Randomized Matrix Decompositions using 'rsvd'

N. Benjamin Erichson
Joint work with Sergey Voronin, Steven Brunton and Nathan Kutz

University of St Andrews

nbe@st-andrews.ac.uk

March 15, 2017

EPSRC
Engineering and Physical Sciences Research Council



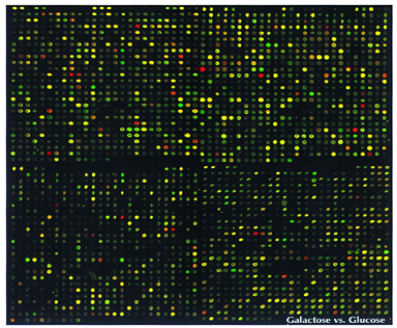
University of St Andrews | FOUNDED 1413

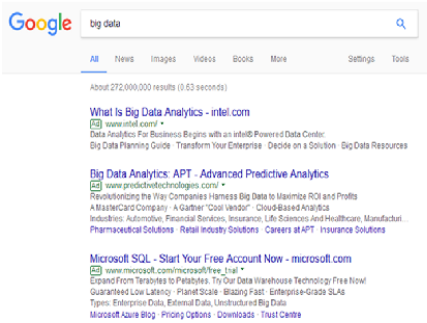
Today's talk

1. Matrix Decompositions
2. Probabilistic Framework
3. Randomized Matrix Decompositions using the 'rsvd' Package
   1. Randomized Singular Value Decomposition
   2. Randomized Principal Component Analysis
   3. Randomized CUR Decomposition
4. Take Aways

Motivation: High-Dimensional Data

We are drowning in information
and starving for knowledge.
- Rutherford D. Roger



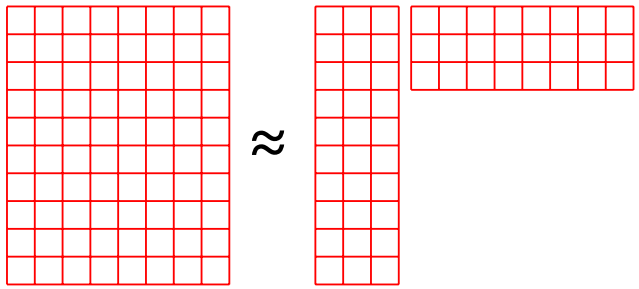


- **Digital images and videos**
- **Microarrays**
- **Language**
- **Web data**
- **User data**

Low-Rank Matrix Approximations

- Many high-dimensional data feature low-rank structure.
- Then, it is possible to construct an approximate factorization.

$$\underset{m \times n}{\mathbf{A}} \approx \underset{m \times k}{\mathbf{E}} \underset{k \times n}{\mathbf{F}} \tag{1}$$



- The factors **E** and **F** can be used to:
  - Summarize or to reveal some interesting structure in the data;
  - Efficiently store the large data matrix;
- **Applications:** Dimension reduction, compression, signal denoising, classification, regression, clustering, reconstruction, ...

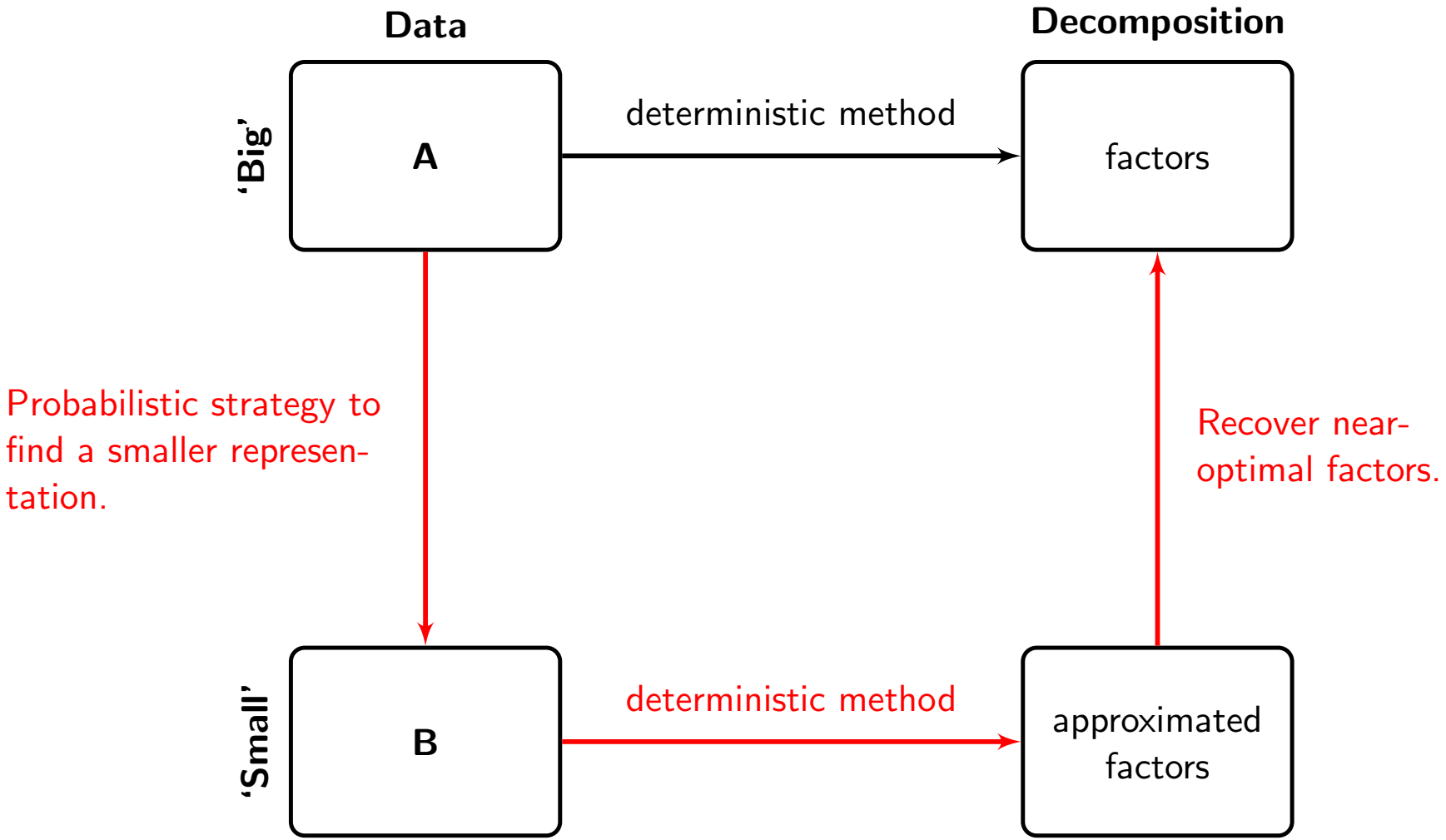
Some Important Matrix Decompositions

Each method yields a unique **view** on the data:

- Singular value decomposition.
- Principal component analysis.
- Sparse principal component analysis.
- Dynamic Mode Decomposition.
- CUR decomposition.
- Non-negative matrix factorizations.

➔ 'Big data' poses a computational challenge for deterministic algorithms.

Probabilistic Framework: Conceptual Overview



How to Find The Small Matrix **B** ?

