UNIVERSITY OF UDINE
hic sunt futura

Department of
Mathematics, Computer Science and Physics

BACHELOR THESIS IN
COMPUTER SCIENCE

A Randomized Algorithm for SVD Calculation

CANDIDATE
Massimo Fedrigo

SUPERVISOR
Prof. Enrico Bozzo

CO-SUPERVISOR
Prof. Rossana Vermiglio

Academic Year 2023-2024

Institute Contacts
Dipartimento di Scienze Matematiche, Informatiche e Fisiche
Università degli Studi di Udine
Via delle Scienze, 206
33100 Udine — Italia
+39 0432 558400
https://www.dmif.uniud.it/

Abstract

In this thesis, we apply a randomized algorithm [1] to the calculation of the Singular Value Decomposition (SVD).

In Chapter 1, we introduce Randomized Numerical Linear Algebra (RandNLA), its applications, and its importance in today's scientific landscape.

In Chapter 2, we outline the mathematical notation adopted and present the fundamental notions of linear algebra referenced throughout the thesis.

In Chapter 3, we introduce the SVD, explaining its utility and main applications, before defining it formally. We will also describe its variants, which are useful for reducing computation time or memory requirements. Of fundamental importance will be the choice of the *threshold*, which can be performed according to specific criteria. In particular, we will analyze the criterion proposed by M. Gavish and D. L. Donoho [6].

In Chapter 4, we introduce the randomized algorithm, named RSVD, which represents the core of this thesis. We will analyze its theoretical complexity and apply it to a preliminary test, followed by an iterative test on the number of columns of an input matrix. In this latter comparison, we will evaluate execution times and relative errors compared to those obtained with the classical truncated SVD. The test will be performed in two modes: first using the *threshold* proposed by M. Gavish and D. L. Donoho [6], then with a lower, logarithmic *threshold*, to observe improvements in execution times and the inevitable increase in errors.

In Chapter 5, we summarize the results obtained and evaluate the convenience, in terms of execution times, of using RSVD compared to the deterministic algorithm.

Finally, in Appendix A, we report the Matlab scripts used to perform the tests and generate the graphs.

Contents

1

Introduction

Randomized Numerical Linear Algebra (RandNLA) is an emerging field of numerical linear algebra that uses randomized algorithms to solve large-scale problems with high computational complexity [3,7]. This discipline has become increasingly relevant in the context of the growing need to process and analyze enormous amounts of data in real-time, especially in areas such as machine learning, statistics, and computational sciences.

Traditionally, many fundamental operations in linear algebra, such as matrix decomposition, solving linear systems, and approximating singular values, have been addressed through deterministic algorithms. While precise, these often struggle to handle the enormous dimensions of modern matrices. RandNLA introduces a significant alternative: leveraging the power of randomization to simplify and accelerate these calculations. Randomized methodologies are capable of providing accurate approximations using fewer computational resources compared to traditional approaches. For example, random sampling and low-dimensional projection allow for accurate approximation of the structures and properties of large matrices, significantly reducing the computation time required for the necessary operations. Below, we list the most commonly used randomization techniques in the context of matrix operations.

- **Element-wise sampling**: A subset of matrix elements is selected randomly and rescaled, obtaining a *sketch* that is an unbiased estimate of the original matrix. More accurate results are obtained by assigning higher probabilities to elements with larger absolute values.
- **Row or column sampling**: Rather than single elements, entire rows or columns are sampled, maintaining a linear structure that is often more efficient. This approach allows for better estimates compared to element-wise sampling.
- **Randomized projections**: The matrix is preprocessed with a random projection matrix to uniformly distribute information, facilitating subsequent uniform sampling without loss of accuracy.

The versatility of randomized algorithms has allowed their application to a wide range of problems, including data compression, information retrieval, and solving systems of linear equations. Furthermore, the ability to effectively handle large datasets has made RandNLA a key element in the development of advanced techniques in machine learning and artificial intelligence, where data dimensions and the need for rapid calculations are particularly pressing. Below are some concrete examples of RandNLA applications.

- **Least Squares and Low-Rank Approximation**: Random sampling and randomized projection allow for solving Least Squares (LS) problems and low-rank approximations more quickly and with acceptable relative errors.

- **Preconditioners for Iterative Algorithms**: RandNLA is used to build preconditioners that improve the performance of classical numerical algorithms, such as Blendenpik and LSRN, on large-scale LS problems. These algorithms now compete with or outperform traditional solutions like LAPACK.

- **Matrix Completion and Column Decomposition**: RandNLA supports low-rank matrix completion, useful for recommendation systems, by identifying and sampling rows and columns that preserve key information. This approach has also been used to decompose kernel matrices in machine learning.

- **Laplacian-based Linear System Solvers**: In contexts such as unsupervised machine learning, randomized techniques are used to create *sparse* versions of Laplacian matrices, reducing the number of edges and speeding up the resolution of linear systems.

- **Machine Learning and Statistics**: RandNLA improves efficiency and interpretability in big data analysis, including CX/CUR decompositions to identify significant relationships in various scientific fields, such as genetics and astronomy, as well as the analysis of kernel matrices, fundamental in kernel-based machine learning.

Concretely, RandNLA acts by randomly synthesizing input data to reduce the problem size while preserving all essential information. Algorithms for calculating low-rank approximations of matrices have historically been important in scientific computing, as they allow complex operations to be simplified and accelerated while maintaining a high degree of accuracy. Today, these algorithms find a new horizon of development in fields like machine learning and data analysis, where they are used to reduce dimensionality and improve the efficiency of computational models [4].

The history of computational linear algebra began in 1947 with Von Neumann and Goldstine, who proposed using computers for matrix calculations. In the following years, progress focused on languages and dedicated libraries like Fortran (1957) and BLAS (1979), which standardized basic operations on vectors and matrices, followed by LINPACK for solving complex linear systems. In the 80s and 90s, BLAS evolved to include advanced operations, and LAPACK extended its functionalities for high-level computing, supporting systems with dense matrices. ScaLAPACK in 1996 expanded LAPACK for distributed processing, while other libraries like ATLAS (automatic optimization on specific hardware), cuBLAS (for NVIDIA GPUs), and GPTune (2022, for advanced autotuning) further improved performance and adaptability. RandBLAS and RandLAPACK are concepts proposed as possible future standards for randomized linear algebra libraries. The idea is to create a software infrastructure similar to BLAS and LAPACK but optimized for algorithms utilizing randomization techniques. These projects aim to formalize and make the use of such algorithms more accessible. Although they are not yet official libraries, the proposal of RandBLAS and RandLAPACK reflects the growing interest in standard solutions for the efficient implementation of randomized algorithms.

In this thesis, we will apply Randomized Numerical Linear Algebra to the Singular Value Decomposition (SVD) and perform a comparative evaluation of the execution times of the classical algorithm versus its randomized equivalent (RSVD), varying the number of columns of the input matrix. We will focus on the computational improvements obtained through RSVD and how its efficiency can be modulated through the choice of column truncation. In particular, we will analyze how adopting smaller *thresholds* leads to increased efficiency, while highlighting the side effect of greater error approximation, already influenced by the randomized nature of the algorithm. The results demonstrate that RSVD is significantly faster than classical SVD, highlighting the optimal conditions for application and the balance between precision and performance to extract the maximum benefit from the randomized approach in practical contexts. The randomized algorithm we will analyze was proposed by Steven L. Brunton and J. Nathan Kutz [1] and is based on dimensional reduction through a randomized projection matrix. We will demonstrate that its complexity in terms of execution time is $\mathcal{O}(mnt)$, where t is the truncation threshold, which is better than classical SVD implementations that have complexity $\mathcal{O}(\max(m,n)(\min(m,n))^2)$.

2

Preliminary Linear Algebra Knowledge

In this chapter, we report the notation, definitions, and theorems of linear algebra that we will use throughout the thesis. Knowledge of fundamental linear algebra definitions, such as vector space, linearly independent or dependent vectors, space generated by a set of vectors, basis of a vector space, etc., is assumed. Main associated theorems, such as the existence of a basis theorem, are also taken for granted.

2.1 Notation

Scalars

$\tilde{e}_t \in \mathbb{R}$	Relative error of $\tilde{\boldsymbol{X}}_t$ with respect to $\boldsymbol{X}$.
$\hat{e}_t \in \mathbb{R}$	Relative error of $\hat{\boldsymbol{X}}_t$ with respect to $\boldsymbol{X}$.
$k \in \mathbb{N}$	Minimum between the number of rows and columns of a matrix.
$q_{ij} \in \mathbb{R}$	ij-th element of $\boldsymbol{Q}$, with $1 \leq i \leq m$ and $1 \leq j \leq m$.
$r \in \mathbb{N}$	Rank of a matrix.
$t \in \mathbb{N}$	Truncation rank (*threshold*) for Truncated SVD or RSVD.
$u_i \in \mathbb{R}$	i-th element of $\boldsymbol{u}$, with $1 \leq i \leq n$.
$v_i \in \mathbb{R}$	i-th element of $\boldsymbol{v}$, with $1 \leq i \leq n$.
$x_{ij} \in \mathbb{R}$	ij-th element of $\boldsymbol{X}$, with $1 \leq i \leq m$ and $1 \leq j \leq n$.
$\beta \in \mathbb{R}$	Ratio between rows and columns (or columns and rows) of $\boldsymbol{X}$.
$\gamma \in \mathbb{R}$	Known magnitude of white noise.
$\lambda \in \mathbb{C}$	Generic eigenvalue of a square matrix.
$\lambda_i \in \mathbb{R}$	i-th eigenvalue of $\boldsymbol{S}_u$, with $1 \leq i \leq m$.
$\lambda(\beta)$	Multiplicative coefficient of the optimal *threshold*.
$\mu_j \in \mathbb{R}$	j-th eigenvalue of $\boldsymbol{S}_v$, with $1 \leq j \leq n$.
$\mu_\beta \in \mathbb{R}$	Median value of the Marčenko-Pastur distribution.
$\pi \in \mathbb{R}$	Pre-established percentage of variance (or energy) in original data.
$\sigma_i \in \mathbb{R}$	i-th singular value of $\boldsymbol{X}$, with $1 \leq i \leq \min(m, n)$.
$\sigma_{\text{median}} \in \mathbb{R}$	Median singular value of $\boldsymbol{X}$.
$\tau \in \mathbb{R}$	Continuous law of the optimal *threshold*.

Vectors

$\boldsymbol{u} \in \mathbb{R}^n$	Generic vector.
$\boldsymbol{u}_i \in \mathbb{R}^m$	i-th left singular vector of $\boldsymbol{X}$, with $1 \leq i \leq m$.
$\boldsymbol{v} \in \mathbb{R}^n$	Generic vector.
$\boldsymbol{v}_j \in \mathbb{R}^n$	j-th right singular vector of $\boldsymbol{X}$, with $1 \leq j \leq n$.
$\boldsymbol{x}_i \in \mathbb{R}^n$	i-th row vector of $\boldsymbol{X}$, with $1 \leq i \leq m$.
$\boldsymbol{y}_j \in \mathbb{R}^m$	i-th column vector of $\boldsymbol{X}$, with $1 \leq j \leq n$.

Matrices

$\mathbf{0} \in \mathbb{R}^{m \times n}$	Generic zero matrix.
$\mathbf{0}_Y \in \mathbb{R}^{t \times (n-t)}$	Zero submatrix of $\boldsymbol{\Sigma}_Y$.
$\boldsymbol{I} \in \mathbb{R}^{m \times m}$	Identity matrix.
$\boldsymbol{P} \in \mathbb{R}^{n \times t}$	Random projection matrix used in RSVD.
$\boldsymbol{Q} \in \mathbb{R}^{m \times m}$	Generic square matrix.
$\boldsymbol{Q} \in \mathbb{R}^{m \times t}$	Matrix with orthonormal columns in QR decomposition.
$\boldsymbol{R} \in \mathbb{R}^{t \times t}$	Upper triangular matrix in QR decomposition.
$\boldsymbol{S} \in \mathbb{R}^{m \times m}$	Generic symmetric matrix.
$\boldsymbol{S}_u \in \mathbb{R}^{m \times m}$	Left symmetric matrix associated with $\boldsymbol{X}$.
$\boldsymbol{S}_v \in \mathbb{R}^{m \times m}$	Right symmetric matrix associated with $\boldsymbol{X}$.
$\boldsymbol{U} \in \mathbb{R}^{m \times m}$	Matrix of left singular vectors of $\boldsymbol{X}$.
$\boldsymbol{U}_\perp \in \mathbb{R}^{m \times (m-n)}$	Submatrix of $\boldsymbol{U}$ containing the last $m-n$ left singular vectors of $\boldsymbol{X}$.
$\boldsymbol{U}_k \in \mathbb{R}^{m \times k}$	Submatrix of $\boldsymbol{U}$ containing the first k left singular vectors of $\boldsymbol{X}$.
$\boldsymbol{U}_n \in \mathbb{R}^{m \times n}$	Submatrix of $\boldsymbol{U}$ containing the first n left singular vectors of $\boldsymbol{X}$.
$\boldsymbol{U}_r \in \mathbb{R}^{m \times r}$	Submatrix of $\boldsymbol{U}$ containing the first r left singular vectors of $\boldsymbol{X}$.
$\boldsymbol{U}_t \in \mathbb{R}^{m \times t}$	Submatrix of $\boldsymbol{U}$ containing the first t left singular vectors of $\boldsymbol{X}$.
$\boldsymbol{U}_Y \in \mathbb{R}^{t \times t}$	Matrix of left singular vectors of $\boldsymbol{Y}$.
$\hat{\boldsymbol{U}}_t \in \mathbb{R}^{m \times t}$	Matrix resulting from the projection $\boldsymbol{Q}\boldsymbol{U}_Y$.
$\boldsymbol{V} \in \mathbb{R}^{n \times n}$	Matrix of right singular vectors of $\boldsymbol{X}$.
$\boldsymbol{V}_\perp \in \mathbb{R}^{n \times (n-m)}$	Submatrix of $\boldsymbol{V}$ containing the last $n-m$ right singular vectors of $\boldsymbol{X}$.
$\boldsymbol{V}_k \in \mathbb{R}^{n \times k}$	Submatrix of $\boldsymbol{V}$ containing the first k right singular vectors of $\boldsymbol{X}$.
$\boldsymbol{V}_m \in \mathbb{R}^{n \times m}$	Submatrix of $\boldsymbol{V}$ containing the first m right singular vectors of $\boldsymbol{X}$.
$\boldsymbol{V}_r \in \mathbb{R}^{n \times r}$	Submatrix of $\boldsymbol{V}$ containing the first r right singular vectors of $\boldsymbol{X}$.
$\boldsymbol{V}_t \in \mathbb{R}^{n \times t}$	Submatrix of $\boldsymbol{V}$ containing the first t right singular vectors of $\boldsymbol{X}$.
$\boldsymbol{X} \in \mathbb{Q}^{m \times n}$	Rational matrix used as input in the tests of this thesis.
$\boldsymbol{X} \in \mathbb{R}^{m \times n}$	Generic matrix.
$\boldsymbol{X}_{\text{noise}} \in \mathbb{R}^{m \times n}$	White noise of $\boldsymbol{X}$.
$\boldsymbol{X}_t \in \mathbb{R}^{m \times n}$	Generic approximation of $\boldsymbol{X}$ at rank t.
$\boldsymbol{X}_{\text{true}} \in \mathbb{R}^{m \times n}$	Pure data of $\boldsymbol{X}$.
$\hat{\boldsymbol{X}}_t \in \mathbb{R}^{m \times n}$	Approximation of $\boldsymbol{X}$ at rank t calculated with RSVD.
$\tilde{\boldsymbol{X}}_t \in \mathbb{R}^{m \times n}$	Best approximation of $\boldsymbol{X}$ at rank t.

