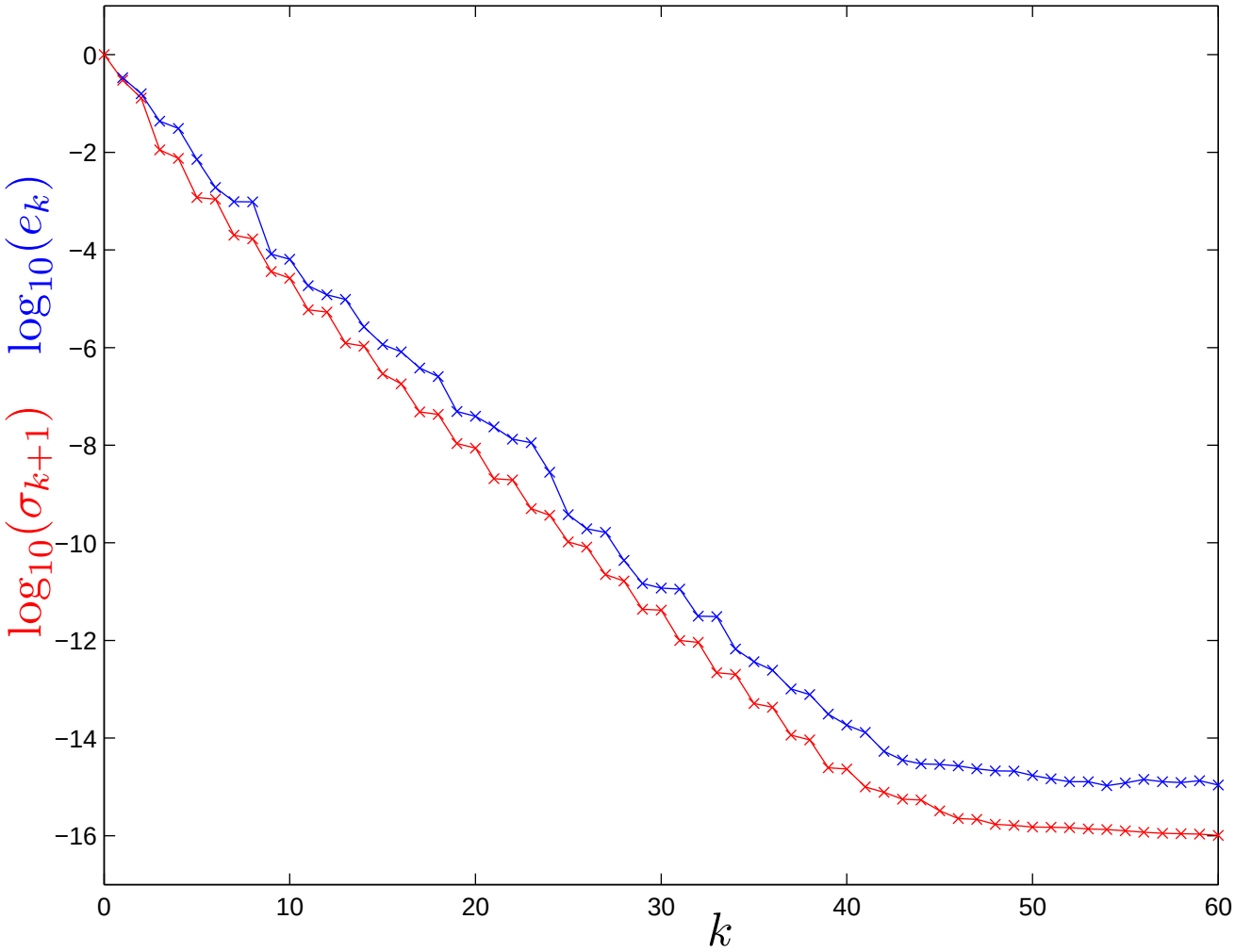
The red line indicates the singular values σ_{k+1} of A. These indicate the theoretically minimal approximation error.

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

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

Example 1 (neither massive nor modern):

