NATIONAL TECHNICAL UNIVERSITY OF ATHENS

Randomized Matrix Decomposition Algorithms using Python

Author:
NIKOLAOS MATSAVELAS

Supervisor:
DR.P.PSARRAKOS ,
PROFESSOR N.T.U.A

A thesis submitted in fulfillment of the requirements
for the degree of MS.c

in the

Applied Mathematical Sciences
School of Applied Mathematical & Physical Sciences

October 3, 2021

Declaration of Authorship

NATIONAL TECHNICAL UNIVERSITY OF ATHENS

Abstract

Faculty Name
School of Applied Mathematical & Physical Sciences

MS.c

Randomized Matrix Decomposition Algorithms using Python

by Nikolaos Matsavelas

Primary goal of this thesis is to gain the knowledge of Singular Value Decomposition and Principal Component Analysis with Randomized Matrix Algorithms and compare their results (error and computational cost) for a much scientific and practical understanding of handling statistical data analysis.

With this thesis we explore the Randomized SVD in data analysis using examples in image compressing. We implement Randomized PCA for dimension reduction of variables, using the sophisticated method of Random Dense Matrices approach with different distributions.

In the 1^{st} chapter we introduce to the reader the theoretical background in the SVD methodology and the process of PCA calculation, as long as, with practical examples presented and produced in Python programming language.

In the 2^{nd} chapter we present the probability structure of random dense matrices, the distributional design of them, the probabilistic framework for low-rank approximations and the mathematical theory behind the generic randomized algorithm.

In the 3^{rd} chapter we dive in the Randomized SVD world, and we explore the all the scientific results related to this method. In addition we become more specific in each stage of the algorithm and we present alternative randomized algorithms for different matrices such as the "Single Pass Algorithm", the "Randomized Nystrom Method" for positive definite matrices and the Randomized Interpolative Decomposition.

In chapter 4^{th}, we introduce to the reader the idea of Randomized Principal Component Analysis for dimension reduction of big data set and the newly discovered Robust SVD which is a decomposition that separates a huge matrix into low rank matrix and a sparse matrix.

Finally we close this thesis in the 5^{th} chapter, presenting the pseudo spectra of the original matrices and the decomposed ones and finally we make a conclusion of the results of this thesis.

Acknowledgements

I want to thank my professor Dr. Panagiotis Psarrakos (supervisor of this thesis), for his precious and substantial scientific aid and guidance through out this thesis.

Contents

List of Figures

List of Tables

This thesis is dedicated to my mum and my sister

Chapter 1

Introduction to SVD and PCA

1.1 Singular Value Decomposition

Singular Value Decomposition, decomposes a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ of rank $\mathbf{r} = \mathbf{n}$ in $\mathbf{U\Sigma V^T}$:

$$\mathbf{A} = \underbrace{\begin{bmatrix} \vdots & \vdots & & \vdots \\ x_1 & x_2 & \dots & x_n \\ \vdots & \vdots & & \vdots \end{bmatrix}}_{m\times n} = \mathbf{U\Sigma V^T} =$$

$$\underbrace{\begin{bmatrix} \vdots & \vdots & & \vdots \\ u_1 & u_2 & \dots & u_m \\ \vdots & \vdots & & \vdots \end{bmatrix}}_{\mathbf{m\times m}} \underbrace{\begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_r \end{bmatrix}}_{\mathbf{r\times r}} \underbrace{\begin{bmatrix} \vdots & \vdots & & \vdots \\ v_1 & v_2 & \dots & v_n \\ \vdots & \vdots & & \vdots \end{bmatrix}^T}_{\mathbf{n\times n}},$$

where, $\mathbf{U}$ and $\mathbf{V}$ are orthonormal matrices of left and right eigenvectors respectively of matrices $\mathbf{AA^T}$ and $\mathbf{A^TA}$.

If $\mathbf{A} \in \mathbb{R}^{m\times n}$ matrix of rank $r > 0$, orthogonal $m \times m$ and $n \times n$ matrices $\mathbf{U}$ and $\mathbf{V}$ exist, such that $\mathbf{A} = \mathbf{U\Sigma V^T}$ and $\mathbf{\Sigma} = \mathbf{U^TAV}$, where the $m \times n$ matrix $\mathbf{\Sigma}$ is given by:

1. $\mathbf{\Sigma} \quad$ if $\quad \mathbf{r} = \mathbf{m} = \mathbf{n}$,
2. $\begin{bmatrix} \mathbf{\Sigma} & 0 \end{bmatrix} \quad$ if $\quad \mathbf{r} = \mathbf{m} < \mathbf{n}$,
3. $\begin{bmatrix} \mathbf{\Sigma} \\ 0 \end{bmatrix} \quad$ if $\quad \mathbf{r} = \mathbf{n} < \mathbf{m}$,
4. $\begin{bmatrix} \mathbf{\Sigma} & 0 \\ 0 & 0 \end{bmatrix} \quad$ if $\quad \mathbf{r} < \mathbf{m}, \mathbf{r} < \mathbf{n}$,

and Σ is an $r \times r$ diagonal matrix with positive diagonal elements. The diagonal elements of $\mathbf{\Sigma^2}$ are the positive eigenvalues of $\mathbf{A^TA}$ and $\mathbf{AA^T}$.

Proof. We will prove the result for the case $r < m$ and $r < n$. The proofs of (1)–(3) only require notational changes.

Let $\mathbf{\Sigma^2}$ be the $r \times r$ diagonal matrix whose diagonal elements are the r positive eigenvalues of $\mathbf{A^TA}$, which are identical to the positive eigenvalues of $\mathbf{AA^T}$ by the Theorem that :

Let $\mathbf{A} \in \mathbb{R}^{m\times n}$ matrix, with $rank(A) = r$. Then the positive eigenvalues of $\mathbf{A^TA}$ are equal to the positive eigenvalues of $\mathbf{AA^T}$.

We define $\mathbf{\Sigma}$ to be the diagonal matrix whose diagonal elements are the positive square roots of the corresponding diagonal elements of Σ^2 . Since $\mathbf{A^T A}$ is an $n \times n$ symmetric matrix, we can find an $n \times n$ orthogonal matrix $\mathbf{V}$, such that:

$$\mathbf{V^T A^T A V} = \begin{bmatrix} \Sigma^2 & 0 \\ 0 & 0 \end{bmatrix}.$$

Partitioning $\mathbf{V}$ as $\mathbf{V} = \begin{bmatrix} V_1 & V_2 \end{bmatrix}$, where V_1 is $n \times r$, the identity above implies that:

$$\mathbf{V_1^T A^T A V_1} = \mathbf{\Sigma^2}, \tag{1.1}$$

and

$$\mathbf{V_2^T A^T A V_2} = \mathbf{0}, \tag{1.2}$$

Note that from (1.2), it follows that :

$$\mathbf{A V_2} = \mathbf{0}. \tag{1.3}$$

Letting $\mathbf{U} = \begin{bmatrix} U_1 & U_2 \end{bmatrix}$ be an $m \times m$ orthogonal matrix, where the $m \times r$ matrix $U_1 = AV_1\Sigma^{-1}$ and the $m \times (m-r)$ matrix U_2 is any matrix that makes $\mathbf{U}$ orthogonal.

Consequently, we must have $\mathbf{U_2^T U_1} = \mathbf{U_2^T A V_1 \Sigma^{-1}} = \mathbf{0}$ or, equivalently:

$$\mathbf{U_2^T A V_1} = \mathbf{0}. \tag{1.4}$$

From the above equations we find that :

$$\begin{aligned}
\mathbf{U^T A V} &= \begin{bmatrix} \mathbf{U_1^T A V_1} & \mathbf{U_1^T A V_2} \\ \mathbf{U_2^T A V_1} & \mathbf{U_2^T A V_2} \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{\Sigma^{-1} V_1^T A V_1} & \mathbf{\Sigma^{-1} V_1^T A V_2} \\ \mathbf{U_2^T A V_1} & \mathbf{U_2^T A V_2} \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{\Sigma^{-1} \Sigma^2} & \mathbf{\Sigma^{-1} V_1^T A^T} \cdot 0 \\ 0 & \mathbf{U_2^T} \cdot 0 \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{\Sigma} & 0 \\ 0 & 0 \end{bmatrix}.
\end{aligned}$$

and so the proof is complete.

The singular value decomposition as given in the theorem above is for real matrices $\mathbf{A}$. However, this decomposition easily extends to complex matrices.

Specifically, an $m \times n$ complex matrix $\mathbf{A}$ can be expressed as $\mathbf{A} = \mathbf{U \Sigma V^\star}$, where the $m \times m$ and $n \times n$ matrices $\mathbf{U}$ and $\mathbf{V}$ are unitary, while $\mathbf{\Sigma}$ has the same form given in theorem with the diagonal elements of $\mathbf{\Sigma}$ given by the positive square roots of the nonzero eigenvalues of $\mathbf{A^\star A}$.

The diagonal elements of the $\mathbf{\Sigma}$ matrix given in theorem, that is, the positive square roots of the positive eigenvalues of $\mathbf{A^T A}$ and $\mathbf{A A^T}$, are called the singular values of $\mathbf{A}$. It is obvious from the proof of theorem that the columns of $\mathbf{V}$ form an orthonormal set of eigenvectors of $\mathbf{A^T A}$, and so :

$$\mathbf{A^T A} = \mathbf{V \Sigma^T \Sigma V^T}. \tag{1.5}$$

It is important to note also that the columns of U form an orthonormal set of eigenvectors of AA^T because :

$$\mathbf{A^T A} = \mathbf{U \Sigma V^T V \Sigma^T U^T} = \mathbf{U \Sigma \Sigma^T U^T}. \tag{1.6}$$

If we again partition $\mathbf{U}$ and $\mathbf{V}$ as $\mathbf{U} = \begin{bmatrix} U_1 & U_2 \end{bmatrix}$ and $V = \begin{bmatrix} V_1 & V_2 \end{bmatrix}$, where U_1 is $m \times r$ and V_1 is $n \times r$, then the singular value decomposition can be restated as follows. If $\mathbf{A}$ is an $m \times n$

matrix of rank $r > 0$, then $m \times r$ and $n \times r$ matrices U_1 and V_1 exist, such that

$$\mathbf{U_1^T U_1} = \mathbf{V_1^T V_1} = \mathbf{I}_r,$$

and $A = U_1 \Sigma V_1^T$, where Σ is an $r \times r$ diagonal matrix with positive diagonal elements. It follows from (1.5) and (1.6) that U_1 and V_1 are semiorthogonal matrices satisfying :

$$\mathbf{U_1^T A A^T V_1} = \mathbf{\Sigma^2}, \mathbf{V_1^T A^T A V_1} = \mathbf{\Sigma^2}. \tag{1.7}$$

However, in the decomposition $\mathbf{A} = \mathbf{U_1 \Sigma V_1^T}$, the choice of the semi-orthogonal matrix U_1 satisfying (1.7) is dependent on the choice of the V_1 matrix. This should be apparent from the proof of Theorem in which any semi-orthogonal matrix V_1 satisfying (1.7) was first selected, but the choice of U_1 was then given by:

$$\mathbf{U_1} = \mathbf{A V_1 \Sigma^{-1}}.$$

Alternatively, we could have first selected a semi-orthogonal matrix U_1 satisfying (1.7) and then have chosen $\mathbf{V_1} = \mathbf{A^T U_1 \Sigma^{-1}}$.

A lot of information about the structure of a matrix **A** can be obtained from its singular value decomposition. The number of singular values gives the rank of A, whereas the columns of U_1 and V_1 are orthonormal bases for the column space and row space of **A**, respectively. Similarly, the columns of U_2 span the null space of $\mathbf{A^T}$, and the columns of V_2 span the null space of **A**.

1.1.1 Orthonormality of eigenvectors

Given a real matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ with $m \geq n$, the singular value decomposition takes the form

$$\mathbf{A} = \mathbf{U \Sigma V^T}.$$

The matrices $\mathbf{U} = [u_1, \ldots, u_m \in \mathbb{R}^{m \times n}$ and $\mathbf{V} = \{v_1, \ldots, v_n\} \in \mathbb{R}^{n \times n}$ are orthonormal so that $\mathbf{U^T U} = \mathbf{I}$ and $\mathbf{V^T V} = \mathbf{I}$. The left singular vectors in **U** provide a basis for the range (column space), and the right singular vectors in **V** provide a basis for the domain (row space) of the matrix **A**.

Taking

$$\sigma_1 u_1 = A v_1, \sigma_2 u_2 = A v_2, ..., \sigma_r u_r = A v_r,$$

$\sigma_1, \sigma_2, ..., \sigma_r$ are positive scaling factors and $u_1, u_2, ..., u_r \in R^m$ are unit vectors in $C(A)$(column space of A).

Now, we have $A(v_1, v_2, ..., v_r) = (u_1, u_2, ..., u_r) \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_r \end{bmatrix}$.

And denote this formula as $\mathbf{AV} = \mathbf{U\Sigma}, \mathbf{V} \in R^{n \times r}, \mathbf{U} \in R^{m \times n}, \Sigma$ is a r×r ,when $r = m = n$, diagonal matrix as given previously.

Proof of orthogonality of $\mathbf{U^T U}$:

$$\begin{aligned}
\mathbf{u_i^T u_j} &= \frac{1}{\sigma_i\sigma_j}\sigma_i\sigma_j u_i^{\mathrm{T}} u_j \\
&= \frac{1}{\sigma_i\sigma_j}\sigma_i u_i^{\mathrm{T}}\sigma_j u_j \\
&= \frac{1}{\sigma_i\sigma_j}(\sigma_i u_i)^{\mathrm{T}}(\sigma_j u_j) \\
&= \frac{1}{\sigma_i\sigma_j}(Av_i)^{\mathrm{T}}(Av_j) \\
&= \frac{1}{\sigma_i\sigma_j}v_i^{\mathrm{T}}(A^{\mathrm{T}}A)v_j \\
&= \frac{1}{\sigma_i\sigma_j}v_i^{\mathrm{T}}\varepsilon_j v_j \\
&= \frac{1}{\sigma_i\sigma_j}\varepsilon_j v_i^{\mathrm{T}} v_j \\
&= \frac{1}{\sigma_i\sigma_j}\varepsilon_j 0 \\
&= 0,
\end{aligned}$$

where ε_j is the eigenvalue of $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ associated to v_j.

Example : We will find the singular value decomposition for the 4×3 matrix: Let A be a matrix,

$$A = \begin{bmatrix} 2 & 0 & 1 \\ 3 & -1 & 1 \\ -2 & 4 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

First, an eigenanalysis of the matrix:

$$A^T A = \begin{bmatrix} 18 & -10 & 4 \\ -10 & 18 & 4 \\ 4 & 4 & 4 \end{bmatrix},$$

reveals that it has eigenvalues $28, 12$, and 0 with associated normalized eigenvectors:

$$(1/\sqrt{2}, -1/\sqrt{2}, 0)^T, (1/\sqrt{3}, 1/\sqrt{3}, 1/\sqrt{3})^T,$$

and

$$(1/\sqrt{6}, 1/\sqrt{6}, -2/\sqrt{6})^T,$$

respectively.

Let these eigenvectors be the columns of the 3×3 orthogonal matrix $\mathbf{Q}$. Clearly, $\text{rank}(\mathbf{A}) = 2$ and the two singular values of $\mathbf{A}$ are $\sigma_1 = \sqrt{28}$ and $\sigma_2 = \sqrt{12}$. Thus, the 4×2 matrix U_1 is given by:

$$U_1 = AV_1\Sigma^{-1} = \begin{bmatrix} 2 & 0 & 1 \\ 3 & -1 & 1 \\ -2 & 4 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{3} \\ -1/\sqrt{2} & 1/\sqrt{3} \\ 0 & 1/\sqrt{3} \end{bmatrix} \times \begin{bmatrix} 1/\sqrt{28} & 0 \\ 0 & 1/\sqrt{12} \end{bmatrix}$$
$$= \begin{bmatrix} 1/\sqrt{14} & 1/2 \\ 2/\sqrt{14} & 1/2 \\ -3/\sqrt{14} & 1/2 \\ 0 & 1/2 \end{bmatrix}.$$

The 4×2 matrix U_2 can be any matrix satisfying $U_1^T U_2 = 0$ and $U_2^T U_2 = I_2$ for instance, $(1/\sqrt{12}, 1/\sqrt{12}, 1/\sqrt{12}, -3/\sqrt{12})$ and $(-5/\sqrt{42}, 4/\sqrt{42}, 1/\sqrt{42}, 0)^T$ can be chosen as the columns of U_2. Then our singular value decomposition of **A** is given by:

$$\begin{bmatrix} 1/\sqrt{14} & 1/2 & 1/\sqrt{12} & -5/\sqrt{42} \\ 2/\sqrt{14} & 1/2 & 1/\sqrt{12} & 4/\sqrt{42} \\ -3/\sqrt{14} & 1/2 & 1/\sqrt{12} & 1/\sqrt{42} \\ 0 & 1/2 & -3/\sqrt{12} & 0 \end{bmatrix} \begin{bmatrix} \sqrt{28} & 0 & 0 \\ 0 & \sqrt{12} & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} & 0 \\ 1/\sqrt{3} & 1/\sqrt{3} & 1/\sqrt{3} \\ 1/\sqrt{6} & 1/\sqrt{6} & -2/\sqrt{6} \end{bmatrix},$$

or in the form $U_1 \Sigma V_1^T$:

$$\begin{bmatrix} 1/\sqrt{14} & 1/2 \\ 2/\sqrt{14} & 1/2 \\ -3/\sqrt{14} & 1/2 \\ 0 & 1/2 \end{bmatrix} \begin{bmatrix} \sqrt{28} & 0 \\ 0 & \sqrt{12} \end{bmatrix} \times \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} & 0 \\ 1/\sqrt{3} & 1/\sqrt{3} & 1/\sqrt{3} \end{bmatrix}.$$

As an alternative way of determining the matrix **U** , we could have used the fact that its columns are eigenvectors of the matrix:

$$AA^T = \begin{bmatrix} 5 & 7 & -3 & 3 \\ 7 & 11 & -9 & 3 \\ -3 & -9 & -21 & 3 \\ 3 & 3 & 3 & 3 \end{bmatrix}.$$

However, when constructing **U** this way, one must check the decomposition $\mathbf{A} = U_1 \Sigma V_1^T$ to determine the correct sign for each of the columns of U_1.

1.1.2 Vector SV decomposition

Let x be an $m \times 1$ non null vector. Its singular value decomposition will be of the form

$$\mathbf{x} = \mathbf{Usv},$$

where , U is an $m \times m$ orthogonal matrix, σ is an $m \times 1$ vector having only its first component nonzero, and v is a scalar satisfying $v^2 = 1$.

The single singular value of x is given by λ, where $\lambda^2 = x^T x$. If we define $x_\star = \lambda^{-1} x$, note that $x_\star^T x_\star = 1$, and

$$xx^T x_\star = xx^T(\lambda^{-1}x) = (\lambda^{-1}x)x^T x = \lambda^2 x_\star,$$

so that $x_\star$ is a normalized eigenvector of xx^T corresponding to its single positive eigenvalue λ^2.

Any non null vector orthogonal to $x_\star$ is an eigenvector of xx^T corresponding to the repeated eigenvalue 0. Thus, if we let $s = (\lambda, 0..., 0)^T, v = 1$, and $U = (x_\star, u_2, \ldots, u_m)$, be any orthogonal matrix with $x_\star$ as its first column, then we have :

$$\mathbf{Usv} = \begin{bmatrix} x_\star & u_2 & \dots & u_m \end{bmatrix} \begin{bmatrix} \lambda \\ 0 \\ \vdots \\ 0 \end{bmatrix} \times 1 = \lambda x_\star = x.$$

For example consider the vector : $x = (9, 10, 2, 7, 1, 3, 4, 6, 8, 5)$ the Usv is then :

$$\mathbf{Usv} = \begin{bmatrix} 0.458 & 0.509 & \dots & 02548 \end{bmatrix} \begin{bmatrix} 19.62 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \times 1.$$

When $\mathbf{A}$ is $m \times m$ and symmetric, the singular values of A are directly related to the eigenvalues of $\mathbf{A}$. This follows from the fact that $\mathbf{AA^T} = \mathbf{A^2}$, and the eigenvalues of $\mathbf{A^2}$ are the squares of the eigenvalues of $\mathbf{A}$.

Thus, the singular values of $\mathbf{A}$ will be given by the absolute values of the eigenvalues of $\mathbf{A}$. If we let the columns of $\mathbf{U}$ be a set of orthonormal eigenvectors of $\mathbf{A}$, then the $\mathbf{V}$ matrix in 1.1 will be identical to $\mathbf{U}$ except that any column of $\mathbf{V}$ that is associated with a negative eigenvalue will be -1 times the corresponding column of $\mathbf{U}$.

If $\mathbf{A}$ is nonnegative definite, then the singular values of $\mathbf{A}$ will be the same as the positive eigenvalues of $\mathbf{A}$ and, in fact, the singular value decomposition of $\mathbf{A}$ is simply the spectral decomposition of $\mathbf{A}$ which is :

$$\mathbf{A} = \mathbf{X\Lambda X^T}$$

that will be discussed in the folllowing subchapters for the Principal Components Analysis method. Surely this nice relationship between the eigenvalues and singular values of a symmetric matrix does not carry over to general square matrices.

Concluding example: Consider the 2×2 matrix :

$$A = \begin{bmatrix} 6 & 6 \\ -1 & 1 \end{bmatrix},$$

which has :

$$AA^T = \begin{bmatrix} 72 & 0 \\ 0 & 2 \end{bmatrix},$$

and

$$A^T A = \begin{bmatrix} 37 & 35 \\ 35 & 37 \end{bmatrix}.$$

As the reader can notice in the Apppendix, the Python programming language in the eigen analysis of a matrix, the eigenvectors are being normalized with the vector length

$$\|e\| = \sqrt{e^T e},$$

of the eigenvector. Back to our example, the singular values of $\mathbf{A}$ are $72 = 6\sqrt{2}$ and 2. The normalized eigenvectors corresponding to 72 and 2 are $(-1, 0)^T$ and $(0, 1)^T$ for $\mathbf{AA^T}$, whereas $\mathbf{A^T A}$ has $(-1/\sqrt{2}, -1/\sqrt{2})^T$ and $(-1/\sqrt{2}, 1/\sqrt{2})^T$. Thus, the singular value decomposition of $\mathbf{A}$ can be written as:

$$\mathbf{A} = \mathbf{U\Sigma V^T} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 6\sqrt{2} & 0 \\ 0 & \sqrt{2} \end{bmatrix} \begin{bmatrix} -1/\sqrt{2} & -1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}^T .$$

1.1.3 Low rank approximation of SVD

As the title reveals, the low rank approximation is a method that we approach the decomposition of a matrix with another matrix of lower rank.This method we have already seen it in previous subchapters when we have partitioned the $\mathbf{U\Sigma V^T}$.

Specifically, the rectangular diagonal matrix $\mathbf{\Sigma} \in \mathbb{R}^{m\times n}$ contains the corresponding non-negative singular values $\sigma_1 \geq \cdots \geq \sigma_n \geq 0$, describing the spectrum of the data. The so called “economy” or “thin” SVD computes only the left singular vectors and singular values corresponding to the number (i.e., n) of right singular vectors

$$\mathbf{A} = \mathbf{U\Sigma V^T} = [\mathbf{u}_1, \ldots, \mathbf{u}_n]\text{diag}(\sigma_1, \ldots, \sigma_n)[\mathbf{v}_1, \ldots, \mathbf{v}_n]^T.$$

If the number of right singular vectors is small (i.e., $n \ll m$), this is a more compact factorization than the full SVD. The “economy” SVD is the default form of the base svd() function in R. Low-rank matrices feature a rank that is smaller than the dimension of the ambient measurement space, i.e., r is smaller than the number of columns and rows. Hence, the singular values $\{\sigma_i : i \geq r+1\}$ are zero, and the corresponding singular vectors span the left and right null spaces. The concept of the “economy” SVD of a low-rank matrix is illustrated in Figure 1:

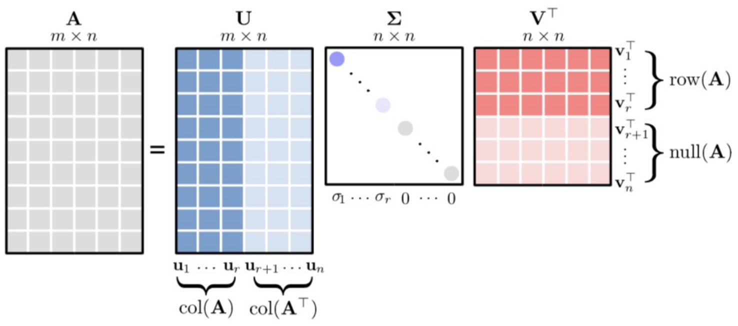


FIGURE 1.1: SVD Low Rank Decomposition

In practical applications matrices are often contaminated by errors, and the effective rank of a matrix can be smaller than its exact rank r. In this case, the matrix can be well approximated by including only those singular vectors which correspond to singular values of a significant magnitude.

Hence, it is often desirable to compute a reduced version of the SVD

$$\mathbf{A_k} := \mathbf{U}_k\Sigma_k\mathbf{V}_k = [\mathbf{u}_1, \ldots, \mathbf{u}_k]\text{diag}(\sigma_1, \ldots, \sigma_k)[\mathbf{v}_1, \ldots, \mathbf{v}_k]^T,$$

where k denotes the desired target rank of the approximation. In other words, this reduced form of the SVD allows one to express A approximately by the sum of k rank-one matrices

$$\mathbf{A}_k \approx \sum_{i=1}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^T.$$

Choosing an optimal target rank k is highly dependent on the task. One can either be interested in a highly accurate reconstruction of the original data, or in a very low dimensional representation of dominant features in the data. In the former case k should be chosen close to the effective rank, while in the latter case k might be chosen to be much smaller.

Truncating small singular values in the deterministic **SVD** gives an optimal approximation of the corresponding target rank k. Specifically, the Eckart-Young theorem (1936), states that the low-rank **SVD** provides the optimal rank-k reconstruction of a matrix in the least-square sense:

$$\mathbf{A}_{(k)} = \underset{rank(A'_k)=k}{\arg\min} \|\mathbf{A} - \mathbf{A'_k}\|,$$

The reconstruction error in both the spectral and Frobenius norms is given by:

$$\|\mathbf{A} - \mathbf{A}_k\|_2 = \sigma_{k+1}(\mathbf{A}) \quad \text{and} \quad \|\mathbf{A} - \mathbf{A}_k\|_F = \sqrt{\sum_{j=k+1}^{\min(m,n)} \sigma_j^2(\mathbf{A})}.$$

For massive datasets, however, the truncated SVD is costly to compute. The cost to compute the full SVD of an $m \times n$ matrix is of the order $\mathcal{O}(mn^2)$, from which the first k components can then be extracted to form $\mathbf{A}_k$.

1.2 Principal Component Analysis

Principal Component Analysis (PCA) has been called one of the most valuable results from applied linear algebra. PCA is used abundantly in all forms of analysis from neuroscience to computer graphics because it is a simple, non-parametric method of extracting relevant information from confusing data sets.

With minimal additional effort PCA provides a roadmap for how to reduce a complex data set to a lower dimension to reveal the sometimes hidden, simplified dynamics that often underlie it.

We define a new matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$ as :

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix} = (x_{ij}) \in \mathbb{R}^{m \times n}.$$

With this assumption PCA is now limited to re- expressing the data as a linear combination of its basis vectors. Let $\mathbf{X}, \mathbf{Y} \in \mathbb{R}^{m \times n}$ matrices related by a linear transformation $\mathbf{P}$. $\mathbf{X}$ is the original recorded data set and $\mathbf{Y}$ is a re-representation of that data set.

$$\mathbf{PX} = \mathbf{Y}. \tag{1.8}$$

Also we define the following quantities:

- p_i are the rows of $\mathbf{P}$,
- x_i are the columns of $\mathbf{X}$,

- y_i are the columns of **Y**.

Equation 1.8 represents a change of basis and thus can have many interpretations:

1. **P** is a matrix that transforms **X** into **Y**,
2. Geometrically, **P** is a rotation and a stretch which again transforms **X** into **Y**,
3. The rows of **P**, $p_1, P_2, \ldots, p_m$, are a set of new basis vectors for expressing the columns of **X**.

The latter interpretation is not obvious but can be seen by writing out the explicit dot products of **PX**:

$$\mathbf{PX} = \begin{bmatrix} p_1 \\ \vdots \\ p_m \end{bmatrix} \begin{bmatrix} x_1 & \ldots & x_n \end{bmatrix},$$
$$\mathbf{Y} = \begin{bmatrix} p_1 \cdot x_1 & \ldots & p_1 \cdot x_n \\ \vdots & \ddots & \vdots \\ p_m \cdot x_1 & \ldots & p_m \cdot x_n \end{bmatrix}.$$

We can note the form of each column of **Y**:

$$\mathbf{y_i} = \begin{bmatrix} p_1 \cdot x_1 \\ \vdots \\ p_m \cdot x_1 \end{bmatrix}.$$

We recognize that each coefficient of $\mathbf{y_i}$ is a dot product of $\mathbf{x_i}$ with the corresponding row in **P**. In other words, the j^{th} coefficient of $\mathbf{y_i}$ is a projection on to the j^{th} row of **P**. This is in fact the very form of an equation where $\mathbf{y_i}$ is a projection on to the basis of

$$\{\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_m}\}.$$

Therefore, the rows of **P** are indeed a new set of basis vectors for representing of columns of **X**.

One interpretation of the information matrix **X** is the following:

- Each row n of **X** corresponds to all sample measurements (x_i).
- Each column of **X** corresponds to a set of measurements from one particular variables m

We now arrive at a definition for the covariance matrix $\mathbf{S_x}$:

$$\mathbf{S_x} = \frac{1}{n-1}\mathbf{X^T X}.$$

Consider how the matrix form $\mathbf{XX^T}$, in that order, computes the desired value for the ij^{th} element of $\mathbf{S_X}$. Specifically, the ij^{th} element of the variance is the dot product between the vector of the i^{th} measurement type with the vector of the j^{th} measurement type.

- $\mathbf{S_X}$ is a square symmetric $m \times m$ matrix.
- The diagonal terms of $\mathbf{S_X}$ are the variance of each variable.

- The off-diagonal terms of $\mathbf{S_X}$ are the covariance of variables between measurement types.

Computing $\mathbf{S_X}$ quantifies the correlations between all possible pairs of measurements. $\mathbf{S_X}$ is special.

The covariance matrix describes all relationships between pairs of measurements in our data set. Pretend we have the option of manipulating $\mathbf{S_X}$. We will suggestively define our manipulated covariance matrix $\mathbf{S_Y}$. What features do we want to optimize in $\mathbf{S_Y}$?

1.2.1 Diagonalize the Covariance Matrix

If our goal is to reduce redundancy, then we would like each variable to covary as little as possible with other variables. More precisely, to remove redundancy we would like the covariances between separate measurements to be zero.

What would the optimized covariance matrix $\mathbf{S_Y}$ look like? Evidently, in an "optimized" matrix all off-diagonal terms in $\mathbf{S_Y}$ are zero. Therefore, removing redundancy diagnolizes $\mathbf{S_Y}$.

There are many methods for diagonalizing $\mathbf{S_Y}$ It is curious to note that PCA arguably selects the easiest method, perhaps accounting for its widespread application.

PCA assumes that all basis vectors

$$\mathbf{p_1}, \mathbf{p_2}, \cdots \mathbf{p_m},$$

are orthonormal , i.e. :

$$\mathbf{p_i} \cdot \mathbf{p_j} = \delta_{ij}.$$

In the language of linear algebra, PCA assumes $\mathbf{P}$ is an orthonormal matrix.

Secondly PCA assumes the directions with the largest variances are the most "important" or in other words, most principal. Why are these assumptions easiest? Envision how PCA might work.

By the variance assumption PCA first selects a normalized direction in m-dimensional space along which the variance in $\mathbf{X}$ is maximized it saves this as p_1. Again it finds another direction along which variance is maximized, however, because of the orthonormality condition, it restricts its search to all directions perpindicular to all previous selected directions.

This could continue until m directions are selected. The resulting ordered set of $\boldsymbol{p}$'s are the principal components. In principle this simple pseudo-algorithm works, however that would bely the true reason why the orthonormality assumption is particularly judicious.

Namely, the true benefit to this assumption is that it makes the solution amenable to linear algebra. There exist decompositions that can provide efficient, explicit algebraic solutions. Notice what we also gained with the second assumption. We have a method for judging the "importance" of the each principal direction. Namely, the variances associated with each direction $\boldsymbol{p_i}$ quantify how principal each direction is.

We could thus rank-order each basis vector $\boldsymbol{p_i}$ according to the corresponding variances. We will now pause to review the implications of all the assumptions made to arrive at this mathematical goal.

1.2.2 Eigenvectors of Covariance Matrix

We will derive our first algebraic solution to PCA using linear algebra. This solution is based on an important property of eigenvector decomposition. Once again, the data set is

$\mathbf{X} \in \mathbb{R}^{m\times n}$ matrix, where m is the number of measurement types and n is the number of data trials.

We are in the position to find some orthonormal matrix $\mathbf{P}$ where $\mathbf{Y} = \mathbf{PX}$ such that

$$\mathbf{S_Y} = \frac{1}{n-1}\mathbf{YY^T},$$

is diagonalized. The rows of $\mathbf{P}$ are the principal components of $\mathbf{X}$. We begin by rewriting $\mathbf{S_Y}$ in terms of our variable of choice $\mathbf{P}$:

$$\begin{aligned}
\mathbf{S_Y} &= \frac{1}{n-1}\mathbf{YY^T} \\
&= \frac{1}{n-1}(\mathbf{PX})(\mathbf{PX})^\mathbf{T} \\
&= \frac{1}{n-1}\mathbf{PXX^TP^T} \\
&= \frac{1}{n-1}\mathbf{P}(\mathbf{XX^T})\mathbf{P^T} \\
&= \frac{1}{n-1}\mathbf{PAP^T}.
\end{aligned}$$

Note that we defined a new matrix $\mathbf{A} = \mathbf{XX^T}$, where $\mathbf{A}$ is symmetric and orthogonal ,i.e $\mathbf{A^T} = \mathbf{A^{-1}}$.

Our roadmap is to recognize that a symmetric matrix $\mathbf{A}$ is diagonalized by an orthogonal matrix of its eigenvectors. For a symmetric matrix $\mathbf{A}$ we know that can be diagonalized as:

$$\mathbf{A} = \mathbf{EDE^T}, \tag{1.9}$$

where , $\mathbf{D}$ is a diagonal matrix and $\mathbf{E}$ is a matrix of eigenvectors of $\mathbf{A}$ arranged as columns. The matrix $\mathbf{A}$ has $r \leq m$ orthonormal eigenvectors where r is the rank of the matrix.

The rank of $\mathbf{A}$ is less than m when $\mathbf{A}$ is degenerate or all data occupy a subspace of dimension $r \leq m$. Maintaining the constraint of orthogonality, we can remedy this situation by selecting $(m - r)$ additional orthonormal vectors to fill up the matrix $\mathbf{E}$. These additional vectors do not effect the final solution because the variances associated with these directions are zero. Now comes the trick. We select the matrix $\mathbf{P}$ to be a matrix where each row p_i is an eigenvector of $\mathbf{XX^T}$. By this selection:

$$\mathbf{P} = \mathbf{ET}.$$

Substituting into Equation 1.9, we find:

$$\mathbf{A} = \mathbf{PT} \cdot \mathbf{DP},$$

and we can finish evaluating $\mathbf{S_Y}$ as :

$$\begin{aligned}\mathbf{S_Y} &= \frac{1}{n-1}\mathbf{PAP^T} \\ &= \frac{1}{n-1}\mathbf{P}(\mathbf{P^T DP})\mathbf{P^T} \\ &= \frac{1}{n-1}(\mathbf{PP^T})\mathbf{D}(\mathbf{PP^T}) \\ &= \frac{1}{n-1}(\mathbf{PP^{-1}})\mathbf{D}(\mathbf{PP^{-1}}) \\ &= \frac{1}{n-1}\mathbf{D}.\end{aligned}$$

It is evident that the choice of $\mathbf{P}$ diagonalizes $\mathbf{S_Y}$. This was the goal for PCA. We can summarize the results of PCA in the matrices $\mathbf{P}$ and $\mathbf{S_Y}$.

- The principal components of $\mathbf{X}$ are the eigenvectors of $\mathbf{XX^T}$ or the rows of $\mathbf{P}$.
- The i^{th} diagonal value of $\mathbf{S_Y}$ is the variance of $\mathbf{X}$ along p_i.

In practice computing PCA of a data set $\mathbf{X}$ we subtract off the mean of each measurement type (centered data) and then computing the eigenvectors of $\mathbf{XX^T}$. In statistics we often work with centered data. The reason is that the data matrix $\mathbf{X}$ might has different units like inched and pounds and years and dollars in each column (variable).

Changing one set of units would have a big impact on that row of $\mathbf{A}$ and $\mathbf{S_X}$. If scaling is a problem, we change from covariance matrix $\mathbf{S_X}$ to correlation matrix $\mathbf{C}$. The diagonal matrix $\mathbf{D}$ rescales matrix $\mathbf{A}$.Each row of $\mathbf{DA}$ has length $\sqrt{n-1}$.

The sample correlation matrix:

$$\mathbf{C} = \mathbf{DAA^TD/(n-1)},$$

has $1'$s on its diagonal and the off diagonal elements represent the correlation between the associated columns (variables).

1.3 SVD and PCA

With similar computations it is evident that the two methods are intimately related. Let us return to the original data information matrix $\mathbf{X} \in \mathbb{R}^{m\times n}$. We can define a new matrix $\mathbf{Y} \in \mathbb{R}^{n\times m}$:

$$\mathbf{Y} = \frac{1}{\sqrt{n-1}}\mathbf{X^T},$$

where , each column of $\mathbf{Y}$ has zero mean. The definition of $\mathbf{Y}$ becomes clear by analyzing $\mathbf{Y^TY}$:

$$\begin{aligned}
\mathbf{Y^TY} &= \left(\frac{1}{\sqrt{n-1}}\mathbf{X^T}\right)^T \left(\frac{1}{\sqrt{n-1}}\mathbf{X^T}\right) \\
&= \frac{1}{(\sqrt{n-1})^2}\mathbf{X^{TT}X^T} \\
&= \frac{1}{n-1}\mathbf{X^{TT}X^T} \\
&= \frac{1}{n-1}\mathbf{XX^T} \\
&= \mathbf{S_X}.
\end{aligned}$$

By construction $\mathbf{Y^TY}$ equals the covariance matrix of $\mathbf{X}$. From the previous sub-chapter 1.6 we know that the principal components of $\mathbf{X}$ are the eigenvectors of $\mathbf{S}$. If we $\mathbf{X}$ calculate the SVD of $\mathbf{Y}$, the columns of matrix $\mathbf{V}$ contain the eigenvectors of

$$\mathbf{Y^TY} = \mathbf{S_X}.$$

Therefore, the columns of V are the principal components of $\mathbf{X}$.
What does this mean? $\mathbf{V}$ spans the row space of :

$$\mathbf{Y} = \frac{1}{\sqrt{n-1}}\mathbf{X^T}.$$

Therefore, $\mathbf{V}$ must also span the column space of :

$$\frac{1}{\sqrt{n-1}}\mathbf{X}.$$

1.3.1 The logic behind PCA method

Correlation in multivariate data is very common. One of the consequences of having a correlation in the data is the repetition of the information collected. For example in the extreme case where 2 variables are perfectly correlated with each other, one of them is completely unnecessary information.

Based on the correlation, it will be possible knowing one variable to calculate exactly the second. The Principal Components Analysis (PCA) method eliminates unnecessary information in the data.

If $\mathbf{S_X}$ is the variance-covariance table for the data and $\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ its eigenvalues, $\{v_1, v_2, \ldots, v_n\}$ the corresponding eigenvectors, then based on the interpretation of eigenvalues and eigenvectors for a table could transform the data in the system of axes defined by eigenvectors to achieve exactly the goal of PCA i.e. not to repeat the same information many times.

For example : for $n = m = 2$ the variance - covariance matrix is :

$$\mathbf{S} = \begin{bmatrix} \sigma_x^2 & \varrho_{xy}\sigma_x\sigma_y \\ \varrho_{xy}\sigma_x\sigma_y & \sigma_y^2, \end{bmatrix},$$

is an ellipse in the 2 dimensions with axis the directions of the eigenvectors and the length of the diagonals $1/\sqrt{\lambda_i}$. The eigen-analysis of the matrix in eigenvalues and eigenvectors for different values of the degree of correlation ϱ has the following forms:

Some useful points about the figures below :

- The higher the ϱ , the smaller the variability along an axis.

- Similarly, the larger the ϱ , the more the variability of the points is limited along a single axis.
- The "information" contained in the data can be extracted from the variability of the points along the v_1 axis with almost no information loss.