- **Stage A:** Find a matrix **Q** with orthonormal columns, such that

$$\underset{m \times n}{\mathbf{A}} \approx \underset{m \times k}{\mathbf{Q}} \quad \underset{k \times m}{\mathbf{Q}^\top} \quad \underset{m \times n}{\mathbf{A}} \tag{2}$$

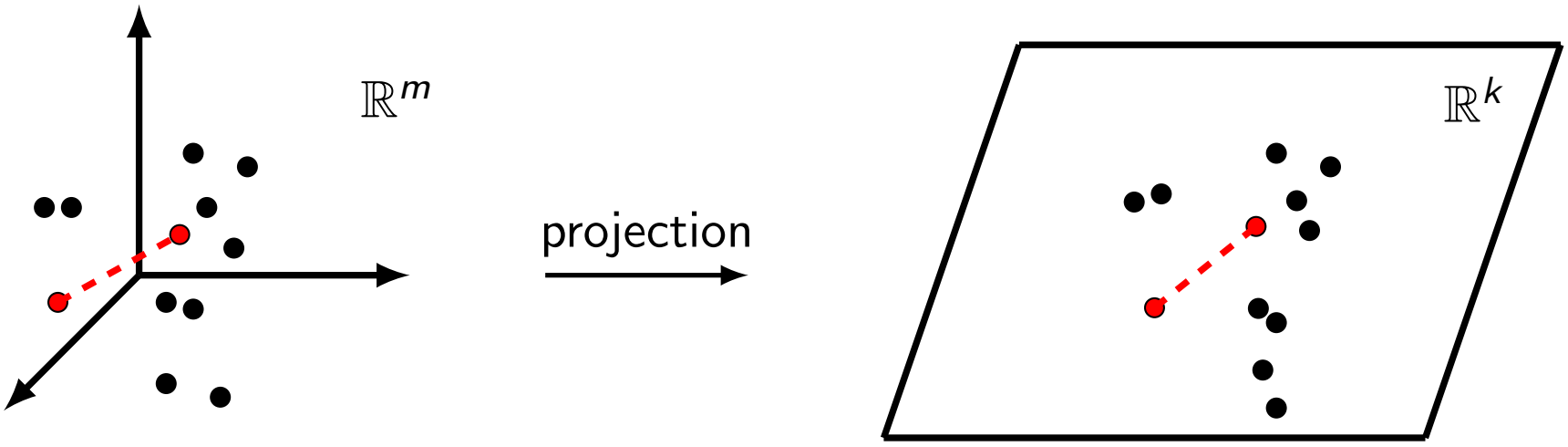
- **Stage B:** Form **B** by restricting the input matrix to the low-dimensional space spanned by the natural basis **Q**:

$$\underset{k \times n}{\mathbf{B}} := \underset{k \times m}{\mathbf{Q}^\top} \quad \underset{m \times n}{\mathbf{A}} \tag{3}$$

➔ See the seminal paper by Halko, Martinsson, and Tropp [1] for technical details.

Geometric Interpretation

The projection $\mathbf{B} = \mathbf{Q}^\top \mathbf{A}$ takes points in a high-dimensional space into corresponding points in a low-dimensional space.



➔ This process preserves the geometric structure in an Euclidean sense, because inner products are preserved.

How Does Randomness Come Into Play?

- We draw k random vectors ω to sample the range of the input matrix

$$\mathbf{y}_i := \mathbf{A}\omega_i \quad \text{for } i=1,2,....,k \tag{4}$$

- Using matrix notation, a sample matrix is formed as

$$\underset{m \times k}{\mathbf{Y}} := \underset{m \times n}{\mathbf{A}} \quad \underset{n \times k}{\mathbf{\Omega}} \tag{5}$$

- Finally, a normal basis $\mathbf{Q}$ is obtained by computing the QR-Decomposition of the sample matrix $\mathbf{Y} = \mathbf{QR}$.

Oversampling

- Most real data matrices do not feature exact rank.
- Better to use $l = k + p$ samples instead of just k samples:

$$\underset{m \times l}{\mathbf{Y}} := \underset{m \times n}{\mathbf{A}} \; \underset{n \times l}{\mathbf{\Omega}} \tag{6}$$

- A small amount of oversampling is often sufficient, e.g., $p = \{5, 10\}$.

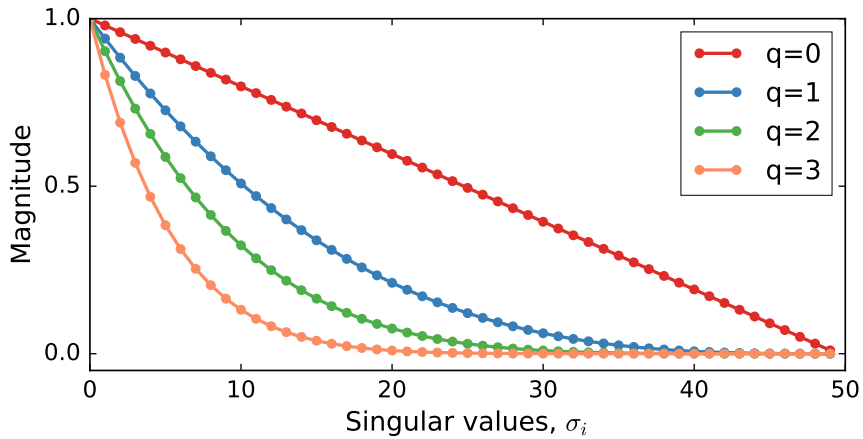
Power Scheme

- The singular value spectrum is often slow decaying.
- Better to preprocess the input matrix $\mathbf{A}$ using power iterations

$$\underset{m \times l}{\mathbf{Y}} := \underset{m \times n}{\mathbf{A}^{(q)}} \underset{n \times l}{\mathbf{\Omega}} \tag{7}$$

- $\mathbf{A}^{(q)} = (\mathbf{A}\mathbf{A}^\top)^q \mathbf{A} = \mathbf{U}\mathbf{\Sigma}^{2q+1}\mathbf{V}^\top$.

- Requires q additional passes over the input matrix.
- A small number of power iterations is often sufficient, e.g., $q = \{1, 2\}$.



Theoretical Performance

- Both the concept of oversampling and the power scheme allow to control the quality of the basis matrix $\mathbf{Q}$.
- Martinsson [2] provides the following (simplified) description of the average case behavior of the outlined probabilistic framework

$$\mathrm{E}\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^\top \mathbf{A}\|_2 \leq \left[1 + \sqrt{\frac{k}{p-1}} + \frac{e\sqrt{l}}{p} \cdot \sqrt{n-k}\right]^{\frac{1}{2q+1}} \sigma_{k+1}(\mathbf{A}).$$

- Assuming that the oversampling parameter $p \geq 2$, and $n \leq m$.
- E denotes the expectation with respect to a Gaussian test matrix $\mathbf{\Omega}$
- $\sigma_{k+1}(\mathbf{A})$ denotes the smallest possible error achievable.

Computational Considerations

- The probabilistic framework reduces the time complexity from $O(mn^2)$ to $O(mnk)$.
- Only two passes over the input matrix $\mathbf{A}$ are required.
- The computational steps are simple to implement.
- The procedure is robust, and reliable.

The ‘rsvd’ Package

- Randomized matrix algorithms are becoming increasingly popular!
- The rsvd package [3] aims to fill the gap in R, providing the routines:
  - Randomized singular value decomposition: rsvd().
  - Randomized principal component analysis: rpca().
  - Randomized robust principal component analysis: rrpca().
- The interface is similar to the R base functions svd() and prcomp().
- Utility functions are provided for the rpca() function.
- Project page: https://github.com/Benli11/rSVD.
- CRAN: https://cran.r-project.org/web/packages/rsvd.

Recap: Singular Value Decomposition (SVD)

- The low-rank SVD attains a factorization of the form

$$\underset{m \times n}{\mathbf{A}} \approx \underset{m \times k}{\mathbf{U}} \quad \underset{k \times k}{\mathbf{\Sigma}} \quad \underset{k \times n}{\mathbf{V}^\top} \tag{8}$$

- **U** are the left singular vectors.
  - They provide a basis for the range of the matrix **A**.
- **V** are the right singular vectors.
  - They provide a basis for the domain of the matrix **A**.
- **Σ** contains the non-negative singular values $\sigma_1 \geq ... \geq \sigma_n$.
  - They describing the spectrum of the data.