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



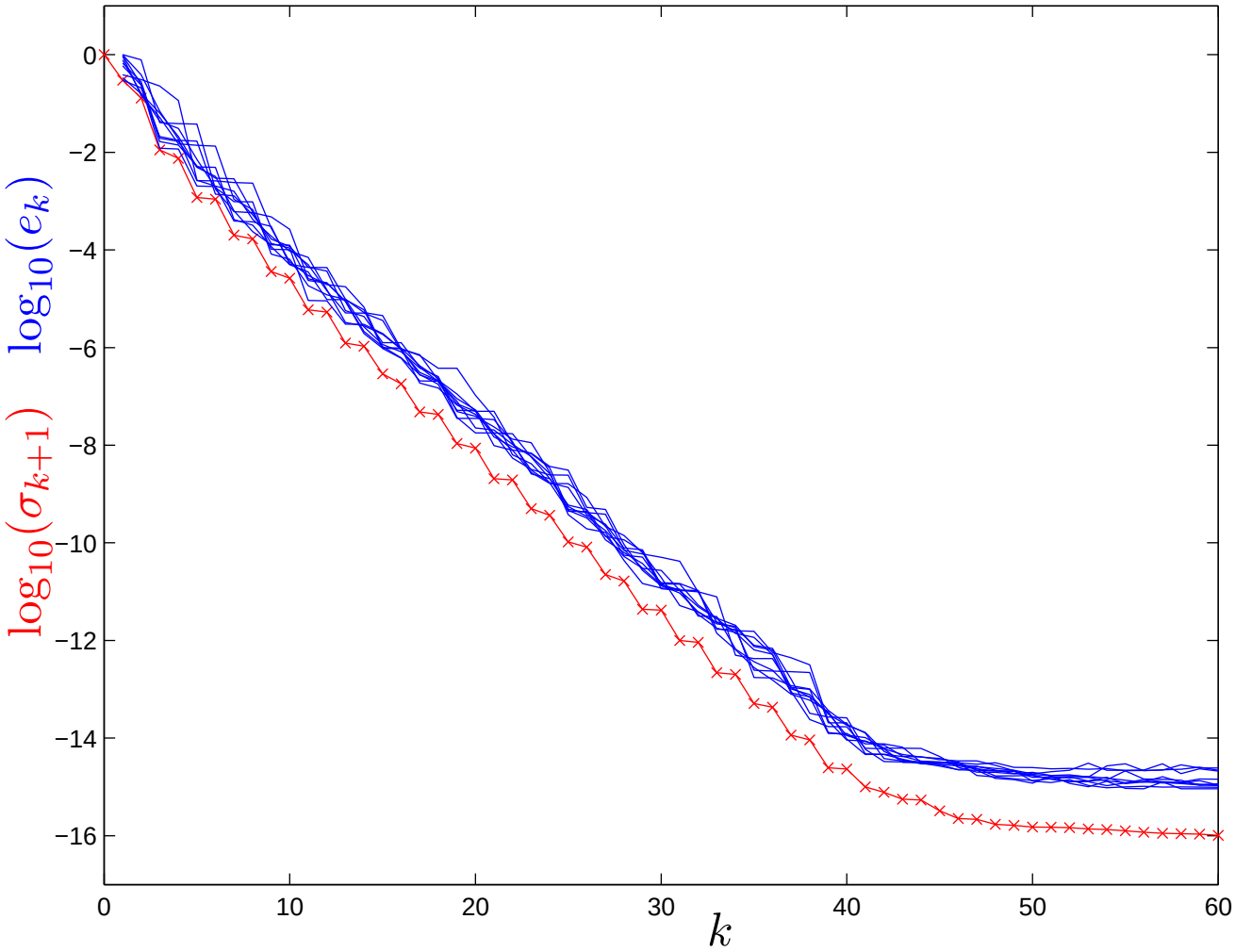
The red line indicates the singular values σ_{k+1} of A. These indicate the theoretically minimal approximation error.

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

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

Example 1 (neither massive nor modern):

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



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

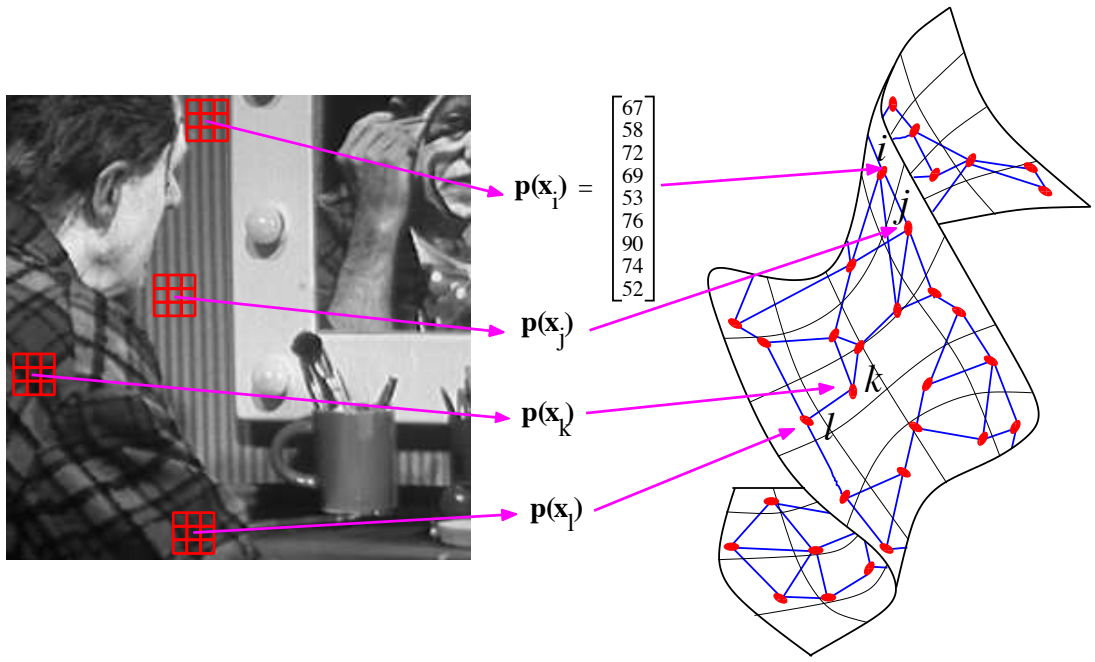
The blue lines indicate the actual errors e_k incurred by 10 different instantiations of the proposed method.

