Project 2: SVD applications
NLA

Clàudia Valverde

December 11, 2024

The goal of this project is to discuss three common applications of the SVD decomposition: LS problem, graphics compression and PCA.

The delivery is organized as follows:
The project is divided into three sections: (1) Least Squares Problem (LSP or LS problem), (2) Graphics Compression, and (3) PCA. The report follows the same structure, providing theoretical explanations for each part when necessary, commenting on relevant sections of the code, and primarily focusing on analyzing and presenting the results from the various SVD applications.
The delivery folder is also structured into three separate folders corresponding to the sections of the report.
I have included the input data, which can be read directly from the scripts without issues. Additionally, I have provided the outputs generated when running the code, which are the ones used in the report. Finally, I have included the notebooks to account for potential compatibility issues when executing the code.

Here is a very brief introduction of what SVD consist of: The Singular Value Decomposition (SVD) of a matrix $A \in \mathbb{R}^{m \times n}$ allows us to represent A as:

$$A = U\Sigma V^T$$

where:

- $U \in \mathbb{R}^{m \times m}$ is an orthogonal matrix of left singular vectors,
- $\Sigma \in \mathbb{R}^{m \times n}$ is a diagonal matrix of singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$ (r is the rank of A),
- $V \in \mathbb{R}^{n \times n}$ is an orthogonal matrix of right singular vectors.

1 Least Squares problem

> Write a program to solve the LS problem using SVD. Compute the LS solution for the datasets datafile and datafile2.csv that were used in pr4: QR factorization and least square problems. Compare the results using SVD with those obtained from the QR solution of the LS problems.

From the mathematical definition:
The least squares solution $\mathbf{x}$ is the one that minimizes

$$\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2.$$

Among all solutions, the minimum-norm solution is the one that minimizes

$$\|\mathbf{x}\|_2.$$

We are asked to solve the LS problem using SVD and QR factorization.

1.1 Solving LS problem with SVD factorization

Using the properties of the pseudo-inverse $\mathbf{A}^+$, the solution can be expressed as:

$$\mathbf{x} = \mathbf{A}^+\mathbf{b},$$

where:

$$\mathbf{A}^+ = \mathbf{V}\Sigma^{-1}\mathbf{U}^T$$

is the pseudo-inverse of $\mathbf{A}$. Here: Σ^{-1} is formed by taking the reciprocal of non-zero singular values of Σ, with zeroes elsewhere. But we have to be careful with the computation of Σ^{-1}, since it can have very mall singular values so to compute the inverse we first have to discard these singular values and then compute the inverse by adding 0 if we find a very small value. This approach ensures numerical stability while accurately computing the pseudo-inverse.

The term $\|\mathbf{Ax} - \mathbf{b}\|_2$ is minimized by construction, and the choice $\mathbf{x} = \mathbf{A}^+\mathbf{b}$ minimizes $\|\mathbf{x}\|_2$.

We have declared the function *least_squares_svd()* that computes the solution $\mathbf{x}_{\text{LS}}$ using SVD factorization taking into account very small values.

1.2 Solving LS problem with QR factorization

QR factorization is a matrix decomposition technique where a matrix $A \in \mathbb{R}^{m\times n}$ $(m \geq n)$ is expressed as the product of an orthogonal matrix Q and an upper triangular matrix R:

$$A = QR$$

In order to compute the QR factorization, we have to distinguish between two cases of the LS problem when the input matrix its the Full-rank or rank defficient.

1.2.1 Full-rank LS problem

For a matrix A with full rank ($\text{rank}(A) = n$), QR factorization is unique and provides an efficient way to decompose A into Q and R, where: - Q is an orthogonal matrix ($Q^TQ = I$). - R is an upper triangular matrix.

In order to use QR factorization in solving the LS problem we can consider the following linear system:

$$A\mathbf{x} = \mathbf{b} \quad \Longleftrightarrow \quad QR\mathbf{x} = \mathbf{b} \quad \Longleftrightarrow \quad R\mathbf{x} = Q^T\mathbf{b}$$

Let $Q^T\mathbf{b} = \begin{pmatrix}\mathbf{y}_1\\ \mathbf{y}_2\end{pmatrix} \in \mathbb{R}^m$, where $\mathbf{y}_1 \in \mathbb{R}^n$. Then, the system reduces to:

$$R\mathbf{x} = Q^T\mathbf{b} \quad \Longleftrightarrow \quad \begin{pmatrix}R_1\\ 0\end{pmatrix}\mathbf{x} = \begin{pmatrix}\mathbf{y}_1\\ \mathbf{y}_2\end{pmatrix}$$

To solve for $\mathbf{x}_{\text{LS}}$, one computes $R_1\mathbf{x} = \mathbf{y}_1$, where R_1 is the upper triangular submatrix of R. The residual error of the least squares solution is then:

$$\|\mathbf{r}\|_2^2 = \|\mathbf{y}_1 - R_1\mathbf{x}\|_2^2 + \|\mathbf{y}_2\|_2^2 = \|\mathbf{y}_2\|_2^2$$

If A is full-rank, the least-squares problem has unique solution.