'Vanilla' rSVD Algorithm

Given an input matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, a target rank k and an oversampling parameter p, the approximated SVD is computed as

Stage A

1. *Draw a random test matrix:* $\mathbf{\Omega} \in \mathbb{R}^{n \times (k+p)}$
2. *Compute sample matrix:* $\mathbf{Y} = \mathbf{A\Omega} \in \mathbb{R}^{m \times (k+p)}$
3. *Compute QR-decomposition:* $\mathbf{Y} = \mathbf{QR}$

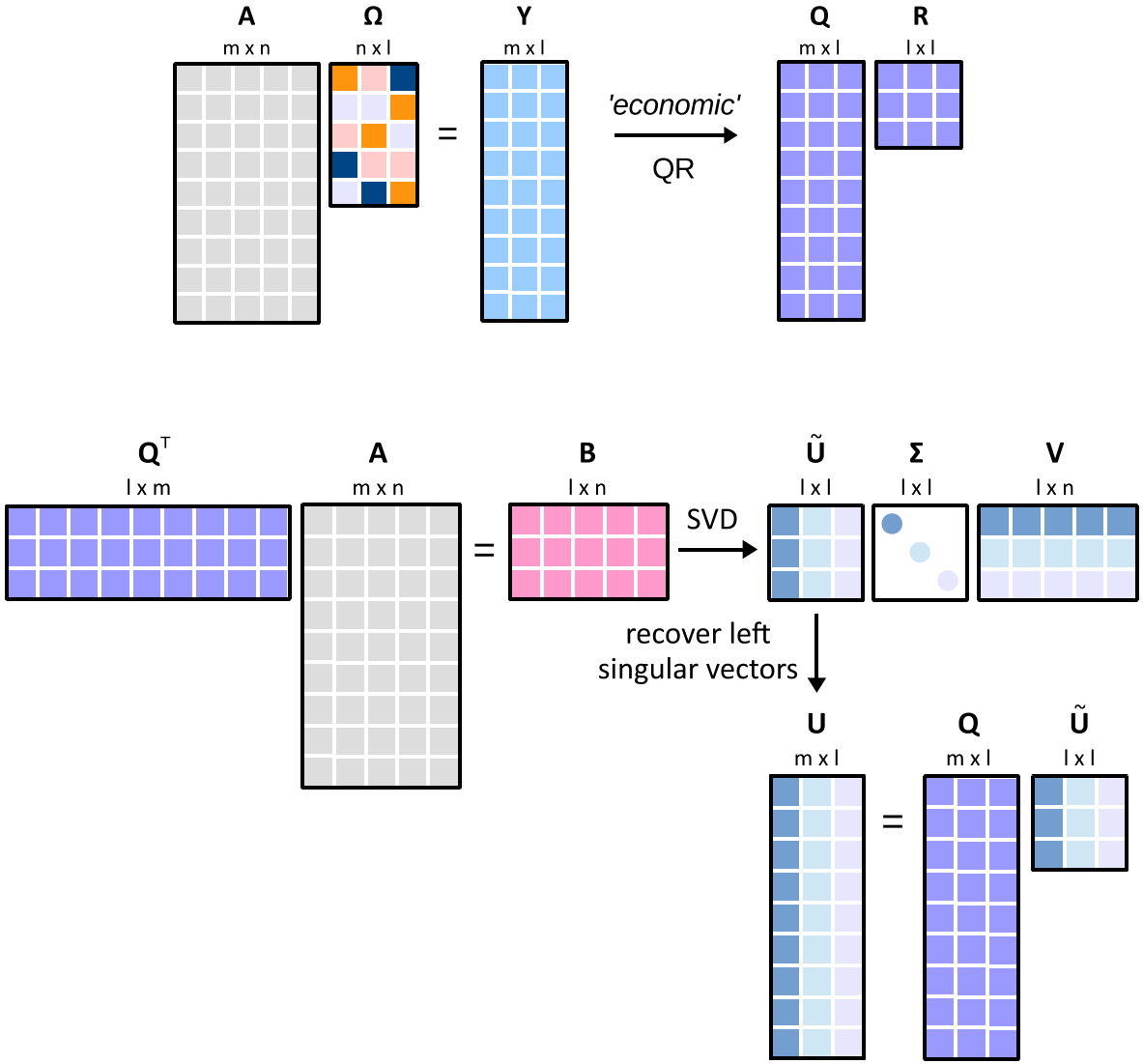
Stage B

4. *Compute matrix product:* $\mathbf{B} = \mathbf{Q}^\top \mathbf{A} \in \mathbb{R}^{(k+p) \times n}$
5. *Compute deterministic SVD:* $\mathbf{B} = \tilde{\mathbf{U}}\mathbf{\Sigma}\mathbf{V}^\top$
6. *Recover right singular vectors:* $\mathbf{U} \approx \mathbf{Q}\tilde{\mathbf{U}}$

Verification

$$\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^\top\mathbf{A} = \mathbf{QB} = \mathbf{Q}\tilde{\mathbf{U}}\mathbf{\Sigma}\mathbf{V}^\top = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$$

Illustrated 'Vanilla' rSVD Algorithm



The rsvd() function

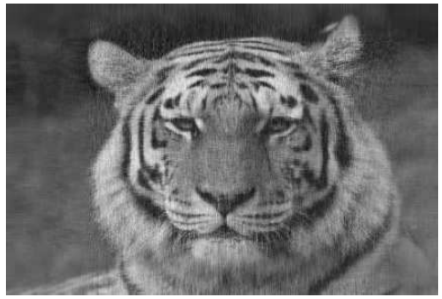
```
s <- rsvd(A, k, p = 10, q = 1, ...)
```

- Arguments:
  - A: input data matrix of dimension $m \times n$.
  - k: target rank.
  - p: controls amount of oversampling (default $p = 10$).
  - q: controls number of additional power iterations (default $q = 1$).
- Returns a list with the following components:
  - d: k-dimensional vector containing the singular values.
  - u: $m \times k$ matrix containing the left singular vectors.
  - v: $n \times k$ matrix containing the right singular vectors.

Example: Image Compression (target rank k=100)

(a) Original image.

(b) SVD. (nrmse=0.121)



(c) rSVD, $q = 0$. (nrmse=0.163)

(d) rSVD, $q = 1$. (nrmse=0.125)

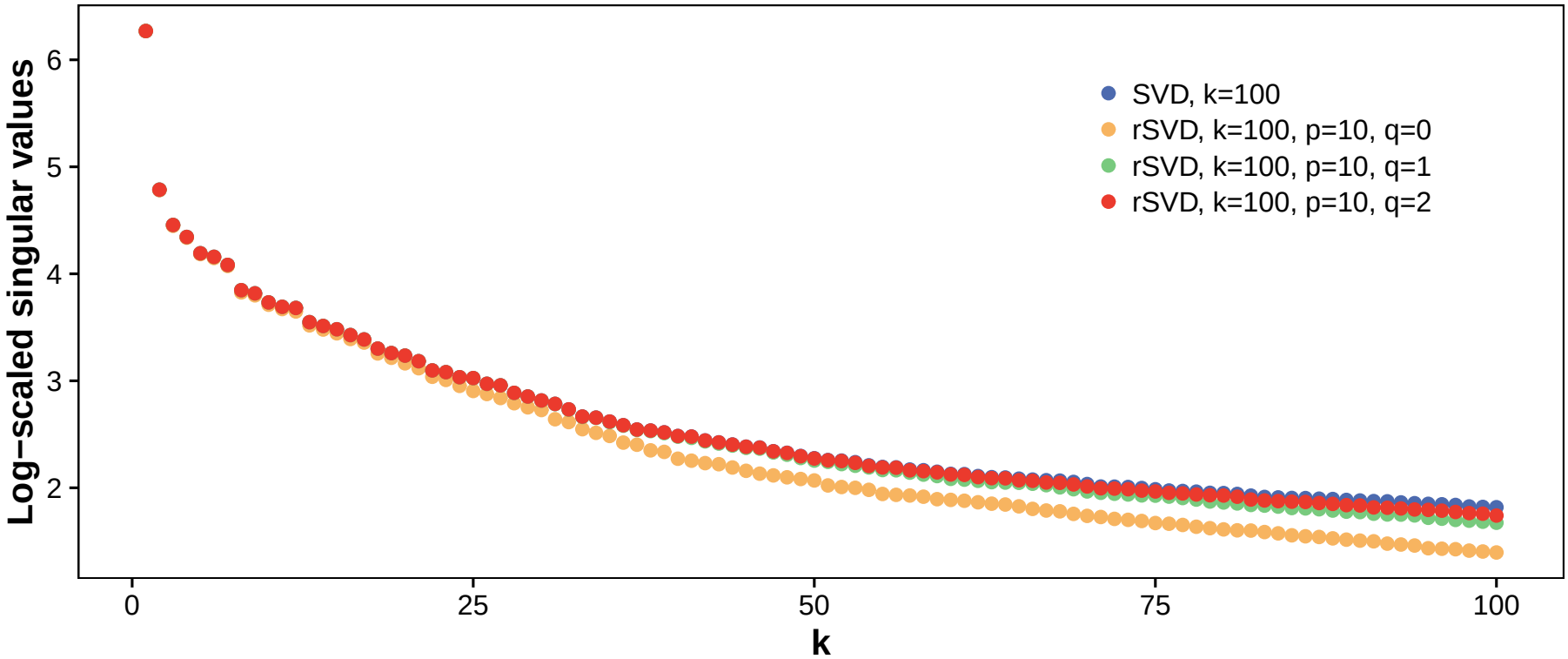
(e) rSVD, $q = 2$. (nrmse=0.122)

(f) rSVD, $q = 3$. (nrmse=0.121)

```
1 R> data("tiger")
2 R> s <- rsvd(tiger, k = 100, p = 10, q = 1)
3 R> tiger.re <- s$u %*% diag(s$d) %*% t(s$v)
4 R> image(tiger.re, col = gray(0:255 / 255))
5 R> nrmse <- sqrt(sum((tiger - tiger.re) ** 2) / sum(tiger ** 2))
```

Example: Image Compression (target rank k=100)

- Singular value spectrum is faithfully captured.

