Randomization: Making Very Large-Scale Linear Algebraic Computations Possible

Gunnar Martinsson
The University of Colorado at Boulder

The material presented draws on work by Nathan Halko, Edo Liberty, Vladimir Rokhlin, Arthur Szlam, Joel Tropp, Mark Tygert, Franco Woolfe, and others.

Goal:

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

$$\underset{n \times n}{\mathsf{A}} \quad \approx \quad \underset{n \times k}{\mathsf{E}} \quad \underset{k \times n}{\mathsf{F}^*} \quad = \quad \sum_{j=1}^{k} \boldsymbol{e}_j\, \boldsymbol{f}_j^*.$$

Solving this problem leads to algorithms for computing:

- Singular Value Decomposition (SVD) / Principal Component Analysis (PCA).
  (Require $\{\boldsymbol{e}_j\}_{j=1}^k$ and $\{\boldsymbol{f}_j\}_{j=1}^k$ to be orthogonal sets.)

- Finding spanning columns or rows.
  (Require $\{\boldsymbol{e}_j\}_{j=1}^k$ to be columns of A, or require $\{\boldsymbol{f}_j^*\}_{j=1}^k$ to be rows of A.)

- Determine eigenvectors corresponding to leading eigenvalues.
  (Require $\boldsymbol{e}_j = \lambda_j\, \boldsymbol{f}_j$, and $\{\boldsymbol{f}_j\}_{j=1}^k$ to be orthonormal.)

- *etc*

1. The new methods enable the handling of significantly larger and noisier matrices on any given computer.

1. The new methods enable the handling of significantly larger and noisier matrices on any given computer.

2. The techniques are intuitive, and simple to implement.
    - Pseudo-code (5 – 7 lines long) for key problems will be given.
        - Compute the SVD.
        - Compute the SVD of a very noisy matrix.
        - Compute the SVD of a matrix with one pass over the data.
        - Instead of the SVD: Find spanning rows or columns.
    - Ready-to-use software packages are available.

1. The new methods enable the handling of significantly larger and noisier matrices on any given computer.

2. The techniques are intuitive, and simple to implement.
    - Pseudo-code (5 – 7 lines long) for key problems will be given.
        - Compute the SVD.
        - Compute the SVD of a very noisy matrix.
        - Compute the SVD of a matrix with one pass over the data.
        - Instead of the SVD: Find spanning rows or columns.
    - Ready-to-use software packages are available.

3. The problem being addressed is ubiquitous in applications.

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.

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

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

Applications:

- *Principal Component Analysis:* Form an empirical covariance matrix from some collec-
  tion … atrix,
  you f
- *Find* … nding
  a set … (Say
  a sma … stocks
  in a p
- *Rela* … which
  each … an be
  found
- *Page* … ne its
  leadi
- *Eigenfaces:* Form a matrix whose columns represent normalized grayscale images of faces. The "eigenfaces" are the left singular vectors.

Caveats:

- Absolutely no expertise on applications is claimed!
  (Deep ignorance, rather.)
- The purpose of the list is simply to illustrate some well-known algorithms based on linear algebraic tools.
- Many of these algorithms are not effective, have been obsoleted, are now subjects of scorn, *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.

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

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

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

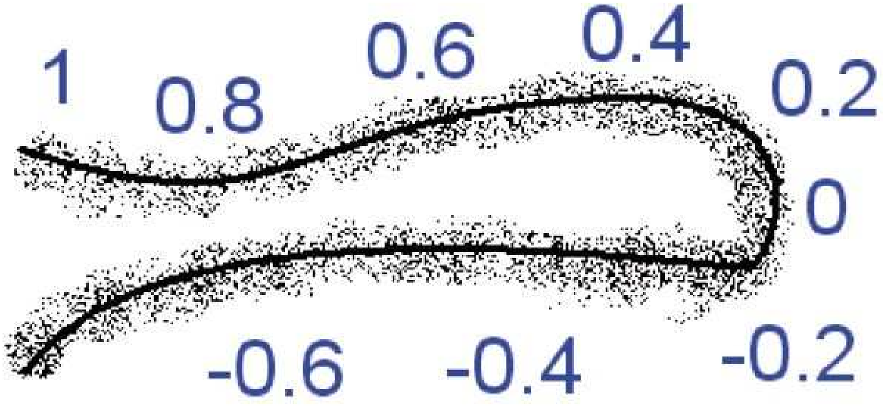
Many non-linear algorithms can only run on small data sets, or operate locally on large data sets. Iterative methods are common, and these often involve linear problems.

Another approach is to recast a non-linear problem as a linear one via a transform or reformulation. As an example, we will briefly discuss the *diffusion geometry* approach.

Diffusion geometry:

Problem: We are given a large data set $\{\boldsymbol{x}_j\}_{j=1}^{N}$ in a high-dimensional space $\mathbb{R}^D$. We have reason to believe that the data is clustered around some low-dimensional manifold, but do not know the manifold.

Its geometry is revealed if we can *parameterize the data* in a way that conforms to its own geometry.



Picture courtesy of Mauro Maggioni of Duke.

A parameterization admits clustering, data completion, prediction, learning, ...

Diffusion geometry:

Problem: We are given a large data set $\{\boldsymbol{x}_j\}_{j=1}^N$ in a high-dimensional space $\mathbb{R}^D$. We have reason to believe that the data is clustered around some low-dimensional manifold, but do not know the manifold.

Diffusion geometry approach:

Measure similarities via a kernel function $W(\boldsymbol{x}_i, \boldsymbol{x}_j)$, say $W(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{-||\boldsymbol{x}_i - \boldsymbol{x}_j||^2/\sigma^2}$.

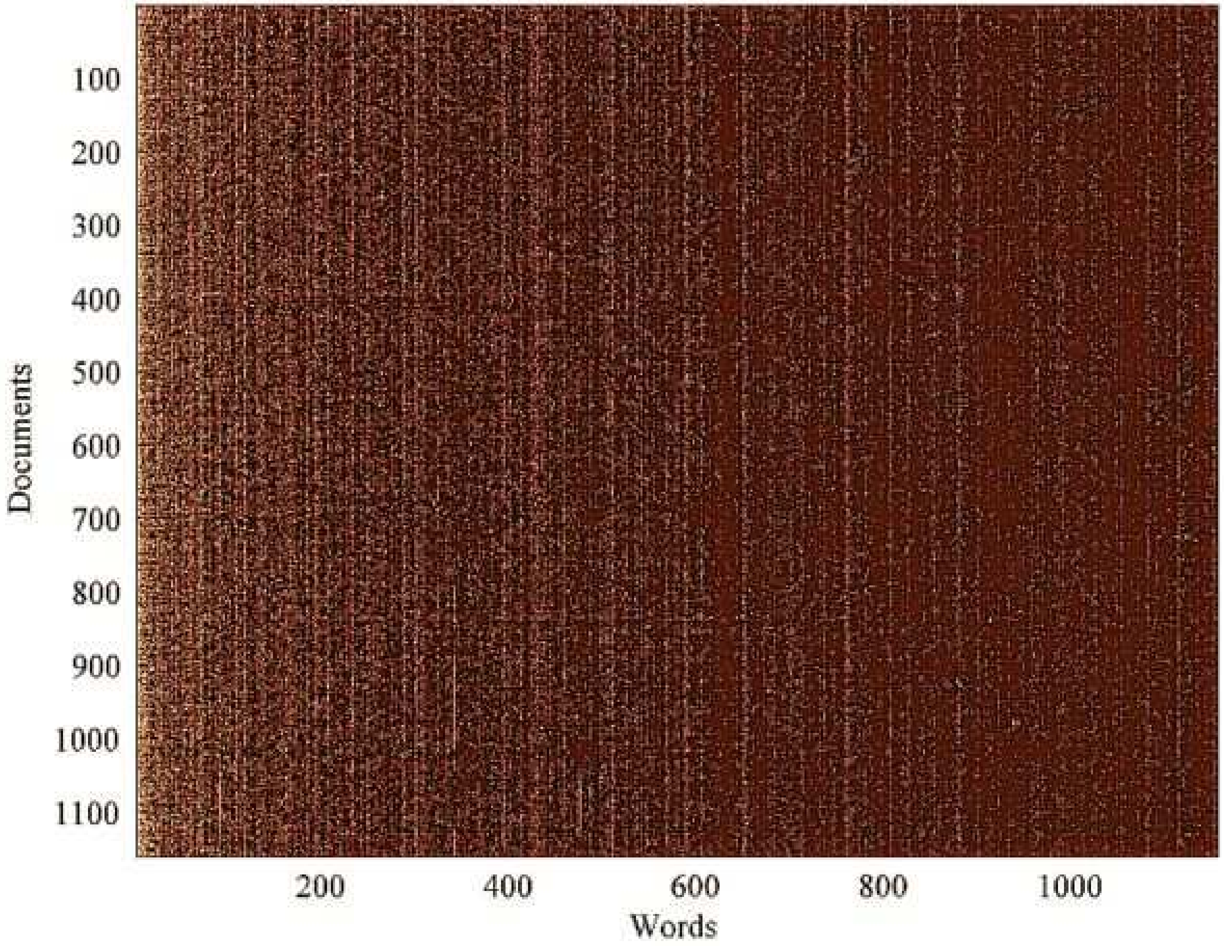
Model the data as a weighted graph (G, E, W): vertices are data points, edges connect nodes i and j with the weight $\mathsf{W}_{ij} = W(\boldsymbol{x}_i, \boldsymbol{x}_j)$. Let $\mathsf{D}_{ii} = \sum_j \mathsf{W}_{ij}$ and set

$$\underbrace{\mathsf{P} = \mathsf{D}^{-1}\mathsf{W}}_{\text{random walk}}, \qquad \underbrace{\mathsf{T} = \mathsf{D}^{-1/2}\mathsf{W}\mathsf{D}^{-1/2}}_{\text{“symmetric random walk”}}, \qquad \underbrace{\mathsf{L} = \mathsf{I} - \mathsf{T}}_{\text{graph Laplacian}}, \qquad \underbrace{\mathsf{H} = e^{-t\mathsf{L}}}_{\text{heat kernel}}.$$

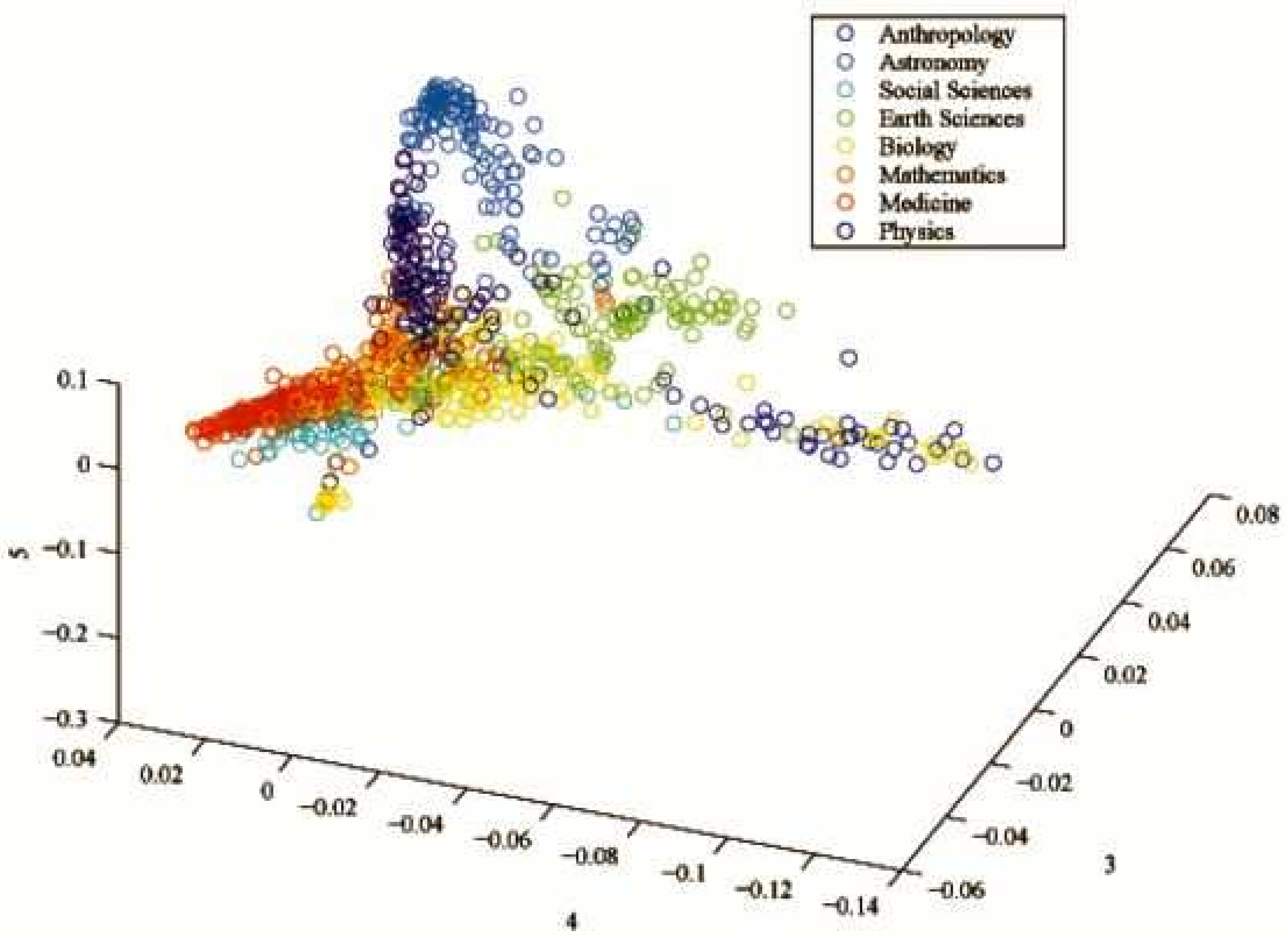
The eigenvectors of the various matrices result in parameterizations of the data.

In effect, the technique reduces a non-linear problem to a linear one.
This linear problem is very large and its eigenvalues decay only slowly in magnitude.

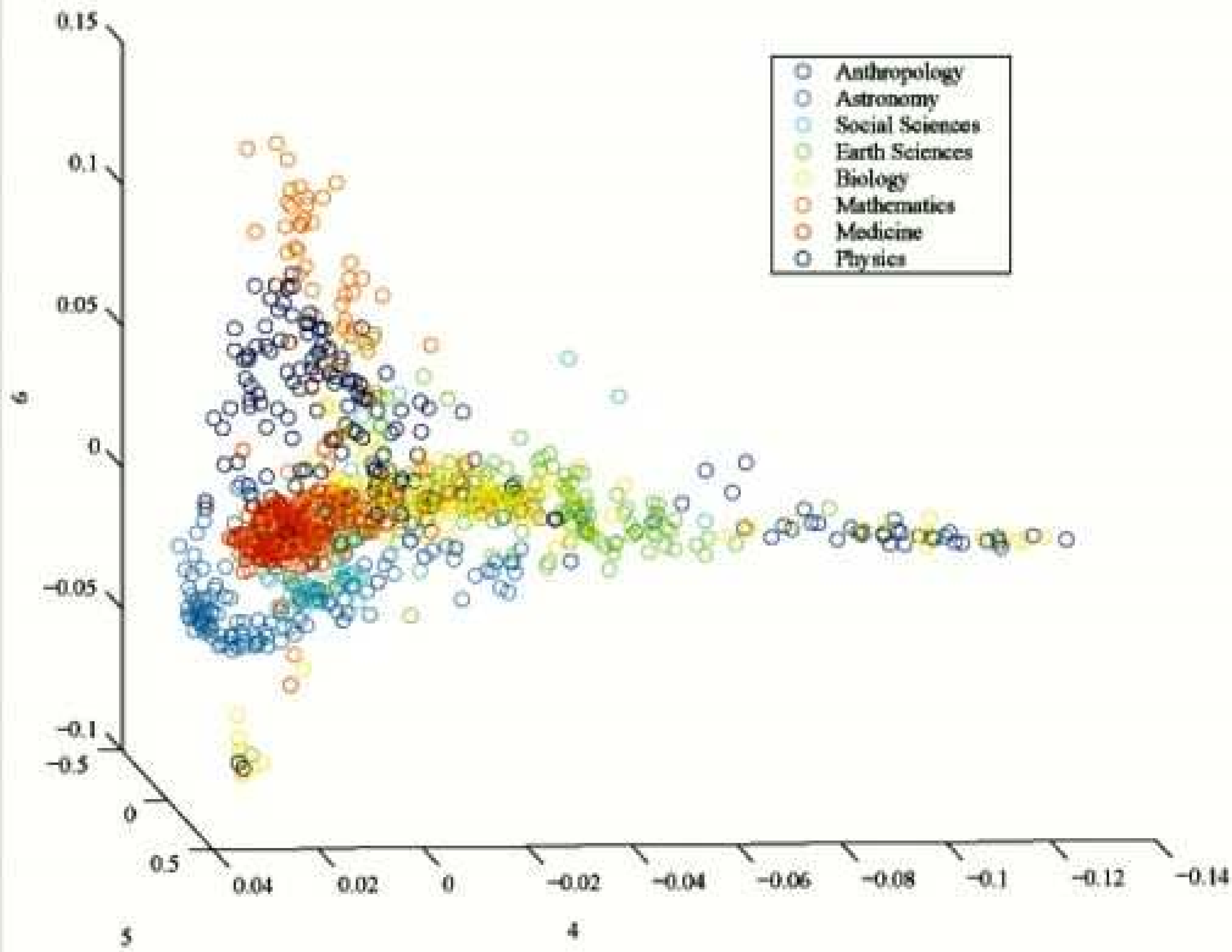
Example — Text Document: About 1100 Science News articles, from 8 different categories. We compute about 1000 coordinates, i-th coordinate of document j represents frequency in document j of the i-th word in a dictionary.



Picture courtesy of Mauro Maggioni of Duke.



Picture courtesy of Mauro Maggioni of Duke.



Picture courtesy of Mauro Maggioni of Duke.

OK, so this is why we want to solve the linear approximation problem.

Question: Isn't it well known how to compute standard factorizations?

Yes — whenever the matrix A fits in RAM.

Excellent algorithms are already part of LAPACK (and Matlab, *etc*).

- Double precision accuracy.
- Very stable.
- $O(n^3)$ asymptotic complexity. 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*).

The state-of-the-art is very satisfactory.
For kicks, we'll improve on it anyway, but this is not the main point.

Problem: The applications we have in mind lead to matrices that are not amenable to the standard techniques. Difficulties include:

- The matrices involved can be very large (size $1\,000\,000 \times 1\,000\,000$, say).
- Constraints on communication — few "passes" over the data.
  - Data stored in slow memory.
  - Parallel processors.
  - Streaming data.
- Lots of noise — hard to discern the "signal" from the "noise."
  High-quality denoising algorithms tend to require global operations.

Characteristic properties of the matrices arising in knowledge extraction.

Question: Computers get more powerful all the time. Can't we just wait it out?

Observation 1: The size of the problems increase too! Tools for acquiring and storing data are improving at an even faster pace than processors.

The famous "deluge" of data: documents, web searching, customer databases, hyper-spectral imagery, social networks, gene arrays, proteomics data, sensor networks, financial transactions, traffic statistics (cars, computer networks), ...

Recently, *tera-scale* computations were a hot keyword. Now it is *peta-scale*.

Example: Do-it-yourself acquisition of a petabyte of data:

Storage: You need 1000 hard drives with 1TB each. Cost: $100k.

Data acquisition: Record video from your cable outlet.

So acquiring 1PB of data is trivial. Processing it is hard!
(Note: For processing, you typically have to work with uncompressed data.)

Question: Computers get more powerful all the time. Can't we just wait it out?

Observation 1: The size of the problems increase too! Tools for acquiring and storing data are improving at an even faster pace than processors.

Question: Computers get more powerful all the time. Can't we just wait it out?

Observation 1: The size of the problems increase too! Tools for acquiring and storing data are improving at an even faster pace than processors.

Observation 2: Good algorithms are necessary. The flop count must scale close to linearly with problem size.

Observation 3: Communication is becoming the real bottleneck. Robust algorithms with good flop counts exist, but many were designed for an environment where you have random access to the data.

- *Communication speeds improve far more slowly than CPU speed and storage capacity.*
- The capacity of fast memory close to the processor is growing very slowly.
- Much of the gain in processor and storage capability is attained via parallelization. This poses particular challenges for algorithmic design.

Existing techniques for handling very large matrices

Krylov subspace methods often yield excellent accuracy for large problems.

The idea is to pick a starting vector $\boldsymbol{\omega}$ (often a random vector), "restrict" the matrix A to the k-dimensionsal "Krylov subspace"

$$\text{Span}(\mathsf{A}\,\boldsymbol{\omega},\ \mathsf{A}^2\,\boldsymbol{\omega},\ \dots,\ \mathsf{A}^k\,\boldsymbol{\omega})$$

and compute approximate eigenvectors of the resulting matrix.

The randomized methods improve on Krylov methods in several regards:

- A Krylov method accesses the matrix k consecutive times.
  In contrast, the randomized methods can be "single pass" or "few-passes."

- Randomized methods allow more flexibility in how data is stored and accessed.

- Krylov methods have a slightly involved convergence analysis.
  (What if $\boldsymbol{\omega}$ is chosen poorly, for instance?)

Drawback of both Krylov and randomized methods: $O(nk)$ fast memory required.

Goal (restated): Given an $n \times n$ matrix A, we seek a factorization

$$\begin{array}{cccccc} \mathsf{A} & \approx & \mathsf{E} & \mathsf{F}^{*} & = & \displaystyle\sum_{j=1}^{k} \boldsymbol{e}_j\, \boldsymbol{f}_j^{*}\,. \\ n \times n & & n \times k & k \times n & & \end{array}$$

Solving this problem leads to algorithms for computing:

- Singular Value Decomposition (SVD) / Principal Component Analysis (PCA).
- Finding spanning columns or rows.
- Determine eigenvectors corresponding to leading eigenvalues.

The goal is to handle matrices that are far larger and far noiser than what can be handled using existing methods.

The new algorithms 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.

Before we describe the algorithms, let us acknowledge related work:

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)

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)

Principal researchers on specific work reported:

- Vladimir Rokhlin, *Yale University*.
- Joel Tropp, *Caltech*.
- Mark Tygert, *NYU*. ← downloadable code on Mark's webpage

Related work by:

- Petros Drineas, *Rensellaer*.
- Mauro Maggioni, *Duke*.
- François Meyer, *U Colorado at Boulder*.