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

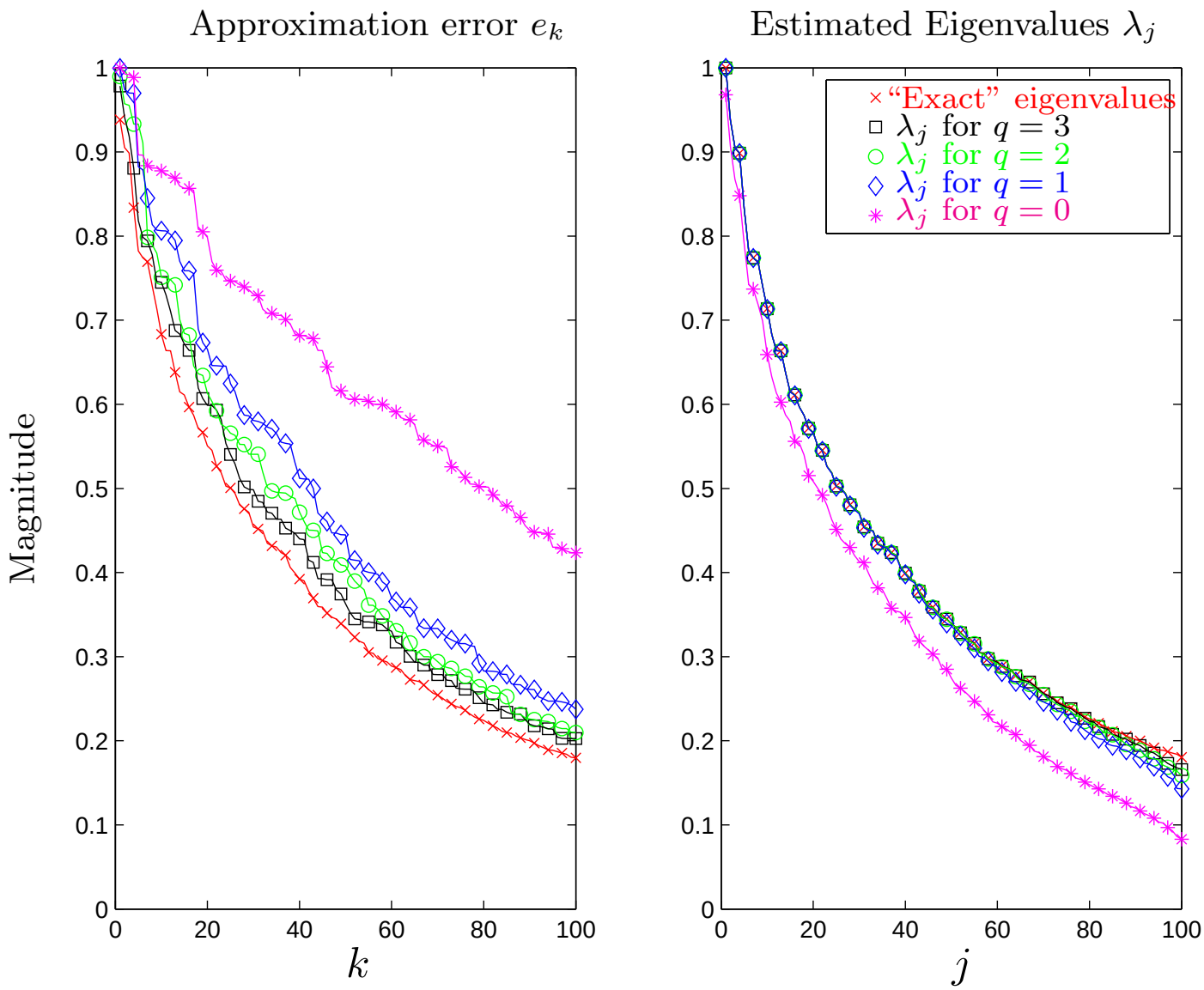
Example 2:

The matrix A being analyzed is a 9025×9025 matrix arising in a diffusion geometry approach to image processing.

To be precise, A is a graph Laplacian on the manifold of 3×3 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.