1.2.2 Rank deficient LS problem

When $r = \text{rank}(A) < n$, there is generally no unique solution to the LS problem. To address this, an additional condition is typically imposed to obtain $\mathbf{x}_{\text{LS}}$, such as requiring it to be the minimum $\|\cdot\|_2$-norm solution.

To handle rank-deficiency, we can apply QR factorization with pivoting. The matrix A is decomposed as:

$$AP = QR, \quad Q \in \mathbb{R}^{m\times m} \text{ orthogonal}, \quad R \in \mathbb{R}^{m\times n} \text{ upper triangular},$$

where P is a pivoting matrix that reorders the columns of A, and R has the form:

$$R = \begin{pmatrix} R_1 & S \\ 0 & 0 \end{pmatrix},$$

with R_1 being a non-singular upper triangular matrix.
Since $A = QRP^T$, the LS problem becomes:

$$\|\mathbf{b} - A\mathbf{x}\|_2^2 = \|\mathbf{b} - Q \begin{pmatrix} R_1 & S \\ 0 & 0 \end{pmatrix} P^T\mathbf{x}\|_2^2.$$

Using the orthogonality of Q, this reduces to:

$$\|\mathbf{b} - A\mathbf{x}\|_2^2 = \|Q^T\mathbf{b} - \begin{pmatrix} R_1 & S \\ 0 & 0 \end{pmatrix} P^T\mathbf{x}\|_2^2.$$

Let $P^T\mathbf{x} = \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \end{pmatrix}$ and $Q^T\mathbf{b} = \begin{pmatrix} \mathbf{c} \\ \mathbf{d} \end{pmatrix}$, where $\mathbf{u}, \mathbf{c} \in \mathbb{R}^r$ and $\mathbf{v}, \mathbf{d} \in \mathbb{R}^{m-r}$. Then, the residual becomes:

$$\|\mathbf{b} - A\mathbf{x}\|_2^2 = \| \begin{pmatrix} \mathbf{c} - R_1\mathbf{u} + S\mathbf{v} \\ \mathbf{d} \end{pmatrix} \|_2^2 = \|\mathbf{c} - R_1\mathbf{u} + S\mathbf{v}\|_2^2 + \|\mathbf{d}\|_2^2.$$

The minimum residual error is given by:

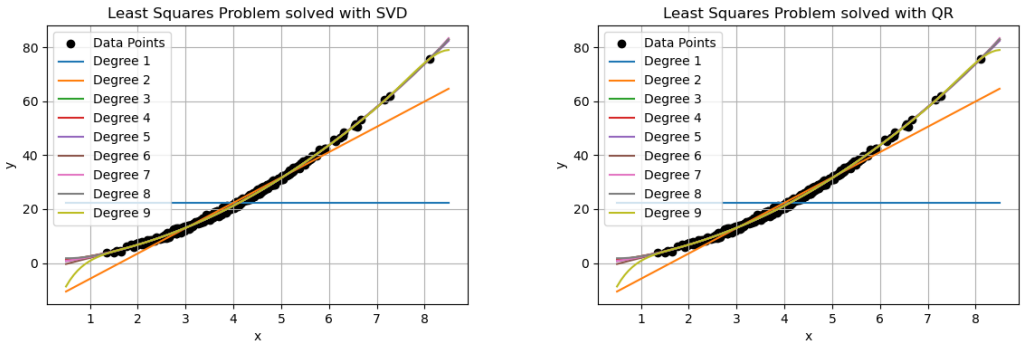
$$\min \|\mathbf{b} - A\mathbf{x}\|_2^2 = \|\mathbf{d}\|_2^2,$$

which is attained when $R_1\mathbf{u} + S\mathbf{v} = \mathbf{c}$.
Different solutions to this equation are possible:
1. **Minimal Solution:** Among all solutions $(\mathbf{u}, \mathbf{v})$, one chooses the one minimizing $\|(\mathbf{u}, \mathbf{v})\|_2$. This requires solving an optimization problem.
2. **Basic Solution:** Setting $\mathbf{v} = 0$, one solves $R_1\mathbf{u} = \mathbf{c}$ to obtain $\mathbf{u}$.
We have implemented the funciton *least_squares_qr()* that handles both full and defficient rank QR factorizations and for the rank-defficient it computes the minimal solution.

1.3 Results

Our input data (*dades.csv* and *data_regressio.csv*) consists of a full-rank and a rank deficient input matrices, the SVD naturally handles rank-deficient matrices since it ignores zero singular values when computing the pseudo-inverse but the QR factorization does not. For this reason the function *least_squares_qr()* that computes the solution $\mathbf{x}_{\text{LS}}$ using QR factorization handles both cases, full and deficient rank.