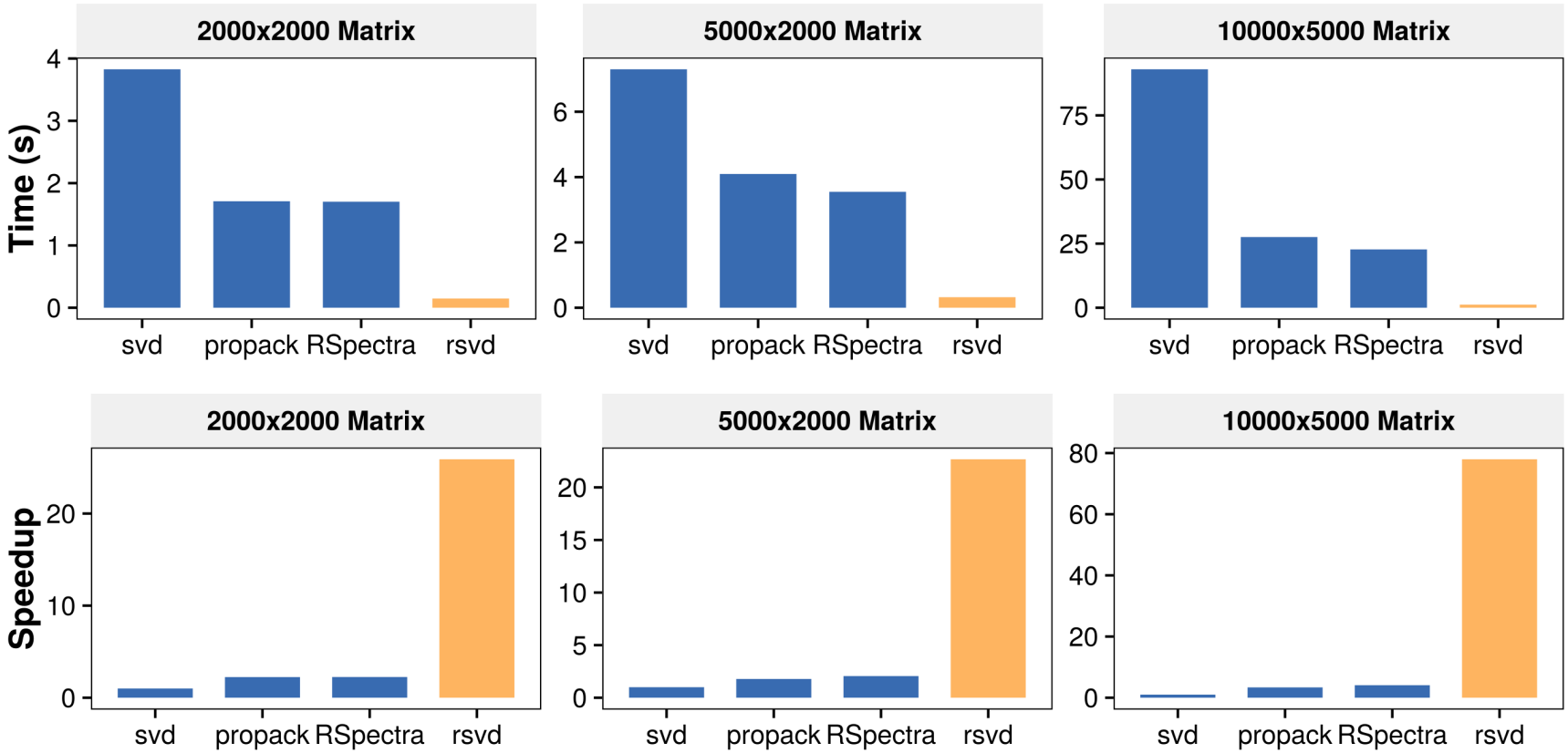


Example: Image Compression (target rank k=100)

Method	Parameters	Time (s)	Speedup	Error
svd()		1.05	-	0.121
propack.svd()	neig=100	0.94	1.11	0.121
svds()	k=100	0.25	4.21	0.121
rsvd()	k=100, q=0	0.08	12.80	0.163
	k=100, q=1	0.15	7.05	0.125
	k=100, q=2	0.22	4.86	0.122
	k=100, q=3	0.29	3.62	0.121

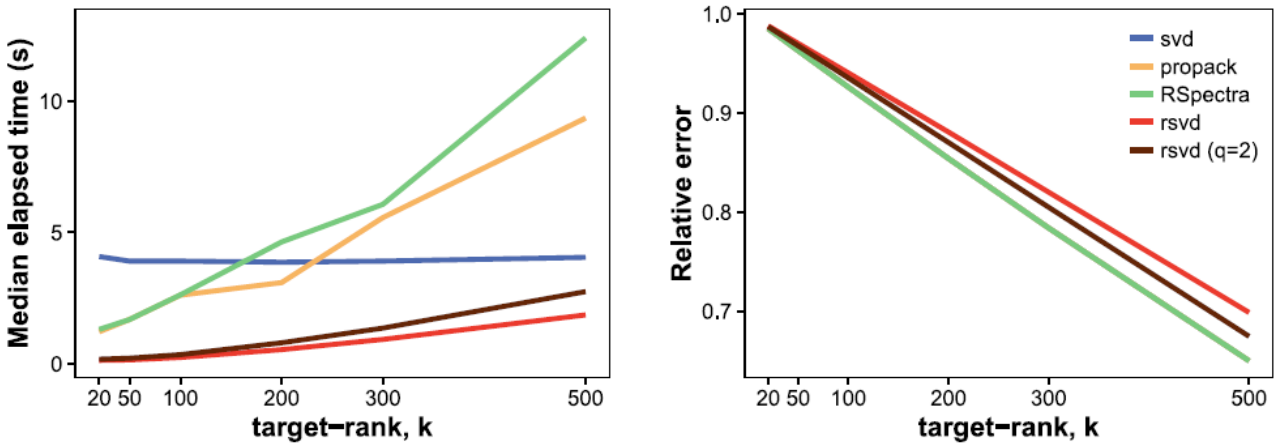
- 'PROPACK': propack.svd() is based on the Lanczos algorithm.
- 'RSpectra': svds() is based on the ARPACK software package.

Computational Performance: Target Rank k=50

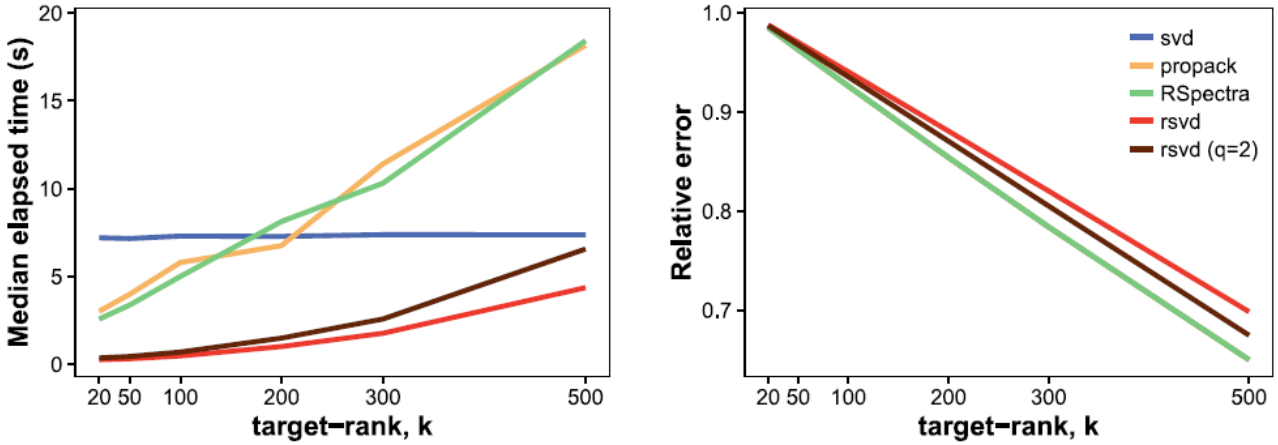


- Computational performance becomes more pronounced for bigger matrices !