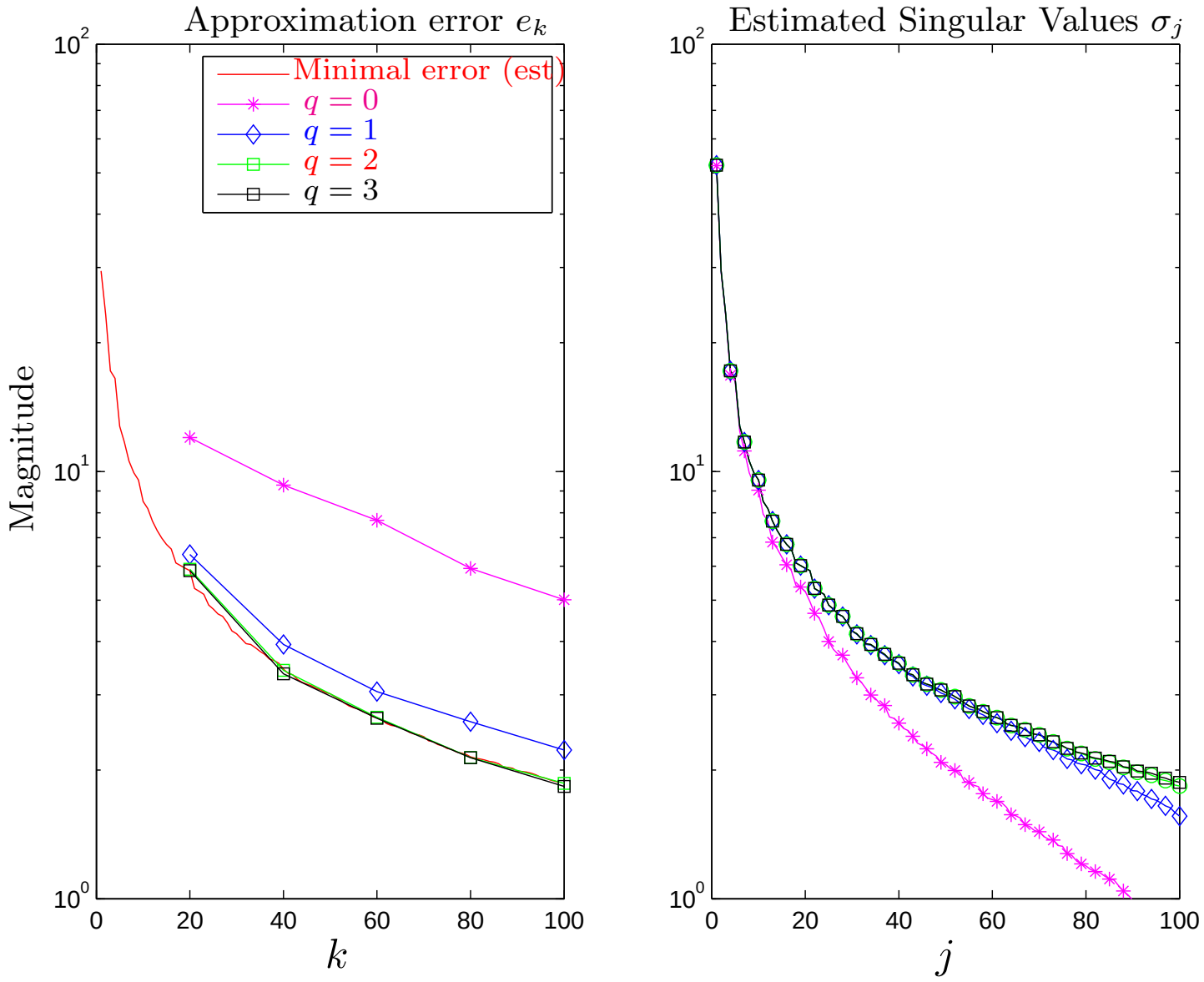
Example 3: "Eigenfaces"

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 A.

The left singular vectors of 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.

Probabilistic error analysis:

The *error* of the method is defined as

$$e_k = ||\mathsf{A} - \mathsf{A}_k^{\text{computed}}||.$$

Recall that e_k is a random variable whose theoretically minimal value is

$$\sigma_{k+1} = \min\{||\mathsf{A} - \mathsf{A}_k|| : \mathsf{A}_k \text{ has rank } k\}.$$

Question: Is e_k with high probability close σ_{k+1}? (This would be an ideal error bound.)

Probabilistic error analysis:

The *error* of the method is defined as

$$e_k = ||\mathsf{A} - \mathsf{A}_k^{\text{computed}}||.$$

Recall that e_k is a random variable whose theoretically minimal value is

$$\sigma_{k+1} = \min\{||\mathsf{A} - \mathsf{A}_k|| : \mathsf{A}_k \text{ has rank } k\}.$$

Question: Is e_k with high probability close σ_{k+1}? (This would be an ideal error bound.)

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

Probabilistic error analysis:

The *error* of the method is defined as

$$e_k = ||\mathsf{A} - \mathsf{A}_k^{\text{computed}}||.$$

Recall that e_k is a random variable whose theoretically minimal value is

$$\sigma_{k+1} = \min\{||\mathsf{A} - \mathsf{A}_k|| : \mathsf{A}_k \text{ has rank } k\}.$$

Question: Is e_k with high probability close σ_{k+1}? (This would be an ideal error bound.)

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

Remedy: Oversample a little. If p is a small integer (think $p = 5$), then we often can bound e_{k+p} by something close to σ_{k+1}. To be precise, we have

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

Moreover, strong concentration of measure $\quad\Rightarrow\quad$ Variance of $||\mathsf{A} - \mathsf{A}_{k+p}^{\text{computed}}||$ is small.

Let us look at the error bound a little closer:

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

Case 1 — the singular values decay rapidly: If (σ_j) decays sufficiently rapidly that $\left(\sum_{j>k}\sigma_j^2\right)^{1/2} \approx \sigma_{k+}$, 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 σ_{k+1}/σ_1 is not that small, we could lose all accuracy.

Problem: We lose accuracy for precisely the data-sets we are most interested in!

- If the data-set is *massive*, then n is large.
- If the data-set comes from data-mining applications (it is *"modern"*), then the singular values level off at maybe 1% or 10% of the value of σ_1.