$\boldsymbol{Y} \in \mathbb{R}^{n \times m}$	Generic matrix.
$\boldsymbol{Y} \in \mathbb{R}^{t \times n}$	Projection of $\boldsymbol{X}$ via $\boldsymbol{Q}$ of the QR decomposition.
$\boldsymbol{Z} \in \mathbb{R}^{m \times t}$	Matrix resulting from the projection of $\boldsymbol{X}$ via $\boldsymbol{P}$.
$\boldsymbol{\Sigma} \in \mathbb{R}^{m \times n}$	Matrix of singular values of $\boldsymbol{X}$.
$\boldsymbol{\Sigma}_k \in \mathbb{R}^{k \times k}$	Square submatrix of $\boldsymbol{\Sigma}$ containing singular values of $\boldsymbol{X}$.
$\boldsymbol{\Sigma}_m \in \mathbb{R}^{m \times m}$	Square submatrix of $\boldsymbol{\Sigma}$ containing singular values of $\boldsymbol{X}$.
$\boldsymbol{\Sigma}_n \in \mathbb{R}^{n \times n}$	Square submatrix of $\boldsymbol{\Sigma}$ containing singular values of $\boldsymbol{X}$.
$\boldsymbol{\Sigma}_r \in \mathbb{R}^{r \times r}$	Square submatrix of $\boldsymbol{\Sigma}$ containing non-zero singular values of $\boldsymbol{X}$.
$\boldsymbol{\Sigma}_t \in \mathbb{R}^{t \times t}$	Square submatrix of $\boldsymbol{\Sigma}$ containing the first t non-zero singular values of $\boldsymbol{X}$.
$\boldsymbol{\Sigma}_Y \in \mathbb{R}^{t \times n}$	Matrix of singular values of $\boldsymbol{Y}$.
$\boldsymbol{\Sigma}_{Y,t} \in \mathbb{R}^{t \times t}$	Square submatrix of $\boldsymbol{\Sigma}_Y$ containing singular values of $\boldsymbol{Y}$.

Operators

$\cdot$	Dot product between two vectors.
$.^{-1}$	Inversion of a matrix.
$.^{T}$	Transposition of a matrix.
$\lceil \cdot \rceil$	Ceiling approximation of a scalar.
$\lfloor \cdot \rfloor$	Floor approximation of a scalar.
$\lvert \cdot \rvert$	Number of elements of a vector or matrix.
$\lvert \cdot \rvert$	Absolute value of a scalar.
$\lVert \cdot \rVert_2$	Euclidean norm of a vector.
$\lVert \cdot \rVert_F$	Frobenius norm of a matrix.
$\langle \cdot \rangle$	Space generated by a set of vectors.
$\underset{\star}{\operatorname{argmin}}\,(\cdot)$	Element of $\star$ that minimizes the expression $\cdot$.
$\log(\cdot)$	Natural logarithm of a scalar.
$\min(\cdot, \cdot)$	Minimum between two scalars.
$\max(\cdot, \cdot)$	Maximum between two scalars.
$\mathcal{O}(\cdot)$	Big-O notation of a function.
$\operatorname{rank}(\cdot)$	Operator to obtain the rank of a matrix.
$\operatorname{tr}(\cdot)$	Trace of a square matrix.

2.2 Definitions and Theorems

The following notions concern vectors and matrices in the field of real numbers $\mathbb{R}$. They are also valid in the field of rational numbers $\mathbb{Q}$, which will be used in the experimental part of this thesis. Definitions and properties related to Singular Value Decomposition (SVD) will be treated in Chapter 3.

Definition 2.1 (Dot Product) *The dot product between two vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^n$ is a scalar in $\mathbb{R}$ given*

by the sum of the products of the coordinates of $\boldsymbol{u}$ *and* $\boldsymbol{v}$*:*

$$\boldsymbol{u} \cdot \boldsymbol{v} = \sum_{i=1}^{n} u_i v_i = u_1 v_1 + u_2 v_2 + \cdots + u_n v_n$$

Definition 2.2 (Orthogonal Vectors) *Two vectors* $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^n$ *are orthogonal if their dot product is zero:* $\boldsymbol{u} \cdot \boldsymbol{v} = 0$*.*

Definition 2.3 (Euclidean Norm or L^2 Norm) *The Euclidean norm (or* L^2 *norm) of a vector* $\boldsymbol{v} \in \mathbb{R}^n$ *is given by:*

$$\|\boldsymbol{v}\|_2 = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}$$

Definition 2.4 (Normal Vector) *A vector* $\boldsymbol{v} \in \mathbb{R}^n$ *is called normal if its norm, according to a specific definition of norm, is 1.*

Definition 2.5 (Orthonormal Vectors) *Two vectors* $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^n$ *are orthonormal if they are orthogonal and both normal.*

Definition 2.6 (Row Space) *Let* $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m \in \mathbb{R}^n$ *be the vectors associated with the rows of the matrix* $\boldsymbol{X} \in \mathbb{R}^{m \times n}$*. We define the row space of* $\boldsymbol{X}$ *as* $\langle \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m \rangle$*.*

Definition 2.7 (Column Space) *Let* $\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_n \in \mathbb{R}^m$ *be the vectors associated with the columns of the matrix* $\boldsymbol{X} \in \mathbb{R}^{m \times n}$*. We define the column space of* $\boldsymbol{X}$ *as* $\langle \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_n \rangle$*.*

Theorem 2.1 *Let* $\langle \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m \rangle$ *and* $\langle \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_n \rangle$ *be the row and column spaces of* $\boldsymbol{X} \in \mathbb{R}^{m,n}$*. It holds that*

$$\dim(\langle \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m \rangle) = \dim(\langle \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_n \rangle)$$

Definition 2.8 (Upper and Lower Triangular Matrix) *A square matrix* $\boldsymbol{Q} \in \mathbb{R}^{m \times m}$ *is upper triangular if* $q_{ij} = 0$ *for* $m \geq i > j \geq 1$*; it is lower triangular if* $q_{ij} = 0$ *for* $m \geq j > i \geq 1$*.*

Definition 2.9 (Rank of a Matrix) *We define the rank* $r \leq \min(m, n)$ *of a matrix* $\boldsymbol{X} \in \mathbb{R}^{m \times n}$*, also denoted as* $\operatorname{rank}(\boldsymbol{X})$*, as the maximum number of linearly independent columns (or rows). Equivalently,* r *is the dimension of the space generated by the rows or columns (by the previous theorem, these are equal).*

Definition 2.10 (Full Rank Matrix) *A matrix* $\boldsymbol{X} \in \mathbb{R}^{m \times n}$ *is full rank if* $r = \min(m, n)$*.*

Unless specified otherwise, matrices described in this thesis will be considered, without loss of generality, to be full rank. Indeed, a large matrix has an almost zero probability of having linearly dependent columns (or rows). This probability is exactly zero if the matrix elements are generated by a continuous probability distribution. Moreover, in this latter case, the columns (or rows) of the matrix are nearly orthogonal vectors.

Definition 2.11 (Transposed Matrix) *The transpose* $\boldsymbol{X}^T \in \mathbb{R}^{n \times m}$ *of a matrix* $\boldsymbol{X} \in \mathbb{R}^{m \times n}$ *is the same matrix with rows and columns swapped.*

Definition 2.12 (Diagonal Matrix) *A square matrix $\boldsymbol{Q} \in \mathbb{R}^{m \times m}$ is diagonal if $q_{ij} = 0$ for $1 \leq i \neq j \leq m$. This definition can also be extended to rectangular matrices $\boldsymbol{X} \in \mathbb{R}^{m \times n}$, by setting $x_{ij} = 0$ for $i \neq j$, with $1 \leq i \leq m$ and $1 \leq j \leq n$.*

Definition 2.13 (Trace) *Given a square matrix $\boldsymbol{Q} \in \mathbb{R}^{m \times m}$, we define its trace as the sum of elements on the diagonal:*

$$\mathrm{tr}(\boldsymbol{Q}) = \sum_{i=1}^{m} x_{ii}$$

Property 2.1 (Properties of the Trace) *Given matrices $\boldsymbol{X} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{Y} \in \mathbb{R}^{n \times m}$, the following properties of the trace can be easily proven:*

- $\mathrm{tr}(\boldsymbol{X}\boldsymbol{X}^T) = \sum_{i=1}^{m} x_{ii}^2$;
- $\mathrm{tr}(\boldsymbol{X}\boldsymbol{Y}) = \mathrm{tr}(\boldsymbol{Y}\boldsymbol{X})$.

Definition 2.14 (Identity Matrix) *The identity matrix $\boldsymbol{I} \in \mathbb{R}^{m \times m}$ is a matrix having all elements on the main diagonal equal to 1 and all others equal to 0.*

Definition 2.15 (Eigenvector and Eigenvalue of a Square Matrix) *An eigenvector of a square matrix $\boldsymbol{Q} \in \mathbb{R}^{m \times m}$ is a vector $\boldsymbol{v} \in \mathbb{R}^m$ which, when subjected to the linear transformation $\boldsymbol{Q}$, produces as a result a scalar multiple of the original vector, i.e., $\boldsymbol{Q}\boldsymbol{v} = \lambda\boldsymbol{v}$. The scalar $\lambda \in \mathbb{C}$ is called an eigenvalue.*

The eigenvectors $\boldsymbol{v} \in \mathbb{R}^m$ of a square matrix $\boldsymbol{Q} \in \mathbb{R}^{m \times m}$ can be found by solving the characteristic equation $(\boldsymbol{Q} - \lambda\boldsymbol{I})\boldsymbol{v} = \boldsymbol{0}$, where the eigenvalues are found by setting $\det(\boldsymbol{Q} - \lambda\boldsymbol{I}) = 0$.

Theorem 2.2 *A symmetric matrix $\boldsymbol{S} \in \mathbb{R}^{m \times m}$ has real eigenvalues and orthogonal eigenvectors, which can be normalized.*

Definition 2.16 (Positive Semidefinite Matrix) *A symmetric matrix $\boldsymbol{S} \in \mathbb{R}^{m \times m}$ is positive semidefinite if, for every vector $\boldsymbol{v} \in \mathbb{R}^m$, the quadratic product is non-negative: $\boldsymbol{v}^T\boldsymbol{S}\boldsymbol{v} \geq 0$. This is equivalent to saying that its eigenvalues are non-negative.*

Definition 2.17 (Orthogonal Matrix) *A square matrix $\boldsymbol{Q} \in \mathbb{R}^{m \times m}$ is orthogonal if its columns or, equivalently, its rows are orthonormal vectors. This implies that*

- $\boldsymbol{Q}\boldsymbol{Q}^T = \boldsymbol{Q}^T\boldsymbol{Q} = \boldsymbol{I}$.
- *$\boldsymbol{Q}$ is always invertible, with its inverse equal to its transpose: $\boldsymbol{Q}^{-1} = \boldsymbol{Q}^T$;*

Definition 2.18 (QR Decomposition) *A matrix $\boldsymbol{X} \in \mathbb{R}^{m \times n}$, with $m \geq n$, can be factored as follows:*

$$\boldsymbol{X} = \boldsymbol{Q}\boldsymbol{R}$$

where $\boldsymbol{Q} \in \mathbb{R}^{m \times n}$ has orthonormal columns ($\boldsymbol{Q}^T\boldsymbol{Q} = \boldsymbol{I}$) and $\boldsymbol{R} \in \mathbb{R}^{n \times n}$ is upper triangular.

Below we report the definition of the Frobenius norm, which we will make extensive use of to compare a matrix $\boldsymbol{X}$ with its approximations. It can be seen as an extension of the Euclidean norm of vectors applied to matrices.

Definition 2.19 (Frobenius Norm) *The Frobenius norm of a matrix $\boldsymbol{X} \in \mathbb{R}^{m \times n}$ is defined as:*

$$\|\boldsymbol{X}\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |x_{ij}|^2},$$

i.e., the square root of the sum of the squares of the absolute values of the elements of $\boldsymbol{X}$.

Property 2.2 (Relation between Trace and Frobenius Norm) *It can be easily proven that the square of the Frobenius norm of a matrix $\boldsymbol{X} \in \mathbb{R}^{m \times n}$ is equal to the trace of $\boldsymbol{X}\boldsymbol{X}^{\boldsymbol{T}} \in \mathbb{R}^{m \times m}$:*

$$\|\boldsymbol{X}\|_F^2 = \mathrm{tr}(\boldsymbol{X}\boldsymbol{X}^T)$$

3

Singular Value Decomposition (SVD)

In this chapter, we discuss SVD, one of the most important matrix decompositions. After illustrating its applications and utility, we will define it formally. Subsequently, we will examine reduced variants of SVD, useful when the original matrix satisfies certain criteria. Finally, for truncated SVD, we will analyze the choice of the *threshold*, with a specific focus on the criterion proposed by M. Gavish and D. L. Donoho [6].

3.1 Motivations and Applications of SVD

Singular Value Decomposition (SVD) is one of the fundamental and most powerful techniques in linear algebra, with a broad impact across numerous scientific and technological fields. It is a methodology that allows decomposing a generic matrix into three distinct components: an orthogonal matrix of variability directions in the row space, a diagonal matrix capturing relative importance through singular values, and an orthogonal matrix representing variability directions in the column space. This decomposition provides a deep insight into the structural properties of the matrix and facilitates the analysis and manipulation of complex data.

One of the most common uses of SVD is in dimensionality reduction, a crucial process when working with large amounts of data containing redundancy or noise. In the field of image processing, for example, SVD is used to compress images by reducing the amount of information while keeping only the principal components: practically, less relevant details are removed while preserving the most important visual features. This process not only facilitates image storage and transmission but also allows for significantly reducing computation times without compromising visual quality too much.

Beyond dimensionality reduction, SVD finds a fundamental application in collaborative filtering, a method used in recommendation platforms like Netflix or Spotify. Here, SVD allows analyzing user preferences and correlating similar tastes, suggesting movies, TV series, or music tracks that might be of interest. In these contexts, decomposition allows reducing problem complexity, extrapolating only relevant information that best describes latent relationships between users and products.

The motivations for using SVD lie in its ability to simplify complex structures and highlight the principal components of a matrix. This technique manages to transform high-dimensional and complex problems into more accessible and interpretable representations, allowing noise reduction in data and

highlighting hidden trends and patterns. For example, in data analysis, SVD can be used to extract principal directions of variation in a dataset, facilitating the interpretation of correlations between variables or the identification of clusters of similar data.

In an increasingly data-oriented world, SVD proves to be an essential tool for extracting relevant information from vast and complex datasets. Its ability to find compact and meaningful representations enables not only more efficient data management but also a greater understanding of underlying structures that might otherwise remain hidden behind the complexity and vastness of available information. Whether it is big data, machine learning, bioinformatics, computational physics, or natural language processing, SVD represents a bridge between mathematical complexity and practical utility, allowing useful and practical results to be derived from heterogeneous and unstructured data.

3.2 General SVD

Consider a matrix $\boldsymbol{X} \in \mathbb{R}^{m \times n}$ and its transpose $\boldsymbol{X}^T \in \mathbb{R}^{n \times m}$. We perform the following matrix multiplications:

- $\boldsymbol{S}_u = \boldsymbol{X}\boldsymbol{X}^T \in \mathbb{R}^{m \times m}$;
- $\boldsymbol{S}_v = \boldsymbol{X}^T\boldsymbol{X} \in \mathbb{R}^{n \times n}$.