Students: Nathan Halko and Daniel Kaslovsky.

Detailed bibliography in recent review article: *Finding structure with randomness: Stochastic algorithms for constructing approximate matrix decompositions*, N. Halko, P.G. Martinsson, J. Tropp

Copy of slides on web: Google Gunnar Martinsson

The presentation is organized around a core difficulty:
How do you handle noise in the computation?

The case where A has exact rank k,

$$\mathsf{A} = \mathsf{E}\mathsf{F}^* = \sum_{j=1}^{k} \boldsymbol{e}_j \boldsymbol{f}_j^*$$

is in many ways exceptional. Almost every algorithm “works.”

In real life, the matrices involved do not have exact rank k approximations.

Instead, we have to solve approximation problems:

Problem: Given a matrix A and a precision ε, find the minimal k such that

$$\min\{||\mathsf{A} - \mathsf{B}|| \,:\, \text{rank}(\mathsf{B}) = k\} \leq \varepsilon.$$

Problem: Given a matrix A and an integer k, determine

$$\mathsf{A}_k = \text{argmin}\{||\mathsf{A} - \mathsf{B}|| \,:\, \text{rank}(\mathsf{B}) = k\}.$$

$$\begin{array}{cccccc} \mathsf{A} & = & \mathsf{E} & \mathsf{F}^* & + & \mathsf{N} \\ n \times n & & n \times k & k \times n & & n \times n \\ & & \text{“signal”} & & & \text{“noise”} \end{array}$$

The larger N is relative to EF^*, the harder the task.

We will handle this problem incrementally:

(1) No noise: $\mathsf{N} = 0$.

(2) Some noise: N is small.

(3) Noise is prevalent: N is large — maybe much larger than EF^*.

Outline of the tutorial:

1. Techniques for computing the SVD of a matrix of exact rank k.

2. Variations of techniques for matrices of exact rank.
   - Single pass algorithms.
   - How to compute spanning rows and spanning columns (CUR, etc).

3. Techniques for matrices of the form $\mathsf{A} = \mathsf{E}\mathsf{F} + \mathsf{N}$ with N small.
   - Error estimation.

4. Techniques for matrices of the form $\mathsf{A} = \mathsf{E}\mathsf{F} + \mathsf{N}$ with N large.
   - The "power method."

5. Random transforms that can be applied rapidly.
   - The "Subsampled Random Fourier Transform" (SRFT) and its cousins.

6. Review / Putting things together / Model problems.

Review of the SVD: Every $n \times n$ matrix A of rank k admits a factorization

$$\mathsf{A} = \sum_{j=1}^{k} \sigma_j \boldsymbol{u}_j \boldsymbol{v}_j^* = [\boldsymbol{u}_1\ \boldsymbol{u}_2\ \ldots\ \boldsymbol{u}_k] \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & \sigma_k \end{bmatrix} \begin{bmatrix} \boldsymbol{v}_1^* \\ \boldsymbol{v}_2^* \\ \vdots \\ \boldsymbol{v}_k^* \end{bmatrix} = \mathsf{U}\Sigma\mathsf{V}^*.$$

The *singular values* σ_j are ordered so that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k \geq 0$.

The *left singular vectors* $\boldsymbol{u}_j$ are orthonormal.

The *right singular vectors* $\boldsymbol{v}_j$ are orthonormal.

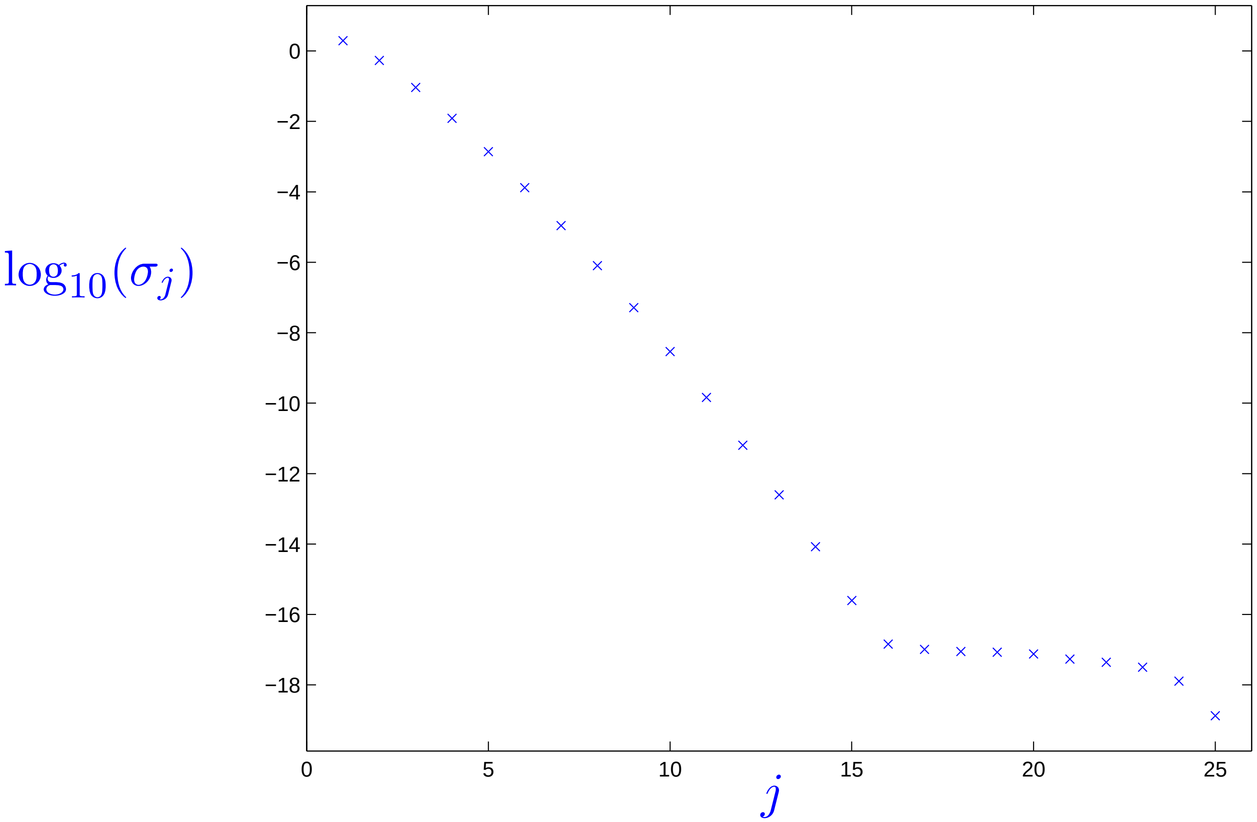
The SVD provides the exact answer to the low rank approximation problem:

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

$$\sum_{i=1}^{j} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^* = \text{argmin}\{||\mathsf{A} - \mathsf{B}|| : \ \mathsf{B} \text{ has rank } j\}.$$

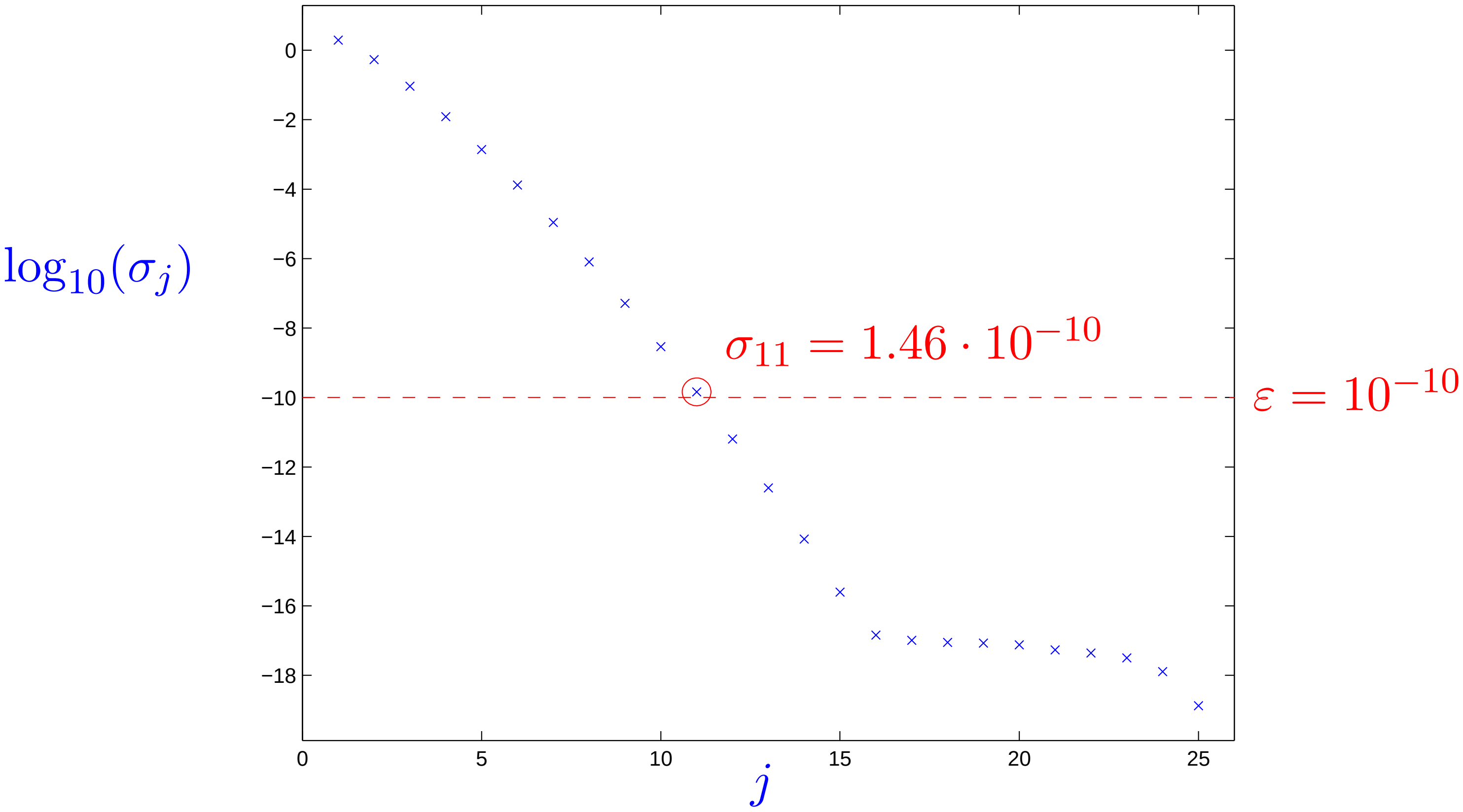
The decay of the singular values determines how well a matrix can be approximated by low-rank factorizations.

Example: Let A be the 25×25 Hilbert matrix, *i.e.* $\mathsf{A}_{ij} = 1/(i + j - 1)$. Let σ_j denote the j'th singular value of A.



The decay of the singular values determines how well a matrix can be approximated by low-rank factorizations.

Example: Let A be the 25×25 Hilbert matrix, *i.e.* $\mathsf{A}_{ij} = 1/(i+j-1)$. Let σ_j denote the j'th singular value of A.



$\sigma_{11} = 1.46 \cdot 10^{-10}$

$\varepsilon = 10^{-10}$

For instance, to precision $\varepsilon = 10^{-10}$, the matrix A has rank 11.

The SVD is a “master algorithm” — if you have the it, you can do anything:

- System solve / least squares.
- Compute other factorizations: LU, QR, eigenvectors, *etc.*
- Data analysis.

Model problem: Given an $n \times n$ matrix A of rank $k \ll n$, compute its SVD

$$\begin{array}{ccccc} \mathsf{A} & = & \mathsf{U} & \Sigma & \mathsf{V}^{*}. \\ n \times n & & n \times k & k \times k & k \times n \end{array}$$

Model problem: Given an $n \times n$ matrix A of rank $k \ll n$, compute its SVD

$$\begin{array}{ccccc} \mathsf{A} & = & \mathsf{U} & \Sigma & \mathsf{V}^*. \\ n \times n & & n \times k & k \times k & k \times n \end{array}$$

A randomized algorithm:

- Draw random vectors $\boldsymbol{\omega}_1, \boldsymbol{\omega}_2, \dots, \boldsymbol{\omega}_k$. (Say from a Gaussian distribution.)

Verification: $\mathsf{A} =$

Model problem: Given an $n \times n$ matrix A of rank $k \ll n$, compute its SVD

$$\begin{array}{ccccc} \mathsf{A} & = & \mathsf{U} & \Sigma & \mathsf{V}^{*}. \\ n \times n & & n \times k & k \times k & k \times n \end{array}$$

A randomized algorithm:

- Draw random vectors $\boldsymbol{\omega}_1,\, \boldsymbol{\omega}_2,\, \dots,\, \boldsymbol{\omega}_k$. (Say from a Gaussian distribution.)

- Compute samples $\boldsymbol{y}_j = \mathsf{A}\,\boldsymbol{\omega}_j$ from $\mathrm{Ran}(\mathsf{A})$. The vectors $\{\boldsymbol{y}_1,\, \boldsymbol{y}_2,\, \dots,\, \boldsymbol{y}_k\}$ are with probability 1 linearly independent and form a basis for the range of A.

Verification: $\mathsf{A} =$

Model problem: Given an $n \times n$ matrix A of rank $k \ll n$, compute its SVD

$$\begin{array}{ccccc} \mathsf{A} & = & \mathsf{U} & \Sigma & \mathsf{V}^*. \\ n \times n & & n \times k & k \times k & k \times n \end{array}$$

A randomized algorithm:

- Draw random vectors $\boldsymbol{\omega}_1,\, \boldsymbol{\omega}_2,\, \dots,\, \boldsymbol{\omega}_k$. (Say from a Gaussian distribution.)
- Compute samples $\boldsymbol{y}_j = \mathsf{A}\,\boldsymbol{\omega}_j$ from $\mathrm{Ran}(\mathsf{A})$. The vectors $\{\boldsymbol{y}_1,\, \boldsymbol{y}_2,\, \dots,\, \boldsymbol{y}_k\}$ are with probability 1 linearly independent and form a basis for the range of A.
- Form an $n \times k$ matrix Q whose columns form an orthonormal basis for the columns of the matrix $\mathsf{Y} = [\boldsymbol{y}_1,\, \boldsymbol{y}_2,\, \dots,\, \boldsymbol{y}_k]$. Then $\mathsf{A} = \mathsf{Q}\mathsf{Q}^*\mathsf{A}$.

Verification: $\mathsf{A} = \mathsf{Q}\mathsf{Q}^*\mathsf{A} =$

Model problem: Given an $n \times n$ matrix A of rank $k \ll n$, compute its SVD

$$\begin{array}{ccccc} \mathsf{A} & = & \mathsf{U} & \Sigma & \mathsf{V}^*. \\ n \times n & & n \times k & k \times k & k \times n \end{array}$$

A randomized algorithm:

- Draw random vectors $\boldsymbol{\omega}_1, \boldsymbol{\omega}_2, \dots, \boldsymbol{\omega}_k$. (Say from a Gaussian distribution.)
- Compute samples $\boldsymbol{y}_j = \mathsf{A}\,\boldsymbol{\omega}_j$ from $\mathrm{Ran}(\mathsf{A})$. The vectors $\{\boldsymbol{y}_1, \boldsymbol{y}_2, \dots, \boldsymbol{y}_k\}$ are with probability 1 linearly independent and form a basis for the range of A.
- Form an $n \times k$ matrix Q whose columns form an orthonormal basis for the columns of the matrix $\mathsf{Y} = [\boldsymbol{y}_1, \boldsymbol{y}_2, \dots, \boldsymbol{y}_k]$. Then $\mathsf{A} = \mathsf{Q}\mathsf{Q}^*\mathsf{A}$.
- Form the $k \times n$ "small" matrix $\mathsf{B} = \mathsf{Q}^*\,\mathsf{A}$. Then $\mathsf{A} = \mathsf{Q}\mathsf{B}$.

Verification: $\mathsf{A} = \mathsf{Q}\mathsf{Q}^*\mathsf{A} = \mathsf{Q}\mathsf{B} =$

Model problem: Given an $n \times n$ matrix A of rank $k \ll n$, compute its SVD

$$\begin{array}{ccccc} \mathsf{A} & = & \mathsf{U} & \Sigma & \mathsf{V}^{*}. \\ n \times n & & n \times k & k \times k & k \times n \end{array}$$

A randomized algorithm:

- Draw random vectors $\boldsymbol{\omega}_1, \boldsymbol{\omega}_2, \dots, \boldsymbol{\omega}_k$. (Say from a Gaussian distribution.)

- Compute samples $\boldsymbol{y}_j = \mathsf{A}\,\boldsymbol{\omega}_j$ from $\mathrm{Ran}(\mathsf{A})$. The vectors $\{\boldsymbol{y}_1, \boldsymbol{y}_2, \dots, \boldsymbol{y}_k\}$ are with probability 1 linearly independent and form a basis for the range of A.

- Form an $n \times k$ matrix Q whose columns form an orthonormal basis for the columns of the matrix $\mathsf{Y} = [\boldsymbol{y}_1, \boldsymbol{y}_2, \dots, \boldsymbol{y}_k]$. Then $\mathsf{A} = \mathsf{Q}\mathsf{Q}^{*}\mathsf{A}$.

- Form the $k \times n$ "small" matrix $\mathsf{B} = \mathsf{Q}^{*}\,\mathsf{A}$. Then $\mathsf{A} = \mathsf{Q}\mathsf{B}$.

- Form the SVD of B (cheap since B is "small") $\mathsf{B} = \hat{\mathsf{U}}\Sigma\mathsf{V}^{*}$.

Verification: $\mathsf{A} = \mathsf{Q}\mathsf{Q}^{*}\mathsf{A} = \mathsf{Q}\mathsf{B} = \mathsf{Q}\hat{\mathsf{U}}\Sigma\mathsf{V}^{*} =$

Model problem: Given an $n \times n$ matrix A of rank $k \ll n$, compute its SVD

$$\begin{array}{ccccc} \mathsf{A} & = & \mathsf{U} & \Sigma & \mathsf{V}^*. \\ n \times n & & n \times k & k \times k & k \times n \end{array}$$

A randomized algorithm:

- Draw random vectors $\boldsymbol{\omega}_1, \boldsymbol{\omega}_2, \dots, \boldsymbol{\omega}_k$. (Say from a Gaussian distribution.)

- Compute samples $\boldsymbol{y}_j = \mathsf{A}\,\boldsymbol{\omega}_j$ from Ran(A). The vectors $\{\boldsymbol{y}_1, \boldsymbol{y}_2, \dots, \boldsymbol{y}_k\}$ are with probability 1 linearly independent and form a basis for the range of A.

- Form an $n \times k$ matrix Q whose columns form an orthonormal basis for the columns of the matrix $\mathsf{Y} = [\boldsymbol{y}_1, \boldsymbol{y}_2, \dots, \boldsymbol{y}_k]$. Then $\mathsf{A} = \mathsf{Q}\mathsf{Q}^*\mathsf{A}$.

- Form the $k \times n$ "small" matrix $\mathsf{B} = \mathsf{Q}^*\,\mathsf{A}$. Then $\mathsf{A} = \mathsf{Q}\mathsf{B}$.

- Form the SVD of B (cheap since B is "small") $\mathsf{B} = \hat{\mathsf{U}}\Sigma\mathsf{V}^*$.

- Form $\mathsf{U} = \mathsf{Q}\hat{\mathsf{U}}$.

Verification: $\mathsf{A} = \mathsf{Q}\mathsf{Q}^*\mathsf{A} = \mathsf{Q}\mathsf{B} = \mathsf{Q}\hat{\mathsf{U}}\Sigma\mathsf{V}^* = \mathsf{U}\Sigma\mathsf{V}^*$.

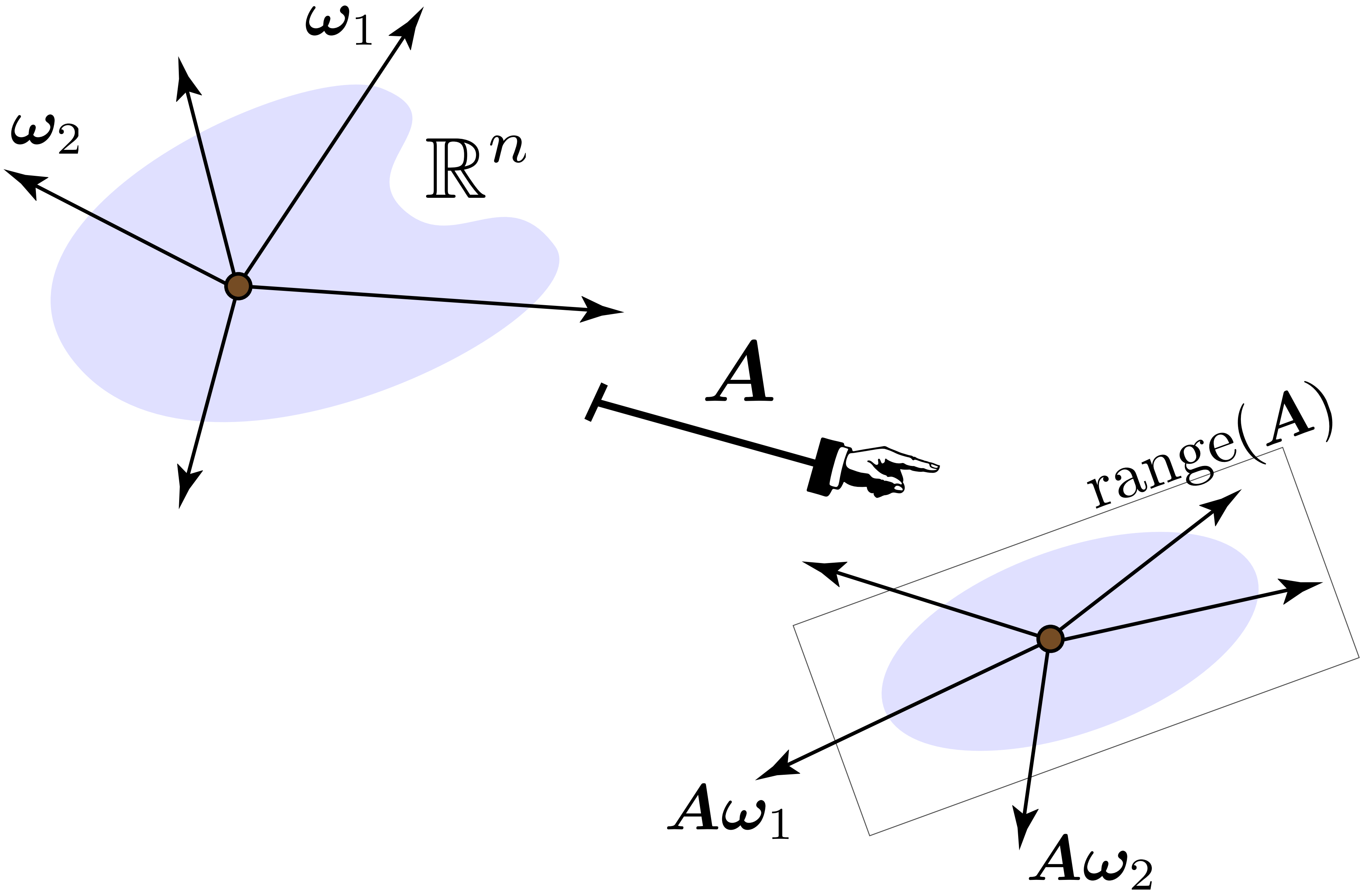
Model problem: Given an $n \times n$ matrix A of rank $k \ll n$, compute its SVD

$$\begin{array}{ccccc} \mathsf{A} & = & \mathsf{U} & \Sigma & \mathsf{V}^*. \\ n \times n & & n \times k & k \times k & k \times n \end{array}$$

A randomized algorithm — blocked (much faster):

- Draw an $n \times k$ random matrix Ω. (Say from a Gaussian distribution.)
- Compute an $n \times k$ sample matrix $\mathsf{Y} = \mathsf{A}\Omega$. The columns of Y are with probability 1 linearly independent and form a basis for the range of A.
- Form an $n \times k$ matrix Q whose columns form an orthonormal basis for the columns of the matrix Y. Then $\mathsf{A} = \mathsf{Q}\mathsf{Q}^*\mathsf{A}$.
- Form the $k \times n$ "small" matrix $\mathsf{B} = \mathsf{Q}^*\,\mathsf{A}$. Then $\mathsf{A} = \mathsf{Q}\mathsf{B}$.
- Form the SVD of B (cheap since B is "small") $\mathsf{B} = \hat{\mathsf{U}}\Sigma\mathsf{V}^*$.
- Form $\mathsf{U} = \mathsf{Q}\hat{\mathsf{U}}$.