Computational Performance... Dense Matrices

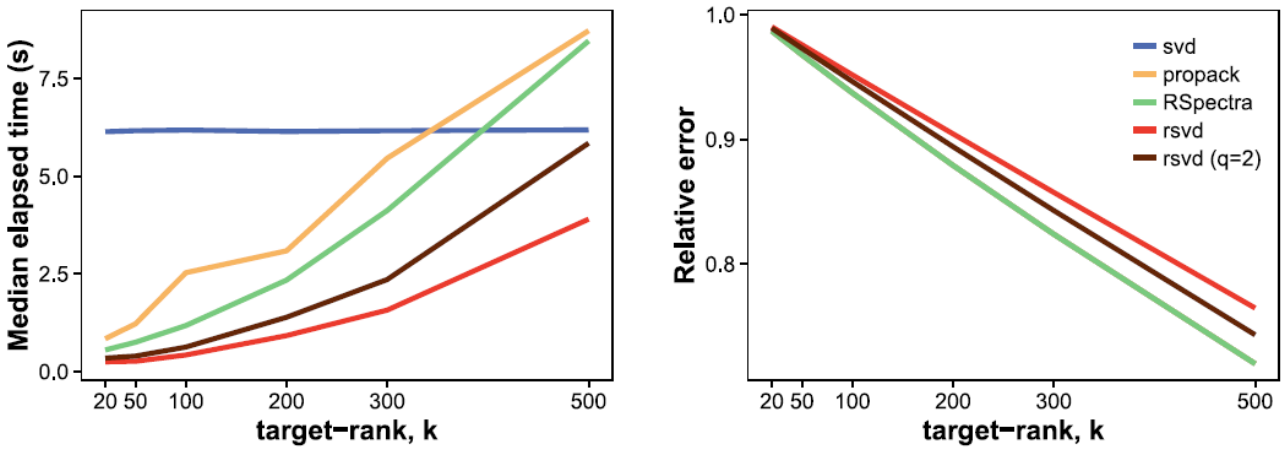


(a) Dense matrix of size 2000×2000.

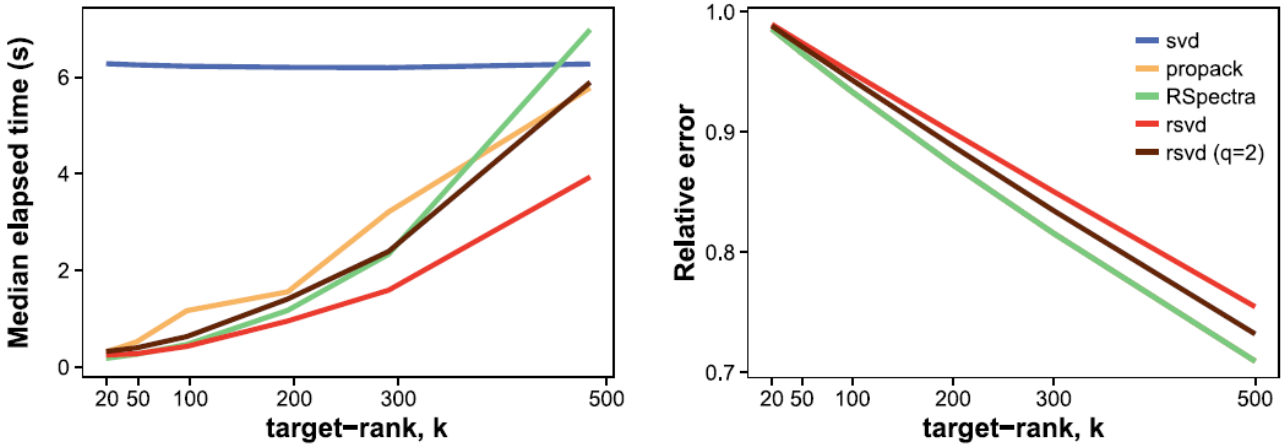


(b) Dense matrix of size 5000×2000.

Computational Performance... Sparse Matrices



(a) Sparse matrix of size 5000×2000, with 5% non-zero entries.



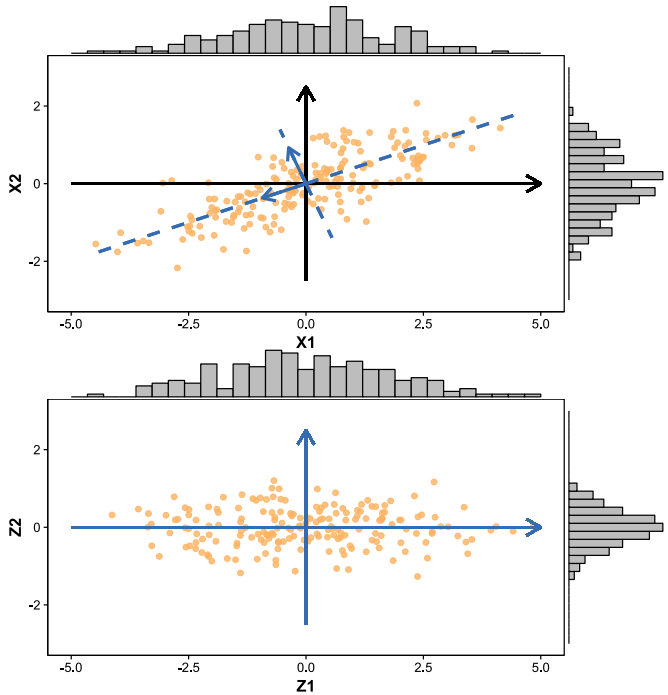
(b) Very sparse matrix of size 5000×2000, with 1% non-zero entries.

Recap: Principal Component Analysis (PCA)

- The essential idea of PCA is to find a new set of uncorrelated variables that retain most of the information present in the data.

- PCA attains a factorization of the form

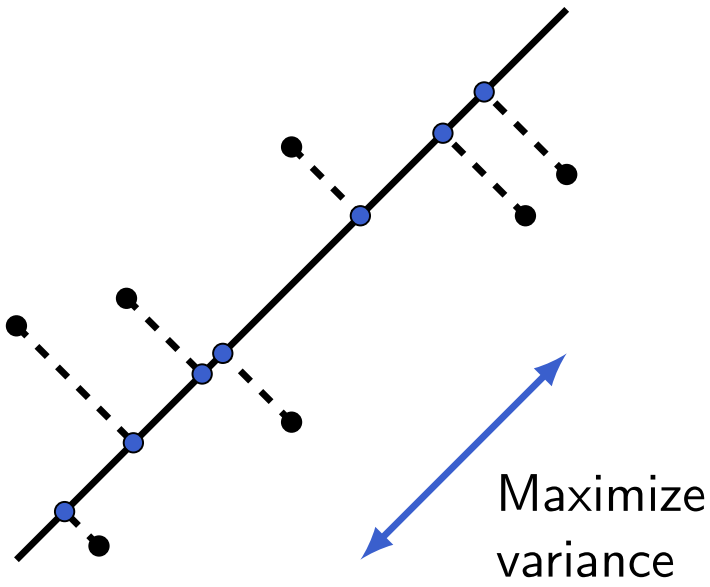
$$\underset{m \times n}{\mathbf{X}} \approx \underset{m \times k}{\mathbf{Z}} \; \underset{k \times n}{\mathbf{W}^\top} \tag{9}$$

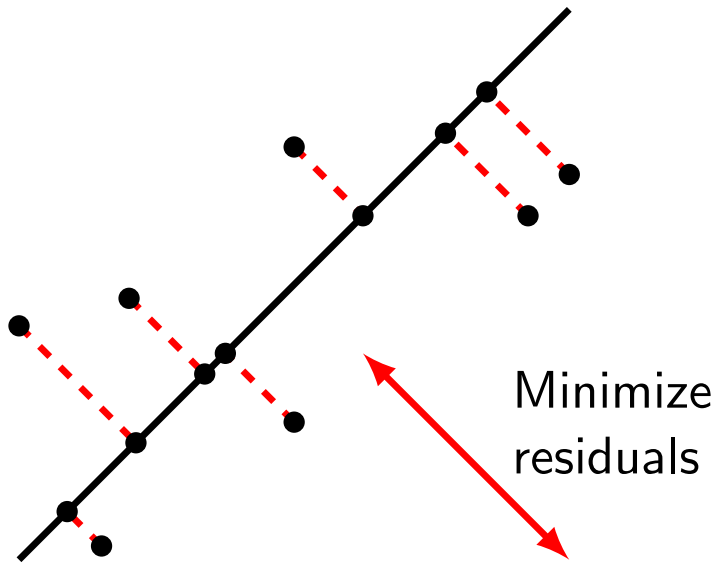


- $\mathbf{X}$: mean centered input matrix.
- $\mathbf{Z}$: principal components (rotated data).
- $\mathbf{W}$: principle directions (eigenvectors).