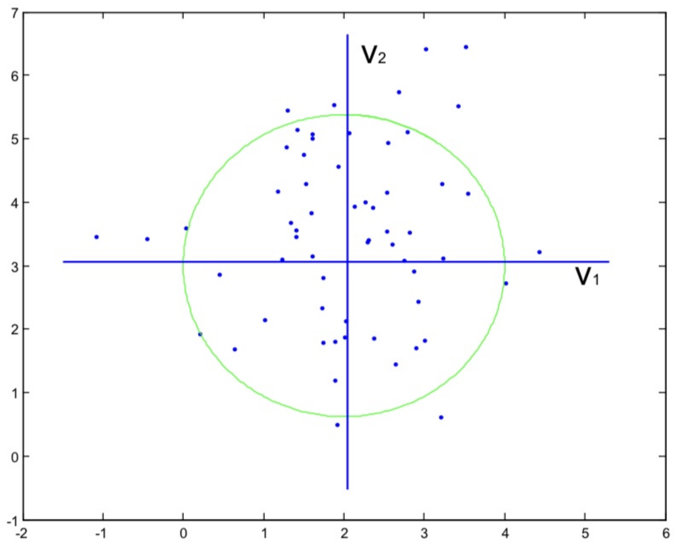


FIGURE 1.2: Eigenvectors v_1 and v_2 in the $x'x, y'y$ axis, with correlation : $\varrho = 0$

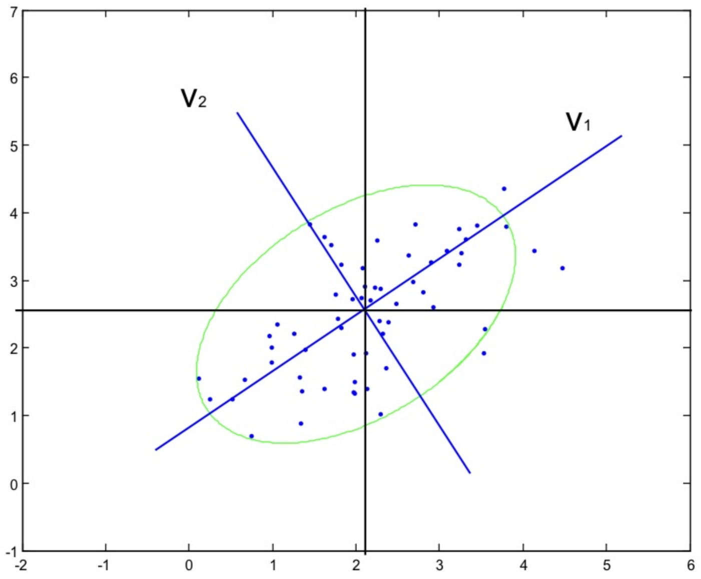


FIGURE 1.3: Eigenvectors v_1 and v_2 in the $x'x, y'y$ axis, with correlation : $\varrho = 0.75$

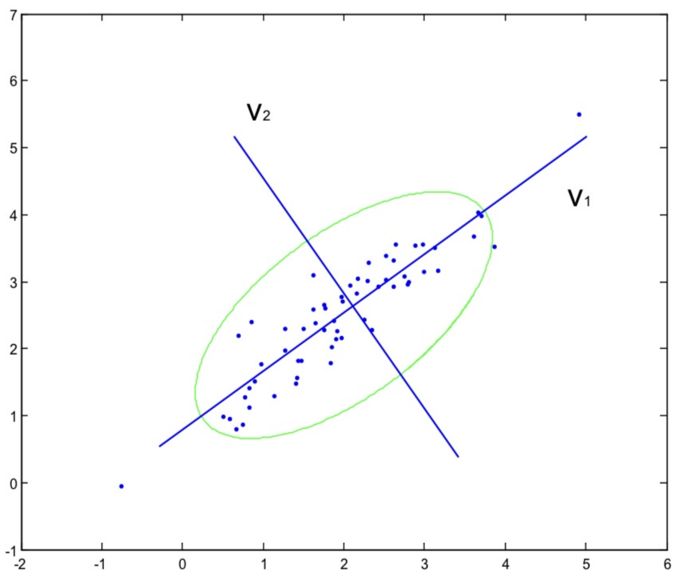


FIGURE 1.4: Eigenvectors v_1 and v_2 in the $x'x, y'y$ axis, with correlation : $\varrho = 0.9$

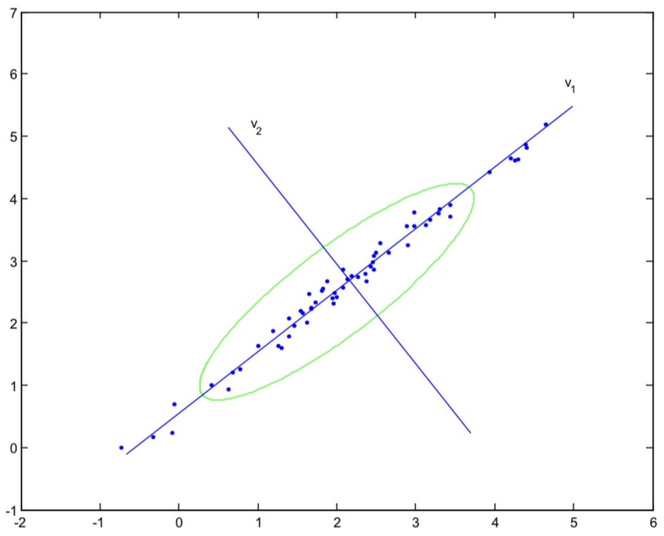


FIGURE 1.5: Eigenvectors v_1 and v_2 in the $x'x, y'y$ axis, with correlation : $\varrho = 1$ even more intense than the previous

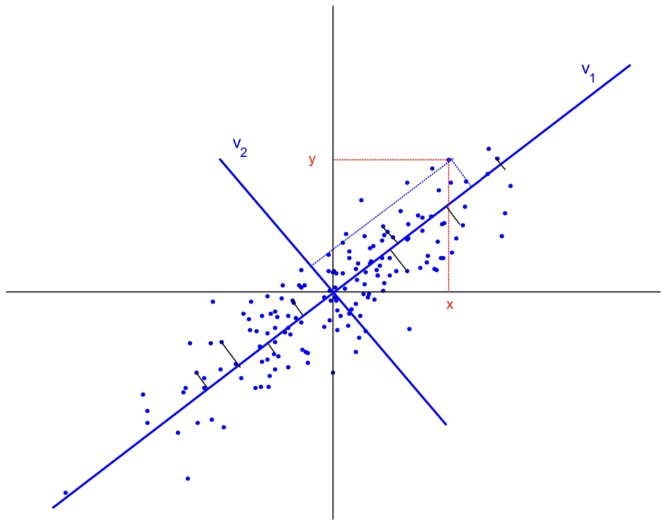


FIGURE 1.6: Eigenvectors v_1 and v_2 in the $x'x, y'y$ axis, with correlation : $\varrho = 0.85$

Scope of PCA :

1. Using PCA we produce an equivalent to the original data set. The difference is that the new variables are not correlated with each other and it is possible to select a smaller number of variables for any future analysis without significant information loss.
2. The new variables calculated by PCA are called Principal Components, PC.
3. "Principal" is in the sense of hierarchy. The 1st PC is more important (contains more information) than the 2nd, the 2nd PC is more important than the 3rd, the 3rd is more important than the 4th, etc.

1.3.2 How we calculate the Principal Components

The result is identified with a well-known mathematical technique (diagonalization). Let $x = (x_1, x_2, x_3, \ldots, x_n)^T$ be the original correlated variables. The measurements even if they were performed on n objects and led to an $m \times n$ data table.

If **S** is the variance-covariance table of the data, then the PC's $y_1, y_2, y_3, \ldots, y_n$ can be shown to be given by the relations:

$$\begin{aligned} y_1 &= a_1 \boldsymbol{x} \\ y_2 &= a_2 \boldsymbol{x} \\ &\cdots \\ y_n &= a_n \boldsymbol{x}, \end{aligned}$$

where , a_1 is the eigenvector corresponding to the 1st order of magnitude eigenvalue of the table **S**, a_2 the second, etc.

Properties of PC's

1. PC's are linear combinations of $\boldsymbol{x}$'s.

$$\begin{aligned} y_1 &= a_1 \cdot x \\ &= (a_{11} a_{12} \ldots a_{1n}) \cdot (x_1 x_2 \ldots x_n)^T \\ &= a_{11} x_1 + a_{12} x_2 + \cdots + a_{1n} x_n. \end{aligned}$$

2. They are unrelated random variables (by definition, from the way they are defined). They are rectangular random vectors.
3. $\boldsymbol{Var}(\sum x_i) = \boldsymbol{Var}(\sum y_i)$. PC's explain the same overall variability as the original data (no loss of information if we keep all PCA components).
4. $\boldsymbol{Var}(\boldsymbol{y_1}) > \boldsymbol{Var}(\boldsymbol{y_2}) > \cdots > \boldsymbol{Var}(\boldsymbol{y_n})$.

PC's can be calculated in either table **S** or the ϱ (correlation table). We choose to use with the correlation matrix **C** when the variations are quite different for the original variables (one reason may be the different units of measurement) as we previously have noted.

Using table ϱ we give the same weight to all the initial variables. The importance of the PCA component is calculated from the percentage of variability it explains. More detail: Importance j component:

$$\frac{\boldsymbol{Var}(\boldsymbol{y_i})}{\sum_{i=1}^{n} \boldsymbol{Var}(\boldsymbol{y_i})},$$

and the importance of $j = 1, \ldots, k, (k \leq n)$ components :

$$\frac{\sum_{j=1}^{k} \boldsymbol{Var}(\boldsymbol{y_i})}{\sum_{i=1}^{n} \boldsymbol{Var}(\boldsymbol{y_i})}.$$

When PC's are calculated on table **S**, then $\boldsymbol{Var}(\boldsymbol{y_i}) = \lambda_i$.

1. When PC's are calculated on the correlation matrix **C** then $\boldsymbol{Var}(y_i) \neq \lambda_i$. The variability then equals: $\frac{\lambda_i}{\varrho}$.
2. The results in terms of the eigenvalues of **S** and therefore in terms of the main components depend on the units of measurement.
3. Also if one of the variables has a much larger variation than the others, then it greatly affects the principal components (imposed on the rest of x_i). For this reason we usually work with correlation matrix **C**, unless all the variables have the same unit of measurement.

4. The PCA technique, in addition to dimension reduction, is also used to investigate the linearity of the data (as many linear relationships as eigenvalues are very close to zero).

The first k components are retained, where k: explain “sufficient” percentage of total variability which is defined by :

$$\frac{\sum_{j=1}^{k} \boldsymbol{Var(y_i)}}{\sum_{i=1}^{n} \boldsymbol{Var(y_i)}} \cdot 100\%.$$

A useful plot that works as a metric of how many principal components to retain is the scree plot which is graph of the number of components in terms of the percentage of variability cumulatively that uses the Rule of "angle".

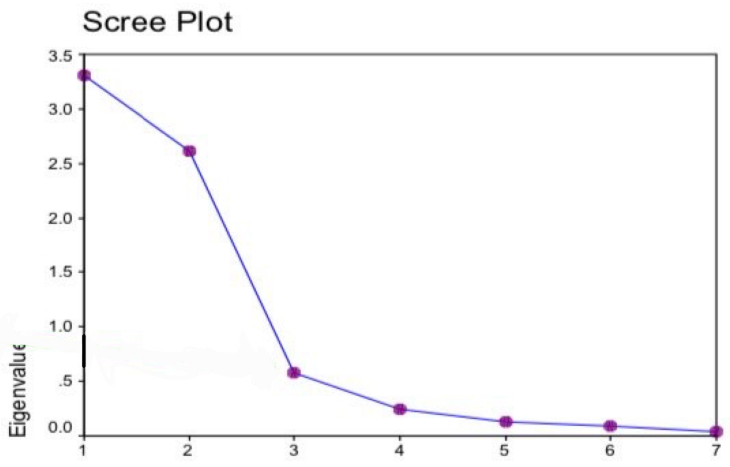


FIGURE 1.7: The angle in the 3rd pc reveals that the 1st and 2nd principal components explain sufficiently the variability of the data set

When PCAs are based on **C** , we can alternatively hold those components that correspond to an eigenvalue greater than 1. For every $i = 1, 2, \ldots, n$,we have :

$$y_i = a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n,$$

in PC y_i contribute those of the initial x_i that have in the above expression a corresponding factor relatively large. One PC can be the overall representation of one group of initial variables or even the representation of one group as opposed to another.

It has already been shown that :

$$y_i = a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n, \quad i = 1, 2, \ldots, n. \tag{1.10}$$

For the object j with initial data: $(x_{j1}, x_{j2}, \ldots, x_{jn})$ the corresponding vector in the new set of variables (scores) will be:

$$\underbrace{(a_{11}x_{j1} + a_{12}x_{j2} + \cdots + a_{1n}x_{jn}}_{Score\quad on\quad 1st\quad PC\quad i=1\quad in\quad 1.10}, \underbrace{a_{21}x_{j1} + a_{22}x_{j2} + \cdots + a_{2n}x_{jn}, \ldots,}_{Score\quad on\quad 2nd\quad PC\quad i=2\quad in\quad 1.10} \underbrace{a_{n1}x_{j1} + a_{n2}x_{j2} + \cdots + a_{nn}x_{jn})}_{Score\quad on\quad n-th\quad PC\quad i=n\quad in\quad 1.10}.$$

1.3.3 PCA Example

The data are values from 5 economic and demographic indicators for 25 countries. They refer to data from 1990 and the 5 variables are:

1. Increase: The annual rate of population growth.
2. Life: The expected average life.

3. Infant mortality rate (IMR): the infant mortality rate per 1000.
4. Total Fertinity Rate (TRF): Prosperity Index
5. Gross Domestic Product per capita in US dollar (GDP): Gross domestic product per head in US dollars.

For the Principal Components Analysis application, because the data variables are measured in different units of measurement and there is also a large difference in their variability, we will proceed to principal components based on the correlation matrix and not the variance matrix.

Looking in the covariance matrix of the data set we observe that there is no meaning working on it since there are different measurements.

TABLE 1.1: Covariance Matrix

	increase	life	imr	tfr	gdp
increase	1.76	-8.65	37.11	2.14	-5115.26
life	-8.65	79.69	-305.90	-14.96	49659.45
imr	37.11	-305.90	1381.33	61.54	-203791.98
tfr	2.14	-14.96	61.54	3.57	-9200.41
gdp	-5115.26	49659.45	-203791.98	-9200.41	65534456.39

In the other hand the correlation matrix seems more adequate to work with:

TABLE 1.2: Correlation Matrix

	increase	life	imr	tfr	gdp
increase	1.00	-0.73	0.75	0.86	-0.48
life	-0.73	1.00	-0.92	-0.89	0.69
imr	0.75	-0.92	1.00	0.88	-0.68
tfr	0.86	-0.89	0.88	1.00	-0.60
gdp	-0.48	0.69	-0.68	-0.60	1.00

The diagonalization of the correlation matrix is :

$$\mathbf{EDE^T} =$$

$$\begin{bmatrix} -0.43 & -0.51 & 0.65 & 0.28 & 0.24 \\ 0.47 & -0.05 & 0.48 & 0.16 & -0.72 \\ -0.47 & 0.01 & -0.42 & 0.63 & -0.45 \\ -0.47 & -0.25 & -0.03 & -0.70 & -0.47 \\ 0.38 & -0.82 & -0.43 & 0.04 & 0.03 \end{bmatrix} \begin{bmatrix} 4.01 & 0 & 0 & 0 & 0 \\ 0 & 0.57 & 0 & 0 & 0 \\ 0 & 0 & 0.25 & 0 & 0 \\ 0 & 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0 & 0.07 \end{bmatrix} \begin{bmatrix} 0.43 & -0.51 & 0.65 & -0.28 & 0.24 \\ -0.47 & -0.05 & 0.48 & -0.16 & -0.72 \\ 0.47 & 0.01 & -0.42 & -0.63 & -0.45 \\ 0.47 & -0.25 & -0.03 & 0.70 & -0.47 \\ -0.38 & -0.82 & -0.43 & -0.04 & 0.03 \end{bmatrix}^T$$

The square roots of the eigen values are revealed in the first row of the summary table below which express the variation of each principal component:

$$\sqrt{\lambda_i} = (2.003, 0.754, 0.502, 0.309, 0.263).$$

TABLE 1.3: Importance of components

	PC1	PC2	PC3	PC4	PC5
Standard deviation	2.0035	0.7542	0.50257	0.30964	0.26230
Proportion of Variance	0.8028	0.1138	0.05051	0.01917	0.01376
Cumulative Proportion	0.8028	0.9165	0.96707	0.98624	1.00000

Our eye in the PCA summary table must be fixed in the "cumulative proportion" line, since its importance is high and its interpretation very easy. In our dataset the first PC1 contains the 80% of the variability of the data set.PC2 and PC1 in the other hand provides us 91.6 % of the variability.

Therefore we need only the first two principal components in order to work with and ignore all the others. Using the Scree plot we can observe with the "Rule of Angle" that the first two PCs are sufficient.

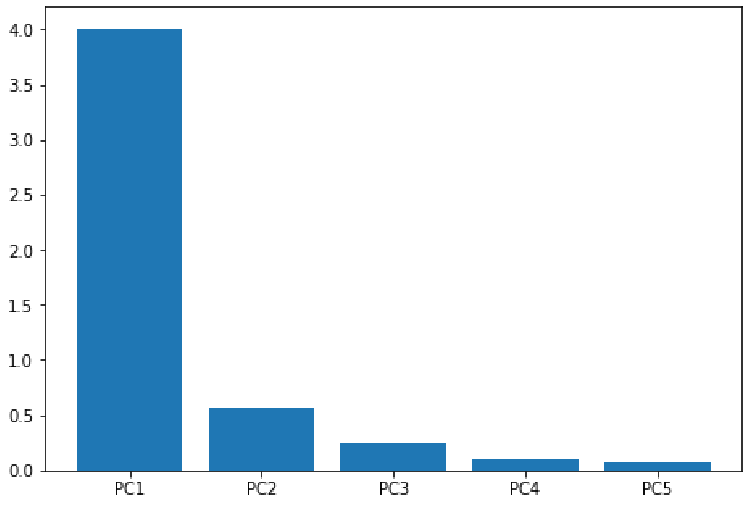


FIGURE 1.8: Scree Plot

Chapter 2

Random Matrices & Randomness as a computational strategy

2.1 Random Matrices

We produce a matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$ matrix, whose entries are independently sampled from a Normal probability density function (pdf) with mean 0 and variance 1. One such matrix for $N = 6$ in R (see Appendix) might look like this:

$$\mathbf{A} = \begin{bmatrix} 0.55 & -1.27 & 0.78 & -0.23 & 0.38 & 0.45 \\ -0.06 & 2.17 & -0.08 & 1.52 & -0.50 & 0.05 \\ -0.31 & 1.21 & 0.25 & -1.55 & -0.33 & 0.92 \\ -0.38 & -1.12 & -0.03 & 0.58 & -1.02 & 2.05 \\ -0.69 & -0.40 & -0.04 & 0.12 & -1.07 & -0.49 \\ -0.21 & -0.47 & 1.37 & 0.22 & 0.30 & -2.31 \end{bmatrix}.$$

Some of the entries are positive, some are negative, none is very far from 0. There is no symmetry in the matrix at this stage,

$$\mathbf{A}_{ij} \neq \mathbf{A}_{ji}.$$

Any time we try, we end up with a different matrix: we call all these matrices samples or instances of our ensemble. The **N** eigenvalues are in general complex numbers.

To get real eigenvalues, the first thing to do is to symmetrize our matrix. Recall that a real symmetric matrix has N real eigenvalues. We will not deal much with ensembles with complex eigenvalues in this thesis.But we will stick for a while in order to obtain the real eigen values.

Every matrix **A** can be decomposed as a sum :

$$\mathbf{A} = \mathrm{Re}(\mathbf{A}) + i\,\mathrm{Im}(\mathbf{A}),$$

where,

$$\mathrm{Re}(\mathbf{A}) = (\mathbf{A} + \mathbf{A}^H)/2,$$

and

$$\mathrm{Im}(\mathbf{A}) = (\mathbf{A} - \mathbf{A}^H)/2i,$$

where , H is conjugate transpose of the matrix.

According to Rajendra Bhatia (Matrix Analysis 1997), this is called **Cartesian Decomposition** of a matrix **A** and decomposes a matrix into its real and imaginary parts.The operators $\mathrm{Re}(\mathbf{A})$ and $\mathrm{Im}(\mathbf{A})$ are both **Hermitian**.

If now we try the following symmetrization is our example matrix :

$$\mathbf{A}_s = (\mathbf{A} + \mathbf{A}^H)/2.$$

the matrix becomes symmetric:

$$\mathbf{A}_s = \begin{bmatrix} 0.55 & -0.66 & 0.24 & -0.30 & -0.16 & 0.12 \\ -0.66 & 2.17 & 0.56 & 0.20 & -0.45 & -0.21 \\ 0.24 & 0.56 & 0.25 & -0.79 & -0.19 & 1.15 \\ -0.30 & 0.20 & -0.79 & 0.58 & -0.45 & 1.13 \\ -0.16 & -0.45 & -0.19 & -0.45 & -1.07 & -0.09 \\ 0.12 & -0.21 & 1.15 & 1.13 & -0.09 & -2.31 \end{bmatrix}.$$

And this happens because a matrix that has only real entries is symmetric if and only if it is Hermitian matrix.

Proof: $\mathbf{H}_{ij} = \overline{\mathbf{H}}_{ji}$ by definition. Thus $\mathbf{H}_{ij} = \mathbf{H}_{ji}$ (matrix symmetry) if and only if

$$\mathbf{H}_{ij} = \overline{\mathbf{H}}_{ij} \quad (\mathbf{H}_{ij} \quad \text{is real}).$$

So, if a real anti-symmetric matrix is multiplied by a multiple of imaginary unit i, then it becomes Hermitian. So, if a real anti-symmetric matrix is multiplied by a multiple of imaginary unit i, then it becomes Hermitian.

Every Hermitian matrix is a normal matrix. That is to say,

$$\mathbf{A}\mathbf{A}^{\mathbf{H}} = \mathbf{A}^{\mathbf{H}}\mathbf{A}.$$

Proof:

$$\mathbf{A} = \mathbf{A}^{\mathbf{H}},$$

therefore ,

$$\mathbf{A}\mathbf{A}^{\mathbf{H}} = \mathbf{A}\mathbf{A} = \mathbf{A}^{\mathbf{H}}\mathbf{A}.$$

Whose six eigenvalues are now all real :

$$\lambda_i = (-3.265, -1.268, 0.069, 0.674, 1.404, 2.564).$$

This procedure is a random matrix drawn from the so-called **GOE (Gaussian Orthogonal Ensemble)**

We can now do several things: for example, you can make the entries complex or quaternionic instead of real. In order to have real eigenvalues, the corresponding matrices need to be hermitian and self-dual respectively better have a look at one example of the former, for N as small as $N = 2$:

$$\mathbf{H}_{er} = \begin{bmatrix} -0.564452 & 1.0844412 - 0.57474i \\ 1.084441 + 0.57474i & -0.8900378 \end{bmatrix}$$

The Figure 2.1 reveals the normal distribution of the eigenvalues of the random normal hermitian matrix of $N = 500$.

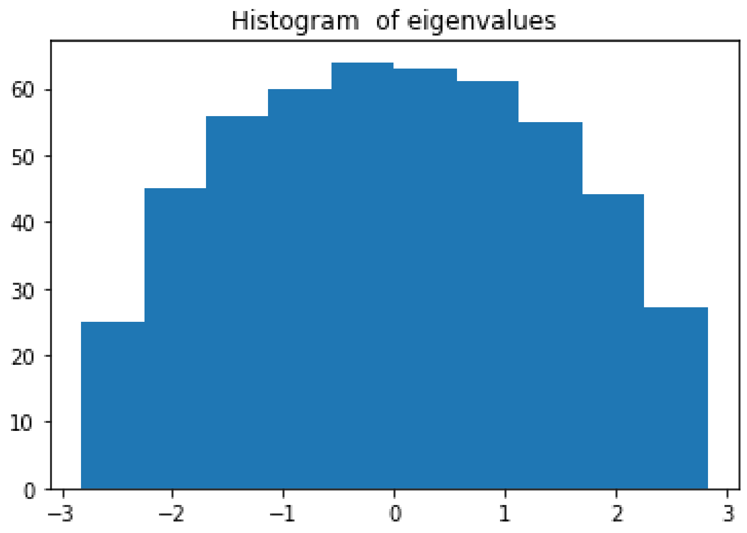


FIGURE 2.1: Histogram of eigenvalues of $N = 500$

2.1.1 Gerschgorin's disk

Let **A** be a complex $n \times n$ matrix, with entries a_{ij}. For $i \in \{1, \dots, n\}$ let R_i be the sum of the absolute values of the non-diagonal entries in the i-th row:

$$R_i = \sum_{j \neq i} |a_{ij}| .$$

Let :

$$D(a_{ii}, R_i) \subseteq \mathbb{C},$$

be a closed disc centered at a_{ii} with radius R_i. Such a disc is called a Gershgorin disc.

Theorem 2.1.1. *Every eigenvalue of* **A** *lies within at least one of the Gershgorin discs* $D(a_{ii}, R_i)$.

Proof: Let λ be an eigenvalue of **A**. Choose a corresponding eigenvector $x = (x_j)$ so that one component x_i is equal to 1 and the others are of absolute value less than or equal to $x_i = 1$ and $|x_j| \leq 1$ for $j \neq i$.

There is always such an x, which can be obtained simply by dividing any eigenvector by its component with largest modulus. Since $Ax = \lambda x$, specifically

$$\sum_j a_{ij} x_j = \lambda x_i = \lambda.$$

So, splitting the sum and taking into account once again that $x_i = 1$, we get

$$\sum_{j \neq i} a_{ij} x_j + a_{ii} = \lambda.$$

Therefore, applying the triangle inequality:

$$|\lambda - a_{ii}| = \left| \sum_{j \neq i} a_{ij} x_j \right| \leq \sum_{j \neq i} |a_{ij}||x_j| \leq \sum_{j \neq i} |a_{ij}| = R_i.$$

Corollary 2.1.1.1. *The eigenvalues of* **A** *must also lie within the Gershgorin discs* C_j *corresponding to the columns of* A.

Example 1 Let **A** matrix:

$$\begin{bmatrix} 8 & 7 & 7 \\ 0 & 2 & \frac{1}{4} \\ 0 & 3 & 1 \end{bmatrix},$$

The Gershgorin Circles are

$$D(8,14),$$
$$D(2,0.25),$$
$$D(1,3).$$

The Eigenvalues however are: $8, 2.5, 0.5$ and therefore the disks are disjoint, so we have each eigenvalue located in its own disk.

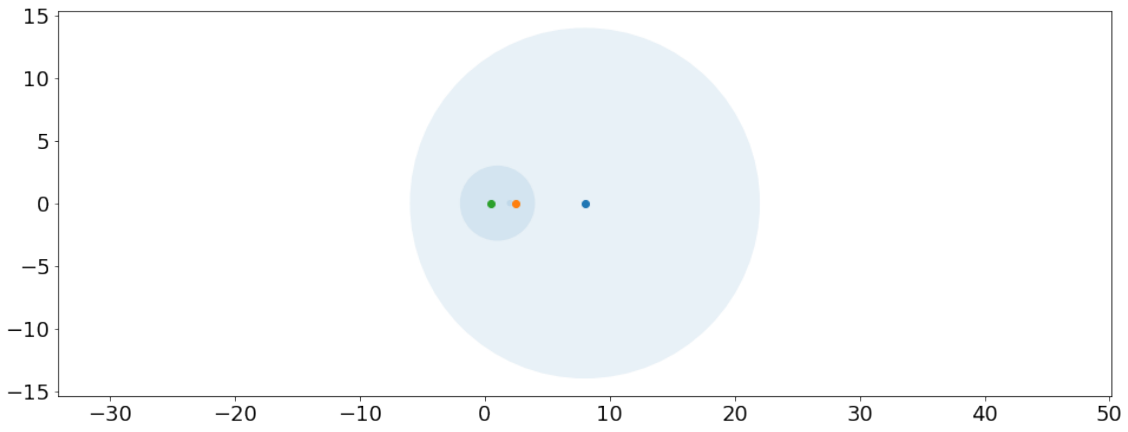


FIGURE 2.2: Gerschgorin disk of Matrix A

Example 2 Suppose

$$\mathbf{A} = \begin{bmatrix} 2 & 2 & 0 \\ 2 & 3 & 4 \\ 0 & 4 & 5 \end{bmatrix}.$$

We will give eigenvalue bounds based on Gershgorin disks for **A**. Note that the disk associated with the $(2,2)$ entry dominates the bounds.

Considering a diagonal similarity $\mathbf{D} = \text{diag}[1, d, 1]$ and the family of Gershgorin disks for $\mathbf{D}^{-1}\mathbf{AD}$ parameterized by $d > 0$ we want to find the best eigenvalue bounds for **A** that we can derive from Gershgorin disks for $\mathbf{D}^{-1}\mathbf{AD}$ by varying d.

The disks are centered at the diagonal entries, and the radius is the sum of the absolute values of the non-diagonal entries in each row.

The currently disks are $[0,4]$, $[-3,9]$, $[1,9]$,

$$\mathbf{D^{-1}AD} = \begin{bmatrix} 2 & 2d & 0 \\ \frac{2}{d} & 3 & \frac{4}{d} \\ 0 & 4d & 5 \end{bmatrix}.$$

$\mathbf{D}^{-1}\mathbf{AD}$ has the same eigenvalues as **DAD**. The disks of $\mathbf{D^{-1}AD}$ are :

$$[2-2d, 2+2d],$$
$$[3-\frac{6}{d}, 3+\frac{6}{d}],$$
$$[5-4d, 5+4d].$$

We are not going to find a d (other than 1) that brings in the upper bound, as we already have 9 as an upper bound on two disks. But we can find a d that brings in the lower bound. There is a d such that :

$$2 - 2d = 3 - \frac{6}{d}$$

$$\begin{aligned} 2d^2 + d - 6 &= 0 \\ (2d-3)(d+2) &= 0 \\ d &= 1.5. \end{aligned}$$

Or :

$$5 - 4d = 3 - \frac{6}{d}$$

$$\begin{aligned} 4d^2 - 2d - 6 &= 0 \\ 2(2d-3)(d+1) &= 0 \\ d &= 1.5. \end{aligned}$$

1.5 is a winner and the lower bound is -1

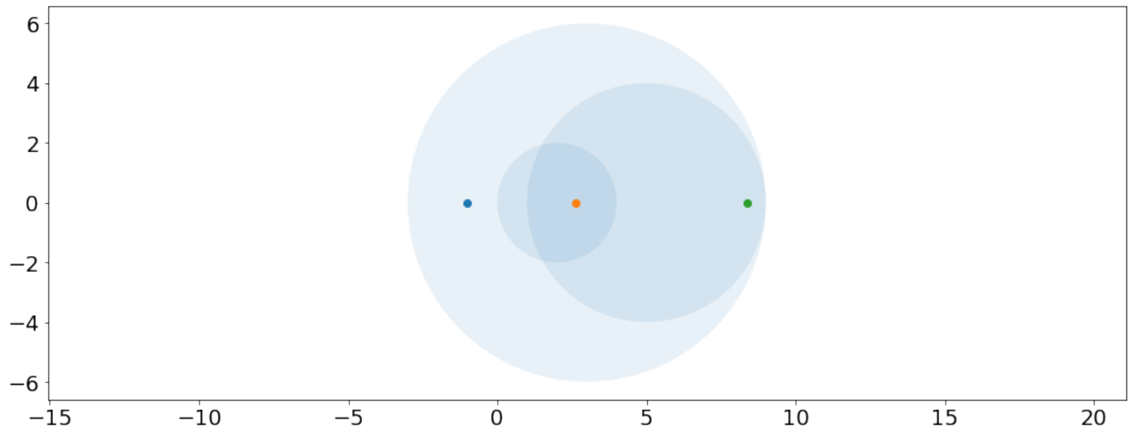


FIGURE 2.3: Gerschgorin disk of Matrix A

2.1.2 ε - Pseudospectrum

Normal matrix is a matrix that satisfies the following condition :

$$\mathbf{A}\mathbf{A}^{\mathbf{H}} = \mathbf{A}^{\mathbf{H}}\mathbf{A}. \tag{2.1}$$

With the opposite $\mathbf{A}\mathbf{A}^{\mathbf{H}} \neq \mathbf{A}^{\mathbf{H}}\mathbf{A}$ to denote the non normal matrix. Consider, as a function of $z \in \mathbb{C}$, the norm of the resolvent:

$$(\mathbf{zI} - \mathbf{A})^{-\mathbf{1}}.$$

When z is a eigenvalue of $\mathbf{A}$, $||(\mathbf{zI} - \mathbf{A})^{-\mathbf{1}}||$ can be thought of as infinite, and we shall use this convention.

Otherwise, is finite. How large? If $\mathbf{A}$ is normal the answer is simple :

$$||(\mathbf{zI} - \mathbf{A})^{-\mathbf{1}}|| = \frac{\mathbf{1}}{\text{dist}(\mathbf{z}, \mathbf{\Lambda}(\mathbf{A}))}.$$

Here $\mathbf{\Lambda}(\mathbf{A})$ again denotes the spectrum of $\mathbf{A}$ (all the eigenvalues of $\mathbf{A}$), and $\text{dist}(\mathbf{z}, \mathcal{S})$ is the usual distance from the point z to the set $\mathcal{S}$.

Thus in the normal case, the surface $||(\mathbf{zI} - \mathbf{A})^{-\mathbf{1}}||$ is determined entirely by the eigenvalues.In the normal case, however, is only a lower bound and the shape of the surface cannot be inferred from the eigenvalues.

Example: Consider a defective singular matrix

$$\mathbf{A} = \begin{bmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \\ 4 & -8 & 4 \end{bmatrix}.$$

Example: We consider a defective matrix that has one eigenvalue $\lambda = 1$. It's resolvent is 0.14073

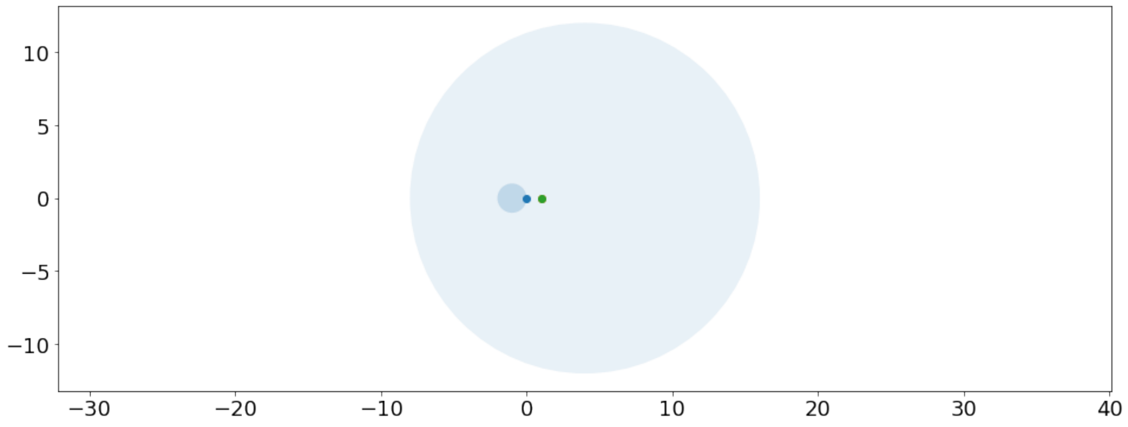


FIGURE 2.4: Gerschgorin disk of Matrix A

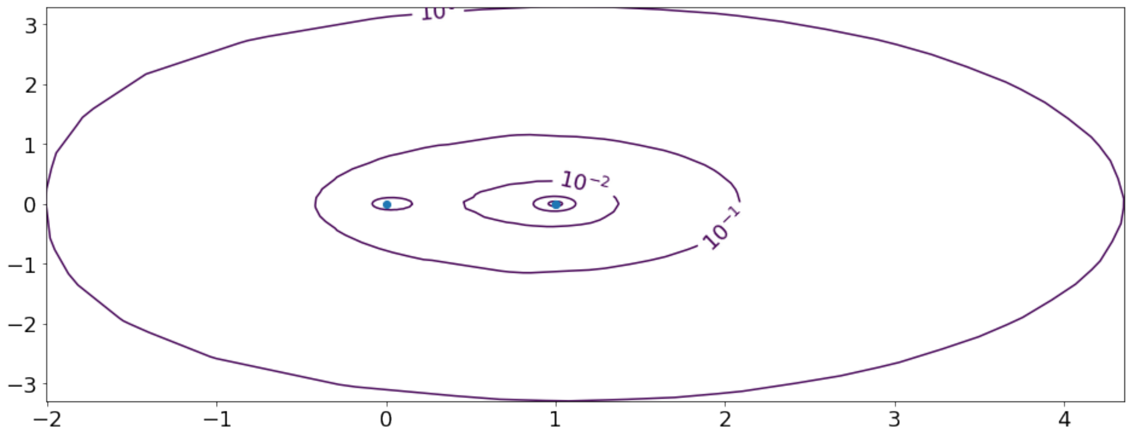


FIGURE 2.5: Pseudomatrix of Matrix A

$$\mathbf{B} = \begin{bmatrix} -13 & -2 & 6 \\ 52 & 5 & -20 \\ -22 & -4 & 11 \end{bmatrix}.$$

Its resolvent is : 0.01488

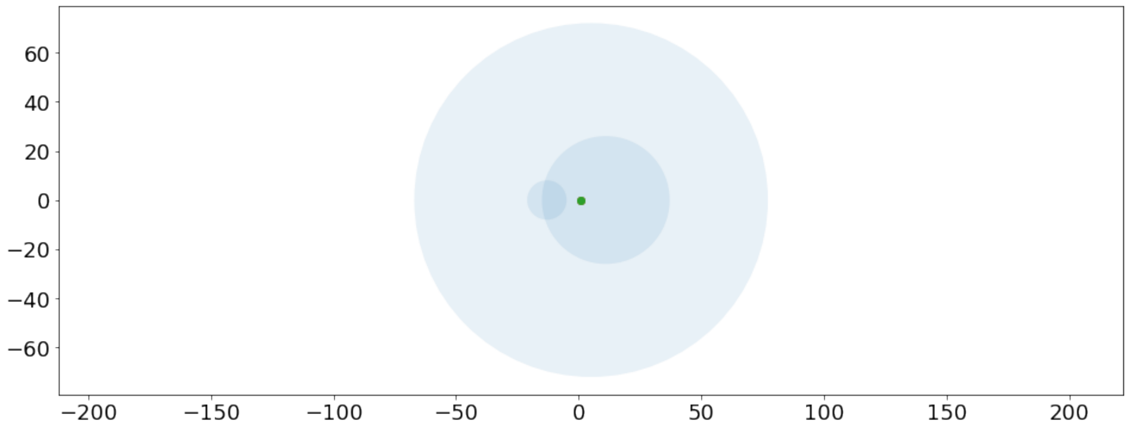


FIGURE 2.6: Gerschgorin disk of Matrix B

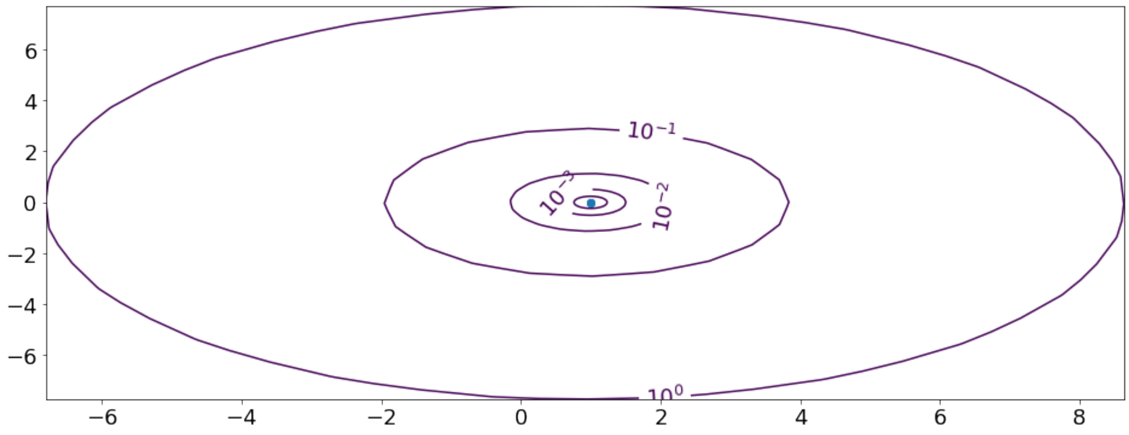


FIGURE 2.7: Pseudomatrix of Matrix B

Given $\mathbf{A} \in \mathbb{C}^{m \times m}$ with spectrum: $\Lambda(\mathbf{A}) \subseteq \mathbb{C}$ and $\epsilon > 0$, define the 2-norm ϵ-pseudospectrum* of $\mathbf{A}$, $\mathbf{A}_\epsilon(\mathbf{A})$, to be the set of numbers $z \in \mathbb{C}$ satisfying any of the following conditions:

- z is an eigenvalue of $\mathbf{A} + \mathbf{E}$ for some pertubation $\mathbf{E}$ with $||\mathbf{E}||_2 \leq \epsilon$,
- there exists a vector $u \in \mathbb{C}^m$ with $||(\mathbf{A} - \mathbf{zI})\mathbf{u}||_2 \leq \epsilon$ and $||\mathbf{u}||_2 = \mathbf{1}$,
- $\sigma_{min}(\mathbf{zI} - \mathbf{A}) \leq \epsilon$,
- $||(\mathbf{zI} - \mathbf{A})^{-\mathbf{1}}||_2 \geq \epsilon^{-1}$.

Very meaningful for interpretation is the first condition that the definition can be stated in terms of pertubations of $\mathbf{A}$:

$$\Lambda_\varepsilon(\mathbf{A}) = \{z \in \mathbb{C} : z \text{ is an eigenvalue of } \quad \mathbf{A} + \mathbf{E} \quad \text{for some} \quad \mathbf{E} \quad \text{with} \quad ||\mathbf{E}|| \leq \varepsilon\}.$$

In other words, a pseudo-eigenvalue of $\mathbf{A}$ is an eigenvalue of a slighlty pertubed matrix.

Pseudospectra are typically computed by establishing a grid with N points on a region of the complex plane, computing the resolvent norm $||(\mathbf{zI} - \mathbf{A})^{-1}||$ at each grid point z, and visualizing with a contour plotter.

Letting $\sigma_{max}()$ and $\sigma_{min}()$ denote the largest and smallest singular values of an input matrix, respectively, we remark that the resolvent norm satisfies :

$$||(\mathbf{zI} - \mathbf{A})^{-1}||_2 = \sigma_{max}((\mathbf{zI} - \mathbf{A})^{-1}) = \frac{1}{\sigma_{min}(\mathbf{zI} - \mathbf{A})}.$$

Thus, one could naively compute pseudospectra by computing the SVD of $\mathbf{zI} - \mathbf{A}$ for each grid point z and reporting the reciprocal of the smallest singular value. However, this involves a total of $\mathcal{O}(Nn^3)$ flops, which is prohibitively expensive for large matrices unless the grid is very coarse.

The **Van Loan/Lui algorithm** improves on the computational cost by proceeding in two stages. It begins by computing a Schur decomposition $\mathbf{A} = \mathbf{QTQ^H}$ where $\mathbf{T}$ is upper triangular and $\mathbf{Q}$ unitary. Since matrix norms are invariant under unitary transformations such as $\mathbf{I} = \mathbf{QQ^H}$:

$$\begin{aligned}
||(\mathbf{zI} - \mathbf{A})^{-1}||_2 &= ||(\mathbf{zI} - \mathbf{QTQ^H})||_2 \\
&= ||(\mathbf{zQQ^H} - \mathbf{QTQ^H})^{-1}||_2 \\
&= ||(-\mathbf{QTQ^H} + \mathbf{zQQ^H})^{-1}||_2 \\
&= ||(-\mathbf{QQ^H}(-\mathbf{T} + \mathbf{zI}))^{-1}||_2 \\
&= ||(-\mathbf{I}(-\mathbf{T} + \mathbf{zI}))^{-1}||_2 \\
&= ||(\mathbf{T} - \mathbf{zI})^{-1}||_2.
\end{aligned}$$

In the following chapters we will calculate the pseudospectra of images and other dataset (design matrix) in PCA.

2.2 Probabilistic framework for low-rank approximations

Assume that a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ has rank $\mathbf{r}$, where $\mathbf{r} \leq \min(\mathbf{m}, \mathbf{n})$. Then, in general, the objective of a low-rank matrix approximation is to find two smaller matrices such that:

$$\underbrace{\mathbf{A}}_{m \times n} \approx \underbrace{\mathbf{E}}_{m \times r} \underbrace{\mathbf{F}}_{r \times n} \tag{2.2}$$

where the columns of the matrix $\mathbf{E} \in \mathbb{R}^{m \times r}$ span the column space of $\mathbf{A}$., and the rows of the matrix $\mathbf{F} \in \mathbb{R}^{r \times n}$ span the row space of $\mathbf{A}$.

The factors $\mathbf{E}$ and $\mathbf{F}$ can then be used to summarize or to reveal some interesting structure in the data. Further, the factors can be used to efficiently store the large data matrix $\mathbf{A}$. Specifically, while $\mathbf{A}$ requires mn words of storage, $\mathbf{E}$ and $\mathbf{F}$ require only $\mathbf{mr} + \mathbf{nr}$ words of storage.