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 $(\mathsf{A}\,\mathsf{A}^*)^q\,\mathsf{A}$ has the same left singular vectors as A, and its singular values are

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

Much faster decay — so let us use the sample matrix

$$\mathsf{Y} = (\mathsf{A}\,\mathsf{A}^*)^q\,\mathsf{A}\,\Omega$$

instead of

$$\mathsf{Y} = \mathsf{A}\,\Omega.$$

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 A, a target rank k, and a small integer q.	
Output: Rank-k factors U, Σ, and V in an approximate SVD $\mathsf{A} \approx \mathsf{U}\Sigma\mathsf{V}^*$.	
(1) Draw an $n \times k$ random matrix Ω.	(4) Form the small matrix $\mathsf{B} = \mathsf{Q}^* \mathsf{A}$.
(2) Form the $n \times k$ sample matrix $\mathsf{Y} = (\mathsf{A}\,\mathsf{A}^*)^q\,\mathsf{A}\Omega$.	(5) Factor the small matrix $\mathsf{B} = \hat{\mathsf{U}}\Sigma\mathsf{V}^*$.
(3) Compute an ON matrix Q s.t. $\mathsf{Y} = \mathsf{Q}\mathsf{Q}^*\mathsf{Y}$.	(6) Form $\mathsf{U} = \mathsf{Q}\hat{\mathsf{U}}$.

Theorem: $\mathbb{E}||\mathsf{A} - \mathsf{A}_{k+p}^{\text{computed}}||$ converges exponentially fast to the optimal value of σ_{k+1} as q increases.

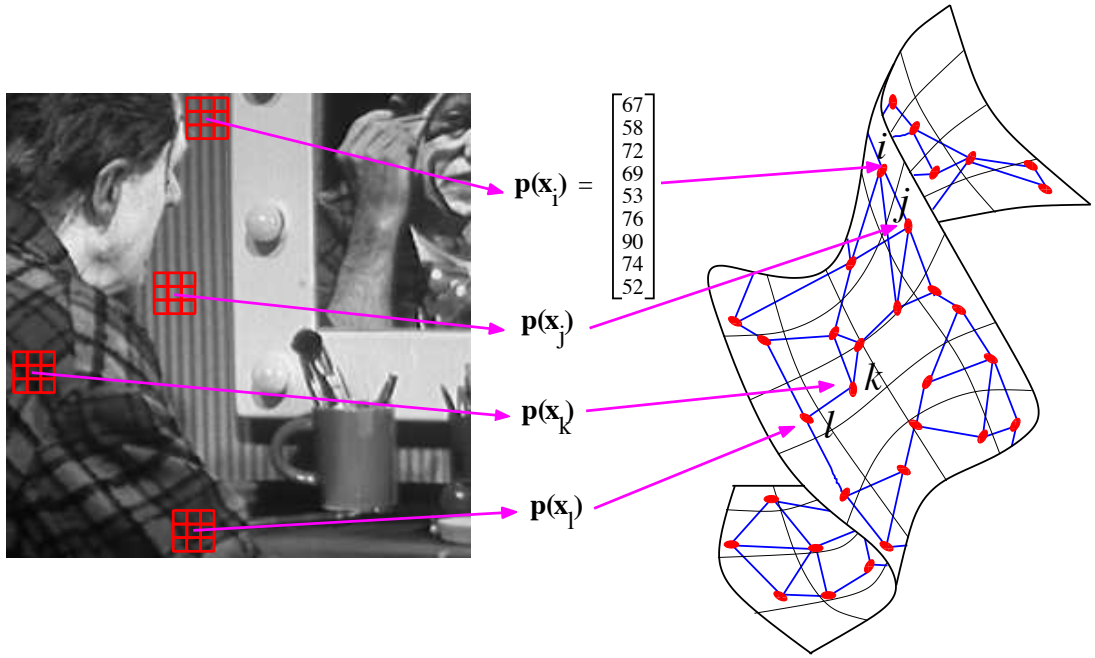
The theorem assumes exact arithmetic — in real life some complications arise.
These can be handled by careful implementation.

The modified scheme obviously comes at a substantial 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.

