This report corresponds to the evaluation of the results obtained in the second project of Numerical Linear Algebra, as well as, the solutions of some theoretical questions asked in the explanatory document of the project. We shall divide the report into three parts. The first part will be destined to the explorations of the results obtained in the **least squares problem**, the second part will correspond to obtain conclusions about the **graphics compression problem** and finally, we will study how **principal component analysis** works.

# 1 General framework, delivery

Before focusing on the proposed problems, I would like to comment on how the delivery has been made. The corrector has received a *.zip* file with this *pdf* file, corresponding to the evaluation of the obtained results, three python scripts, *Least_squares.py*, *Graphic_compr.py* and *PCA.py*, each one corresponding to each of the three proposed problems, three photos for use in the second problem with names *landscape.jpg*, *graduacio.jpg* and *vieiro.jpg*, and the datasets used to complete the project.

# 2 Least squares problem

This first problem consists of applying SVD and QR factorization to solve the least squares problem (LSP from now on) for two different datasets. Before explaining the methods that I have just mentioned to solve the LSP, I would like to explain how I load the datasets. Let us start with the first dataset, the datafile, called *dades*. In this dataset, we are given an amount of points, and we desire to compute the best polynomial fitting. Therefore, with the *load_dades* python function, I have built a matrix $A$ and a vector $b$, in order to be on the framework of the LSP. As seen in theory, the matrix $A$ and the vector $b$ are the following:

$$A = \begin{pmatrix} 1 & x_1 & \cdots & x_1^{d-1} \\ 1 & x_2 & \cdots & x_2^{d-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & \cdots & x_m^{d-1} \end{pmatrix} \qquad b = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix}$$

This is what the function *load_dades* returns for a given degree. The other dataset is easier, we are given directly the matrix $A$ and the vector $b$. Thus, once we have reached this point, we are ready to apply SVD and QR factorization to solve the LSP with any of the two datasets. Let us start explaining the method with the SVD factorization. As explained in the project, the solution of the LSP with minimum norm is given by $x = A^+b$ where $A^+$ is the pseudoinvers matrix of $A$. Therefore, I have built two python functions to solve this, called *LS_sol_svd* and *pseudo*. The *pseudo* python function computes the pseudoinvers of a given matrix by SVD factorization, concretely, if we have the factorization $A = U\Sigma V^T$, then $A^+ = V\Sigma^{-1}U^T$. In addition, I want to mention that we have to be careful on the computation of $\Sigma^{-1}$, since we can have really small singular values, hence, to compute the inverse, firstly, we have to discard this small singular values, and then, compute the inverse by adding 0 if we have found a small one. Finally, with the function *LS_sol_svd*, we compute the final desire solution, $x = A^+b$.

Let us continue with the QR factorization to solve the LSP. For this, we have implemented a python function called *LR_qr_sol* which computes the solution of the LSP by applying the QR factorization. For this purpose, we have to distinguish two cases. Assume that $A$ is a full rank matrix and we have the thin QR factorization of A, i.e., $A = Q_1R_1$ where $Q = (Q_1|Q_2)$ and $R = (R_1|0)^T$. Then, the solution of the LSP is the solution of the following triangular

system, $R_1 x = Q_1^T b$. Now, assume that A is rank deficient. We will not explain in detail this case, since is explained in a practical exercise, but the main idea is to perform QR factorization with pivoting, i.e.,