In practice, most data matrices do not feature a precise rank $\mathbf{r}$. Rather we are commonly interested in finding a rank-k matrix $\mathbf{A_k}$, which is as close as possible to an arbitrary input matrix $\mathbf{A}$ in the least-square sense. We refer to $\mathbf{k}$ as the target rank in the following. In particular, modern data analysis and scientific computing largely rely on low-rank approximations, since low-rank matrices are ubiquitous throughout the sciences.

However, in the era of "big data", the emergence of massive data poses a significant computational challenge for traditional deterministic algorithms. In the following, we advocate the probabilistic framework, formulated by Halko et al. (2011b), to compute a near-optimal low-rank approximation. Conceptually, this framework splits the computational task into two logical stages:

- **Stage A**: Construct a low dimensional subspace that approximates the column space of $\mathbf{A}$. This means, the aim is to find a matrix $\mathbf{Q} \in \mathbb{R}^{m \times k}$ with orthonormal columns such that $\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^{\mathbf{T}}\mathbf{A}$ is satisfied.
- **Stage B**: Form a smaller matrix $\mathbf{B} := \mathbf{Q}^{\mathbf{T}}\mathbf{A} \in \mathbb{R}^{k \times n}$, i.e., restrict the high-dimensional input matrix to the low-dimensional space spanned by the near-optimal basis $\mathbf{Q}$. The smaller matrix $\mathbf{B}$ can then be used to compute a desired low-rank approximation.

The first computational stage is where randomness comes into the play, while the second stage is purely deterministic. In the following sub chapter , the two stages are described in detail.

2.2.1 The generic randomized algorithm

- **Stage A: Computing the near-optimal basis** First, we aim to find a near-optimal basis **Q** for the matrix **A** such that:

$$\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^{\mathbf{T}}\mathbf{A}, \tag{2.3}$$

is satisfied.

The desired target rank k is assumed to be $k \ll \min(m, n)$. Specifically:

$$\mathbf{P} := \mathbf{Q}\mathbf{Q}^{\mathbf{T}}, \tag{2.4}$$

is a linear orthogonal projector. A projection operator corresponds to a linear subspace, and transforms any vector to its orthogonal projection on the subspace.

This is illustrated in Figure 1, where a vector **x** is confined to the column space col(A).

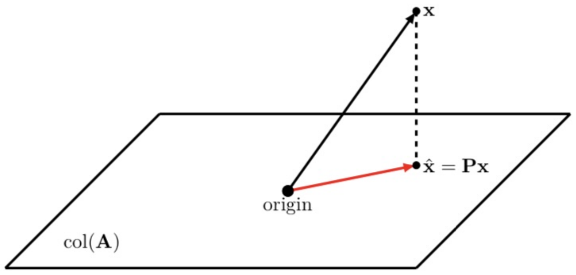


FIGURE 2.8: Geometric illustration of the orthogonal projection operator P. A vector $x \in \mathbb{R}^m$ is restricted to the column space of A, where $Px \in$ col(A).

The concept of random projections can be used to sample the range (column space) of the input matrix **A** in order to efficiently construct such a orthogonal projector.

Random projections are data agnostic, and constructed by first drawing a set of k random vectors $\{\omega_i\}k_i = 1$, for instance, from the standard normal distribution. Probability theory guarantees that random vectors are linearly independent with high probability.

Then, a set of random projections $\{y_i\}k_i = 1$ is computed by mapping **A** to low-dimensional space:

$$y_i := \mathbf{A}\omega_i, \quad \forall i = 1, 2, \ldots, k. \tag{2.5}$$

In other words, this process forms a set of independent randomly weighted linear combinations of the columns of A, and reduces the number of columns from n to k. While the input matrix is compressed, the Euclidean distances between the original data points are approximately preserved. Equation 2.3 can be efficiently executed in parallel. Therefore, let us define the random test matrix $\mathbf{\Omega} \in \mathbb{R}^{n\times k}$, which is again drawn from the standard normal distribution, and the columns of which are given by the vectors $\{\omega_i\}$.

The samples matrix $\mathbf{Y} \in \mathbb{R}^{m\times k}$, also denoted as sketch, is then obtained by post-multiplying the input matrix by the random test matrix

$$\mathbf{Y} := \mathbf{A}\mathbf{\Omega}. \tag{2.6}$$

Once **Y** is obtained, it only remains to orthonormalize the columns in order to form a natural basis $\mathbf{Q} \in \mathbb{R}^{m \times k}$. This can be efficiently achieved using the QR-decomposition

$$\mathbf{Y} =: \mathbf{QR},$$

and it follows that Equation 2 is satisfied.

- **Stage B:Compute the smaller matrix**

  Now, given the near-optimal basis **Q**, we aim to find a smaller matrix $\mathbf{B} \in \mathbb{R}^{k \times n}$. Therefore, we project the high-dimensional input matrix A to low-dimensional space:

  $$\mathbf{B} := \mathbf{Q}^{\mathbf{T}}\mathbf{A}. \tag{2.7}$$

  Geometrically, this is a projection (i.e., a linear transformation) which takes points in a high-dimensional space into corresponding points in a low-dimensional space, illustrated in

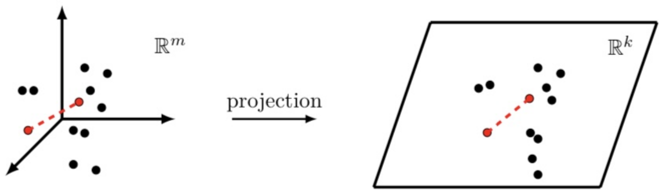


FIGURE 2.9: Figure 2.3

  Figure 2.3. This process preserves the geometric structure of the data in an Euclidean sense, i.e., the length of the projected vectors as well as the angles between the projected vectors are preserved. This is, due to the invariance of inner products (Trefethen and Bau 1997). Substituting Equation 2.7 into 2.3 yields then the following low-rank approximation

  $$\underbrace{\mathbf{A}}_{m \times n} \approx \underbrace{\mathbf{Q}}_{m \times k}\underbrace{\mathbf{B}}_{k \times n}.$$

  This decomposition is referred to as the QB decomposition. Subsequently, the smaller matrix B can be used to compute a matrix decomposition using a traditional algorithm.

The intuitive approach of **the generic randomized algorithm** can be applied to general matrices. Omitting computational details for now, we formalize the procedure in the figure labeled Proto-Algorithm.

This simple algorithm is by no means new. It is essentially the first step of a subspace iteration with a random initial subspace. The novelty comes from the additional observation that the initial subspace should have a slightly higher dimension than the invariant subspace we are trying to approximate.

With this revision, it is often the case that no further iteration is required to obtain a high quality solution to:

$$\mathbf{A} - \mathbf{QQ}^{\mathbf{H}}\mathbf{A}.$$

The basic challenge in producing low-rank maTrix approximations is a primitive question that we call the fixed-precision approximation problem. Suppose we are given a matrix **A** and a positive error tolerance ε. We seek a matrix **Q** with $k = k(\varepsilon)$ orthonormal columns such that :

$$|||\mathbf{A} - \mathbf{QQ}^{\mathbf{H}}\mathbf{A}|| \leq \varepsilon, \tag{2.8}$$

where , $||\cdot||$ denotes the l_2 operator norm. The range of $\mathbf{Q}$ is a k-dimensional subspace that captures most of the action of $\mathbf{A}$, and we would like k to be as small as possible.

In order to invoke the proto-algorithm with confidence, we must address several practical and theoretical issues:

- What random matrix $\mathbf{\Omega}$ should we use? How much oversampling do we need?
- The matrix $\mathbf{Y}$ is likely to be ill-conditioned. How do we orthonormalize its columns to form the matrix $\mathbf{Q}$?
- What are the computational costs?
- How can we solve the fixed-precision problem (2.8) when the numerical rank of the matrix is not known in advance?
- How can we use the basis $\mathbf{Q}$ to compute other matrix factorizations?
- Does the randomized method work for problems of practical interest? How does its speed/accuracy/robustness compare with standard techniques?
- What error bounds can we expect? With what probability?

The next few sections provide a summary of the answers to these questions. We describe several problem regimes where the proto-algorithm can be implemented efficiently, and we present a theorem that describes the performance of the most important instantiation.

Finally, we elaborate on how these ideas can be applied to approximate the truncated SVD of a large data matrix. The rest of the paper contains a more exhaustive treatment including pseudocode, numerical experiments, and a detailed theory.

2.2.2 Comparison between randomized and traditional techniques

To select an appropriate computational method for finding a low-rank approximation to a matrix, the practitioner must take into account the properties of the matrix. Is it dense or sparse? Does it fit in fast memory or is it stored out of core?

Does the singular spectrum decay quickly or slowly? The behavior of a numerical linear algebra algorithm may depend on all these factors. To facilitate a comparison be- tween classical and randomized techniques, we summarize their relative performance in each of three representative environments.

We focus on the task of computing an approximate SVD of an $m \times n$ matrix A with numerical rank k. For randomized schemes, **Stage A** generally dominates the cost of **Stage B** in our matrix approximation framework.

Within **Stage A**, the computational bottleneck is usually the matrix–matrix product $\mathbf{A\Omega}$ in Step 2 of the proto-algorithm (1.3.3). The power of randomized algorithms stems from the fact that we can reorganize this matrix multiplication for maximum efficiency in a variety of computational architectures. A standard deterministic technique for computing an approximate **SVD** is to perform a rank-revealing **QR** factorization of the matrix, and then to manipulate the factors to obtain the final decomposition. The cost of this approach is typically $\mathcal{O}(kmn)$ floating-point operations, or flops, although these methods require slightly longer running times in rare cases.

In contrast, randomized schemes can produce an approximate SVD using only $\mathcal{O}(mn\log(k) + (m+n)k^2)$ flops. The gain in asymptotic complexity is achieved by using a random matrix $\mathbf{\Omega}$ that has some internal structure, which allows us to evaluate the product $\mathbf{A\Omega}$ rapidly.

When the matrix **A** is sparse or structured, we may be able to apply it rapidly to a vector. In this case, the classical prescription for computing a partial SVD is to invoke a Krylov subspace method, such as the Lanczos or Arnoldi algorithm. It is difficult to summarize the computational cost of these methods because their performance depends heavily on properties of the input matrix and on the amount of effort spent to stabilize the algorithm. (Inherently, the Lanczos and Arnoldi methods are numerically unstable.) For the same reasons, the error analysis of such schemes is unsatisfactory in many important environments.

At the risk of being overly simplistic, we claim that the typical cost of a Krylov method for approximating the k leading singular vectors of the input matrix is proportional to $kT_{mult} + (m+n)k^2$, where T_{mult} denotes the cost of a matrix–vector multiplication with the input matrix and the constant of proportionality is small. We can also apply randomized methods using a Gaussian test matrix Ω to complete the factorization at the same cost, $\mathcal{O}(kT_{mult} + (m+n)k^2)$ flops.

With a given budget of floating-point operations, Krylov methods sometimes deliver a more accurate approximation than randomized algorithms. Nevertheless, the methods described in this survey have at least two powerful advantages over Krylov methods. First, the randomized schemes are inherently stable, and they come with very strong performance guarantees that do not depend on subtle spectral properties of the input matrix. Second, the matrix–vector multiplies required to form $\mathbf{A\Omega}$ can be performed in parallel.

This fact allows us to restructure the calculations to take full advantage of the computational platform, which can lead to dramatic accelerations in practice, especially for parallel and distributed machines. When the input matrix is too large to fit in core memory, the cost of transferring the matrix from slow memory typically dominates the cost of performing the arithmetic.

The standard techniques for low-rank approximation described in the first chapter of this thesis require $\mathcal{O}(k)$ passes over the matrix, which can be prohibitively expensive. In contrast, the proto-algorithm requires only one pass over the data to produce the approximate basis Q for Stage A of the approximation framework.

This straightforward approach, unfortunately, is not accurate enough for matrices whose singular spectrum decays slowly, but we can address this problem using very few (say, 2 to 4) additional passes over the data. Typically, Stage B uses one additional pass over the matrix to construct the approximate SVD. With slight modifications, however, the two-stage randomized scheme can be revised so that it only makes a single pass over the data.

2.3 Oversampling

Most data matrices do not feature an exact rank, which means that the singular values σ_i of the input matrix A are non-zero. As a consequence, the sketch Y does not exactly span the column space of the input matrix. Oversampling can be used to overcome this issue by using $l := k + p$ random projections to form the sketch, instead of just k. Here, p denotes the number of additional projections, and a small number $p = (5, 10)$ is often sufficient to obtain a good basis that is comparable to the best possible basis (Martinsson 2016).

The intuition behind the oversampling scheme is the following. The sketch Y is a random variable, as it depends on the drawing of a random test matrix Ω. Increasing the number of additional random projections allows one to decrease the variation in the singular value spectrum of the random test matrix, which subsequently improves the quality of the sketch.

Chapter 3

Randomized Singular Value Decomposition

3.1 Randomized algorithm

Randomized algorithms have been recently popularized, in large part due to their "surprising" reliability and computational efficiency (Gu 2015). These techniques can be used to obtain an approximate rank-k singular value decomposition at a cost of $\mathcal{O}(mnk)$. When the dimensions of **A** are large, this is substantially more efficient than truncating the full SVD.

We present details of the randomized low-rank SVD algorithm, which comes with favorable error bounds relative to the optimal truncated SVD, as presented in the seminal paper by Halko et al. (2011b), and further analyzed and implemented in Voronin and Martinsson (2015). Let $\mathbf{A} \in \mathbb{R}^{m \times n}$ be a low-rank matrix, and without loss of generality $m \geq n$. In the following, we seek the near-optimal low-rank approximation of the form:

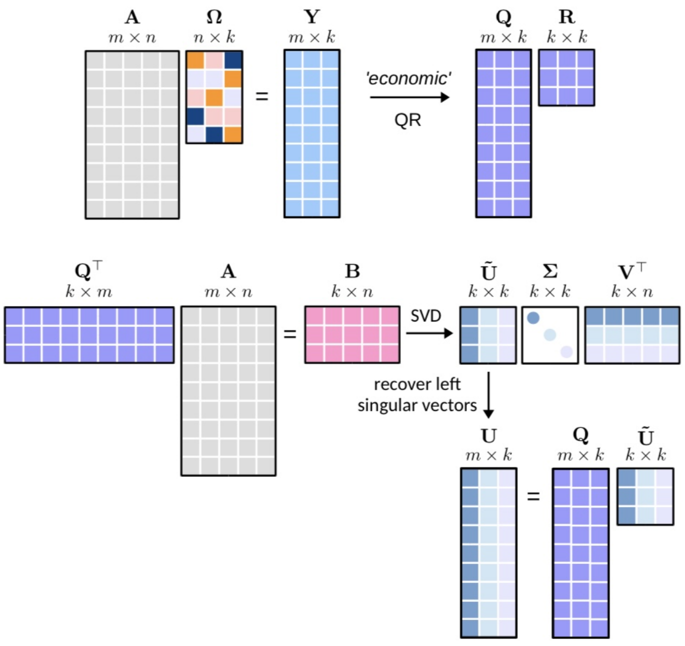


FIGURE 3.1: Figure 3.1

Figure 3.1 : Conceptual architecture of the randomized singular value decomposition (rSVD). First, a natural basis **Q** is computed in order to derive the smaller matrix B. Then, the SVD is efficiently computed using this smaller matrix. Finally, the left singular vectors **U** may be reconstructed from the approximate singular vectors $\bar{\mathbf{U}}$ by the expression in Equation 3.1 where k denotes the target rank. Instead of computing the singular value decomposition directly, we embed the SVD into the probabilistic framework presented in Section 2. The principal concept is sketched in Figure 3.1.

Specifically, we first compute the near-optimal basis $\mathbf{Q} \in \mathbb{R}^{m \times l}$ using the randomized scheme as outlined in detail above. Note that we allow for both oversampling ($\mathbf{l} = \mathbf{k} + \mathbf{p}$), and additional power iterations $\mathbf{q}$, in order to obtain the near-optimal basis matrix. The matrix $\mathbf{B} \in \mathbb{R}^{l \times n}$ is relatively small if $\mathbf{l} \ll \mathbf{n}$, and it is obtained by projecting the input matrix to low-dimensional space, i.e., $\mathbf{B} := \mathbf{Q^T A}$. The full SVD of $\mathbf{B}$ is then computed using a deterministic algorithm

$$\mathbf{B} = \tilde{\mathbf{U}}\mathbf{\Sigma}\mathbf{V^T}.$$

Thus, we efficiently obtain the first l right singular vectors $V \in \mathbb{R}^{n \times l}$ as well as the corresponding singular values $\mathbf{\Sigma} \in \mathbb{R}^{l \times l}$. It remains to recover the left singular vectors $\mathbf{U} \in \mathbb{R}^{m \times l}$ from the approximate left singular vectors $\tilde{\mathbf{U}} \in \mathbb{R}^{l \times l}$ by pre-multiplying by $\mathbf{Q}$

$$\mathbf{U} \approx \mathbf{Q}\tilde{\mathbf{U}}. \tag{3.1}$$

The justification for the randomized SVD can be sketched as follows:

$$\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^T\mathbf{A} = \mathbf{Q}\mathbf{B} = \mathbf{Q}\tilde{\mathbf{U}}\mathbf{\Sigma}\mathbf{V}^T = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T.$$

3.1.1 Improved randomized algorithm

The basis matrix $\mathbf{Q}$ often fails to provide a good approximation for the column space of the input matrix. This is because most real-world data matrices do not feature a precise rank r, and instead exhibit a gradually decaying singular value spectrum. The performance can be considerably improved using the concept of oversampling and the power iteration scheme.

Oversampling

Most data matrices do not feature an exact rank, which means that the singular values $\{\sigma_i\}_{i=k+1}^{n}$ of the input matrix $\mathbf{A}$ are non-zero.

As a consequence, the sketch $\mathbf{Y}$ does not exactly span the column space of the input matrix. Oversampling can be used to overcome this issue by using $l := k + p$ random projections to form the sketch, instead of just k. Here, p denotes the number of additional projections, and a small number $p = \{5, 10\}$ is often sufficient to obtain a good basis that is comparable to the best possible basis Martinsson 2016.

The intuition behind the oversampling scheme is the following. The sketch $\mathbf{Y}$ is a random variable, as it depends on the drawing of a random test matrix $\mathbf{\Omega}$. Increasing the number of additional random projections allows one to decrease the variation in the singular value spectrum of the random test matrix, which subsequently improves the quality of the sketch.

Power iteration scheme

The second method for improving the quality of the basis $\mathbf{Q}$ involves the concept of power sampling iterations as presented from Rokhlin 2009, Halko 2011 and Gu 2015.

Instead of obtaining the sketch $\mathbf{Y}$ directly, the data matrix $\mathbf{A}$ is first preprocessed as :

$$\mathbf{A}^{(q)} := (\mathbf{A}\mathbf{A}^\top)^q\mathbf{A}, \tag{3.2}$$

where , q is an integer specifying the number of power iterations. This process enforces a more rapid decay of the singular values. Thus, we enable the algorithm to sample the relevant information related to the dominant singular values, while unwanted information is suppressed.

Let $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ be the singular value decomposition. It is simple to show that

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

Here, **U** and **V** are orthonormal matrices whose columns are the left and right singular vectors of **A**, and **Σ** is a diagonal matrix containing the singular values in descending order. Hence, for $q > 0$, the modified matrix $\mathbf{A}^{(q)}$ has a relatively fast decay of singular values even when the decay in **A** is modest. This is illustrated in Figure 3.2 showing the singular values of a 50×50 low-rank matrix before (red) and after computing $q = \{1, 2, 3\}$ power iterations.

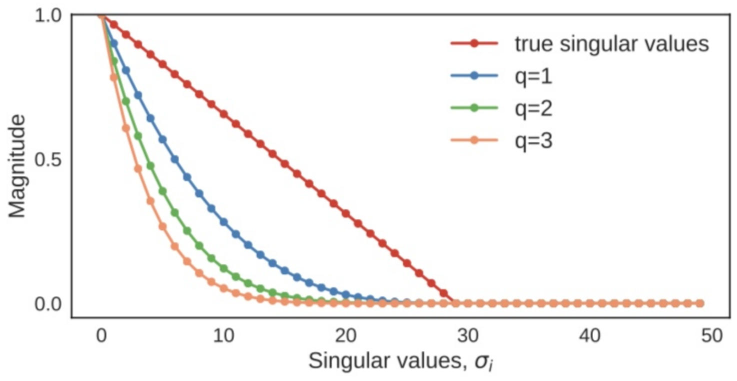


FIGURE 3.2: Power Itertions

Thus, substituting Equation 3.2 into :

$$\mathbf{Y} := \mathbf{A}\mathbf{Q},$$

yields an improved sketch

$$\mathbf{Y} := \mathbf{A}^{(q)}\mathbf{\Omega}.$$

When the singular values of the data matrix decay slowly, as few as $q = \{1, 2, 3\}$ power iterations can considerably improve the accuracy of the approximation. The drawback of the power scheme is that q additional passes over the input matrix are required.

The algorithm of power iteration shows a direct implementation of the power iteration scheme. Due to potential round-off errors, however, this algorithm is not recommended in practice.

3.2 A two-stage approach

The problem of computing an approximate low-rank factorization to a given matrix can conveniently be split into two distinct "stages." For concreteness, we describe the split for the specific task of computing an approximate singular value decomposition. To be precise, given an $m \times n$ matrix **A** and a target rank k, we seek to compute factors **U**, **D**, and **V** such that

$$\underset{m \times n}{\mathbf{A}} \approx \underset{m \times k}{\mathbf{U}} \; \underset{k \times k}{\mathbf{D}} \; \underset{k \times n}{\mathbf{V}^*}.$$

We split this task into two computational stages:

Stage A — find an approximate range: Construct an $m \times k$ matrix **Q** with orthonormal columns such that $\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^*\mathbf{A}$.

(In other words, the columns of **Q** form an approximate basis for the column space of **A**.) This step will be executed via a randomized process described in Section 3.3.

Stage B — form a specific factorization: Given the matrix **Q** computed in Stage A, form the factors **U**, **D**, and **V** using classical deterministic techniques. For instance, this stage can be executed via the following steps:

1. Form the $k \times n$ matrix $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$.
2. Compute the SVD of the (small) matrix $\mathbf{B}$ so that $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$.
3. Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

The point here is that in a situation where $k \ll \min(m, n)$, the difficult part of the computation is all in Stage A. Once that is finished, the post-processing in Stage B is easy, as all matrices involved have at most k rows or columns.

Stage B is exact up to floating point arithmetic so all errors in the factorization process are incurred at Stage A. To be precise, we have

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

In other words, if the factor $\mathbf{Q}$ satisfies $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\| \le \varepsilon$, then automatically:

$$\|\mathbf{A} - \mathbf{U}\mathbf{D}\mathbf{V}^*\| = \|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\| \le \varepsilon, \tag{3.3}$$

unless ε is close to the machine precision.

A bound of the form (3.3) implies that the diagonal elements $\{\mathbf{D}(i,i)\}_{i=1}^{k}$ of $\mathbf{D}$ are accurate approximations to the singular values of $\mathbf{A}$ in the sense that $|\sigma_i - \mathbf{D}(i,i)| \le \varepsilon$ for $i = 1, 2, \ldots, k$.

However, a bound like (3.3) does not provide assurances on the *relative errors* in the singular values; nor does it provide strong assurances that the columns of $\mathbf{U}$ and $\mathbf{V}$ are good approximations to the singular vectors of $\mathbf{A}$.

ALGORITHM: RSVD — BASIC RANDOMIZED SVD

Inputs: An $m \times n$ matrix $\mathbf{A}$, a target rank k, and an over-sampling parameter p (say $p = 10$).
Outputs: Matrices $\mathbf{U}$, $\mathbf{D}$, and $\mathbf{V}$ in an approximate rank-$(k+p)$ SVD of $\mathbf{A}$ (so that $\mathbf{U}$ and $\mathbf{V}$ are orthonormal, $\mathbf{D}$ is diagonal, and $\mathbf{A} \approx \mathbf{U}\mathbf{D}\mathbf{V}^*$.)
Stage A:

1. Form an $n \times (k+p)$ Gaussian random matrix $\mathbf{G}$.
```
G = np.random.normal(n,k+p)
```
2. Form the sample matrix $\mathbf{Y} = \mathbf{A}\,\mathbf{G}$.
```
Y = A@G
```
3. Orthonormalize the columns of the sample matrix $\mathbf{Q} = \mathtt{orth}(\mathbf{Y})$.
```
[Q,~] = np.linalg.qr(Y)
```

Stage B:

4. Form the $(k+p) \times n$ matrix $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$.
```
B = Q' @A
```
5. Form the SVD of the small matrix $\mathbf{B}$: $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$.
```
[Uhat,D,V] = np.linalg.svd(B)
```
6. Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.
```
U = Q @Uhat
```

FIGURE 3.3: A basic randomized algorithm. If a factorization of precisely rank k is desired, the factorization in Step 5 can be truncated to the k leading terms. The text in red is Python code using numpy module for executing each line.

3.3 A randomized algorithm for "Stage A" — the range finding problem

This section describes a randomized technique for solving the range finding problem introduced as "Stage A" in Section 3.2.

As a preparation for this discussion, let us recall that an "ideal" basis matrix $\mathbf{Q}$ for the range of a given matrix $\mathbf{A}$ is the matrix $\mathbf{U}_k$ formed by the k leading left singular vectors of $\mathbf{A}$.

Letting $\sigma_j(\mathbf{A})$ denote the j'th singular value of $\mathbf{A}$, the Eckart-Young theorem states that:

$$\inf\{||\mathbf{A} - \mathbf{C}|| : \ \mathbf{C} \text{ has rank } k\} = ||\mathbf{A} - \mathbf{U}_k\mathbf{U}_k^*\mathbf{A}|| = \sigma_{k+1}(\mathbf{A}).$$

Now consider a simplistic randomized method for constructing a spanning set with k vectors for the range of a matrix $\mathbf{A}$:

- Draw k random vectors $\{\mathrm{g}_j\}_{j=1}^k$ from a Gaussian distribution, map these to vectors $\mathrm{y}_j = \mathbf{A}\mathrm{g}_j$ in the range of $\mathbf{A}$, and then use the resulting set $\{\mathrm{y}_j\}_{j=1}^k$ as a basis.
- Upon orthonormalization via, e.g., Gram-Schmidt, an orthonormal basis $\{\mathrm{q}_j\}_{j=1}^k$ would be obtained. For the special case where the matrix $\mathbf{A}$ has *exact* rank k, one can prove that the vectors $\{\mathbf{A}\mathrm{g}_j\}_{j=1}^k$ would with probability 1 be linearly independent, and the resulting orthonormal (ON) basis $\{\mathrm{q}_j\}_{j=1}^k$ would therefore exactly span the range of $\mathbf{A}$.

This would in a sense be an ideal algorithm. The problem is that in practice, there are almost always many non-zero singular values beyond the first k ones. The left singular vectors associated with these modes all "pollute" the sample vectors $\mathrm{y}_j = \mathbf{A}\mathrm{g}_j$ and will therefore shift the space spanned by $\{\mathrm{y}_j\}_{j=1}^k$ so that it is no longer aligned with the ideal space spanned by the k leading singular vectors of $\mathbf{A}$.

In consequence, the process described can (and frequently does) produce a poor basis. Luckily, there is a fix: Simply take a few extra samples. It turns out that if we take, say, $k + 10$ samples instead of k, then the process will with probability almost 1 produce a basis that is comparable to the best possible basis.

To summarize the discussion in the previous paragraph, the randomized sampling algorithm for constructing an approximate basis for the range of a given $m \times n$ matrix $\mathbf{A}$ proceeds as follows: First pick a small integer p representing how much "over-sampling" we do. (The choice $p = 10$ is often good.) Then execute the following steps:

1. Form a set of $k + p$ random Gaussian vectors $\{\mathrm{g}_j\}_{j=1}^{k+p}$.
2. Form a set $\{\mathrm{y}_j\}_{j=1}^{k+p}$ of samples from the range where $\mathrm{y}_j = \mathbf{A}\mathrm{g}_j$.
3. Perform Gram-Schmidt on the set $\{\mathrm{y}_j\}_{j=1}^{k+p}$ to form the ON-set $\{\mathrm{q}_j\}_{j=1}^{k+p}$.

Now observe that the $k + p$ matrix-vector products are independent and can advantageously be executed in parallel. A full algorithm for computing an approximate SVD using this simplistic sampling technique for executing "Stage A" is summarized in Figure 3.3.

The error incurred by the randomized range finding method described in this section is a random variable. There exist rigorous bounds for both the expectation of this error, and for the likelihood of a large deviation from the expectation. These bounds demonstrate that when the singular values of $\mathbf{A}$ decay "reasonably fast," the error incurred is close to the theoretically optimal one.

The question arises : How many basis vectors should someone use? As you have observed that while our stated goal was to find a matrix $\mathbf{Q}$ that holds k orthonormal columns, the randomized process discussed in this section and summarized in Figure 3.3 results in a matrix with $k+p$ columns instead.

The p extra vectors are needed to ensure that the basis produced in "Stage A" accurately captures the k dominant left singular vectors of $\mathbf{A}$. In a situation where an approximate SVD with precisely k modes is sought, one can drop the last p components when executing Stage B. Using programming laguange notation, we would after Step (5) run the commands

```
Uhat = Uhat(:,1:k); D = D(1:k,1:k); V = V(:,1:k);.
```

From a practical point of view, the cost of carrying around a few extra samples in the intermediate steps is often entirely negligible.

PROTO-ALGORITHM: SOLVING THE FIXED-RANK PROBLEM

Given an $m \times n$ matrix $\mathbf{A}$, a target rank k, and an oversampling parameter p, this procedure computes an $m \times (k+p)$ matrix $\mathbf{Q}$ whose columns are orthonormal and whose range approximates the range of $\mathbf{A}$.

1. Draw a random $n \times (k+p)$ test matrix $\mathbf{\Omega}$.
2. Form the matrix product $\mathbf{Y} = \mathbf{A\Omega}$.
3. Construct a matrix $\mathbf{Q}$ whose columns form an orthonormal basis for the range of $\mathbf{Y}$.

FIGURE 3.4: Prototype-Algorithm

A principal goal of this thesis is to provide a detailed analysis of the performance of the proto-algorithm described in figure 3.3.

This investigation produces precise error bounds, expressed in terms of the singular values of the input matrix. Furthermore, we determine how several choices of the random matrix $\mathbf{\Omega}$ impact the behavior of the algorithm.

Let us offer a taste of this theory. The following theorem describes the average-case behavior of the proto-algorithm with a Gaussian test matrix, assuming we perform the computation in exact arithmetic.

Theorem 3.3.1. *Suppose that $\mathbf{A}$ is a real $m \times n$ matrix. Select a target rank $k \geq 2$ and an oversampling parameter $p \geq 2$, where $k + p \leq \min\{m, n\}$. Execute the proto-algorithm with a standard Gaussian test matrix to obtain an $m \times (k+p)$ matrix $\mathbf{Q}$ with orthonormal columns. Then*

$$\mathbb{E}[||\mathbf{A} - \mathbf{QQ}^T\mathbf{A}||] \leq \left[1 + \frac{4\sqrt{k+p}}{p-1} \cdot \sqrt{\min\{m,n\}}\right] \sigma_{k+1}, \tag{3.4}$$

where $\mathbb{E}$ denotes expectation with respect to the random test matrix and σ_{k+1} is the $(k+1)$th singular value of $\mathbf{A}$.

We recall that the term σ_{k+1} is the smallest possible error achievable with any basis matrix $\mathbf{Q}$.

The theorem asserts that, on average, the algorithm produces a basis whose error lies within a small polynomial factor of the theoretical minimum. Moreover, the error bound in the randomized algorithm is slightly sharper than comparable bounds for deterministic techniques based on rank-revealing QR algorithms.

The probability that the error satisfies :

$$||\mathbf{A} - \mathbf{Q}\mathbf{Q}^T\mathbf{A}|| \leq \left[1 + 11\sqrt{k+p} \cdot \sqrt{\min\{m,n\}}\right] \sigma_{k+1}, \tag{3.5}$$

is at least $1 - 6 \cdot p^{-p}$ under very mild assumptions on p. This fact justifies the use of an oversampling term as small as $p = 5$. The theory developed in this thesis provides much more detailed information about the performance of the proto-algorithm.

3.4 Single pass algorithms

The randomized algorithm described in Figure 3.3 accesses the matrix **A** twice, first in “Stage A” where we build an orthonormal basis for the column space, and then in “Stage B” where we project **A** on to the space spanned by the computed basis vectors. It turns out to be possible to modify the algorithm in such a way that each entry of **A** is accessed only *once.* This is important because it allows us to compute the factorization of a matrix that is too large to be stored.

For *Hermitian* matrices, the modification to Algorithm 3.3 is very minor and we describe it in Section 3.4.2. Section 3.4.4 then handles the case of a general matrix. Loss of accuracy The single-pass algorithms described in this section tend to produce a factorization of lower accuracy than what Algorithm 3.3 would yield. In situations where one has a choice between using either a one-pass or a two-pass algorithm, the latter is generally preferable since it yields higher accuracy, at only moderately higher cost.

Streaming Algorithms We say that an algorithm for processing a matrix is a *streaming algorithm* if each entry of the matrix is accessed only once, and if, in addition, it can be fed the entries in any order. (In other words, the algorithm is not allowed to dictate the order in which the elements are viewed.) The algorithms described in this section satisfy both of these conditions.

3.4.1 Simple Single Pass Algorithm

The idea here is that we must "visit" the primary matrix **A** one time (single pass) and no more.In order to do that in a simple form we introduce the following algorithm:

```
ALGORITHM:SIMPLE SINGLE-PASS RANDOMIZED FOR $m \times n$ MATRIX

Input: An $m \times n$ matrix $\mathbf{A}$, a target rank $k$, and an over-sampling parameter $l$ (we can take $l = k$).

Output: Matrix $\hat{\mathbf{A}}$ which is an approximate $k$ rank of the original matrix $\mathbf{A}$ with the same dimensions.

Stage A:
    1. Form an $n \times k$ standardized Gaussian random matrix $\mathbf{\Omega}$.        $n \times k$
    2. Form an $l \times k$ standardized Gaussian random matrix $\mathbf{\Psi}$.        $l \times m$
    3. Form the sample matrix $\mathbf{Y} = \mathbf{A\Omega}$.        $m \times n \cdot n \times k = m \times k$
    4. Form the sample matrix $\mathbf{W} = \mathbf{\Psi A}$.        $l \times m \cdot m \times n = l \times n$
Stage B:
    4. Orthonormalize the sketch matrix $\mathbf{Y}$ with $\mathbf{QR}$ decomposition.Keep $\mathbf{Q}$ and discard
       the upper triangular matrix $\mathbf{R}$.        $m \times k$
    5. Solve a least-squares problem to obtain: $\mathbf{X} := (\mathbf{\Psi Q})^{\dagger}\mathbf{W}$        $k \times n$
    6. Construct the rank-k approximation : $\hat{\mathbf{A}} = \mathbf{QX}$        $m \times k \cdot k \times n = m \times n$
```

FIGURE 3.5: A basic randomized algorithm single-pass algorithm suitable for a Hermitian matrix.

The total cost of this computation is $\mathcal{O}(kl(m+n))$ flops (floating operations). For example the Cat gray scale photo is a matrix 596×738

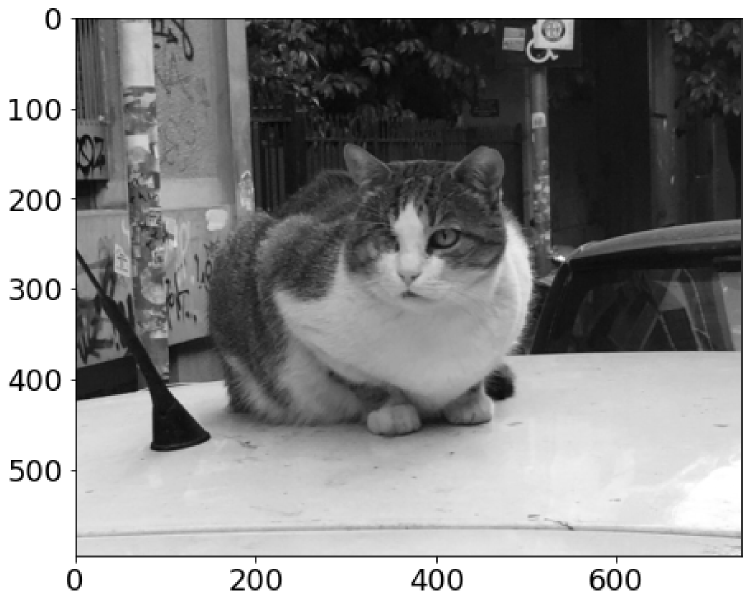


FIGURE 3.6: Original gray scale photo

After implementing the Single Pass algorithm with $k = l = 400$ we obtain :

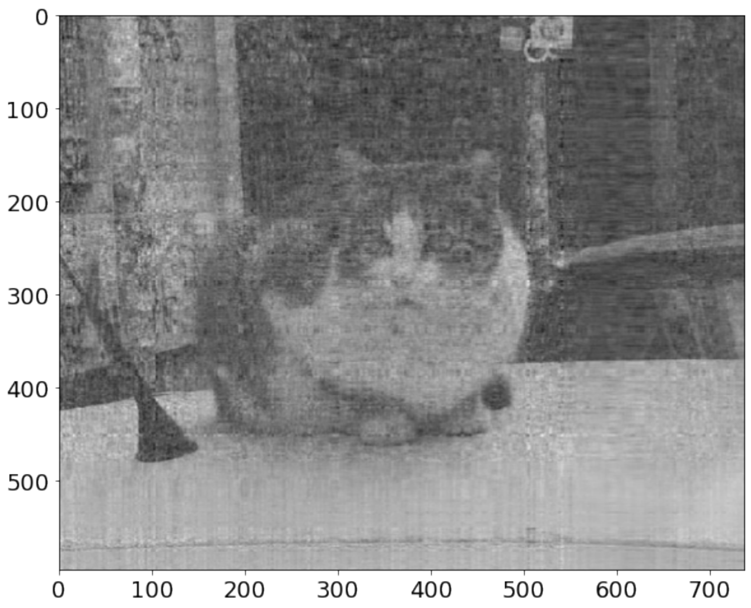


FIGURE 3.7: Compressed image after single pass algorithm approximation with $k = 100, p = 10$

the expected error is :

$$\mathbb{E}\left[\frac{||A - \hat{A}||_F}{||A||_F}\right] \approx 0.13$$

But in addition we have implemented the algorithm for various decreasing k values with fixed l parameter $p = 10$ resulting to $l = k + p = 110$ for example in the first algorithm.

The image results that appear in Figure 3.6 are characteristic of the choice of k parameter, because when we are shortening the value of k parameter the clearance of the image is getting "polluted".

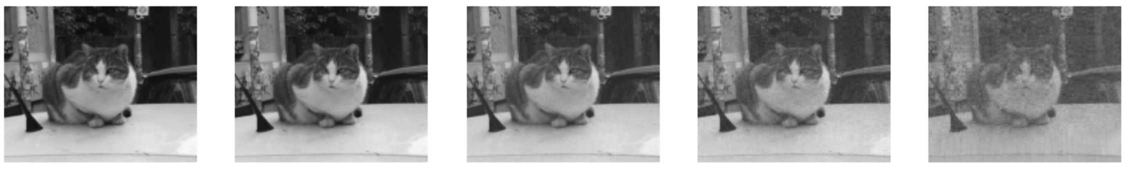


FIGURE 3.8: From left to right the $k = [500, 400, 300, 200, 100]$

The table shows the errors of the five choices of k and the error is getting bigger as we decrease k:

k,p	Errors
k = 500,p=10	0.70422
k = 400,p=10	1.82406
k = 300,p=10	3.79406
k = 200,p=10	7.55797
k = 100,p=10	15.92835

And this result comes from the establishment of the Low-Rank Approximation with Frobenius norm error bound according to Tropp et al 2016.

They have defined the function :

$$f(s,t) = \frac{s}{t - s - a},$$

for integers that satisfy $t > s + a > a$ where ,

$$a = \left\{ \begin{array}{ll} 1, & \text{for } \mathbb{R} \\ 0, & \text{for } \mathbb{C} \end{array} \right\},$$

and since we are dealing only with real matrices in this subsection we set $a = 1$.

Assuming the sketch size parameters satisfy $k > r + a$ and $l > k + a$. Draw random test matrices $\Omega \in \mathbb{R}^{n \times k}$ and $\Psi \in \mathbb{R}^{l \times m}$ independently from the standard normal distribution. Then the rank-k approximation $\hat{A}$ obtained from formula

$$\hat{\mathbf{A}} = \mathbf{QX},$$

satisfies:

$$\mathbb{E}||\mathbf{A} - \hat{\mathbf{A}}||_F^2 \leq (1 + f(r,k))(1 + f(k,l)) \cdot ||\mathbf{A} - \mathbf{A_r}||_F^2.$$

The error bound of the sketch matrix $\mathbf{Y} = \mathbf{A\Omega}$ is precise enough to predict the actual performance of the approximation $\hat{\mathbf{A}} = \mathbf{QX}$. As a consequence, we can use the result to make a priori decisions about the best sketch size parameters (k,l).

To appreciate the meaning of matrix $\mathbf{Y}$, it is helpful to consider specific choices for the sketch size parameters. First, notice that the selection

$$k = 2r + a \quad \text{and} \quad l = 2k + a,$$

yields the error bound:

$$\sqrt{\mathbb{E}||\mathbf{A} - \hat{\mathbf{A}}||_F^2} \leq 2 \cdot ||\mathbf{A} - \mathbf{A_r}||_F.$$

The approximation error decomposes as :

$$||\mathbf{A} - \hat{\mathbf{A}}||_F^2 = ||\mathbf{A} - \mathbf{QQ^T A}||_F^2 + ||\mathbf{X} - \mathbf{Q^T A}||_F^2.$$

Proof : Recall that $\hat{\mathbf{A}} = \mathbf{QX}$ and then we have :

$$\begin{aligned} ||\mathbf{A} - \mathbf{QX}||_F^2 &= ||\mathbf{I} - \mathbf{QQ^T})\mathbf{A} + \mathbf{Q}(\mathbf{Q^T A} - \mathbf{X})||_F^2 \\ &= ||\mathbf{I} - \mathbf{QQ^T})\mathbf{A}||_F^2 + ||\mathbf{Q}(\mathbf{Q^T A} - \mathbf{X})||_F^2 \\ &= ||\mathbf{A} - \mathbf{QQ^T A}||_F^2 + ||\mathbf{Q^T A} - \mathbf{X})||_F^2. \end{aligned}$$

The second line follows from the Pythagorean theorem, which is valid because the ranges of $\mathbf{I} - \mathbf{QQ^T}$ and $\mathbf{Q}$ are orthogonal. The last identity holds because $\mathbf{Q}$ has orthonormal columns and the Frobenius norm is invariant under unitary transformations.

3.4.2 Single Pass Algorithm for Hermitian matrices

Suppose that $\mathbf{A} = \mathbf{A}^*$, and that our objective is to compute an approximate eigenvalue decomposition

$$\begin{array}{ccccc} \mathbf{A} & \approx & \mathbf{U} & \mathbf{D} & \mathbf{U}^* \\ n \times n & & n \times k & k \times k & k \times n, \end{array} \tag{3.6}$$

with $\mathbf{U}$ an orthonormal matrix and $\mathbf{D}$ diagonal. (Note that for a Hermitian matrix, the EigenValue Decomposition and the Singular Value Decomposition are essentially equivalent, and that the EVD is the more natural factorization.)