- PCA is closely related to the SVD!

The Two Views of Principal Component Analysis





- PCA can be formulated either as a variance maximization or a least square minimization problem.
- Both views are equivalent and lead to an eigenvalue problem.

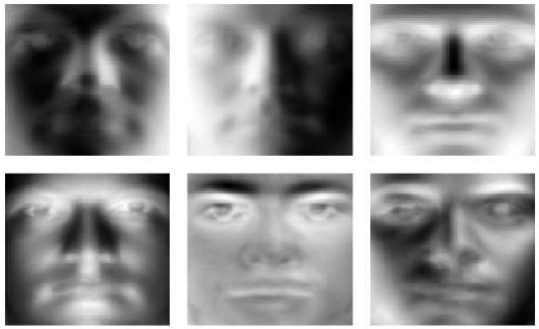
The rpca() function

```
s <- rpca(A, k = NULL, center = TRUE, scale = TRUE,
          retx = FALSE, p = 10, q = 1, ...)
```

- Arguments:
  - A: input data matrix of dimension $m \times n$.
  - k: target rank.
  - center and scale: use covariance or correlation matrix.
  - retx: if TRUE, rotated variables (scores) are returned.
  - p: controls amount of oversampling (default $p = 10$).
  - q: controls number of additional power iterations (default $q = 1$).
- Returns a list with the following components:
  - rotation: $n \times k$ matrix containing the rotations (eigenvectors).
  - eigvals: k-dimensional vector containing the eigenvalues.
  - x: $m \times k$ matrix containing the scores (rotated data).

.

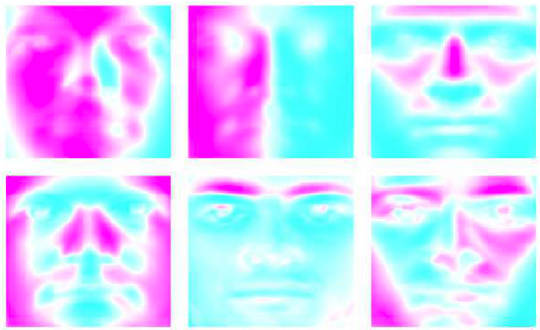
Example: Eigenfaces (target rank k=20)



(a) Eigenfaces.

(b) Colored eigenfaces.

(c) Randomized eigenfaces.



(d) Colored randomized eigenfaces.

```
R> data("faces")
R> s <- rpca(t(faces), k = 20, center = TRUE, scale = TRUE)
R> eigenface <- matrix(rev(s$rotation[ , 1]), nrow = 84, ncol = 96)
R> image(eigenface, col = gray(0:255 / 255))
```

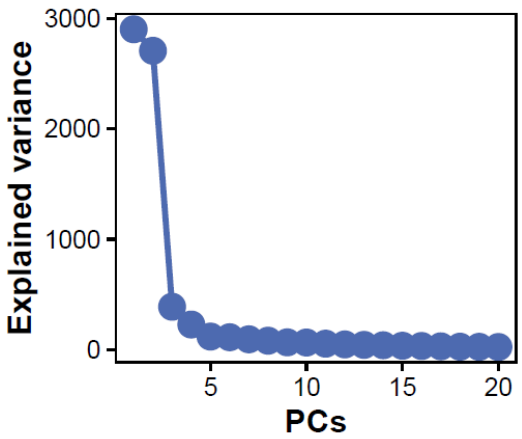
Example: Eigenfaces (target rank k=20)

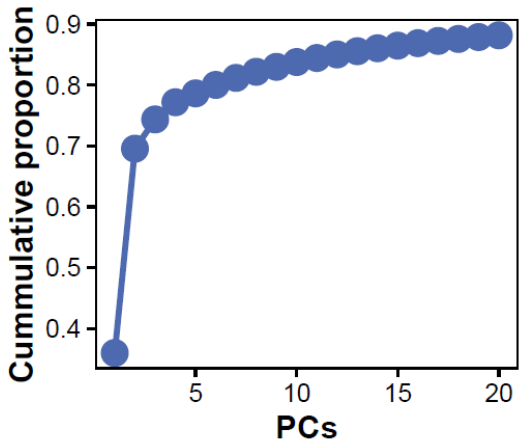
Method	Parameters	Time (s)	Speedup	Error
prcomp()		19.13	-	0.228
rpca()	k=20, q=1	1.90	10.05	0.232
	k=20, q=2	2.11	9.08	0.229

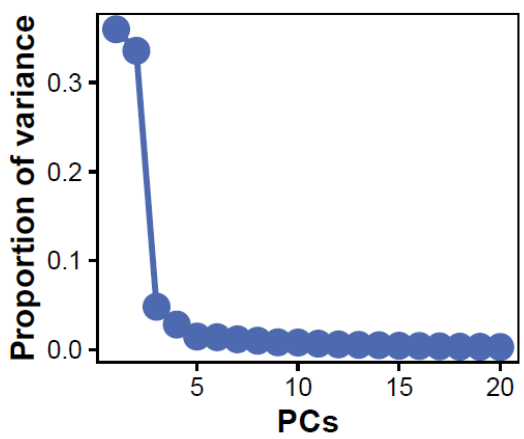
Utility Functions for rpca()

```
R> summary(s)
                              PC1         PC2       PC3     ...
Explained variance       2901.539   2706.699   388.053   ...
Standard deviations        53.866     52.026    19.699   ...
Proportion of variance      0.360      0.336     0.048   ...
Cumulative proportion       0.360      0.695     0.744   ...
Eigenvalues              2901.539   2706.699   388.053   ...
```

```
R> ggscreeplot(s)
R> ggscreeplot(s, type = "cum")
R> ggscreeplot(s, type = "ratio")
```







Recap: CUR Decomposition

- Let $\mathbf{A}$ be an $m \times n$ matrix, then the CUR decomposition is

$$\underset{m \times n}{\mathbf{A}} \approx \underset{m \times c}{\mathbf{C}} \quad \underset{c \times r}{\mathbf{U}} \quad \underset{r \times n}{\mathbf{R}} \tag{10}$$