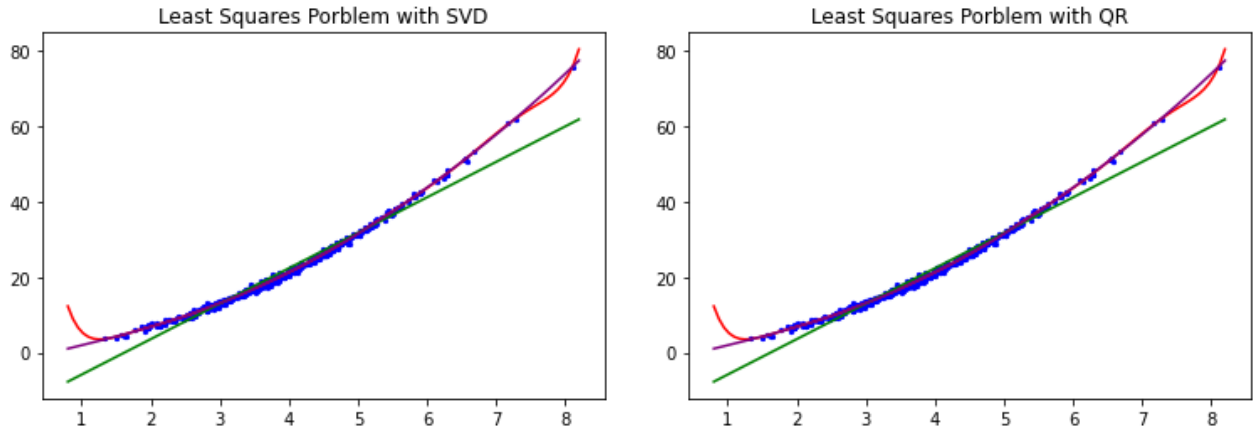
$$AP = QR, \;\; Q \text{ orthogonal}, \;\; R \text{ upper triangular}$$

with $P$ a pivoting matrix so that

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

with $R_1$ non-singular upper triangular matrix and $rank(R_1) = rank(A)$.

Finally, I would like to comment the main of this program. For the first dataset, I called *load_dades* function to compute the polynomial fitting for degrees 2 up to 11 with SVD and QR. As expected, the degree of the best polynomial fitting is the last one. Nevertheless, it is not necessary a polynomial of degree 11, see Figure 1 and notice that a polynomial of degree 6 is more than enough.[1] More precisely, the error obtained with the polynomial of degree 11 is 10.82 and with the polynomial of degree 6 is 10.86, i.e., we are not winning "too much".




(a) SVD (b) QR

Figure 1: Solution of the LSP using SVD or QR with polynomials of degree 2, 6 and 11 in green, purple and red respectively.

On the other hand, with the second dataset, we compute the solution using both methods, SVD and QR, and the results are approximately the same, a vector with solution norm 4774736.29 and the error is about 1.15.

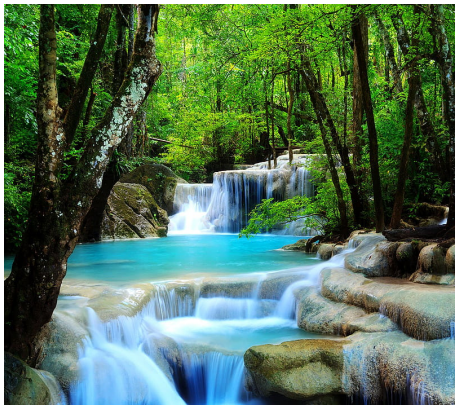
# 3 Graphics compression

Let us move on to a graphic compression problem. We have made a simple program with one python function called *compress* that use SVD factorization to compress an image. The intuitive idea of this is the following. Assume that we have done the SVD factorization of a matrix, $A = U\Sigma V^T$, in our case, the matrix $A$ will be the image. Then, the data in the matrices $U, \Sigma$ and $V$ is sorted by how much it contributes to the matrix A in the product, thus, this enables us to get quite a good approximation by simply using only the most important parts of the matrices. More precisely, we have to choose a number of singular values, $r$, to

[1] This is related to the concept of overfitting. A degree 11 polynomial will probably overfit the data. Nevertheless, for this particular problem, we are not worrying about this.

do our approximation. Evidently, the higher this number is, the better the quality of the approximation gets, but also the more data is needed to encode it.