Then execute Stage A with an over-sampling parameter p to compute an orthonormal matrix $\mathbf{Q}$ whose columns form an approximate basis for the column space of $\mathbf{A}$:

1. Draw a Gaussian random matrix $\mathbf{G}$ of size $n \times (k+p)$.
2. Form the sampling matrix $\mathbf{Y} = \mathbf{AG}$.
3. Orthonormalize the columns of $\mathbf{Y}$ to form $\mathbf{Q}$, in other words $\mathbf{Q} = \texttt{orth}(\mathbf{Y})$.

Then

$$\mathbf{A} \approx \mathbf{QQ}^*\mathbf{A}. \tag{3.7}$$

Since $\mathbf{A}$ is Hermitian, its row and column spaces are identical, so we also have

$$\mathbf{A} \approx \mathbf{AQQ}^*. \tag{3.8}$$

Inserting (3.7) into (3.8), we (informally!) find that

$$\mathbf{A} \approx \mathbf{QQ}^*\mathbf{AQQ}^*. \tag{3.9}$$

We define

$$\mathbf{C} = \mathbf{Q}^*\mathbf{AQ}. \tag{3.10}$$

If $\mathbf{C}$ is known, then the post-processing is straight-forward: Simply compute the EVD of $\mathbf{C}$ to obtain $\mathbf{C} = \hat{\mathbf{U}}\mathbf{D}\hat{\mathbf{U}}^*$, then define $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$, to find that

$$\mathbf{A} \approx \mathbf{QCQ}^* = \mathbf{Q}\hat{\mathbf{U}}\mathbf{D}\hat{\mathbf{U}}^*\mathbf{Q}^* = \mathbf{UDU}^*.$$

The problem now is that since we are seeking a single-pass algorithm, we are not in position to evaluate $\mathbf{C}$ directly from formula (3.10). Instead, we will derive a formula for $\mathbf{C}$ that can be evaluated without revisiting $\mathbf{A}$. To this end, multiply (3.10) by $\mathbf{Q}^*\mathbf{G}$ to obtain

$$\mathbf{C}(\mathbf{Q}^*\mathbf{G}) = \mathbf{Q}^*\mathbf{AQQ}^*\mathbf{G}. \tag{3.11}$$

We use that $\mathbf{AQQ}^* \approx \mathbf{A}$ (cf. (3.8)), to approximate the right hand side in (3.11):

$$\mathbf{Q}^*\mathbf{AQQ}^*\mathbf{G} \approx \mathbf{Q}^*\mathbf{AG} = \mathbf{Q}^*\mathbf{Y}. \tag{3.12}$$

Combining, (3.11) and (3.12), and ignoring the approximation error, we define $\mathbf{C}$ as the solution of the linear system (recall that $\ell = k+p$)

$$\underset{\ell \times \ell}{\mathbf{C}} \quad \underset{\ell \times \ell}{(\mathbf{Q}^*\mathbf{G})} \quad = \quad \underset{\ell \times \ell}{(\mathbf{Q}^*\mathbf{Y})}. \tag{3.13}$$

At first, it may appears that (3.13) is perfectly balanced in that there are ℓ^2 equations for ℓ^2 unknowns.

However, we need to enforce that $\mathbf{C}$ is Hermitian, so the system is actually over-determined by roughly a factor of two. Putting everything together, we obtain the method summarized in Figure 3.9.

```
ALGORITHM: SINGLE-PASS RANDOMIZED EIGEN VALUE DECOMPOSITION FOR A HERMITIAN MATRIX

Inputs: An $n \times n$ Hermitian matrix $\mathbf{A}$, a target rank $k$, and an over-sampling parameter $p$ (say $p = 10$).
Outputs: Matrices $\mathbf{U}$ and $\mathbf{D}$ in an approximate rank-$k$ EVD of $\mathbf{A}$ (so that $\mathbf{U}$ is an orthonormal $n \times k$ matrix, $\mathbf{D}$ is a diagonal $k \times k$ matrix, and $\mathbf{A} \approx \mathbf{U}\mathbf{D}\mathbf{U}^*$).
Stage A:
    1. Form an $n \times (k+p)$ Gaussian random matrix $\mathbf{G}$.
    2. Form the sample matrix $\mathbf{Y} = \mathbf{A}\,\mathbf{G}$.
    3. Let $\mathbf{Q}$ denote the orthonormal matrix formed by the $k$ dominant left singular vectors of $\mathbf{Y}$.
Stage B:
    4. Let $\mathbf{C}$ denote the $k \times k$ least squares solution of $\mathbf{C}\left(\mathbf{Q}^*\mathbf{G}\right) = \left(\mathbf{Q}^*\mathbf{Y}\right)$ obtained by enforcing that $\mathbf{C}$ should be Hermitian.
    5. Compute that eigenvalue decomposition of $\mathbf{C}$ so that $\mathbf{C} = \hat{\mathbf{U}}\mathbf{D}\hat{\mathbf{U}}^*$.
    6. Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.
```

FIGURE 3.9: A basic randomized algorithm single-pass algorithm suitable for a Hermitian matrix.

The procedure described in this section is less accurate than the procedure described in Figure 3.3 for two reasons:

1. The approximation error in formula (3.9) tends to be larger than the error in (3.7).
2. While the matrix $\mathbf{Q}^*\mathbf{G}$ is invertible, it tends to be very ill-conditioned.

3.4.3 Extra over-sampling

To combat the problem that $\mathbf{Q}^*\mathbf{G}$ tends to be ill-conditioned, it is helpful to over-sample more aggressively when using a single pass algorithm, even to the point of setting $p = k$ if memory allows.

Once the sampling stage is completed, we form $\mathbf{Q}$ as the leading k left singular vectors of $\mathbf{Y}$ (compute these by forming the full SVD of $\mathbf{Y}$, and then discard the last p components). Then $\mathbf{C}$ will be of size $k \times k$, and the equation that specifies $\mathbf{C}$ reads

$$\underset{k \times k}{\mathbf{C}} \quad \underset{k \times \ell}{(\mathbf{Q}\mathbf{G})} \quad = \quad \underset{k \times \ell}{\mathbf{Q}^*\mathbf{Y}.} \tag{3.14}$$

Since (3.14) is over-determined, we solve it using a least-squares technique. Observe that we are now looking for less information (a $k \times k$ matrix rather than an $\ell \times \ell$ matrix), and have more information in order to determine it.

3.4.4 Single Pass Algorithm for General matrices

We next consider a general $m \times n$ matrix $\mathbf{A}$. In this case, we need to apply randomized sampling to both its row space and its column space simultaneously. We proceed as follows:

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

3. Compute two basis matrices $\mathbf{Q}_\mathrm{c} = \texttt{orth}(\mathbf{Y}_\mathrm{c})$ and $\mathbf{Q}_\mathrm{r} = \texttt{orth}(\mathbf{Y}_\mathrm{r})$.

Now define the small projected matrix via

$$\mathbf{C} = \mathbf{Q}_\mathrm{c}^* \mathbf{A} \mathbf{Q}_r. \tag{3.15}$$

We will derive two relationships that together will determine $\mathbf{C}$ in a manner that is analogous to (3.11). First left multiply (3.15) by $\mathbf{G}_\mathrm{r}^* \mathbf{Q}_\mathrm{c}$ to obtain

$$\mathbf{G}_\mathrm{r}^* \mathbf{Q}_\mathrm{c} \mathbf{C} = \mathbf{G}_\mathrm{r}^* \mathbf{Q}_\mathrm{c} \mathbf{Q}_\mathrm{c}^* \mathbf{A} \mathbf{Q}_\mathrm{r} \approx \mathbf{G}_\mathrm{r}^* \mathbf{A} \mathbf{Q}_\mathrm{r} = \mathbf{Y}_\mathrm{r}^* \mathbf{Q}_\mathrm{r}. \tag{3.16}$$

Next we right multiply (3.15) by $\mathbf{Q}_\mathrm{r}^* \mathbf{G}_\mathrm{c}$ to obtain

$$\mathbf{C} \mathbf{Q}_\mathrm{r}^* \mathbf{G}_\mathrm{c} = \mathbf{Q}_\mathrm{c}^* \mathbf{A} \mathbf{Q}_r \mathbf{Q}_\mathrm{r}^* \mathbf{G}_\mathrm{c} \approx \mathbf{Q}_\mathrm{c}^* \mathbf{A} \mathbf{G}_\mathrm{c} = \mathbf{Q}_\mathrm{c}^* \mathbf{Y}_\mathrm{c}. \tag{3.17}$$

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

$$\left(\mathbf{G}_\mathrm{r}^* \mathbf{Q}_\mathrm{c}\right) \mathbf{C} = \mathbf{Y}_\mathrm{r}^* \mathbf{Q}_\mathrm{r} \qquad \text{and} \qquad \mathbf{C} \left(\mathbf{Q}_\mathrm{r}^* \mathbf{G}_\mathrm{c}\right) = \mathbf{Q}_\mathrm{c}^* \mathbf{Y}_\mathrm{c}.$$

Again, the system is over-determined by about a factor of 2, and it is advantageous to make it further over-determined by more aggressive over-sampling, cf. Remark **??**. Figure 3.10 summarizes the single-pass method for a general matrix.

ALGORITHM: SINGLE-PASS RANDOMIZED SVD FOR A GENERAL MATRIX

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

Outputs: Matrices $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{D}$ in an approximate rank-k SVD of $\mathbf{A}$ (so that $\mathbf{U}$ and $\mathbf{V}$ are orthonormal with k columns each, $\mathbf{D}$ is diagonal, and $\mathbf{A} \approx \mathbf{U}\mathbf{D}\mathbf{V}^*$.)

Stage A:

1. Form two Gaussian random matrices $\mathbf{G}_\mathrm{c}$ and $\mathbf{G}_\mathrm{r}$ of sizes $n \times (k+p)$ and $m \times (k+p)$, respectively.
2. Form the sample matrices $\mathbf{Y}_\mathrm{c} = \mathbf{A}\,\mathbf{G}_\mathrm{c}$ and $\mathbf{Y}_\mathrm{r} = \mathbf{A}^*\,\mathbf{G}_\mathrm{r}$.
3. Form orthonormal matrices $\mathbf{Q}_\mathrm{c}$ and $\mathbf{Q}_\mathrm{r}$ consisting of the k dominant left singular vectors of $\mathbf{Y}_\mathrm{c}$ and $\mathbf{Y}_\mathrm{r}$.

Stage B:

4. Let $\mathbf{C}$ denote the $k \times k$ least squares solution of the joint system of equations formed by $\left(\mathbf{G}_\mathrm{r}^* \mathbf{Q}_\mathrm{c}\right) \mathbf{C} = \mathbf{Y}_\mathrm{r}^* \mathbf{Q}_\mathrm{r}$ and $\mathbf{C} \left(\mathbf{Q}_\mathrm{r}^* \mathbf{G}_\mathrm{c}\right) = \mathbf{Q}_\mathrm{c}^* \mathbf{Y}_\mathrm{c}$.
5. Compute the SVD of $\mathbf{C}$ so that $\mathbf{C} = \hat{\mathbf{U}}\mathbf{D}\hat{\mathbf{V}}^*$.
6. Form $\mathbf{U} = \mathbf{Q}_\mathrm{c}\hat{\mathbf{U}}$ and $\mathbf{V} = \mathbf{Q}_\mathrm{r}\hat{\mathbf{V}}$.

FIGURE 3.10: A basic randomized algorithm single-pass algorithm suitable for a general matrix.

3.5 A method with complexity $\mathcal{O}(mn \log k)$ for general dense matrices

The Randomized SVD (RSVD) algorithm as given in Figure 3.3 is highly efficient when we have access to fast methods for evaluating matrix-vector products $\mathrm{x} \mapsto \mathbf{A}\mathrm{x}$.

For the case where $\mathbf{A}$ is a general $m \times n$ matrix given simply as an array or real numbers, the cost of evaluating the sample matrix $\mathbf{Y} = \mathbf{A}\mathbf{G}$ (in Step (2) of the algorithm in Figure 3.3) is

$O(mnk)$.

RSVD is still often faster than classical methods since the matrix-matrix multiply can be highly optimized, but it does not have an edge in terms of asymptotic complexity.

However, it turns out to be possible to modify the algorithm by replacing the Gaussian random matrix $\mathbf{G}$ with a different random matrix $\mathbf{\Omega}$ that has two seemingly contradictory properties:

1. $\mathbf{\Omega}$ is sufficiently *structured* that $\mathbf{A\Omega}$ can be evaluated in $O(mn\log(k))$ flops;

2. $\mathbf{\Omega}$ is sufficiently *random* that the columns of $\mathbf{A\Omega}$ accurately span the range of $\mathbf{A}$.

For instance, a good choice of random matrix $\mathbf{\Omega}$ is :

$$\underset{n\times\ell}{\mathbf{\Omega}} = \underset{n\times n}{\mathbf{D}}\;\underset{n\times n}{\mathbf{F}}\;\underset{n\times\ell}{\mathbf{S}}, \tag{3.18}$$

where , $\mathbf{D}$ is a diagonal matrix whose diagonal entries are complex numbers of modulus one drawn from a uniform distribution on the unit circle in the complex plane, where $\mathbf{F}$ is the discrete Fourier transform,

$$\mathbf{F}(p,q) = n^{-1/2}\,e^{-2\pi \mathrm{i}(p-1)(q-1)/n}, \qquad p,q \in \{1,2,3,\ldots,n\},$$

and where , $\mathbf{S}$ is a matrix consisting of a random subset of ℓ columns from the $n \times n$ unit matrix (drawn without replacement). In other words, given an arbitrary matrix $\mathbf{X}$ of size $m \times n$, the matrix $\mathbf{XS}$ consists of a randomly drawn subset of ℓ columns of $\mathbf{X}$.

For the matrix $\mathbf{\Omega}$ specified by (3.18), the product $\mathbf{X\Omega}$ can be evaluated via a subsampled FFT in $\mathcal{O}(mn\log(\ell))$ operations. The parameter ℓ should be chosen slightly larger than the target rank k; the choice $\ell = 2k$ is often good.

By using the structured random matrix described in this section, we can reduce the complexity of "Stage A" in the RSVD from $\mathcal{O}(mnk)$ to $\mathcal{O}(mn\log(k))$. In order to attain overall cost $\mathcal{O}(mn\log(k))$, we must also modify "Stage B" to eliminate the need to compute $\mathbf{Q}^*\mathbf{A}$ (since direct evaluation of $\mathbf{Q}^*\mathbf{A}$ has cost $\mathcal{O}(mnk)$).

One option is to use the single pass algorithm described in 3.10, using the structured random matrix to approximate both the row and the column spaces of $\mathbf{A}$. Our theoretical understanding of the errors incurred by the accelerated range finder is not as satisfactory as what we have for Gaussian random matrices.

In the general case, only quite weak results have been proven. In practice, the accelerated scheme is often as accurate as the Gaussian one, but we do not currently have good theory to predict precisely when this happens.

3.6 Theoretical performance bounds

In this section, we will briefly summarize some proven results concerning the error in the output of the basic RSVD algorithm in Figure 3.3. Observe that the factors $\mathbf{U}$, $\mathbf{D}$, $\mathbf{V}$ depend not only on $\mathbf{A}$, but also on the draw of the random matrix $\mathbf{G}$.

This means that the error that we try to bound is a *random variable.* It is therefore natural to seek bounds on first the expected value of the error, and then on the likelihood of large deviations from the expectation.

Before we start, let us recall :

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

that all the error incurred by the RSVD algorithm in Figure 3.3 is incurred in Stage A.

The reason is that the "post-processing" in Stage B is exact (up to floating point arithmetic). Consequently, we can (and will) restrict ourselves to providing bounds on $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|$. The theoretical investigation of errors resulting from randomized methods in linear algebra is an active area of research that draws heavily on random matrix theory, theoretical computer science, classical numerical linear algebra, and many other fields.

Our objective here is merely to state a couple of representative results, without providing any proofs or details about their derivation. Both results are taken from Martinsson 2011 random survey and Martinsson 2019, where the interested reader can find an in-depth treatment of the subject.

3.6.1 Bounds on the expectation of the error

A basic result on the *typical* error observed is Theorem 10.6 of Martinsson 2011 random survey, which states:

Theorem 1. *Let* $\mathbf{A}$ *be an* $m \times n$ *matrix with singular values* $\{\sigma_j\}_{j=1}^{\min(m,n)}$.

Let k *be a target rank, and let* p *be an over-sampling parameter such that* $p \geq 2$ *and* $k + p \leq \min(m, n)$. *Let* $\mathbf{G}$ *be a Gaussian random matrix of size* $n \times (k + p)$ *and set* $\mathbf{Q} = \texttt{orth}(\mathbf{AG})$.

Then the average error, as measured in the Frobenius norm, satisfies

$$\mathbb{E}\left[\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|_{\mathrm{Fro}}\right] \leq \left(1 + \frac{k}{p-1}\right)^{1/2} \left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2\right)^{1/2}, \tag{3.19}$$

where $\mathbb{E}$ *refers to expectation with respect to the draw of* $\mathbf{G}$*.Matrix* $\mathbf{G}$ *is random variable, and thus all the properties of a random variable hold for this matrix such as a multiplication by a constant the result is also a random variable.*

According to Halko et al 2011 and Martinsson et al 2019

$$\mathbb{E}\left[\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|_{\mathrm{Fro}}\right] \xrightarrow{a.s} \|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|_{\mathrm{Fro}}$$

almost sure by the Strong Law of Large Numbers that $\mathbf{E}[\mathbf{X}] \xrightarrow{a.s} \mathbf{X}$. *Specifically:*

Let $\mathbf{X_1}, \mathbf{X_2}, \ldots, \mathbf{X_n}$ *be i.i.d random varibles with a finite expected value* $\mathbb{E}[X_i] = \mu < +\infty$.

Let also :

$$\mathbf{M_n} = \frac{X_1, X_2, \ldots, X_n}{n},$$

Then:

$$\mathbf{M_n} \xrightarrow{a.s} \mu.$$

Because the test matrix $\mathbf{G}$ *or* $\boldsymbol{\Omega}$*, as denoted on this thesis, are simulated pseudo numbers so they obey the SLLN.*
In addition the corresponding result for the spectral norm reads

$$\mathbb{E}\left[\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|\right] \leq \left(1 + \sqrt{\frac{k}{p-1}}\right)\sigma_{k+1} + \frac{e\sqrt{k+p}}{p}\left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2\right)^{1/2}. \tag{3.20}$$

When errors are measured in the *Frobenius norm,* the above Theorem is very gratifying.The standard recommendation found in literature is that of $p = 10$, which we are basically within a factor of $\sqrt{1 + k/9}$ of the theoretically minimal error.

(Recalling to the reader that the Eckart-Young theorem states that $\left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2\right)^{1/2}$ is a lower bound on the residual for any rank-k approximant.)

If we over-sample more aggressively and set for instance $p = k + 1$, then we are within a distance of $\sqrt{2}$ of the theoretically minimal error. When errors are measured in the *spectral norm,* the situation is much less promising. The first term in the bound in (3.20) is perfectly acceptable, but the second term is unfortunate in that, it involves the minimal error in the Frobenius norm, which can be much larger, especially when m or n are large. The theorem is quite sharp, as it turns out, so the sub-optimality expressed in (3.20) reflects a true limitation on the accuracy to be expected from the basic randomized scheme.

The extent to which the error in (3.20) is problematic depends on how rapidly the "tail" singular values $\{\sigma_j\}_{j=1}^{\min(m,n)}$ decay. If they decay fast, then the spectral norm error and the Frobenius norm error are similar, and the RSVD works well. If they decay slowly, then the RSVD performs fine when errors are measured in the Frobenius norm, but not very well when the spectral norm is the one of interest. To illustrate the difference, let us consider two situations:

1. Case 1) — fast decay: Suppose that the tail singular values decay exponentially fast, so that for some $\beta \in (0, 1)$ we have $\sigma_j \approx \sigma_{k+1}\,\beta^{j-k-1}$ for $j > k$. Then $\left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2\right)^{1/2} \approx \sigma_{k+1}\left(\sum_{j=k+1}^{\min(m,n)} \beta^{2(j-k-1)}\right)^{1/2} \leq \sigma_{k+1}(1-\beta^2)^{-1/2}$. As long as β is not very close to 1, we see that the contribution from the tail singular values is modest in this case.
2. Case 2) — no decay: Suppose that the tail singular values exhibit *no* decay, so that $\sigma_j = \sigma_{k+1}$ for $j > k$. This represents the worst case scenario, and now $\left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2\right)^{1/2} = \sigma_{k+1}\sqrt{\min(m,n) - k}$. Since we want to allow for n and m to be very large, this represents devastating suboptimality.

Fortunately, it is possible to modify the RSVD algorithm in such a way that the errors produced are close to optimal in both the spectral and the Frobenius norms. The price to pay is a modest increase in the computational cost. The error estimates given in this section are very satisfactory in the case where the singular values decay rapidly.

3.6.2 Bounds on the likelihood of large deviations

One can prove that the likelihood of a large deviation from the mean depends only on the over-sampling parameter p, and decays extraordinarily fast. For instance, one can prove that if $p \geq 4$, then

$$||\mathbf{A} - \mathbf{QQ}^*\mathbf{A}|| \leq \left(1 + 17\sqrt{1 + k/p}\right)\sigma_{k+1} + \frac{8\sqrt{k+p}}{p+1}\left(\sum_{j>k}\sigma_j^2\right)^{1/2}, \tag{3.21}$$

For instance, if we have exponential decay in the tail singular values, $\sigma_j \sim \sigma_{k+1}\,\alpha^{j-k-1}$ for $j > k$, then the sums in (3.19) and (3.21) are close to the minimal error σ_{k+1} (since $(\sum_{j>k}\sigma_j^2)^{1/2} \sim \sigma_{k+1}\,(1-\alpha^2)^{-1/2}$).

3.7 An accuracy enhanced randomized scheme or sub iterations scheme

3.7.1 The key idea — power iteration

We saw in Section 3.6 that the basic randomized scheme (see, e.g., Figure 3.3) gives accurate results for matrices whose singular values decay rapidly, but tends to produce suboptimal results when they do not.

To recap, suppose that we compute a rank-k approximation to an $m \times n$ matrix $\mathbf{A}$ with singular values $\{\sigma_j\}_{j=1}^{\min(m,n)}$. The theory shows that the error measured in the spectral norm is bounded only by a factor that scales with $\left(\sum_{j>k} \sigma_j^2\right)^{1/2}$. When the singular values decay slowly, this quantity can be much larger than the theoretically minimal approximation error (which is σ_{k+1}). Recall that the objective of the randomized sampling is to construct a set of orthonormal vectors $\{\mathrm{q}_j\}_{j=1}^{\ell}$ that capture to high accuracy the space spanned by the k dominant left singular vectors $\{\mathrm{u}_j\}_{j=1}^{k}$ of $\mathbf{A}$. The idea is now to sample not $\mathbf{A}$, but the matrix $\mathbf{A}^{(q)}$ defined by

$$\mathbf{A}^{(q)} = \left(\mathbf{A}\mathbf{A}^*\right)^q \mathbf{A},$$

where q is a small positive integer (say, $q = 1$ or $q = 2$). A simple calculation shows that if $\mathbf{A}$ has the SVD $\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^*$, then the SVD of $\mathbf{A}^{(q)}$ is

$$\mathbf{A}^{(q)} = \mathbf{U}\,\mathbf{D}^{2q+1}\,\mathbf{V}^*.$$

In other words, $\mathbf{A}^{(q)}$ has the same left singular vectors as $\mathbf{A}$, while its singular values are $\{\sigma_j^{2q+1}\}_j$. Even when the singular values of $\mathbf{A}$ decay slowly, the singular values of $\mathbf{A}^{(q)}$ tend to decay fast enough for our purposes. The accuracy enhanced scheme now consists of drawing a Gaussian matrix $\mathbf{G}$ and then forming a sample matrix

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

Then orthonormalize the columns of $\mathbf{Y}$ to obtain $\mathbf{Q} = \texttt{orth}(\mathbf{Y})$, and proceed as before. The resulting scheme is shown in Figure 3.11.

Remark. *The scheme described in Figure 3.11 can lose accuracy due to round-off errors. The problem is that as q increases, all columns in the sample matrix $\mathbf{Y} = \left(\mathbf{A}\mathbf{A}^*\right)^q \mathbf{A}\mathbf{G}$ tend to align closer and closer to the dominant left singular vector.*

This means that essentially all information about the singular values and singular vectors associated with smaller singular values get lots to round-off errors. Roughly speaking, if

$$\frac{\sigma_j}{\sigma_1} \le \epsilon_{\text{mach}}^{1/(2q+1)},$$

where ϵ_{mach} is machine precision, then all information associated with the j'th singular value and beyond is lost.

This problem can be fixed by orthonormalizing the columns between each iteration, as shown in Figure 3.12.

The modified scheme is more costly due to the extra calls to `orth`. (However, note that `orth` can be executed using unpivoted Gram-Schmidt, which is quite fast.)

3.7.2 Theoretical results

A detailed error analysis of the scheme described in Figure 3.11 is provided in Martinsson et al 2011. In particular, the key theorem states:

ALGORITHM: ACCURACY ENHANCED RANDOMIZED SVD

Inputs: An $m \times n$ matrix $\mathbf{A}$, a target rank k, an over-sampling parameter p (say $p = 10$), and a small integer q denoting the number of steps in the power iteration.
Outputs: Matrices $\mathbf{U}$, $\mathbf{D}$, and $\mathbf{V}$ in an approximate rank-$(k+p)$ SVD of $\mathbf{A}$. (I.e. $\mathbf{U}$ and $\mathbf{V}$ are orthonormal and $\mathbf{D}$ is diagonal.)

```
(1)   $\mathbf{G} = \texttt{randn}(n, k+p)$;
(2)   $\mathbf{Y} = \mathbf{A}\mathbf{G}$;
(3)   for $j = 1 : q$
(4)       $\mathbf{Z} = \mathbf{A}^*\mathbf{Y}$;
(5)       $\mathbf{Y} = \mathbf{A}\mathbf{Z}$;
(6)   end for
(7)   $\mathbf{Q} = \texttt{orth}(\mathbf{Y})$;
(8)   $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$;
(9)   $[\hat{\mathbf{U}}, \mathbf{D}, \mathbf{V}] = \texttt{svd}(\mathbf{B}, \texttt{'econ'})$;
(10)  $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$;
```

FIGURE 3.11: The accuracy enhanced randomized SVD. If a factorization of precisely rank k is desired, the factorization in Step 9 can be truncated to the k leading terms.

ALGORITHM: ACCURACY ENHANCED RANDOMIZED SVD
(WITH ORTHONORMALIZATION)

```
(1)   $\mathbf{G} = \texttt{randn}(n, k+p)$;
(2)   $\mathbf{Q} = \texttt{orth}(\mathbf{A}\mathbf{G})$;
(3)   for $j = 1 : q$
(4)       $\mathbf{W} = \texttt{orth}(\mathbf{A}^*\mathbf{Q})$;
(5)       $\mathbf{Q} = \texttt{orth}(\mathbf{A}\mathbf{W})$;
(6)   end for
(7)   $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$;
(8)   $[\hat{\mathbf{U}}, \mathbf{D}, \mathbf{V}] = \texttt{svd}(\mathbf{B}, \texttt{'econ'})$;
(9)   $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$;
```

FIGURE 3.12: This algorithm takes the same inputs and outputs as the method in Figure 3.11. The only difference is that orthonormalization is carried out between each step of the power iteration, to avoid loss of accuracy due to rounding errors.

Theorem 2. *Let* $\mathbf{A}$ *denote an* $m \times n$ *matrix, let* $p \geq 2$ *be an over-sampling parameter, and let* q *denote a small integer.*

Draw a Gaussian matrix $\mathbf{G}$ *of size* $n \times (k+p)$*, set* $\mathbf{Y} = (\mathbf{A}\mathbf{A}^*)^q\mathbf{A}\mathbf{G}$*, and let* $\mathbf{Q}$ *denote an* $m \times (k+p)$ *orthonormal matrix resulting from orthonormalizing the columns of* $\mathbf{Y}$*.*

Then:

$$\mathbb{E}\left[\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|\right] \leq \left[\left(1 + \sqrt{\frac{k}{p-1}}\right)\sigma_{k+1}^{2q+1} + \frac{e\sqrt{k+p}}{p}\left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^{2(2q+1)}\right)^{1/2}\right]^{1/(2q+1)}. \tag{3.22}$$

The bound in (3.22) is slightly opaque. To simplify it, let us consider a worst case scenario where there is no decay in the singular values beyond the truncation point, so that :

$$\sigma_{k+1} = \sigma_{k+2} = \cdots = \sigma_{\min(m,n)}.$$

Then: (3.22) simplifies to :

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

In other words, as we increase the exponent q, the power scheme drives the factor that multiplies σ_{k+1} to one exponentially fast. This factor represents the degree of "sub-optimality" you can expect to see.

3.8 The Nyström method for symmetric positive definite matrices

When the input matrix $\mathbf{A}$ is symmetric positive definite (spd), the *Nyström method* can be used to improve the quality of standard factorizations at almost no additional cost.

To describe the idea, we first recall from Section 3.4.2 that when $\mathbf{A}$ is Hermitian (which of course every spd matrix is), then it is natural to use the approximation

$$\mathbf{A} \approx \mathbf{Q}\left(\mathbf{Q}^*\mathbf{A}\mathbf{Q}\right)\mathbf{Q}^*. \tag{3.23}$$

In contrast, the so called "Nyström scheme" relies on the rank-k approximation

$$\mathbf{A} \approx (\mathbf{A}\mathbf{Q})\left(\mathbf{Q}^*\mathbf{A}\mathbf{Q}\right)^{-1}(\mathbf{A}\mathbf{Q})^*. \tag{3.24}$$

For both stability and computational efficiency, we typically rewrite (3.24) as

$$\mathbf{A} \approx \mathbf{F}\mathbf{F}^*,$$

where $\mathbf{F}$ is an approximate *Cholesky* factor of $\mathbf{A}$ of size $n \times k$, defined by

$$\mathbf{F} = (\mathbf{A}\mathbf{Q})\left(\mathbf{Q}^*\mathbf{A}\mathbf{Q}\right)^{-1/2}.$$

To compute the factor $\mathbf{F}$ numerically, first form the matrices $\mathbf{B}_1 = \mathbf{A}\mathbf{Q}$ and $\mathbf{B}_2 = \mathbf{Q}^*\mathbf{B}_1$. Observe that $\mathbf{B}_2$ is necessarily spd, which means that we can compute its Cholesky factorization $\mathbf{B}_2 = \mathbf{C}^*\mathbf{C}$. Finally compute the factor $\mathbf{F} = \mathbf{B}_1\mathbf{C}^{-1}$ by performing a triangular solve. The low-rank factorization (3.24) can be converted to a standard decomposition using the techniques from Section 3.2.

The Nyström technique for computing an approximate eigenvalue decomposition is given in Figure 3.13.

Let us compare the cost of this method to the more straight-forward method resulting from using the formula (3.23). In both cases, we need to twice apply $\mathbf{A}$ to a set of $k + p$ vectors (first in computing $\mathbf{AG}$, then in computing $\mathbf{AQ}$).

But the Nyström method tends to result in substantially more accurate results. Informally speaking, the reason is that by exploiting the spd property of $\mathbf{A}$, we can take one step of power iteration "for free."

ALGORITHM: EIGENVALUE DECOMPOSITION VIA THE NYSTRÖM METHOD

Given an $n \times n$ non-negative matrix $\mathbf{A}$, a target rank k and an over-sampling parameter p, this procedure computes an approximate eigenvalue decomposition $\mathbf{A} \approx \mathbf{U\Lambda U}^*$, where $\mathbf{U}$ is orthonormal, and $\mathbf{\Lambda}$ is nonnegative and diagonal.

```
1. Draw a Gaussian random matrix $\mathbf{G} = \texttt{randn}(n, k+p)$.
2. Form the sample matrix $\mathbf{Y} = \mathbf{AG}$.
3. Orthonormalize the columns of the sample matrix to obtain the basis matrix
   $\mathbf{Q} = \texttt{orth}(\mathbf{Y})$.
4. Form the matrices $\mathbf{B}_1 = \mathbf{AQ}$ and $\mathbf{B}_2 = \mathbf{Q}^*\mathbf{B}_1$.
5. Perform a Cholesky factorization $\mathbf{B}_2 = \mathbf{C}^*\mathbf{C}$.
6. Form $\mathbf{F} = \mathbf{B}_1\mathbf{C}^{-1}$ using a triangular solve.
7. Compute an SVD of the Cholesky factor $[\mathbf{U}, \mathbf{\Sigma}, \sim] = \texttt{svd}(\mathbf{F}, \texttt{'econ'})$.
8. Set $\mathbf{\Lambda} = \mathbf{\Sigma}^2$.
```

FIGURE 3.13: The Nyström method for low-rank approximation of self-adjoint matrices with non-negative eigenvalues. It involves two applications of $\mathbf{A}$ to matrices with $k + p$ columns, and has comparable cost to the basic RSVD in Figure 3.3. However, it exploits the symmetry of $\mathbf{A}$ to boost the accuracy.

3.8.1 Example for Nyström Method

For example the second derivative matrix of finite difference approximation is symmetric and positive definite. Let $\mathbf{A} \in \mathbb{R}^{m \times n}$ where $m, n = 300$

$$\mathbf{A} = \begin{pmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & -1 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -1 \\ 0 & \dots & 0 & -1 & 2 \end{pmatrix}$$

If we apply the Nyström method for eigenvalue decomposition and take the Σ of the last SVD decomposition and compare the singular values of this method with the standard economy SVD we will obtain the following plot in Figure 3.8.(See appendix for Python code implementation).

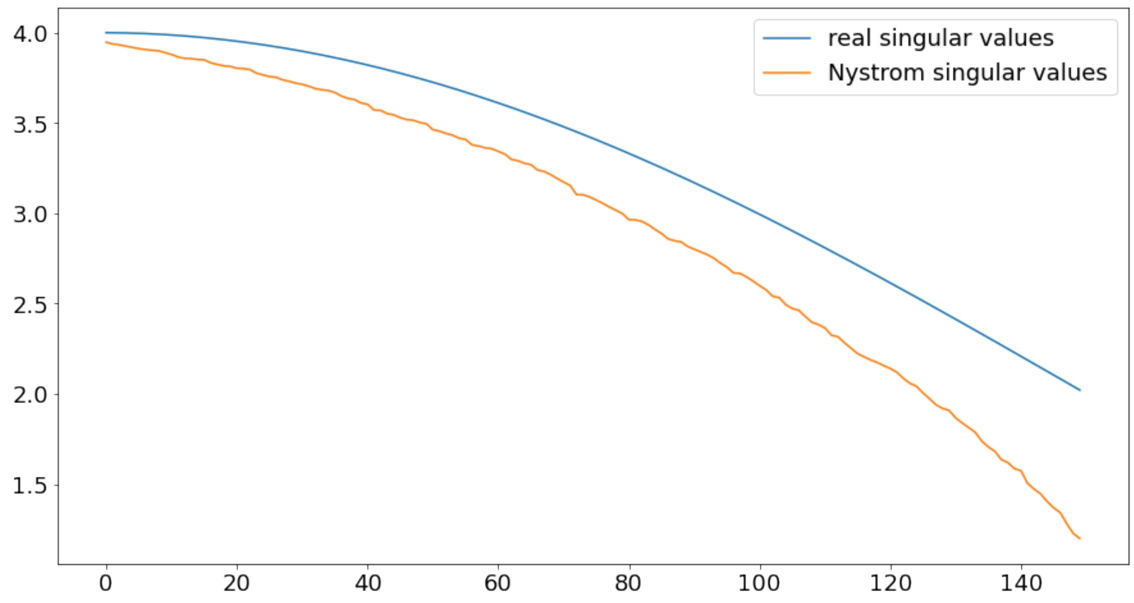


FIGURE 3.14: Nyström Method Versus SVD Decomposition comparison

3.9 Randomized SVD example: Image compression application in Python

The singular value decomposition can be used to obtain a low-rank approximation of high-dimensional data. Image compression is a simple, yet illustrative example. The underlying structure of natural images can often be represented by a very sparse model.

This means that images can be faithfully recovered from a relatively small set of basis functions. For demonstration, we use the following 1204×1880 apple grayscale image:

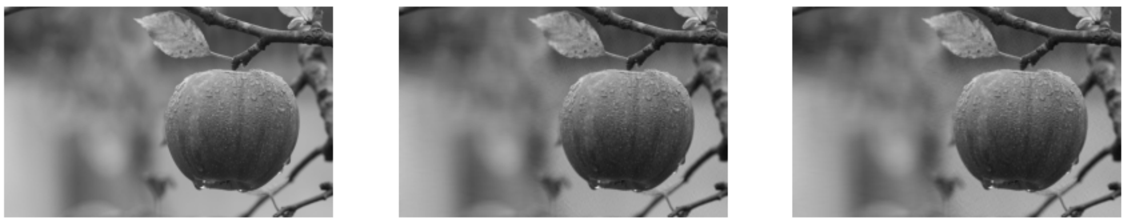


FIGURE 3.15: Original high-resolution (left) and rank-400 approximations from the SVD (middle) and rSVD (right)

A grayscale image may be thought of as a real-valued matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, where m and n are the number of pixels in the vertical and horizontal directions, respectively. To compress the image we need to first decompose the matrix $\mathbf{A}$.

The singular vectors and values provide a hierarchical representation of the image in terms of a new coordinate system defined by dominant correlations within rows and columns of the image.

Thus, the number of singular vectors used for approximation poses a trade-off between the compression rate (i.e., the number of singular vectors to be stored) and the reconstruction fidelity. In the present image example, we use the arbitrary choice $k = 100$ as target rank and choose different q values and observe how the singular values with each method decay in time.

The np.linalg.svd() function returns three objects: u, v and d. The first two objects are $m \times k$ and $n \times k$ arrays, namely the truncated left and right singular vectors. The vector d is comprised of the min m, n singular values in descending order.For full code implementation of this example see Appendix for Python link.

Now, the dominant $k = 100$ singular values are retained to approximate/reconstruct

$$\mathbf{A_k} := \mathbf{U_k D_k V_k^T}$$

the original image:

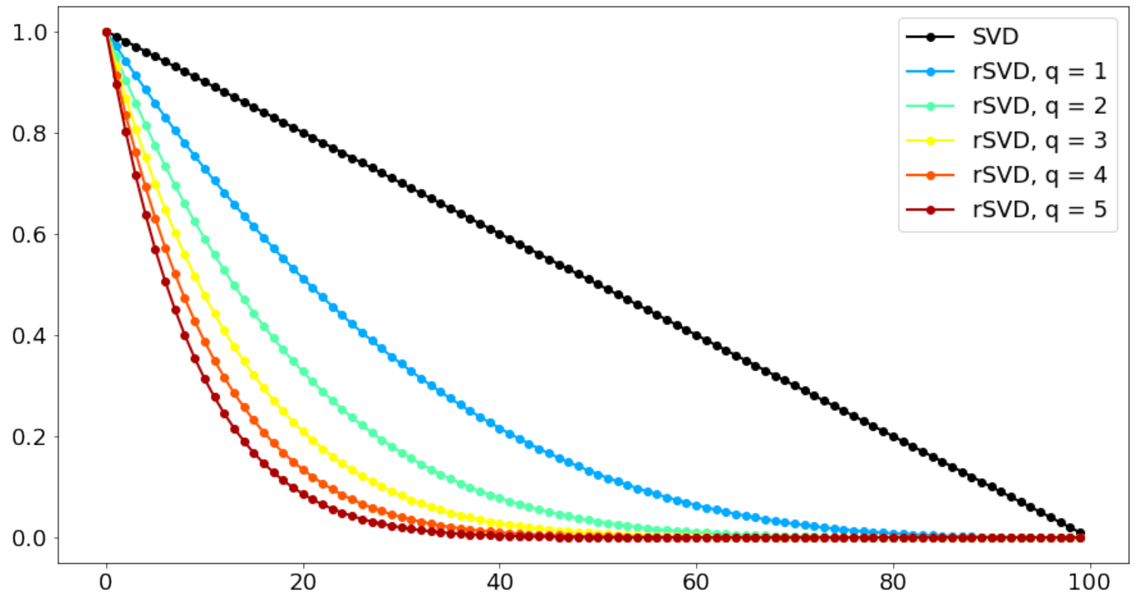


FIGURE 3.16: Singular values decay according to q value of power iterations

3.10 Randomized algorithms for computing the Interpolatory Decomposition (ID)

3.10.1 Structure preserving factorizations

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

$$\underset{m \times n}{\mathbf{A}} = \underset{m \times k}{\mathbf{C}} \; \underset{k \times n}{\mathbf{Z}}, \tag{3.25}$$

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

- If $\mathbf{A}$ is sparse or non-negative, then $\mathbf{C}$ shares these properties.
- The ID requires less memory to store than either the QR or the singular value decomposition.
- Finding the indices associated with the spanning columns is often helpful in *data interpretation.*
- In the context of numerical algorithms for discretizing PDEs and integral equations, the ID often preserves "the physics" of a problem in a way that the QR or SVD do not.

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

For future reference, let $\mathbf{J}_s$ be an index vector in $\{1, 2, \ldots, n\}$ that identifies the k columns in $\mathbf{C}$ so that

$$\mathbf{C} = \mathbf{A}(:, \mathbf{J}_s).$$

3.10.2 Three flavors of ID: The row, column, and double-sided ID

The previous section 3.10.1 describes a factorization where we use a subset of the *columns* of $\mathbf{A}$ to span its *column space*. Naturally, this factorization has a sibling which uses the *rows* of $\mathbf{A}$ to span its *row space.* In other words $\mathbf{A}$ also admits the factorization :

$$\underset{m \times n}{\mathbf{A}} = \underset{m \times k}{\mathbf{X}} \; \underset{k \times n}{\mathbf{R}}, \tag{3.26}$$

where , $\mathbf{R}$ is a matrix consisting of k rows of $\mathbf{A}$, and where $\mathbf{X}$ is a matrix that contains the $k \times k$ identity matrix. We let $\mathbf{I}_s$ denote the index vector of length k that marks the "skeleton" rows so that $\mathbf{R} = \mathbf{A}(\mathbf{I}_s, :)$.

Finally, there exists a so called *double-sided ID* which takes the form:

$$\underset{m \times n}{\mathbf{A}} = \underset{m \times k}{\mathbf{X}} \; \underset{k \times k}{\mathbf{A}_s} \; \underset{k \times n}{\mathbf{Z}}, \tag{3.27}$$

where , $\mathbf{X}$ and $\mathbf{Z}$ are the same matrices as those that appear in (3.25) and (3.26), and where $\mathbf{A}_s$ is the $k \times k$ submatrix of $\mathbf{A}$ given by :

$$\mathbf{A}_s = \mathbf{A}(\mathbf{I}_s, \mathbf{J}_s).$$

3.10.3 Deterministic techniques for computing the ID

In this section we demonstrate that there is a close connection between the column ID and the classical Column Pivoted QR factorization (CPQR).