Multiplying a matrix by its transpose results in a symmetric square matrix. Such a matrix possesses real eigenvalues and orthonormal eigenvectors. In this case, the products $\boldsymbol{S}_u$ and $\boldsymbol{S}_v$ are symmetric square matrices: $\boldsymbol{S}_u$ is called the left symmetric matrix and $\boldsymbol{S}_v$ the right symmetric matrix (based on the position of $\boldsymbol{X}$ in the product). The eigenvectors $\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_m$ of $\boldsymbol{S}_u$ are called left singular vectors of $\boldsymbol{X}$, while the eigenvectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n$ of $\boldsymbol{S}_v$ are called right singular vectors of $\boldsymbol{X}$. The eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_m$ of $\boldsymbol{S}_u$ and $\mu_1, \mu_2, \ldots, \mu_m$ of $\boldsymbol{S}_v$ enjoy the following properties:

- $(\lambda_1 = \mu_1) \geq (\lambda_2 = \mu_2) \geq \cdots \geq \left(\lambda_{\min(m,n)} = \mu_{\min(m,n)}\right) \geq 0$;
- If $m >= n$, then $\lambda_i = 0$ for $n < i \leq m$; otherwise, if $n > m$, $\mu_i = 0$ for $m < i \leq n$.

Therefore, the matrices $\boldsymbol{S}_u$ and $\boldsymbol{S}_v$ are symmetric and positive semidefinite. The square roots of the eigenvalues are the singular values of the matrix $\boldsymbol{X}$ and we will denote them by $\sigma_1 = \sqrt{\lambda_1} = \sqrt{\mu_1}$, $\sigma_2 = \sqrt{\lambda_2} = \sqrt{\mu_2}$, $\ldots$, $\sigma_r = \sqrt{\lambda_{\min(m,n)}} = \sqrt{\mu_{\min(m,n)}}$. They provide an important measure of the degree of linear dependence of a matrix's columns. This makes singular values a crucial tool in data analysis and dimensionality reduction. It can be shown that $\boldsymbol{X}$ has rank $r \leq \min(m, n)$ if and only if $\boldsymbol{\Sigma}$ has r non-zero values on the diagonal:

- $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r \geq 0$;
- $\sigma_{r+1} = \sigma_{r+2} = \cdots = \sigma_{\min(m,n)} = 0$.

Definition 3.1 (Singular Value Decomposition) *Singular Value Decomposition (SVD) factors the matrix $\boldsymbol{X} \in \mathbb{R}^{m \times n}$ as follows:*

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$$

where

- $\boldsymbol{U} \in \mathbb{R}^{m\times m}$ *is an orthogonal square matrix whose columns are the left singular vectors* $\boldsymbol{u}_1, \boldsymbol{u}_2, ..., \boldsymbol{u}_m$ *of* $\boldsymbol{X}$*.*
- $\boldsymbol{\Sigma} \in \mathbb{R}^{m\times n}$ *is a rectangular diagonal matrix, whose elements on the diagonal are the singular values* $\sigma_1, \sigma_2, \ldots, \sigma_{\min(m,n)}$ *of* $\boldsymbol{X}$*, arranged in descending order;*
- $\boldsymbol{V} \in \mathbb{R}^{n\times n}$ *is an orthogonal square matrix whose columns are the right singular vectors* $\boldsymbol{v}_1, \boldsymbol{v}_2$, $\ldots, \boldsymbol{v}_n$ *of* $\boldsymbol{X}$*. In the decomposition, its transpose* $\boldsymbol{V}^T$ *is considered.*

We can thus express the matrix $\boldsymbol{X}$ as a sum of rank-1 matrices:

$$\boldsymbol{X} = \sum_{i=1}^{r} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T$$

Observe that it is more convenient to store individual column (or row) vectors $\boldsymbol{v}_i, \boldsymbol{u}_i$ rather than the entire matrix $\boldsymbol{X}$. Indeed, each addend $\boldsymbol{v}_i\boldsymbol{u}_i$ is formed by $m+n$ elements (m for $\boldsymbol{u}_i$, n for $\boldsymbol{v}_i$), from which we obtain that the number of elements needed is $r(m+n)$.

Using properties 2.1 and 2.2 we obtain

$$\|\boldsymbol{\Sigma}\|_F^2 = \operatorname{tr}\left(\boldsymbol{\Sigma}\boldsymbol{\Sigma}^T\right) = \sum_{i=1}^{r} {\sigma_i}^2$$

that is, the product $\boldsymbol{\Sigma}\boldsymbol{\Sigma}^T \in \mathbb{R}^{m\times m}$ is a diagonal matrix with the squares of the singular values on the diagonal. Now calculating the square of the Frobenius norm of $\boldsymbol{X}$, exploiting properties 2.1 and 2.2:

$$\begin{aligned}
\|\boldsymbol{X}\|_F^2 &= \operatorname{tr}(\boldsymbol{X}\boldsymbol{X}^T) \\
&= \operatorname{tr}\left(\left(\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T\right)\left(\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T\right)^T\right) \\
&= \operatorname{tr}(\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T\boldsymbol{V}\boldsymbol{\Sigma}^T\boldsymbol{U}^T) \\
&= \operatorname{tr}(\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{\Sigma}^T\boldsymbol{U}^T) \\
&= \operatorname{tr}(\boldsymbol{\Sigma}^T\boldsymbol{U}^T\boldsymbol{U}\boldsymbol{\Sigma}) \\
&= \operatorname{tr}(\boldsymbol{\Sigma}\boldsymbol{\Sigma}^T) \\
&= \|\boldsymbol{\Sigma}\|_F^2
\end{aligned}$$

We have thus demonstrated the following property linking the Frobenius norm to singular values:

Property 3.1

$$\|\boldsymbol{X}\|_F^2 = \sum_{i=1}^{r} \sigma_i^2$$

3.3 Economy SVD

When the dimensions of the matrix $\boldsymbol{X} \in \mathbb{R}^{m\times n}$ are very large, it is useful to consider a reduced or *economy* version of the decomposition, which preserves essential information by reducing the number of operations and memory space required.

If $m \geq n$, i.e., the number of rows exceeds that of columns, then the singular value matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{m \times n}$ can be subdivided into the following form:

$$\begin{bmatrix} \boldsymbol{\Sigma}_n \\ \mathbf{0} \end{bmatrix}$$

where

- $\boldsymbol{\Sigma}_n \in \mathbb{R}^{n \times n}$ contains the singular values of $\boldsymbol{X}$ on the diagonal;
- $\mathbf{0} \in \mathbb{R}^{(m-n) \times n}$ is the zero submatrix.

It is also possible to subdivide the matrix $\boldsymbol{U} \in \mathbb{R}^{m \times m}$ into two submatrices identifying two orthogonal and complementary spaces:

$$\begin{bmatrix} \boldsymbol{U}_n & \boldsymbol{U}_\perp \end{bmatrix}$$

where

- $\boldsymbol{U}_n \in \mathbb{R}^{m \times n}$ contains the first n left singular vectors;
- $\boldsymbol{U}^\perp \in \mathbb{R}^{m \times (m-n)}$ contains the remaining $m - n$ left singular vectors.

It is easy to verify the following equality:

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T = \begin{bmatrix} \boldsymbol{U}_n & \boldsymbol{U}_\perp \end{bmatrix} \begin{bmatrix} \boldsymbol{\Sigma}_n \\ \mathbf{0} \end{bmatrix} \boldsymbol{V}^T = \boldsymbol{U}_n \boldsymbol{\Sigma}_n \boldsymbol{V}^T$$

from which it is evident that $\boldsymbol{U}_\perp$ is not considered in the calculation, as it cancels out with $\mathbf{0}$.

Equivalently, it is possible to reformulate SVD in its economy version if the number of columns exceeds that of rows, $n \geq m$:

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T = \boldsymbol{U} \begin{bmatrix} \boldsymbol{\Sigma}_m & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{V}_m^T \\ \boldsymbol{V}_\perp^T \end{bmatrix} = \boldsymbol{U}\boldsymbol{\Sigma}_m \boldsymbol{V}_m^T$$

where

- $\boldsymbol{\Sigma}_m \in \mathbb{R}^{m \times m}$ contains the singular values of $\boldsymbol{X}$ on the diagonal;
- $\mathbf{0} \in \mathbb{R}^{m \times (n-m)}$ is the zero submatrix;
- $\boldsymbol{V}_m^T \in \mathbb{R}^{m \times n}$ contains the first m right singular vectors;
- $\boldsymbol{V}_\perp^T \in \mathbb{R}^{(n-m) \times n}$ contains the remaining $n - m$ left singular vectors.

where, this time, the useless part is $\boldsymbol{V}_\perp^T$, which is eliminated from the calculation.

Both cases can be unified into a general formulation of economy SVD:

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T = \boldsymbol{U}_k \boldsymbol{\Sigma}_k \boldsymbol{V}_k^T$$

where

- $k = \min(m, n)$;

- $\boldsymbol{U}_k \in \mathbb{R}^{m \times k}$ contains the first k columns of $\boldsymbol{U}$, i.e., the first k left singular vectors;
- $\boldsymbol{\Sigma}_k \in \mathbb{R}^{k \times k}$ contains the first k singular values of $\boldsymbol{X}$;
- $\boldsymbol{V}_k^T \in \mathbb{R}^{k \times n}$ contains the first k rows of $\boldsymbol{V}^T$, i.e., the first k right singular vectors.

Economy SVD is particularly useful when $k \ll \max(m, n)$. In this thesis, every SVD calculation will be performed with such a reduced version.

3.4 Compact SVD

The economy version of SVD can be further improved if the matrix $\boldsymbol{\Sigma}$ contains zero singular values. Indeed, we can rewrite SVD as follows

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T = \boldsymbol{U}_r\boldsymbol{\Sigma}_r\boldsymbol{V}_r^T$$

where

- r is the number of non-zero singular values present in $\boldsymbol{\Sigma}$;
- $\boldsymbol{U}_r \in \mathbb{R}^{m \times r}$ contains r left singular (column) vectors of $\boldsymbol{X}$;
- $\boldsymbol{\Sigma}_r \in \mathbb{R}^{r \times r}$ contains, on the diagonal, the r non-zero singular values of $\boldsymbol{X}$;
- $\boldsymbol{V}_r^T \in \mathbb{R}^{r \times n}$ contains r right singular (row) vectors of $\boldsymbol{X}$.

3.5 Truncated SVD and Approximation Hierarchy

In many cases, one even seeks to eliminate some non-zero singular values of $\boldsymbol{X}$ from the SVD calculation. In such case, SVD no longer represents an exact factorization of the original matrix $\boldsymbol{X}$, but provides an approximation $\boldsymbol{X}_t$ of lower rank $t \leq r$, called *threshold*, equal to the number of singular values kept. But that is not all: SVD truncated at a certain rank t provides the best approximation $\tilde{\boldsymbol{X}}_t$ of the original matrix $\boldsymbol{X}$ at rank t in terms of quadratic distance[1]. This is formalized in the following theorem:

Theorem 3.1 (Eckard-Young) *The best approximation $\tilde{\boldsymbol{X}}_t$ at rank t of a matrix $\boldsymbol{X} \in \mathbb{R}^{m \times n}$ is obtained with SVD truncated (TSVD) at rank t:*

$$\tilde{\boldsymbol{X}}_t = \underset{\boldsymbol{X}_t \ s.t. \ \text{rank}(\boldsymbol{X}_t)=t}{\text{argmin}} (\|\boldsymbol{X} - \boldsymbol{X}_t\|_F) = \boldsymbol{U}_t\boldsymbol{\Sigma}_t\boldsymbol{V}_t^T$$

where

- $\boldsymbol{U}_t \in \mathbb{R}^{m \times t}$ contains the first t columns of $\boldsymbol{U}$, i.e., the first t left singular vectors of $\boldsymbol{X}$;
- $\boldsymbol{\Sigma}_t \in \mathbb{R}^{t \times t}$ contains, on the diagonal, the first t non-zero singular values of $\boldsymbol{X}$;

[1] In other words, it is not possible to find another matrix of rank t that deviates from $\boldsymbol{X}$ less than $\tilde{\boldsymbol{X}}_t$ calculated with truncated SVD does. The measure of *deviation* is given by the Frobenius norm.

- $\boldsymbol{V}_t^T \in \mathbb{R}^{t\times n}$ contains the first t rows of $\boldsymbol{V}^T$, i.e., the first t right singular vectors of $\boldsymbol{X}$;
- $\|\boldsymbol{X} - \boldsymbol{X}_t\|_F$ is the Frobenius norm of the *deviation* of $\boldsymbol{X}_t$ from $\boldsymbol{X}$.

TSVD, therefore, by varying its truncation at rank t, provides a hierarchy of *optimal* approximations of the original matrix $\boldsymbol{X}$. Since the matrix $\boldsymbol{\Sigma}_t$ is diagonal, we obtain that the approximation $\tilde{\boldsymbol{X}}_t$ at rank t of $\boldsymbol{X}$ provided by SVD is given by

$$\tilde{\boldsymbol{X}}_t = \sum_{i=1}^{t} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T = \sigma_1 \boldsymbol{u}_1 \boldsymbol{v}_1^T + \sigma_2 \boldsymbol{u}_2 \boldsymbol{v}_2^T + \cdots + \sigma_t \boldsymbol{u}_t \boldsymbol{v}_t^T \approx \boldsymbol{X}$$

where the equality $\tilde{\boldsymbol{X}}_t = \boldsymbol{X}$ holds only if $t \geq r$. As demonstrated in Section 3.2 we obtain

$$\|\tilde{\boldsymbol{X}}_t\|_F^2 = \sum_{i=1}^{t} \sigma_i^2$$

It can be easily shown that the square of the Frobenius norm of the *deviation* is equal to the sum of squares of singular values *lost* after truncation:

$$\|\boldsymbol{X} - \tilde{\boldsymbol{X}}_t\|_F^2 = \sum_{i=t+1}^{r} \sigma_i^2$$

The ratio between the number of elements of the truncated factorization $\boldsymbol{U}_t \boldsymbol{\Sigma}_t \boldsymbol{V}_t^T$ and that of $\boldsymbol{X}$ is given by

$$\frac{|\boldsymbol{U}_t| + |\boldsymbol{\Sigma}_t| + |\boldsymbol{V}_t^T|}{|\boldsymbol{X}|} = \frac{mt + t^2 + tn}{mn}$$

If this ratio is ≤ 1 then, thanks to TSVD, $mn - (mt + t^2 + tn)$ memory space will be saved. It can be easily shown that, by storing the factors $\boldsymbol{U}_t$, $\boldsymbol{\Sigma}_t$, and $\boldsymbol{V}_t^T$ instead of the entire matrix $\boldsymbol{X}$, space can be saved if and only if

$$t \leq \left\lfloor \frac{\sqrt{(m+n)^2 + 4mn} - (m+n)}{2} \right\rfloor$$

Suppose, for example, that the matrix $\boldsymbol{X}$ has 2000 rows and 1500 columns. We obtain that the maximum *threshold* that allows saving space is $t = 712$. Below are listed some values of the ratio between total elements of the factorization and those of the original matrix, for some threshold values $t \leq 712$:

- If $t = 150$, then the ratio is approximately 18%;
- If $t = 100$, then the ratio is 12%;
- If $t = \left\lceil \sqrt{1500} \right\rceil = 39$, then the ratio is approximately 5%;
- If $t = \lceil \log 1500 \rceil = 8$, then the ratio is approximately 0%;

Observe that in the case of economy SVD, $t = r$. If $\boldsymbol{X}$ is full rank, then $t = r = n = 1500$ and the previous ratio becomes

$$\frac{mr + r^2 + nr}{mn} = \frac{2n^2 + mn}{mn} = \frac{2n + m}{m}$$

Substituting $m = 2000$ and $n = 1500$ we obtain that the ratio is $\dfrac{5}{2} = 250\%$.