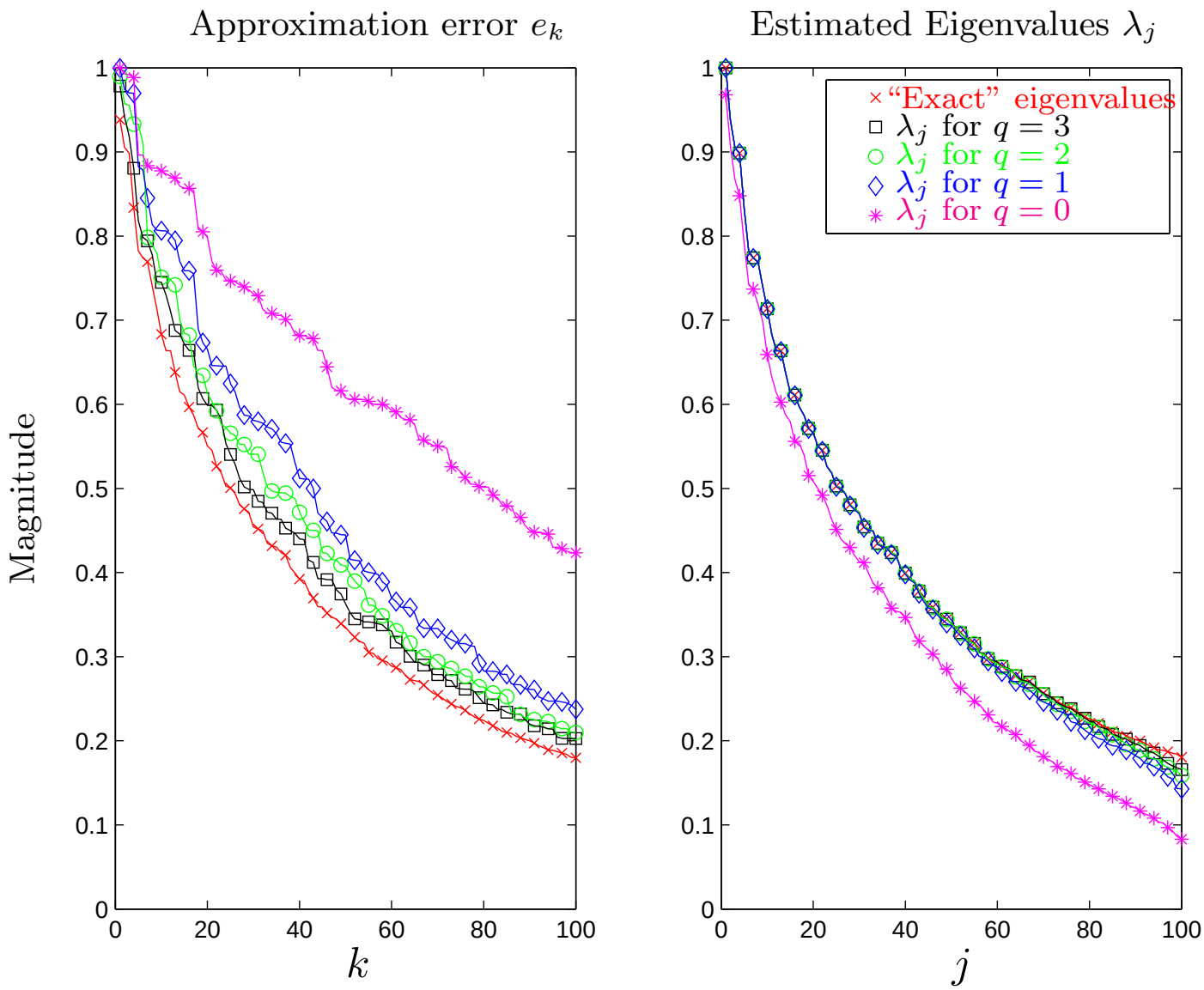
Example 2 (revisited):

The matrix A being analyzed is a 9025×9025 matrix arising in a diffusion geometry approach to image processing.

To be precise, A is a graph Laplacian on the manifold of 3×3 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.

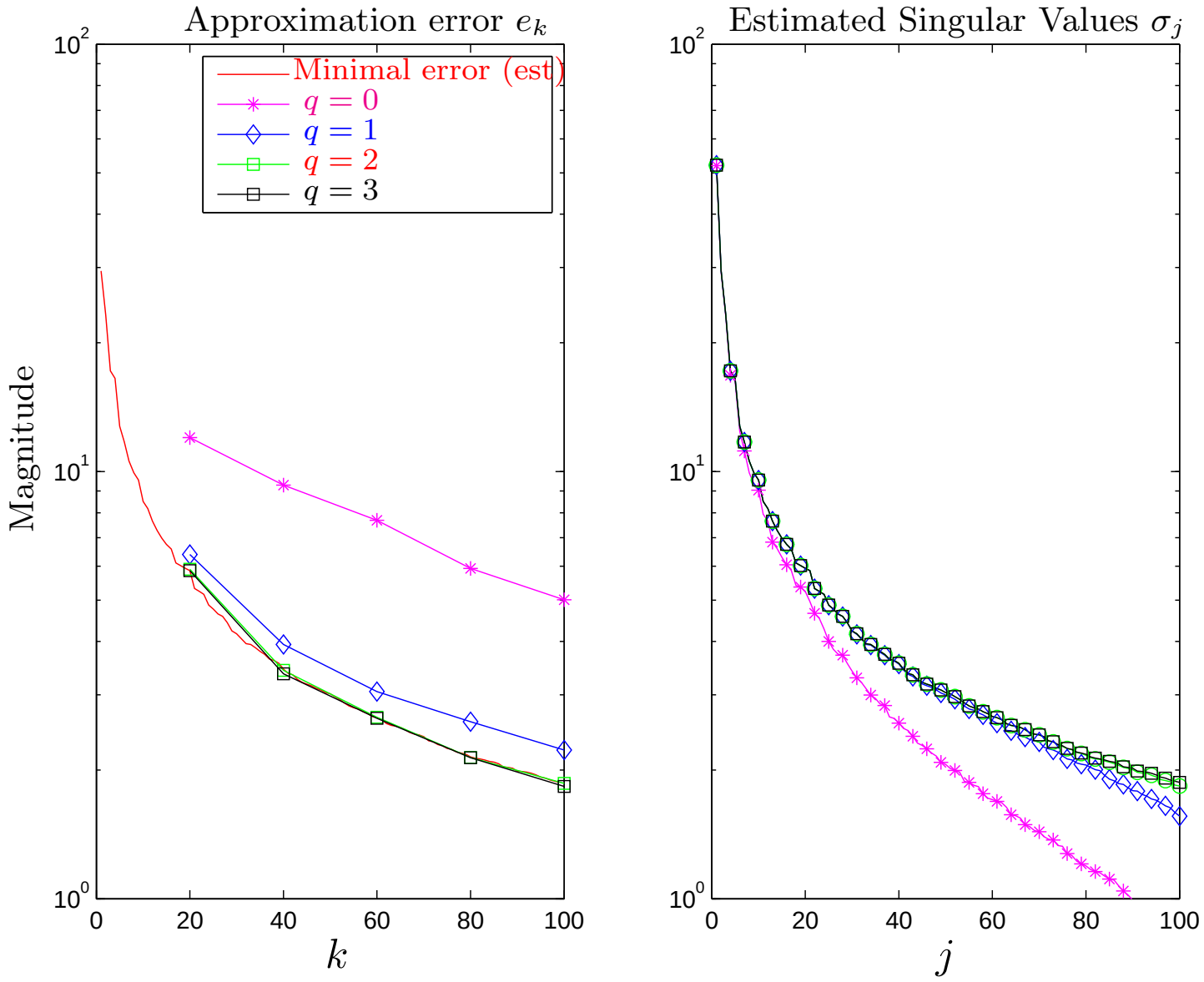
Example 3 (revisited): "Eigenfaces"