Verification: $\mathsf{A} = \mathsf{Q}\mathsf{Q}^*\mathsf{A} = \mathsf{Q}\mathsf{B} = \mathsf{Q}\hat{\mathsf{U}}\Sigma\mathsf{V}^* = \mathsf{U}\Sigma\mathsf{V}^*$.


$\boldsymbol{\omega}_1$
$\boldsymbol{\omega}_2$
$\mathbb{R}^n$
$\boldsymbol{A}$
$\mathrm{range}(\boldsymbol{A})$
$\boldsymbol{A}\boldsymbol{\omega}_1$
$\boldsymbol{A}\boldsymbol{\omega}_2$

Given an $n \times n$ matrix A of rank k, compute its Singular Value Decomposition $\mathsf{A} = \mathsf{U}\Sigma\mathsf{V}^*$.

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

(2) Form the $n \times k$ sample matrix $\mathsf{Y} = \mathsf{A}\Omega$.

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

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

(5) Factor the small matrix $\mathsf{B} = \hat{\mathsf{U}}\Sigma\mathsf{V}^*$.

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

Let us compare the randomized method to a standard deterministic method — the so called *Golub-Businger algorithm.* It has two steps:

1. Determine an orthonormal basis $\{\boldsymbol{q}_j\}_{j=1}^{k}$ for the range of A.
   This can typically be done via a simple Gram-Schmidt processes.
   Set $\mathsf{Q} = [\boldsymbol{q}_1,\, \boldsymbol{q}_2,\, \dots,\, \boldsymbol{q}_k]$.
   (There are some complications — resolved by Gu and Eisenstat in 1997.)

2. Form $\mathsf{B} = \mathsf{Q}^*\,\mathsf{A}$ and proceed as in the randomized algorithm above.
   (Actually, Gram-Schmidt gives you the factor B directly — it is the "R" factor in the QR-factorization of A.)

The difference lies simply in how we form the basis for the range of A.

While Golub-Businger restricts A to a basis that assuredly spans the range of the matrix, the randomized algorithm restricts A to the range of the sample matrix

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

where $\mathsf{\Omega}$ is a random matrix.

The appeal of the randomized method is that the product $\mathsf{A}\mathsf{\Omega}$ can be evaluated in a single sweep (and is amenable to BLAS3, *etc*).

Question: Is this not dangerous? Things could fail, right?

Given an $n \times n$ matrix A of rank k, compute its Singular Value Decomposition $\mathsf{A} = \mathsf{U}\Sigma\mathsf{V}^*$.

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

(2) Form the $n \times k$ sample matrix $\mathsf{Y} = \mathsf{A}\Omega$.

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

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

(5) Factor the small matrix $\mathsf{B} = \hat{\mathsf{U}}\Sigma\mathsf{V}^*$.

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

Theorem: The probability that $\mathsf{A} \neq \mathsf{U}\Sigma\mathsf{V}^*$ is zero if exact arithmetic is used.

Proof:

$$\begin{aligned}
\text{The method fails.} \quad &\Leftrightarrow \quad \text{The columns of } \mathsf{Y} \text{ are not linearly independent.} \\
&\Leftrightarrow \quad \alpha_1\,\boldsymbol{y}_1 + \alpha_2\,\boldsymbol{y}_2 + \cdots + \alpha_k\,\boldsymbol{y}_k = 0 \\
&\Leftrightarrow \quad \mathsf{A}\left(\alpha_1\,\boldsymbol{\omega}_1 + \alpha_2\,\boldsymbol{\omega}_2 + \cdots + \alpha_k\,\boldsymbol{\omega}_k\right) = 0 \\
&\Leftrightarrow \quad \alpha_1\,\boldsymbol{\omega}_1 + \alpha_2\,\boldsymbol{\omega}_2 + \cdots + \alpha_k\,\boldsymbol{\omega}_k \in \mathrm{Null}(\mathsf{A})
\end{aligned}$$

Null(A) is a linear subspace of dimension $k < n$, so it has measure zero.

Note: When executed in floating point arithmetic, the toy algorithm can behave poorly since $\{\boldsymbol{y}_1,\,\boldsymbol{y}_2,\,\dots,\,\boldsymbol{y}_k\}$ may well be an ill-conditioned basis for Ran(A).

This issue will be dealt with by taking a few extra samples.

Given an $n \times n$ matrix A of rank k, compute its Singular Value Decomposition $A = U\Sigma V^*$.	
(1) Draw an $n \times k$ random matrix Ω.	(4) Form the small matrix $B = Q^*\,A$.
(2) Form the $n \times k$ sample matrix $Y = A\Omega$.	(5) Factor the small matrix $B = \hat{U}\Sigma V^*$.
(3) Compute an ON matrix Q s.t. $Y = QQ^*Y$.	(6) Form $U = Q\hat{U}$.

Notice that in the absence of a zero-probability event, the output of the algorithm is exact.

The goal of the first three steps of the algorithm is to find the linear dependencies between the rows of A. These dependencies are preserved *exactly* in Y.

Note that the algorithm is quite unlike Monte Carlo sampling, or standard Johnson-Lindenstrauss techniques in this regard. We do not need distances to be preserved to high accuracy, we only need them to not be disastrously distorted (which would lead to ill-conditioning in this toy problem).

We will return to this issue later.

Preview: We will return to this point in great length, but for curiosity's sake, let us simply mention that the case where A only has approximate rank k is handled by introducing a small amount of *over-sampling.*

Let p denote a small parameter. Say $p = 5$. Then the new algorithm is:

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

(2) Compute an $n \times (k+p)$ sample matrix $\mathsf{Y} = \mathsf{A}\mathsf{\Omega}$.

(3) Compute an $n \times (k+p)$ ON matrix Q s.t. $\mathsf{Y} = \mathsf{Q}\mathsf{Q}^{*}\mathsf{Y}$.

(4) Compute the $(k+p) \times n$ matrix $\mathsf{B} = \mathsf{Q}^{*}\,\mathsf{A}$.

(5) Compute SVD $\mathsf{B} = \hat{\mathsf{U}}\mathsf{\Sigma}\mathsf{V}^{*}$.

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

(7) Truncate the approximate SVD $\mathsf{A} = \mathsf{U}\mathsf{\Sigma}\mathsf{V}^{*}$ to the first k components.

Preview: We will return to this point as well, but let us quickly assuage any concerns that the rank must be known in advance.

First we observe that the vectors in the algorithm can be computed sequentially.

```
1. Set $j = 0$.
2. Draw a random vector $\boldsymbol{\omega}_{j+1}$.
3. Compute a sample vector $\boldsymbol{y}_{j+1} = \mathsf{A}\boldsymbol{\omega}_{j+1}$.
4. Let $\boldsymbol{z}$ denote the projection of $\boldsymbol{y}_{j+1}$ onto $\big(\text{Span}(\boldsymbol{y}_1,\,\boldsymbol{y}_2,\,\dots,\,\boldsymbol{y}_j\}\big)^{\perp}$.
   IF $\boldsymbol{z} \neq 0$ THEN
       $\boldsymbol{q}_{j+1} = \boldsymbol{z}$.
       $j = j + 1$.
       GOTO (2).
   END IF
5. Set $k = j$ and $\mathsf{Q} = [\boldsymbol{q}_1,\,\boldsymbol{q}_2,\,\dots,\,\boldsymbol{q}_k]$.
6. $\mathsf{B} = \mathsf{Q}^*\,\mathsf{A}$.
7. $\mathsf{B} = \hat{\mathsf{U}}\Sigma\mathsf{V}^*$.
8. $\mathsf{U} = \mathsf{Q}\hat{\mathsf{U}}$.
```

The algorithm where the vectors are computed sequentially requires multiple passes over the data.

This drawback can be dealt with:

- Blocking is possible.
- Rank "guessing" strategies where the rank is doubled every step can be implemented.

The key is that *we can estimate how well we are doing.*
The estimators that we use are typically random themselves.

Other randomized sampling strategies have been suggested, often precisely with the goal of overcoming the obstacles faced by huge matrices, streamed data, *etc.*

Many algorithms are based on randomly pulling columns or rows from the matrix.

- Random subselection of columns to form a basis for the column space.
  - Can be improved by computing probability weights based on, *e.g.*, magnitude of elements.
- Randomly drawn submatrices give indication of the spectrum / determinant / norm of the matrix.
- Random sparsification by zeroing out elements.

When the matrix A is itself in some sense sampled from a restricted probability space, such algorithms can perform well. Sub-linear complexity can be achieved.

However, these methods typically give inaccurate results for sparse matrices. Consider a matrix consisting of all zeros with a randomly placed entry 1. Basically, unless you sample *a lot*, you may easily miss important information.

The randomized methods of this talk are slightly more expensive, but give highly accurate results, with (provably) extremely high probability ($1 - 10^{-15}$ is typical).

Given an $n \times n$ matrix A of rank k, compute its Singular Value Decomposition $\mathsf{A} = \mathsf{U}\Sigma\mathsf{V}^*$.	
(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 $\mathsf{U} = \mathsf{Q}\hat{\mathsf{U}}$.

At this point, we have described some features of a basic algorithm and compared it to other deterministic and randomized methods.

Next, we will describe some variations on the basic pattern:

- How to obtain a single-pass method.
  In particular, we need to eliminate the matrix-matrix product in "Step 4."

- How to determine spanning rows and spanning columns.

Outline of the tutorial:

1. Techniques for computing the SVD of a matrix of exact rank k.
2. Variations of techniques for matrices of exact rank.
   - Single pass algorithms.
   - How to compute spanning rows and spanning columns (CUR, etc).
3. Techniques for matrices of the form $\mathsf{A} = \mathsf{E}\mathsf{F} + \mathsf{N}$ with N small.
   - Error estimation.
4. Techniques for matrices of the form $\mathsf{A} = \mathsf{E}\mathsf{F} + \mathsf{N}$ with N large.
   - The "power method."
5. Random transforms that can be applied rapidly.
   - The "Subsampled Random Fourier Transform" (SRFT) and its cousins.
6. Review / Putting things together / Model problems.

A single pass algorithm for a symmetric matrix: Start as before:

(1) Draw random $\mathsf{\Omega}$	(2) Compute sample $\mathsf{Y} = \mathsf{A}\,\mathsf{\Omega}$	(3) Compute Q such that $\mathsf{Y} = \mathsf{Q}\mathsf{Q}^{*}\mathsf{Y}$

Since $\mathsf{A} = \mathsf{Q}\mathsf{Q}^{*}\mathsf{A}$ and A is symmetric:
$$\mathsf{A} = \mathsf{Q}\,\mathsf{Q}^{*}\,\mathsf{A}\,\mathsf{Q}\,\mathsf{Q}^{*} \qquad (1)$$

Set $\mathsf{B} = \mathsf{Q}^{*}\,\mathsf{A}\,\mathsf{Q}$, then:
$$\mathsf{A} = \mathsf{Q}\,\mathsf{B}\,\mathsf{Q}^{*} \qquad (2)$$

Right-multiply (2) by $\mathsf{\Omega}$ and recall that $\mathsf{Y} = \mathsf{A}\,\mathsf{\Omega}$:
$$\mathsf{Y} = \mathsf{Q}\,\mathsf{B}\,\mathsf{Q}^{*}\,\mathsf{\Omega} \qquad (3)$$

Left-multiply (3) by Q^{*} to obtain:
$$\mathsf{Q}^{*}\mathsf{Y} = \mathsf{B}\,\mathsf{Q}^{*}\,\mathsf{\Omega} \qquad (4)$$

The blue factors in (4) are known, so we can solve for B.

Form the eigenvalue decomposition of B:
$$\mathsf{B} = \hat{\mathsf{U}}\,\mathsf{\Lambda}\,\hat{\mathsf{U}}^{*} \qquad (5)$$

Insert (5) into (2) and set $\mathsf{U} = \mathsf{Q}\,\hat{\mathsf{U}}$. Then:
$$\mathsf{A} = \mathsf{U}\,\mathsf{\Lambda}\,\mathsf{U}^{*} \qquad (6)$$

A single pass algorithm, continued:

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 B the linear system $\mathsf{Q}^*\,\mathsf{Y} = \mathsf{B}\,(\mathsf{Q}^*\,\Omega)$.
Factor B so that $\mathsf{B} = \hat{\mathsf{U}}\,\Lambda\,\hat{\mathsf{U}}^*$.
Form $\mathsf{U} = \mathsf{Q}\,\hat{\mathsf{U}}$.
Output: $\mathsf{A} = \mathsf{U}\Lambda\mathsf{U}^*$

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

A single pass algorithm, continued:

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 B the linear system $Q^*\,Y = B\,(Q^*\,\Omega)$.	Solve for B the linear systems $Q^*\,Y = B\,(W^*\,\Omega)$ and $W^*\,Z = B^*\,(Q^*\,\Psi)$.
Factor B so that $B = \hat{U}\,\Lambda\,\hat{U}^*$.	Factor B so that $B = \hat{U}\,\Sigma\,\hat{V}^*$.
Form $U = Q\,\hat{U}$.	Form $U = Q\,\hat{U}$ and $V = W\,\hat{V}$.
Output: $A = U\Lambda U^*$	Output: $A = U\Sigma V^*$

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

Outline of the tutorial:

1. Techniques for computing the SVD of a matrix of exact rank k.

2. Variations of techniques for matrices of exact rank.
   - Single pass algorithms.
   - How to compute spanning rows and spanning columns (CUR, etc).

3. Techniques for matrices of the form $\mathsf{A} = \mathsf{EF} + \mathsf{N}$ with N small.
   - Error estimation.

4. Techniques for matrices of the form $\mathsf{A} = \mathsf{EF} + \mathsf{N}$ with N large.
   - The "power method."

5. Random transforms that can be applied rapidly.
   - The "Subsampled Random Fourier Transform" (SRFT) and its cousins.

6. Review / Putting things together / Model problems.

Finding "spanning columns" of a matrix:

Let A be an $n \times n$ matrix of rank k with columns $\mathsf{A} = [\boldsymbol{a}_1 \ \boldsymbol{a}_2 \ \cdots \ \boldsymbol{a}_n]$.

We are interested in finding an index vector $J = [j_1,\, j_2,\, \dots,\, j_k]$ such that

$$\text{(1)} \qquad \mathrm{Ran}(\mathsf{A}) = \mathrm{Span}\{\boldsymbol{a}_1,\, \boldsymbol{a}_2,\, \dots,\, \boldsymbol{a}_n\} = \mathrm{Span}\{\boldsymbol{a}_{j_1},\, \boldsymbol{a}_{j_2},\, \dots,\, \boldsymbol{a}_{j_k}\}.$$

In matrix notation, (1) says that for some $k \times n$ matrix X we have

$$\begin{array}{ccccc} \mathsf{A} & = & \mathsf{A}(:\,,J) & & \mathsf{X}. \\ n \times n & & n \times k & & k \times n \end{array}$$

The matrix X contains the $k \times k$ identity matrix.

Question: Why is it of interest to find spanning columns (or rows)?

- The matrix A may be structured; say each column is sparse (or "data-sparse").
- Useful in data analysis. Suppose for instance that each column of A represents the price history of a stock. The stocks marked by J "explain" the market.

Finding “spanning columns” of a matrix $\mathsf{A} = [\boldsymbol{a}_1\ \boldsymbol{a}_2\ \cdots\ \boldsymbol{a}_n]$

Question: How do you do it for a small matrix?

Answer: Plain column-pivoted Gram-Schmidt usually does the job.

- Let j_1 be the index of the largest column of A. Set $\mathsf{A}^{(1)} = \mathsf{P}^{\perp}_{\boldsymbol{a}_{j_1}}\,\mathsf{A}$, where $\mathsf{P}^{\perp}_{\boldsymbol{a}}$ is the orthogonal projection onto $\text{Span}(\boldsymbol{a})^{\perp}$.

- Let j_2 be the index of the largest column of $\mathsf{A}^{(1)}$. Set $\mathsf{A}^{(2)} = \mathsf{P}^{\perp}_{\boldsymbol{a}_{j_2}}\,\mathsf{A}^{(1)}$.

- Let j_3 be the index of the largest column of $\mathsf{A}^{(2)}$. Set $\mathsf{A}^{(3)} = \mathsf{P}^{\perp}_{\boldsymbol{a}_{j_3}}\,\mathsf{A}^{(2)}$.

- *Etc.*

Caveat 1: It is imperative that orthonormality be preserved.

Caveat 2: There is some controversy about whether Gram-Schmidt is a good choice for this task. In certain situation (which could be claimed to be unusual) more sophisticated algorithms might be prescribed; for instance the *Rank-Revealing QR* algorithm of Gu and Eisenstat.

Finding “spanning columns” of a matrix $\mathsf{A} = [\boldsymbol{a}_1\ \boldsymbol{a}_2\ \cdots\ \boldsymbol{a}_n]$

Now, suppose that A is very large, but we *are given* a factorization

$$(2)\qquad \begin{array}{ccccc} \mathsf{A} & = & \mathsf{E} & \mathsf{Z}. \\ n \times n & & n \times k & k \times n \end{array}$$

Determine k spanning columns of Z:

$$\mathsf{Z} = \mathsf{Z}(:\,,J)\mathsf{X},$$

where X is a $k \times n$ matrix such that $\mathsf{X}(:\,,J) = \mathsf{I}$. Then

$$(3)\qquad \mathsf{A} = \mathsf{E}\mathsf{Z}(:\,,J)\mathsf{X}.$$

Restricting A to the columns in J, we find that

$$(4)\qquad \mathsf{A}(:\,,J) = \mathsf{E}\mathsf{Z}(:\,,J)\underbrace{\mathsf{X}(:\,,J)}_{=\mathsf{I}} = \mathsf{E}\mathsf{Z}(:\,,J).$$

Combining (3) and (4), we obtain:

$$(5)\qquad \mathsf{A} = \mathsf{A}(:\,,J)\mathsf{X}.$$

The matrices A and E are never used! Now, how do you find Z?

Given a large $n \times n$ matrix A of rank k, we seek a matrix Z such that

$$\underset{n \times n}{\mathsf{A}} \quad = \quad \underset{n \times k}{\mathsf{E}} \quad \underset{k \times n}{\mathsf{Z}}. \tag{6}$$

Given a large $n \times n$ matrix A of rank k, we seek a matrix Z such that

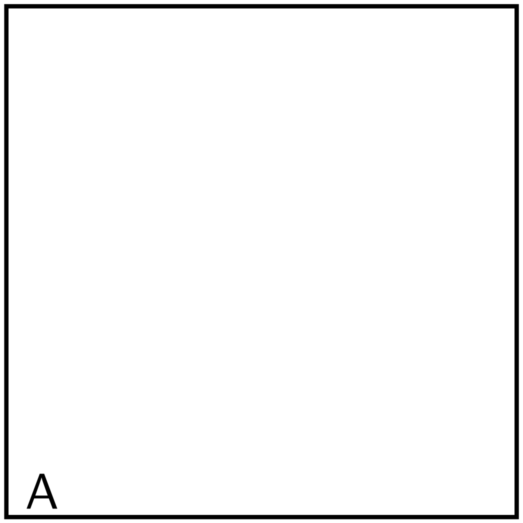
$$\underset{n \times n}{\mathsf{A}} = \underset{n \times k}{\mathsf{E}} \quad \underset{k \times n}{\mathsf{Z}}. \tag{6}$$

It can be determined via the randomized sampling technique.

1. Draw an $n \times k$ Gaussian random matrix $\mathsf{\Omega}$.
2. Form the sample matrix $\mathsf{Z} = \mathsf{\Omega}^{*}\mathsf{A}$.
   Then with probability one, (6) holds for some (unknown!) matrix E.
3. Determine k spanning columns of Z so that $\mathsf{Z} = \mathsf{Z}(:\,, J)\mathsf{X}$.
4. Do nothing! Just observe that, automatically, $\mathsf{A} = \mathsf{A}(:\,, J)\mathsf{X}$.

Given: An $n \times n$ matrix A of rank k.

Task: Find k spanning columns of A.