In this thesis, such *truncated* version of the decomposition will be indicated with the acronym TSVD.

3.6 Choice of *Threshold* for Truncated SVD

The choice of the optimal *threshold* $t \in \mathbb{N}$ that captures the majority of significant singular values of $\boldsymbol{X} \in \mathbb{R}^{m \times n}$ is of fundamental importance when wanting to work with approximations of the original matrix.

A common technique consists of keeping only the singular values that preserve a pre-established percentage $\pi \leq 1$ of the variance (or energy) of the original data, i.e.,

$$\|\tilde{\boldsymbol{X}}_t\|_F^2 = \sum_{i=1}^{t} \sigma_i^2 \geq \pi \sum_{i=1}^{r} \sigma_i^2 = \pi \|\boldsymbol{X}\|_F^2$$

where commonly $0.9 \leq \pi \leq 1$, so that at least 90% of the variance can be captured. Although this approach is relatively simple, it is widely used.

Other techniques involve identifying the *elbow* point of the singular values of the matrix, i.e., the transition from significant singular values to relatively *small* ones. M. Gavish and D. L. Donoho [6] attempted to analytically calculate the optimal *threshold*. Suppose, by hypothesis, that the original matrix $\boldsymbol{X}$ is given by the sum of *pure* data $\boldsymbol{X}_{\text{true}} \in \mathbb{R}^{m \times n}$ and white noise[2] $\boldsymbol{X}_{\text{noise}} \in \mathbb{R}^{m \times n}$, according to the equation

$$\boldsymbol{X} = \boldsymbol{X}_{\text{true}} + \gamma \boldsymbol{X}_{\text{noise}}$$

where $\gamma \in \mathbb{R}$ is the noise magnitude. If γ is known, then the *threshold* t is given by the discretization of the following continuous law

$$\tau = \lambda(\beta)\sqrt{n}\gamma$$

where

$$\lambda(\beta) = \sqrt{\left(2(\beta+1) + \frac{8\beta}{\beta + 1 + \sqrt{\beta^2 + 14\beta + 1}}\right)}$$

is a function of the ratio between rows and columns (or columns and rows)

$$\beta = \begin{cases} \frac{m}{n} & \text{if } n \geq m, \\ \frac{n}{m} & \text{otherwise.} \end{cases}$$

Observe that, in the case of square matrices, $m = n$, we have $\beta = 1$ and $\lambda(\beta) = \dfrac{4}{\sqrt{3}}$, from which

$$\tau = \frac{4}{\sqrt{3}}\sqrt{n}\gamma$$

In Section A.1, we report the Matlab script we will use to calculate the *threshold* in the case of known γ.

In most practical cases, the magnitude γ is unknown and it is necessary to estimate the white noise.

[2] In this context, white noise refers to a statistical model where noise consists of independent and identically distributed (i.i.d.) random components, with zero mean and constant variance.

Gavish and Donoho's idea is to use the median singular value σ_{median}. In this case, it can be shown that the optimal *threshold* follows the law

$$\tau = \frac{\lambda(\beta)}{\mu_\beta}\sigma_{\text{median}}$$

where μ_β is the solution to the following integral equation[3]

$$\int_{(1-\beta)^2}^{\mu_\beta} \frac{\sqrt{((1+\sqrt{\beta})^2-\tau)(\tau-(1-\sqrt{\beta})^2)}}{2\pi\tau}\, d\tau = \frac{1}{2}$$

which can be approximated with a Matlab script provided by the authors themselves [5].

In the computation phase, since both $\lambda(\beta)$ and μ_β are real values, it is necessary to perform a discretization of $\tau \in \mathbb{R}$ to obtain an integer value of $t \in \mathbb{N}$. In our test, we chose to approximate the result by excess, using the operator $\lceil\cdot\rceil$. Furthermore, in some configurations, especially for boundary values of m and n, it might happen that $\tau > n$, a situation that makes no practical sense. Considering these observations, the calculation of t, both in the presence of known noise and when noise is estimated, becomes

Definition 3.2 (Optimal *Threshold* according to M. Gavish and D. L. Donoho)

$$t = \min\left(\lceil\tau\rceil, n\right), \quad \textit{where}$$

$$\tau = \begin{cases} \lambda(\beta)\sqrt{n}\gamma & \textit{if } \gamma \textit{ is known,} \\ \frac{\lambda(\beta)}{\mu_\beta}\sigma_{median} & \textit{otherwise.} \end{cases}$$

This formulation ensures that the value of t is always consistent with the dimensions of the matrix $\boldsymbol{X}$, respecting the practical and theoretical constraints of the problem. It can be shown, in both cases, that $t \in \mathcal{O}(\sqrt{n})$.

[3] This integral is known as the Marčenko-Pastur distribution. In the equation, μ_β is the median value of the distribution.

4

Randomized SVD

In this chapter, we introduce the use of randomization in SVD calculation. The algorithm we use is the one proposed by Steven L. Brunton and J. Nathan Kutz [1]. Initially, we will perform a simple preliminary test to observe, at first analysis, the performance improvements brought by RSVD and the errors introduced. Subsequently, we will perform an iterative test that will allow us to analyze execution times and errors of RSVD varying the number of columns of the input matrix, comparing results with those obtained with deterministic SVD. Finally, we will repeat the test using a logarithmic *threshold* and compare these results with those obtained using the *threshold* criterion proposed by M. Gavish and D. L. Donoho [6].

4.1 Motivations for Randomization

If a matrix has a low-rank structure, decomposition algorithms based on random sampling can obtain a good approximation of the low rank, but at a fraction of the computational cost compared to traditional (deterministic) methods. This approach is closely linked to the theory of sparsity and the geometry of high-dimensional spaces. With the exponential increase in data dimensions (e.g., 4K and 8K video, Internet of Things), the volume of measurements grows enormously, but the intrinsic rank of data does not increase proportionally. This means that, even if dataset dimensions grow, the relevant part of the information contained remains contained in a few principal components. Consequently, randomized techniques for matrix decomposition, like *Randomized SVD*, are becoming increasingly important as they allow handling this *deluge* of data with lower computational costs.

4.2 A Randomized Algorithm for SVD Calculation

Consider a *tall* and *skinny* matrix $\boldsymbol{X} \in \mathbb{R}^{m \times n}$, i.e., $m \geq n$ (possibly $m \gg n$). We study as an algorithm for randomized SVD calculation the one proposed by Steven L. Brunton and Nathan Kutz [1].

1. Projection $\boldsymbol{Z}$.

   We use $\boldsymbol{P} \in \mathbb{R}^{n \times t}$, with t obtained according to Definition 3.2, to project $\boldsymbol{X}$ onto a lower-dimensional space: $\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{P} \in \mathbb{R}^{m \times t}$. The obtained matrix $\boldsymbol{Z}$ is much smaller and more manage-

able than $\boldsymbol{X}$. The randomness of matrix $\boldsymbol{P}$ makes it highly unlikely that relevant components of matrix $\boldsymbol{X}$ are eliminated during projection.

2. QR Decomposition of $\boldsymbol{Z}$.

   We factorize $\boldsymbol{Z}$ using QR decomposition, obtaining $\boldsymbol{Z} = \boldsymbol{QR}$, with $\boldsymbol{Q} \in \mathbb{R}^{m \times t}$ matrix with orthonormal columns and $\boldsymbol{R} \in \mathbb{R}^{t \times t}$ upper triangular matrix. In this way, we find an orthonormal basis for matrix $\boldsymbol{Z}$ that approximates the orthonormal basis for the original matrix $\boldsymbol{X}$. This is very useful because it simplifies subsequent calculations: multiplying a matrix by a matrix with orthogonal columns will not introduce numerical distortions and preserves the geometric properties of the subspace. This is due to the fact that $\boldsymbol{Q}$ covers the entire subspace generated by the columns of $\boldsymbol{Z}$.

3. Orthogonal Projection $\boldsymbol{Y}$.

   We project $\boldsymbol{X}$ onto the subspace identified by $\boldsymbol{Q}$, obtaining $\boldsymbol{Y} = \boldsymbol{Q}^T\boldsymbol{X} \in \mathbb{R}^{t \times n}$. This reduces the dimensions of the problem, allowing SVD to be performed on a much smaller matrix, while still maintaining the most important components of $\boldsymbol{X}$. This happens because QR decomposition provides an orthonormal basis for the space generated by the columns of $\boldsymbol{Z}$, preserving the main directions of the projected subspace. In other words, $\boldsymbol{Q}$ approximates a rank-t space containing almost all essential information of $\boldsymbol{X}$. Thus, $\boldsymbol{X} \approx \boldsymbol{QY}$ holds.

4. SVD of $\boldsymbol{Y}$.

   We decompose $\boldsymbol{Y}$ with SVD, preferably using its economy version, obtaining thus

$$\boldsymbol{Y} = \boldsymbol{U}_Y\boldsymbol{\Sigma}_Y\boldsymbol{V}^T = \boldsymbol{U}_Y \begin{bmatrix} \boldsymbol{\Sigma}_{Y,t} & \mathbf{0}_Y \end{bmatrix} \begin{bmatrix} \boldsymbol{V}_t^T \\ \boldsymbol{V}_\perp^T \end{bmatrix} = \boldsymbol{U}_Y\boldsymbol{\Sigma}_{Y,t}\boldsymbol{V}_t^T$$

   with $\boldsymbol{U}_Y \in \mathbb{R}^{t \times t}$, $\boldsymbol{\Sigma}_Y \in \mathbb{R}^{t \times n}$, $\boldsymbol{\Sigma}_{Y,t} \in \mathbb{R}^{t \times t}$, $\mathbf{0}_Y \in \mathbb{R}^{t \times (n-t)}$, $\boldsymbol{V} \in \mathbb{R}^{n \times n}$, $\boldsymbol{V}_t^T \in \mathbb{R}^{t \times n}$ and $\boldsymbol{V}_\perp^T \in \mathbb{R}^{(n-t) \times n}$. This will be computationally less expensive than directly decomposing $\boldsymbol{X}$, because $\boldsymbol{Y}$ has smaller dimensions. For comparison, consider economy SVD on the entire original matrix:

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T = \begin{bmatrix} \boldsymbol{U}_n & \boldsymbol{U}_\perp \end{bmatrix} \begin{bmatrix} \boldsymbol{\Sigma}_n \\ \mathbf{0} \end{bmatrix} \boldsymbol{V}^T$$

   with $\boldsymbol{U} \in \mathbb{R}^{m \times m}$, $\boldsymbol{U}_n \in \mathbb{R}^{m \times n}$, $\boldsymbol{U}_\perp \in \mathbb{R}^{m \times (m-n)}$, $\boldsymbol{\Sigma} \in \mathbb{R}^{m \times n}$, $\boldsymbol{\Sigma}_n \in \mathbb{R}^{n \times n}$, $\mathbf{0} \in \mathbb{R}^{(m-n) \times n}$, $\boldsymbol{V} \in \mathbb{R}^{n \times n}$.

5. Construction of the approximated matrix $\hat{\boldsymbol{X}}_t$.

   To obtain the approximation $\hat{\boldsymbol{X}}_t$ of $\boldsymbol{X}$ it is sufficient to construct the matrix $\hat{\boldsymbol{U}}_t \in \mathbb{R}^{m \times t}$ such that

$$\hat{\boldsymbol{X}}_t = \hat{\boldsymbol{U}}_t\boldsymbol{\Sigma}_{Y,t}\boldsymbol{V}_t^T$$

   This can be done by multiplying $\boldsymbol{Q} \in \mathbb{R}^{m \times t}$ by the matrix $\boldsymbol{U}_Y$, obtained from the SVD of $\boldsymbol{Y}$:

$$\hat{\boldsymbol{U}}_t = \boldsymbol{QU}_Y$$

In other words, the approximation of the first t left singular vectors of $\boldsymbol{X}$ can be performed by projecting backward, into the space generated by the columns of $\boldsymbol{Q}$, the left singular vectors of $\boldsymbol{Y}$.

In Section A.2 we provide the Matlab code of the algorithm just described. The algorithm can be easily adapted for short and fat matrices ($m < n$). However, in this thesis, we will not analyze this case either from a theoretical or experimental point of view, as the obtained results would be analogous.

4.3 Theoretical Complexity Analysis

We now study the benefits, in terms of computational time, brought by randomization in calculating the SVD factorization of a matrix $\boldsymbol{X} \in \mathbb{R}^{m \times n}$. As a term of comparison, we use the complexity of the classic algorithm used to calculate SVD, whose complexity is $\mathcal{O}(mn^2)$. [2] For simplicity, we assume that matrix products $\boldsymbol{AB}$, with $\boldsymbol{A} \in \mathbb{R}^{i \times j}$ and $\boldsymbol{B} \in \mathbb{R}^{j \times k}$, are calculated using classic algorithms of complexity $\mathcal{O}(ijk)$. Let's analyze the complexity of each step of the randomized algorithm:

1. The projection $\boldsymbol{Z} = \boldsymbol{XP}$ has complexity $\mathcal{O}(mnt)$, where t is the number of columns of matrix $\boldsymbol{P}$. Matrix $\boldsymbol{P} \in \mathbb{R}^{n \times t}$ is randomly generated, but this has negligible cost compared to matrix multiplication.
2. QR decomposition on $\boldsymbol{Z} \in \mathbb{R}^{m \times t}$ has complexity $\mathcal{O}(mt^2)$, because $\boldsymbol{Z}$ has only t columns.
3. The projection $\boldsymbol{Y} = \boldsymbol{Q}^T\boldsymbol{X}$ has complexity $\mathcal{O}(mnt)$.
4. SVD of matrix $\boldsymbol{Y} \in \mathbb{R}^{t \times n}$ has complexity $\mathcal{O}(t^2 n)$. Note that $\boldsymbol{Y}$ has much smaller dimensions compared to $\boldsymbol{X}$ if $t \ll m$.
5. Reconstruction of $\boldsymbol{U}$ from multiplication $\boldsymbol{U} = \boldsymbol{Q}\hat{\boldsymbol{U}}_Y$ has complexity $\mathcal{O}(mt^2)$.

Summing all contributions, we obtain the total complexity:

$$\mathcal{O}(mnt) + \mathcal{O}(mt^2) + \mathcal{O}(t^2 n) = \mathcal{O}(\max(mnt, mt^2, t^2 n))$$

Since the randomized algorithm aims at reducing the dimensionality of the problem, we can assume $t \ll n$ without loss of generality. We thus obtain $\mathcal{O}(mnt)$. Assuming, plausibly, that the *threshold* is logarithmic with respect to column dimension, $t \in \mathcal{O}(\log n)$, complexity becomes $\mathcal{O}(mn \log n)$. If, instead, the *threshold* is the optimal one [6], $t \in \mathcal{O}(\sqrt{n})$, then $\mathcal{O}(mnt) = \mathcal{O}(mn\sqrt{n})$.