In our code, we implement a python function called *compress*, which, for a given number $r$, computes the SVD factorization for every channel of color RGB cutting it in $r$. The function returns the relative error between the original picture and the compressed one and the new compressed picture with name *compressed-r_error*. Let us visualize some results. We have chosen three photos to work on:



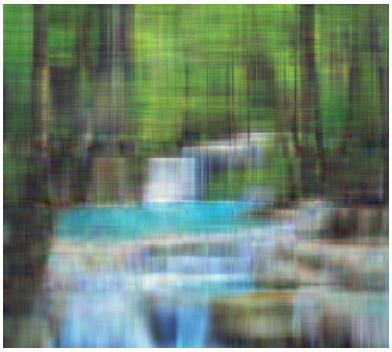
(a) Landscape

(b) Graduation

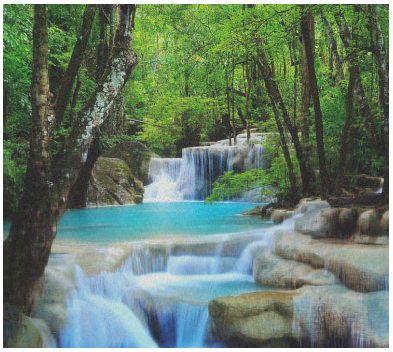

(c) You

Figure 2: Photographs to be compressed. The weights of the images are 221.3KB, 4.6MB and 2.1MB respectively.

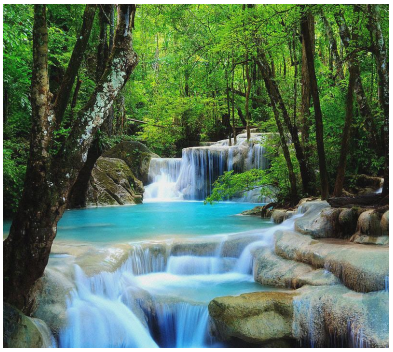
For the first photograph, since the image weight is not too big, we have chosen several values of $r$ to compare them with each other. More precisely, we have chosen values from 10 to 600 with a step of 10. In Figure 3 we can see some examples.



(a) $r = 10$



(b) $r = 150$



(c) $r = 400$

Figure 3: Compressed images with the value $r$. The error is 29.53%, 13.75% and 3.88% respectively.

As explained before, the higher value of $r$, the better quality we get. Nevertheless, notice that the weights of the images have considerably decreased. The weights are 63.8KB, 134.6KB and 168.3 KB respectively. To finish with this picture, I want to show an interesting chart that I build. It shows the error rate depending on the value $r$. Notice (Figure 4) how the error decreases when the value of $r$ increases.

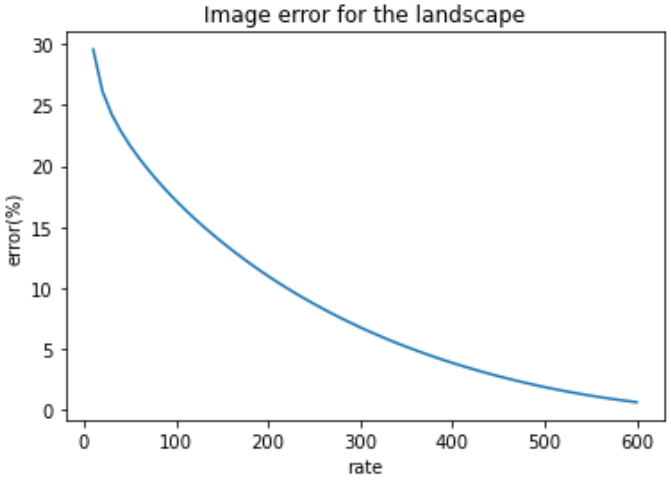



Figure 4: Error rate depending on the value $r$

Let us continue with the picture of my graduation and your picture, Arturo. Since the weights of these photographs are huge, we only have chosen four values of $r$, otherwise the program was taking a long time to finish. See Figures 5 and 6 to visualize the results.

(a) $r = 10$ (b) $r = 100$ (c) $r = 500$

Figure 5: Compressed images with the value $r$. The error is 18.25%, 7.32% and 4.82% respectively.

(a) $r = 10$ (b) $r = 100$ (c) $r = 500$

Figure 6: Compressed images with the value $r$. The error is 9.37%, 3.49% and 1.30% respectively.