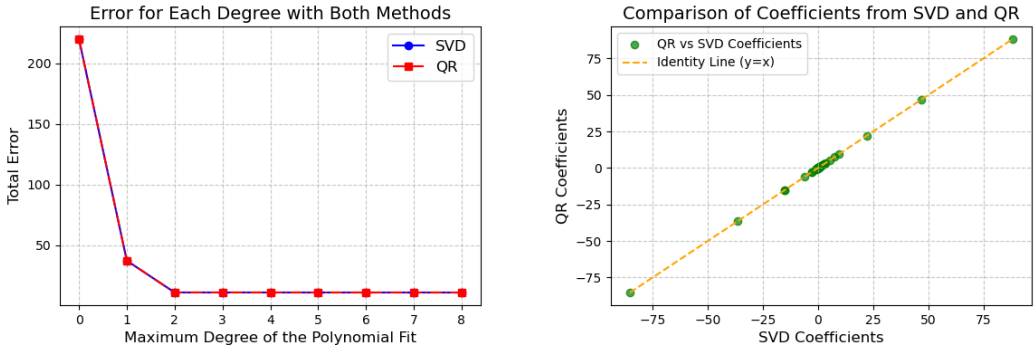
In order to load the data I have implemented two functions, one for each type of $rank(A)$ which construct the matrices A and b. For the full-rank case we can construct the input matrices based on the degree of the polynomial we are going to fit. This enables us to experiment with various polynomial degrees and visually assess the fitted results through plotted data points and the fitted lines.



(a) SVD (b) QR

Figure 1: Solution of the LS problem using full-rank matrices solved with SVD (left) and QR (right) with polynomial degree from 1-9.

When solving the LS problem for full-rank case, we tried explored solving the LS problem for polynomial degrees ranging from 1 to 9. The results in Figure 1 indicate that for both methods (SVD and QR) the best solution is achieved as we increase the degree of the polynomial. Although the best solution is given by the highest degree, looking at the plot, it seems that lower degrees provide a sufficiently accurate fit.



(a) Error (b) Identity

Figure 2: Full-rank polynomial fitting using both methods (SVD and QR) a) Error plot of the fitting, b) coefficients obtained in fitting

When analyzing the error for each degree using both SVD and QR methods (Figure 2), we observe that beyond degree 2, the errors from both methods converge. This suggests that a quadratic polynomial is adequate for fitting the data. Finally, comparing the coefficients obtained from SVD and QR methods reveals identical results, as they align perfectly along the identity line.

The data file *data_regressio.csv* is rank-deficient, which prevented us from plotting the polynomial fitting of the LS problem. However, we were still able to solve it. The obtained solution norm is approximately 4,774,736.29, with an error of about 1.15. When plotting the coefficients (Fig. 3), we observe that both the SVD and QR factorization methods show once again identical results.

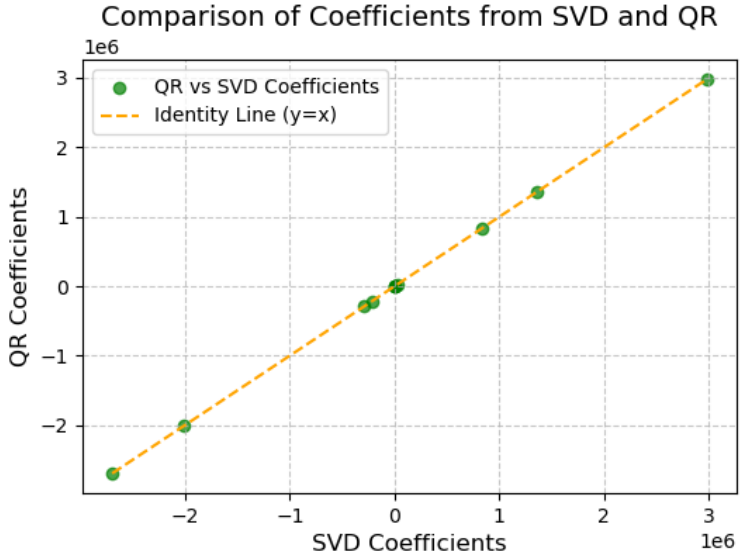


Figure 3: Coefficients obtained in polynomial fitting for rank defficient using both methods.

2 Graphics Compression

The SVD factorization has the property of giving the best low rank approximation matrix with respect to the Frobenius and/or the 2-norm to a given matrix. State properly the previous statement and write down the corresponding proofs for the Frobenius norm and the 2-norm.

As defined in the introduction of the report the Singular Value Decomposition (SVD) of a matrix $A \in \mathbb{R}^{m \times n}$ allows us to represent A as:

$$A = U\Sigma V^T$$

The concept of matrix compression arises from the idea of retaining only a subset of the singular values in *Sigma*, which reduces the rank of the matrix while attempting to preserve most of its original information. The resulting matrices, called low-rank approximations, provide a compressed representation of A. Importantly, it can be rigorously proven that these low-rank approximations are the best possible with respect to both the Frobenius norm and the 2-norm.