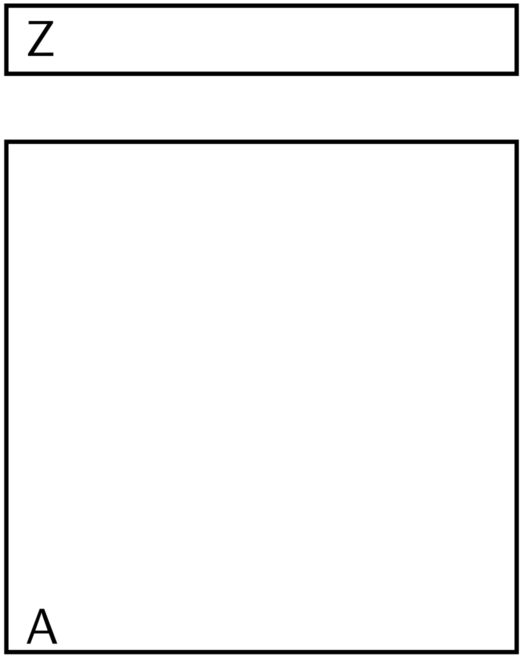


Given: An $n \times n$ matrix A of rank k.

Task: Find k spanning columns of A.

1. Generate an $n \times k$ random matrix Ω.
2. Generate the sample matrix

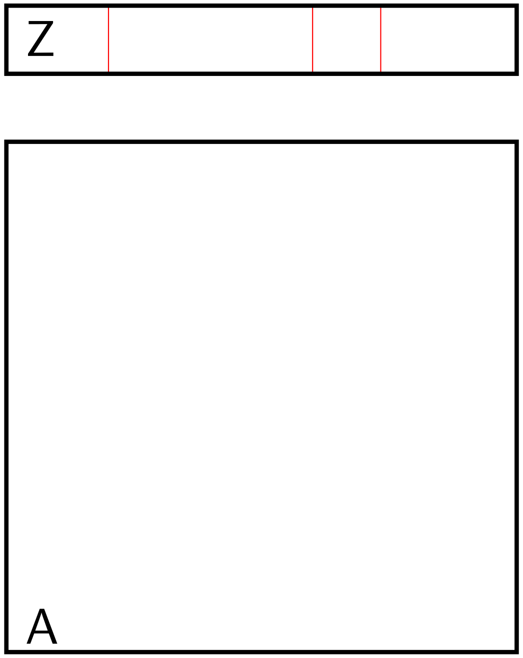
$$\mathsf{Z} = \Omega^{*}\mathsf{A}.$$



Given: An $n \times n$ matrix A of rank k.

Task: Find k spanning columns of A.

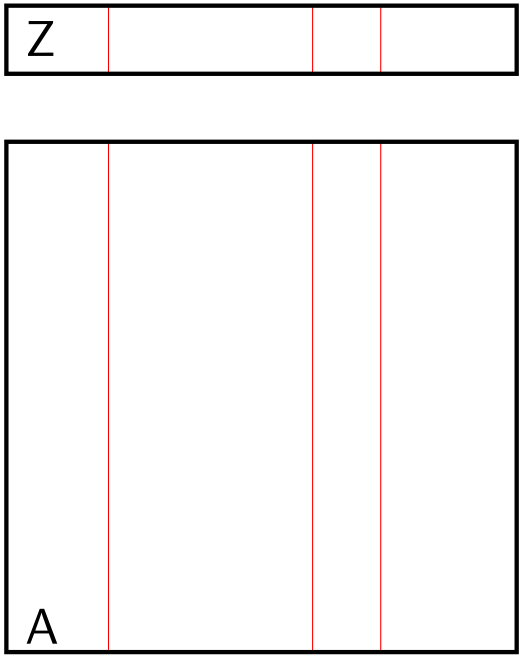
1. Generate an $n \times k$ random matrix Ω.
2. Generate the sample matrix
$$\mathsf{Z} = \Omega^{*}\mathsf{A}.$$
3. Find spanning columns of Z:
$$\mathsf{Z} = \mathsf{Z}(:\,,J)\mathsf{X}.$$



Given: An $n \times n$ matrix A of rank k.

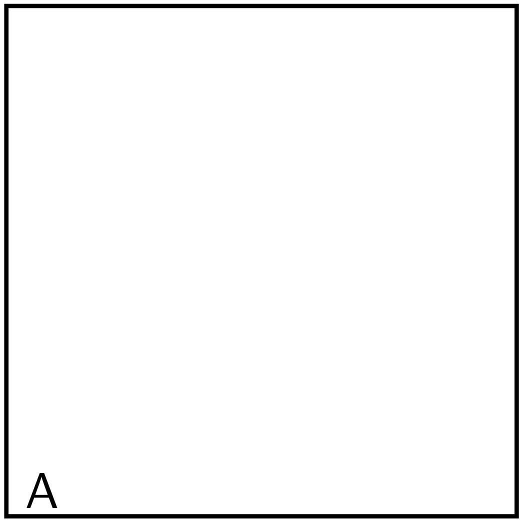
Task: Find k spanning columns of A.

1. Generate an $n \times k$ random matrix Ω.
2. Generate the sample matrix
$$\mathsf{Z} = \Omega^* \mathsf{A}.$$
3. Find spanning columns of Z:
$$\mathsf{Z} = \mathsf{Z}(:\,, J)\mathsf{X}.$$
4. Do nothing. Simply observe that now:
$$\mathsf{A} = \mathsf{A}(:\,, J)\mathsf{X}.$$



Given: An $n \times n$ matrix A of rank k.

Task: Find k spanning columns and rows of A.



Given: An $n \times n$ matrix A of rank k.

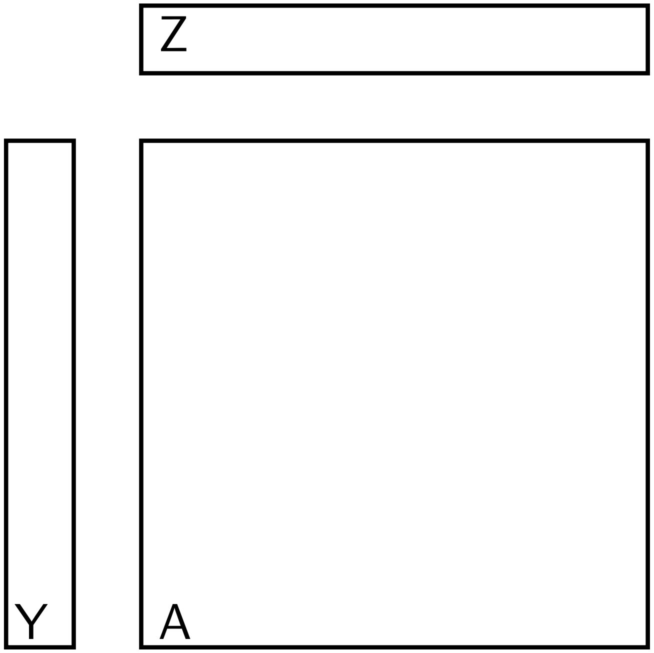
Task: Find k spanning columns and rows of A.

1. Generate an $n \times k$ random matrix $\mathsf{\Omega}$.
2. Generate the sample matrices

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

and

$$\mathsf{Z} = \mathsf{\Omega}^{*}\mathsf{A}.$$



Given: An $n \times n$ matrix A of rank k.

Task: Find k spanning columns and rows of A.

1. Generate an $n \times k$ random matrix $\mathsf{\Omega}$.
2. Generate the sample matrices

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

   and

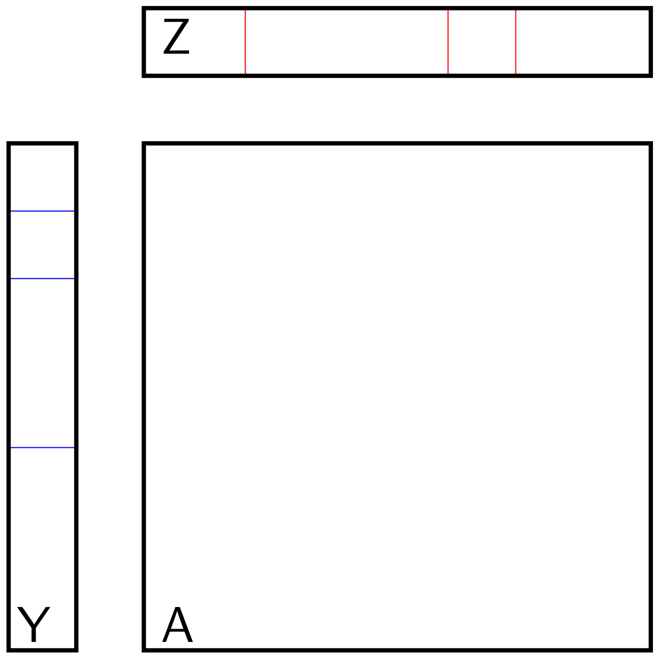
$$\mathsf{Z} = \mathsf{\Omega}^*\mathsf{A}.$$

3. Find spanning rows of Y

$$\mathsf{Y} = \mathsf{X}_{\rm row}\mathsf{Y}(J_{\rm row},\,:\,)$$

   and spanning columns of Z:

$$\mathsf{Z} = \mathsf{Z}(\,:\,,J_{\rm col})\mathsf{X}_{\rm col}.$$



Given: An $n \times n$ matrix A of rank k.

Task: Find k spanning columns and rows of A.

1. Generate an $n \times k$ random matrix Ω.
2. Generate the sample matrices

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

and

$$\mathsf{Z} = \Omega^{*}\mathsf{A}.$$

3. Find spanning rows of Y

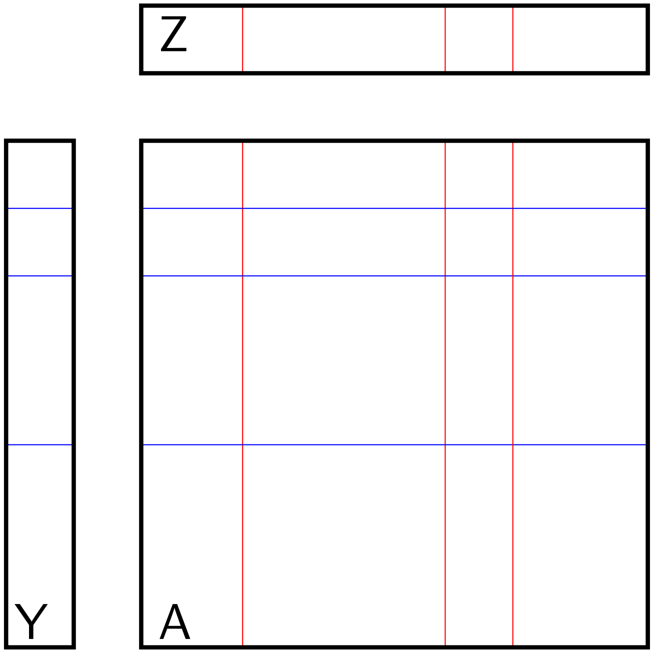
$$\mathsf{Y} = \mathsf{X}_{\rm row}\mathsf{Y}(J_{\rm row},\,:\,)$$

and spanning columns of Z:

$$\mathsf{Z} = \mathsf{Z}(\,:\,,J_{\rm col})\mathsf{X}_{\rm col}.$$

4. Do nothing. Simply observe that now:

$$\mathsf{A} = \mathsf{X}_{\rm row}\mathsf{A}(J_{\rm row},J_{\rm col})\mathsf{X}_{\rm col}.$$



Given: An $n \times n$ matrix A of rank k.

Task: Find k spanning columns and rows of A.

1. Generate an $n \times k$ random matrix Ω.
2. Generate the sample matrices

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

and

$$\mathsf{Z} = \Omega^{*}\mathsf{A}.$$

3. Find spanning rows of Y

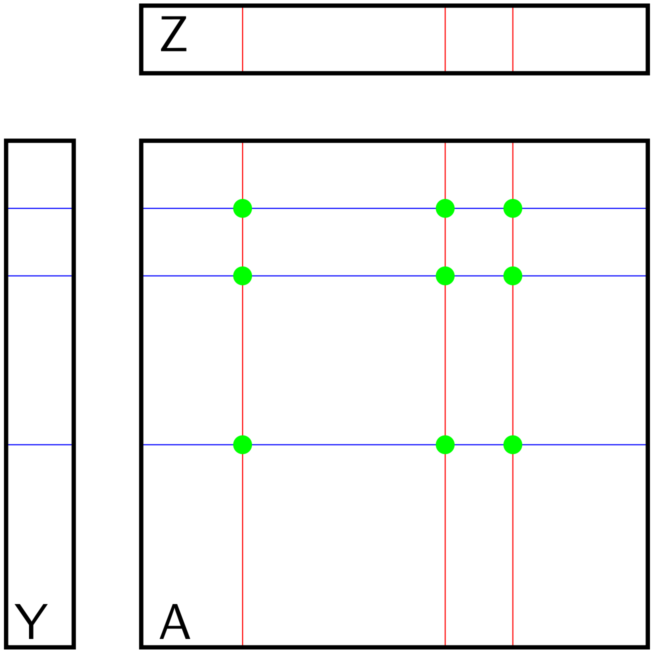
$$\mathsf{Y} = \mathsf{X}_{\text{row}}\mathsf{Y}(J_{\text{row}}, :)$$

and spanning columns of Z:

$$\mathsf{Z} = \mathsf{Z}(:, J_{\text{col}})\mathsf{X}_{\text{col}}.$$

4. Do nothing! Simply observe that now:

$$\mathsf{A} = \mathsf{X}_{\text{row}}\mathsf{A}(J_{\text{row}}, J_{\text{col}})\mathsf{X}_{\text{col}}.$$



The green dots mark the matrix

$\mathsf{A}_{\text{skel}} = \mathsf{A}(J_{\text{row}}, J_{\text{col}})$

With this definition:

$\mathsf{A} = \mathsf{X}_{\text{row}}\mathsf{A}_{\text{skel}}\mathsf{X}_{\text{col}}$

Recall from the previous slide that we have constructed a factorization

$$\mathsf{A} = \mathsf{X}_{\rm row}\mathsf{A}_{\rm skel}\mathsf{X}_{\rm col} \tag{7}$$

where $\mathsf{A}_{\rm skel}$ is the $k \times k$ submatrix of A given by

$$\mathsf{A}_{\rm skel} = \mathsf{A}(J_{\rm row}, J_{\rm col}). \tag{8}$$

The factorization (7) was obtained from

- A single sweep of A to construct the samples $\mathsf{Y} = \mathsf{A}\Omega$ and $\mathsf{Z} = \Omega^{*}\mathsf{A}$.
- Inexpensive operations on the small matrices Y and Z.
- The extraction of the (small!) $k \times k$ matrix $\mathsf{A}_{\rm skel}$.

Observation 1: The factorization (7) can be converted to the SVD

$$\mathsf{A} = \mathsf{U}\Sigma\mathsf{V}^{*}$$

via three simple steps:

Form the QR factorizations:	$\mathsf{X}_{\rm row} = \mathsf{Q}_{\rm row}\,\mathsf{R}_{\rm row}$ and $\mathsf{X}_{\rm col} = \mathsf{R}_{\rm col}^{*}\mathsf{Q}_{\rm col}^{*}$.
Form the SVD of a small matrix:	$\mathsf{R}_{\rm row}\mathsf{A}_{\rm skel}\mathsf{R}_{\rm col}^{*} = \hat{\mathsf{U}}\Sigma\hat{\mathsf{V}}^{*}$.
Form products:	$\mathsf{U} = \mathsf{Q}_{\rm row}\hat{\mathsf{U}}$ and $\mathsf{V} = \mathsf{Q}_{\rm col}\hat{\mathsf{V}}$.

Recall from the previous slide that we have constructed a factorization

$$\mathsf{A} = \mathsf{X}_{\rm row}\mathsf{A}_{\rm skel}\mathsf{X}_{\rm col} \tag{7}$$

where $\mathsf{A}_{\rm skel}$ is the $k \times k$ submatrix of A given by

$$\mathsf{A}_{\rm skel} = \mathsf{A}(J_{\rm row},\,J_{\rm col}). \tag{8}$$

The factorization (7) was obtained from

- A single sweep of A to construct the samples $\mathsf{Y} = \mathsf{A}\Omega$ and $\mathsf{Z} = \Omega^{*}\mathsf{A}$.
- Inexpensive operations on the small matrices Y and Z.
- The extraction of the (small!) $k \times k$ matrix $\mathsf{A}_{\rm skel}$.

Observation 2: The factorization (7) can be converted to a "CUR" factorization:

$$\mathsf{A} = \mathsf{X}_{\rm row}\,\mathsf{A}_{\rm skel}\,\big(\mathsf{A}_{\rm skel}\big)^{-1}\,\mathsf{A}_{\rm skel}\,\mathsf{X}_{\rm col} = \mathsf{C}\,\mathsf{U}\,\mathsf{R},$$

where

- $\mathsf{C} = \mathsf{X}_{\rm row}\,\mathsf{A}_{\rm skel} = \mathsf{A}(:\,,J_{\rm col})$ is a matrix consisting of columns of A.
- $\mathsf{R} = \mathsf{A}_{\rm skel}\,\mathsf{X}_{\rm col} = \mathsf{A}(J_{\rm row},\,:\,)$ is a matrix consisting of rows of A.
- $\mathsf{U} = \big(\mathsf{A}_{\rm skel}\big)^{-1}$ is a "linking matrix." It could be ill-conditioned.

It is time to proceed to the case of matrices that do not have exact rank k. But before we do, let us clean up in the plethora of algorithms a bit.

First we observe that all the algorithms start the same way:

Given: An $n \times n$ matrix A.

Task: Compute a k-term SVD $\mathsf{A} = \mathsf{U}\Sigma\mathsf{V}^*$.

Stage A:	(1) Draw an $n \times k$ random matrix Ω.
(Recurring part)	(2) Compute a sample matrix $\mathsf{Y} = \mathsf{A}\Omega$.
	(3) Compute an ON matrix Q such that $\mathsf{Y} = \mathsf{Q}\mathsf{Q}^*\mathsf{Y}$.
Stage B:	(4) Form $\mathsf{B} = \mathsf{Q}^*\,\mathsf{A}$.
	(5) Factorize $\mathsf{B} = \hat{\mathsf{U}}\Sigma\mathsf{V}^*$.
	(6) Form $\mathsf{U} = \mathsf{Q}\hat{\mathsf{U}}$.

It is time to proceed to the case of matrices that do not have exact rank k. But before we do, let us clean up in the plethora of algorithms a bit.

First we observe that all the algorithms start the same way:

Given: An $n \times n$ matrix A.

Task: Compute a k-term eigenvalue decomposition $\mathsf{A} = \mathsf{U}\Lambda\mathsf{U}^*$ in one pass.

Stage A:	(1) Draw an $n \times k$ random matrix Ω.
(Recurring part)	(2) Compute a sample matrix $\mathsf{Y} = \mathsf{A}\Omega$.
	(3) Compute an ON matrix Q such that $\mathsf{Y} = \mathsf{Q}\mathsf{Q}^*\mathsf{Y}$.
Stage B:	(4) Solve $\mathsf{Q}^*\mathsf{Y} = \mathsf{B}\big(\mathsf{Q}^*\Omega\big)$ for B.
	(5) Factor the reduced matrix $\mathsf{B} = \hat{\mathsf{U}}\Lambda\hat{\mathsf{U}}^*$.
	(6) Form the product $\mathsf{U} = \mathsf{Q}\hat{\mathsf{U}}$.

It is time to proceed to the case of matrices that do not have exact rank k. But before we do, let us clean up in the plethora of algorithms a bit.

First we observe that all the algorithms start the same way:

Given: An $n \times n$ matrix A.

Task: Find spanning rows of A.

Stage A: (Recurring part)	(1) Draw an $n \times k$ random matrix $\mathsf{\Omega}$. (2) Compute a sample matrix $\mathsf{Y} = \mathsf{A}\mathsf{\Omega}$. (3) Compute an ON matrix Q such that $\mathsf{Y} = \mathsf{Q}\mathsf{Q}^{*}\mathsf{Y}$.
Stage B:	(4) Find spanning rows of Y so that $\mathsf{Y} = \mathsf{X}\mathsf{Y}(J,:)$. (5) Simply observe that now $\mathsf{A} = \mathsf{X}\mathsf{A}(J,:)$.

(Note that in this case, we actually do not need step (3) $\dots$)

To keep things simple, we will henceforth focus on “Stage A”:

Stage A: (Recurring part)	(1) Draw an $n \times k$ random matrix $\mathsf{\Omega}$. (2) Compute a sample matrix $\mathsf{Y} = \mathsf{A}\mathsf{\Omega}$. (3) Compute an ON matrix Q such that $\mathsf{Y} = \mathsf{Q}\mathsf{Q}^{*}\mathsf{Y}$.

Stage A solves what we call it our *primitive problem:*

Primitive problem: Given an $n \times n$ matrix A, and an integer ℓ, find an $n \times \ell$ orthonormal matrix Q_{ℓ} such that $\mathsf{A} \approx \mathsf{Q}_{\ell}\,\mathsf{Q}_{\ell}^{*}\,\mathsf{A}$.

Outline of the tutorial:

1. Techniques for computing the SVD of a matrix of exact rank k.

2. Variations of techniques for matrices of exact rank.
   - Single pass algorithms.
   - How to compute spanning rows and spanning columns (CUR, etc).

3. Techniques for matrices of the form $\mathsf{A} = \mathsf{EF} + \mathsf{N}$ with N small.
   - Error estimation.

4. Techniques for matrices of the form $\mathsf{A} = \mathsf{EF} + \mathsf{N}$ with N large.
   - The “power method.”

5. Random transforms that can be applied rapidly.
   - The “Subsampled Random Fourier Transform” (SRFT) and its cousins.

6. Review / Putting things together / Model problems.

We now consider the case where A takes the form

$$\begin{array}{ccccc} \mathsf{A} & = & \mathsf{E} \qquad \mathsf{F}^* & + & \mathsf{N} \\ n \times n & & n \times k \quad k \times n & & n \times n \\ \text{Matrix to} & & \text{“signal”} & & \text{“noise”} \\ \text{be analyzed} & & & & \end{array}$$

At first, we consider the case where N is “small.”

This situation is common in many areas of scientific computation (*e.g.* in construction of “fast” methods for solving PDEs) but is perhaps not the one typically encountered in data mining applications. This part may be viewed as a transition section.

Assumption: $\mathsf{A} = \mathsf{E}\mathsf{F}^* + \mathsf{N}$

Compute samples: $\mathsf{Y} = \mathsf{E}\,\big(\mathsf{F}^*\Omega\big) + \mathsf{N}\Omega$

We want $\mathrm{Ran}(\mathsf{U}) = \mathrm{Ran}(\mathsf{Y})$.