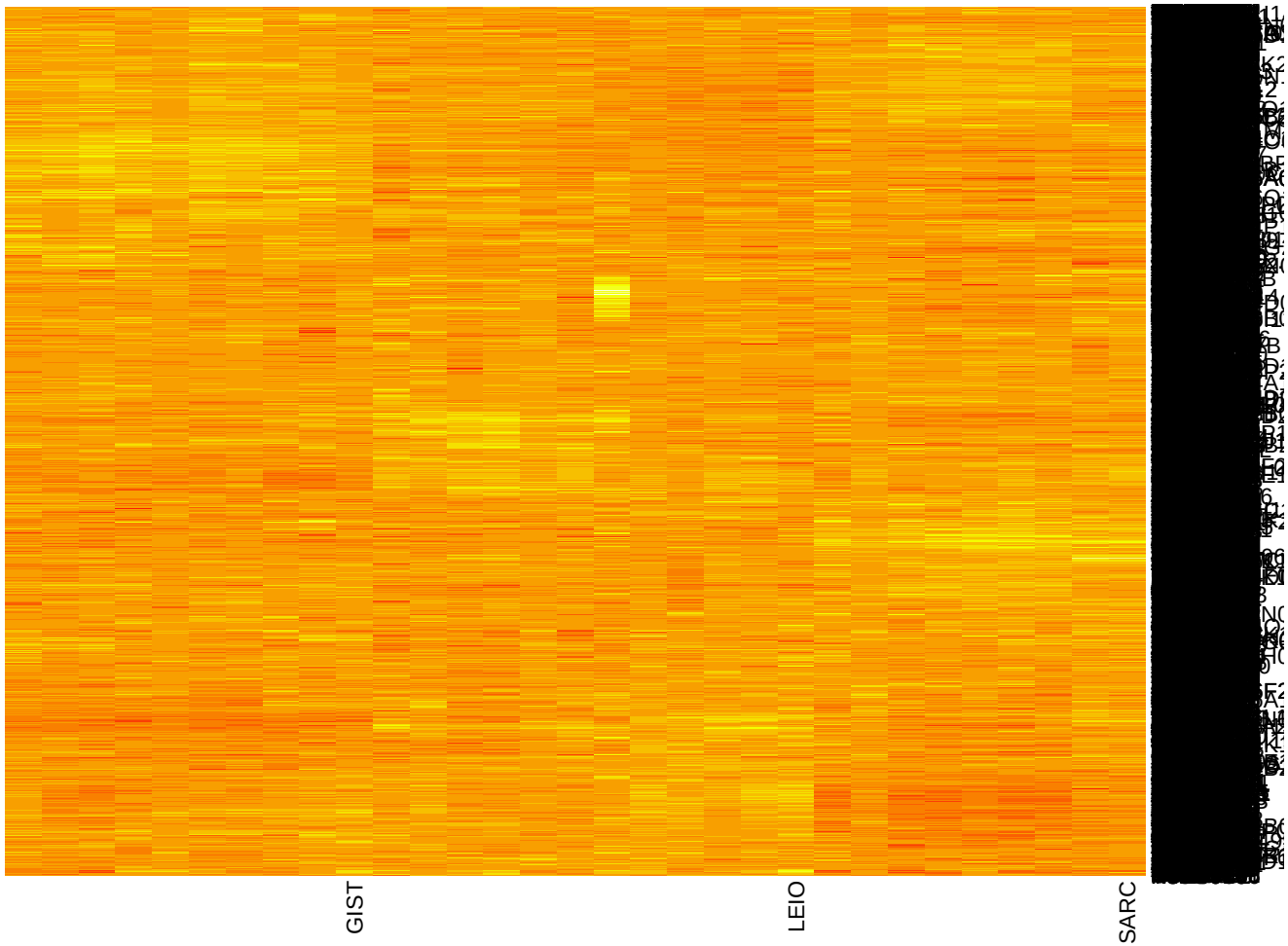
- $\mathbf{C}$ contains a small subset of columns of $\mathbf{A}$.
- $\mathbf{R}$ contains a small subset of rows of $\mathbf{A}$.
- $\mathbf{U}$ is a matrix such that Equation (10) is satisfied.

Why Using the CUR Decomposition?

- Better **interpretability**, because actual columns/rows are used.
- Useful if data feature structure like
  - low-rank.
  - sparsity.
  - non-negativity.
- Great for genetic data, digital images, and web/user data.

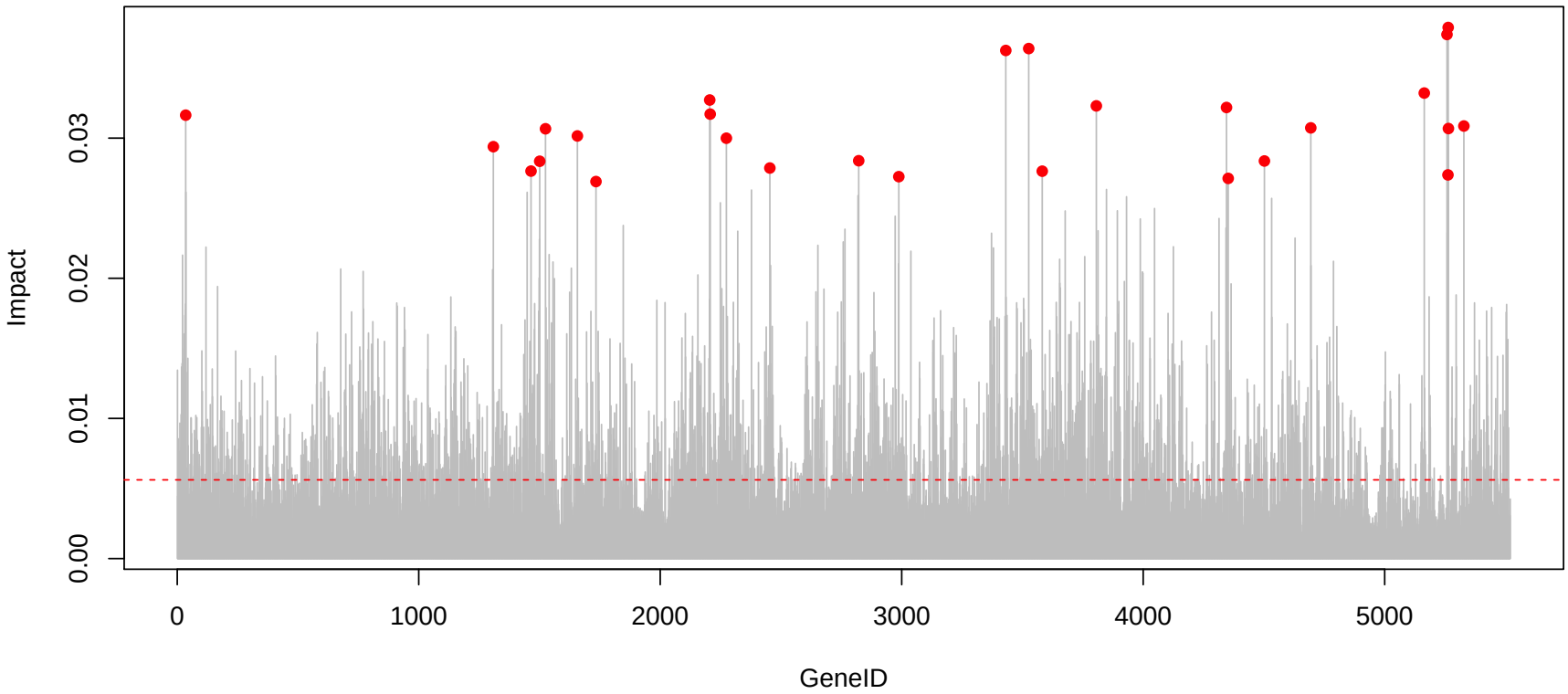
- Note: The CUR decomposition is not unique!