This foundational result is formalized in the **Eckart-Young-Mirsky** theorem, which guarantees that the low-rank matrices constructed using the SVD minimize the reconstruction error under these norms. The following sections detail the formal statement of this property and provide the corresponding proofs for the Frobenius norm and the 2-norm

The statement about the best low-rank approximation is as follows:
Statement: For any $k \leq \text{rank}(A)$, let

$$A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T,$$

where u_i and v_i are the i-th columns of U and V, respectively. Then:

1. A_k is the matrix of rank k that minimizes the Frobenius norm of the error:

$$\|A - B\|_F \quad \text{for all matrices } B \text{ of rank } k.$$

2. A_k is the matrix of rank k that minimizes the spectral (2-norm) of the error:

$$\|A - B\|_2 \quad \text{for all matrices } B \text{ of rank } k.$$

Proof for the Frobenius Norm:
The Frobenius norm of a matrix A is defined as:

$$\|A\|_F = \sqrt{\sum_{i,j} |A_{ij}|^2} = \sqrt{\text{trace}(A^T A)}.$$

Let B be any matrix of rank k with SVD $B = U_B \Sigma_B V_B^T$. Then:

$$\|A - B\|_F^2 = \text{trace}((A - B)^T (A - B)).$$

Using the Eckart-Young-Mirsky theorem, the best approximation in terms of the Frobenius norm occurs when B is the truncation A_k, which retains only the top k singular values of A. Explicitly:

$$A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T.$$

This truncation minimizes the Frobenius norm because the discarded singular values $\sigma_{k+1}, \ldots, \sigma_r$ contribute the least energy in terms of the Frobenius norm, as the singular values are squared in the Frobenius norm computation:

$$\|A - A_k\|_F^2 = \sum_{i=k+1}^{r} \sigma_i^2.$$

Any other rank-k matrix B would result in a higher Frobenius norm.

Proof for the Spectral Norm (2-Norm):
The spectral norm (2-norm) of a matrix A is defined as:

$$\|A\|_2 = \max_{\|x\|_2=1} \|Ax\|_2,$$

which corresponds to the largest singular value of A, i.e., $\|A\|_2 = \sigma_1$.

For any matrix B of rank k, the error matrix $A - B$ will have at least $r - k$ non-zero singular values, where $r = \text{rank}(A)$. The spectral norm of $A - B$ is minimized when the largest discarded singular value σ_{k+1} is the largest singular value of $A - B$.

Thus, the best rank-k approximation is achieved by truncating the smallest $r - k$ singular values, which corresponds to A_k. The error in the spectral norm is given by:

$$\|A - A_k\|_2 = \sigma_{k+1}.$$

Any other rank-k matrix B would result in a higher spectral norm, as the singular values discarded would not correspond to the smallest ones.

Conclusion: The SVD-based truncation A_k provides the optimal rank-k approximation of A with respect to both the Frobenius norm and the spectral norm.

Use the previous results to obtain a lossy compressed graphic image from a .jpeg graphic file. A .jpeg graphic file can be read as a matrix using the function `scipy.ndimage.imread()`. Use SVD decomposition to create approximations of lower rank to the image. Compare different approximations. The function `scipy.misc.imsave()` can be useful to save the approximated graphic files as .jpeg. The code must generate different compressed files for a given graphic file. Hence, to organize the output files, the name of the compressed file must reflect the percentage of the Frobenius norm captured in each compressed file. Use different .jpeg images (of different sizes and having letters or pictures) and compare results.

After having theoretically shown that SVD can be used to compress images we can apply it practically. Assuming we have done the SVD factorization of a matrix, $A = U\Sigma V^T$, where A is an image. The data stored in the matrices U, Σ, and V is ordered by how much it contributes to the matrix A in the product (contribution to reconstructing A). This property allows us to get quite good approximation by simply using the most significant components of the matrices, therefore achieving compression while preserving much of the image's essential information.