Problem: The term $\mathsf{N}\Omega$ shifts the range of Y so that $\mathrm{Ran}(\mathsf{U}) \neq \mathrm{Ran}(\mathsf{Y})$.

Observation: We actually do not need equality — we just need $\mathrm{Ran}(\mathsf{U}) \subseteq \mathrm{Ran}(\mathsf{Y})$.

Remedy: Take a few extra samples!

1. Pick a parameter p indicating oversampling. Say $p = 5$.
2. Draw an $n \times (k+p)$ random matrix Ω.
3. Compute an $n \times (k+p)$ sample matrix $\mathsf{Y} = \mathsf{A}\Omega$.
4. Compute an $n \times (k+p)$ ON matrix Q such that $\mathsf{Y} = \mathsf{Q}\mathsf{Q}^*\mathsf{Y}$.
5. Given Q, form a $(k+p)$-term SVD of A as before.
   If desired, truncate the last p terms.

Primitive problem: Given an $n \times n$ matrix A, and an integer ℓ, find an $n \times \ell$ orthonormal matrix Q_ℓ such that $\mathsf{A} \approx \mathsf{Q}_\ell\,\mathsf{Q}_\ell^*\,\mathsf{A}$.

Randomized algorithm — formal description:

1. Construct a random matrix Ω_ℓ of size $n \times \ell$. $\leftarrow \ell = k + p$

   Suppose for now that Ω_ℓ is Gaussian.

2. Form the $n \times \ell$ sample matrix $\mathsf{Y}_\ell = \mathsf{A}\,\Omega_\ell$.

3. Construct an $n \times \ell$ orthonormal matrix Q_ℓ such that $\mathsf{Y}_\ell = \mathsf{Q}_\ell\,\mathsf{Q}_\ell^*\,\mathsf{Y}_\ell$.

   (In other words, the columns of Q_ℓ form an ON basis for $\text{Ran}(\mathsf{Y}_\ell)$.)

Error measure:

The error incurred by the algorithm is $e_\ell = ||\mathsf{A} - \mathsf{Q}_\ell\,\mathsf{Q}_\ell^*\,\mathsf{A}||$.

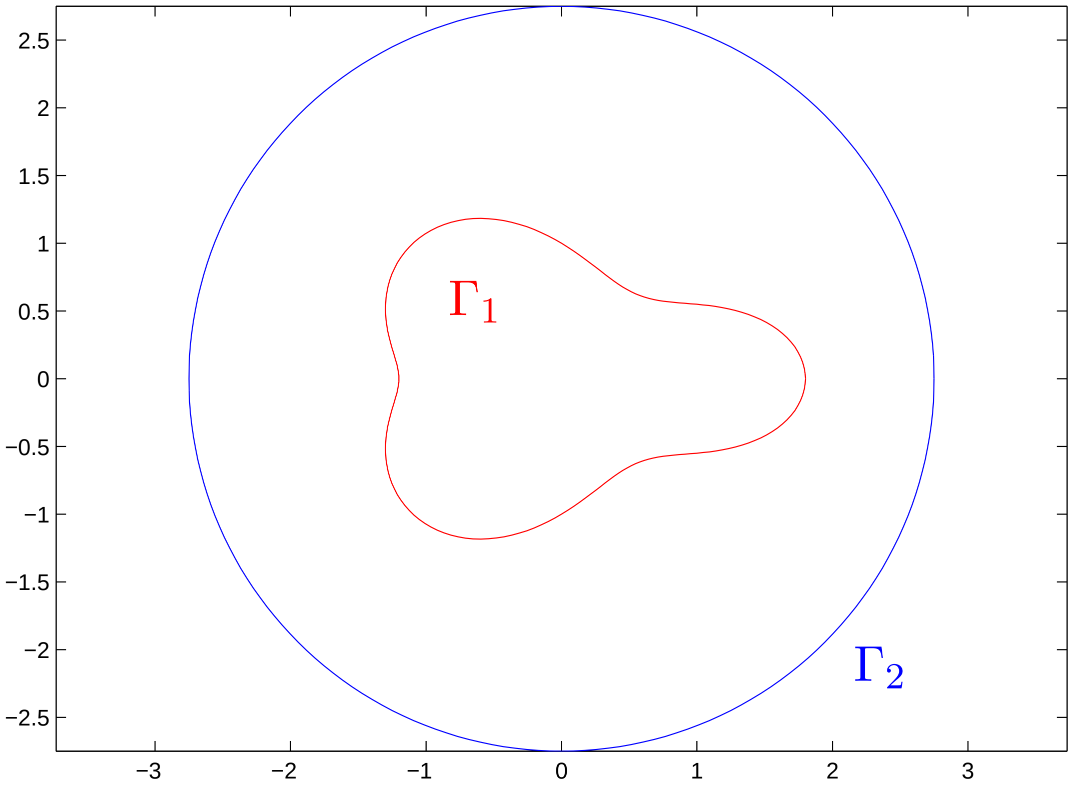
The error e_ℓ is bounded from below by $\sigma_{\ell+1} = \inf\{||\mathsf{A} - \mathsf{B}|| :\ \mathsf{B} \text{ has rank } \ell\}$.

Specific example to illustrate the performance:

Let A be a 200×200 matrix arising from discretization of

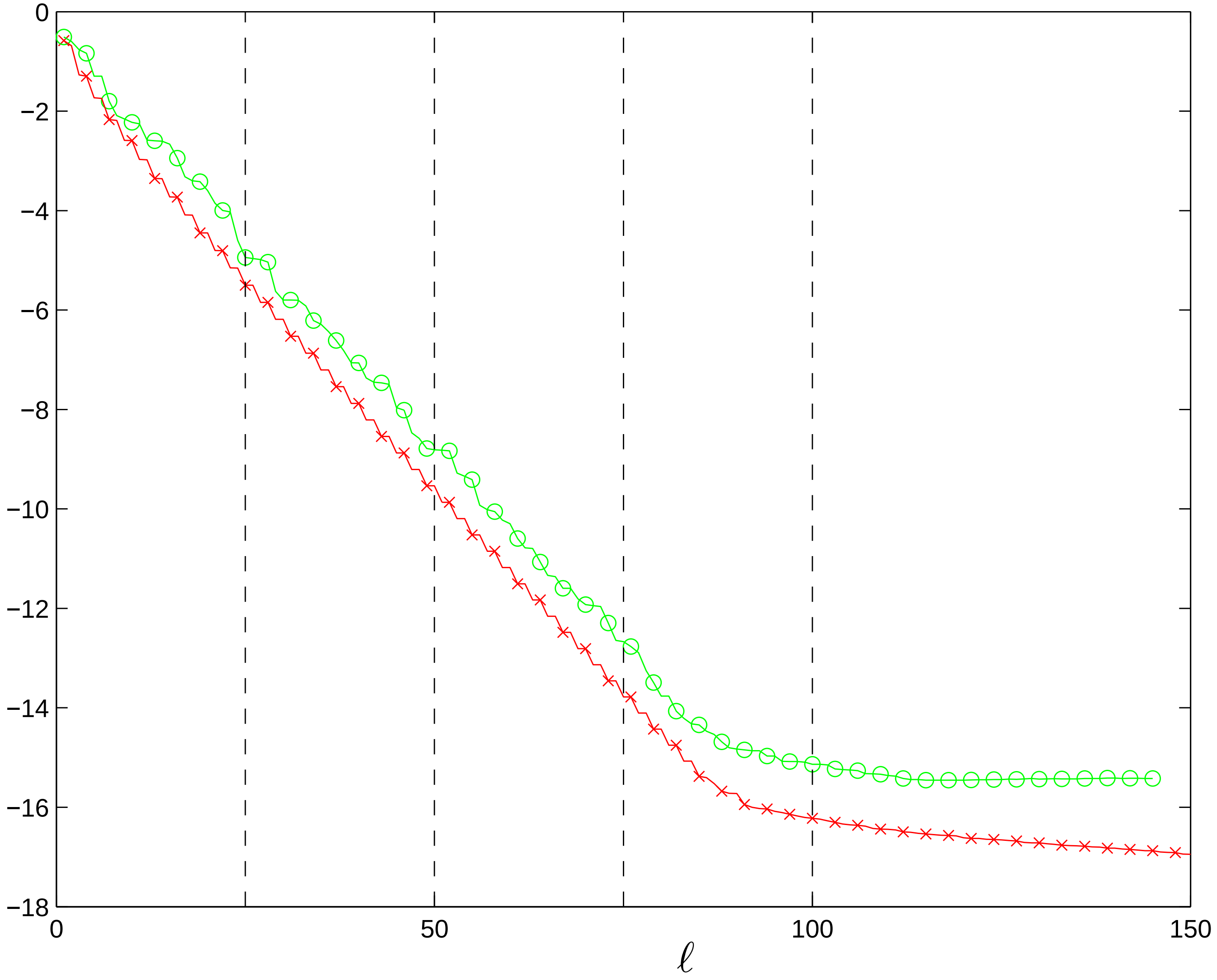
$$[\mathcal{S}_{\Gamma_2 \leftarrow \Gamma_1}\, u](x) = \alpha \int_{\Gamma_1} \log|x-y|\, u(y)\, ds(y), \qquad x \in \Gamma_2,$$

where Γ_1 is shown in red and Γ_2 is shown in blue:



The number α is chosen so that $||\mathsf{A}|| = \sigma_1 = 1$.

Results from one realization of the randomized algorithm



$\log_{10}(e_\ell)$

(actual error)

$\log_{10}(\sigma_{\ell+1})$

(theoretically

minimal error)

Primitive problem: Given an $n \times n$ matrix A, and an integer ℓ, find an $n \times \ell$ orthonormal matrix Q_ℓ such that $\mathsf{A} \approx \mathsf{Q}_\ell\,\mathsf{Q}_\ell^*\,\mathsf{A}$.

1. Construct a random matrix Ω_ℓ of size $n \times \ell$.
   Suppose for now that Ω_ℓ is Gaussian.

2. Form the $n \times \ell$ sample matrix $\mathsf{Y}_\ell = \mathsf{A}\,\Omega_\ell$.

3. Construct an $n \times \ell$ orthonormal matrix Q_ℓ such that $\mathsf{Y}_\ell = \mathsf{Q}_\ell\,\mathsf{Q}_\ell^*\,\mathsf{Y}_\ell$.
   (In other words, the columns of Q_ℓ form an ON basis for $\mathrm{Ran}(\mathsf{Y}_\ell)$.)

Error measure:

The error incurred by the algorithm is $e_\ell = ||\mathsf{A} - \mathsf{Q}_\ell\,\mathsf{Q}_\ell^*\,\mathsf{A}||$.

The error e_ℓ is bounded from below by $\sigma_{\ell+1} = \inf\{||\mathsf{A} - \mathsf{B}|| \,:\, \mathsf{B} \text{ has rank } \ell\}$.

Primitive problem: Given an $n \times n$ matrix A, and an integer ℓ, find an $n \times \ell$ orthonormal matrix Q_ℓ such that $\mathsf{A} \approx \mathsf{Q}_\ell\,\mathsf{Q}_\ell^*\,\mathsf{A}$.

1. Construct a random matrix Ω_ℓ of size $n \times \ell$.
   Suppose for now that Ω_ℓ is Gaussian.

2. Form the $n \times \ell$ sample matrix $\mathsf{Y}_\ell = \mathsf{A}\,\Omega_\ell$.

3. Construct an $n \times \ell$ orthonormal matrix Q_ℓ such that $\mathsf{Y}_\ell = \mathsf{Q}_\ell\,\mathsf{Q}_\ell^*\,\mathsf{Y}_\ell$.
   (In other words, the columns of Q_ℓ form an ON basis for $\mathrm{Ran}(\mathsf{Y}_\ell)$.)

Error measure:

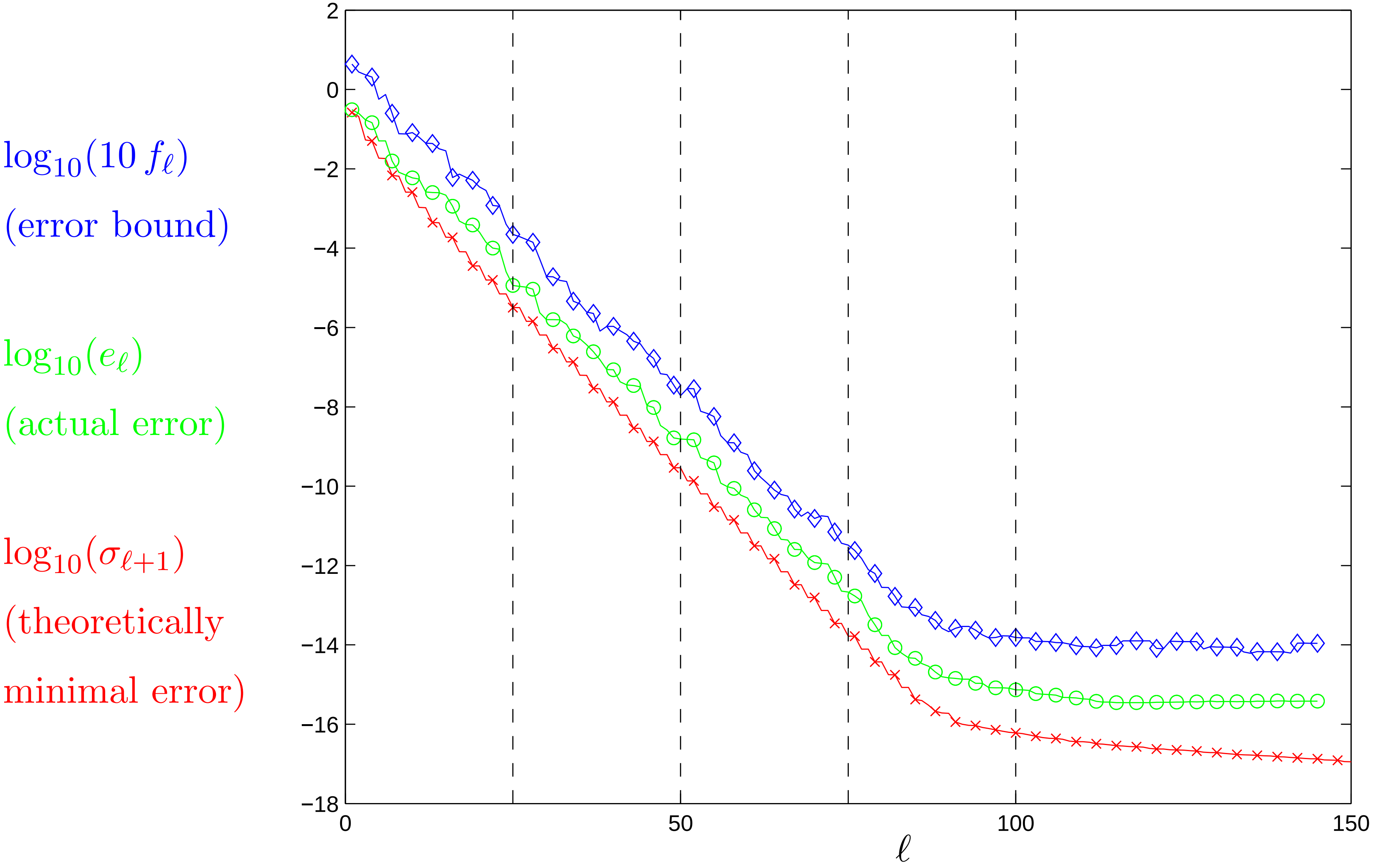
The error incurred by the algorithm is $e_\ell = ||\mathsf{A} - \mathsf{Q}_\ell\,\mathsf{Q}_\ell^*\,\mathsf{A}||$.

The error e_ℓ is bounded from below by $\sigma_{\ell+1} = \inf\{||\mathsf{A} - \mathsf{B}|| : \ \mathsf{B} \text{ has rank } \ell\}$.

Error estimate: $f_\ell = \max_{1 \leq j \leq 10} \big|\big|\big(\mathsf{I} - \mathsf{Q}_\ell\,\mathsf{Q}_\ell^*\big)\,y_{\ell+j}\big|\big|$.

The computation stops when we come to an ℓ such that $f_\ell < \varepsilon \times [\text{constant}]$.

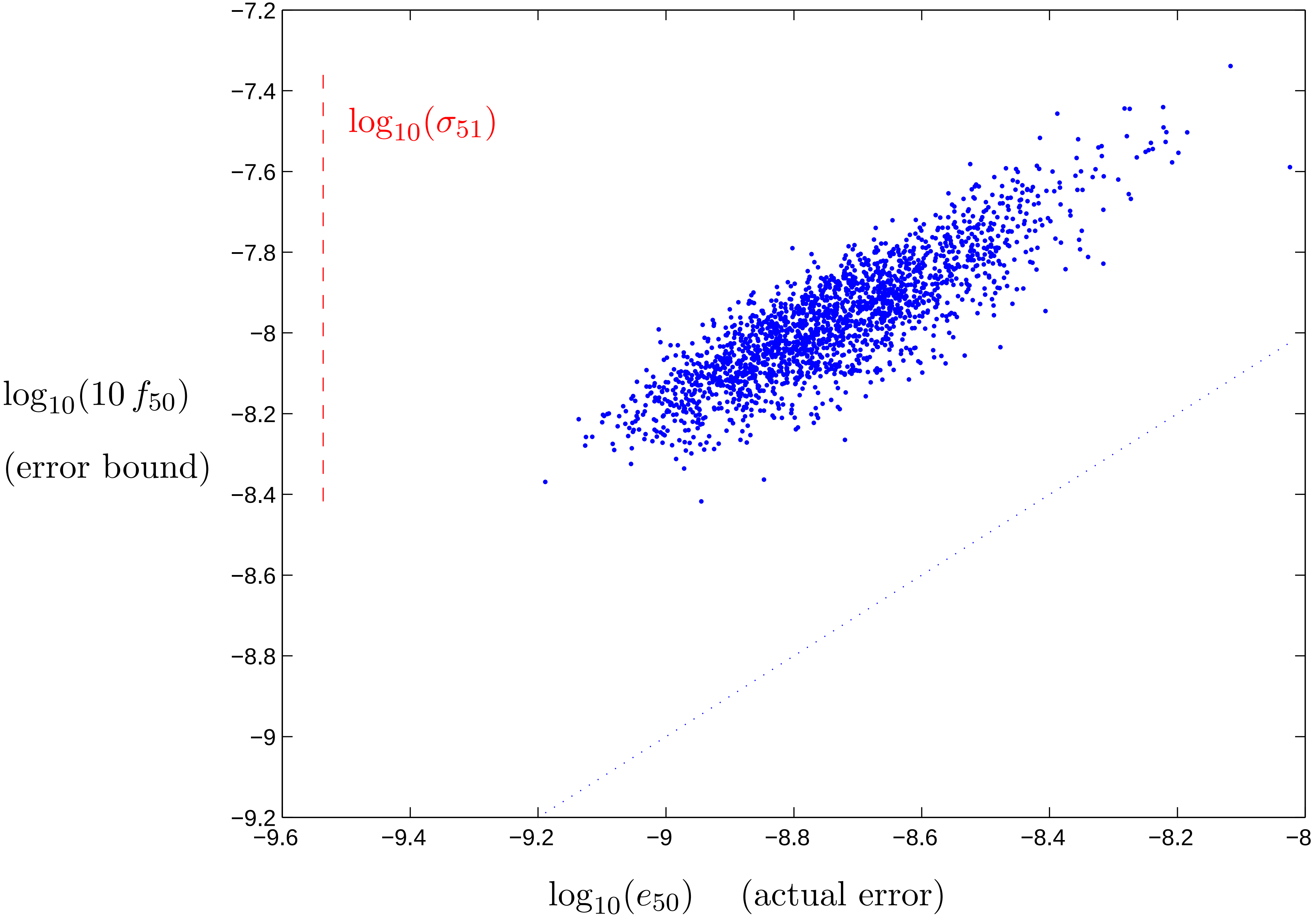
Results from one realization of the randomized algorithm

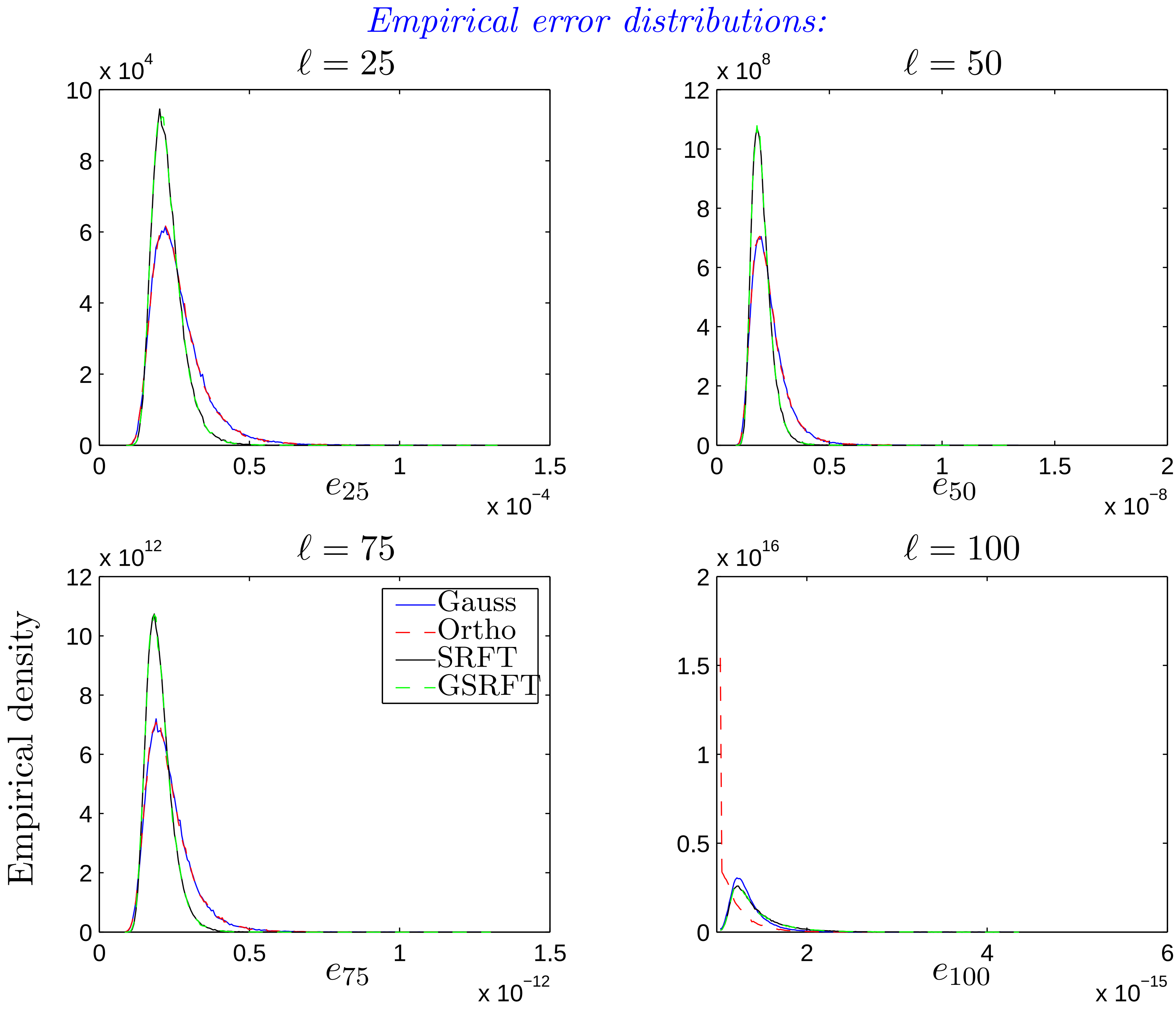


Note: The development of an error estimator resolves the issue of not knowing the numerical rank in advance!