4.4 Execution of Randomized SVD on an Image

The proposed algorithm provides an approximation $\boldsymbol{U}_t = \boldsymbol{Q}\hat{\boldsymbol{U}}_Y \in \mathbb{R}^{m \times t}$ of the orthogonal matrix $\boldsymbol{U} \in \mathbb{R}^{m \times m}$ containing the left singular vectors of the original matrix $\boldsymbol{X} \in \mathbb{R}^{m \times n}$. We calculate the approximation error due to the use of randomized SVD, comparing it with that deriving from truncated SVD (without randomization). Let

- $\tilde{e}_t = \dfrac{\|\boldsymbol{X} - \tilde{\boldsymbol{X}}_t\|_F}{\|\boldsymbol{X}\|_F}$ relative error of $\tilde{\boldsymbol{X}}_t$, obtained from SVD of $\boldsymbol{X}$ truncated at rank t. From Theorem 3.1 we know that $\tilde{\boldsymbol{X}}_t$ is the matrix that best approximates $\boldsymbol{X}$ at rank t.

- $\hat{e}_t = \dfrac{\|\boldsymbol{X} - \hat{\boldsymbol{X}}_t\|_F}{\|\boldsymbol{X}\|_F}$ relative error of $\hat{\boldsymbol{X}}_t$, obtained from randomized SVD of $\boldsymbol{X}$ truncated at rank t. This will be greater than the previous one and we want to evaluate its magnitude.

We consider as an example matrix an image $\boldsymbol{X} \in \mathbb{Q}^{2397\times 1795}$ whose elements are pixels (Figure 4.1). Let $\gamma = 1$. We set the magnitude of white noise. The *threshold* t, calculated as indicated in Definition 3.2, will be $t = \min\left(\lceil \lambda(\beta)\sqrt{n}\gamma \rceil, n\right) = 92$. The reduced column space is $\dfrac{t+p}{n} = \dfrac{92}{1795} \approx 5\%$ of the original one. The original matrix, of dimension $2397 \cdot 1795 = 4302615$, can be factored into three matrices whose overall dimension is $2397 \cdot 92 + 92^2 + 92 \cdot 1795 = 394128$, about 9%.

Figure 4.1: Original image.

Figure 4.2: Image after truncated SVD.

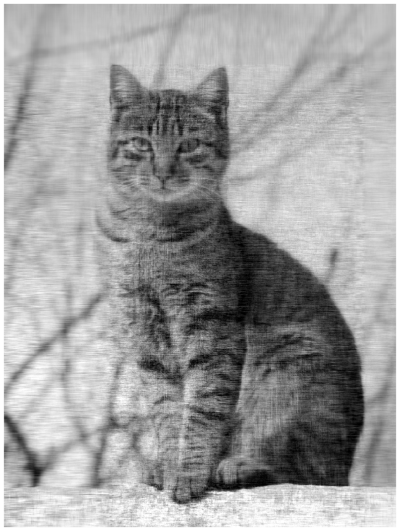

Figure 4.3: Image after randomized SVD.

We obtain, as output, the relative errors and execution times described in Table 4.1. The error

	Relative Error	**Execution Time** $[s]$
TSVD	0.007,076	2.5259
RSVD	0.020,356	0.047,03

Table 4.1: Errors and execution times for the two SVD variants.

$e_{\text{RSVD}} = 0.020356$ of randomized SVD is about 3 times the error $e_{\text{TSVD}} = 0.007076$ of truncated SVD. Such loss of accuracy is justified by an execution time 54 times lower compared to that of truncated SVD. Based on context and concrete application, one must choose which of the two aspects, performance or accuracy, to value. For large-scale matrices[1], it is often useful to reduce execution time even if this might induce an error on the final result.

In Figure 4.4, which depicts singular values in descending order as a percentage of their total sum, we can note that only some of them are truly significant. The highest singular value, alone, has a magnitude equal to 21% of the sum of all singular values. The *threshold* used to truncate SVD captures the first 92 singular values, whose sum is about 56% of the total.

In summary, results obtained indicate that the *optimal* truncation, corresponding in this case to 5% of column space, allowed capturing 56% of significant components of the original matrix, while

[1] Usually, in the context of RandNLA, matrices considered *large* have millions of rows and thousands, or millions, of columns.

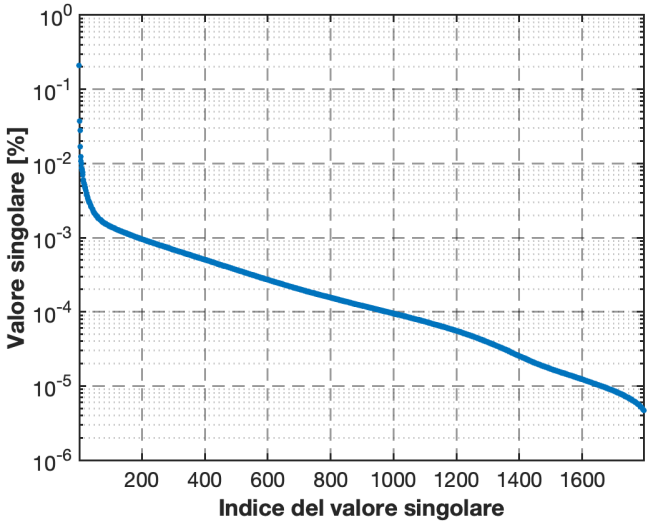

Figure 4.4: Singular values of $\boldsymbol{X}$ as a percentage of the total sum of singular values (logarithmic scale on the y-axis).

randomization reduced execution time by a factor of 54 compared to the deterministic method. In Section A.3 we report the Matlab script that allowed performing this test.

4.5 Performance and Accuracy Test of RSVD

In this section, we analyze, varying input matrix columns, the performance and relative error of randomized and truncated SVD. The input matrix will be a square image $\boldsymbol{X} \in \mathbb{Q}^{12000\times12000}$, reported in Figure 4.5 together with its approximations through the two SVD variants (Figure 4.6 and 4.7).

Figure 4.5: Original image.

Figure 4.6: Image after truncated SVD.

Figure 4.7: Image after randomized SVD.

The experiment consists of keeping the number of rows fixed (12000) and increasing the number of columns from 120 to 12000; at each step we calculate both versions of SVD, storing execution times, relative errors, and the *threshold*. We set increment granularity to 120, so as to acceptably reduce computations by 120 times. The *threshold* t will be calculated at each computation, as indicated in Definition 3.2, with $\gamma = 1$. Data obtained from the experiment result similar keeping the number of columns fixed and varying rows; consequently, we will not further deepen this case.

Results show a significant correlation[2] between execution time and relative error, as well as between time and *threshold*. Specifically, correlation between time and relative error is equal to -0.860530, indicating a strong inverse relationship: as execution time increases, error tends to decrease. Conversely, correlation between time and *threshold* is positive (0.926408), suggesting that matrices with a greater number of columns require longer calculation times, and this is predictably linked to the increase in target rank. Moreover, error decreases as *threshold* increases, as highlighted by negative correlation between error and *threshold* (-0.919566).

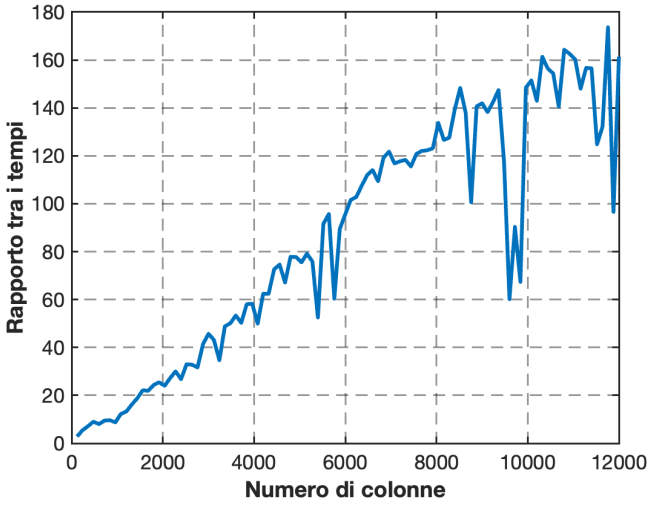

Figure 4.8: Ratio, varying column count, between execution time of TSVD and that of RSVD.

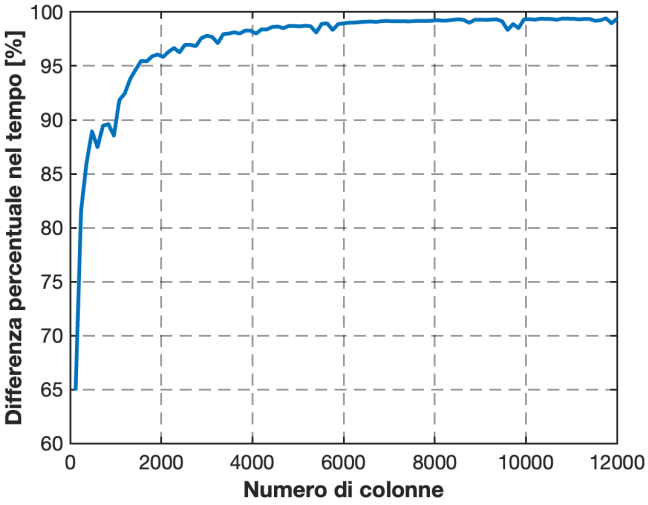

Figure 4.9: Differences between execution times of TSVD and RSVD as percentage of TSVD times, varying columns.

As discussed in Section 4.3, complexity of the deterministic algorithm is $\mathcal{O}(mn^2)$, while that of the randomized algorithm is $\mathcal{O}(mnt)$. This latter complexity becomes $\mathcal{O}(mn\sqrt{n})$ when the optimal value of parameter t, as indicated in Definition 3.2, is $t = \min\left(\lceil \lambda(\beta)\sqrt{n}\gamma \rceil, n\right) \in \mathcal{O}(\sqrt{n})$. Consequently, the ratio between execution times should follow a trend equal to $\mathcal{O}\left(\frac{mn^2}{mn\sqrt{n}}\right) = \mathcal{O}(\sqrt{n})$, a trend confirmed by the graph obtained experimentally (Figure 4.8).

Average execution time is 88.870873 seconds for the deterministic algorithm and 0.765446 for the randomized one, the latter equal to 0.86% of that of TSVD. Maximum time measured to execute TSVD, as intuitive, is that to decompose the complete matrix (12000×12000): 299.058596 seconds. RSVD has maximum time of 2.921871 seconds for 11880 columns. Maximum duration measured for RSVD is 102 times lower compared to that of TSVD. Minimum execution time of TSVD is trivially recorded for the smallest matrix (120×120): 0.086681 seconds. Minimum time of RSVD is 0.017116 seconds, corresponding to 240 columns.

[2]Pearson correlation between two variables X and Y is defined by the formula

$$\frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2 \sum_{i=1}^{n}(Y_i - \bar{Y})^2}}$$

where

- X_i and Y_i are values of the two variables;
- $\bar{X}$ and $\bar{Y}$ are means of variables X and Y, respectively;
- n is the number of data pairs.

Regarding differences in execution times, average difference between TSVD and RSVD is 88.105 seconds, with a maximum difference of 297.206 seconds for 12000 columns, while minimum difference is only 0.056 seconds for 120 columns. Such differences are better visualized as percentage of TSVD execution times, as reported in Figure 4.9. Observe that such difference starts to exceed 90% after 1000 columns. That is, it is sufficient that the ratio between columns and rows is 1/12, for tall and skinny matrices, for RSVD to start being 90% more efficient than TSVD.

Average relative error, varying columns, is 0.058845 for TSVD and 0.094308 for RSVD, 1.6 times more imprecise. As we can note in Figure 4.11 relative error has a similar trend between TSVD and RSVD. Indeed, maximum error is 0.084452 for TSVD and 0.137948 for RSVD both corresponding to 1680 columns. Minima varying columns are 0.043995, corresponding to 10080 columns for TSVD, and 0.069688, corresponding to 9960 columns, for RSVD. The *irregular* trend of relative error varying columns, with a maximum around 7/14 of total columns and a minimum towards 5/6, and with inflections around 4000 and 7000 columns, can be interpreted by analyzing singular value distribution in different submatrices. In particular, we will observe the trend of percentage ratio between sum of first t singular values and their overall sum:

$$\frac{\sum_{i=1}^{t} \sigma_i}{\sum_{i=1}^{r} \sigma_i} \cdot 100$$

As shown in Figure 4.10, the sum of the first t singular values varies significantly: going from 75% to 58% of the total when the number of columns increases from 120 to 1680. This explains the initial increase in error, up to the maximum. Subsequently, between 1680 and 8760 columns, the percentage sum of the first t singular values starts to grow slightly again, going from 58% to 61%. This variation is reflected in a corresponding decrease in relative error. Beyond 8760 columns, the percentage sum begins to decrease again, although more slowly than in the initial phase. This behavior justifies the final increase in relative error.

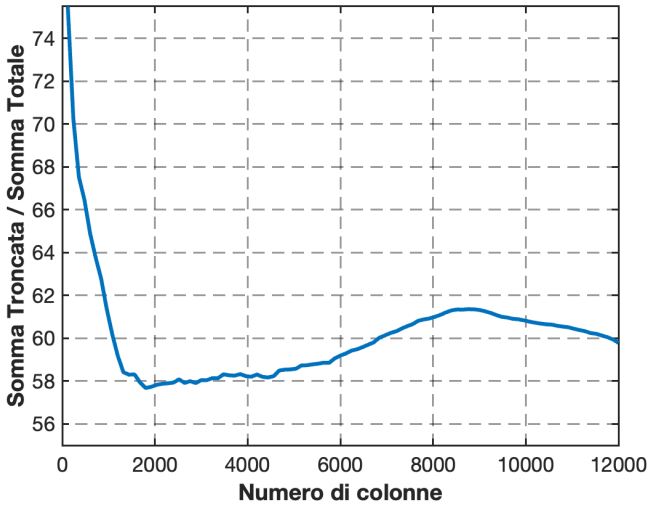

Figure 4.10: Sum of truncated singular values as percentage of sum of all singular values.

Differences in relative errors of RSVD and TSVD are as follows: average difference is 0.0355, with a maximum difference of 0.0535 for 1680 columns, and a minimum difference of 0.0257 for 9960 columns. Differences as percentage of RSVD relative errors are reported in Figure 4.12. Differences of errors as

percentage of RSVD errors stabilize at 37% after 4000 columns (corresponding, respectively, to a matrix with row-column ratio of 3 : 1). This means that, for matrices with such ratio, RSVD will have a relative error 20% higher compared to that of TSVD once 4000 columns are exceeded.

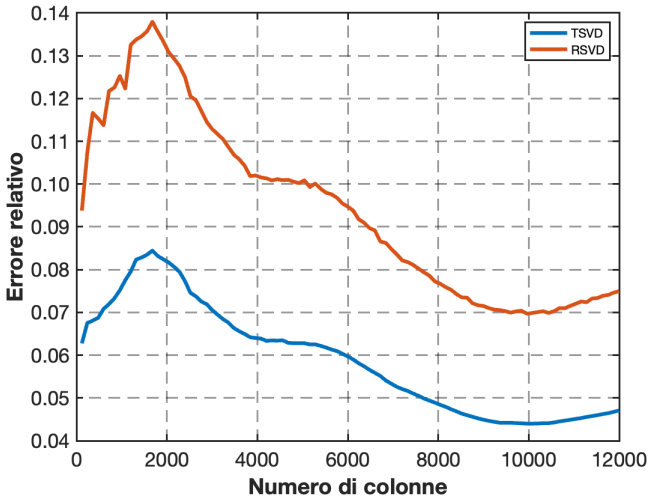

Figure 4.11: Relative error varying number of columns.