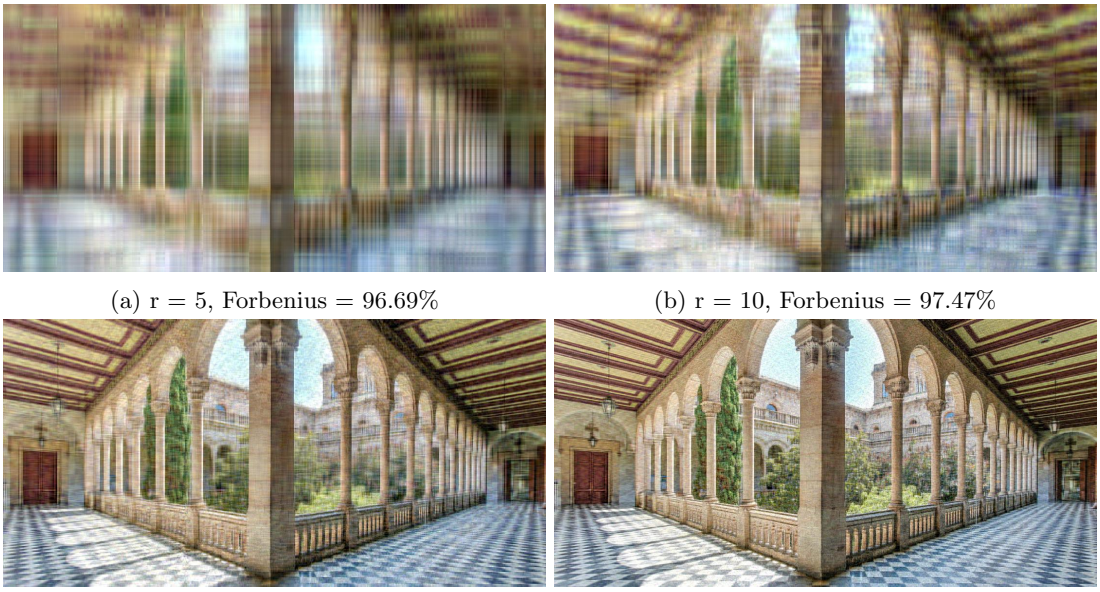
I have chosen 3 different images to experiment with shown in Figure 4. They consist of 2 color images - a picture of Universitat de Barcelona, and another one of a paradisiac beach - and one black-and-white image of the Zara logo. Lets see if when using a compressed version of these images, reconstructed using only the most significant values, we can still retain sufficient visual quality to be recognizable and informative of the images.

(a) Universitat de Barcelona (b) Paradisiac Beach (c) Zara logo

Figure 4: Input pictures to be compressed. The weights of the images are 341KB, 2.5MB, 29KB respectively.

In the function `svd_compressed_color()` we generate the compressed images based on *rank*, the value that indicates until when the U, Σ and V matrices should be truncated. The images were compressed by

processing all three color channels (RGB), even for the black-and-white image, as it also contains three channels. For the experiment, I selected four different ranks for the approximation: $rank = 5, 10, 50, 100$. We expect higher ranks yield better-quality approximations but require more data to encode the image. This trade-off between quality and compression efficiency is analyzed in the results.



(a) r = 5, Forbenius = 96.69%

(b) r = 10, Forbenius = 97.47%

(c) r = 50, Forbenius = 98.82%

(d) r = 100, Forbenius = 99.31%

Figure 5: Universitat de Barcelona compressed images with chosen rank (r) and Frobenius norm %

(a) r = 5, Forbenius = 98.31%

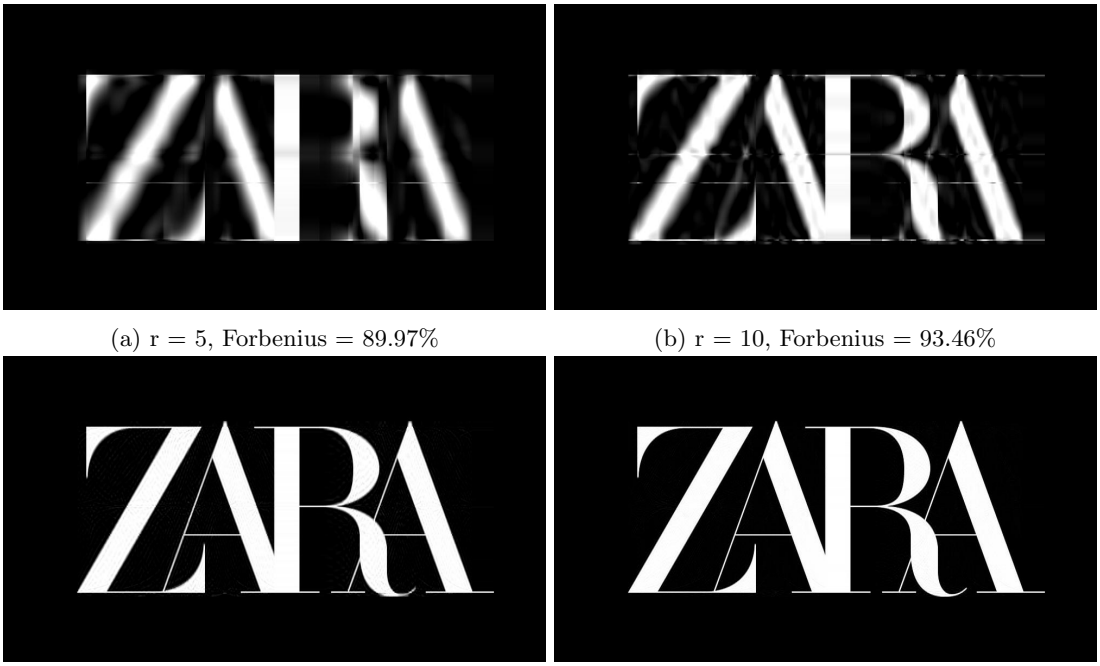
(b) r = 10, Forbenius = 98.79%

(c) r = 50, Forbenius = 99.41%

(d) r = 100, Forbenius = 99.58%

Figure 6: Paradisiac beach compressed images with chosen rank (r) and Frobenius norm %

See also Table 1 for clearer comparisons of the Frobenius norms.