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 A.

The left singular vectors of 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.

We note that the power method can be viewed as a hybrid between the "basic" randomized method, and a Krylov subspace method:

Krylov method: Restrict A to the linear space

$$\mathcal{V}_q(\boldsymbol{\omega}) = \mathrm{Span}(\mathsf{A}\boldsymbol{\omega},\ \mathsf{A}^2\boldsymbol{\omega},\ \ldots,\ \mathsf{A}^q\boldsymbol{\omega}).$$

"Basic" randomized method: Restrict A to the linear space

$$\mathrm{Span}(\mathsf{A}\boldsymbol{\omega}_1,\ \mathsf{A}\boldsymbol{\omega}_2,\ \ldots,\ \mathsf{A}\boldsymbol{\omega}_\ell) = \mathcal{V}_1(\boldsymbol{\omega}_1) \times \mathcal{V}_1(\boldsymbol{\omega}_2) \times \cdots \times \mathcal{V}_1(\boldsymbol{\omega}_\ell).$$

"Power" method: Restrict A to the linear space

$$\mathrm{Span}(\mathsf{A}^q\boldsymbol{\omega}_1,\ \mathsf{A}^q\boldsymbol{\omega}_2,\ \ldots,\ \mathsf{A}^q\boldsymbol{\omega}_\ell).$$

Modified "power" method: Restrict A to the linear space

$$\mathcal{V}_q(\boldsymbol{\omega}_1) \times \mathcal{V}_q(\boldsymbol{\omega}_2) \times \cdots \times \mathcal{V}_q(\boldsymbol{\omega}_\ell).$$

This could be a promising area for further work.

Now what about those $O(m\,n\,\log(k))$ methods for general unstructured matrices?

Recall the single pass algorithm for an approximate rank-k SVD:

Given a $m \times n$ matrix A, find U, V, $\mathsf{\Sigma}$ of rank k such that $\mathsf{A} \approx \mathsf{U}\,\mathsf{\Sigma}\,\mathsf{V}^*$.

1. Choose an over-sampling parameter p and set $\ell = k + p$. (Say $p = k$ so $\ell = 2k$.)

2. Generate Gaussian random matrices $\mathsf{\Omega}$ and $\mathsf{\Psi}$ of sizes $n \times \ell$, and $m \times \ell$.

3. Form the sample matrices $\mathsf{Y} = \mathsf{A}\,\mathsf{\Omega}$ and $\mathsf{Z} = \mathsf{A}^*\,\mathsf{\Psi}$.

4. Find ON matrices Q and W such that $\mathsf{Y} = \mathsf{Q}\,\mathsf{Q}^*\,\mathsf{Y}$ and $\mathsf{Z} = \mathsf{W}\,\mathsf{W}^*\,\mathsf{Z}$.

5. Solve for the $k \times k$ matrix T the systems $\mathsf{Q}^*\,\mathsf{Y} = \mathsf{T}\,(\mathsf{W}^*\,\mathsf{\Omega})$ and $\mathsf{W}^*\,\mathsf{Z} = \mathsf{T}^*\,(\mathsf{Q}^*\,\mathsf{\Psi})$.

6. Compute the SVD of the small matrix $\mathsf{T} = \hat{\mathsf{U}}\,\mathsf{\Sigma}\,\hat{\mathsf{V}}^*$ (and truncate if desired).

7. Form $\mathsf{U} = \mathsf{Q}\,\hat{\mathsf{U}}$ and $\mathsf{V} = \mathsf{W}\,\hat{\mathsf{V}}$.

Observation 1: Forming $\mathsf{A}\,\mathsf{\Omega}$ and $\mathsf{A}^*\,\mathsf{\Psi}$ in Step 2 has cost $O(m\,n\,\ell)$, and $\ell \sim k$.