The end result is that standard software used to compute the CPQR can with some light post-processing be used to compute the column ID. As a starting point, recall that for a given $m \times n$ matrix $\mathbf{A}$, with $m \geq n$, the QR factorization can be written as

$$\underset{m \times n}{\mathbf{A}} \; \underset{n \times n}{\mathbf{P}} = \underset{m \times n}{\mathbf{Q}} \; \underset{n \times n}{\mathbf{S}}, \tag{3.28}$$

where , $\mathbf{P}$ is a permutation matrix, where $\mathbf{Q}$ has orthonormal columns and where $\mathbf{S}$ is upper triangular. [1]

Since our objective here is to construct a rank-k approximation to $\mathbf{A}$, we split off the leading k columns from $\mathbf{Q}$ and $\mathbf{S}$ to obtain partitions:

$$\mathbf{Q} = \begin{array}{cc} & \begin{array}{cc} k & n-k \end{array} \\ m & \begin{bmatrix} \mathbf{Q}1 & \mathbf{Q}2 \end{bmatrix} \end{array}, \tag{3.29}$$

and

$$\mathbf{Q} = \begin{array}{cc} & \begin{array}{cc} k & n-k \end{array} \\ \begin{array}{c} k \\ m-k \end{array} & \begin{bmatrix} \mathbf{S}11 & \mathbf{S}12 \\ 0 & \mathbf{S}22 \end{bmatrix} \end{array}, \tag{3.30}$$

[1] We use the letter $\mathbf{S}$ instead of the traditional $\mathbf{R}$ to avoid confusion with the "R"-factor in the row ID, (3.26).

Combining (3.28) and (3.29), we then find that :

$$\mathbf{AP} = [\mathbf{Q}_1 \mid \mathbf{Q}_2] \begin{bmatrix} \mathbf{S}_{11} & \mathbf{S}_{12} \\ \mathbf{0} & \mathbf{S}_{22} \end{bmatrix} = [\mathbf{Q}_1\mathbf{S}_{11} \mid \mathbf{Q}_1\mathbf{S}_{12} + \mathbf{Q}_2\mathbf{S}_{22}]. \tag{3.31}$$

Equation (3.31) tells us that the $m \times k$ matrix :

$$\mathbf{Q}_1\mathbf{S}_{11},$$

consists precisely of first k columns of **AP**. These columns were the first k columns that were chosen as "pivots" in the QR-factorization procedure.

They typically form a good (approximate) basis for the column space of **A**. We consequently define our $m \times k$ matrix **C** as this matrix holding the first k pivot columns. Letting J denote the permutation vector associated with the permutation matrix **P**, so that

$$\mathbf{AP} = \mathbf{A}(:, \mathbf{J}),$$

we define $\mathbf{J}_s = \mathbf{J}(1:k)$ as the index vector identifying the first k pivots, and set

$$\mathbf{C} = \mathbf{A}(:, \mathbf{J}_s) = \mathbf{Q}_1\mathbf{S}_{11}. \tag{3.32}$$

Now let us rewrite (3.31) by extracting the product $\mathbf{Q}_1\mathbf{S}_{11}$:

$$\mathbf{AP} = \mathbf{Q}_1[\mathbf{S}_{11} \mid \mathbf{S}_{12}] + \mathbf{Q}_2[\mathbf{0} \mid \mathbf{S}_{22}] = \mathbf{Q}_1\mathbf{S}_{11}[\mathbf{I}_k \mid \mathbf{S}_{11}^{-1}\mathbf{S}_{12}] + \mathbf{Q}_2[\mathbf{0} \mid \mathbf{S}_{22}]. \tag{3.33}$$

Now we define:

$$\mathbf{T} = \mathbf{S}_{11}^{-1}\mathbf{S}_{12}, \qquad \text{and} \qquad \mathbf{Z} = [\mathbf{I}_k \mid \mathbf{T}]\mathbf{P}^* \tag{3.34}$$

so that (3.33) can be rewritten (upon right-multiplication by $\mathbf{P}^*$, which equals $\mathbf{P}^{-1}$ since **P** is unitary) as :

$$\mathbf{A} = \mathbf{C}[\mathbf{I}_k \mid \mathbf{T}]\mathbf{P}^* + \mathbf{Q}_2[\mathbf{0} \mid \mathbf{S}_{22}]\mathbf{P}^* = \mathbf{CZ} + \mathbf{Q}_2[\mathbf{0} \mid \mathbf{S}_{22}]\mathbf{P}^*. \tag{3.35}$$

Equation (3.35) is precisely the column ID we sought, with the additional bonus that the remainder term is explicitly identified.

Observe that when the spectral or Frobenius norms are used, the error term is of *exactly* the same size as the error term obtained from a truncated QR factorization:

$$\|\mathbf{A} - \mathbf{CZ}\| = \|\mathbf{Q}_2[\mathbf{0} \mid \mathbf{S}_{22}]\mathbf{P}^*\| = \|\mathbf{S}_{22}\| = \|\mathbf{A} - \mathbf{Q}_1[\mathbf{S}_{11} \mid \mathbf{S}_{12}]\mathbf{P}^*\|.$$

Of course, the row ID can be computed via an entirely analogous process that starts with a CPQR of the *transpose* of **A**. In other words, we execute a pivoted Gram-Schmidt orthonormalization process on the rows of **A**.

Finally, to obtain the double-sided ID, we start with using the CPQR-based process to build the column ID (3.25). Then compute the row ID by performing Gram-Schmidt on the rows of the tall thin matrix **C**.

The three deterministic algorithms described for computing the three flavors of ID are summarized in Figure 3.17.

```
Compute a column ID so that $\mathbf{A} \approx \mathbf{A}(:, J_s)\,\mathbf{Z}$.
function $[J_s,\ \mathbf{Z}] = \texttt{ID\_col}(\mathbf{A}, k)$
  $[\mathbf{Q}, \mathbf{S}, J] = \texttt{qr}(\mathbf{A}, 0)$;
  $\mathbf{T} = (\mathbf{S}(1:k, 1:k))^{-1}\mathbf{S}(1:k, (k+1):n)$;
  $\mathbf{Z} = \texttt{zeros}(k, n)$
  $\mathbf{Z}(:, J) = [\mathbf{I}_k\ \mathbf{T}]$;
  $J_s = J(1:k)$;
Compute a row ID so that $\mathbf{A} \approx \mathbf{X}\,\mathbf{A}(I_s, :)$.
function $[I_s,\ \mathbf{X}] = \texttt{ID\_row}(\mathbf{A}, k)$
  $[\mathbf{Q}, \mathbf{S}, J] = \texttt{qr}(\mathbf{A}^*, 0)$;
  $\mathbf{T} = (\mathbf{S}(1:k, 1:k))^{-1}\mathbf{S}(1:k, (k+1):m)$;
  $\mathbf{X} = \texttt{zeros}(m, k)$
  $\mathbf{X}(J, :) = [\mathbf{I}_k\ \mathbf{T}]^*$;
  $I_s = J(1:k)$;
Compute a double-sided ID so that $\mathbf{A} \approx \mathbf{X}\,\mathbf{A}(I_s, J_s)\,\mathbf{Z}$.
function $[I_s, J_s, \mathbf{X}, \mathbf{Z}] = \texttt{ID\_double}(\mathbf{A}, k)$
  $[J_s,\ \mathbf{Z}] = \texttt{ID\_col}(\mathbf{A}, k)$;
  $[I_s,\ \mathbf{X}] = \texttt{ID\_row}(\mathbf{A}(:, J_s), k)$;
```

FIGURE 3.17: Deterministic algorithms for computing the column, row, and double-sided ID via the column pivoted QR factorization. The input is in every case an $m \times n$ matrix $\mathbf{A}$ and a target rank k. Since the algorithms are based on the CPQR, it is elementary to modify them to the situation where a tolerance rather than a rank is given. (Recall that the errors resulting from these ID algorithms are *identical* to the error in the first CPQR factorization executed.)

3.10.4 Randomized techniques for computing the ID

The ID is particularly well suited to being computed via randomized algorithms. To describe the ideas, suppose temporarily that $\mathbf{A}$ is an $m \times n$ matrix of *exact* rank k, and that we have by some means computed an approximate rank-k factorization

$$\begin{array}{cccc} \mathbf{A} & = & \mathbf{Y} & \mathbf{F}. \\ m \times n & & m \times k & k \times n \end{array} \tag{3.36}$$

Once the factorization (3.36) is available, let us use the algorithm `ID_row` described in Figure 3.17 to compute a row ID $[I_s, \mathbf{X}] = \texttt{ID\_row}(\mathbf{Y}, k)$ of $\mathbf{Y}$ so that :

$$\begin{array}{cccc} \mathbf{Y} & = & \mathbf{X} & \mathbf{Y}(I_s, :). \\ m \times k & & m \times k & k \times k \end{array} \tag{3.37}$$

As we have seen, the task of finding a matrix $\mathbf{Y}$ whose columns form a good basis for the column space of a matrix is ideally suited to randomized sampling. To be precise, we showed in Section 3.3 that given a matrix $\mathbf{A}$, we can find a matrix $\mathbf{Y}$ whose columns approximately span the column space of $\mathbf{A}$ via the formula $\mathbf{Y} = \mathbf{A}\mathbf{G}$, where $\mathbf{G}$ is a tall thin Gaussian random matrix.

The algorithm that results from combining these two insights is summarized in Figure 3.18.

ALGORITHM: RANDOMIZED ID

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

Outputs: An $m \times k$ interpolation matrix $\mathbf{X}$ and an index vector $I_s \in \mathbb{N}^k$ such that $\mathbf{A} \approx \mathbf{X}\mathbf{A}(I_s, :)$.

```
(1)  $\mathbf{G} = \texttt{randn}(n, k + p)$;
(2)  $\mathbf{Y} = \mathbf{AG}$;
(3)  for $j = 1 : q$
(4)      $\mathbf{Y}' = \mathbf{A}^*\mathbf{Y}$;
(5)      $\mathbf{Y} = \mathbf{A}\mathbf{Y}'$;
(6)  end for
(7)  Form an ID of the $n \times (k + p)$ sample matrix: $[I_s, \mathbf{X}] = \texttt{ID\_row}(\mathbf{Y}, k)$.
```

FIGURE 3.18: Randomized ID Algorithm

3.10.5 Example of Randomized ID technique

In order to implement the Randomized ID algorithm in image compressing we will work with a grey scale "Rubik's cube" which is a matrix $\mathbf{A} \in \mathbb{R}m \times n$ with $m = 3024$ and $n = 4032$.

We run the algorithm for three different values of k parameter $k = 500, 100, 50$ and plot the results as shown in figure 3.9.For full code see Appendix.

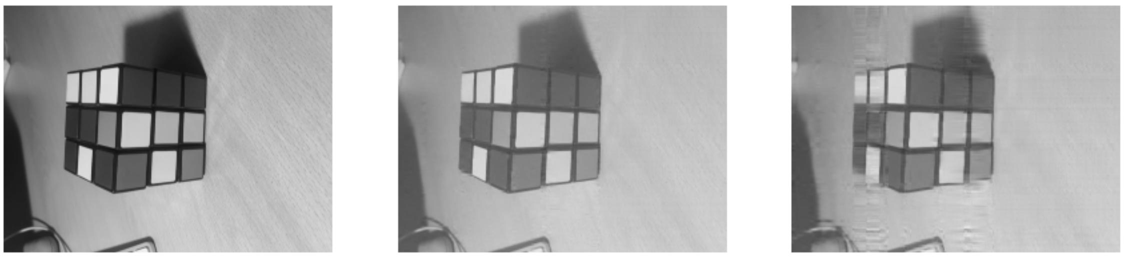


FIGURE 3.19: Interpolatory Decomposition for $k = [500, 100, 50]$

Chapter 4

Randomized Principal Component Analysis

4.1 PCA Overview

Dimensionality reduction is a fundamental concept in modern data analysis. The idea is to exploit relationships among points in high-dimensional space in order to construct some low-dimensional summaries. This process aims to eliminate redundancies, while preserving interesting characteristics of the data (Burges 2010). Dimensionality reduction is used to improve the computational tractability, to extract interesting features, and to visualize data which are comprised of many interrelated variables.

The most important linear dimension reduction technique is principal component analysis (PCA), originally formulated by Pearson (1901) and Hotelling (1933). PCA plays an important role, in particular, due to its simple geometric interpretation. Jolliffe (2002) provides a comprehensive introduction to PCA. Principal component analysis aims to find a new set of uncorrelated variables. The so called principal components (PCs) are constructed such that the first PC explains most of the variation in the data; the second PC most of the remaining variation and so on.

This property ensures that the PCs sequentially capture most of the total variation (information) present in the data. In practice, we often aim to retain only a few number of PCs which capture a "good" amount of the variation, where "good" depends on the application. To be more formal, assume a data matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$ with m observations and n variables (column-wise, mean-centered). Then, the principal components can be expressed as a weighted linear combination of the original variables :

$$z_i = \mathbf{X}\boldsymbol{w}_i,$$

where $z_i \in \mathbb{R}^m$ denotes the i^{th} principal component.

The vector $\boldsymbol{w}_i \in \mathbb{R}^n$ is the i^{th} principal direction, where the elements of $w_i = [w_1, \ldots, w_n]^T$ are the principal component coefficients. The problem is now to find a suitable vector $\boldsymbol{w}_1$ such that the first principal component z_1 captures most of the variation in the data. Mathematically, this problem can be formulated either as a least square problem or as a variance maximization problem (Cunningham and Ghahramani 2015). This is, because the total variation equals the sum of the explained and unexplained variation (Jolliffe 2002), illustrated in Figure 4.1.

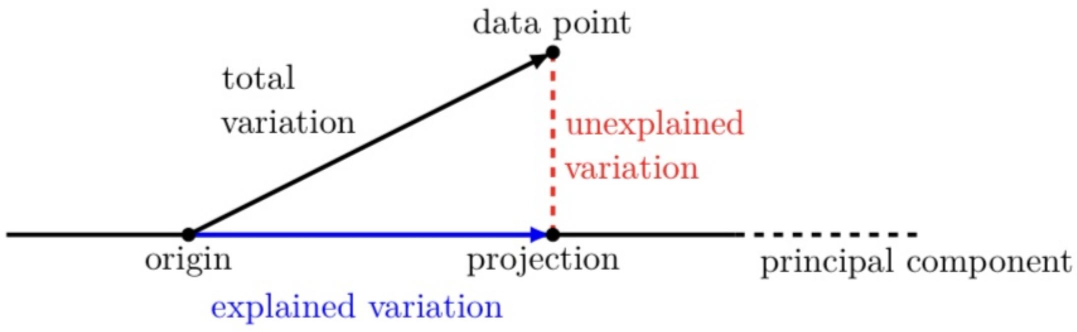


FIGURE 4.1: The PCs can be obtained by either maximizing the variance or by minimizing the unexplained variation (squared residuals) of the data

We follow the latter view, and maximize the variance of the first principal component $\boldsymbol{z}_i = \mathbf{X}\boldsymbol{w}_i$, subject to the normalization constraint : $||\boldsymbol{w}||_2^2 = 1$

4.2 Randomized algorithm of PCA

The singular value decomposition provides a computationally efficient and numerically stable approach for computing the principal components. Specifically, the eigenvalue decomposition of the inner and outer dot product of $\mathbf{X} = \mathbf{U\Sigma V}^\top$ can be related to the SVD as:

$$\mathbf{X}^\top\mathbf{X} = (\mathbf{V\Sigma U}^\top)(\mathbf{U\Sigma V}^\top) = \mathbf{V\Sigma}^2\mathbf{V}^\top, \tag{4.1}$$

$$\mathbf{XX}^\top = (\mathbf{U\Sigma V}^\top)(\mathbf{V\Sigma U}^\top) = \mathbf{U\Sigma}^2\mathbf{U}^\top. \tag{4.2}$$

It follows that the eigenvalues are equal to the squared singular values. Thus, we recover the eigenvalues of the sample covariance matrix :

$$\mathbf{C} := (m-1)^{-1}\mathbf{X}^\top\mathbf{X},$$

as

$$\mathbf{\Lambda} = (m-1)^{-1}\mathbf{\Sigma}^2.$$

The left singular vectors $\mathbf{U}$ correspond to the eigenvectors of the outer product $\mathbf{XX}^\top$, from (4.2) and the right singular vectors $\mathbf{V}$ from (4.1) correspond to the eigenvectors of the inner product $\mathbf{X}^\top\mathbf{X}$. This allows us to define the projection (rotation) matrix $\mathbf{W} := \mathbf{V}$. Having established the connection between the singular value decomposition and principal component analysis, it is straight forward to show that the principal components can be computed as

$$\mathbf{Z} := \mathbf{XW} = \mathbf{U\Sigma V}^\top\mathbf{W} = \mathbf{U\Sigma}. \tag{4.3}$$

The randomized singular value decomposition can then be used to efficiently approximate the dominant k principal components. This approach in (4.3) is denoted as randomized principal component analysis, first introduced by Rokhlin 2009 , and later by Halko 2011 providing us some additional interesting implementation details for large-scale applications in rPCA.

ALG:RPCA ALGORITHM

*Input:*Centered/scaled input matrix $\mathbf{X}$ with dimensions $m \times n$, and target rank $k < \min m, n$, p oversampling parameter and q the number for iterations.

```
1. $\mathbf{U},\mathbf{\Sigma},\mathbf{W}$ = rsvd(X,k,p,q).
                                        Randomized SVD
2. $\mathbf{\Lambda} = \mathbf{\Sigma}^2/m - 1$.
                                        recover eigenvalues
3. $\mathbf{Z} = \mathbf{U\Sigma}$
                    optional: compute k principal components
```

Output: Matrices $\mathbf{W} \in \mathbb{R}^{n\times k}$, $\mathbf{\Lambda} \in \mathbb{R}^{k\times k}$, and $\mathbf{Z} \in \mathbb{R}^{m\times k}$

FIGURE 4.2: A randomized PCA algorithm.

The above algorithm presents an implementation of the randomized PCA. The approximation accuracy can be controlled via oversampling and additional power iterations as described in Section 3.1.1 for power iteration scheme.

4.3 Randomized PCA application in Python with eigenfaces

Python programming language from Scikit-Learn library has a built in function in order to calculate the Randomized PCA. Here we will explore the Labeled Faces in the Wild dataset made available through Scikit-Learn that contains 966 faces.

Let's take a look at the principal axes that span this dataset. Because this is a large dataset, we will use Randomized PCA it contains a randomized method to approximate the first k principal components much more quickly than the standard PCA estimator, and thus is very useful for high-dimensional data (here, a dimensionality of nearly 3,000). We will take a look at the first $k = 150$ components.

In this case, it can be interesting to visualize the images associated with the first several principal components (these components are technically known as "eigenvectors," so these types of images are often called "eigenfaces").

The results are very interesting, and give us insight into how the images vary: for example, the first few eigenfaces (from the top left) seem to be associated with the angle of lighting on the face, and later principal vectors seem to be picking out certain features, such as eyes, noses, and lips. Let's take a look at the cumulative variance of these components to see how much of the data information the projection is preserving:

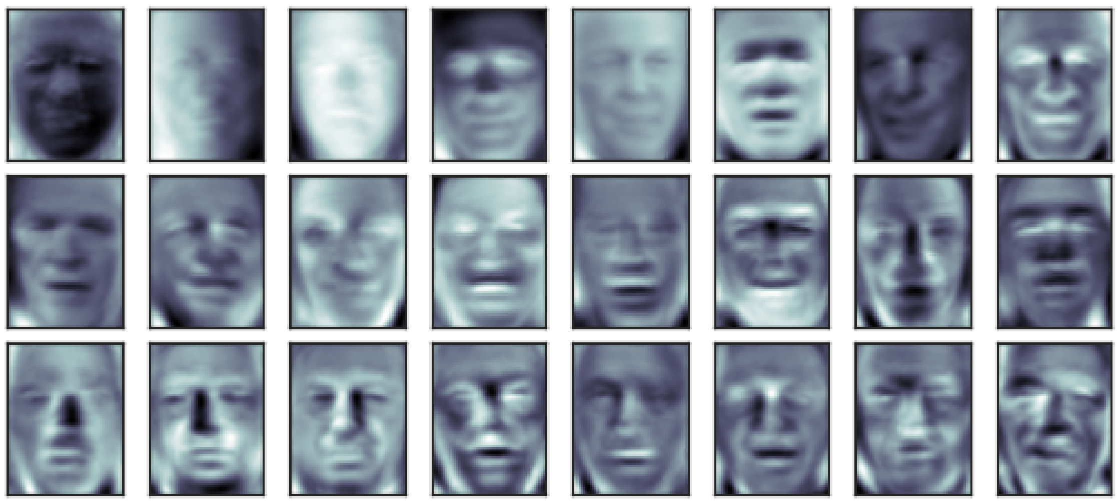


FIGURE 4.3: Faces Randomized PCA

We see that these 150 components account for just over 90% of the variance. That would lead us to believe that using these 150 components, we would recover most of the essential characteristics of the data. To make this more concrete, we can compare the input images with the images reconstructed from these 150 components.

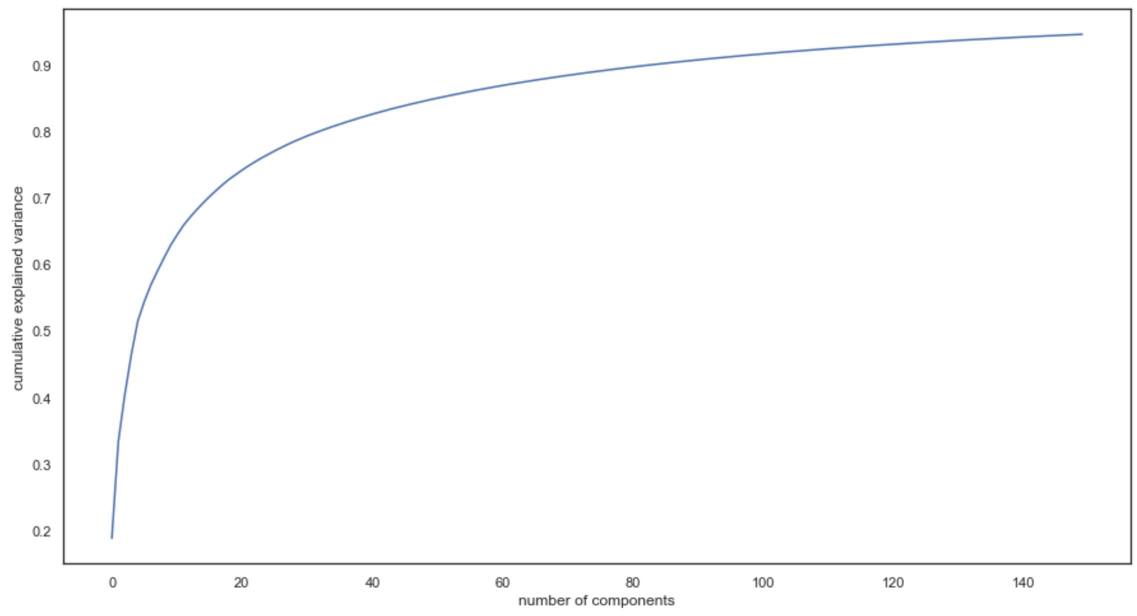


FIGURE 4.4: Cumulative explained variance

The top row here shows the input images, while the bottom row shows the reconstruction of the images from just 150 of the 3,000 initial features. This visualization makes clear why the PCA feature selection reduces the dimensionality of the data by nearly a factor of 20.

The projected images contain enough information that we might, by eye, recognize the individuals in the image. What this means is that our classification algorithm needs to be trained on 150-dimensional data rather than 3,000-dimensional data, which depending on the particular algorithm we choose, can lead to a much more efficient classification.

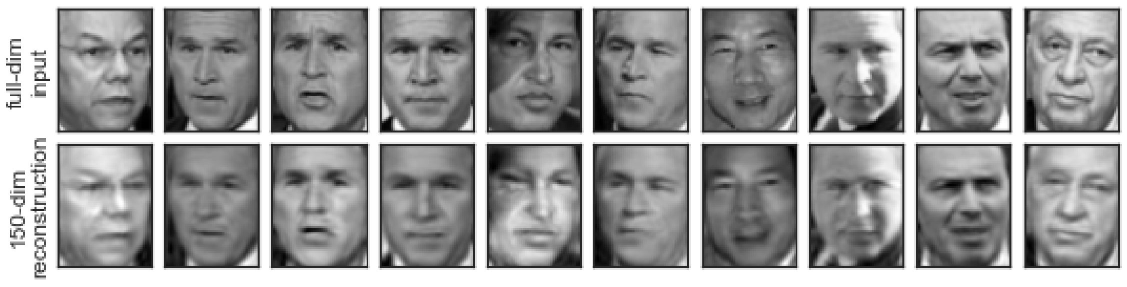


FIGURE 4.5: Comparison of original faces and reconstructed

4.4 Robust Principal Component Analysis

Thus far, we have viewed matrix approximations as a factorization of a given matrix into a product of smaller (low-rank) matrices. However, there is another interesting class of matrix decompositions, which aim to separate a given matrix into low-rank, sparse and noise components.

Such decompositions are motivated by the need for robust methods which can more effectively account for corrupt or missing data. Indeed, outlier rejection is critical in many applications as data is rarely free of corrupt elements.

Robustification methods decompose data matrices as follows:

$$\underset{m\times n}{\mathbf{A}} \approx \underset{m\times n}{\mathbf{L}} + \underset{m\times n}{\mathbf{S}} + \underset{m\times n}{\mathbf{E}},$$

where , $\mathbf{L} \in \mathbb{R}^{m\times n}$ denotes the low-rank matrix, $\mathbf{S} \in \mathbb{R}^{m\times n}$ the sparse matrix, and $\mathbf{E} \in \mathbb{R}^{m\times n}$ the noise matrix.

Note that the sparse matrix $\mathbf{S}$ represents the corrupted entries (outliers) of the data matrix $\mathbf{A}$. In the following we consider only the special case $\mathbf{A} = \mathbf{L} + \mathbf{S}$, i.e., the decomposition of a matrix into its sparse and low-rank components. This form of additive decomposition is also denoted as robust principal component analysis (RPCA), and its remarkable ability to separate high-dimensional matrices into low-rank and sparse component makes RPCA an invaluable tool for data science.

The additive decomposition is, however, different from classical robust PCA methods known in the statistical literature. These techniques are concerned with computing robust estimators for the empirical covariance matrix, for instance, see the seminal work by Hubert 2005 and Croux 2005 . While traditional principal component analysis minimizes the spectral norm of the reconstruction error, robust PCA aims to recover the underlying low-rank matrix of a heavily corrupted input matrix.

Candes et al 2011, proved that it is possible to exactly separate such a data matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ into both its low-rank and sparse components, under rather broad assumptions. This is achieved by solving a convenient convex optimization problem called *principal component pursuit* (PCP). The objective is to minimize a weighted combination of the nuclear norm:

$$\min_{\mathbf{L},\mathbf{S}} \operatorname{rank}(\mathbf{L}), + \|\mathbf{S}\|_0 \text{ subject to } \mathbf{L} + \mathbf{S} = \mathbf{X}, \tag{4.4}$$

However, neither the $\operatorname{rank}(\mathbf{L})$ nor the $\|\mathbf{S}\|_0$ terms are convex, and this is not a tractable optimization problem. Similar to the compressed sensing problem, it is possible to solve for the optimal $\mathbf{L}$ and $\mathbf{S}$ with high probability using a convex relaxation of (4.4) :

$$\min_{\mathbf{L},\mathbf{S}} \|(\mathbf{L})\|_\star + \lambda\|\mathbf{S}\|_1 \text{ subject to } \mathbf{L} + \mathbf{S} = \mathbf{X}, \tag{4.5}$$

Here $\|\cdot\|_\star$ denotes the nuclear norm, given by the sum of singular values, which is a proxy for rank. Remarkably, the solution to 4.5 converges to the solution of 4.4 , with high probability, if $\lambda = 1/\sqrt{m,n}$, where , n and m are the dimensions of $\mathbf{X}$, given that $\mathbf{L}$ and $\mathbf{S}$ satisfy the following conditions:

1. $\mathbf{L}$ is not sparse,
2. $\mathbf{S}$ is not low-rank.We assume that the entries are randomly distributed so that they do not have low-dimensional column space.

The convex problem in 4.4) is known as principal component pursuit (PCP), and may be solved using the augmented Lagrange multiplier (ALM) algorithm. Specifically, an augmented Lagrangian may be constructed:

$$\mathcal{L}(\mathbf{L},\mathbf{S},\mathbf{Z},\mu) = \|\mathbf{L}\|_* + \lambda\|\mathbf{S}\|_1 + \langle \mathbf{Z}, \mathbf{A}-\mathbf{L}-\mathbf{S}\rangle + \frac{\mu}{2}\|\mathbf{A}-\mathbf{L}-\mathbf{S}\|_F^2, \tag{4.6}$$

where , μ and λ are positive scalars, and $\mathbf{Z}$ the Lagrange multiplier.

Further, $\langle\cdot,\cdot\rangle$ is defined as $\langle \mathbf{A},\mathbf{B}\rangle := \text{trace}(\mathbf{A}^\top\mathbf{B})$.

The method of augmented Lagrange multipliers can be used to solve the optimization problem Bertsekas 1999 and Lin et al 2011 have both proposed an exact and inexact algorithm to solve Equation 4.6. Here, we advocate the latter approach.

Specifically, the inexact algorithm avoids solving the problem:

$$\mathbf{L}_{i+1}, \mathbf{E}_{i+1} = \arg\min_{\mathbf{L},\mathbf{E}} \mathcal{L}(\mathbf{L},\mathbf{E},\mathbf{Z}_i,\mu_k),$$

by alternately solving the following two sub-problems at step i:

$$\mathbf{L}_{i+1} = \arg\min_{\mathbf{L}} \mathcal{L}(\mathbf{L},{}_i,\mathbf{Z}_i,\mu_k), \tag{4.7}$$

and

$$\mathbf{E}_{i+1} = \arg\min_{\mathbf{E}} \mathcal{L}(\mathbf{L}_{i+1},\mathbf{E},\mathbf{Z}_i,\mu_k).$$

A general solution would solve for the $\mathbf{L}_k$ and $\mathbf{S}_k$ that minimize $\mathcal{L}$, update the Lagrange multipliers :

$$\mathbf{Y}_{k+1} = \mathbf{Y}_k + \mu(\mathbf{X}-\mathbf{L}_k-\mathbf{S}_k),$$

and iterate until the solution converges.

However, for this specific system, the alternating directions method , Lin et al 2011 provided a simple procedure to find $\mathbf{L}$ and $\mathbf{S}$. First, we construct the shrinkage operator :

$$\mathcal{S}_\tau(x) = \text{sign}(x)\max(|x|-\tau,0).$$

Next, the singular value threshold operator

$$\mathbf{SVT}_\tau(X) = \mathbf{U}\mathcal{S}_\tau(\mathbf{\Sigma})\mathbf{V}^\star,$$

is constructed.

Finally, it is possible to use $\mathcal{S}_\tau$ and $\mathbf{SVT}$ operators iteratively to solve for $\mathbf{L}$ and $\mathbf{S}$. In Appendix the reader can find linked the code in Python which contains the Robust PCA implementation as described in this subsection. For this implementation the dataset is from Yale B database as found in Steven L. Brunton & Nathan Kutz in the book Data Driven Science & Engineering Machine Learning, Dynamical Systems, and Control.

Chapter 5

Pseudospectra comparison & Conclusions

5.1 Pseudospectra comparison of Original & Randomized matrix

As discussed in section 2.1 and more specifically in subsection 2.1.2 about the pseudospectra of square random matrices we are ready to compare the original versus the randomized edition of each algorithm.

5.1.1 RSVD pseudo

The original matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ with $m = 1204, n = 1880$ the pseudospectra of the original matrix which has been block partitioned into the first $m \times m$ matrix $\mathbf{A}$ is :

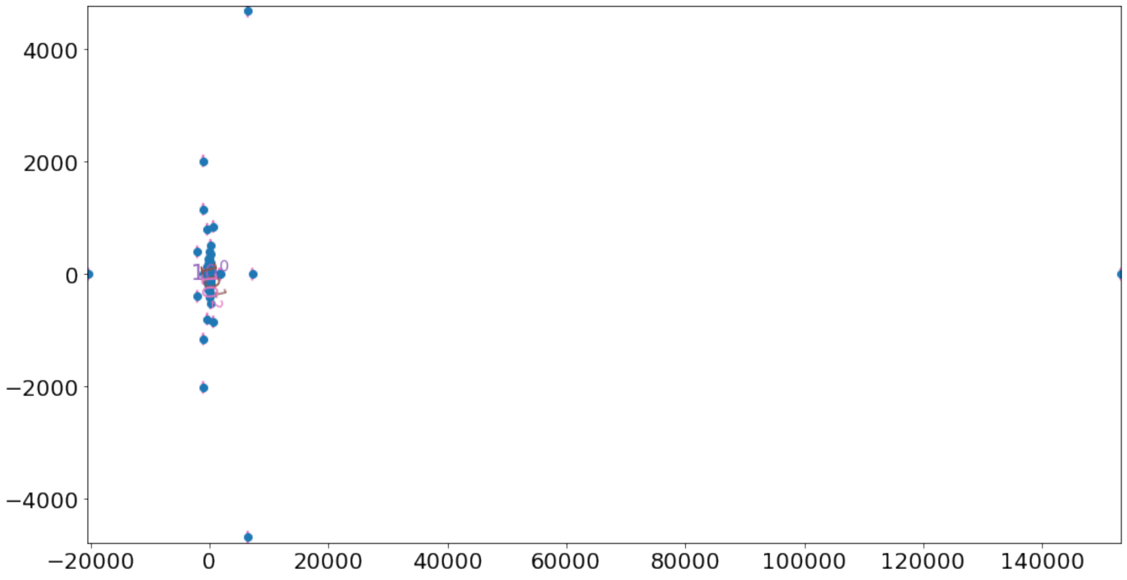


FIGURE 5.1: Pseudo spectrum of original matrix square block $m \times m$

The pseudopsectrum of low rank $k = 400$ approximation is:

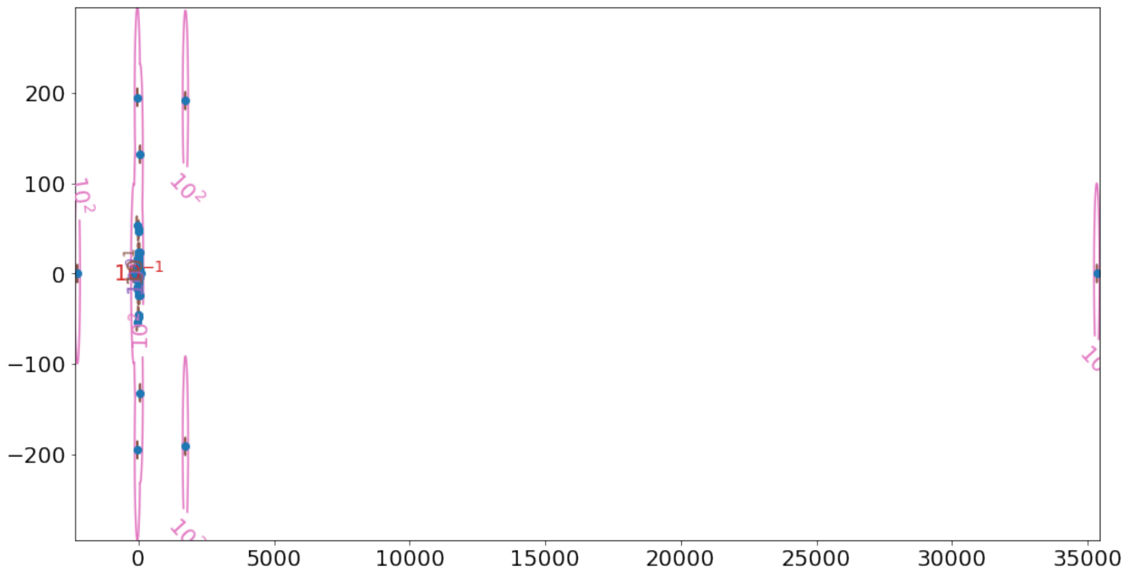


FIGURE 5.2: Pseudo spectrum of the low rank approximation

The pseudopsectrum of randomized singular value decomposition $k = 400$ approximation is :

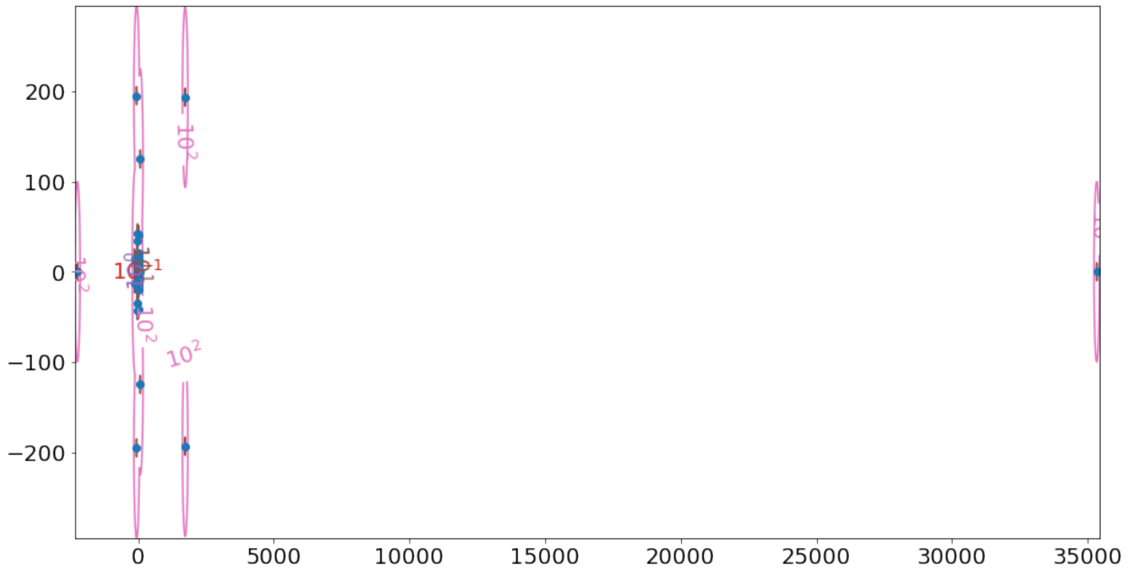


FIGURE 5.3: Pseudo spectrum of the low rank approximation

5.1.2 Single-pass pseudo

Subject to Single pass algorithm matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ with $m = 1204, n = 1880$ the pseudospectra of the original matrix which has been block partitioned into the first $m \times m$ matrix $\mathbf{A}$ is :

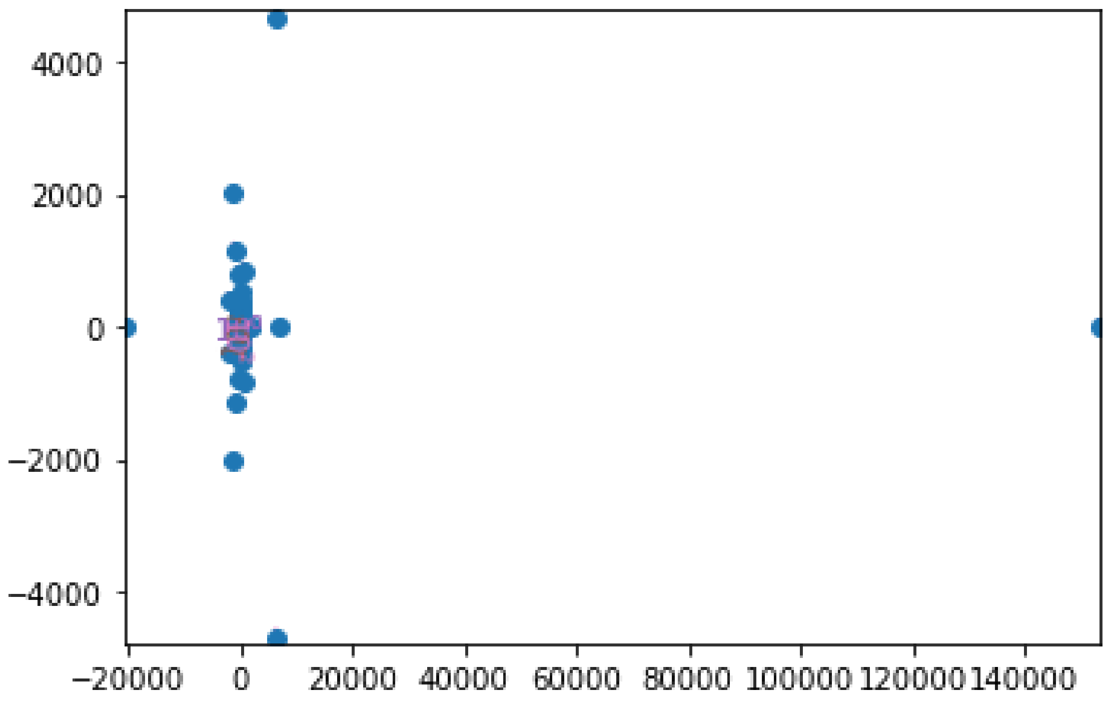


FIGURE 5.4: Single pass pseudo spectrum of the pure matrix

where the randomized decomposition again partitioned into now $l \times l$ is the :

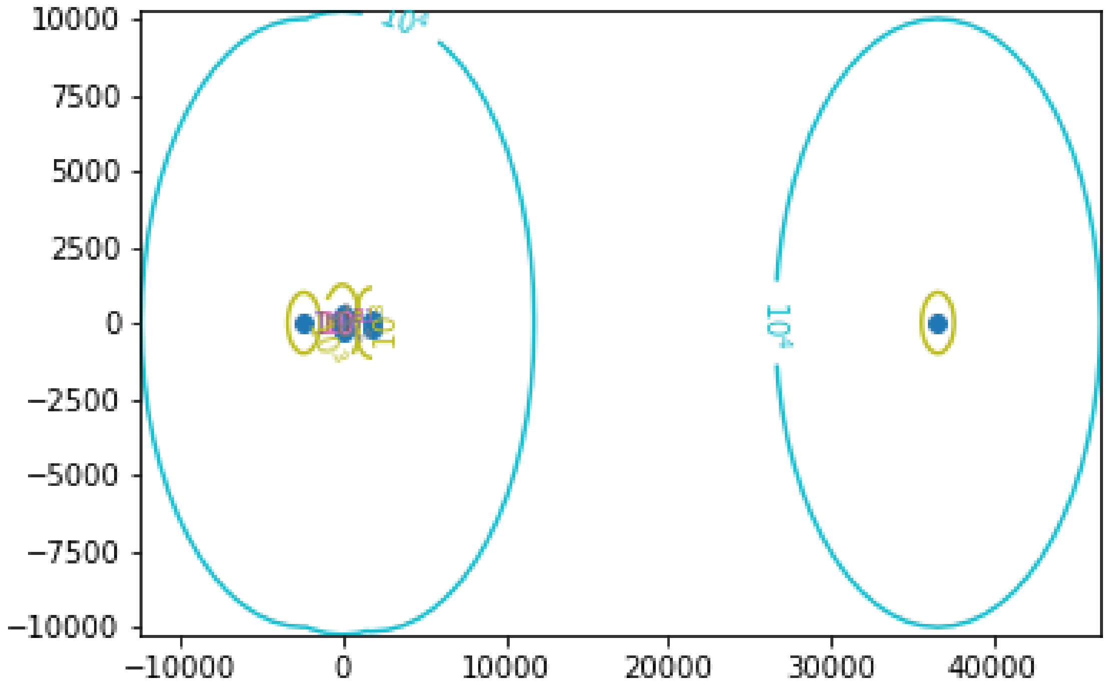


FIGURE 5.5: Pseudo spectrum of the randomized decomposed matrix of Single pass algorithm