Was this just a lucky realization?

Each dots represents one realization of the experiment with $\ell = 50$ samples:




Empirical error distributions:
$\ell = 25$
x 10^4
10
8
6
4
2
0
0
0.5
1
1.5
e_{25}
x 10^-4
$\ell = 50$
x 10^8
12
10
8
6
4
2
0
0
0.5
1
1.5
2
e_{50}
x 10^-8
$\ell = 75$
x 10^12
12
10
8
6
4
2
0
0
0.5
1
1.5
e_{75}
x 10^-12
Gauss
Ortho
SRFT
GSRFT
Empirical density
$\ell = 100$
x 10^16
2
1.5
1
0.5
0
2
4
6
e_{100}
x 10^-15

Important:

- What is stochastic in practice is the *run time*, not the accuracy.
- The error in the factorization is (practically speaking) always within the prescribed tolerance.
- Post-processing (practically speaking) always determines the rank correctly.

Results from a high-frequency Helmholtz problem (complex arithmetic)

$\log_{10}(10\, f_\ell)$

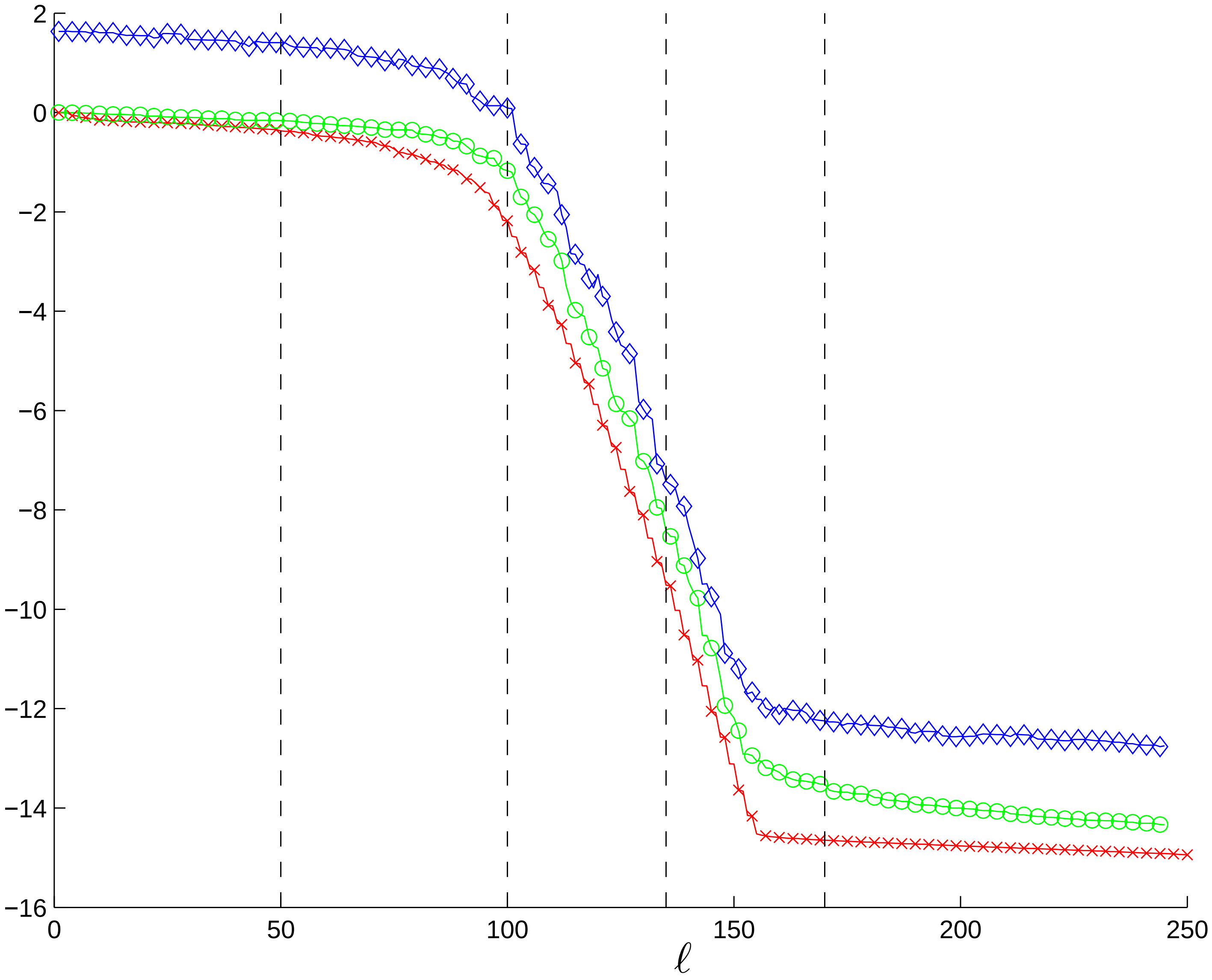
(error bound)

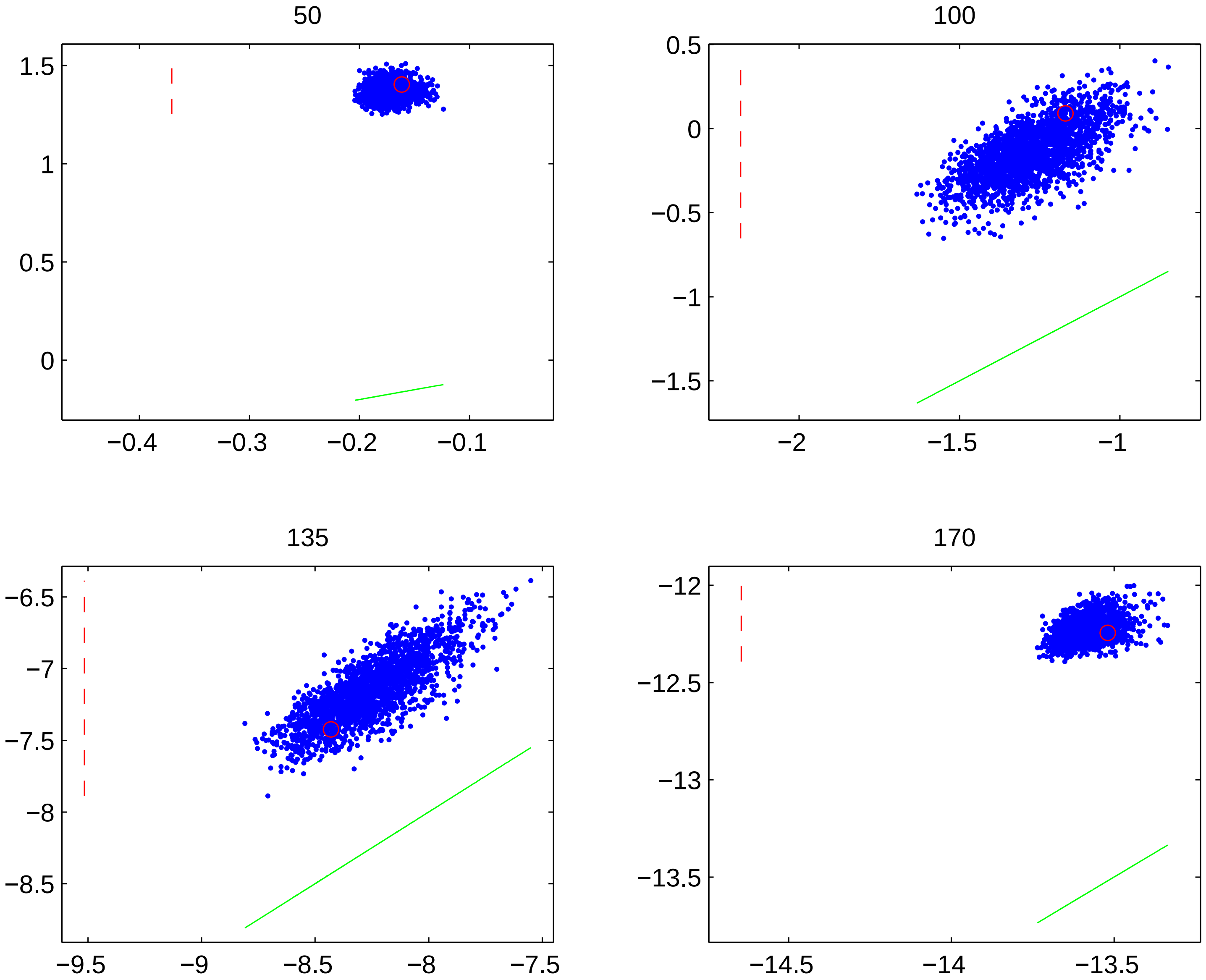
$\log_{10}(e_\ell)$

(actual error)

$\log_{10}(\sigma_{\ell+1})$

(theoretically minimal error)



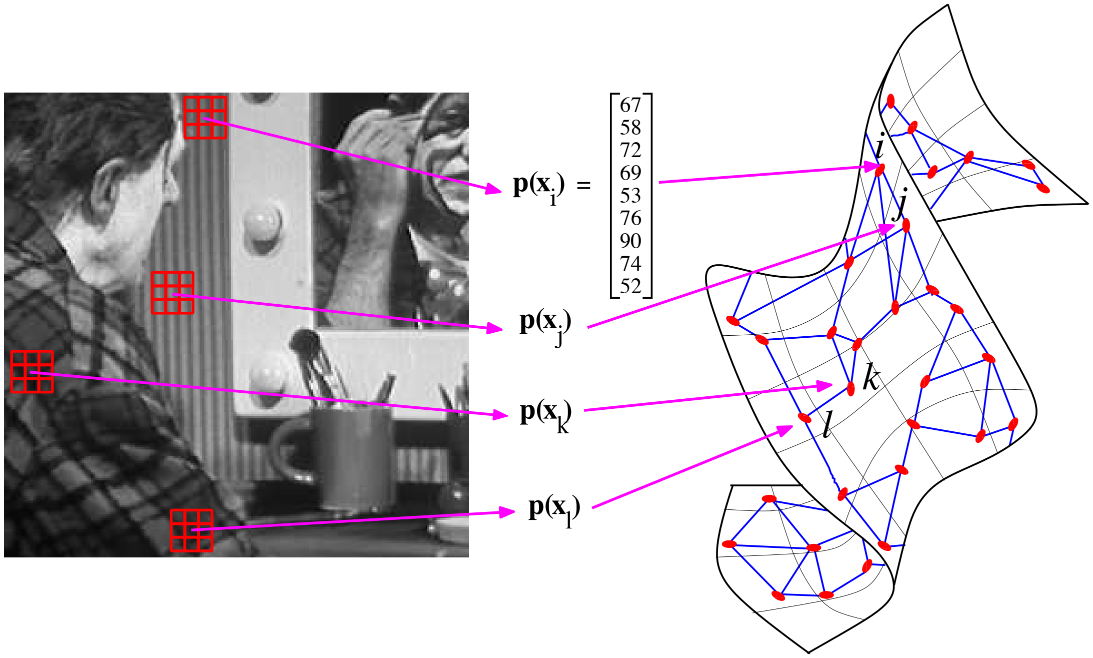

50
1.5
1
0.5
0
−0.4
−0.3
−0.2
−0.1
100
0.5
0
−0.5
−1
−1.5
−2
−1.5
−1
135
−6.5
−7
−7.5
−8
−8.5
−9.5
−9
−8.5
−8
−7.5
170
−12
−12.5
−13
−13.5
−14.5
−14
−13.5

Let us apply the method to problems whose singular values do *not* decay rapidly.

Example 1:

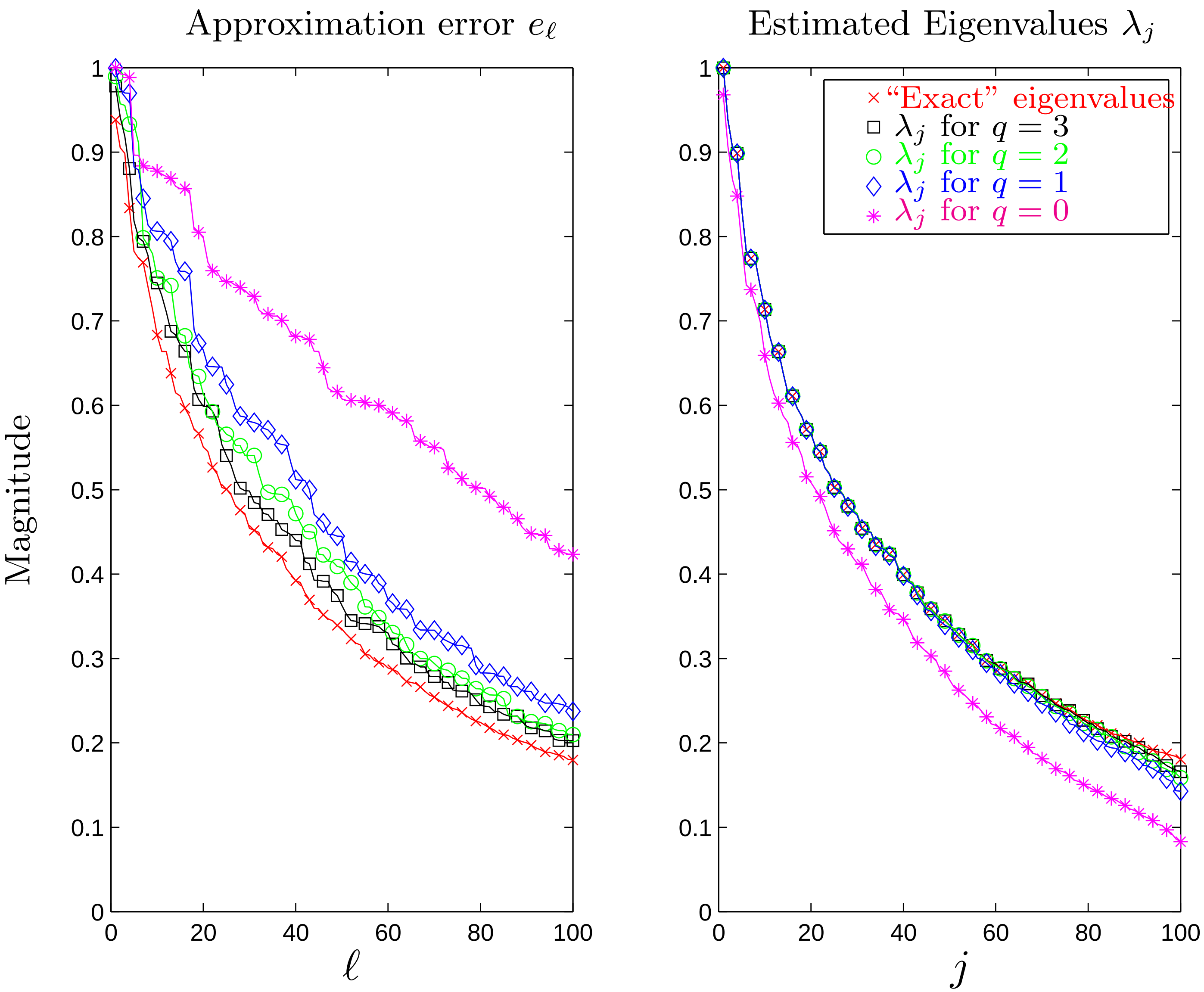
The matrix A being analyzed is a 9025 × 9025 matrix arising in a diffusion geometry approach to image processing.

To be precise, A is a graph Laplacian on the manifold of 9 × 9 patches.



$p(x_i) = \begin{bmatrix} 67 \\ 58 \\ 72 \\ 69 \\ 53 \\ 76 \\ 90 \\ 74 \\ 52 \end{bmatrix}$
$p(x_j)$
$p(x_k)$
$p(x_l)$
i
j
k
l

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.

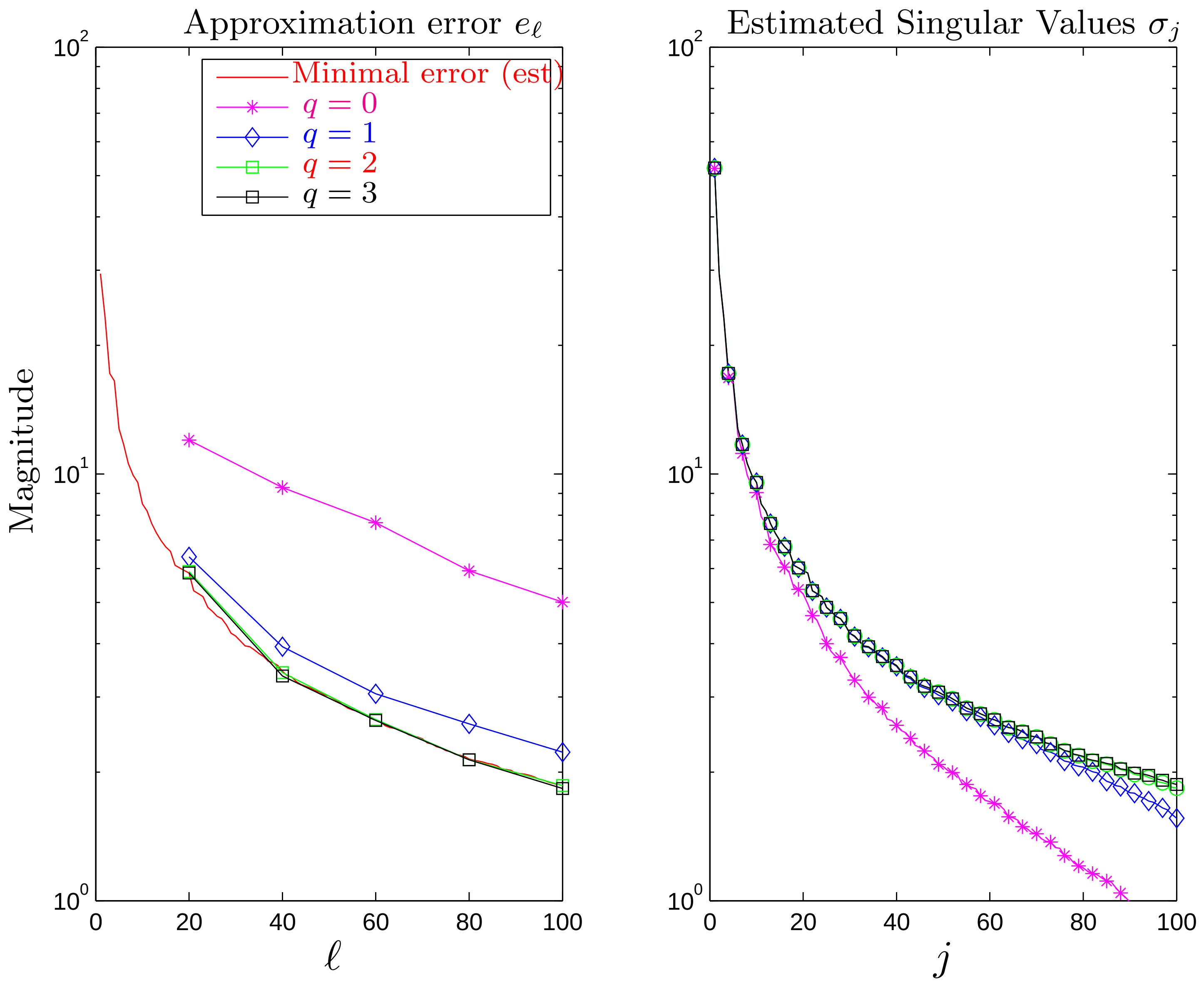
Example 2: “Eigenfaces”

We next process 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.

It turns out that the errors can be explained in detail by analysis.

The framework in the following theorems is:

We are given an $n \times n$ matrix A and seek a rank-k approximation

$$\begin{array}{ccccc} \mathsf{A} & \approx & \mathsf{B} & & \mathsf{C} \\ n \times n & & n \times k & & k \times n \end{array}$$

Fix a small integer p representing how much we "over-sample." Set $\ell = k + p$.

1. Construct a Gaussian random matrix Ω_ℓ of size $n \times \ell$.

2. Form the $n \times \ell$ matrix $\mathsf{Y}_\ell = \mathsf{A}\,\Omega_\ell$.

3. Construct an $n \times \ell$ orthogonal matrix Q_ℓ such that $\mathsf{Y}_\ell = \mathsf{Q}_\ell\,\mathsf{Q}_\ell^*\,\mathsf{Y}_\ell$.

Question: How does the error $||\mathsf{A} - \mathsf{Q}_\ell\,\mathsf{Q}_\ell^*\,\mathsf{A}||$ compare to σ_{k+1}?