Observation 2: All other steps cost at most $O((m+n)\,\ell^2)$.

Now what about those $O(m\,n\,\log(k))$ methods for general unstructured matrices?

Recall the single pass algorithm for an approximate rank-k SVD:
Given a $m \times n$ matrix A, find U, V, $\mathsf{\Sigma}$ of rank k such that $\mathsf{A} \approx \mathsf{U}\,\mathsf{\Sigma}\,\mathsf{V}^*$.

1. Choose an over-sampling parameter p and set $\ell = k + p$. (Say $p = k$ so $\ell = 2k$.)
2. Generate Gaussian random matrices Ω and Ψ of sizes $n \times \ell$, and $m \times \ell$.
3. Form the sample matrices $\mathsf{Y} = \mathsf{A}\,\Omega$ and $\mathsf{Z} = \mathsf{A}^*\,\Psi$.
4. Find ON matrices Q and W such that $\mathsf{Y} = \mathsf{Q}\,\mathsf{Q}^*\,\mathsf{Y}$ and $\mathsf{Z} = \mathsf{W}\,\mathsf{W}^*\,\mathsf{Z}$.
5. Solve for the $k \times k$ matrix T the systems $\mathsf{Q}^*\,\mathsf{Y} = \mathsf{T}\,(\mathsf{W}^*\,\Omega)$ and $\mathsf{W}^*\,\mathsf{Z} = \mathsf{T}^*\,(\mathsf{Q}^*\,\Psi)$.
6. Compute the SVD of the small matrix $\mathsf{T} = \hat{\mathsf{U}}\,\mathsf{\Sigma}\,\hat{\mathsf{V}}^*$ (and truncate if desired).
7. Form $\mathsf{U} = \mathsf{Q}\,\hat{\mathsf{U}}$ and $\mathsf{V} = \mathsf{W}\,\hat{\mathsf{V}}$.

Observation 1: Forming $\mathsf{A}\,\Omega$ and $\mathsf{A}^*\,\Psi$ in Step 2 has cost $O(m\,n\,\ell)$, and $\ell \sim k$.

Observation 2: All other steps cost at most $O((m+n)\,\ell^2)$.

Question: Can we somehow reduce the costs of forming the products

$$\underset{m \times \ell}{\mathsf{Y}} = \underset{m \times n}{\mathsf{A}} \; \underset{n \times \ell}{\Omega} \qquad \text{and} \qquad \underset{n \times \ell}{\mathsf{Z}} = \underset{n \times m}{\mathsf{A}^*} \; \underset{m \times \ell}{\Psi}$$

A is unstructured,

Question: Can we somehow reduce the costs of forming the products

$$\underset{m \times \ell}{\mathsf{Y}} = \underset{m \times n}{\mathsf{A}} \; \underset{n \times \ell}{\Omega} \qquad \text{and} \qquad \underset{n \times \ell}{\mathsf{Z}} = \underset{n \times m}{\mathsf{A}^*} \; \underset{m \times \ell}{\Psi}$$

A is unstructured, but it is possible to introduce structure in Ω and Ψ!
For instance, use a *subsampled random Fourier Transform (SRFT):*

$$\underset{n \times \ell}{\Omega} = \underset{n \times n}{\mathsf{D}} \; \underset{n \times n}{\mathsf{F}} \; \underset{n \times \ell}{\mathsf{S}}$$

where,

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

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

Notes:

- Significant speed-ups are achieved for common problem sizes. For instance, $m = n = 2\,000$ and $k = 200$ leads to a speed-up by roughly a factor of 4.

- Many other choices of random matrices have been found.
  - Subsampled Hadamard transform.
  - Wavelets.
  - Random chains of Given's rotations. (Seems to work the best.)

  Such maps are sometimes referred to as *fast Johnson-Lindenstrauss transforms.*

- Current theoretical results seem to overstate the risk of inaccurate results, at least in typical environments. What is the relation between ℓ and k?

- This idea was proposed by Liberty, Rokhlin, Tygert, Woolfe (2006).
  - The SRFT described above was suggested by Nir Ailon and Bernard Chazelle (2006) in a related context.
  - Dissertation by Edo Liberty.
  - Related recent work by Sarlós (on randomized regression).
  - Very interesting follow-up paper on overdetermined linear least-squares regression by Rokhlin and Tygert (2008).

Final remarks:

- For large scale SVD/PCA of dense matrices, these algorithms are highly recommended; they compare favorably to existing methods in almost every regard. Free software can be downloaded → google *Mark Tygert*.

- The approximation error is a random variable, but its distribution is very narrowly concentrated. Any preset accuracy can be met to within probability $1 - \eta$ where η is a user set "failure probability" (*e.g.* $\eta = 10^{-10}$ or 10^{-20}).

- This talk 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, but *experimentation is easy!* Concentration of measure makes the algorithms behave as if deterministic.

- To find out more:
  - A tutorial long version of this talk is available on the *NIPS 2009* website.
  - Review: *Finding structure with randomness: Stochastic algorithms for constructing approximate matrix decompositions*
    N. Halko, P.G. Martinsson, J. Tropp — arXiv.org report 0909.4061.