5.1.3 Nystrom pseudo

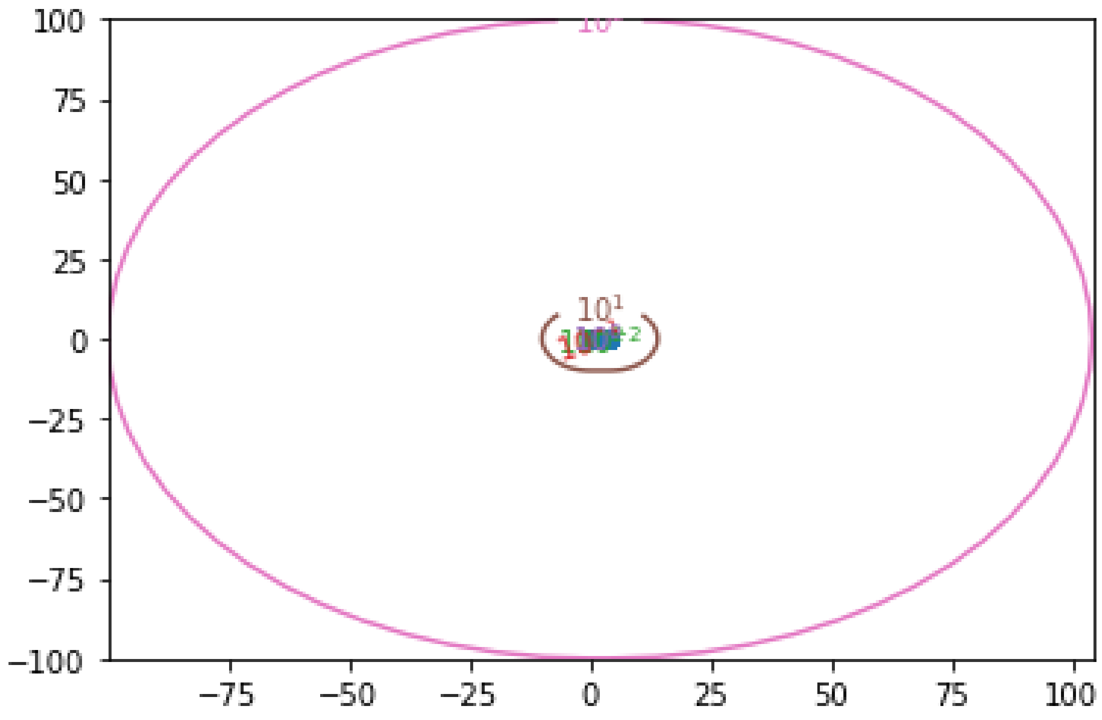


FIGURE 5.6: pseudo spectrum of the pure PSD matrix

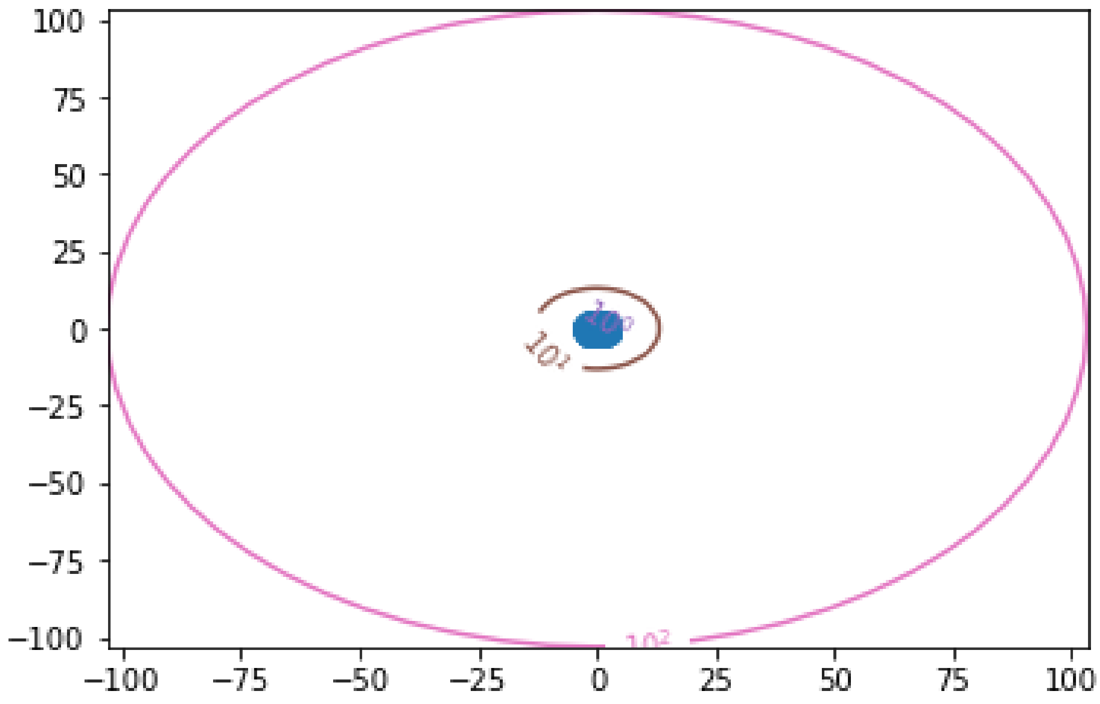


FIGURE 5.7: Pseudo spectrum of the randomized decomposed matrix of Nystrom method

5.1.4 Interpolatory decomposition pseudo spectra

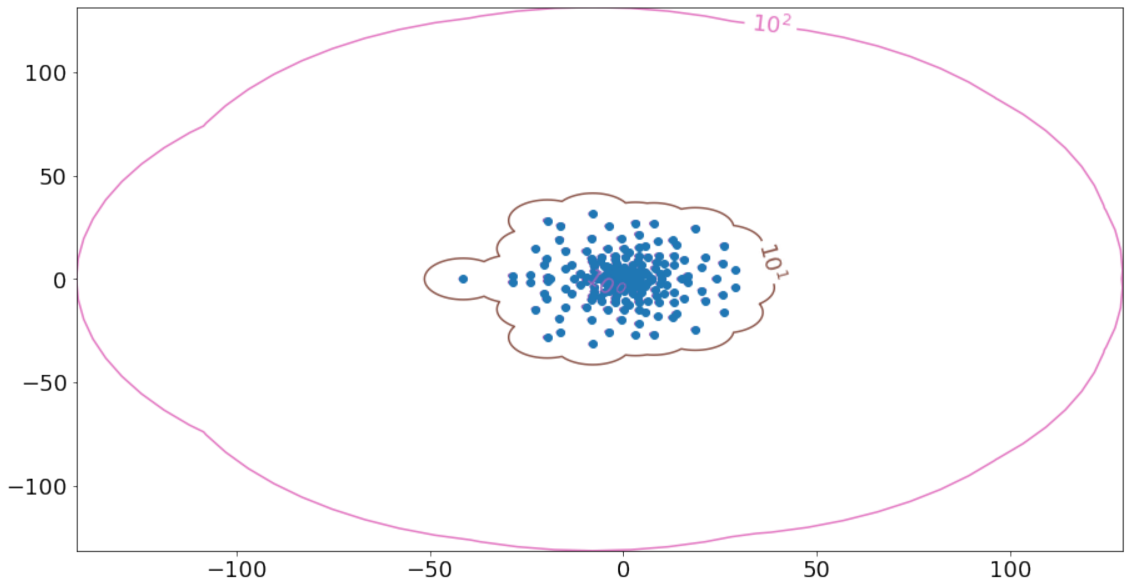


FIGURE 5.8: Pseudo spectrum of the randomized interpolatory decomposition

5.2 Conclusions

Summarising this thesis we have succeeded a more deep understanding and knowledge of Randomized Singular Value Decomposition.The randomized approach gave us an alternative scientific tool from the classical approach of svd economy or low rank , for computing the $k + p$ randomized column selection of large matrices that has practical use in statistical and machine learning algorithms.

5.3 Bibliography

Bibliography

[1] Rishi Advani & Sean O'Hagan (2021). *Efficient Algorithms for Constructing an Interpolative Decomposition*. ArXiv

[2] Rishi Advani & Sean O'Hagan (2006). *Fast Monte Carlo algorithms for matrices 2: Computing a low rank approximation to a matrix*. Siam

[3] Daniel Ahfock, William J. Astle & Sylvia Richardson (2019) *Statistical properties of sketching algorithms*. ArXiv

[4] Robert Andrew & Nicholas Dingle (2014). *Implementing QR factorization updating algorithms on GPUs*. Elsevier

[5] Rajendra Bhatia (2007). *Positive Definite Matrices*. Princeton University Press

[6] Rajendra Bhatia (1997). *Matrix Analysis*. Springer

[7] Claude Brezinski (1975). *Computation of Eigenelements of a Matrix by the ϵ-algorithm*. Elsevier

[8] Steven L. Brunton, & J. Nathan Kutz (2017). *Data Driven Science & Engineering Machine Learning, Dynamical Systems, and Control* . University of Washington

[9] Thierry Bouwmans, Necdet Serhat Aybat , El-hadi Zahzah (2016). *Handbook of Robust low-rank & sparse matrix decomposition (Application in image & video processing)*. CRC Press

[10] Christopher J. C. Burges (2010). *Dimension Reduction: A Guided Tour*. Now:The Essence of Knowledge

[11] Emmanuel J. Candes, Xiaodong Li, Yi Ma & John Wright, (2009) *Robust Principal Component Analysis?*. ArXiv

[12] Ryan Compton (2010). *A Randomized Algorithm for PCA*.

[13] Amit Deshpande , Santosh Vempala. *Adaptive Sampling and Fast Low-Rank Matrix Approximation*. ArXiv

[14] Petros Drineas & Michael W. Mahoney (2005). *On the Nystrom Method for Approximating a Gram Matrix for Improved Kernel-Based Learning*. Journal of Machine Learning Research

[15] Petros Drineas & Michael W. Mahoney (2017) *Lectures on Randomized Numerical Linear Algebra*. ArXiv

[16] Kui Dua & Yimin Wei (2005) *Statistical properties of sketching algorithms*. Elsevier

[17] Carl Eckart & Gale Young (1936). *The approximation of one matrix by another of lower rank* . Psychometrika

[18] N. Benjamin Erichson, Sergey Voronin, Steven L. Brunton & J. Nathan Kutz (2019) *Randomized Matrix Decompositions Using R*. ArXiv

[19] Nick Feller (2016). *Basics of Matrix Algebra for Statistics with R*. CRC Press

[20] Xu Feng, Wenjian Yu & Yaohang Li (2018) *Faster Matrix Completion Using Randomized SVD*. ArXiv

[21] Xu Feng, Yuyang Xie ,Mingye Song, Wenjian Yu & Jie Tang (2018). *Fast Randomized PCA for Sparse Data*. ArXiv

[22] James E.Gentle (2017). *Matrix Algebra,Theory, Computations and Applications in Statistics 2nd Edition*. Springer Texts in Statistics.

[23] Yanina Gimeneza, & Guido Giussani (2017) *Searching for the core variables in principal components analysis*. ArXiv

[24] Gene H.Golub & Charles F.Van Loan (2013). *Matrix Computations 4th Edition*. The Johns Hopkins University Press

[25] M.Gu (2014) *Subspace Iteration randomization & singular value problems*. ArXiv

[26] N. Halko, P. G. Martinsson & J. A. Tropp (2010) *Finding Structure with randomness: Probabilistic algorithms for constructing approximate matrix decomposition* . ArXiv

[27] Sven Hammarling, (1985). *The Singular Value Decomposition in Multivariate Statistics*. MIT Press

[28] Hotelling, Harold (1933). *Analysis of a Complex of Statistical Variables into Principal Components.*. Journal of Educational Psychology.

[29] Higham, Nicholas J. & Tisseur, Françoise (2006). *A block algorithm for matrix 1-norm estimation, with an application to 1-norm pseudospectra*. Society for Industrial and Applied Mathematics.

[30] Roger A.Horn & Charles R.Johnson (1994). *Matrix Analysis*. Cambridge University Press

[31] Arne Jensen (2009) *Lecture Notes on Spectra and Pseudospectra of Matrices and Operators*. Department of Mathematical Sciences Aalborg University

[32] Qiang Ji (2020). *Probabilistic Graphical Models for Computer Vision*. Elsevier

[33] I.T Jolliffe (2007). *Principal Component Analysis, 2nd Edition*. Springer

[34] Sanjiv Kumar , Mehryar Mohri & Ameet Talka (2012) *Sampling Methods for the Nystrom Method*. Journal of Machine Learning Research

[35] Giacomo Livan, Marcel Novaes &Pierpaolo Vivo (2017). *Introduction to Random Matrices Theory and Practice*. ArXiv

[36] Michael W. Mahoneya, & Petros Drineas. *CUR matrix decompositions for improved data analysis*. Pnas

[37] Per-Gunnar Martinsson, Vladimir Rokhlin , Mark Tygert (2011). *A randomized algorithm for the decomposition of matrices* . Applied and Computational Harmonic Analysis,(Elsevier)

[38] Per-Gunnar Martinsson, Vladimir Rokhlin, Yoel Shkolnisky, & Mark Tygert (2014). *A software package for low-rank approximation of matrices via interpolative decompositions*.

[39] Per-Gunnar Martinsson (2021) *Randomized Numerical Linear Algebra: Foundations & Algorithms*. ArXiv

[40] Per-Gunnar Martinsson (2019) *Randomized methods for matrix computations*. ArXiv

[41] Carl D.Meyer. *Matrix Analysis and Applied Linear Algebra*. SIAM

[42] Tim Moon and Jack Poulson,(2016). *Accelerating eigenvector and pseudospectra computation using blocked multi-shift triangular solves*. ArXiv

[43] Rajeev Motwani & Prabhakar Raghavan (1995) *Randomized Algorithms*. Cambridge University Press

[44] Panayiotis J. Psarrakos (2000). *Numerical range of linear pencils*. Elsevier

[45] Vladimir Rokhlin, Arthur Szlam, & Mark Tygert (2009). *A Randomized Algorithm for Principal Component Analysis* . ArXiv

[46] Mark Rudelson & Roman Vershynin (2006) *Sampling from large matrices: An approach through geometric functional analysis*. ArXiv

[47] James Sethna (2016) *Random Matrix Theory*. Sethna, "Entropy, Order Parameters, and Complexity"

[48] James Schott (2017). *Matrix Analysis for Statistics*. Wiley Series in Probability & Statistics

[49] Lloyd N.Trefethen (1999). *Computation of Pseudospectra*. Cambridge University Press

[50] Lloyd N. Trefethen & Mark Embree (2005) *Spectra & Pseudospectra (The Behavior of Nonormal Matrices & Operators)*. Princeton University Press

[51] Jake VanderPlas (2017). *Python Data Science Handbook*. O'Reilly

[52] Hrishikesh D.Vinod (2016). *Hands-on Matrix Algebra Using R*. World Scientific

[53] Sergey Voronin & Per-Gunnar Martinsson,(2016). *An implementation of randomized algorithms for computing the singular value, interpolative, and CUR decompositions of matrices on multi-core and GPU architectures*. ArXiv

[54] Thomas G. Wright & Lloyd N. Trefethen (2002). *Pseudospectra of rectangular matrices*. IMA Journal of Numerical Analysis

[55] John Wright, Yigang Peng, Yi Ma , Arvind Ganesh & Shankar Rao (2009) *Robust Principal Component Analysis: Exact Recovery of Corrupted Low-Rank Matrices by Convex Optimization*. ArXiv

Appendix A

Appendix

A.1 Greek Summary Edition

Σχολη Εφαρμοσμενων Μαθηματικων και Φυσικων Επιστημων
Τμημα Εφαρμοσμενων Μαθηματικων Επιστημων

Τυχαιοποιημένοι αλγόριθμοι παραγοντοποίησης πινάκων με την χρήση της **Python**

Μεταπτυχιακη Διπλωματικη Εργασια

του

ΝΙΚΟΛΑΟΥ Χ. ΜΑΤΣΑΒΕΛΑ

Επιβλέπων: Π.Ψαρράκος
Καθηγητής ΕΜΠ

Αθήνα, Σεπτέμβριος 2021

Σχολή Εφαρμοσμένων Μαθηματικών και Φυσικών Επιστημών
Τμήμα Εφαρμοσμένων Μαθηματικών Επιστημών

Τυχαιοποιημένοι αλγόριθμοι παραγοντοποίησης πινάκων με την χρήση της **Python**

Μεταπτυχιακη Διπλωματικη Εργασια

του

ΝΙΚΟΛΑΟΥ Χ. ΜΑΤΣΑΒΕΛΑ

Επιβλέπων: Π.Ψαρράκος
Καθηγητής ΕΜΠ

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή Σεπτέμβριος 2021.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
....	....	....
Π.Στεφανέας	Κ. Χρυσαφίνος	Π.Ψαρράκος
Αν. Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής ΕΜΠ

Αθήνα, Σεπτέμβριος 2021

Σχολή Εφαρμοσμένων Μαθηματικών και Φυσικών Επιστημών
Τμήμα Εφαρμοσμένων Μαθηματικών Επιστημών

Υπεύθυνη Δήλωση

Βεβαιώνω ότι είμαι συγγραφέας αυτής της πτυχιακής εργασίας, και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην πτυχιακή εργασία. Επίσης έχω αναφέρει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επίσης, βεβαιώνω ότι αυτή η πτυχιακή εργασία προετοιμάστηκε από εμένα προσωπικά ειδικά για τις απαιτήσεις του προγράμματος σπουδών του Τμήματος Εφαρμοσμενων Μαθηματικων Επιστημων του ΕΜΠ .

(Υπογραφή)

....
Νικόλαος Ματσαβέλας

Περίληψη

Αυτό το έγγραφο αποτελεί μέρος της πρωτότυπης διπλωματικής μου εργασίας (αγγλικής έκδοσης) στο μεταπτυχιακό τμήμα των Εφαρμοσμένων Μαθηματικών Επιστημών της ΣΕΜΦΕ του ΕΜΠ, στην ελληνική γλώσσα.

Κύριο θέμα της διπλωματικής εργασίας ειναι η παραγοντοποίηση πινάκων μεγάλων διαστάσεων (γραμμών και στηλών) μέσω τυχαιοποιημένων αργορίθμων με την χρήση της Python.

Πιο συγκεκριμένα χρησιμοποιώντας ένα δειγματοπηπτικό τυχαίο πίνακα (τυποποιημενης κανονικής κατανομής) ορθοκανονικοποιούμε το γινόμενο του τελευταίου με τον αρχικό πίνακα μέσω της παραγοντοποίησης QR Gram-Schmidt και έπειτα χρησιμοποιούμε την ιδιάζουσα παραγοντοποίηση πίνακα για να ανακτήσουμε το μεγαλύτερο μέρος των επεξηγήσιμων στηλών χαμηλού στοχευμένου βαθμού k.

Στο πρώτο κεφάλαιο της παρούσας εργασίας περιγράφεται το πιθανολογικό πλαίσιο της ιδιάζουσας παραγοντοποίησης πίνακα και παρουσιάζονται τα βήματα του πρωτότυπου αλγορίθμου τυχαιοποιημένης SVD.

Στο δεύτερο κεφάλαιο παρουσιάζεται η τυχαιοποιημένη εκδοχή της ανάλυσης κυρίων συνιστωσών και της εύρωστης PCA.

Παρουσιάζοντας την παραδοσιακή PCA αντιλαμβανόμαστε οτι η επέκταση της σε πίνακες μεγάλων διαστάσεων, απαιτεί υπολογιστικό χρόνο και κόστος που ειναι σχεδόν αδύνατοι απο τους σύχρονους υπολογιστές.

Η τυχαιοποιημένη PCA ειναι το υπολογιστικό εκείνο εργαλείο οπου κανοντας χρήση της rSVD προβαίνει στην ανάλυση των κυρίαρχων (σημαντικότερων) κυρίων συνιστωσών.

Τελος παρουσιαζεται μερος της εύρωστης PCA όπου μεσω μιας κυρτής βελτιστοποιησης παραγοντοποιει ενα πίνακα A σε ενα πίνακα L χαμηλού βαθμού και ενα αραιό πίνακα ακραιων παρατηρήσεων S.

Στην μητέρα μου και στην αδερφή μου.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον καθηγητή κ. Ψαρράκο για την επιστημονική επίβλεψη και ουσιαστική καθοδήγηση αυτής της διπλωματικής εργασίας.

Περιεχόμενα

Κατάλογος Πινάκων

Κεφάλαιο 1

Τυχαιοποιημένη Ιδιάζουσα Παραγοντοποίηση Πίνακα

1.1 Ιδιάζουσα Παραγοντοποίηση Πίνακα

Η Ιδιάζουσα Παραγοντοποίηση Πίνακα, παραγοντοποιεί ένα πίνακα $\mathbf{A} \in \mathbb{R}^{m\times n}$ βαθμού $\mathbf{r} = \mathbf{n}$ στη μορφή $\mathbf{U\Sigma V^T}$:

$$\mathbf{A} = \underbrace{\begin{bmatrix} \vdots & \vdots & & \vdots \\ x_1 & x_2 & \dots & x_n \\ \vdots & \vdots & & \vdots \end{bmatrix}}_{m\times n} = \mathbf{U\Sigma V^T} =$$

$$\underbrace{\begin{bmatrix} \vdots & \vdots & & \vdots \\ u_1 & u_2 & \dots & u_m \\ \vdots & \vdots & & \vdots \end{bmatrix}}_{\mathbf{m\times m}} \underbrace{\begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_r \end{bmatrix}}_{\mathbf{r\times r}} \underbrace{\begin{bmatrix} \vdots & \vdots & & \vdots \\ v_1 & v_2 & \dots & v_n \\ \vdots & \vdots & & \vdots \end{bmatrix}^T}_{\mathbf{n\times n}},$$

όπου,$\mathbf{U}$ και $\mathbf{V}$ είναι ορθοκανονικοί πίνακες των αριστερών και δεξιών ιδιοδιανυσμάτων των πινάκων $\mathbf{AA^T}$ και $\mathbf{A^TA}$ αντίστοιχα.

Αν $\mathbf{A} \in \mathbb{R}^{m\times n}$ πίνακας τάξης $r > 0$, ορθογώνιος, και $m \times m$ και $n \times n$ οι διαστάσεις των πινάκων $\mathbf{U}$ και $\mathbf{V}$ τότε υπάρχει ένας πίνακας $\mathbf{A}$ τέτοιος ώστε $\mathbf{A} = \mathbf{U\Sigma V^T}$ και $\mathbf{\Sigma} = \mathbf{U^TAV}$, όπου ο $m \times n$ πίνακας $\mathbf{\Sigma}$ του οποίου η μορφή ποικίλει ανάλογα με τις παρακάτω προυποθέσεις:

1. $\mathbf{\Sigma}$ αν $\mathbf{r} = \mathbf{m} = \mathbf{n}$,
2. $\begin{bmatrix} \mathbf{\Sigma} & 0 \end{bmatrix}$ αν $\mathbf{r} = \mathbf{m} < \mathbf{n}$,
3. $\begin{bmatrix} \mathbf{\Sigma} \\ 0 \end{bmatrix}$ αν $\mathbf{r} = \mathbf{n} < \mathbf{m}$,

4. $\begin{bmatrix} \mathbf{\Sigma} & 0 \\ 0 & 0 \end{bmatrix}$ αν $\mathbf{r < m, r < n}$,

όπου , Σ είναι ένας $r \times r$ διαγώνιος πίνακας με θετικά στοιχεία στη κύρια διαγώνιο.

Τα στοιχεία του πίνακα $\mathbf{\Sigma^2}$ είναι θετικά καθώς αποτελούν τις τετραγωνισμένες τιμές των ιδιοτιμών των πινάκων $\mathbf{A^T A}$ και $\mathbf{A A^T}$.

1.1.1 Προσέγγιση Χαμηλού Βαθμού

Όπως μπορούμε να υποθέσουμε απο τον τίτλο της υποενότητας η Προσέγγιση Χαμηλού Βαθμού ενός πίνακα είναι η παραγοντοποίηση ενός πίνακα από έναν άλλον πίνακα χαμηλότερου βαθμού (Eckart–Young–Mirsky theorem).
Αυτή τη μέθοδο παραγοντοποίησης την συναντήσαμε και στη εισαγωγή αυτού του κεφαλαίου, για ένα πίνακα $\mathbf{A} \in \mathbb{R}^{m \times n}$, όπου παραγοντοποιήθηκε ως : $\mathbf{U \Sigma V^T}$.

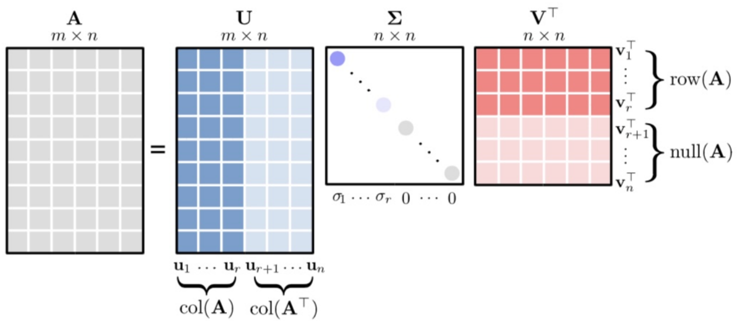
Πιο συγκεκριμένα ένας ορθογώνιος διαγώνιος πίνακας ιδιοτιμών $\mathbf{\Sigma} \in \mathbb{R}^{m \times n}$ περιέχει όλες τις ιδιοτιμές (θετικές) $\sigma_1 \geq \cdots \geq \sigma_n \geq 0$ όπου περιγράφουν το φάσμα του πίνακα (ή των δεδομένων). Με αυτη τη μέθοδο επιλέγουμε $r < rank(A)$ στήλες ώστε να οδηγηθούμε σε μια τετραγωνικής μορφής 'κουτσουρεμένη' (truncated SVD) ιδιάζουσα παραγοντοποίηση του πίνακα $\mathbf{A}$. Μια εναλλακτική παραλλαγή της παραγοντοποίησης SVD όπου υπολογίζει μόνο τα αριστερά ιδιάζοντα διανύσματα και ιδιάζουσες τιμές των αντίστοιχων n δεξιών ιδιαζόντων διανυσμάτων :

$$\mathbf{A} = \mathbf{U \Sigma V^T} = [\mathbf{u}_1, \ldots, \mathbf{u}_n]\mathrm{diag}(\sigma_1, \ldots, \sigma_n)[\mathbf{v}_1, \ldots, \mathbf{v}_n]^T$$

ονομάζεται 'λεπτή' ή όικονομική' SVD.

Εάν ο αριθμός των δεξιών ιδιαζόντων διανυσμάτων είναι μικρός (δηλ., $n \ll m$), αυτή η παραγοντοποίηση είναι πιο συμπαγής από τη πλήρες παραγοντοποιηση SVD. Η όικονομική' SVD είναι η προεπιλεγμένη μορφή της συνάρτησης βάσης svd() σε πάρα πολλές γλώσσες προγραμματισμού, όπως για παράδειγμα στη R ή στη Python.

Οι πίνακες χαμηλού βαθμού διαθέτουν μια ταξη μικρότερη από τη (αφηρημένη) διάσταση του χώρου, δηλαδή το r είναι μικρότερο από τον αριθμό των στηλών και των σειρών. Ως εκ τούτου, οι ιδιάζουσες τιμές $\{\sigma_i : i \geq r+1\}$ είναι μηδενικές και τα αντίστοιχα ιδιάζοντα διανύσματα εκτείνονται στο αριστερό και το δεξιό Null Space. Η έννοια της όικονομικής' SVD από ενα πίνακα χαμηλού βαθμού απεικονίζεται στο σχήμα 1.1.



Σχήμα 1.1: SVD Low Rank Decomposition

Σε πρακτικές εφαρμογές, οι πίνακες συχνά 'μολύνονται' από σφάλματα και ο πραγματικός βαθμός ενός πίνακα μπορεί να είναι μικρότερος από τον ακριβή βαθμό r. Σε αυτήν την περίπτωση, ο πίνακας μπορεί να προσεγγιστεί καλά συμπεριλαμβάνοντας μόνο εκείνα τα ιδιάζοντα διανύσματα που αντιστοιχούν σε μοναδικές ιδιαζουσες τιμές σημαντικού μεγέθους.

Ως εκ τούτου, είναι συχνά επιθυμητό να υπολογιστεί μια 'μειωμένη έκδοση' του SVD:

$$\mathbf{A_k} := \mathbf{U}_k \Sigma_k \mathbf{V}_k = [\mathbf{u}_1, \ldots, \mathbf{u}_k]\delta\iota\alpha\gamma(\sigma_1, \ldots, \sigma_k)[\mathbf{v}_1, \ldots, \mathbf{v}_k]^T,$$

όπου, με k συμβολίζουμε τον επιθυμητό βαθμό ή στόχευμενό βαθμό αυτής της προσέγγισης. Με άλλα λόγια, αυτή η μειωμένη μορφή της SVD μας επιτρέπει να εκφράσουμε τον πίνακα $\mathbf{A}$ κατά προσέγγιση με το άθροισμα των πινάκων k βαθμού.

$$\mathbf{A}_k \approx \sum_{i=1}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^T.$$

Η επιλογή του βέλτιστου στόχευμενου βαθμού k εξαρτάται σε μεγάλο βαθμό από την επιλογή μας αναφορικά με τις ανάγκες της ανάλυσης μας επάνω σε αυτόν τον πίνακα. Δηλαδή, κάποιος μπορεί είτε να ενδιαφερθεί για μια εξαιρετικά ακριβή παραγοντοποίηση ή με μια μικρότερη ανασυγκρότηση των αρχικών δεδομένων, ως μια χαμηλών διαστάσεων αναπαράσταση των κυρίαρχων χαρακτηριστικών, των δεδομενων.

Στην πρώτη περίπτωση, το k πρέπει να επιλεγεί έτσι ώστε να είναι όσο πιο κοντά στο πραγματικό βαθμό, ενώ στην τελευταία περίπτωση το k μπορεί να επιλεγεί ώστε να είναι πολύ μικρότερο. Η περικοπή ('κουτσούρεμα') αυτών των μικρών ιδιαζουσών τιμών στη ντετερμινιστική εκδοχή της SVD μας προσφέρει τη βέλτιστη προσέγγιση του αντίστοιχου βαθμού στόχου k. Συγκεκριμένα, το θεώρημα Eckart-Young (1936) αναφέρει ότι η χαμηλού βαθμού SVD παρέχει τον βέλτιστο βαθμό k ενός πίνακα με την έννοια των ελαχίστων τετραγώνων:

$$\mathbf{A}_{(k)} = \underset{rank(A'_k)=k}{\arg\min} \|\mathbf{A} - \mathbf{A'_k}\|.$$

Το σφάλμα ανακατασκευής τόσο ως προς την φασματική νόρμα όσο και ως προς την νόρμα Frobenius δίνεται από:

$$\|\mathbf{A}-\mathbf{A}_k\|_2 = \sigma_{k+1}(\mathbf{A}) \quad \text{και} \quad \|\mathbf{A}-\mathbf{A}_k\|_F = \sqrt{\sum_{j=k+1}^{\min(m,n)} \sigma_j^2(\mathbf{A})}.$$

Για δεδομένα υψηλών διαστάσεων, ωστόσο, η ' κουτσουρεμενη' SVD είναι δαπανηρή όσον αφορά το υπολογιστικό κόστος. Το κόστος υπολογισμού της πλήρους SVD ενός πίνακα $m \times n$ είναι της τάξης $\mathcal{O}(mn^2)$, από το οποίο μπορούν στη συνέχεια να εξαχθούν τα πρώτα k συστατικά: $\mathbf{A}_k$.

1.2 Πιθανολογικό Πλαίσιο Τυχαιοποιημένης Ιδιάζουσας Παραγοντοποίησης Πίνακα

Υποθέτουμε οτι έχουμε ένα πίνακα $\mathbf{A} \in \mathbb{R}^{m\times n}$ έχει βαθμό $\mathbf{r}$, όπου $\mathbf{r} \leq \min(\mathbf{m},\mathbf{n})$. Στη συνέχεια, γενικά, ο στόχος της προσέγγισης του πίνακα χαμηλού βαθμού είναι να βρεθούν δύο μικρότεροι πίνακες, τέτοιοι ώστε:

$$\underbrace{\mathbf{A}}_{m\times n} \approx \underbrace{\mathbf{E}}_{m\times r}\underbrace{\mathbf{F}}_{r\times n}, \tag{1.1}$$

όπου οι στήλες του πίνακα $\mathbf{E} \in \mathbb{R}^{m\times r}$ αποτελούν την γραμμική θήκη (span) του χώρου των στηλών (column space) του πίνακα $\mathbf{A}$, και οι γραμμές του πίνακα $\mathbf{F} \in \mathbb{R}^{r\times n}$ αποτελούν την γραμμική θήκη του χώρου των γραμμών (row space) του πίνακα $\mathbf{A}$.

Οι παραγοντοποιημένοι πίνακες $\mathbf{E}$ και $\mathbf{F}$ μπορούν στη συνέχεια να χρησιμοποιηθούν για να συνοψίσουμε ή να μας αποκαλυφθεί κάποια ενδιαφέρουσα δομή των δεδομένων. Επιπλέον, οι παραπάνω παραγοντοποιημένοι πίνακες μπορούν να χρησιμοποιηθούν για την αποτελεσματική αποθήκευση του μεγάλου πίνακα δεδομένων $\mathbf{A}$. Ειδικότερα, καθώς ο πίνακας $\mathbf{A}$ απαιτεί mn στοιχεία αποθήκευσης, οι πίνακες $\mathbf{E}$ και $\mathbf{F}$ απαιτούν μόνο $\mathbf{mr} + \mathbf{nr}$ στοιχεία αποθήκευσης.

Στην πράξη, οι περισσότεροι πίνακες δεδομένων μεγάλης κλίμακας δεν διαθέτουν ακριβή βαθμό $\mathbf{r}$. Αντίθετα με αυτό το γεγονός και εναλλακτικά, μας ενδιαφέρει συνήθως να βρούμε ένα βαθμό k όπου θα παράξει τον πίνακα $\mathbf{A_k}$, ο οποίος είναι όσο το δυνατόν πιο κοντά σε έναν αυθαίρετο πίνακα εισόδου $\mathbf{A}$ υπό την έννοια των ελαχίστων τετραγώνων. Αναφερόμαστε στο βαθμό **k** ως τον βαθμό στόχου όπως προείπαμε και στη προηγούμενη υποενότητα.

Ειδικότερα, η σύγχρονη ανάλυση δεδομένων και ο επιστημονικός υπολογισμός βασίζονται σε μεγάλο βαθμό σε προσεγγίσεις χαμηλού βαθμού, καθώς οι πίνακες χαμηλού βαθμού είναι πανταχού παρόντες σε όλες τις επιστήμες. Ωστόσο, στην εποχή των «δεδομένων μεγάλης κλίμακας», η εμφάνιση μαζικών δεδομένων αποτελεί σημαντική υπολογιστική πρόκληση για τους παραδοσιακούς ντετερμινιστικούς αλγόριθμους.

Στη συνέχεια, θα αναπτύξουμε το πιθανολογικό πλαίσιο μιας παραγοντοποιήσης SVD, όπως διατυπώθηκε από τους Halko et al. (2011), για τον υπολογισμό μιας σχεδόν βέλτιστης προσέγγισης χαμηλού βαθμού. Εννοιολογικά, αυτό το πλαίσιο χωρίζει την υπολογιστική εργασία σε δύο λογικά στάδια:

- **Βήμα A**: Δημιουργούμε έναν υπόχωρο μικρότερης διάστασης που προσεγγίζει το χώρο των στηλών **A**. Αυτό σημαίνει, ότι ο στόχος είναι να βρεθεί ένας πίνακας $\mathbf{Q} \in \mathbb{R}^{m\times k}$ με ορθοκανονικές στήλες, τέτοιος ώστε να ικανοποιείται η σχέση : $\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^{\mathbf{T}}\mathbf{A}$.
- **Βήμα B**: Σχηματίζουμε ενα μικρότερο πίνακα $\mathbf{B} := \mathbf{Q}^{\mathbf{T}}\mathbf{A} \in \mathbb{R}^{k\times n}$, δηλαδή, περιορίζουμε τον πίνακα εισόδου υψηλής διάστασης σε ενα χώρο χαμηλότερης διαστάσης με βέλτιστη βάση τον **Q**. O μικρότερος πίνακας **B** μπορεί στη συνέχεια να χρησιμοποιηθεί για τον υπολογισμό μιας επιθυμητής προσέγγισης χαμηλού βαθμού.

Το πρώτο υπολογιστικό στάδιο είναι εκεί όπου η τυχαιότητα μπαίνει στο παιχνίδι, ενώ το δεύτερο στάδιο είναι καθαρά ντετερμινιστικό. Στο επόμενο υποκεφάλαιο, θα περιγραφουμε τα δύο στάδια λεπτομερώς.

1.2.1 Γενικός Αλγόριθμος Τυχαιοποιημένης SVD

- **Βήμα Α: Υπολογισμός την κοντινότερης βέλτιστης βάσης**

  Πρώτον, στοχεύουμε να βρούμε μια κατα προσέγγιση βέλτιστη βάση $\mathbf{Q}$ για τον πίνακα $\mathbf{A}$ τέτοια ώστε να ικανοποιείται:

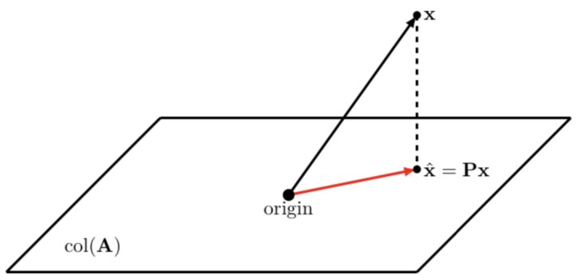
$$\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^{\mathbf{T}}\mathbf{A}. \tag{1.2}$$

Ο επιθυμητός βαθμός στόχος k υποθέτουμε οτι είναι $k \ll \min(m, n)$. Πιο συγκεκριμένα:

$$\mathbf{P} := \mathbf{Q}\mathbf{Q}^{\mathbf{T}}, \tag{1.3}$$

είναι ένας γραμμικός ορθογώνιος προβολέας (projector). Ένας τελεστής προβολής που αντιστοιχεί σε έναν γραμμικό υπόχωρο και μετατρέπει οποιοδήποτε διάνυσμα στην ορθογώνια προβολή του στον υποχώρο αυτόν.

Αυτό απεικονίζεται στο σχήμα 1.2, όπου ένα διάνυσμα $\mathbf{x}$ περιορίζεται στο χώρο των στηλών col(A).



Σχήμα 1.2: Γεωμετρική αναπαράσταση της ορθογώνιας προβολής του πίνακα $\mathbf{P}$. Το διάνυσμα $Ax \in \mathbb{R}^m$ ειναι περιορισμένο στο χώρο στηλών του $\mathbf{A}$, όπου $Px \in$ col$(\mathbf{A})$.

Η έννοια των τυχαίων προβολών (random projections), μπορεί να χρησιμοποιηθεί για τη δειγματοληψία του εύρους (χώρος στηλών) του πίνακα εισόδου $\mathbf{A}$ προκειμένου να κατασκευαστεί αποτελεσματικά ένας τέτοιος ορθογώνιος προβολέας.

Τέτοιες τυχαίες προβολές είναι αγνώστων δεδομένων και κατασκευάζονται αν σχεδιάσουμε πρώτα ένα σύνολο k τυχαίων διανυσμάτων $\{\omega_i\}k_i = 1$, για παράδειγμα, από την τυπική κανονική κατανομή. Η θεωρία πιθανοτήτων εγγυάται ότι τα τυχαία διανύσματα είναι γραμμικά ανεξάρτητα με μεγάλη πιθανότητα. Στη συνέχεια, ένα σύνολο τυχαίων προβολών $\{y_i\}k_i = 1$ υπολογίζεται με αντιστοίχιση του $\mathbf{A}$ σε χώρο χαμηλών διαστάσεων:

$$y_i := \mathbf{A}\omega_i, \quad \forall i = 1, 2, \ldots, k. \tag{1.4}$$

Με άλλα λόγια, αυτή η διαδικασία σχηματίζει ένα σύνολο ανεξάρτητων τυχαίων σταθμισμένων γραμμικών συνδυασμών των στηλών A και μειώνει τον αριθμό των στηλών από n σε k. Ενώ ο πίνακας εισόδου $\mathbf{A}$ συμπιέζεται, οι Ευκλείδειες αποστάσεις μεταξύ των αρχικών σημείων (δεδομένων) διατηρούνται κατα προσέγγιση.

Η εξίσωση 1.3 μπορεί να εκτελεστεί αποτελεσματικά και παράλληλα. Επομένως, ορίζουμε τον τυχαίο πίνακα δοκιμής $\Omega \in \mathbb{R}^{n\times k}$, ο οποίος θα προέρχεται από την τυπική κανονική κατανομή και οι στήλες του οποίου δίνονται από τα διανύσματα $\{\omega_i\}$. Ο πίνακας δειγματοληψίας $Y \in \mathbb{R}^{m\times k}$, που ονομάζεται και διαφορετικά ¨σκίτσο' (sketch), λαμβάνεται στη συνέχεια με πολλαπλασιασμό του πίνακα εισόδου με τον τυχαίο πίνακα δοκιμής

$$\mathbf{Y} := \mathbf{A\Omega}. \tag{1.5}$$

Μόλις υπολογιστεί ο πίνακας $\mathbf{Y}$, μένει μόνο να ορθοκανονικοποιήσουμε τις στήλες για να σχηματίσουμε μια κανονική βάση $Q \in \mathbb{R}^{m\times k}$. Αυτό μπορεί να επιτευχθεί αποτελεσματικά χρησιμοποιώντας τη παραγοντοποίηση QR (Gram–Schmidt process)

$$\mathbf{Y} =: \mathbf{QR},$$

και έτσι ότι η εξίσωση 1.2 ικανοποιείται.

- **Βήμα Β : Υπολογισμός μικρότερου πίνακα**

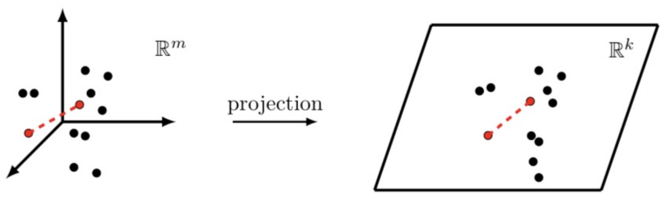
  Στη συνέχεια, δεδομένης της κατα προσέγγισης βέλτιστης βάσης Q, στοχεύουμε να βρούμε έναν μικρότερο πίνακα $B \in \mathbb{R}^{k\times n}$. Προβάλλουμε λοιπόν τον αρχικό μας πίνακα εισόδου υψηλής διάστασης A σε ένα χώρο χαμηλών διαστάσεων:

$$\mathbf{B} := \mathbf{Q^T A}. \tag{1.6}$$

  Γεωμετρικά, πρόκειται για μια προβολή (δηλαδή, έναν γραμμικό μετασχηματισμό) που παίρνει σημεία σε έναν χώρο υψηλής διάστασης σε αντίστοιχα σημεία σε ένα χώρο χαμηλής διάστασης, όπως απεικονίζεται στην εικόνα 1.3.