(a) r = 5, Forbenius = 89.97%

(b) r = 10, Forbenius = 93.46%

(c) r = 50, Forbenius = 98.56%

(d) r = 100, Forbenius = 99.32%

Figure 7: Zara logo compressed images with chosen rank (r) and Frobenius norm %

Rank	**5**	**10**	**50**	**100**
Universitat de Barcelona	96.69	97.47	98.82	99.31
Paradisiac Beach	98.31	98.79	99.41	99.58
Zara logo	89.97	93.46	98.56	99.32

Table 1: Comparison of Frobenius norm (%) for the experimented input images and ranks

From the experiments, we observe that colorful images require higher ranks to be "reconstructed" correctly. For the Universitat de Barcelona image (Fig. 5), it is evident that higher ranks should have been tested, as it has not been fully reconstructed. Interestingly, while the paradisiac beach image (Fig. 6) has the highest Frobenius norm retained percentage, it is visually the least well-reconstructed.

For the Zara logo (Fig. 7), although visually it appears that a rank of 50 provides a satisfactory reconstruction, it has the lowest Frobenius norm retained percentage overall. This discrepancy can also be observed in the file sizes (Tab. 2) at ranks 50 and 100, the compressed versions of the Zara logo weigh even more than the original image.

Rank	**Original**	**5**	**10**	**50**	**100**
Universitat de Barcelona	341KB	68KB	82KB	125KB	146KB
Paradisiac Beach	2.5MB	653KB	761KB	1.2MB	1.4MB
Zara logo	29KB	25KB	28KB	34KB	30KB

Table 2: Comparison of the evolution of image weight for the experimented images and ranks

On this topic, looking at the image weights in Table 2, we can see that the compression successfully reduces file sizes. However, as the rank r increases, the file sizes also increase since retaining more singular values involves storing more data.

Finally, Figure 8 illustrates the evolution of the Frobenius norm retained percentage as the rank increases. While higher ranks were not tested due to time constraints, it is evident that higher ranks have better reconstructions but they also result in higher weights images (less compressed) and take longer to process.

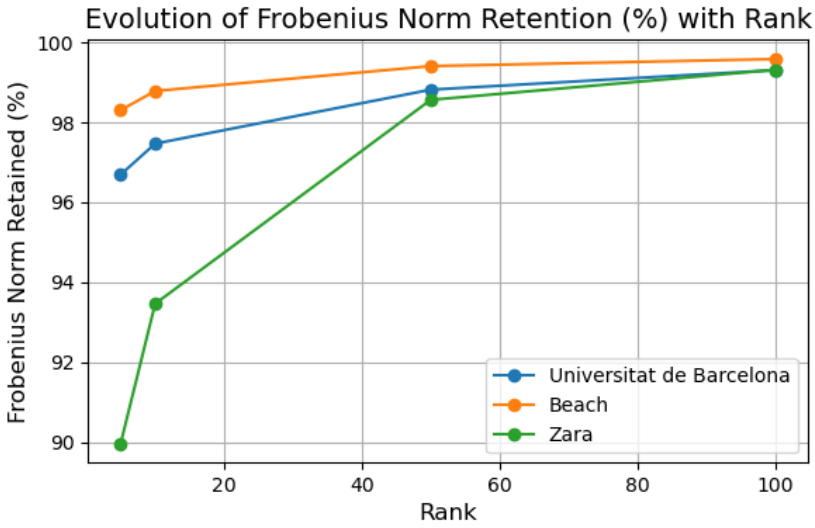


Figure 8: Frobenius norm % evolution

Overall Analysis:

- A higher rank (more singular values retained) preserves a higher percentage of the Frobenius norm, leading to better-quality reconstructions but requiring more storage and time.
- A lower rank sacrifices some of the image's information for greater compression, resulting in reduced quality but significantly smaller file sizes.
- The Frobenius norm retained percentage effectively demonstrates the trade-off between compression efficiency and image quality, providing a quantitative measure to evaluate the performance of SVD-based compression.

3 PCA

The file `example.dat` contains a dataset of 16 observations of 4 variables. Perform PCA analysis using both the covariance matrix and the correlation matrix. The code must write down the portion of the total variance accumulated in each of the principal components, the standard deviation of each of the principal components and the expression of the original dataset in the new PCA coordinates.

Principal Component Analysis (PCA) is a statistical technique used to reduce the dimensionality of a dataset while preserving as much variability as possible. It achieves this by transforming the original variables into a new set of orthogonal variables called principal components (PCs). Each principal component represents a direction of maximum variance in the data, and the components are ordered by the amount of variance they explain.