Our set-up is

$$\mathsf{A} = \underbrace{\mathsf{U}_1\Sigma_1\mathsf{V}_1^*}_{\text{“Signal”}} + \underbrace{\mathsf{U}_2\Sigma_2\mathsf{V}_2^*}_{\text{“Noise”}}.$$

The sample matrix is then

$$\mathsf{Y} = \mathsf{U}_1\big(\Sigma_1\mathsf{V}_1^*\Omega\big) + \mathsf{U}_2\big(\Sigma_2\mathsf{V}_2^*\Omega\big).$$

Let P_Y denote the orthogonal projection onto the range of Y.
Our “restricted” matrix is

$$\mathsf{P}_\mathsf{Y}\mathsf{A} = \big(\mathsf{P}_\mathsf{Y}\mathsf{U}_1\big)\Sigma_1\mathsf{V}_1^* + \underbrace{\big(\mathsf{P}_\mathsf{Y}\mathsf{U}_2\big)\Sigma_2\mathsf{V}_2^*}_{\text{Small regardless of } \mathsf{P}_\mathsf{Y}}.$$

We find that

$$||\mathsf{A} - \mathsf{P}_\mathsf{Y}\mathsf{A}|| \approx ||\mathsf{U}_1\Sigma_1\mathsf{V}_1^* - \mathsf{P}_\mathsf{Y}\mathsf{U}_1\Sigma_1\mathsf{V}_1^*|| = ||\mathsf{U}_1\Sigma_1 - \mathsf{P}_\mathsf{Y}\mathsf{U}_1\Sigma_1||.$$

Extreme “bad” case: A row of $\mathsf{V}_1^*\,\Omega$ is very small.

More realistic “bad” case: $\mathsf{V}_1^*\Omega$ has a small singular value.

Theorem: *[Halko, Martinsson, Tropp 2009] Fix a real $n \times n$ matrix A with singular values $\sigma_1, \sigma_2, \sigma_3, \dots$. Choose integers $k \geq 1$ and $p \geq 2$, and draw an $n \times (k+p)$ standard Gaussian random matrix Ω. Construct the sample matrix $\mathsf{Y} = \mathsf{A}\Omega$, and let Q denote an orthonormal matrix such that Ran(Q) = Ran(Y). Then*

$$\mathbb{E}||\mathsf{A} - \mathsf{Q}\mathsf{Q}^*\mathsf{A}||_{\mathrm{F}} \leq \left(1 + \frac{k}{p-1}\right)^{1/2} \left(\sum_{j=k+1}^{n} \sigma_j^2\right)^{1/2}.$$

Moreover,

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

(Numerical experiments indicate that these estimates are close to sharp.)

When the singular values decay rapidly, the output of the randomized algorithm is close to optimal. Say for instance that $\sigma_j \sim \beta^j$ for $\beta \in (0,1)$. Then

$$\mathbb{E}||\mathsf{A} - \mathsf{Q}\mathsf{Q}^*\mathsf{A}|| \sim \left(\sum_{j=k+1}^{n} \sigma_j^2\right)^{1/2} \sim \sigma_{k+1}\left(\sum_{j=0}^{n} \beta^2\right)^{1/2} \sim \sigma_{k+1}\frac{1}{\sqrt{1-\beta^2}},$$

Theorem: *[Halko, Martinsson, Tropp 2009] Fix a real $n \times n$ matrix A with singular values $\sigma_1, \sigma_2, \sigma_3, \ldots$. Choose integers $k \geq 1$ and $p \geq 2$, and draw an $n \times (k+p)$ standard Gaussian random matrix Ω. Construct the sample matrix $\mathsf{Y} = \mathsf{A}\Omega$, and let Q denote an orthonormal matrix such that Ran(Q) = Ran(Y). Then*

$$\mathbb{E}||\mathsf{A} - \mathsf{QQ}^*\mathsf{A}||_{\mathrm{F}} \leq \left(1 + \frac{k}{p-1}\right)^{1/2} \left(\sum_{j=k+1}^{n} \sigma_j^2\right)^{1/2}.$$

Moreover,

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

On the other hand, if the singular values stay do not decay beyond σ_{k+1}, then

$$\mathbb{E}||\mathsf{A} - \mathsf{QQ}^*\mathsf{A}|| \sim \left(\sum_{j=k+1}^{n} \sigma_j^2\right)^{1/2} \sim \sqrt{n-k}\,\sigma_{k+1}.$$

If n is very large, then the factor $\sqrt{n-k}$ spells doom.

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

Let A_k denote the best possible rank k approximation to A:

$$\mathsf{A}_k = \sum_{j=1}^{k} \sigma_j \boldsymbol{u}_j \boldsymbol{v}_j^{*}.$$

Then

$$\mathsf{A} = \mathsf{A}_k + \mathsf{N},$$

where N is the residual. We now observe that

$$||\mathsf{N}|| = \sigma_{k+1}, \qquad \text{and} \qquad ||\mathsf{N}||_{\mathrm{F}} = \left(\sum_{j=k+1}^{n} \sigma_j^2\right)^{1/2}.$$

Observation: The error measured in the *spectral norm* depends on the residual measured in the *Frobenius norm*. This is bad news, since for a large matrix

$$||\mathsf{N}|| = \sigma_{k+1} \ll \left(\sum_{j=k+1}^{n} \sigma_j^2\right)^{1/2} = ||\mathsf{N}||_{\mathrm{F}}.$$

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

Let A_k denote the best possible rank k approximation to A:

$$\mathsf{A}_k = \sum_{j=1}^{k}\sigma_j\boldsymbol{u}_j\boldsymbol{v}_j^*.$$

Then

$$\mathsf{A} = \mathsf{A}_k + \mathsf{N},$$

where N is the residual. We now observe that

$$||\mathsf{N}|| = \sigma_{k+1}, \qquad \text{and} \qquad ||\mathsf{N}||_{\mathrm{F}} = \left(\sum_{j=k+1}^{n}\sigma_j^2\right)^{1/2}.$$

Observation: One definition of the "signal-to-noise ratio":

$$\frac{\text{signal}}{\text{noise}} = \frac{||\mathsf{A}_k||_{\mathrm{F}}}{||\mathsf{N}||_{\mathrm{F}}} = \left(\frac{\sum_{j=1}^{k}\sigma_j^2}{\sum_{j=k+1}^{n}\sigma_j^2}\right)^{1/2}.$$

What about bounds on tail probabilities?

What about bounds on tail probabilities? Due to overwhelmingly strong concentration of measure effects, these are often in a practical sense irrelevant.

What about bounds on tail probabilities? Due to overwhelmingly strong concentration of measure effects, these are often in a practical sense irrelevant.

However,

Theorem: *[Halko, Martinsson, Tropp 2009] Fix a* real *$m \times n$ matrix A with singular values $\sigma_1, \sigma_2, \sigma_3, \dots$. Choose integers $k \geq 1$ and $p \geq 4$, and draw an $n \times (k+p)$ standard Gaussian random matrix Ω. Construct the sample matrix $\mathsf{Y} = \mathsf{A}\,\Omega$, and let Q denote an orthonormal matrix such that Ran$(\mathsf{Q}) =$ Ran(Y). For all $u, t \geq 1$,*

$$||\mathsf{A} - \mathsf{Q}\mathsf{Q}^{*}\mathsf{A}|| \leq \left(1 + t\,\sqrt{12\,k/p} + u\,t\,\frac{e\,\sqrt{k+p}}{p+1}\right)\,\sigma_{k+1} + \frac{t\,e\,\sqrt{k+p}}{p+1}\,\left(\sum_{j>k}\sigma_j^2\right)^{1/2}$$

except with probability at most $5\,t^{-p} + 2\,e^{-u^2/2}$.

The theorem can be simplified by by choosing t and u appropriately. For instance,

$$||\mathsf{A} - \mathsf{Q}\mathsf{Q}^{*}\mathsf{A}|| \leq \left(1 + 8\sqrt{(k+p)\cdot p\log p}\right)\,\sigma_{k+1} + 3\sqrt{k+p}\,\left(\sum\nolimits_{j>k}\sigma_j^2\right)^{1/2},$$

except with probability at most $6\,p^{-p}$.

Outline of the tutorial:

1. Techniques for computing the SVD of a matrix of exact rank k.

2. Variations of techniques for matrices of exact rank.
   - Single pass algorithms.
   - How to compute spanning rows and spanning columns (CUR, etc).

3. Techniques for matrices of the form $\mathsf{A} = \mathsf{EF} + \mathsf{N}$ with N small.
   - Error estimation.

4. Techniques for matrices of the form $\mathsf{A} = \mathsf{EF} + \mathsf{N}$ with N large.
   - The "power method."

5. Random transforms that can be applied rapidly.
   - The "Subsampled Random Fourier Transform" (SRFT) and its cousins.

6. Review / Putting things together / Model problems.

We are faced with a problem:

- The randomized algorithm performs poorly for large matrices whose singular values decay slowly.
- The matrices of interest to us are large and have slowly decaying singular values.

What to do?

Idea [Rokhlin, Szlam, Tygert 2008]:
Apply the algorithm to the auxiliary matrix

$$\mathsf{B} = (\mathsf{A}\,\mathsf{A}^*)^q\,\mathsf{A}.$$

The matrices A and B have the same left singular vectors, and the singular values of B decay much faster. In fact:

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

So use the sample matrix

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

instead of

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

Algorithm for computing a rank-k SVD of a given matrix A:

(1) Pick a parameter p and set $\ell = k + p$. Draw an $n \times \ell$ random matrix Ω.

(2) Compute a sample matrix $\mathsf{Y} = \big(\mathsf{A}\mathsf{A}^{*}\big)^{q}\mathsf{A}\Omega$.
Note: The product is typically evaluated via alternating applications of A *and* A^{*}*.*

(3) Compute an orthonormal matrix Q such that $\mathsf{Y} = \mathsf{Q}\mathsf{Q}^{*}\mathsf{Y}$.
Then with high probability, $\mathsf{A} \approx \mathsf{Q}\mathsf{Q}^{*}\mathsf{A}$.

(4) Form $\mathsf{B} = \mathsf{Q}^{*}\,\mathsf{A}$.

(5) Factorize $\mathsf{B} = \hat{\mathsf{U}}\Sigma\mathsf{V}^{*}$.

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

(7) If desired, truncate the ℓ-term SVD $\mathsf{A} = \mathsf{U}\Sigma\mathsf{V}^{*}$ to its leading k terms.

Output: A factorization $\mathsf{A} \approx \mathsf{U}\Sigma\mathsf{V}^{*}$.

Power method for improving accuracy:

Theorem: *[Halko, Martinsson, Tropp 2009] Let m, n, and ℓ be positive integers such that $\ell < n \leq m$. Let* A *be an* $m \times n$ *matrix and let* Ω *be an* $n \times \ell$ *matrix. Let* q *be a non-negative integer, set* $\mathsf{B} = (\mathsf{A}\mathsf{A}^*)^q\mathsf{A}$*, and construct the sample matrix* $\mathsf{Z} = \mathsf{B}\,\Omega$*. Let* P_Z *denote the orthogonal projector onto the range of* Z*. Then*

$$||(\mathsf{I} - \mathsf{P}_\mathsf{Z})\,\mathsf{A}|| \leq ||(I - \mathsf{P}_\mathsf{Z})\,\mathsf{B}||^{1/(2q+1)}.$$

Since the ℓ'th singular value of $\mathsf{B} = (\mathsf{A}\mathsf{A}^*)^q\mathsf{A}$ is $\sigma_\ell^{2\,q+1}$, any result of the type

$$||(\mathsf{I} - \mathsf{P}_\mathsf{Y})\,\mathsf{A}|| \leq C\,\sigma_{k+1},$$

where $\mathsf{Y} = \mathsf{A}\,\Omega$ and $C = C(m, n, k)$, gets improved to a result

$$||(\mathsf{I} - \mathsf{P}_\mathsf{Z})\,\mathsf{A}|| \leq C^{1/(2\,q+1)}\,\sigma_{k+1}$$

when $\mathsf{Z} = (\mathsf{A}\mathsf{A}^*)^q\mathsf{A}\,\Omega$.

We note that the new 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

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

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

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

"Power" method: Restrict A to the linear space

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

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

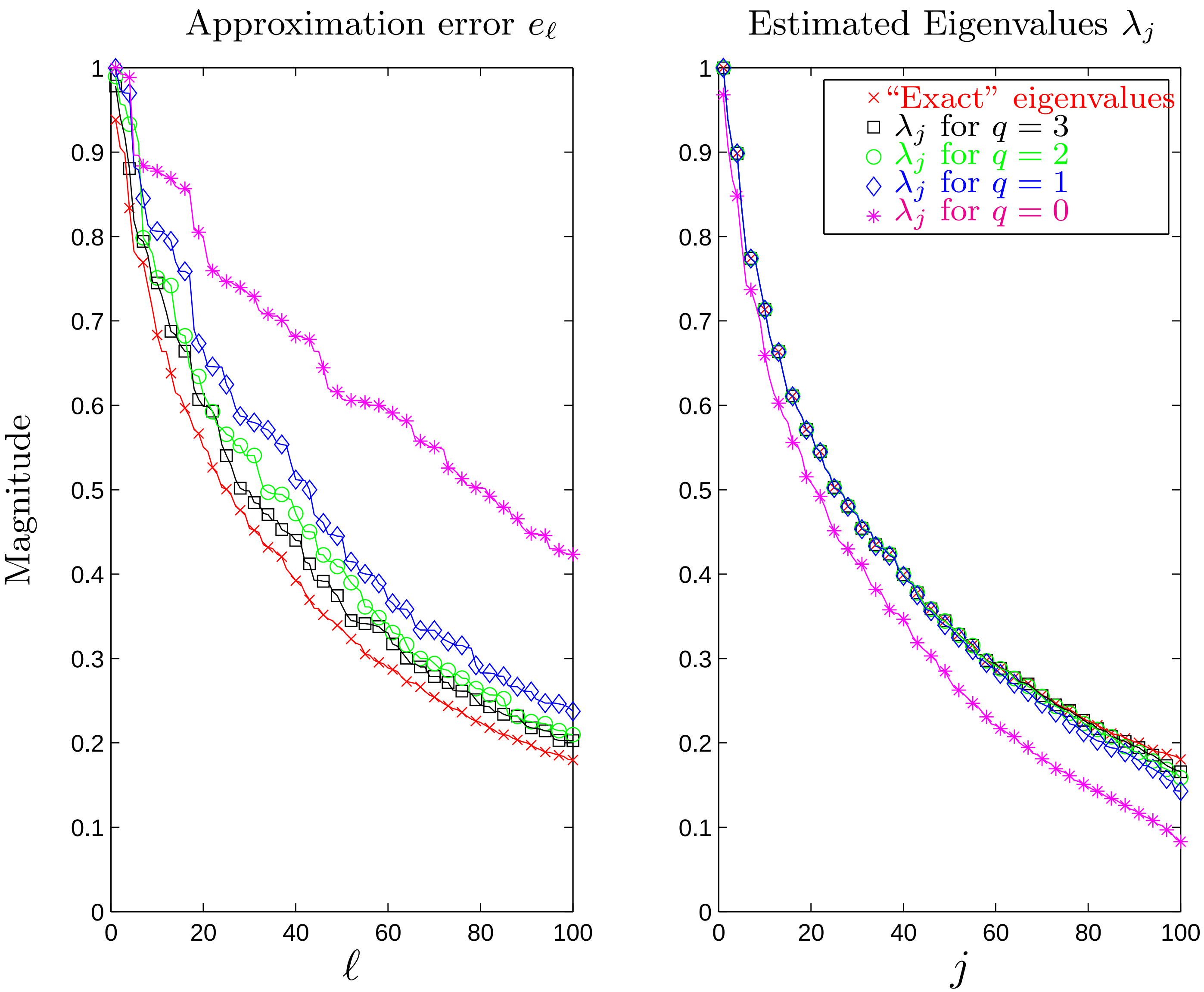
$$\mathcal{V}_q(\boldsymbol{\omega}_1) \times \mathcal{V}_q(\boldsymbol{\omega}_2) \times \cdots \times \mathcal{V}_q(\boldsymbol{\omega}_\ell).$$

This could be a promising area for further work.

Example 1:

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 9×9 patches.



The parameter q is the "power" in the "power method".

Note the speed with which accuracy improves!

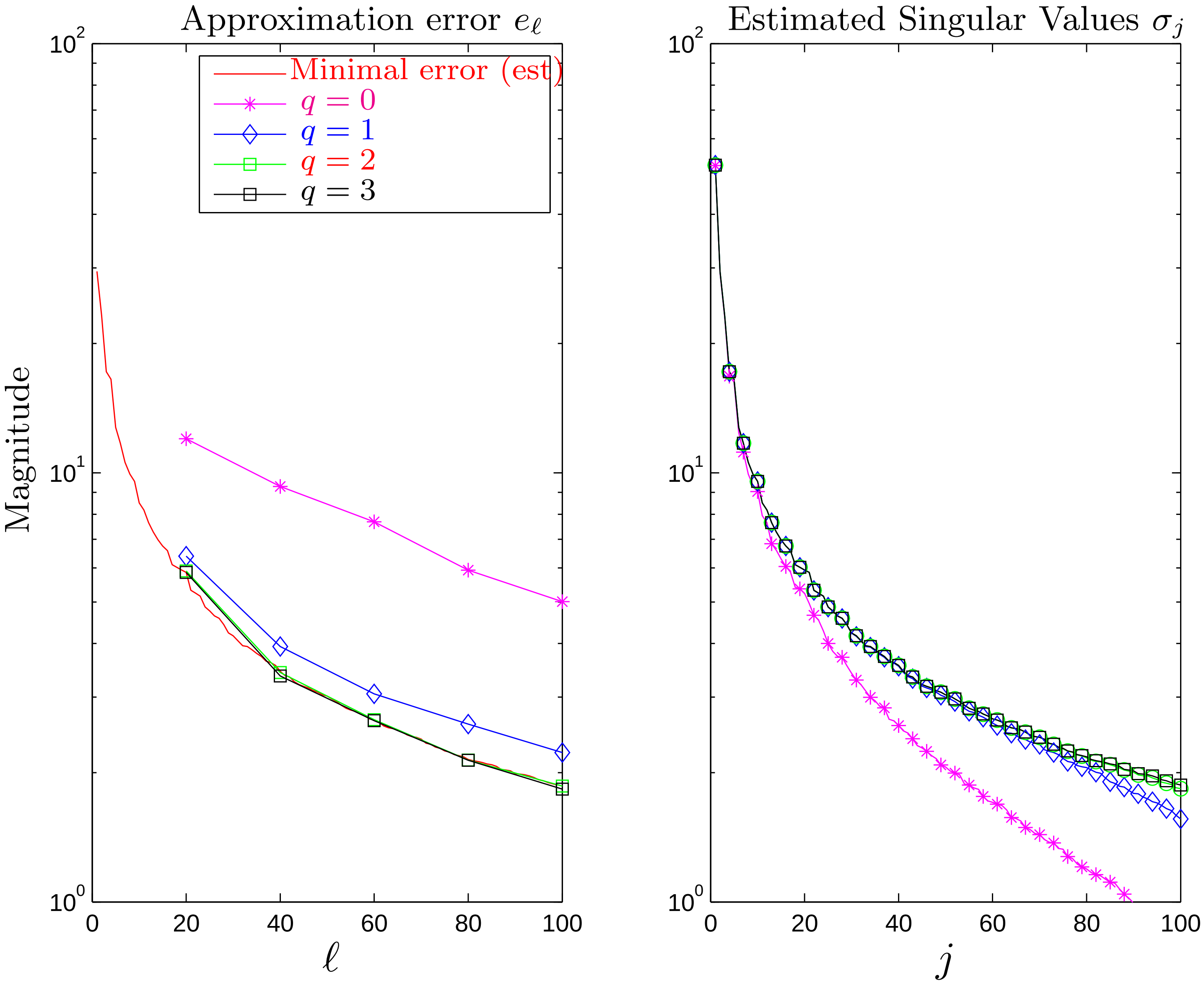
Example 2: “Eigenfaces”

We next process 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 parameter q is the "power" in the "power method".

Note the speed with which accuracy improves!

Example 3 — entirely artificial [Yoel Shkolnisky and Mark Tygert]:

An $m \times n$ matrix was synthesized so that its singular values were

$$\sigma_j = \begin{cases} 1.00, & j = 1, 2, 3, \\ 0.67, & j = 4, 5, 6, \\ 0.34, & j = 7, 8, 9, \\ 0.01, & j = 10, 11, 12, \\ 0.01\frac{n-j}{n=13}, & j = 13, 14, 15, \dots \end{cases}$$

The accuracy enhanced algorithm was implemented with $q = 3$ in Matlab.

Computer: Laptop, single-core 32-bit 2GHz Pentium M, 1.5GB of RAM.

Storage: External hard drive connected via USB 2.0.

A matrix of size $500\,000 \times 80\,000$ (stored at single precision using 160GB) was processed in 18 hours. A rank-12 approximation was computed. The accuracy was estimated to be 0.01 ± 0.001.

Outline of the tutorial:

1. Techniques for computing the SVD of a matrix of exact rank k.

2. Variations of techniques for matrices of exact rank.
   - Single pass algorithms.
   - How to compute spanning rows and spanning columns (CUR, etc).

3. Techniques for matrices of the form $\mathsf{A} = \mathsf{E}\mathsf{F} + \mathsf{N}$ with N small.
   - Error estimation.

4. Techniques for matrices of the form $\mathsf{A} = \mathsf{E}\mathsf{F} + \mathsf{N}$ with N large.
   - The "power method."

5. Random transforms that can be applied rapidly.
   - The "Subsampled Random Fourier Transform" (SRFT) and its cousins.

6. Review / Putting things together / Model problems.

We have an outstanding promise to deliver on.

Early in the lecture, we claimed that randomized sampling can improve the performance even in the situation where a matrix A fits in RAM.

For a normal desktop, the matrix sizes we have in mind are, say:

$n = 2\,000$ and $k \in [20,\ 500]$

or

$n = 4\,000$ and $k \in [20,\ 1\,000]$

Can you beat classical algorithms here?

Yes, factor 2 to 7 improvement in speed if some loss of accuracy is tolerated (say 12 correct digits instead of 15).

Let us revisit the basic randomized scheme:

Primitive problem: Given an $n \times n$ matrix A, and an integer ℓ, find an $n \times \ell$ orthonormal matrix Q_ℓ such that $\mathsf{A} \approx \mathsf{Q}_\ell\,\mathsf{Q}_\ell^*\,\mathsf{A}$.

Pick a parameter of over-sampling p. (Say $p = 5$.) Set $\ell = k + p$.

1. Construct a Gaussian random matrix Ω_ℓ of size $n \times \ell$.

2. Form the $n \times \ell$ sample matrix $\mathsf{Y}_\ell = \mathsf{A}\,\Omega_\ell$.
   Cost is $O(\ell\, n^2)$.

3. Construct an $n \times \ell$ orthonormal matrix Q_ℓ such that $\mathsf{Y}_\ell = \mathsf{Q}_\ell\,\mathsf{Q}_\ell^*\,\mathsf{Y}_\ell$.
   Cost is $O(\ell^2\, n)$.