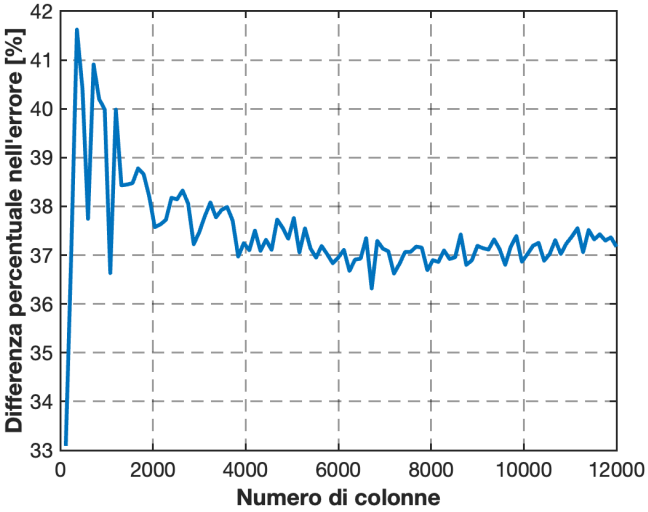

Figure 4.12: Differences between relative errors between RSVD and TSVD as percentage of RSVD errors, varying columns.

The *threshold* follows the trend of $\sqrt{n}$. Not surprisingly, it is sufficient to analyze the function that defines it. In the case of fixed rows and variable columns we have $\beta(m,n) = \min(m/n, n/m) = n/m$ since in this case columns, during iterations, are fewer than or equal to rows. Moreover, we note that $m = 12000$ is fixed, thus $\beta(m,n) = \beta(n)$ is a function in the domain of columns. The same will hold for $\lambda(\beta) = \lambda(n)$ and for τ, which will be

$$\tau(n) = \lambda(n)\sqrt{n}\gamma = \gamma\sqrt{\left(2\left(\frac{n}{m}+1\right) + \frac{8\frac{n}{m}}{\frac{n}{m}+1+\sqrt{\frac{n}{m}^2+14\frac{n}{m}+1}}\right)n}$$

in which known parameters are $m = 12000$ and $\gamma = 1$. We obtain the graph in Figure 4.13. Obviously, the graph of *threshold* t obtained during the experiment follows[3] the continuous behavior τ, therefore we will refer indistinctly to both. Varying columns, the maximum obtained is $t = 253$ corresponding to the maximum matrix (12000×12000). The minimum is $t(n = 120) = 16$. Average *threshold* is 150. In Figure 4.14 we report the ratio between *threshold* and columns as a percentage.

In conclusion, results indicate that, although RSVD offers significant advantages in terms of execution time compared to TSVD, especially for large matrices, accuracy of the randomized method tends to be slightly lower, although the difference is relatively small. The choice of *threshold* confirms to be a key parameter for both execution time and decomposition accuracy. In Section A.4 we report the Matlab code that allowed performing the test on the input image and obtaining all previously described graphs and data.

[3]Naturally, except for approximations or unacceptable boundary values, as in the case where $t > n$, for which we impose $t = n$. As indicated in Definition 3.2, we set $t = \min(\lceil\tau\rceil, n)$.

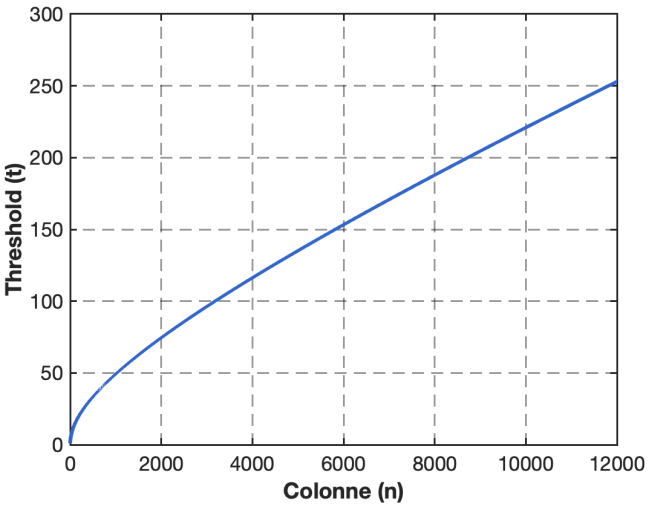

Figure 4.13: Graph of continuous *threshold* τ varying columns.

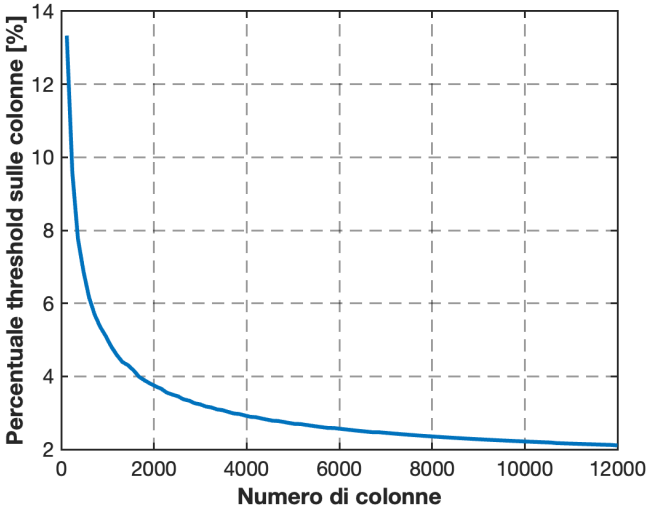

Figure 4.14: *Threshold* t as percentage of column count of the test.

	Time vs Error	**Time vs Threshold**	**Error vs Threshold**
Correlation	−0.860,530	0.926,408	−0.919,566

Table 4.2: Linear correlations between execution time, relative error, and threshold.

	Avg Time [s]	**Max Time** [s]		**Min Time** [s]	
		Value	**Cols**	**Value**	**Cols**
TSVD	88.870,873	299.058,596	12,000	0.086,681	120
RSVD	0.765,446	2.921,871	11,880	0.017,116	240

Table 4.3: Execution times of TSVD and RSVD, with corresponding column counts.

	Avg Error	**Max Error**		**Min Error**	
		Value	**Cols**	**Value**	**Cols**
TSVD	0.058,845	0.084,452	1680	0.043,995	10,080
RSVD	0.094,308	0.137,948	1680	0.069,688	9960

Table 4.4: Relative errors of TSVD and RSVD, with corresponding column counts.

	Mean	**Max**		**Min**	
		Value	**Cols**	**Value**	**Cols**
Time Difference [s]	88.105	297.206	12,000	0.056	120
Error Difference	0.0355	0.0535	1680	0.0257	9960

Table 4.5: Differences in execution times and errors of TSVD and RSVD, with corresponding column counts.

4.6 Test with Logarithmic *Threshold*

In the previous section, we compared performance and accuracy of TSVD and RSVD using a *threshold* chosen as described in Definition 3.2. Now we explore performance of both algorithms considering, experimentally, a *threshold* lower than the optimal one proposed by Donoho and Gavish [6]. Also in this

	Avg	Max		Min	
		Value	Cols	Value	Cols
Threshold	150	253	12,000	16	120

Table 4.6: *Threshold* of TSVD and RSVD, with corresponding column counts.

case, we will vary the number of columns of the same image $\boldsymbol{X} \in \mathbb{R}^{12000\times12000}$.

Figure 4.15: Original image.

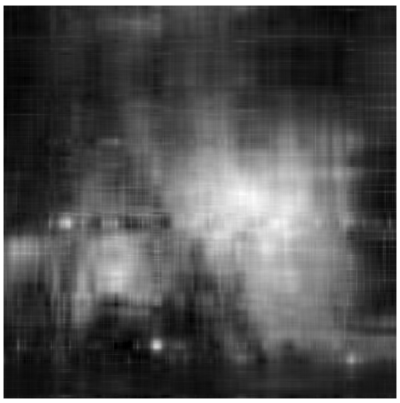

Figure 4.16: Image after truncated SVD.

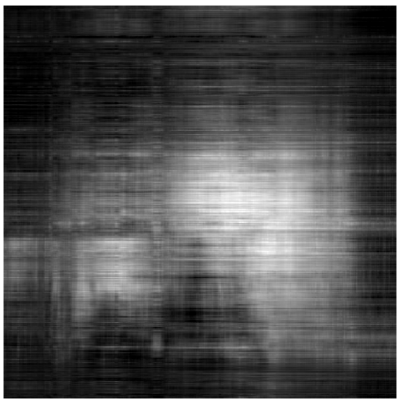

Figure 4.17: Image after randomized SVD.

We set $t = \lceil \log n \rceil$. Theoretical complexity of the randomized algorithm is $\mathcal{O}(mn \log n)$, from which the ratio between TSVD and RSVD complexity is derived: $\mathcal{O}\left(\frac{mn^2}{mn \log n}\right) = \mathcal{O}\left(\frac{n}{\log n}\right)$. The graph in Figure 4.18, obtained experimentally, confirms this theoretical trend. We now study average,

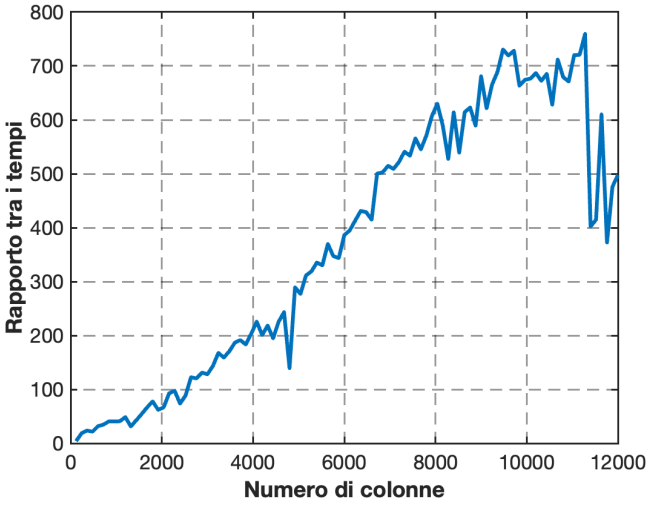

Figure 4.18: Ratio between execution time of TSVD and that of RSVD, with $t = \lceil \log n \rceil$.

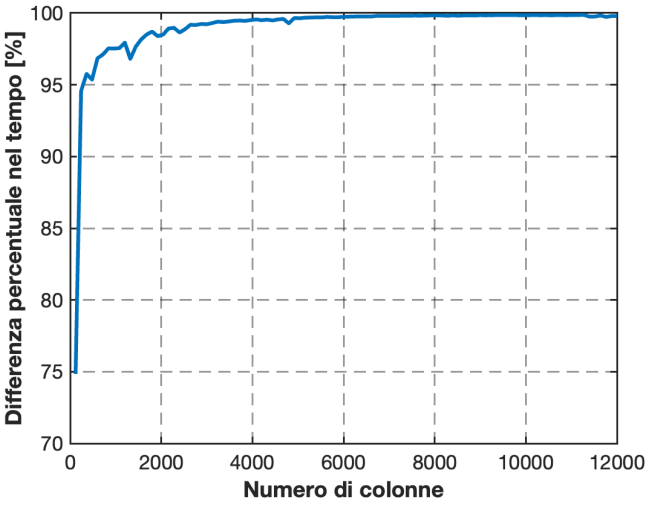

Figure 4.19: Differences between execution times of TSVD and RSVD as percentage of TSVD times, with $t = \lceil \log n \rceil$.

maximum, and minimum execution times obtained with the new threshold, comparing them with those of the previous experiment. Since execution time of TSVD does not depend on the *threshold* used[4], results are completely similar to previous ones. Average time is 0.1777 seconds, showing a significant reduction of 76.8%. Regarding maximum time, we observe a decrease of 74%, corresponding to 0.7556

[4]Truncation happens after executing classical SVD on the entire input matrix.

seconds on 11760 columns. Minimum execution time for RSVD, measured on 240 columns (minimum matrix dimension), is 0.0048 seconds, about 72% lower than before.

Figure 4.19 shows percentage difference in execution times between the two algorithms, compared to TSVD times. It has a trend analogous to that of the previous section (95% is however reached already at 240 columns, instead of after 1000). Average difference is equal to 88.34 seconds, with a maximum difference of 283.70 seconds (for the 12000×12000 matrix) and a minimum difference of 0.063 seconds (for the 12000×120 matrix). These results are quite similar to those obtained in the experiment of the previous section, as efficiency improvements, even of 75%, for RSVD are not significant when matrix dimensions are reduced.

Thus, from time comparison, it is evident that setting $t = \lceil \log n \rceil$ significantly increases RSVD efficiency. However, it is necessary to consider that reducing the threshold can lead to an increase in error; it will therefore be necessary to evaluate whether such increase is acceptable or compromises data integrity. Average relative error is equal to 0.1803 for TSVD, with an increase of 206%, and 0.2520 for

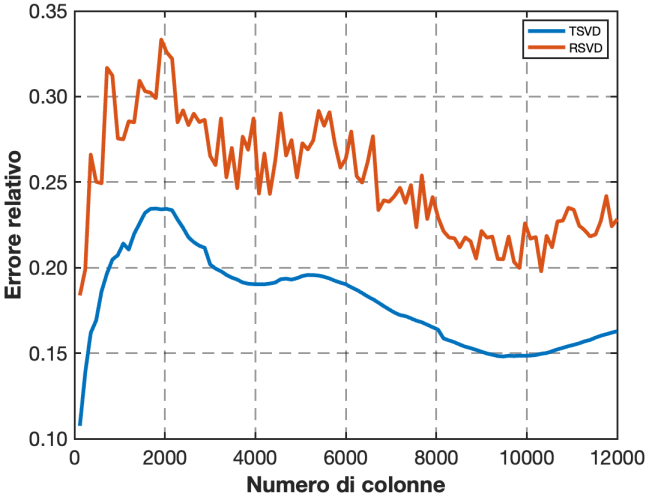

Figure 4.20: Relative error, with $t = \log n$.

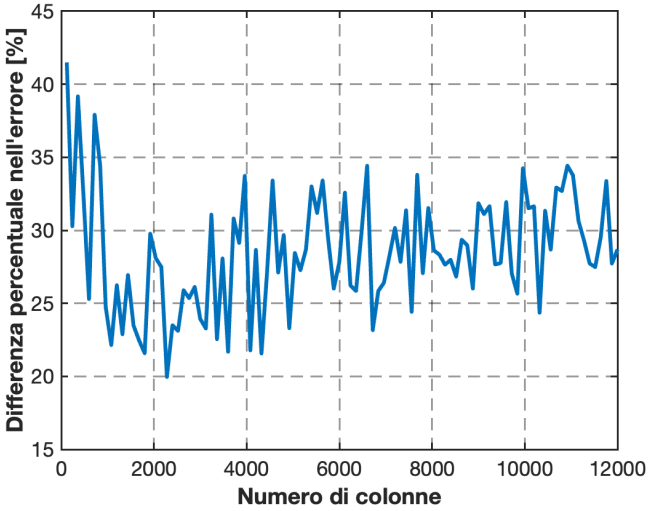

Figure 4.21: Differences between relative errors between RSVD and TSVD as percentage of RSVD errors, with $t = \log n$.

RSVD, with an increase of 166%. Maximum error for TSVD reaches 0.2345 (increase of 178%) on 1800 columns, while for RSVD maximum error is 0.3333 (increase of 142%) on 1920 columns. Minimum values are recorded with 120 columns for both algorithms, with 0.1075 for the deterministic algorithm (increase of 144%) and 0.1838 for the randomized algorithm (increase of 164%). Also in this case inflection points are observed around 4000 and 7000 columns, highlighting similarity in relative error trend between the two tests. This behavior can be explained, analogously to the previous test, by analyzing percentage ratio between truncated sum and total sum of singular values (Figure 4.22). In range between 120 and 1680 columns, ratio decreases from 64% to 30%, highlighting a decidedly more marked drop compared to that recorded in the previous test. Subsequently, percentage stabilizes, with negligible variations both increasing and decreasing. Starting from 8760 columns, a new decrease is noted culminating in a minimum of about 27%.