  Αυτή η διαδικασία διατηρεί τη γεωμετρική δομή των δεδομένων με ευκλείδεια έννοια, δηλαδή διατηρείται το μήκος των προβαλλόμενων διανυσμάτων καθώς και οι γωνίες μεταξύ των προβαλλόμενων διανυσμάτων. Αυτό οφείλεται στο οτι τα εσωτερικά γινόμενα παραμένουν αμετάβλητα (invariance of inner products) (Trefethen και Bau 1997). Η αντικατάσταση της εξίσωσης 1.6 στη 1.2 αποδίδει την ακόλουθη προσέγγιση χαμηλού βαθμού

$$\underbrace{\mathbf{A}}_{m\times n} \approx \underbrace{\mathbf{Q}}_{m\times k}\underbrace{\mathbf{B}}_{k\times n}.$$



Σχήμα 1.3: Προβολή σε ένα υπόχωρο χαμηλότερης διάστασης

Αυτή η παραγοντοποίηση αναφέρεται ως παραγοντοποίηση QB. Στη συνέχεια, ο μικρότερος πίνακας B μπορεί να χρησιμοποιηθεί για τον υπολογισμό μιας παραγοντοποίησης πίνακα χρησιμοποιώντας τον παραδοσιακό αλγόριθμο παραγοντοποίησης SVD.

Η διαισθητική προσέγγιση του γενικού τυχαίου αλγορίθμου μπορεί να εφαρμοστεί σε γενικούς πίνακες. Παραλείποντας υπολογιστικές λεπτομέρειες προς το παρόν, ονομάζουμε τον αλγόριθμο Proto-Algorithm.. Αυτός ο απλός αλγόριθμος δεν είναι καθόλου νέος. Είναι ουσιαστικά το πρώτο βήμα μιας επανάληπτικής διαδικασίας σε εναν υπόχωρο, με έναν τυχαίο αρχικό υπόχωρο. Η καινοτομία προέρχεται από την πρόσθετη παρατήρηση ότι ο αρχικός υπόχωρος θα πρέπει να έχει λίγο υψηλότερη διάσταση από τον αμετάβλητο υποχώρο (invariant sub-space) που προσπαθούμε να προσεγγίσουμε.

Με αυτήν την αναθεώρηση, συμβαίνει συχνά, ότι δεν απαιτείται περαιτέρω επανάληψη για την απόκτηση λύσης υψηλής ποιότητας για την εξίσωση:

$$\mathbf{A} - \mathbf{Q}\mathbf{Q}^{\mathbf{H}}\mathbf{A}.$$

Η βασική πρόκληση στην παραγωγή προσεγγίσεων πινάκων χαμηλού βαθμού είναι μια πρωτόγονη ερώτηση, που ονομάζουμε πρόβλημα προσέγγισης σταθερής ακρίβειας. Ας υποθέσουμε ότι μας δίνεται ένας πίνακας $\mathbf{A}$ και μια θετική ανοχή σφάλματος ε. Αναζητούμε έναν πίνακα $\mathbf{Q}$ με $k = k(\varepsilon)$ ορθοκανονικές στήλες έτσι ώστε:

$$|||\mathbf{A} - \mathbf{Q}\mathbf{Q}^{\mathbf{H}}\mathbf{A}|| \leq \varepsilon, \tag{1.7}$$

όπου, $|| \cdot ||$ συμβολίζουμε την l_2 νόρμα. Το εύρος του πίνακα $\mathbf{Q}$ είναι ένας k-διάστασης υπόχωρος ο οποίος περιέχει το μεγαλύτερο ποσοστό πληροφορίας του αρχικού πίνακα A, και επομενως για λογους υπολογιστικού κόστους θα θέλαμε το k να είναι οσο το δυνατόν μικρότερο.

Κεφάλαιο 2

Τυχαιοποιημένη PCA & και Εύρωστη PCA

2.1 Περίληψη PCA

Η μείωση της διάστασης είναι μια θεμελιώδης έννοια στη σύγχρονη ανάλυση δεδομένων. Η ιδέα είναι να εκμεταλλευτούμε τις σχέσεις μεταξύ σημείων σε ένα χώρο υψηλών διαστάσεων, προκειμένου να κατασκευάσουμε ορισμένες περιλήψεις χαμηλών διαστάσεων. Αυτή η διαδικασία στοχεύει στην εξάλειψη των περιττών μεταβλητών (λόγω συσχέτισης), διατηρώντας παράλληλα τα ενδιαφέροντα χαρακτηριστικά των δεδομένων (Burges 2010). Η μείωση διαστάσης χρησιμοποιείται για τη βελτίωση της υπολογιστικής ικανότητας, για την εξαγωγή ενδιαφέροντων χαρακτηριστικών και την οπτικοποίηση δεδομένων που αποτελούνται από πολλές αλληλένδετες μεταβλητές.

Η πιο σημαντική τεχνική γραμμικής μείωσης διαστάσεων είναι η ανάλυση κύριας συνιστώσας (PCA), που διατυπώθηκε αρχικά από τους Pearson (1901) και Hotelling (1933). Η PCA παίζει σημαντικό ρόλο, ιδίως, λόγω της απλής γεωμετρικής ερμηνείας του. Η βιβλιογραφία από τον Jolliffe (2002) παρέχει μια ολοκληρωμένη εισαγωγή στο PCA. Η ανάλυση των κύριων συνιστωσών στοχεύει στην εύρεση ενός νέου συνόλου μη συσχετισμένων μεταβλητών. Οι επονομαζόμενες κύριες συνιστώσες (Principal Components) είναι κατασκευασμένες έτσι ώστε η πρώτη κύρια συνιστώσα να εξηγεί τις περισσότερες διακυμάνσεις των δεδομένων. Η δεύτερη κύρια συνιστώσα το μεγαλύτερο μέρος της εναπομείνουσας διακύμανσης και ούτω καθεξής.

Αυτή η ιδιότητα διασφαλίζει ότι οι συνιστώσες καταγράφουν διαδοχικά το μεγαλύτερο μέρος της συνολικής επεξηγηματικότητας (πληροφοριών) που υπάρχει στα δεδομένα. Στην πράξη, συχνά στοχεύουμε να διατηρήσουμε μόνο λίγες συνιστώσες που αποτυπώνουν ένα ¨καλό' ποσοστό της επεξηγηματικότητας, όπου με τον όρο ¨καλό' αναφερομαστε στο εκαστοτε πινακα $m \times n$.

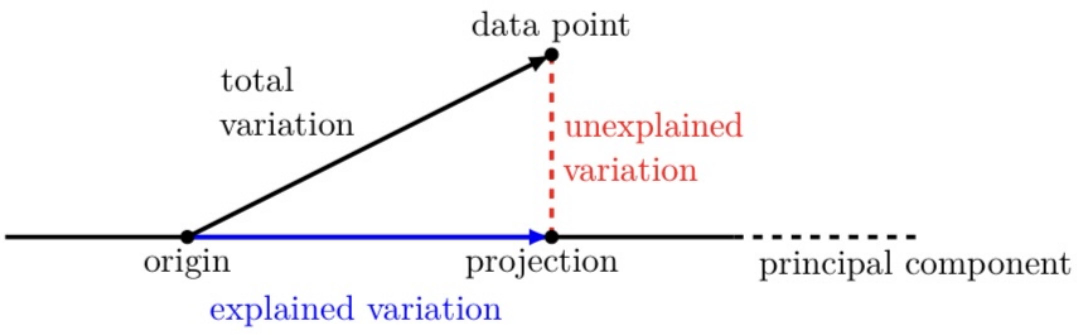
Πιο συγκεκριμένα, υποθέτουμε έναν πίνακα δεδομένων $\mathbf{X} \in \mathbb{R}^{m \times n}$ με παρατηρήσεις m και n

μεταβλητές (συνήθως τυποποιημένες (scaled)). Στη συνέχεια, οι κύριες συνιστώσες μπορούν να εκφραστούν ως σταθμισμένος γραμμικός συνδυασμός των αρχικών μεταβλητών:

$$z_i = \mathbf{X} w_i,$$

όπου, με $z_i \in \mathbb{R}^m$ συμβολίζουμε την i^{th} κύρια συνιστώσα. Το διάνυσμα $w_i \in \mathbb{R}^n$ αποτελείται απο την i^{th} συνιστώσα κατεύθυνση, όπου τα στοιχεία του $w_i = [w_1, \ldots, w_n]^T$ είναι οι συντελεστές της κάθε κύριας συνιστώσας.

Το πρόβλημα τώρα είναι να βρούμε ένα διάνυσμα w_1 τέτοιο ώστε ο πρώτος κύριος συντελεστής z_1 να εξηγεί το μεγαλύτερο μερος της διακύμανσης περισσότερες των δεδομένων. Μαθηματικά, αυτό το πρόβλημα μπορεί να διατυπωθεί είτε ως πρόβλημα ελαχίστων τετραγώνων είτε ως πρόβλημα μεγιστοποίησης της διακύμανσης (Cunningham and Ghahramani 2015). Και τούτο διότι η συνολική διακύμανση των δεδομένων ισούται με το άθροισμα της επεξηγήσιμης και μη επεξηγήσιμης διακύμανσης των δεδομένων (Jolliffe 2002), όπως περιγράφεται στην εικόνα παρακάτω.



Σχήμα 2.1: Οι ΚΣ μπορούν να ληφθούν είτε με τη μεγιστοποίηση της διακύμανσης είτε με την ελαχιστοποίηση της μη επεξηγήσιμης (ανεξήγητης) διακύμανσης (καταλοιπα ελαχίστων τετραγώνων) των δεδομένων

Ακολουθούμε την τελευταία άποψη και μεγιστοποιούμε τη διακύμανση του πρώτου κύριου στοιχείου $z_i = \mathbf{X} w_i$, υπό τον περιορισμό της κανονικοποίησης: $||w||_2^2 = 1$.

2.2 Τυχαιοποιημένη PCA

Η ιδιάζουσα παραγοντοποίηση πίνακα μας παρέχει μια υπολογιστικά επαρκής και αριθμητικά σταθερή προσέγγιση της ανάλυσης κυρίων συνιστωσών. Πιο συγκεκριμένα η ανάλυση ιδιοτιμών και ιδιοδιανυσμάτων του εσωτερικού και εξωτερικού γινομένου του : $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ μπορεί να σχετιστεί με την SVD ως :

$$\mathbf{X}^\top \mathbf{X} = (\mathbf{V}\mathbf{\Sigma}\mathbf{U}^\top)(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top) = \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^\top, \tag{2.1}$$

$$\mathbf{X}\mathbf{X}^\top = (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top)(\mathbf{V}\mathbf{\Sigma}\mathbf{U}^\top) = \mathbf{U}\mathbf{\Sigma}^2\mathbf{U}^\top. \tag{2.2}$$

Από αυτό προκύπτει ότι οι ιδιοτιμές είναι ίσες με τις τετραγωνισμένες ιδιάζουσες τιμές. Έτσι, ανακτάμε τις ιδιοτιμές του δείγματος συνδιακύμανσης δείγματος

$$\mathbf{C} := (m-1)^{-1}\mathbf{X}^\top\mathbf{X},$$

ως

$$\mathbf{\Lambda} = (m-1)^{-1}\mathbf{\Sigma}^2.$$

Τα αριστερά ιδιάζοντα διανύσματα $\mathbf{U}$ αντιστοιχούν στα ιδιοδιανύσματα του εξωτερικού γινομένου $\mathbf{X}\mathbf{X}^\top$, από την (2.2) και τα δεξιά ιδιάζοντα διανύσματα $\mathbf{V}$ λόγω της (2.1) αντιστοιχούν στα ιδιοδιανύσματα του εσωτερικού γινομένου : $\mathbf{X}^\top\mathbf{X}$.

Συμπερασματικά απο τα παραπάνω μπορούμε να σχηματίσουμε τον πίνακα προβολής ή οπως επικρατεί στην ξένη βιβλιογραφία πίνακα περιστροφής: $\mathbf{W} := \mathbf{V}$. Έχοντας δείξει τη σύνδεση μεταξύ της ιδιάζουσας παραγοντοποίησης πίνακα και της ανάλυσης των κύριων συνιστωσών, είναι εύκολο να δείξουμε ότι τα κύριες συνιστώσες μπορούν να υπολογιστούν ως:

$$\mathbf{Z} := \mathbf{X}\mathbf{W} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top\mathbf{W} = \mathbf{U}\mathbf{\Sigma}. \tag{2.3}$$

Η τυχαιοποιημένη μέθοδος ιδιάζουσας παραγοντοποίησης πίνακα μπορεί να χρησιμοποιηθεί αποτελεσματικά στη προσέγγιση των κυρίαρχων k κυρίων συνιστωσών. Αυτή η προσέγγιση (2.3) αναφέρεται ως τυχαιοποιημένη ανάλυση κυρίων συνιστωσών και πρωτάθηκε από τον Rokhlin 2009 , και αργότερα από τον Halko 2011 ο οποίος μας προσφέρει μερικές πρόσθετες ενδιαφέρουσες λεπτομέρειες αναφορικά με την Randomized PCA σε δεδομένα μεγάλης κλίμακας.

```
ΑΛΓ:ΑΛΓΟΡΙΘΜΟΣ RPCA

Εισάγω: Κανονικοποιημένο πίνακα X διαστάσεων $m \times n$, με στοχευμένο βαθμό
$k < \min m, n$, $p$ παράμετρο υπερδειγματοληψίας $q$ αριθμό επαναλήψεων.

  1. $U$,$\Sigma$,$W$ = rsvd(X,k,p,q).
                                                        Τυχαιοποιημένη SVD
  2. $\Lambda = \Sigma^2/m - 1$.
                                                        ανάκτηση ιδιοτιμών
  3. $Z = U\Sigma$
                                          υπολογισμός $k$ κυρίων συνιστωσών

Εξάγω: Πίνακες $W \in \mathbb{R}^{n\times k}$, $\Lambda \in \mathbb{R}^{k\times k}$, ανδ $Z \in \mathbb{R}^{m\times k}$
```

Σχήμα 2.2: Αλγόριθμος τυχαιοποιημένης PCA

2.3 Εύρωστη PCA

Μέχρι στιγμής, έχουμε δει προσεγγίσεις πινάκων ως παραγοντοποίηση ενός δεδομένου πίνακα σε ένα γινόμενο μικρότερων (χαμηλού βαθμού) πινάκων. Ωστόσο, υπάρχει μια άλλη ενδιαφέρουσα κατηγορία παραγοντοποίησης, η οποία παραγοντοποιεί ένα πίνακα σε ενα πίνακα χαμηλού βαθμού, σε ενα πίνακα αραιό και σε ενα πίνακα διαταραχής.

Τέτοιου είδους παραγοντοποιήσεις βασίζονται στην ανάγκη για εύρωστες μεθόδους που μπορούν να εξηγήσουν αποτελεσματικότερα τα δεδομένα διαταραχής. Πράγματι, η απόρριψη των ακραίων παρατηρήσεων είναι κρίσιμη σε πολλές εφαρμογές, καθώς τα δεδομένα σπάνια δεν περιέχουν στοιχεία διαταραχής.

Οι συγκεκριμένες μέθοδοι εύρωστης παραγοντοποίησης, παραγοντοποιούν ένα πίνακα ως εξής:

$$\underset{m\times n}{\mathbf{A}} \approx \underset{m\times n}{\mathbf{L}} + \underset{m\times n}{\mathbf{S}} + \underset{m\times n}{\mathbf{E}},$$

όπου , $\mathbf{L} \in \mathbb{R}^{m\times n}$ είναι ενας πίνκας χαμηλου βαθμού , $\mathbf{S} \in \mathbb{R}^{m\times n}$ ένας αραιός πίνακας, και $\mathbf{E} \in \mathbb{R}^{m\times n}$ ο πίνακας διαταραχής (θορύβου).

Ο πίνακας $\mathbf{S}$ περιέχει τις ακραίες παρατηρήσεις (outliers) του αρχικού πίνακα $\mathbf{A}$.

Παρακάτω θα ασχοληθούμε μόνο με την ειδική περίπτωση της εύρωστης παραγοντοποίησης

PCA , όπου περιλαμβάνει τους πίνακες $\mathbf{A} = \mathbf{L} + \mathbf{S}$, δηλαδή την παραγοντοποίηση σε έναν πίνακα χαμηλού βαθμού και σε έναν αραιό. Αυτή η μορφή προσθετικής παραγοντοποίησης χαρακτηρίζεται ως εύρωστη ανάλυση κύριων συνιστωσών (RPCA) και η αξιοσημείωτη ικανότητά της να διαχωρίζει πίνακες υψηλής διάστασης σε χαμηλού βαθμού και αραιή συνιστώσα καθιστά τη Randomized Principal Component Analysis ένα σημαντικό εργαλείο για την επιστήμη των δεδομένων.

Η προσθετική παραγοντοποίηση, ωστόσο, είναι διαφορετική από τις κλασικές ισχυρές μεθόδους PCA που είναι γνωστές στη στατιστική βιβλιογραφία. Αυτές οι τεχνικές αφορούν τον υπολογισμό ισχυρών εκτιμητών για τον εμπειρικό πίνακα συνδιακύμανσης, όπως παρουσιάζεται, απο τους Hubert 2005 και Croux 2005. Ενώ η παραδοσιακή κύρια ανάλυση κυρίων συνιστωσών ελαχιστοποιεί την φασματική νόρμα του σφάλματος ανάκτησης του αρχικού πίνακα, η εύρωστη PCA στοχεύει στην ανάκτηση του αρχικού πίνακα μέσω ενός χαμηλού βαθμού που περιλαμβάνει και τις ακραίες παρατηρήσεις.

Οι Candes et al 2011, απέδειξαν ότι είναι δυνατόν να διαχωριστεί ακριβώς ένας τέτοιος πίνακας δεδομένων $\mathbf{A} \in \mathbb{R}^{m \times n}$ τόσο ως προς τις συνιστώσες χαμηλού βαθμού όσο και ως πρός τις αραιές συνιστώσες του, κάτω από μάλλον ευρείες υποθέσεις. Αυτό μπορεί να επιτευχθεί ανάγοντας τον πρόβλημα ενός τετοιου διαχωρισμού ως μια επίλυση ενός προβλήματος κυρτής βελτιστοποίησης που ονομάζεται Principal Component Pursuit.

Ο σκοπός ειναι η ελαχιστοποίηση της φυσικής νόρμας ενος συνδυασμού σταθμισμένων πινάκων:

$$\min_{\mathbf{L},\mathbf{S}} \operatorname{rank}(\mathbf{L}), + \|\mathbf{S}\|_0 \text{ υπό τους περιορισμούς } \mathbf{L} + \mathbf{S} = \mathbf{X}, \tag{2.4}$$

Ωστόσο, ούτε το $\operatorname{rank}(\mathbf{L})$ ούτε το $\|\mathbf{S}\|_0$ είναι κυρτοί όροι και αυτό δεν είναι πρόβλημα βελτιστοποίησης που μπορεί να αντιμετωπιστεί. Παρόμοια με το (compressed sensing problem), είναι δυνατό να επιλυθεί ως κυρτό, για τα βέλτιστα $\mathbf{L}$ και $\mathbf{S}$ με μεγάλη πιθανότητα, χαλαρώνώντας τις υποθέσεις κυρτότητας της (2.4) :

$$\min_{L,S} \|(L)\|_\star + \lambda \|S\|_1 \text{ υπό τους περιορισμούς } L + S = X, \tag{2.5}$$

Εδώ με $\|\cdot\|_\star$ συμβολίζουμε την φυσική νόρμα των ιδιαζουσών τιμών, οπου αποτελεί μια προσέγγιση του βαθμού του αρχικού πινακα Q. Αξιοσημείωτα, η λύση της 2.5) συγκλίνει στη λύση της 2.4) με υψηλή πιθανότητα, αν $\lambda = 1/\sqrt{m,n}$, όπου n και m είναι οι διαστάσεις του αρχικού πίνακα X, δοθέντος οτι ο L και ο S ικανοποιούν τις :

1. L δεν είναι αραιός,

2. S δεν ειναι χαμηλού βαθμού. Εδώ υποθέτουμε ότι οι καταχωρήσεις κατανέμονται τυχαία έτσι ώστε να μην έχουν χώρο στηλών (column space) χαμηλής διάστασης.

Το πρόβλημα κυρτής βελτιστοποίησης 2.4) είναι γνωστό ως principal component pursuit (PCP), και μπορεί να λυθεί χρησιμοποιώντας τον προσαυξημένο (augmented) αλγόριθμο Lagrange (ALM). Συγκεκριμένα, μπορεί να κατασκευαστεί μια προσπαυξημένη συνάρτηση Lagrange:

$$\mathcal{L}(\mathbf{L},\mathbf{S},\mathbf{Z},\mu) = \|\mathbf{L}\|_* + \lambda\|\mathbf{S}\|_1 + \langle \mathbf{Z}, \mathbf{A}-\mathbf{L}-\mathbf{S}\rangle + \frac{\mu}{2}\|\mathbf{A}-\mathbf{L}-\mathbf{S}\|_F^2, \tag{2.6}$$

όπου , μ και λ είναι θετικές σταθερές , και Z ειναι ο πολλαπλασιαστής Lagrange.

Επί πρόσθετα, ως $\langle\cdot,\cdot\rangle$ ορίζουμε $\langle A,B\rangle := \text{trace}(A^\top B)$.

Η μέθοδος των προσπαυξημένων πολλαπλασιαστών Lagrange μπορεί να χρησιμοποιηθεί για την επίλυση του προβλήματος βελτιστοποίησης Bertsekas 1999. Οι Lin et al 2011 έχουν προτείνει έναν ακριβή και έναν μη ακριβή αλγόριθμο για την επίλυση της εξίσωσης 2.6. Παρακάτω κλείνοντας θα αναφερθούμε μόνο με την τελευταία προσέγγιση.

Πιο συγκεκριμένα, ο μη ακριβής αλγόριθμος αποφεύγει να λύσει το πρόβλημα :

$$L_{i+1}, E_{i+1} = \arg\min_{L,E} \mathcal{L}(L, E, Z_i, \mu_k),$$

αλλα εναλλακτικά επιλύει το προβλημα σπαζοντας το αρχικο σε δυο και το λύνει επαναληπτικά με βήμα i:

$$L_{i+1} = \arg\min_{L} \mathcal{L}(L, _i, Z_i, \mu_k), \tag{2.7}$$

και

$$E_{i+1} = \arg\min_{E} \mathcal{L}(L_{i+1}, E, Z_i, \mu_k).$$

Μια γενική λύση, θα επιλύσει τα L_k και S_k που ελαχιστοποιούν $\mathcal{L}$, ενημερώνοντας (updating) τους πολλαπλασιαστές Lagrange

$$Y_{k+1} = Y_k + \mu(X - L_k - S_k),$$

και επαναλαμβάνουμε μέχρι η λύση να συγκλίνει.

Ωστόσο, για αυτό το συγκεκριμένο σύστημα, με τη μέθοδο εναλλασσόμενων κατευθύνσεων, οι Lin et al 2011 παρείχαν μια απλή διαδικασία εύρεσης των L και S. Αρχικά, κατασκευάζουμε τον τελεστή συρρίκνωσης:

$$\mathcal{S}_\tau(x) = \text{sign}(x)\max(|x| - \tau, 0).$$

Στη συνέχεια κατασκευάζουμε τον τελεστής ανώτατου ορίου των ιδιαζουσών τιμών:

$$SVT_\tau(X) = U\mathcal{S}_\tau(\Sigma)V^\star,$$

και τέλος χρησιμοποιούμε τον $\mathcal{S}_\tau$ και τους πίνακες SVT επαναληπτικά, επιλύοντας τους L και S.

Παράρτημα Α′

Παράρτημα

Α′.1 Βιβλιογραφία

Βιβλιογραφία

[1] Rishi Advani & Sean O'Hagan (2021). *Efficient Algorithms for Constructing an Interpolative Decomposition.* ArXiv

[2] Rishi Advani & Sean O'Hagan (2006). *Fast Monte Carlo algorithms for matrices 2: Computing a low rank approximation to a matrix.* Siam

[3] Daniel Ahfock, William J. Astle & Sylvia Richardson (2019) *Statistical properties of sketching algorithms.* ArXiv

[4] Robert Andrew & Nicholas Dingle (2014). *Implementing QR factorization updating algorithms on GPUs.* Elsevier

[5] Rajendra Bhatia (2007). *Positive Definite Matrices.* Princeton University Press

[6] Rajendra Bhatia (1997). *Matrix Analysis.* Springer

[7] Claude Brezinski (1975). *Computation of Eigenelements of a Matrix by the ϵ-algorithm.* Elsevier

[8] Steven L. Brunton, & J. Nathan Kutz (2017). *Data Driven Science & Engineering Machine Learning, Dynamical Systems, and Control* . University of Washington

[9] Thierry Bouwmans, Necdet Serhat Aybat , El-hadi Zahzah (2016). *Handbook of Robust low-rank & sparse matrix decomposition (Application in image & video processing).* CRC Press

[10] Christopher J. C. Burges (2010). *Dimension Reduction: A Guided Tour.* Now:The Essence of Knowledge

[11] Emmanuel J. Candes, Xiaodong Li, Yi Ma & John Wright, (2009) *Robust Principal Component Analysis?.* ArXiv

[12] Ryan Compton (2010). *A Randomized Algorithm for PCA.*

[13] Amit Deshpande , Santosh Vempala. *Adaptive Sampling and Fast Low-Rank Matrix Approximation.* ArXiv

[14] Petros Drineas & Michael W. Mahoney (2005). *On the Nystrom Method for Approximating a Gram Matrix for Improved Kernel-Based Learning.* Journal of Machine Learning Research

[15] Petros Drineas & Michael W. Mahoney (2017) *Lectures on Randomized Numerical Linear Algebra.* ArXiv

[16] Kui Dua & Yimin Wei (2005) *Statistical properties of sketching algorithms.* Elsevier

[17] Carl Eckart & Gale Young (1936). *The approximation of one matrix by another of lower rank* . Psychometrika

[18] N. Benjamin Erichson, Sergey Voronin, Steven L. Brunton & J. Nathan Kutz (2019) *Randomized Matrix Decompositions Using R.* ArXiv

[19] Nick Feller (2016). *Basics of Matrix Algebra for Statistics with R.* CRC Press

[20] Xu Feng, Wenjian Yu & Yaohang Li (2018) *Faster Matrix Completion Using Randomized SVD.* ArXiv

[21] Xu Feng, Yuyang Xie ,Mingye Song, Wenjian Yu & Jie Tang (2018). *Fast Randomized PCA for Sparse Data.* ArXiv

[22] James E.Gentle (2017). *Matrix Algebra,Theory, Computations and Applications in Statistics 2nd Edition.* Springer Texts in Statistics.

[23] Yanina Gimeneza, & Guido Giussani (2017) *Searching for the core variables in principal components analysis.* ArXiv

[24] Gene H.Golub & Charles F.Van Loan (2013). *Matrix Computations 4th Edition.* The Johns Hopkins University Press

[25] M.Gu (2014) *Subspace Iteration randomization & singular value problems.* ArXiv

[26] N. Halko, P. G. Martinsson & J. A. Tropp (2010) *Finding Structure with randomness: Probabilistic algorithms for constructing approximate matrix decomposition* . ArXiv

[27] Sven Hammarling, (1985). *The Singular Value Decomposition in Multivariate Statistics.* MIT Press

[28] *Analysis of a Complex of Statistical Variables into Principal Components..* Journal of Educational Psychology.

[29] Higham, Nicholas J. & Tisseur, Françoise (2006). *A block algorithm for matrix 1-norm estimation, with an application to 1-norm pseudospectra.* Society for Industrial and Applied Mathematics.

[30] Roger A.Horn & Charles R.Johnson (1994). *Matrix Analysis.* Cambridge University Press

[31] Arne Jensen (2009) *Lecture Notes on Spectra and Pseudospectra of Matrices and Operators.* Department of Mathematical Sciences Aalborg University

[32] Qiang Ji (2020). *Probabilistic Graphical Models for Computer Vision.* Elsevier

[33] I.T Jolliffe (2007). *Principal Component Analysis, 2nd Edition.* Springer

[34] Sanjiv Kumar , Mehryar Mohri & Ameet Talka (2012) *Sampling Methods for the Nystrom Method.* Journal of Machine Learning Research

[35] Giacomo Livan, Marcel Novaes &Pierpaolo Vivo (2017). *Introduction to Random Matrices Theory and Practice.* ArXiv

[36] Michael W. Mahoneya, & Petros Drineas. *CUR matrix decompositions for improved data analysis.* Pnas

[37] Per-Gunnar Martinsson, Vladimir Rokhlin , Mark Tygert (2011). *A randomized algorithm for the decomposition of matrices* . Applied and Computational Harmonic Analysis,(Elsevier)

[38] Per-Gunnar Martinsson, Vladimir Rokhlin, Yoel Shkolnisky, & Mark Tygert (2014). *A software package for low-rank approximation of matrices via interpolative decompositions.*

[39] Per-Gunnar Martinsson (2021) *Randomized Numerical Linear Algebra: Foundations & Algorithms.* ArXiv

[40] Per-Gunnar Martinsson (2019) *Randomized methods for matrix computations.* ArXiv

[41] Carl D.Meyer. *Matrix Analysis and Applied Linear Algebra.* SIAM

[42] Tim Moon and Jack Poulson,(2016). *Accelerating eigenvector and pseudospectra computation using blocked multi-shift triangular solves.* ArXiv

[43] Rajeev Motwani & Prabhakar Raghavan (1995) *Randomized Algorithms.* Cambridge University Press

[44] Panayiotis J. Psarrakos (2000). *Numerical range of linear pencils.* Elsevier

[45] Vladimir Rokhlin, Arthur Szlam, & Mark Tygert (2009). *A Randomized Algorithm for Principal Component Analysis* . ArXiv

[46] Mark Rudelson & Roman Vershynin (2006) *Sampling from large matrices: An approach through geometric functional analysis.* ArXiv

[47] James Sethna (2016) *Random Matrix Theory.* Sethna, "Entropy, Order Parameters, and Complexity"

[48] James Schott (2017). *Matrix Analysis for Statistics.* Wiley Series in Probability & Statistics

[49] Lloyd N.Trefethen (1999). *Computation of Pseudospectra.* Cambridge University Press

[50] Lloyd N. Trefethen & Mark Embree (2005) *Spectra & Pseudospectra (The Behavior of Nonormal Matrices & Operators).* Princeton University Press

[51] Jake VanderPlas (2017). *Python Data Science Handbook.* O'Reilly

[52] Hrishikesh D.Vinod (2016). *Hands-on Matrix Algebra Using R.* World Scientific

[53] Sergey Voronin & Per-Gunnar Martinsson,(2016). *An implementation of randomized algorithms for computing the singular value, interpolative, and CUR decompositions of matrices on multi-core and GPU architectures.* ArXiv

[54] Thomas G. Wright & Lloyd N. Trefethen (2002). *Pseudospectra of rectangular matrices.* IMA Journal of Numerical Analysis

[55] John Wright, Yigang Peng, Yi Ma , Arvind Ganesh & Shankar Rao (2009) *Robust Principal Component Analysis: Exact Recovery of Corrupted Low-Rank Matrices by Convex Optimization.* ArXiv

Βιβλιογραφία

Randomized Matrix Decomposition Algorithms using Python

ΝΙΚΟΛΑΟΣ ΜΑΤΣΑΒΕΛΑΣ

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΑΘΗΝΑ, ΣΕΠΤΕΜΒΡΙΟΣ 2021

Θέση barcode

Randomized Matrix Decomposition Algorithms using Python

ΝΙΚΟΛΑΟΣ ΜΑΤΣΑΒΕΛΑΣ

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΠΛΩΜΤΙΚΗ ΕΡΓΑΣΙΑ

ΑΘΗΝΑ, ΣΕΠΤΕΜΒΡΙΟΣ 2021

Θέση barcode

ΜΕΤΑΠΠΤΥΧΙΑΚΗ ΔΙΠΛΩΜΤΙΚΗ ΕΡΓΑΣΙΑ

Randomized Matrix Decomposition Algorithms using Python

ΝΙΚΟΛΑΟΣ ΜΑΤΣΑΒΕΛΑΣ

ΑΘΗΝΑ

ΣΕΠΤΕΜΒΡΙΟΣ 2021

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΠΛΩΜΤΙΚΗ ΕΡΓΑΣΙΑ

Randomized Matrix Decomposition Algorithms using Python

ΝΙΚΟΛΑΟΣ ΜΑΤΣΑΒΕΛΑΣ

ΑΘΗΝΑ

ΣΕΠΤΕΜΒΡΙΟΣ 2021

A.2 Python Code for this thesis

Thesis NTUA MS.c Applied Mathematics

September 23, 2021

[1]:
```
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
from pseudopy import *
from matplotlib import pyplot
from scipy import linalg
from scipy.linalg import eigvals
from matplotlib.patches import Circle
from matplotlib.collections import PatchCollection
import matplotlib.pyplot as plt
from matplotlib.image import imread
import os
import scipy.io
plt.rcParams['figure.figsize'] = [16,6]
plt.rcParams.update({'font.size': 18})
```

[2]:
```
A = np.matrix([[2,  0, 1],
               [3, -1, 1],
               [-2,  4, 1],
               [1,  1, 1]]);A
```

[2]:
```
matrix([[ 2,  0,  1],
        [ 3, -1,  1],
        [-2,  4,  1],
        [ 1,  1,  1]])
```

[3]:
```
d = A.T@A
```

[4]:
```
Λ,X = np.linalg.eig(d)
```

[5]:
```
D = np.diag(np.round(Λ));D
```

[5]:
```
array([[28.,  0.,  0.],
       [ 0., 12.,  0.],
       [ 0.,  0., -0.]])
```

[6]:
```
np.round(X,3)
```

[6]:
```
array([[-0.707,  0.577, -0.408],
       [ 0.707,  0.577, -0.408],
       [ 0.   ,  0.577,  0.816]])
```

[7]:
```
U,S,V = np.linalg.svd(A,full_matrices=False)
S = np.diag(S)
S
```

[7]:
```
array([[5.29150262e+00, 0.00000000e+00, 0.00000000e+00],
       [0.00000000e+00, 3.46410162e+00, 0.00000000e+00],
       [0.00000000e+00, 0.00000000e+00, 5.72212401e-16]])
```

[8]:
```
U
```

[8]:
```
matrix([[-2.67261242e-01,  5.00000000e-01, -8.04910061e-01],
        [-5.34522484e-01,  5.00000000e-01,  5.69500183e-01],
        [ 8.01783726e-01,  5.00000000e-01,  1.11363435e-01],
        [ 2.49800181e-16,  5.00000000e-01,  1.24046443e-01]])
```

[9]:
```
V
```

[9]:
```
matrix([[-7.07106781e-01,  7.07106781e-01,  2.10475052e-16],
        [ 5.77350269e-01,  5.77350269e-01,  5.77350269e-01],
        [ 4.08248290e-01,  4.08248290e-01, -8.16496581e-01]])
```

[10]:
```
np.round(U@S@V.T,2)
```

[10]:
```
array([[ 2.22,  0.18,  0.13],
       [ 3.22, -0.63, -0.45],
       [-1.78,  3.45,  2.44],
       [ 1.22,  1.  ,  0.71]])
```

[11]:
```
x = np.array([9,10,2,7,1,3,4,6,8,5]);x
x = np.asmatrix(x)
U,S,V = np.linalg.svd(x)
```

[12]:
```
B = np.matrix([[ 6, 6],
               [-1, 1]]);B
```

[12]:
```
matrix([[ 6,  6],
        [-1,  1]])
```

[13]:
```
U,S,V = np.linalg.svd(B,full_matrices=False)
S = np.diag(S)
np.round(U@S@V.T,2)
```

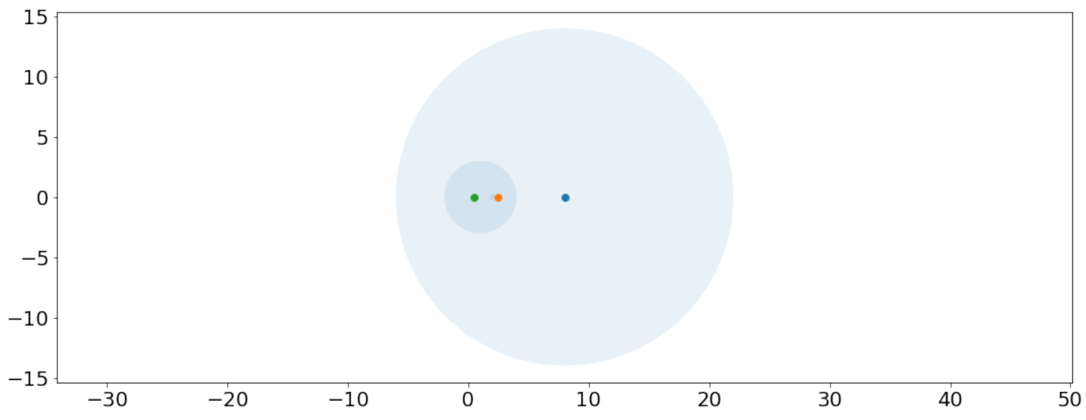
```
[13]: array([[ 6.,  6.],
             [-1.,  1.]])
```

```
[14]: np.linalg.eig(B)
```

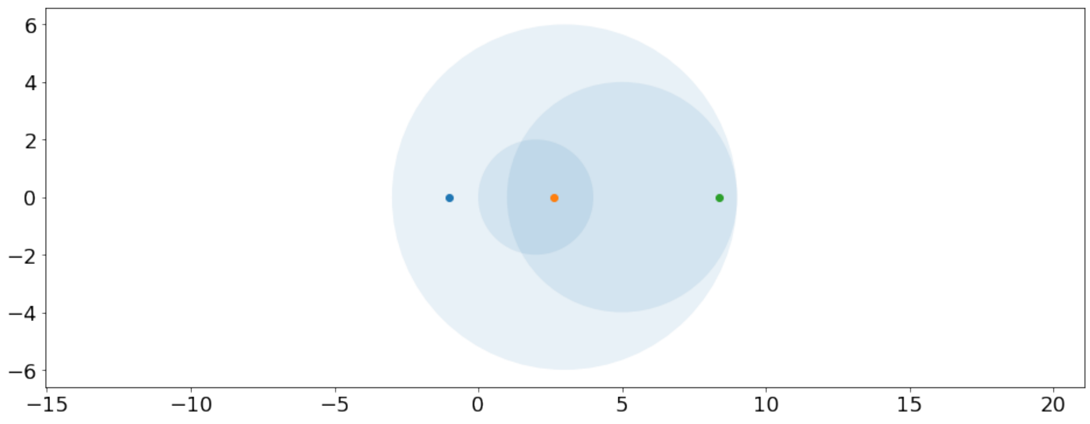
```
[14]: (array([4., 3.]),
       matrix([[ 0.9486833 , -0.89442719],
               [-0.31622777,  0.4472136 ]]))
```

```
[15]: def Gerschgorin(A):

          n = len(A)
          eval, evec = np.linalg.eig(A)

          patches = []

          # draw discs

          for i in range(n):
              xi = np.real(A[i,i])
              yi = np.imag(A[i,i])
              ri = np.sum(np.abs(A[i,:])) - np.abs(A[i,i])

              circle = Circle((xi, yi), ri)
              patches.append(circle)

          fig, ax = plt.subplots()

          p = PatchCollection(patches, cmap=matplotlib.cm.jet, alpha=0.1)
          ax.add_collection(p)
          plt.axis('equal')

          for xi, yi in zip(np.real(eval), np.imag(eval)):
              plt.plot(xi, yi,'o')

          plt.show()
```

```
[16]: A = np.matrix([[ 8, 7,7],
                     [0,2,1/4],
                    [0,3,1]]);A
      Gerschgorin(A)
```



```
[17]: B = np.matrix([[2,2,0],
                     [2,3,4],
                    [0,4,5]]);B
      Gerschgorin(B)
```



```
[18]: np.random.seed(123)
      n   = 6
      mu  = 0
      sigma = 1
      A   = np.random.normal(mu, sigma, (n,n))
      A   = np.asmatrix(A)
      C = (A+A.conj().T)/2
```

```
[19]: def check_symmetric(a, rtol=1e-05, atol=1e-08):
          return np.allclose(a, a.T, rtol=rtol, atol=atol)
      check_symmetric(C)
```

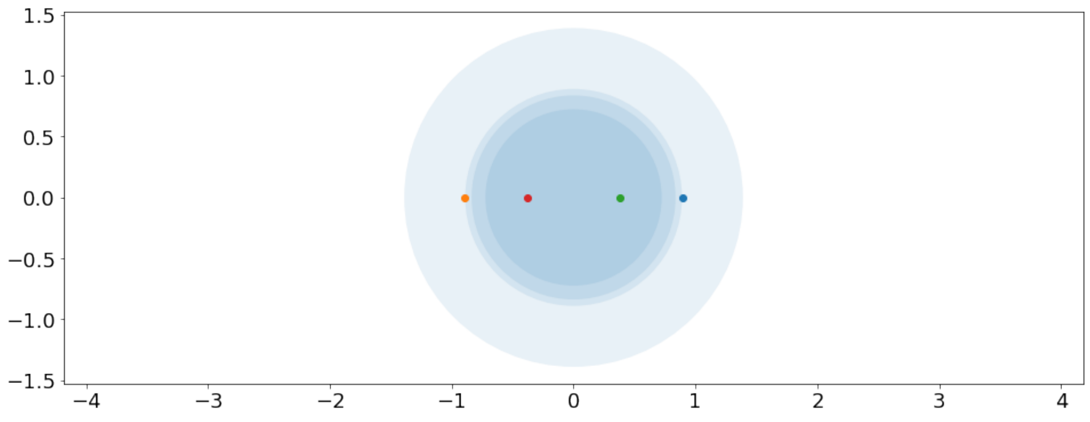
```
[19]: True
```

```
[20]: def randomhermitian(n):
          np.random.seed(123)
          mu = 0
          sigma = n**(-1/2)
          A  = np.random.normal(mu, sigma, (n,n))* 1j
          A  = np.asmatrix(A);A
          C = (A+A.conj().T)/2
          return(C)
```