The mathematical foundation of PCA relies on finding the eigenvalues and eigenvectors of the covariance or correlation matrix of the data. These eigenvectors define the principal components, while the eigenvalues correspond to the amount of variance explained by each component. A key advantage of PCA is that it can be effectively computed using the Singular Value Decomposition (SVD).

To compute the principal components using SVD, one can start by noting that the covariance matrix of a dataset X (assuming the data is mean-centered) is given by:

$$C_X = \frac{1}{n-1} X^T X,$$

where n is the number of observations. Defining $Y = \frac{1}{\sqrt{n-1}} X^T$, it follows that:

$$C_X = Y^T Y.$$

By applying the SVD to Y, we can efficiently find the eigenvalues and eigenvectors of C_X. The reduced SVD factorization of Y is:

$$Y = USV^T,$$

where $U \in \mathbb{R}^{n \times r}$, $S \in \mathbb{R}^{r \times r}$, and $V \in \mathbb{R}^{r \times m}$, with $r = \text{rank}(Y)$.
Using the properties of the SVD factorization:

- The **singular values** s_i of Y satisfy $s_i = \sqrt{\lambda_i}$, where λ_i are the eigenvalues of C_X, ordered in decreasing magnitude. The proportion of variance explained by the i-th principal component is given by:
$$\frac{s_i^2}{\sum_{j=1}^{p} s_j^2}.$$
- The columns of the matrix V correspond to the **eigenvectors** of C_X (or $Y^T Y$), and thus represent the **principal components**. Consequently, the transformed dataset in PCA coordinates can be expressed as:
$$Z = V^T X.$$

If the correlation matrix is used instead of the covariance matrix, the data matrix X must first be standardized so that each variable has a mean of zero and a standard deviation of one. Let Z be the standardized data matrix, where:

$$Z_{ij} = \frac{X_{ij} - \mu_j}{\sigma_j},$$

where μ_j is the mean and σ_j is the standard deviation of the j-th variable. The correlation matrix C_R of Z is given by:

$$C_R = \frac{1}{n-1} Z^T Z.$$

Now, defining

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

we have:

$$C_R = Y^T Y.$$

And we can apply SVD to Y as explained above.

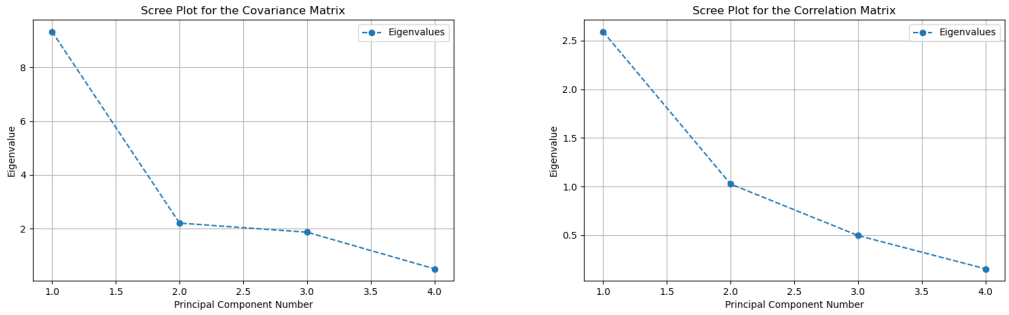
This demonstrates the unification of PCA and SVD, where the SVD not only facilitates the computation of the principal components but also provides a direct way to transform the data into its PCA representation.

The `pca.py` script handles both covariance as well as correlation matrices to compute the PCA using SVD. Moreover, we have made three more functions in order to implement some rules to discuss about the number of optimal principal components needed to explain the datasets:

1. Scree Plot: Is a line plot of the eigenvalues of a matrix. The rule dictates that the correct number of components is the number that appear prior to the elbow.
2. Kasier Rule: A component should be retained if it has eigenvalue greater than or equal to one.
3. Cumulative Variance Rule (e.g., 75% Rule): keep components until they reach 75% of total variance.

Metric	Covariance Matrix	Correlation Matrix
Total Variance		
PC1	0.66979347	0.60757275
PC2	0.15887735	0.24030055
PC3	0.13452316	0.11616759
PC4	0.03680602	0.03595911
Std. Deviation		
PC1	3.05353618	1.61006534
PC2	1.48718027	1.01256227
PC3	1.36845699	0.70402299
PC4	0.71580055	0.39169574

Table 3: PCA Results for Covariance and Correlation Matrices on `example.dat` file



(a) Covariance (b) Correlation

Figure 9: Scree plots for the Covariance and Correlation matrices on `example.dat` file

We have first applied it on the `example.dat` file for both types of matrices. In Table 3 are the results of the total variance and the standard deviation for both types of matrices. As it can be observed, the results for covariance and correlation matrices are different this can be due to the fact that the method for the correlation matrix has to be applied to data from the Gaussian distribution, which in our case is not.