Example: Soft Tissue Tumour Dataset

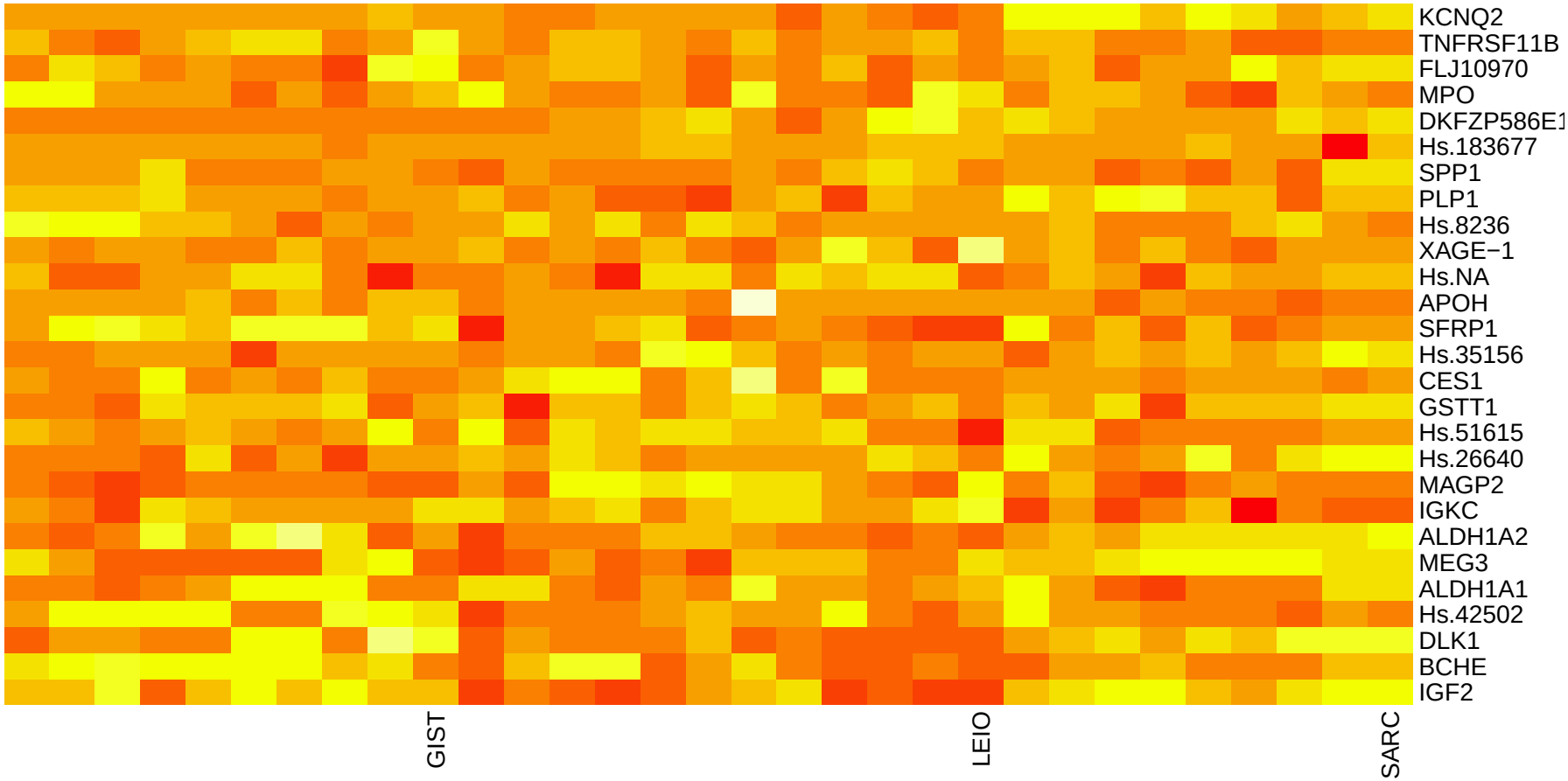


- The 'STTm' data are provided by the 'rCUR' package [5].

Which Genes are Important?

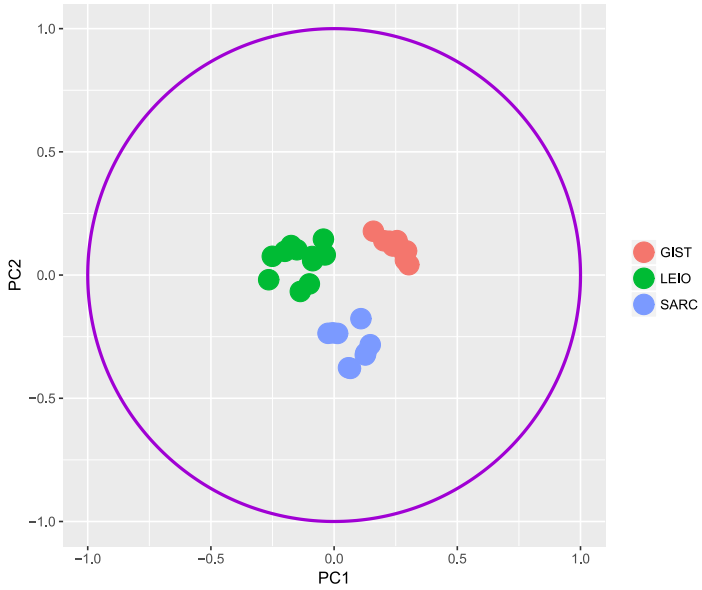


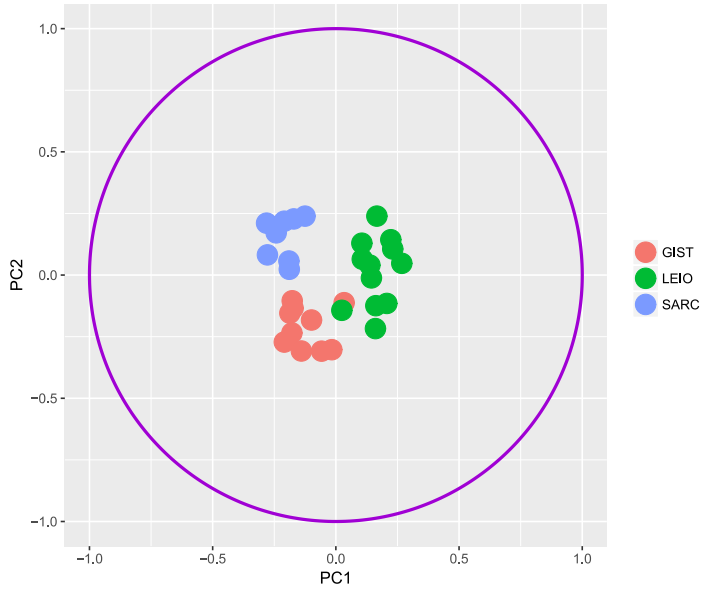
... 27 most influential Genes!



Enough to Discriminate Between Types of Tumor?

- Correlation plots, showing how the tumors are correlated with the first two principal components (PCs):





- Left plot shows the correlation with the PCs of the full data set.
- Right plot shows the correlation with the PCs of the 27 most influential genes.

Take Aways

- Randomized algorithms are a powerful tool to explore large datasets.
- Most applications do not require machine precision.
- Approximation quality can be controlled via
  - oversampling
  - power scheme
- Randomized routines are handy and easy to use.
- Save some time using the ‘rsvd’ package!

References

[1] Nathan Halko, Per-Gunnar Martinsson, Joel A. Tropp (2011).

Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions.

SIAM review 53(2), 217–288.

[2] Per-Gunnar Martinsson (2016).

Randomized methods for matrix computations and analysis of high dimensional data.

arXiv preprint arXiv:1607.01649.

[3] N. Benjamin Erichson, Voronin Sergey , Steven L. Brunton, and J. Nathan Kutz. (2016).

Randomized matrix decompositions using R.

arXiv preprint arXiv:1608.02148.

[4] Michael W. Mahoney (2011).

Randomized algorithms for matrices and data.

Foundations and Trends® in Machine Learning 3, no. 2 (2011): 123-224.

[5] András Bodor, Csabai István , Michael W. Mahoney, and Norbert Solymosi (2012).

rCUR: an R package for CUR matrix decomposition.

BMC Bioinformatics 13, no. 1 (2012): 103.

Randomized SVD algorithm

function $[\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}] = \texttt{rsvd}(\mathbf{A}, k, p, q)$

(1)	$l = k + p$	slight oversampling
(2)	$\mathbf{\Omega} = \texttt{rnorm}(n, l)$	generate sampling matrix
(3)	$\mathbf{Y} = \mathbf{A\Omega}$	draw range samples
(4)	**for** $j = 1, \ldots, q$	perform optional subspace iterations
(5)	$\quad[\mathbf{Q}, \sim] = \texttt{qr}(\mathbf{Y})$	
(6)	$\quad\mathbf{Z} = \mathbf{A}^\top\mathbf{Q}$	
(7)	$\quad[\mathbf{Q}, \sim] = \texttt{qr}(\mathbf{Z})$	
(8)	$\quad\mathbf{Y} = \mathbf{AQ}$	
(9)	**end for**	
(10)	$[\mathbf{Q}, \sim] = \texttt{qr}(\mathbf{Y})$	form orthonormal samples matrix
(11)	$\mathbf{B} = \mathbf{Q}^\top\mathbf{A}$	project to smaller space
(12)	$[\tilde{\mathbf{U}}, \mathbf{\Sigma}, \mathbf{V}] = \texttt{svd}(\mathbf{B})$	compact SVD of smaller matrix B
(13)	$\mathbf{U} = \mathbf{Q}\tilde{\mathbf{U}}$	recover left singular vectors
(14)	$\mathbf{U} = \mathbf{U}(:, 1:k)$	extract k components
(15)	$\mathbf{\Sigma} = \mathbf{\Sigma}(1:k, 1:k)$	extract k components
(16)	$\mathbf{V} = \mathbf{V}(:, 1:k)$	extract k components