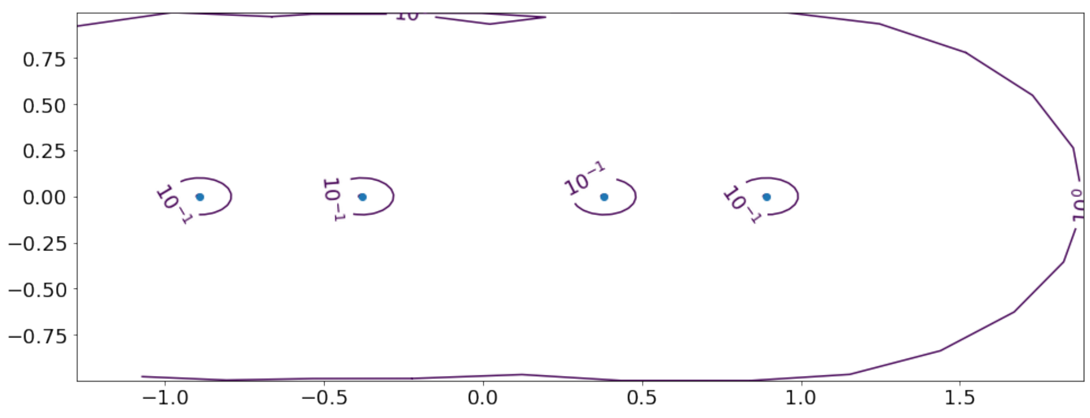
```
[21]: A = randomhermitian(4);A
```

```
[21]: matrix([[ 0.+0.j        ,  0.+0.39398642j,  0.-0.24573944j,
                0.-0.74942109j],
              [ 0.-0.39398642j,  0.+0.j        , -0.-0.38998471j,
                0.+0.05249734j],
              [ 0.+0.24573944j,  0.+0.38998471j,  0.+0.j        ,
                0.+0.08731825j],
              [ 0.+0.74942109j, -0.-0.05249734j, -0.-0.08731825j,
                0.+0.j        ]])
```

```
[22]: Gerschgorin(A)
```



```
[23]: pseudo = NonnormalAuto(A, 1e-5, 1)
      pseudo.plot([10**k for k in range(-4, 3)], spectrum=eigvals(A))
      pyplot.show()
```



```
[24]: U,S,V = np.linalg.svd(B,full_matrices=False)
```

```
[25]: df = pd.read_csv("/Users/nikosmatsavelas/Desktop/Thesis NTUA/econ - Sheet1.csv")
      df
```

```
[25]:        Country  Increase  Life  imr  tfr       gdp
      0      Albania       1.2  69.2   30  2.9    659.91
      1    Argentina       1.2  68.6   24  2.8   4343.04
      2    Australia       1.1  74.7    7  1.9  17529.98
      3      Austria       1.0  73.0    7  1.5  20561.88
      4        Benin       3.2  45.9   86  7.1    398.21
      5      Bolivia       2.4  57.7   75  4.8    812.19
      6       Brazil       1.5  64.0   58  2.9   3219.22
      7     Cambodia       2.8  50.1  116  5.3     97.39
      8        China       1.1  66.7   44  2.0    341.31
      9     Colombia       1.7  66.4   37  2.7   1246.87
      10     Croatia      -1.5  67.1    9  1.7   5400.66
      11  ElSalvador       2.2  63.9   46  4.0    988.58
      12      France       0.4  73.0    7  1.7  21076.77
      13      Greece       0.6  75.0   10  1.4   6501.23
      14   Guatemala       2.9  62.4   48  5.4    831.81
      15        Iran       2.3  67.0   36  5.0   9129.34
      16       Italy      -0.2  74.2    8  1.3  19204.92
      17      Malawi       3.3  45.0  143  7.2    229.01
      18 Netherlands       0.7  74.4    7  1.6  18961.90
      19    Pakistan       3.1  60.6   91  6.2    385.59
      20     PapuaNG       1.9  55.2   68  5.1    839.03
      21        Peru       1.7  64.1   64  3.4   1674.15
      22     Romania      -0.5  66.6   23  1.5   1647.97
      23          US       1.1  72.5    9  2.1  21965.08
      24    Zimbabwe       4.4  52.4   67  5.0    686.75
```

```
[26]: df.cov()
```

```
[26]:             Increase          Life            imr          tfr           gdp
      Increase    1.758900     -8.653950      37.113333     2.143083 -5.115258e+03
      Life       -8.653950     79.685267    -305.898333   -14.964667  4.965945e+04
      imr        37.113333   -305.898333    1381.333333    61.537500 -2.037920e+05
      tfr         2.143083    -14.964667      61.537500     3.571667 -9.200412e+03
      gdp     -5115.257723  49659.453270 -203791.976750 -9200.412267  6.553446e+07
```

```
[27]: d = df.corr();d
```

```
[27]:         Increase      Life       imr       tfr       gdp
      Increase  1.000000 -0.730979  0.752940  0.855033 -0.476444
      Life     -0.730979  1.000000 -0.922017 -0.887039  0.687192
      imr       0.752940 -0.922017  1.000000  0.876103 -0.677334
      tfr       0.855033 -0.887039  0.876103  1.000000 -0.601363
      gdp      -0.476444  0.687192 -0.677334 -0.601363  1.000000
```

```
[28]: Λ,X = np.linalg.eig(d)
      np.sqrt(np.diag(Λ))
```

```
[28]: array([[2.00348196, 0.        , 0.        , 0.        , 0.        ],
             [0.        , 0.75419734, 0.        , 0.        , 0.        ],
             [0.        , 0.        , 0.50256601, 0.        , 0.        ],
             [0.        , 0.        , 0.        , 0.30963564, 0.        ],
             [0.        , 0.        , 0.        , 0.        , 0.26229679]])
```

```
[29]: X
```

```
[29]: array([[-0.42768621,  0.51149975,  0.64548884, -0.28397399,  0.2411544 ],
             [ 0.47437702,  0.04578062,  0.47607336, -0.16217624, -0.72105728],
             [-0.47450184, -0.01195852, -0.41723864, -0.63391726, -0.4458316 ],
             [-0.47409829,  0.24919979, -0.02745948,  0.69995012, -0.47164189],
             [ 0.377001  ,  0.82099245, -0.42644603, -0.03572625,  0.02662827]])
```

```
[30]: (Λ[0]+Λ[1])/np.sum(Λ)
```

```
[30]: 0.916550714071298
```

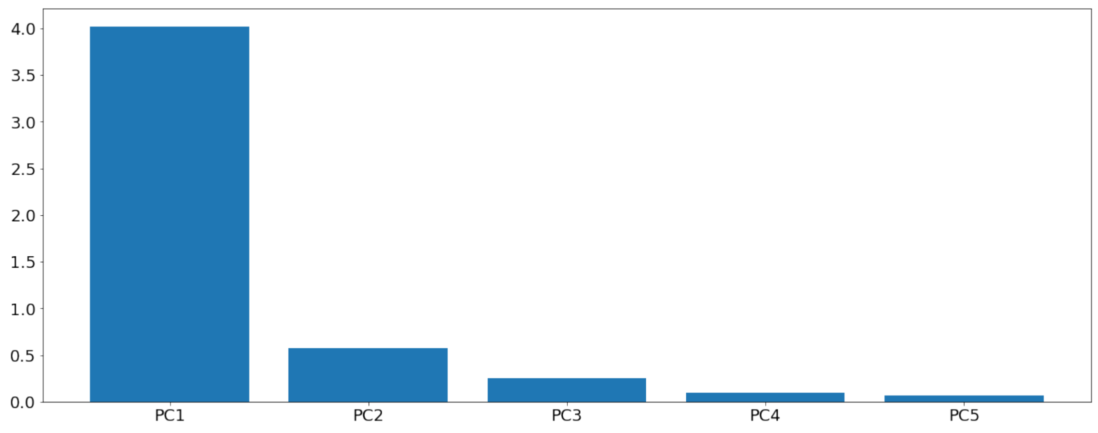
```
[31]: df1 = pd.DataFrame({ 'PC1': [round(np.sqrt(Λ[0]),4), (Λ[0])/np.sum(Λ),(Λ[0])/np.
      ↪sum(Λ) ],
                          'PC2': [round(np.sqrt(Λ[1]),4), (Λ[1])/np.
      ↪sum(Λ),(Λ[0]+Λ[1])/np.sum(Λ)],
                          'PC3': [round(np.sqrt(Λ[2]),4), (Λ[2])/np.
      ↪sum(Λ),(Λ[0]+Λ[1]+Λ[2])/np.sum(Λ)],
                          'PC4': [round(np.sqrt(Λ[3]),4), (Λ[3])/np.
      ↪sum(Λ),(Λ[0]+Λ[1]+Λ[2]+Λ[3])/np.sum(Λ)],
                          'PC5': [round(np.sqrt(Λ[4]),4), (Λ[4])/np.
      ↪sum(Λ),(Λ[0]+Λ[1]+Λ[2]+Λ[3]+Λ[4])/np.sum(Λ) ]},
                             index=[ "Standard deviation", "Proportion of␣
      ↪Variance","Cumulative Proportion"])
      df1
```

[31]:

```
                             PC1       PC2       PC3       PC4      PC5
Standard deviation      2.003500  0.754200  0.502600  0.309600  0.26230
Proportion of Variance  0.802788  0.113763  0.050515  0.019175  0.01376
Cumulative Proportion   0.802788  0.916551  0.967065  0.986240  1.00000
```

[32]:

```
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
pcs = ['PC1', 'PC2', 'PC3', 'PC4', 'PC5']
eigen = [Λ[0],Λ[1],Λ[2],Λ[3],Λ[4]]
ax.bar(pcs,eigen)
plt.show()
```



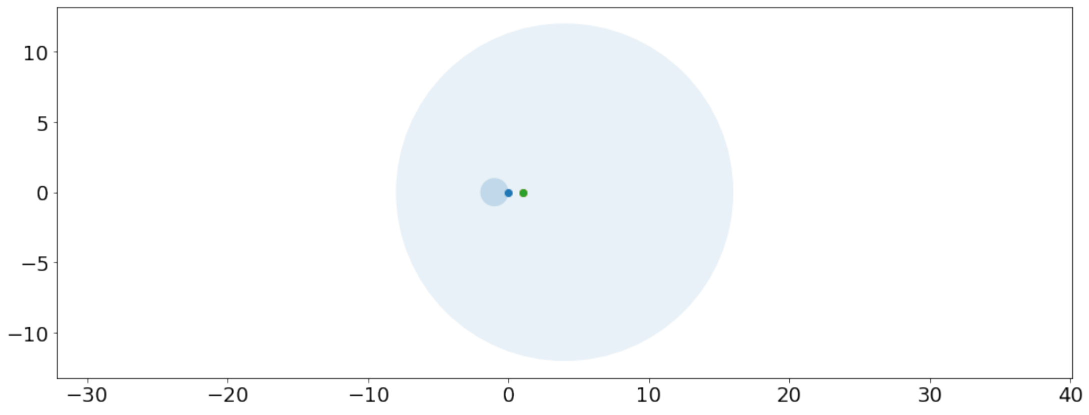
[33]:

```
from scipy.linalg import svdvals
def inv_resolvent_norm(A, z, method='svd'):

    if method == 'svd':
        return np.min(svdvals(A - z*np.eye(*A.shape)))
    elif method == 'lanczos':
        m, n = A.shape
        if m > n:
            raise ValueError('m > n is not allowed')
        AH = A.T.conj()

        def matvec(x):

            x1 = x[:m]
            x2 = x[m:]
            ret1 = AH.dot(x2) - np.conj(z)*x2
            ret2 = np.array(A.dot(x1), dtype=np.complex)
            ret2[:n] -= z*x1
            return np.c_[ret1, ret2]
        AH_A = LinearOperator(matvec=matvec, dtype=np.complex,
                              shape=(m+n, m+n))

        evals = eigsh(AH_A, k=2, tol=1e-6, which='SM', maxiter=m+n+1,
                      ncv=2*(m+n),
                      return_eigenvectors=False)

        return np.min(np.abs(evals))
```

[34]:
```
A = np.matrix([[-1,1,0],
               [0,-1,1],
               [4,-8,4]])
A
```

[34]:
```
matrix([[-1,  1,  0],
        [ 0, -1,  1],
        [ 4, -8,  4]])
```

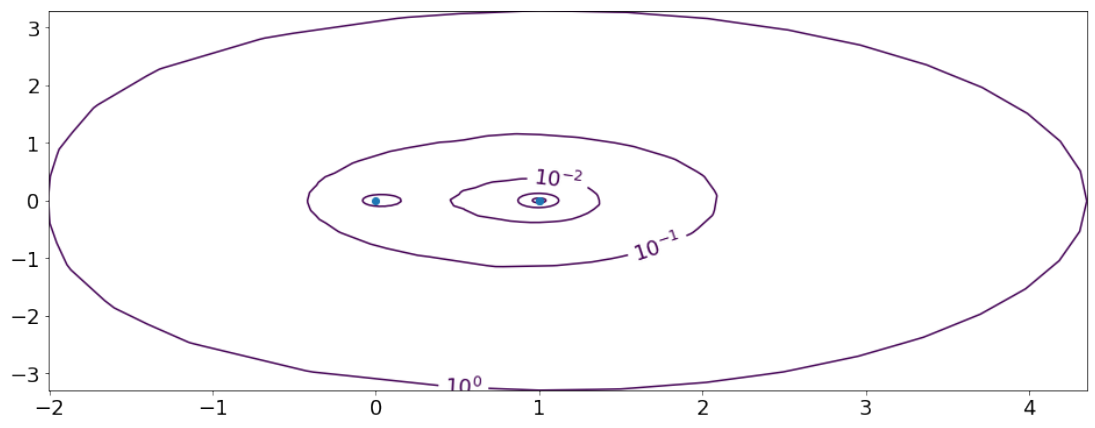
[35]:
```
Gerschgorin(A)
```



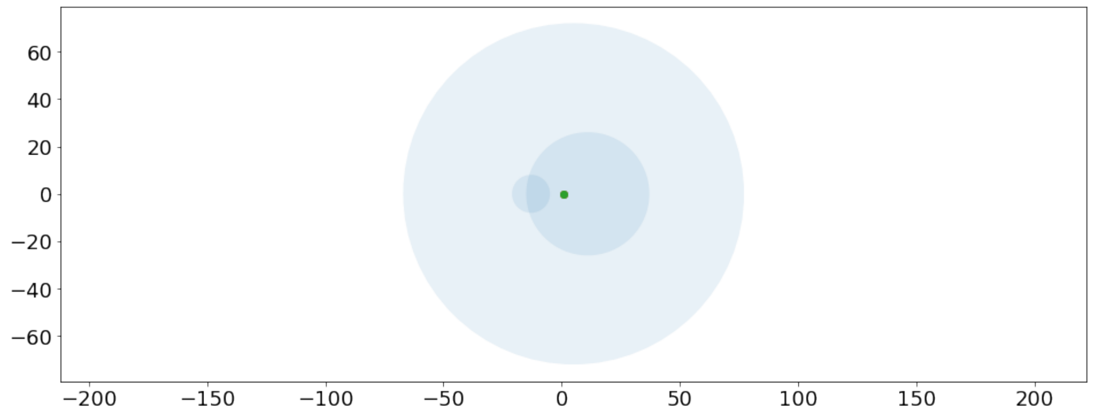
[36]:
```
inv_resolvent_norm(A, z=1j, method='svd')
```

[36]:
```
0.14073330835911999
```

[37]:
```
pseudo = NonnormalAuto(A, 1e-5, 1)
pseudo.plot([10**k for k in range(-4, 3)], spectrum=eigvals(A))
pyplot.show()
```



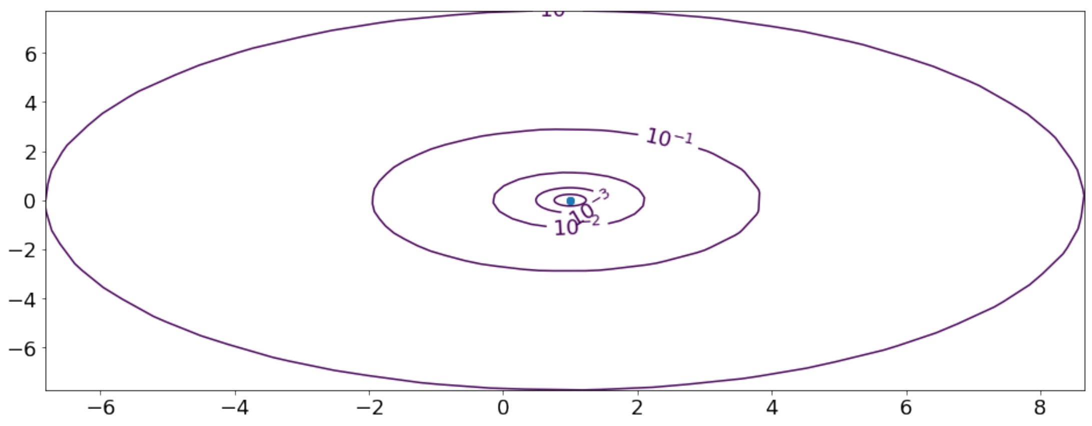
```
[38]: A = np.matrix([[-13,-2,6],
                     [52,5,-20],
                     [-22,-4,11]])
      A
      Gerschgorin(A)
```



```
[39]: inv_resolvent_norm(A, z=1j, method='svd')
```

```
[39]: 0.014880045695576412
```

```
[40]: pseudo = NonnormalAuto(A, 1e-5, 1)
      pseudo.plot([10**k for k in range(-4, 3)], spectrum=eigvals(A))
      pyplot.show()
```



```
[41]: n   = 2000
      mu = 0
      sigma = 1
      A   = np.random.normal(mu, sigma, (n,n))
```

```
[ ]: pseudo = Normal(A)
     pseudo.plot([10**k for k in range(-1, 1)], spectrum=eigvals(A))
     pyplot.show()
```

```
[ ]: def rsvd(A, Omega):
         Y = A @ Omega
         Q, _ = np.linalg.qr(Y)
         B = Q.T @ A
         u_tilde, s, v = np.linalg.svd(B, full_matrices = 0)
         u = Q @ u_tilde
         return u, s, v
```

```
[ ]: np.random.seed(1000)
     A = np.array([[1, 3, 2],
                   [5, 3, 1],
                   [3, 4, 5]])
     k = 2
     p = 1
     l = k+p
     Omega = np.random.randn(A.shape[1], l)
     Omega
     pseudo = NonnormalAuto(A, 1e-5, 1)
     pseudo.plot([10**k for k in range(-4, 3)], spectrum=eigvals(A))
     pyplot.show()
```

```
[ ]: Gerschgorin(A)
[ ]: u, s, v = rsvd(A, Omega)
     print('Left singular vectors:')
     print(u)
     print()
     print('Singular values:')
     print(s)
     print()
     print('Right singular vectors:')
     print(v)
     print()
```

```
[ ]: def power_iteration(A,k,p,q = 3):
         m = A.shape[0]
         n = A.shape[1]
         l = k + p
         Y = A @ Omega
         for q in range(q):
             Y = A @ (A.T @ Y)
         Q, _ = np.linalg.qr(Y)
         return Q
```

```
[ ]: def rsvd(A,k,p,q):
         Q = power_iteration(A,k,p,q)
         B = Q.T @ A
         u_tilde, s, v = np.linalg.svd(B, full_matrices = 0)
         u = Q @ u_tilde
         return u, s, v
```

```
[ ]: np.random.seed(1000)
     a = np.array([[1, 3, 2],
                   [5, 3, 1],
                   [3, 4, 5]])

     np.linalg.cond(a)
```

```
[ ]: power_iteration(A,k=10,p=3,q=3)
```

```
[ ]: rsvd(A,k=30,p=10,q=5)
```

```
[ ]: U,S,V = np.linalg.svd(a,full_matrices=False)
     np.max(S)/np.min(S)
```

```
[ ]: u, s, v = rsvd(A, k,p,q=1)
     print('Left singular vectors:')
     print(u)
     print()
     print('Singular values:')
print(s)
print()
print('Right singular vectors:')
print(v)
print()
```

[ ]:
```
power_iteration(A,k=10,p=10,q = 3)
```

[ ]:
```
def singlepass(A,k,p):
    np.random.seed(123)
    m = A.shape[0]
    n = A.shape[1]
    l = k+p
    Ω = np.random.normal(0,1,size=(n,k)) # n x k
    Ψ = np.random.normal(0,1,size=(l,m)) # l x m
    Y = A@Ω         # m x n  * n x k =     m x k
    W = Ψ@A         # l x m  * m x n =     l x n
    Q, _ = np.linalg.qr(Y,mode='reduced')# m x k
    X = np.linalg.pinv(Ψ@Q)@W            # k x n
    Ahat = Q@X     # m x k  * k x n =      m x n
    error = np.linalg.norm(A-Q@Q.T@A,"fro")
    return(Ahat,error)
```

[ ]:
```
A = imread(os.path.join("/Users/nikosmatsavelas/Desktop/Thesis NTUA/cat.jpg"))
print(A.shape)
A[:, :, 0].shape
img_array = A / 255
print(img_array.shape)
img_gray = img_array @ [0.2126, 0.7152, 0.0722]
plt.imshow(img_gray, cmap="gray")
img_gray.shape
```

[ ]:
```
AH = singlepass(A=img_gray,k=100,p=10)
plt.imshow(AH[0], cmap="gray")
plt.show()
```

[ ]:
```
AH1 = singlepass(A=img_gray,k=500,p=10)
AH2 = singlepass(A=img_gray,k=400,p=10)
AH3 = singlepass(A=img_gray,k=300,p=10)
AH4 = singlepass(A=img_gray,k=200,p=10)
AH5 = singlepass(A=img_gray,k=100,p=10)
```

[ ]:
```
## Plot
fig, axs = plt.subplots(1,5)

plt.set_cmap('gray')
```

```
axs[0].imshow(AH1[0])
axs[0].axis('off')
axs[1].imshow(AH2[0])
axs[1].axis('off')
axs[2].imshow(AH3[0])
axs[2].axis('off')
axs[3].imshow(AH4[0])
axs[3].axis('off')
axs[4].imshow(AH5[0])
axs[4].axis('off')
plt.show()
```

```
[ ]: import pandas as pd
     df = pd.DataFrame({'error':[AH1[1],
                        AH2[1],
                        AH3[1],
                        AH4[1],
                        AH5[1]]})
     df.index = ['Error k =500 , p =10 ',
                 'Error k =400 , p =10',
                 'Error k =300 , p =10',
                 'Error k =200 , p =10',
                 'Error k =100 , p =10']
     np.round(df,5)
```

```
[ ]: def singlepasshermitian(A,k,p):
         np.random.seed(5)
         n = A.shape[1]
         l = k+p
         G = np.random.normal(0,1,size=(n,l)) # n x l
         Y = A@G          # n x n  * n x l =     n x l
         Q, _ = np.linalg.qr(Y,mode='reduced')  # n x l
         C = Q.T@Y
         L , Uhat = np.linalg.eig(C)
         U = Q@Uhat
         Ahat = U@np.diag(L)@U.T
         error = np.linalg.norm(A - Ahat,ord=2)/np.linalg.norm(A,ord=2)
         return(Ahat,error)
```

```
[ ]: def is_pos_def(x):
         return np.all(np.linalg.eigvals(x) > 0)
```

```
[ ]: from sklearn.datasets import make_sparse_spd_matrix
     F1 = make_sparse_spd_matrix(1000)
     is_pos_def(F1)
```

```
[ ]: U,error =singlepasshermitian(A=F1,k=300,p=10)
     error
```

```
[ ]: def singlepassGENERAL(A,k,p):
         np.random.seed(4)
         m = A.shape[0]
         n = A.shape[1]
         l = k + p
         GC = np.random.normal(0,1,size=(n,l))                 # n x l
         GR = np.random.normal(0,1,size=(m,l))                 # m x l
         YC = A@GC ;YC.shape           # m x n  * n x l =      m x l
         YR = A.T@GR ;YR.shape         # n x m  * m x l =      n x l
         QC, _ = np.linalg.qr(YC,mode='reduced');QC.shape   # m x l
         QR, _ = np.linalg.qr(YR,mode='reduced');QR.shape   # n x l
         C = QC.T@A@QR ;C.shape                             # l x l
         Uhat,D,Vhat = np.linalg.svd(C,full_matrices = False)
         U = QC@Uhat                                        # m x l
         V = QR@Vhat                                        # n x l
         X = U@np.diag(D)@V.T  # m x l* l x l  * l x n =      m x n
         error = np.linalg.norm(A-X,ord=2)/np.linalg.norm(A,ord=2)
         return(X,error)
```

```
[ ]: A = imread(os.path.join("/Users/nikosmatsavelas/Desktop/Thesis NTUA/gray.jpg"))
     A.shape
```

```
[ ]: plt.imshow(A, cmap="gray")
```

```
[ ]: SP1 = singlepassGENERAL(A=A,k=500,p=50)
     SP2 = singlepassGENERAL(A=A,k=400,p=50)
     SP3 = singlepassGENERAL(A=A,k=300,p=50)
     SP4 = singlepassGENERAL(A=A,k=200,p=50)
     SP5 = singlepassGENERAL(A=A,k=100,p=50)
```

```
[ ]: import pandas as pd
     df = pd.DataFrame({'error':[SP1[1],
                        SP2[1],
                        SP3[1],
                        SP4[1],
                        SP5[1]]})
     df.index = ['Realtive Error k =500 , p =50 ',
                 'Realtive Error k =400 , p =50',
                 'Realtive Error k =300 , p =50',
                 'Realtive Error k =200 , p =50',
                 'Realtive Error k =100 , p =50']
     np.round(df,5)
```

[ ]:
```
## Plot
fig, axs = plt.subplots(1,5)

plt.set_cmap('gray')
axs[0].imshow(SP1[0])
axs[0].axis('off')
axs[1].imshow(SP2[0])
axs[1].axis('off')
axs[2].imshow(SP3[0])
axs[2].axis('off')
axs[3].imshow(SP4[0])
axs[3].axis('off')
axs[4].imshow(SP5[0])
axs[4].axis('off')
plt.show()
```

[ ]:
```
def Accurancy_enhanced(A,k,p,power_iter = 3):

    np.random.seed(3)
    m = A.shape[0] # rows
    n = A.shape[1] # columns
    G = np.random.normal(0,1, size=(n,k+p))
    Y = A@G
    for q in range(power_iter):
        Z = A.T@Y
        Y = A@Z
    Q, _ = np.linalg.qr(Y)
    B = Q.T@A
    u_tilde, s, v = np.linalg.svd(B, full_matrices = False)
    U = Q @ u_tilde
    return(Q,U)
```

[ ]:
```
def Accurancy_enhanced_ortho(A,k,p,power_iter = 3):
    np.random.seed(2)
    m = A.shape[0] # rows
    n = A.shape[1] # columns
    G = np.random.normal(0,1, size=(n,k+p))
    Q, _ = np.linalg.qr(A@G)
    for q in range(power_iter):
        W, _ = np.linalg.qr(A.T@Q)
        Q, _ = np.linalg.qr(A@W)
    B = Q.T@A
    u_tilde, s, v = np.linalg.svd(B, full_matrices = False)
    U = Q @ u_tilde
    return(U,Q)
```

```
[ ]: U, S, VT = np.linalg.svd(A,full_matrices=False) # Deterministic SVD
     r = 100 # Target rank
     q = 1   # Power iterations
     p = 5   # Oversampling parameter
```

```
[ ]: print(A.shape[0])
     print(A.shape[1])
```

```
[ ]:
```

```
[ ]: %%time
     Accurancy_enhanced(A=A,k=2,p=10,power_iter = 20)
```

```
[ ]: %%time
     Accurancy_enhanced_ortho(A=A,k=2,p=10,power_iter = 20)
```

```
[ ]: def Nystrom(A,k,p):
         np.random.seed(15)
         m = A.shape[0] # rows
         n = A.shape[1] # columns
         G = np.random.normal(0,1, size=(n,k+p))
         Y = A@G
         Q, _ = np.linalg.qr(Y)
         B1 = A@Q
         B2 = Q.T@B1
         C = np.linalg.cholesky(B2)
         F = C.T@C
         u_tilde, Σ, v = np.linalg.svd(F, full_matrices = False)
         Λ = Σ**2
         return(Σ)
```

```
[ ]: from scipy import sparse
     from scipy.sparse import diags
     N = 300
     main_diag = 2*np.ones((N+1, 1)).ravel()
     off_diag = -1*np.ones((N, 1)).ravel()

     a = main_diag.shape[0]

     diagonals = [main_diag, off_diag, off_diag]

     A = sparse.diags(diagonals, [0,-1,1], shape=(a,a)).toarray();A
```

```
[ ]: Ny = Nystrom(A=A,k=150,p=10)
     Ny.shape[0]
```

```
[ ]: from scipy.linalg import svdvals
     Ei = svdvals(A)[0:Ny.shape[0]]
     plt.rcParams['figure.figsize'] = [15, 8]
     plt.rcParams.update({'font.size': 18})
     plt.plot(Ei)
     plt.plot(Ny)
     plt.gca().legend(('real eigenvalues','Nystrom eigenvalues'))
     plt.show()
```

```
[ ]: A = imread(os.path.join("/Users/nikosmatsavelas/Desktop/Thesis NTUA/gray.jpg"))
     print(A.shape)
     U, S, VT = np.linalg.svd(A,full_matrices=False) # Deterministic SVD
```

```
[ ]: k = 100 # Target rank
     q = 2   # Power iterations
     p = 5   # Oversampling parameter
     def rSVD(A,k,q,p):
         # Step 1: Sample column space of X with P matrix
         m = A.shape[0]
         n = A.shape[1]
         l = k+p
         P = np.random.normal(0,1,size=(n,l))
         Z = A @ P
         for i in range(q):
             Z = A @ (A.T @ Z)

         Q, R = np.linalg.qr(Z)

         # Step 2: Compute SVD on projected Y = Q.T @ X
         Y = Q.T @ A
         UY, S, VT = np.linalg.svd(Y,full_matrices=0)
         U = Q @ UY

         return U, S, VT
```

```
[ ]: U, S, VT = np.linalg.svd(A,full_matrices=False) # Deterministic SVD

     r = 100 # Target rank
     q = 3   # Power iterations
     p = 5   # Oversampling parameter

     rU, rS, rVT = rSVD(A,k = r,q=p,p=p)
```

```
[ ]: %%time
     U, S, VT = np.linalg.svd(A,full_matrices=False)
     print(U.shape)
     print(S.shape)
print(VT.shape)
```

```
[ ]: %%time
     rU, rS, rVT = rSVD(A,r,q,p)
```

```
[ ]: # Reconstruction
```

```
[ ]: %%time
     XSVD = U[:,:(r+1)] @ np.diag(S[:(r+1)]) @ VT[:(r+1),:] # SVD approximation
     errSVD1 = np.linalg.norm(A-XSVD,ord=2) / np.linalg.norm(A,ord=2)
     print(errSVD1)
```

```
[ ]: %%time
     XrSVD = rU[:,:(r+1)] @ np.diag(rS[:(r+1)]) @ rVT[:(r+1),:] # SVD approximation
     errSVD2 = np.linalg.norm(A-XrSVD,ord=2) / np.linalg.norm(A,ord=2)
     print(errSVD2)
```

```
[ ]: ## Plot
     fig, axs = plt.subplots(1,3)

     plt.set_cmap('gray')
     axs[0].imshow(A)
     axs[0].axis('off')
     axs[1].imshow(XSVD)
     axs[1].axis('off')
     axs[2].imshow(XrSVD)
     axs[2].axis('off')

     plt.show()
```

```
[ ]: ## Illustrate power iterations
     X = np.random.randn(1000,100)
     U, S, VT = np.linalg.svd(X,full_matrices=0)
     S = np.arange(1,0,-0.01)
     X = U @ np.diag(S) @ VT

     color_list = np.array([[0,0,2/3],  # Define color map
                      [0,0,1],
                      [0,1/3,1],
                      [0,2/3,1],
                      [0,1,1],
                      [1/3,1,2/3],
                      [2/3,1,1/3],
                      [1,1,0],
                      [1,2/3,0],
                      [1,1/3,0],
                      [1,0,0],
                      [2/3,0,0]])

    plt.plot(S,'o-',color='k',LineWidth=2,label='SVD')

     Y = X
     for q in range(1,6):
         Y = X.T @ Y
         Y = X @ Y
         Uq, Sq, VTq = np.linalg.svd(Y,full_matrices=0)
         plt.plot(Sq,'-o',color=tuple(color_list[2*q+1]),LineWidth=2,label='rSVD, q␣
      ↪= '+str(q))

     plt.legend()
     plt.show()
```

1 Interpolative Decomposition (Optimal)

[ ]:
```
from scipy import linalg
def optim_id(A, k):
    _, R, P = linalg.qr(A,pivoting=True,
                              mode='economic',
                              check_finite=False)
    R_k = R[:k,:k]
    cols = P[:k]
    C = A[:,cols]
    Z = linalg.solve(R_k.T @ R_k,C.T @ A,
                           overwrite_a=True,
                           overwrite_b=True,
                           assume_a='pos')
    approx = C @ Z
    return(approx , cols , Z)
```

[ ]:
```
A = imread(os.path.join("/Users/nikosmatsavelas/Desktop/Thesis NTUA/Figures/
 ↪IMG_0004.jpg"))
A[:, :, 0].shape
img_array = A / 255
print(img_array.shape)
img_gray = img_array @ [0.2126, 0.7152, 0.0722]
plt.imshow(img_gray, cmap="gray")
img_gray.shape
A = img_gray
```

[ ]:
```
k = 50
h1 = optim_id(A, k)
```

[ ]:
```
plt.imshow(h1[0], cmap="gray")
```

2 Interpolative Decomposition (Randomized) - RID

[ ]:
```
rng = np.random.default_rng()
def optim_rid(A, k):
    oversampling = int(0.2 * k)
    p = k + oversampling
    idx = rng.choice(A.shape[1],
                     replace=False,
                     size=p)
    AS = A[:,idx]
    _, R, P = linalg.qr(AS, pivoting=True,
                                mode='economic',
                                check_finite=False)
    R_k = R[:k,:k]
    _cols = P[:k]
    cols = idx[_cols]
    C = AS[:,_cols]
    Z = linalg.solve(R_k.T @ R_k,
    C.T @ A, overwrite_a=True,
                             overwrite_b=True,
                             assume_a='sym')
    approx = C @ Z
    return(approx , cols , Z)
```

[ ]:
```
k = 50
h2 = optim_rid(A, k)
```

[ ]:
```
plt.imshow(h2[0], cmap="gray")
```

[ ]:
```
AH1 = optim_rid(A, k=500)
AH2 = optim_rid(A, k=100)
AH3 = optim_rid(A, k=50)
```

[ ]:
```
## Plot
fig, axs = plt.subplots(1,3)

plt.set_cmap('gray')
axs[0].imshow(AH1[0])
axs[0].axis('off')
axs[1].imshow(AH2[0])
axs[1].axis('off')
axs[2].imshow(AH3[0])
axs[2].axis('off')


plt.show()
```

3 Randomized PCA

[ ]:
```
def rpca(A,k,p):
    l = k+p
    m = A.shape[0]
    n = A.shape[1]
    Omega = np.random.normal(0,1,size=(n,l))
    Y = A@Omega
    Q,_ = np.linalg.qr(Y)
    B = Q.T@A
    Uhat,Σ,W = np.linalg.svd(B, full_matrices = 0)
    U = Q@Uhat
    np.diag(Σ).shape
    Λ = Σ**2/(m-1)
    Z =U@Σ
    return(W,Λ,Z)
```

3.1 Randomized PCA implementation (eigenfaces)

[ ]:
```
from sklearn.decomposition import PCA as RandomizedPCA
from sklearn.datasets import fetch_lfw_people
faces = fetch_lfw_people(min_faces_per_person=60)
print(faces.target_names)
print(faces.images.shape)
pca = RandomizedPCA(150,svd_solver='randomized')
pca.fit(faces.data)
```

[ ]:
```
fig, axes = plt.subplots(3, 8, figsize=(9, 4),
                         subplot_kw={'xticks':[], 'yticks':[]},
                         gridspec_kw=dict(hspace=0.1, wspace=0.1))
for i, ax in enumerate(axes.flat):
    ax.imshow(pca.components_[i].reshape(62, 47), cmap='bone')
```

[ ]:
```
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance');
```

[ ]:
```
# Compute the components and projected faces
pca = RandomizedPCA(150).fit(faces.data)
components = pca.transform(faces.data)
projected = pca.inverse_transform(components)
```

[ ]:
```
# Plot the results
fig, ax = plt.subplots(2, 10, figsize=(10, 2.5),
                       subplot_kw={'xticks':[], 'yticks':[]},
                       gridspec_kw=dict(hspace=0.1, wspace=0.1))
for i in range(10):
    ax[0, i].imshow(faces.data[i].reshape(62, 47), cmap='binary_r')
    ax[1, i].imshow(projected[i].reshape(62, 47), cmap='binary_r')

ax[0, 0].set_ylabel('full-dim\ninput')
ax[1, 0].set_ylabel('150-dim\nreconstruction');
```

3.2 Robust PCA

[ ]:
```
from scipy.optimize import minimize
```

[ ]:
```
mat = scipy.io.loadmat(os.path.join('/Users/nikosmatsavelas/Desktop/MS.c␣
 ↪Applied Mathematics/NTUA 2nd Semester/Matrix Analysis/DATA/allFaces.mat'))
faces = mat['faces']
nfaces = mat['nfaces'].reshape(-1)
```

[ ]:
```
## Function Definitions

def shrink(X,tau):
    Y = np.abs(X)-tau
    return np.sign(X) * np.maximum(Y,np.zeros_like(Y))
def SVT(X,tau):
    U,S,VT = np.linalg.svd(X,full_matrices=0)
    out = U @ np.diag(shrink(S,tau)) @ VT
    return out
def RPCA(X):
    m,n = X.shape
    mu = m*n/(4*np.sum(np.abs(X.reshape(-1))))
    lambd = 1/np.sqrt(np.maximum(m,n))
    thresh = 10**(-7) * np.linalg.norm(X)

    S = np.zeros_like(X)
    Y = np.zeros_like(X)
    L = np.zeros_like(X)
    count = 0
    while (np.linalg.norm(X-L-S) > thresh) and (count < 1000):
        L = SVT(X-S+(1/mu)*Y,1/mu)
        S = shrink(X-L+(1/mu)*Y,lambd/mu)
        Y = Y + mu*(X-L-S)
        count += 1
    return L,S
```

[ ]:
```
X = faces[:,:nfaces[0]]
X.shape
```

[ ]:
```
X = faces[:,:nfaces[0]]
L,S = RPCA(X)
```

```
[ ]: inds = (3,4,14,15,17,18,19,20,21,32,43)

     for k in inds:
         fig,axs = plt.subplots(2,2)
         axs = axs.reshape(-1)
         axs[0].imshow(np.reshape(X[:,k-1],(168,192)).T,cmap='gray')
         axs[0].set_title('X')
         axs[1].imshow(np.reshape(L[:,k-1],(168,192)).T,cmap='gray')
         axs[1].set_title('L')
         axs[2].imshow(np.reshape(S[:,k-1],(168,192)).T,cmap='gray')
         axs[2].set_title('S')
         for ax in axs:
             ax.axis('off')
```

```
[ ]: def singlepass(A,k,p):
         m = A.shape[0]
         n = A.shape[1]
         l = k+p
         Ω = np.random.normal(0,1,size=(n,k)) # n x k
         Ψ = np.random.normal(0,1,size=(l,m)) # l x m
         Y = A@Ω          # m x n  * n x k =     m x k
         W = Ψ@A          # l x m  * m x n =     l x n
         Q, R = np.linalg.qr(Y,mode='reduced')# m x k
         X = np.linalg.pinv(Ψ@Q)@W            # k x n
         Ahat = Q@X      # m x k  * k x n =      m x n
         pseudo = Normal(Ahat[:l,:l])
         error = np.linalg.norm(A-Q@Q.T@A,"fro")
         return(pseudo.plot([10**i for i in range(-5,5)], spectrum=eigvals(Ahat[:l,:
      ↪l])))
```

3.3 Pseudospectrum of approximations

```
[ ]: A = imread(os.path.join("/Users/nikosmatsavelas/Desktop/Thesis NTUA/gray.jpg"))
     m,n = A.shape
     pseudo = Normal(A[:m,:m])
     pseudo.plot([10**k for k in range(-4, 3)], spectrum=eigvals(A[:m,:m]))
```

```
[ ]: U, S, VT = np.linalg.svd(A,full_matrices=False) # Deterministic SVD

     r = 400 # Target rank
     q = 3   # Power iterations
     p = 5   # Oversampling parameter

     rU, rS, rVT = rSVD(A,k = r,q=p,p=p)
```

[ ]:
```
%%time
U, S, VT = np.linalg.svd(A,full_matrices=False)
```

[ ]:
```
%%time
rU, rS, rVT = rSVD(A,r,q,p)
```

[ ]:
```
# Reconstruction
```

[ ]:
```
%%time
XSVD = U[:,:(r+1)] @ np.diag(S[:(r+1)]) @ VT[:(r+1),:] # SVD approximation
pseudo = Normal(XSVD[:r+1,:r+1])
pseudo.plot([10**k for k in range(-4, 3)], spectrum=eigvals(XSVD[:r+1,:r+1]))
```

[ ]:
```
%%time
XrSVD = rU[:,:(r+1)] @ np.diag(rS[:(r+1)]) @ rVT[:(r+1),:] # SVD approximation
pseudo = Normal(XrSVD[:r+1,:r+1])
pseudo.plot([10**k for k in range(-4, 3)], spectrum=eigvals(XrSVD[:r+1,:r+1]))
```

[ ]:
```
k = 400
p = 10
l = k+p
pseudo = Normal(A[:m,:m])
pseudo.plot([10**k for k in range(-4, 3)], spectrum=eigvals(A[:m,:m]))
```

[ ]:
```
%%time
AH = singlepass(A=A,k=400,p=10)
print("The error is:",AH)
AH
```

[ ]:
```
def Nystrom(A,k,p):
    np.random.seed(15)
    m = A.shape[0] # rows
    n = A.shape[1] # columns
    G = np.random.normal(0,1, size=(n,k+p))
    Y = A@G
    Q, _ = np.linalg.qr(Y)
    B1 = A@Q
    B2 = Q.T@B1
    C = np.linalg.cholesky(B2)
    F = C.T@C
    u_tilde, Σ, v = np.linalg.svd(F, full_matrices = False)
    Λ = Σ**2
    h = u_tilde @ np.diag(Σ) @v.T
    return(h)
```

[ ]:
```
from scipy import sparse
from scipy.sparse import diags
```

```
N = 300
main_diag = 2*np.ones((N+1, 1)).ravel()
off_diag = -1*np.ones((N, 1)).ravel()

a = main_diag.shape[0]

diagonals = [main_diag, off_diag, off_diag]

A = sparse.diags(diagonals, [0,-1,1], shape=(a,a)).toarray()
```

```
[ ]: Ny = Nystrom(A=A,k=150,p=10)
     Ny.shape
```

```
[ ]: pseudo = Normal(A)
     pseudo.plot([10**k for k in range(-4, 3)], spectrum=eigvals(A))
```

```
[ ]: pseudo = Normal(Ny)
     pseudo.plot([10**k for k in range(-4, 3)], spectrum=eigvals(Ny))
```

```
[ ]: A = imread(os.path.join("/Users/nikosmatsavelas/Desktop/Thesis NTUA/gray.jpg"))
     m,n = A.shape
```

```
[ ]: pseudo = Normal(A[:k,:k])
     pseudo.plot([10**k for k in range(-4, 3)], spectrum=eigvals(A[:m,:m]))
```

```
[ ]: h2 = optim_rid(A, k)
     a_int = h2[0]
     pseudo = Normal(a_int[:k,:k])
     pseudo.plot([10**k for k in range(-4, 3)], spectrum=eigvals(a_int[:k,:k]))
```

```
[ ]:
```

```
[ ]:
```

A.3 Python Code Link (Github)

For full Python code implementation of tis thesis check the following link: https://github.com/nikosmatsa/Thesis-NTUA/blob/main/Thesis%20NTUA%20MS.c%20Applied%20Mathematics%20.ipynb