The expression of the original dataset in the new PCA coordinates can be seen in the file `expression_pca_exampledat.csv`

When looking for the optimal number of components, 3/4 rule tells us that for the covariance matrix its 2 where as Kaiser says 3. Looking at the scree plot the elbow seems to be between PC 2 and 3 so the 3 methods more or less correlate. If we analyse the correlation matrix, we obtain that both 3/4 and Kaiser propose 2 PCs and the scree plot, although it is highly subjective, could also be interpreted as PC 2.

The file `RCsGoff.csv` contains data from the experiment reported in [3]. Each observation consists in measuring the amount of a total number of 58581 genes. There are a total of 20 observations grouped by day of observation. The code must perform a PCA analysis on the covariance matrix. The output file must contain rows with the following format

$$\text{Sample}, \text{PC1}, \text{PC2}, \ldots, \text{PC20}, \text{Variance}$$

where `Sample` corresponds to labels such as `day0 rep1`, `day18 rep3` (i.e., the individual observations), and `PCi` represents the coordinate of the observation in the space of the i-th principal component. Finally, `Variance` indicates the proportion of the total variance explained by each of the principal components.

The memory should include a discussion about the number of principal components needed to explain the data sets (using for example the Kaiser rule, Scree plot, the 3/4 of the total variance rule, or any other method; explain the main idea of the methods used).

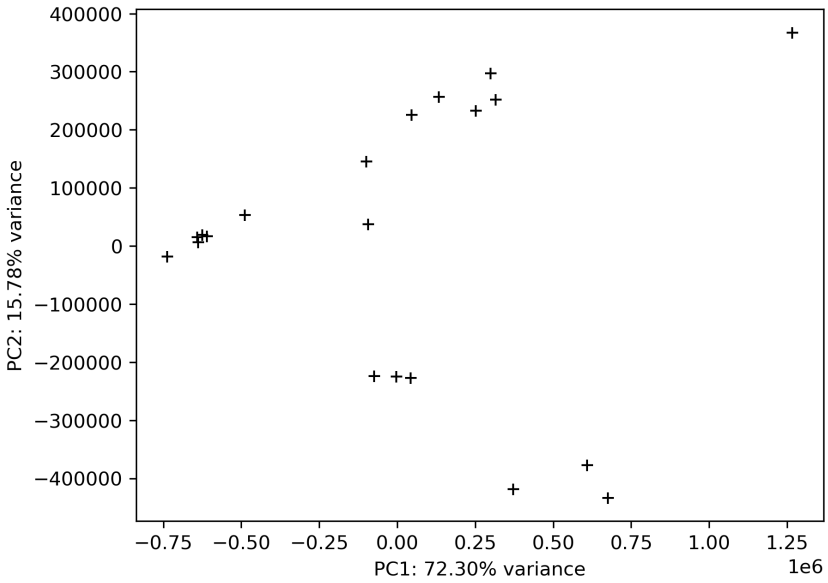


Figure 10: PC2 in terms of PC1 plot for comparison with the solution.

Finally, we perform the same analysis on the second dataset, `RCsGoff.csv`. This dataset consists of 20 observations (grouped by day of observation) of 58,581 variables (genes), resulting in a dataset of shape (58581, 20). From this, we obtain 20 principal components. The total variance and the expression of the original dataset in the new PCA coordinates are provided in the file `expression_pca.csv`, where each row corresponds to a sample (observation). We can also confirm that the PCA has worked correctly as when we plot the 1st and 2nd components the plot results in an exact copy to the one in the project formulation (Figure 10).

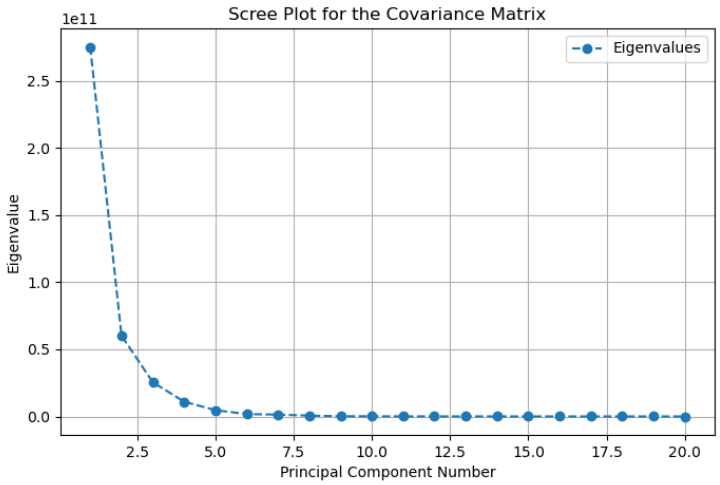


Figure 11: Scree plot for the RCsGoff dataset

Returning to the rules for determining the optimal number of principal components, the results are as follows: the $\frac{3}{4}$ rule suggests 2 components, Kaiser's criterion suggests 19, and the Scree plot (Figure 11) indicates 2 or 3 components, as it is difficult to pinpoint the exact location of the elbow.