Mean error difference between the two algorithms is equal to 0.0717 (increase of 102%), with a maximum difference of 0.1201 (increase of 124%) on 720 columns and a minimum difference of 0.0482

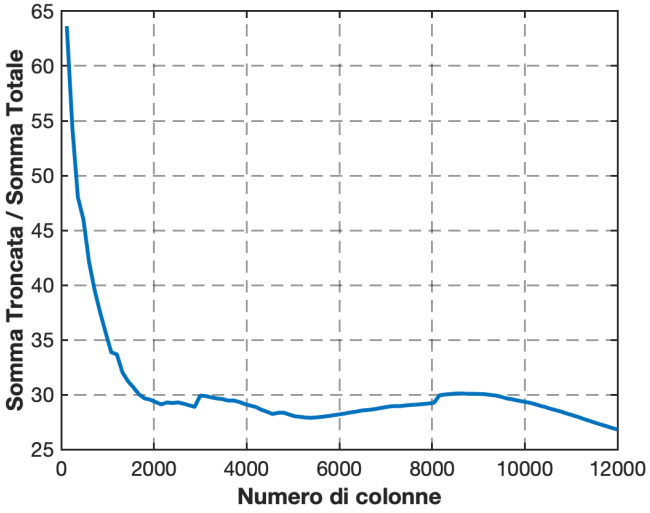

Figure 4.22: Sum of truncated singular values as percentage of sum of all singular values.

(increase of 87.5%) on 10320 columns. Figure 4.21 shows error difference as percentage compared to RSVD errors.

In summary, adoption of a logarithmic *threshold* entails performance increase of 75% for RSVD, but implies doubling (at least) of error, which must be considered in evaluating result quality. The magnitude of errors can be visualized comparing versions of input image (Figure 4.15, 4.16 and 4.17). As can be noted, it becomes quite unrecognizable after RSVD.

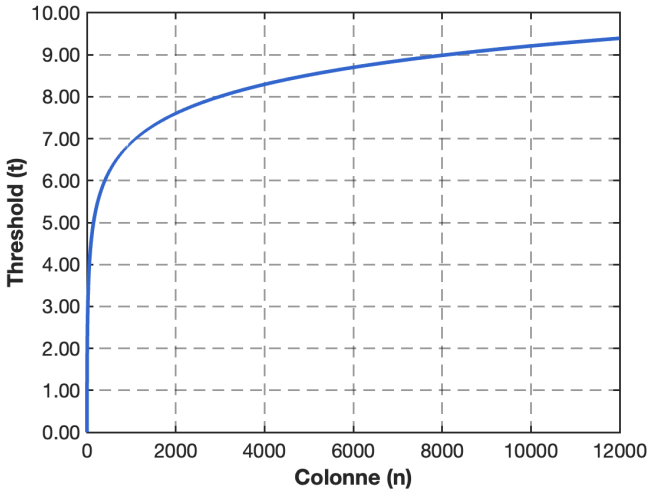

Figure 4.23: Analytical graph of *threshold* $t = \log n$.

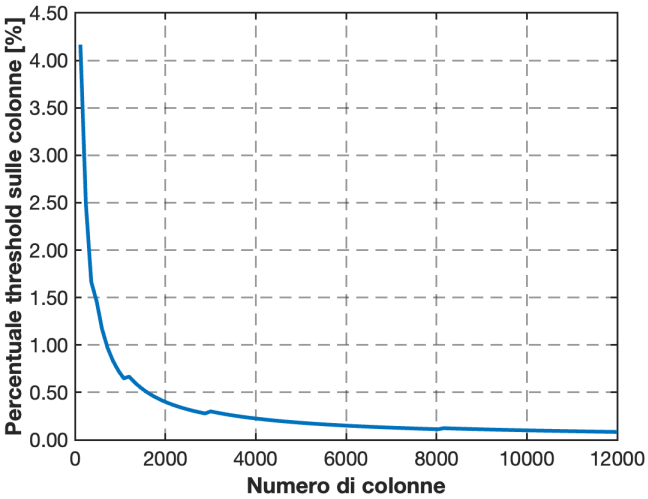

Figure 4.24: *Threshold* $t = \log n$ as percentage of column count of the test.

	Time vs Error	**Time vs *Threshold***	**Error vs *Threshold***
Correlation	−0.7256	0.7601	−0.4619

Table 4.7: Linear correlations between execution time, relative error, and *threshold*.

	Avg Time [s]	**Max Time [s]**		**Min Time [s]**	
		Value	**Cols**	**Value**	**Cols**
TSVD	88.5171	284.2675	12,000	0.0841	120
RSVD	0.1777	0.7556	11,760	0.0048	240

Table 4.8: Execution times of TSVD and RSVD, with corresponding column counts.

	Avg Error	**Max Error**		**Min Error**	
		Value	**Cols**	**Value**	**Cols**
TSVD	0.1803	0.2345	1800	0.1075	120
RSVD	0.2520	0.3333	1920	0.1838	120

Table 4.9: Relative errors of TSVD and RSVD, with corresponding column counts.

	Mean	**Max**		**Min**	
		Value	**Cols**	**Value**	**Cols**
Time Difference $[s]$	88.105	297.206	12,000	0.056	120
Error Difference	0.0717	0.1201	720	0.0482	10,320

Table 4.10: Differences in execution times and errors of TSVD and RSVD, with corresponding column counts.

	Avg	**Max**		**Min**	
		Value	**Cols**	**Value**	**Cols**
Threshold	9	10	8160	5	120

Table 4.11: *Threshold* of TSVD and RSVD, with corresponding column counts.

5

Conclusions

The randomized algorithm for Singular Value Decomposition (RSVD) has proven to be an extremely effective solution in reducing computational limits of classical SVD, offering a fast and flexible method for large-scale analysis. Thanks to the use of random projections, the algorithm allows preserving, with high probability, essential information of the matrix, allowing significant dimensional reduction without critically compromising result accuracy. This makes it particularly suitable for application areas where balance between speed and precision is fundamental.

One of the most important aspects of the randomized approach is its scalability. In scenarios where data grows exponentially, such as big data, image processing, or recommendation systems, RSVD stands as an indispensable tool. The reduction of computational complexity from $\mathcal{O}(mn^2)$ to $\mathcal{O}(mnt)$, where $t \ll n$, represents a turning point for analyzing matrix dimensions not manageable with traditional methods. Added to this is the possibility of modulating parameter t to customize the algorithm according to specific needs, guaranteeing precise control over the compromise between calculation time and approximation error.

Experimental results, summarized in Table 5.1, confirmed the method's effectiveness, showing how different thresholds influence algorithm performance. Adoption of a truncation threshold, like the one proposed by M. Gavish and D. L. Donoho [6], allowed obtaining a reduction in execution time of over two orders of magnitude compared to deterministic SVD, maintaining moderate approximation error. Simultaneously, the use of logarithmic thresholds, while entailing an increase in relative errors, allowed further accelerating calculation times.

A further advantage of RSVD is its implementation simplicity and compatibility with modern computational architectures. Thanks to the random nature of projections, the method lends itself to be combined with compression and machine learning techniques, opening interesting perspectives for research and innovation.

Application implications of this approach are manifold. In machine learning, RSVD can be used to reduce dimensionality of input data, improving speed of algorithms like PCA, regression, or *clustering*. In the field of natural language processing, it can accelerate decomposition of *embedding* matrices, fundamental for techniques like dimensionality reduction in *word embedding*. In scientific and engineering fields, RSVD allows rapidly analyzing complex systems, such as transfer matrices, solving problems that would otherwise require prohibitive computational resources.

Algorithm	**Threshold** t	**Time** $[s]$	**Speedup**	**Error**	**Discrepancy**
TSVD	$\mathcal{O}(\sqrt{n})$	88.870,873	1x	0.058,845	1 x
TSVD	$\lceil \log n \rceil$	88.5171	1x	0.1803	3 x
RSVD	$\mathcal{O}(\sqrt{n})$	0.765,446	116x	0.094,308	1.6x
RSVD	$\lceil \log n \rceil$	0.1777	500x	0.2520	4.3x

Table 5.1: Comparison of execution times, relative errors, and speedup for different thresholds in the randomized algorithm for SVD. Speedup and discrepancy refer to TSVD with optimal *threshold*. [6]

Another relevant theme is RSVD's statistical robustness. Random projections, while intrinsically approximate, rely on probabilistic principles guaranteeing with high probability the maintenance of main matrix characteristics. This paves the way for future studies to further improve approximation quality, for example by developing hybrid techniques combining *random sampling* with deterministic approaches in critical processing phases.

Finally, from a theoretical point of view, results of this thesis confirm the importance of threshold t in designing randomized algorithms and suggest that further deepening may lead to an even deeper understanding of optimal thresholds for specific application contexts. In this sense, RSVD not only represents a practical tool but also constitutes a basis for theoretical investigations on behavior of probabilistic algorithms for dimensional reduction.

In perspective, growing adoption of randomized methods in computational science promises to revolutionize the way we face complex problems. RSVD, with its combination of computational efficiency, flexibility, and robustness, is destined to play a central role in development of future technologies for data analysis, machine learning, and solving scientific and engineering problems on a large scale.

A

Matlab Code

In this chapter, we report the Matlab code used to perform the tests, save data, and generate graphs.

A.1 Optimal *Threshold* Calculation

In this thesis, to calculate the optimal *threshold*, we approximated by excess the values returned by the expression $\lambda(\beta)\sqrt{n}\gamma$ using the function `ceil`. In line 14, it is possible to replace the function `ceil` with `floor` if one wishes to approximate by defect, or with `round` if one prefers to round to the nearest integer.

```
1  % Function to calculate target rank.
2  % Input:
3  %     numRows > 0,
4  %     numCols > 0,
5  %     gamma: magnitude of white noise, gamma > 0.
6  % Output:
7  %     t: threshold, t <= n.
8  function t = optimalThreshold(numRows, numCols, gamma)
9      % Threshold is not defined for degenerate matrices.
10     assert(all(numRows>0));
11     assert(all(numCols>0));
12     beta = min(numRows/numCols,numCols/numRows);
13     lambda = sqrt(2*(beta+1)+8*beta/(beta+1+sqrt(beta^2+14*beta+1)));
14     t = min([ceil(lambda * sqrt(numCols) * gamma),numCols]);
15 end
```

A.2 Randomized SVD

The code below works only for tall and skinny matrices ($m \geq n$). The algorithm can be easily modified to adapt it to the dual case ($n \geq m$), but in this thesis, we do not deal with it.

```
% Function for randomized calculation of svd.
% Input:
%     X: matrix m x n,
%     t: threshold, t <= n <= m.
% Output:
%     U_approx: matrix m x t,
%     S: matrix t x t,
%     U_approx: matrix t x n.
function [U_approx,S,V] = rsvd(X,t)
% 0. Input check.
[m,n] = size(X); % Dimensions of X.
assert((t <= n) && (n <= m));

% 1. Z Projection
P = randn(n,t); % Random projection matrix.
Z = X*P; % Projection of X onto space of P.

% 2. QR Decomposition of Z.
[Q,R] = qr(Z,0);

% 3. Orthogonal projection Y.
Y = Q'*X;

% 4. SVD of Y.
[U,S,V] = svd(Y,'econ');

% 5. Approximate reconstruction of modes of U.
U_approx = Q*UY;
end
```

A.3 Single Test on an Image

In the following code, the optimal *threshold* suggested by M. Gavish and D. L. Donoho [6], calculated with the previous function, was used. To use another *threshold* (such as the logarithmic one), modify line 19 accordingly.

```
% Test for calculation of TSVD and RSVD of an image.
% Shows graph of singular values.
% Shows original image, after TSVD and after RSVD.
% Shows threshold used and its ratio with respect to columns.
% Shows time and error of TSVD and RSVD.
% Saves graphs and data listed above.
```

```
clear all; close all; clc;

% General experiment parameters.
name = 'cat'; % Experiment name.
imgName = 'cat';
imgExtension = '.jpg';
A = imread(['Sample Images/',imgName,imgExtension]);
X = double(rgb2gray(A)); % Matrix associated with image.
[m, n] = size(X); % Matrix dimensions.

%% Threshold calculation.
gamma = 1; % White noise magnitude.
t = optimalThreshold(m,n,gamma);  % Target rank.

%% Truncated SVD.
tic; % Start measuring time.
[U, S, V] = svd(X, 'econ');
timeTSVD = toc;  % Save time taken.
% Reconstruction from truncated SVD: mins serve to handle small
    matrices.
XTSVD = U(:, 1:min(t, size(U, 2))) * S(1:min(t, size(S, 1)), 1:min(t,
    size(S, 2))) * V(:, 1:min(t, size(V, 2)))';
errTSVD = norm(X - XTSVD, 2) / norm(X, 2); % Relative error of
    truncated SVD.

%% Randomized SVD.
tic; % Start measuring time.
[rU, rS, rV] = rsvd(X, t); % Randomized SVD.
timeRSVD = toc;  % Save time taken.
XRSVD = rU * rS * rV'; % Reconstruction from randomized SVD.
errRSVD = norm(X - XRSVD, 2) / norm(X, 2); % Relative error of
    randomized SVD.

%% Image visualization.

% Original image.
figure(1);
imshow(uint8(X)); % Show original image.
exportgraphics(gcf, ['Images/', name, '_SVD.png']);

% Image after truncated SVD.
```

```
44 figure(2);
45 imshow(uint8(XTSVD)); % Show image approximated with truncated SVD.
46 exportgraphics(gcf, ['Images/', name, '_TSVD.png']);
47
48 % Image after randomized SVD.
49 figure(3);
50 imshow(uint8(XRSVD)); % Show image approximated with randomized SVD.
51 exportgraphics(gcf, ['Images/', name, '_RSVD.png']);
52
53 %% Visualization of singular value graph (percentage).
54
55 % Sum of singular values in percentage.
56 singularValues = diag(S); % Extract singular values from matrix S.
57 totalSingularValueSum = sum(singularValues); % Total sum of singular
       values.
58
59 figure(4); clf;  % Create new figure and clear previous content.
60
61 x = linspace(1, min(m,n), min(m,n));
62 y = singularValues / totalSingularValueSum;
63 plot(x, y, 'o', 'MarkerFaceColor', 'b', 'LineWidth', 1, 'MarkerSize',
       10);
64
65 % Format graph.
66 formatGraph('Singular Value Index', 'Singular Value [%]');
67
68 % Adjust axes.
69 xlim([1, min(m,n)]);
70 set(gca, 'YScale', 'log'); % Set y-axis to logarithmic scale.
71
72 % Save figure.
73 exportgraphics(gcf, ['Images/', name, '_singular_values.png']);
74
75 %% Output of used threshold, sum of first t singular values, errors and
       execution times.
76
77 % Show used threshold.
78 sumFirstTSingularValues = sum(singularValues(1:t)); % Sum of first t
       singular values
79 percentageFirstTSingularValues = (sumFirstTSingularValues /
       totalSingularValueSum) * 100;
```

```
80 disp(['Threshold: ', num2str(t)]);
81
82 % Print percentage.
83 disp(['Sum of first t singular values in percentage: ', num2str(
       percentageFirstTSingularValues), '%']);
84
85 % Show execution times.
86 disp(['Truncated SVD Time: ', num2str(timeTSVD)]);
87 disp(['Randomized SVD Time: ', num2str(timeRSVD)]);
88
89 % Show relative errors.
90 disp(['Truncated SVD Error: ', num2str(errTSVD)]);
91 disp(['Randomized SVD Error: ', num2str(errRSVD)]);
92
93 %% Save data.
94 save(['Data/', name, '_test.mat']);
```