The asymptotic scaling is the same as rank-revealing QR!

Let us change the random matrix Ω [Liberty,Rokhlin,Tygert,Woolfe 2006]:

1. Construct an "SRFT" random matrix Ω_ℓ of size $n \times \ell$.
   An "SRFT" admits evaluation of $\boldsymbol{x} \mapsto \boldsymbol{x}\,\Omega$ in $O(n\,\log(\ell))$ operations.

2. Form the $n \times \ell$ sample matrix $\mathsf{Y}_\ell = \mathsf{A}\,\Omega_\ell$.
   Cost is $O(n^2\,\log(\ell))$.

3. Construct an $n \times \ell$ orthonormal matrix Q_ℓ such that $\mathsf{Y}_\ell = \mathsf{Q}_\ell\,\mathsf{Q}_\ell^*\,\mathsf{Y}_\ell$.
   Cost is $O(n\,\ell^2)$.

Now the total cost is $O(n^2\,\log(\ell))$.

We have $\ell \sim k$ so the cost is in fact $O(n^2\,\log(k))$!

What is an “SRFT”?

A Subsampled Random Fourier Transform. A random matrix with structure.

One possible choice:

$$\begin{array}{ccccc} \Omega & = & \mathsf{D} & \mathsf{F} & \mathsf{S} \\ n \times \ell & & n \times n & n \times n & n \times \ell \end{array}$$

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, $\mathsf{F}_{jk} = \dfrac{1}{n^{1/2}}\, e^{-2\pi i(j-1)(k-1)/n}$.

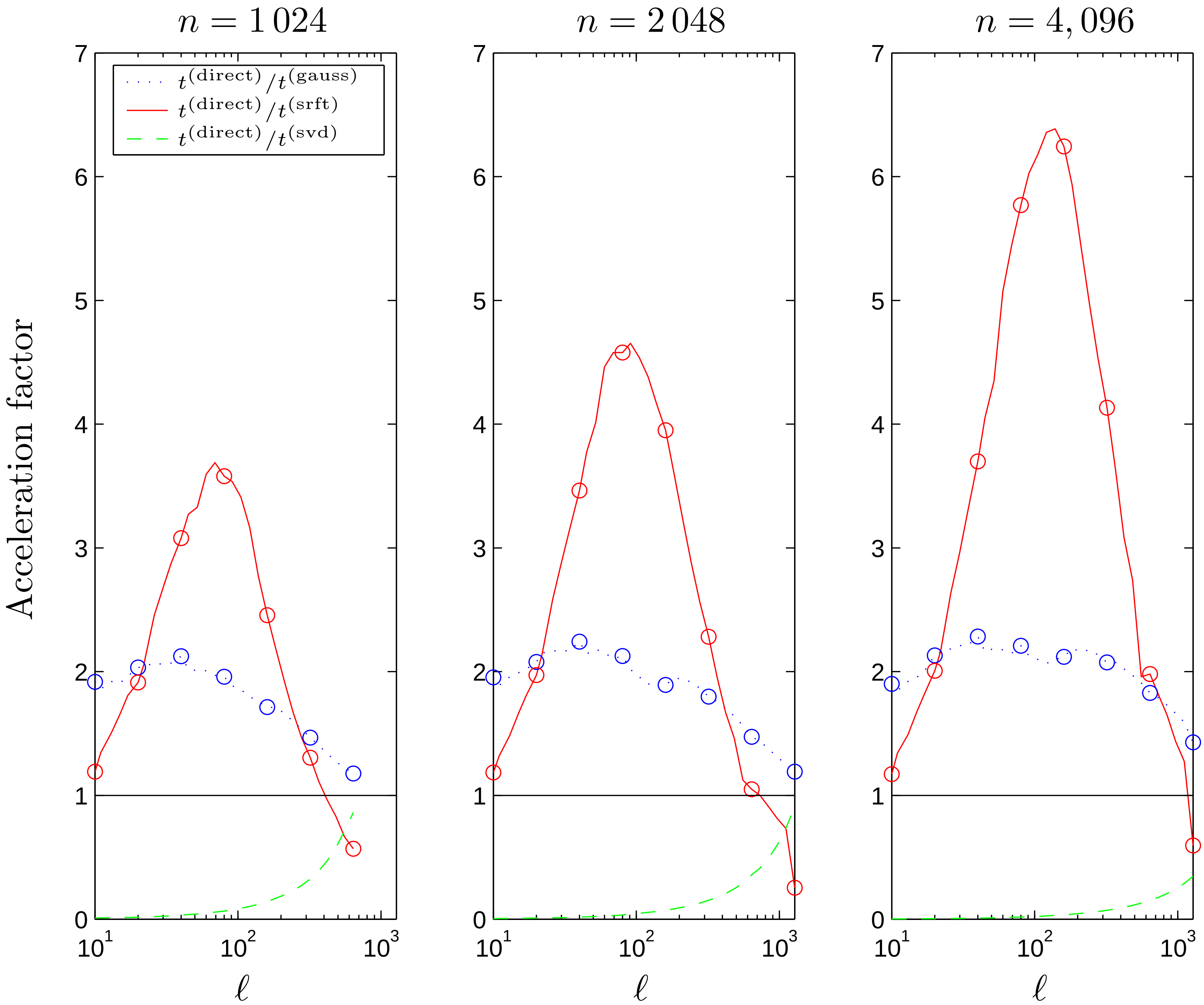
- S is a matrix whose entries are all zeros except for a single, randomly placed 1 in each column. (In other words, the action of S is to draw ℓ columns at random from $\mathsf{D}\,\mathsf{F}$.)

The next graph illustrates the relative speed of three methods:

- *The "direct" method:*
  Standard QR followed by SVD on the reduced matrix ("Golub-Businger").

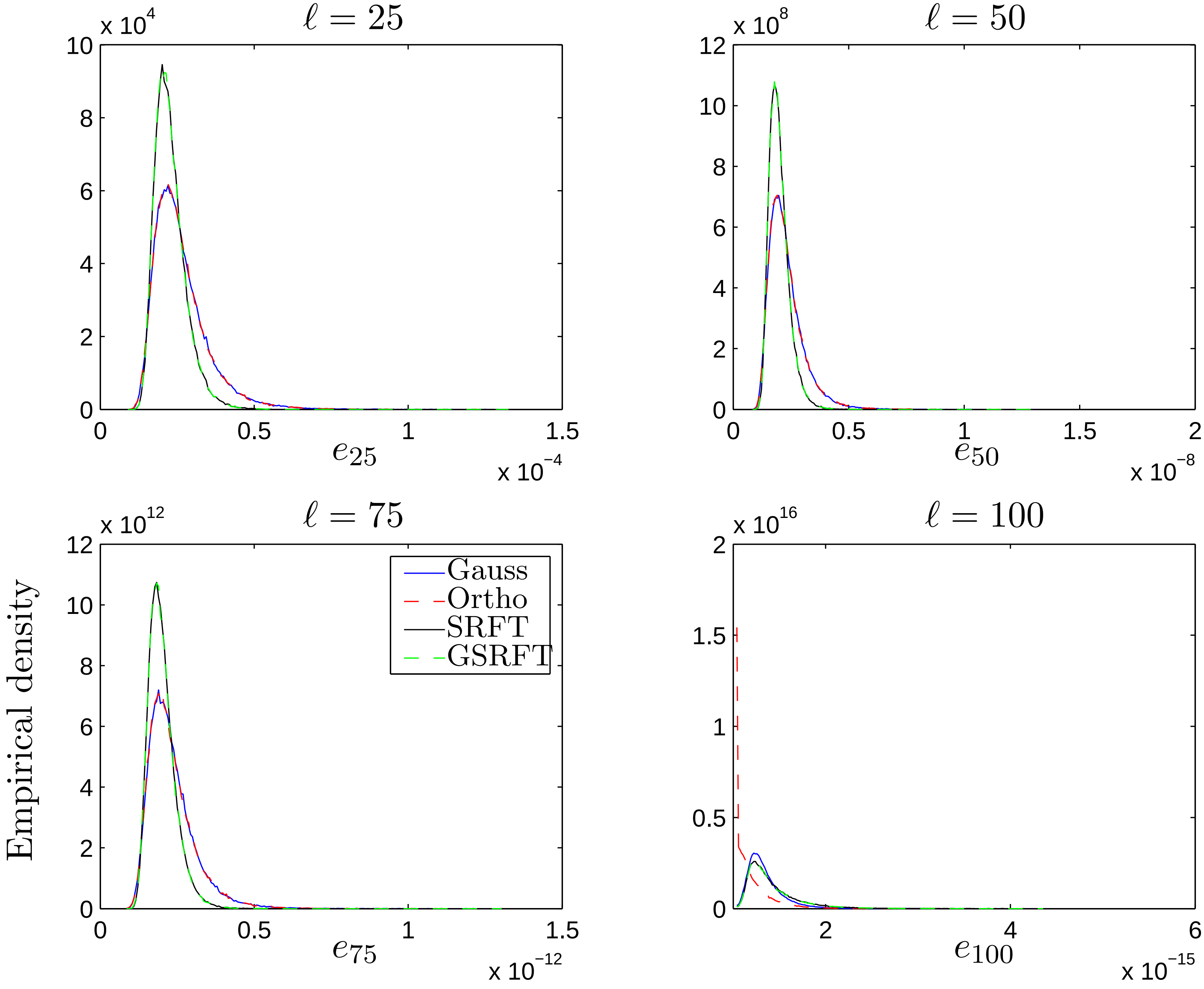
- *The "Gauss" method:*
  Gaussian random projection followed by SVD on the reduced matrix.

- *The "SRFT" method:*
  SRFT random projection followed by SVD on the reduced matrix.



$n = 1\,024$
$n = 2\,048$
$n = 4,096$
Acceleration factor
$t^{(\mathrm{direct})}/t^{(\mathrm{gauss})}$
$t^{(\mathrm{direct})}/t^{(\mathrm{srft})}$
$t^{(\mathrm{direct})}/t^{(\mathrm{svd})}$
0
1
2
3
4
5
6
7
10^1
10^2
10^3
ℓ

Empirical error distributions:



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.)

- Theory is poorly understood. Vastly overstates the risk of inaccurate results.

- Bibliographical notes:
  - The SRFT described was suggested by Nir Ailon and Bernard Chazelle (2006) in a related context.
  - Application to low-rank approximation by Liberty, Rokhlin, Tygert, Woolfe (2006).
  - Related recent work by Sarlós (on randomized regression).
  - Very interesting follow-up paper on overdetermined linear least-squares regression by Rokhlin and Tygert (2008).

Other applications of “structured” random projections:

- Fast algorithms for solving “least squares” problems (and hence linear regression).
  - Recent work by Vladimir Rokhlin and Mark Tygert in PNAS.

- Analysis of point sets in high dimensional spaces.
  - Recent development of a fast algorithm for finding nearest neighbors by Peter Jones and Vladimir Rokhlin.

- Pre-processing tool for putting linear problems in a “general” position. Possible “generic” pre-conditioner for linear systems.

Much more in the recent dissertation of Edo Liberty.

Very active area of research!

Outline of the tutorial:

1. Techniques for computing the SVD of a matrix of exact rank k.

2. Variations of techniques for matrices of exact rank.
   - Single pass algorithms.
   - How to compute spanning rows and spanning columns (CUR, etc).

3. Techniques for matrices of the form $\mathsf{A} = \mathsf{EF} + \mathsf{N}$ with N small.
   - Error estimation.

4. Techniques for matrices of the form $\mathsf{A} = \mathsf{EF} + \mathsf{N}$ with N large.
   - The "power method."

5. Random transforms that can be applied rapidly.
   - The "Subsampled Random Fourier Transform" (SRFT) and its cousins.

6. Review / Putting things together / Model problems.

Computing the SVD (or PCA):

Given: An $m \times n$ matrix A.

Task: Compute a k-term approximate SVD $\mathsf{A} \approx \mathsf{U}\Sigma\mathsf{V}^*$.

Pick an over sampling parameter p, say $p = 10$.

Stage A: Form an $m \times (k + p)$ random matrix Ω.

Form the $m \times (k + p)$ sample matrix $\mathsf{Y} = \mathsf{A}\,\Omega$.

Form the $m \times (k + p)$ orthonormal matrix Q such that $\text{Ran}(\mathsf{Q}) = \text{Ran}(\mathsf{Y})$.

Stage B: Compute $\mathsf{B} = \mathsf{Q}^*\mathsf{A}$.

Form the SVD of B so that $\mathsf{B} = \hat{\mathsf{U}}\,\Sigma\,\mathsf{V}^*$.

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

Notes:

- Error estimators can be incorporated so that k is determined adaptively.

Computing the SVD (or PCA) of A — accuracy enhanced version:

Given: An $m \times n$ matrix A.

Task: Compute a k-term approximate SVD $\mathsf{A} \approx \mathsf{U}\Sigma\mathsf{V}^*$:

Pick an over sampling parameter p, say $p = k$.

Stage A: Form an $m \times (k+p)$ random matrix Ω.

Form the $m \times (k+p)$ sample matrix $\mathsf{Y} = (\mathsf{A}\mathsf{A}^*)^q\,\mathsf{A}\,\Omega$.

Form the $m \times (k+p)$ orthonormal matrix Q such that $\mathrm{Ran}(\mathsf{Q}) = \mathrm{Ran}(\mathsf{Y})$.

Stage B: Compute $\mathsf{B} = \mathsf{Q}^*\mathsf{A}$.

Form the SVD of B so that $\mathsf{B} = \hat{\mathsf{U}}\,\Sigma\,\mathsf{V}^*$.

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

Notes:

- Requires $2q + 1$ passes over the matrix.

Computing spanning rows:

Given: An $m \times n$ matrix A.

Task: Find an index vector J and a matrix A such that $\mathsf{A} = \mathsf{X}\mathsf{A}(J, :\,)$.

Pick an over sampling parameter p, say $p = 10$.

Stage A: Form an $m \times (k + p)$ random matrix Ω.

Form the $m \times (k + p)$ sample matrix $\mathsf{Y} = \mathsf{A}\,\Omega$.

Stage B: Execute Gram-Schmidt on Y to find spanning rows: $\mathsf{Y} = \mathsf{X}\mathsf{Y}(J, :\,)$.

Notes:

- Further post-processing leads to SVD on the cheap.
- Can be combined with the “power scheme” for improved accuracy.

Computing spanning rows and columns:

Given: An $m \times n$ matrix A.

Task: Find a factorization $\mathsf{A} = \mathsf{X}_{\rm row}\mathsf{A}(J_{\rm row}, J_{\rm col})\mathsf{X}_{\rm col}$.

Pick an over sampling parameter p, say $p = 10$.

Stage A: Form random matrices with Ω and Ψ of sizes $n \times (k + p)$ and $m \times (k + p)$.

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

Stage B: Execute Gram-Schmidt on Y and Z to find spanning rows and columns

$\mathsf{Y} = \mathsf{X}_{\rm row}\mathsf{Y}(J_{\rm row}, :\,)$ and $\mathsf{Z} = \mathsf{Z}(:\,, J_{\rm col})\mathsf{X}_{\rm col}$.

Notes:

- Further post-processing leads to SVD on the cheap.
- Can be combined with the “power scheme” for improved accuracy.
- The method is one-pass (not counting extraction of $\mathsf{A}(J_{\rm row}, J_{\rm col})$).

A one-pass algorithm for a symmetric matrix:

Given: An $n \times n$ symmetric matrix A.

Task: Compute a k-term approximate eigenvalue decomposition $\mathsf{A} \approx \mathsf{U}\mathsf{\Lambda}\mathsf{U}^*$:

Pick an over sampling parameter p, say $p = 10$.

Stage A: Form an $m \times (k+p)$ random matrix $\mathsf{\Omega}$.

Form the $m \times (k+p)$ sample matrix $\mathsf{Y} = \mathsf{A}\,\mathsf{\Omega}$.

Form the $m \times (k+p)$ orthonormal matrix Q such that $\text{Ran}(\mathsf{Q}) = \text{Ran}(\mathsf{Y})$.

Stage B: Solve for B the system $\mathsf{Q}^*\mathsf{Y} = \mathsf{B}\big(\mathsf{Q}^*\mathsf{\Omega}\big)$.

Factor the small matrix $\mathsf{B} = \mathsf{U}\mathsf{\Lambda}\mathsf{U}^*$.

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

Notes:

- Not currently known how to achieve an "accuracy enhanced" single-pass method.

A one-pass algorithm for a non-symmetric matrix:

Given: An $m \times n$ matrix A.

Task: Compute a k-term approximate singular value decomposition $\mathsf{A} \approx \mathsf{U}\Sigma\mathsf{V}^*$:

Pick an over sampling parameter p, say $p = 10$.

Stage A: Form random matrices with Ω and Ψ of sizes $n \times (k+p)$ and $m \times (k+p)$.

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

Form the orthonormal matrices Q and W

such that $\mathrm{Ran}(\mathsf{Q}) = \mathrm{Ran}(\mathsf{Y})$ and $\mathrm{Ran}(\mathsf{W}) = \mathrm{Ran}(\mathsf{Z})$.

Stage B: Solve for B the systems

$\mathsf{Q}^*\mathsf{Y} = \mathsf{B}\big(\mathsf{W}^*\Omega\big)$ and $\mathsf{W}^*\mathsf{Z} = \mathsf{B}^*\big(\mathsf{Q}^*\Psi\big)$.

Factor the small matrix $\mathsf{B} = \mathsf{U}\Sigma\mathsf{V}^*$.

Compute the matrices $\mathsf{U} = \mathsf{Q}\,\hat{\mathsf{U}}$ and $\mathsf{V} = \mathsf{W}\,\hat{\mathsf{V}}$.

Miscellaneous remarks:

Connection to "Johnson-Lindenstrauss theory":

Lemma: *Let* ε *be a real number such that* $\varepsilon \in (0, 1)$, *let* n *be a positive integer, and let* k *be an integer such that*

$$k \geq 4 \left(\frac{\varepsilon^2}{2} - \frac{\varepsilon^3}{3} \right)^{-1} \log(n). \tag{9}$$

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

$$(1-\varepsilon)\,||\boldsymbol{u}-\boldsymbol{v}||^2 \leq ||f(\boldsymbol{u}) - f(\boldsymbol{v})|| \leq (1+\varepsilon)\,||\boldsymbol{u}-\boldsymbol{v}||^2, \qquad \forall\ \boldsymbol{u},\, \boldsymbol{v} \in V. \tag{10}$$

Further, such a map can be found in randomized polynomial time.

It has been shown that an excellent choice of the map f is the linear map whose coefficient matrix is a $k \times d$ matrix whose entries are i.i.d. Gaussian random variables (see, *e.g.* Dasgupta & Gupta (1999)).
When k satisfies, (9), this map satisfies (10) with probability close to one.

The related Bourgain embedding theorem shows that such statements are not restricted to Euclidean space:

Theorem:. *Every finite metric space* (X, d) *can be embedded into* ℓ^2 *with distortion* $O(\log n)$ *where* n *is the number of points in the space.*

Again, random projections can be used as the maps.

The Johnson-Lindenstrauss lemma (and to some extent the Bourgain embedding theorem) expresses a theme that is recurring across a number of research areas that have received much attention recently. These include:

- Compressed sensing (Candès, Tao, Romberg, Donoho).
- Approximate nearest neighbor search (Jones, Rokhlin).
- Geometry of point clouds in high dimensions (Coifman, Jones, Lafon, Lee, Maggioni, Nadler, Singer, Warner, Zucker, *etc*).
- Construction of multi-resolution SVDs.
- Clustering algorithms.
- Search algorithms / knowledge extraction.

Note: Omissions! No ordering. Missing references. Etc etc.

Many of these algorithms work "unreasonably well."

The randomized algorithm presented here is 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.)

Observation: Mathematicians working on these problems often focus on minimizing the distortion factor

$$\frac{1+\varepsilon}{1-\varepsilon}$$

arising in the Johnson-Lindenstrauss bound:

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

In our environments, we do not need this constant to be particularly close to 1. It should just not be “large” — say less that 10 or some such.

This greatly reduces the number of random projections needed!
Recall that in the Johnson-Lindenstrauss theorem:

$$\text{number of samples required } \sim \frac{1}{\varepsilon^2}\log(N).$$

Observation: Multiplication by a random unitary matrix reduces any matrix to its "general" form. All information about the singular vectors vanish. (The singular *values* remain the same.)

This opens up the possibility for general pre-conditioners — counterexamples to various algorithms can be disregarded.

The feasibility has been demonstrated for the case of least squares solvers for very large, very over determined systems. (Work by Rokhlin & Tygert, Sarlós, ...)

Work on $O(N^2\,(\log N)^2)$ solvers of general linear systems is under way. (Random pre-conditioning + iterative solver.)

May stable fast matrix inversion schemes for general matrices be possible?

Observation: Robustness with respect to the quality of the random numbers.

The assumption that the entries of the random matrix are i.i.d. normalized Gaussians simplifies the analysis since this distribution is invariant under unitary maps.

In practice, however, one can use a low quality random number generator. The entries can be uniformly distributed on $[-1, 1]$, they be drawn from certain Bernouilli-type distributions, *etc.*

Remarkably, they can even have enough internal structure to allow fast methods for matrix-vector multiplications. For instance:

- Subsampled discrete Fourier transform.
- Subsampled Walsh-Hadamard transform.
- Givens rotations by random angles acting on random indices.

This was exploited in the $O(n^2 \log k)$ technique (see also Ailon-Chazelle results).
Our theoretical understanding of such problems is unsatisfactory.
Numerical experiments perform *far* better than existing theory indicates.

Even though it is thorny to *prove* some of these results (they draw on techniques from numerical analysis, probability theory, functional analysis, theory of randomized algorithms, *etc*), work on randomized methods in linear algebra is progressing fast.

Important: Computational prototyping of these methods is extremely simple.

- Simple to code an algorithm.
- They work so well that you immediately know when you get it right.

Final remarks:

- The theory can be hard, but *experimentation is easy!*
  Concentration of measure makes the algorithms behave as if deterministic.

- The tutorial mentioned *error estimators* only briefly, but they are important.
  Can operate independently of the algorithm for improved robustness.
  Typically cheap and easy to implement.

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

- Other web resources:
  - Notes are posted → google *Gunnar Martinsson*.
    Will also be posted by NIPS.
  - Review article: *Finding structure with randomness: Stochastic algorithms for constructing approximate matrix decompositions*
    N. Halko, P.G. Martinsson, J. Tropp — arXiv.org report 0909.4061.