A.4 Iterative Test on Image Columns

Also in the following code, the optimal *threshold* suggested by M. Gavish and D. L. Donoho [6] was used. To use another *threshold* (such as the logarithmic one), modify line 114 accordingly.

```
1 % Test for calculation of TSVD and RSVD of an image varying columns.
2 % Shows the following graphs:
3 %     times, errors, threshold, ratio of times,
4 %     time differences between TSVD and RSVD,
5 %     error differences between TSVD and RSVD,
6 %     percentage truncated sum and total sum of singular values,
7 %     ratio between threshold and columns.
8 % Prints the following results:
9 %     correlations,
10 %     average, maximum, minimum time difference between TSVD and RSVD,
11 %     average, maximum, minimum error difference between TSVD and RSVD,
12 %     average, maximum, minimum times of TSVD and RSVD,
13 %     average, maximum, minimum errors of TSVD and RSVD,
14 %     average, maximum and minimum threshold.
15 % Saves graphs and data listed above.
16 clear all, close all, clc
17
18 % General experiment parameters.
19 name = 'nebula'; % Experiment name.
20 imgName = 'nebula';
```

```
imgExtension = '.jpg';
A = imread(['Sample Images/',imgName,imgExtension]); % Demo image.
X = double(rgb2gray(A)); % Matrix associated with image.
[m,n] = size(X); % Image dimensions.
gamma = 1; % White noise magnitude.

% Customizable steps.
step = 120;  % Increment for columns.

%% Function to calculate Pearson correlation between two vectors.
function r = pearsonCorr(x, y)
    assert(length(x) == length(y), 'Vectors must have the same length.');

    % Calculate mean of vectors.
    meanX = mean(x);
    meanY = mean(y);

    % Calculate numerator and denominator of Pearson correlation.
    numerator = sum((x - meanX) .* (y - meanY));
    denominator = sqrt(sum((x - meanX).^2) * sum((y - meanY).^2));

    % Correlation.
    r = numerator / denominator;
end

%% Function to execute truncated SVD and calculate time and error.
function [timeTSVD, errorTSVD] = execTSVD(XDynamic, t)
    tic;
    [U, S, V] = svd(XDynamic, 'econ');
    timeTSVD = toc;
    % Reconstruction from truncated SVD: mins serve to handle small matrices.
    XTSVD = U(:, 1:min(t, size(U, 2))) * S(1:min(t, size(S, 1)), 1:min(t, size(S, 2))) * V(:, 1:min(t, size(V, 2)))';
    errorTSVD = norm(XDynamic - XTSVD, 'fro') / norm(XDynamic, 'fro');
end

%% Function to execute randomized SVD and calculate time and error.
function [timeRSVD, errorRSVD] = execRSVD(XDynamic, t, p)
    tic;
```

```
59      [rU, rS, rV] = rsvd(XDynamic, t);
60      timeRSVD = toc;
61      XRSVD = rU * rS * rV';
62      errorRSVD = norm(XDynamic - XRSVD, 'fro') / norm(XDynamic, 'fro');
63  end
64
65  %% Function to plot results and save them.
66  function plotRisultati(xValues, sumsRatio, timeTSVD, timeRSVD,
        errorTSVD, errorRSVD, thresholds, name)
67      xlabelText = 'Number of columns';
68
69      % Plot execution time
70      figure;
71      plot(xValues, [timeTSVD, timeRSVD], 'LineWidth', 3);
72      legend('TSVD', 'RSVD', 'FontSize', 10);
73      formatGraph(xlabelText, "Execution Time [s]");
74      exportgraphics(gcf, ['Images/', name, '_', '_times.png']);
75
76      % Plot relative error
77      figure;
78      plot(xValues, [errorTSVD, errorRSVD], 'LineWidth', 3);
79      legend('TSVD', 'RSVD', 'FontSize', 10);
80      formatGraph(xlabelText, "Relative Error");
81      exportgraphics(gcf, ['Images/', name, '_', '_errors.png']);
82
83      % Plot threshold
84      figure;
85      plot(xValues, thresholds, 'LineWidth', 3);
86      formatGraph(xlabelText, "Threshold");
87      exportgraphics(gcf, ['Images/', name, '_', '_thresholds.png']);
88
89      % Plot truncated sum / total sum
90      figure;
91      plot(xValues, sumsRatio * 100, 'LineWidth', 3);
92      formatGraph(xlabelText, 'Truncated Sum / Total Sum [%]');
93      exportgraphics(gcf, ['Images/', name, '_', '
           singular_values_sums_perc.png']);
94
95      % Time ratio.
96      figure;
97      plot(xValues, timesTSVD./timesRSVD, 'LineWidth', 3);
```

```
98      formatGraph('Number of columns',"Ratio between times")
99      exportgraphics(gcf, ['Images/', name, '_time_ratio.png']);
100 end
101
102 %% Test: Keep number of rows fixed and vary number of columns.
103 numTests = ceil(n / step);
104 timesTSVD = zeros(numTests, 1);
105 timesRSVD = zeros(numTests, 1);
106 errorsTSVD = zeros(numTests, 1);
107 errorsRSVD = zeros(numTests, 1);
108 thresholds = zeros(numTests, 1);
109 sumsRatio = zeros(numTests, 1);
110 singularValues = cell(numTests, 1); % To save singular values.
111
112 index = 1;
113 for numCols = step:step:n
114     t = optimalThreshold(m, numCols, gamma);
115     thresholds(index) = t;
116     XDynamic = X(:, 1:numCols);
117
118     % Calculation of truncated and randomized SVD.
119     [timesTSVD(index), errorsTSVD(index)] = execTSVD(XDynamic, t);
120     [timesRSVD(index), errorsRSVD(index)] = execRSVD(XDynamic, t, gamma
        );
121
122     % Calculation of singular values.
123     s = svd(XDynamic, 'econ');
124     singularValues{index} = s;
125
126     % Calculation of ratio truncated/total singular values sum.
127     sumsRatio(index) = sum(s(1:t)) / sum(s);
128
129     index = index + 1;
130 end
131
132 xValues = step:step:n;
133
134 %% Plot of results
135 plotRisultati(xValues, sumsRatio, timesTSVD, timesRSVD, errorsTSVD,
    errorsRSVD, thresholds, name);
136
```

```
137 %% Percentage difference in execution time between TSVD and RSVD.
138 deltaTimes = timesTSVD - timesRSVD;
139 deltaPercTimes = deltaTimes ./ timeTSVDs * 100;
140 figure;
141 plot(xValues, deltaPercTimes, 'LineWidth', 3);
142 formatGraph('Number of columns','Percentage difference in time [%]')
143 exportgraphics(gcf, ['Images/', name, '_delta_times.png']);
144
145 %% Percentage difference in error between RSVD and TSVD.
146 deltaErrors = errorsRSVD - errorsTSVD;
147 deltaPercErrors = deltaErrors ./ errorsRSVD * 100;
148 figure;
149 plot(xValues, deltaPercErrors, 'LineWidth', 3);
150 formatGraph('Number of columns',"Percentage difference in error [%]")
151 exportgraphics(gcf, ['Images/', name, '_delta_errors.png']);
152
153 %% Percentage threshold on number of columns.
154 percThresholds = thresholds ./ xValues' * 100;
155 figure;
156 plot(xValues, percThresholds, 'LineWidth', 3);
157 formatGraph('Number of columns','Threshold percentage on columns [%]')
158 exportgraphics(gcf, ['Images/', name, 'thresholds_perc.png']);
159
160 %% Calculation of correlations.
161 corrTimesErrors = pearsonCorr(timesTSVD, errorsTSVD);
162 corrTimesThresholds = pearsonCorr(timesTSVD, thresholds);
163 corrErrorsThresholds = pearsonCorr(errorsTSVD, thresholds);
164 fprintf('Correlation between time and error: %f.\n', corrTimesErrors);
165 fprintf('Correlation between time and threshold: %f.\n',
      corrTimesThresholds);
166 fprintf('Correlation between error and threshold: %f.\n',
      corrErrorsThresholds);
167
168 %% Average, minimum and maximum difference of times between TSVD and
      RSVD.
169 meanDeltaTime = mean(deltaTimes);
170 [maxDeltaTime, idxMaxTime] = max(deltaTimes);
171 [minDeltaTime, idxMinTime] = min(deltaTimes);
172 colMaxTime = xValues(idxMaxTime);
173 colMinTime = xValues(idxMinTime);
174 fprintf('Average time difference: %f seconds\n', meanDeltaTime);
```

```
175 fprintf('Maximum time difference: %f seconds, corresponding to %d
    columns.\n', maxDeltaTime, colMaxTime);
176 fprintf('Minimum time difference: %f seconds, corresponding to %d
    columns.\n', minDeltaTime, colMinTime);
177
178 %% Average, minimum and maximum difference of errors between TSVD and
    RSVD.
179 meanDeltaError = mean(deltaErrors);
180 [maxDeltaError, idxMaxError] = max(deltaErrors);
181 [minDeltaError, idxMinError] = min(deltaErrors);
182 colMaxError = xValues(idxMaxError);
183 colMinError = xValues(idxMinError);
184 fprintf('Average error difference: %f\n', meanDeltaError);
185 fprintf('Maximum error difference: %f, corresponding to %d columns.\n',
     maxDeltaError, colMaxError);
186 fprintf('Minimum error difference: %f, corresponding to %d columns.\n',
     minDeltaErrors, colMinError);
187
188 %% Average, minimum and maximum times.
189 meanTimeTSVD = mean(timesTSVD);
190 meanTimeRSVD = mean(timesRSVD);
191 [maxTimeTSVD, idxMaxTimeTSVD] = max(timesTSVD);
192 [maxTimeRSVD, idxMaxTimeRSVD] = max(timesRSVD);
193 colMaxTimeTSVD = xValues(idxMaxTimeTSVD);
194 colMaxTimeRSVD = xValues(idxMaxTimeRSVD);
195 [minTimeTSVD, idxMinTimeTSVD] = min(timesTSVD);
196 [minTimeRSVD, idxMinTimeRSVD] = min(timesRSVD);
197 colMinTimeTSVD = xValues(idxMinTimeTSVD);
198 colMinTimeRSVD = xValues(idxMinTimeRSVD);
199 fprintf('Average times: TSVD: %f seconds; RSVD: %f seconds\n',
    meanTimeTSVD, meanTimeRSVD);
200 fprintf('Maximum time: TSVD: %f seconds, corresponding to %d columns;
    RSVD: %f seconds, corresponding to %d columns.\n', maxTimeTSVD,
    colMaxTimeTSVD, maxTimeRSVD, colMaxTimeRSVD);
201 fprintf('Minimum time: TSVD: %f seconds, corresponding to %d columns;
    RSVD: %f seconds, corresponding to %d columns.\n', minTimeTSVD,
    colMinTimeTSVD, minTimeRSVD, colMinTimeRSVD);
202
203 %% Average, minimum and maximum errors.
204 meanErrorTSVD = mean(errorsTSVD);
205 meanErrorRSVD = mean(errorsRSVD);
```

```
206 [maxErrorTSVD, idxMaxErrorTSVD] = max(errorsTSVD);
207 [maxErrorRSVD, idxMaxErrorRSVD] = max(errorsRSVD);
208 colMaxErrorTSVD = xValues(idxMaxErrorTSVD);
209 colMaxErrorRSVD = xValues(idxMaxErrorRSVD);
210 [minErrorTSVD, idxMinErrorTSVD] = min(errorsTSVD);
211 [minErrorRSVD, idxMinErrorRSVD] = min(errorsRSVD);
212 colMinErrorTSVD = xValues(idxMinErrorTSVD);
213 colMinErrorRSVD = xValues(idxMinErrorRSVD);
214 fprintf('Average errors: TSVD: %f; RSVD: %f\n', meanErrorTSVD,
      meanErrorRSVD);
215 fprintf('Maximum error: TSVD: %f, corresponding to %d columns; RSVD: %f
      , corresponding to %d columns.\n', maxErrorTSVD, colMaxErrorTSVD,
      maxErrorRSVD, colMaxErrorRSVD);
216 fprintf('Minimum error: TSVD: %f, corresponding to %d columns; RSVD: %f
      , corresponding to %d columns.\n', minErrorTSVD, colMinErrorTSVD,
      minErrorRSVD, colMinErrorRSVD);
217
218 %% Average, minimum and maximum Threshold.
219 meanThreshold = mean(thresholds);
220 [maxThreshold, idxMaxThreshold] = max(thresholds);
221 colMaxThreshold = xValues(idxMaxThreshold);
222 [minThreshold, idxMinThreshold] = min(thresholds);
223 colMinThreshold = xValues(idxMinThreshold);
224 fprintf('Average Threshold: %f.\n', meanThreshold);
225 fprintf('Maximum Threshold: %f, corresponding to %d columns.\n',
      maxThreshold, colMaxThreshold);
226 fprintf('Minimum Threshold: %f, corresponding to %d columns.\n',
      minThreshold, colMinThreshold);
227
228 %% Save data.
229 save(['Data/', name, '.mat']);
```

A.5 Graph Formatting

```
1 % Function to consistently format graphs.
2 % Input:
3 %     xLabel: x-axis title (string);
4 %     yLabel: y-axis title (string).
5 function formatGraph(xLabel, yLabel)
6     % Name axes.
7     xlabel(xLabel, 'FontSize', 18, 'FontWeight', 'bold');
```

```
8       ylabel(yLabel, 'FontSize', 18, 'FontWeight', 'bold');
9
10      % Set axis characteristics.
11      set(gca, 'FontSize', 16, 'LineWidth', 1.5, 'Box', 'on', '
            GridLineStyle', '--');
12
13      % Activate grid.
14      grid on;
15      ax = gca;
16      ax.XGrid = 'on';
17      ax.YGrid = 'on';
18      ax.GridAlpha = 0.4;
19      ax.GridColor = [0, 0, 0];
20
21      % Get current limits for y-axis.
22      yLimits = get(gca, 'YLim');
23
24      % Check maximum limit of y-axis.
25      if yLimits(2) <= 10
26          % Set y-axis label format.
27          ytickformat('%.2f');  % Decimal format with two places.
28      end
29  end
```

Bibliography

[1] Steven L. Brunton and J. Nathan Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, USA, 1st edition, 2019.

[2] Alan Cline and Inderjit Dhillon. *Computation of the Singular Value Decomposition*, pages 1027–1039. Chapman and Hall/CRC, 12 2013.

[3] Petros Drineas and Michael W. Mahoney. Randnla: randomized numerical linear algebra. *Commun. ACM*, 59(6):80–90, May 2016.

[4] Petros Drineas and Michael W. Mahoney. Lectures on randomized numerical linear algebra. *CoRR*, abs/1712.08880, 2017.

[5] M. Gavish and D. L. Donoho. Code supplement to "the optimal hard threshold for singular values is $4/\sqrt{3}$", 2014. `https://purl.stanford.edu/vg705qn9070`.

[6] Matan Gavish and David L. Donoho. The optimal hard threshold for singular values is $4/\sqrt{3}$. *IEEE Transactions on Information Theory*, 60(8):5040–5053, Aug 2014.

[7] Mutual Information. Is the future of linear algebra.. random?, 2024. `https://www.youtube.com/watch?v=6htbyY3rH1w&t=1134s&ab_channel=MutualInformation`.