As before, the weights of these pictures have considerably decreased. For my graduation picture, the weights are 478.5KB, 830KB and 1.3MB respectively and for your picture, Arturo,

the weights are 176KB, 301KB and 411.5KB respectively.

To finish this section, let us add two theoretical results.

**Theorem 3.1** (Eckart–Young–Mirsky theorem, for Frobenius norm)**.** *Let $A \in \mathbb{R}^{m\times n}$ be a real matrix with $m \geq n$. Suppose that $A = U\Sigma V^T$ is the singular value decomposition of $A$. Then, the best rank $k$ approximation to $A$ in the Frobenius norm, denoted by $||\cdot||_F$, is given by*

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

*where $u_i$ and $v_i$ denote the $i$-th column of $U$ and $V$ respectively.*

*Proof.* Firstly, notice that, since the norm is invariant for orthogonal matrices, as seen in theoretical lessons

$$||A - A_k||_F^2 = \left\| \sum_{i=k+1}^{n} \sigma_i u_i v_i^T \right\|_F^2 = \sum_{i=k+1}^{n} \sigma_i^2.$$

Therefore, we need to show that for any other matrix $B_k$ of rank at most $k$, we have that

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

It is not difficult to see that for every matrix $X$ and $Y$ we have that

$$\sigma_i(X) + \sigma_j(Y) \geq \sigma_{i+j-1}(X + Y).$$

where $\sigma_i(T)$ is the $i$-singular value of the matrix $T$. Since $\sigma_{k+1}(B_k) = 0$, when $X = A - B_k$ and $Y = B_k$, we conclude that for $i \geq 1$, $j = k + 1$,

$$\sigma_i(A - B_k) \geq \sigma_{k+1}(A)$$

Therefore,

$$||A - B_k||_F^2 = \sum_{i=1}^{n} \sigma_i(A - B_k)^2 \geq \sum_{i=k+1}^{n} \sigma_i(A)^2 = ||A - A_k||_F^2.$$

□

**Theorem 3.2** (Eckart–Young–Mirsky theorem, for the 2-norm)**.** *Let $A \in \mathbb{R}^{m\times n}$ be a real matrix with $m \geq n$. Suppose that $A = U\Sigma V^T$ is the singular value decomposition of $A$. Then, the best rank $k$ approximation to $A$ in the 2-norm, denoted by $||\cdot||_2$, is given by*

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

*where $u_i$ and $v_i$ denote the $i$-th column of $U$ and $V$ respectively.*

*Proof.* Firstly, notice that, as seen in theoretical lessons,

$$||A - A_k||_2^2 = \left\| \sum_{i=k+1}^{n} \sigma_i u_i v_i^T \right\|_2^2 = \sigma_{k+1}^2.$$

Therefore, we need to show that for any other matrix $B_k$ of rank at most $k$, we have that

$$||A - A_k||_2^2 = \sigma_{k+1}^2 \leqslant ||A - B_k||_2^2.$$

Assume $rank(B) = k$ and notice that the dimension of the null space of $B_k$, $\mathcal{N}(B_k)$, is $n - k$. Due to the dimensions, calling $V_{k+1} = Span(v_1, \cdots, v_{k+1})$ the space generated by the first $k+1$ singular vectors of $A$, we have that there exist $w \in \mathcal{N}(B_k) \cap V_{k+1}$, $||w||_2 = 1$. Then,

$$||A - B_k||_2^2 \geq ||(A - B_k)w||_2^2 = ||Aw||_2^2 = \sum_{i=1}^{k+1} \sigma_i^2 |v_i^T z|^2 \geq \sigma_{k+1}^2 \sum_{i=1}^{k+1} |v_i^T z|^2 = \sigma_{k+1}^2$$

since $u_i$ are orthogonal vectors, the ordering of the singular values and $z \in V_{k+1}$. Therefore,

$$||A - A_k||_2^2 = \sigma_{k+1}^2 \leqslant ||A - B_k||_2^2.$$

□

# 4 Principal component analysis (PCA)

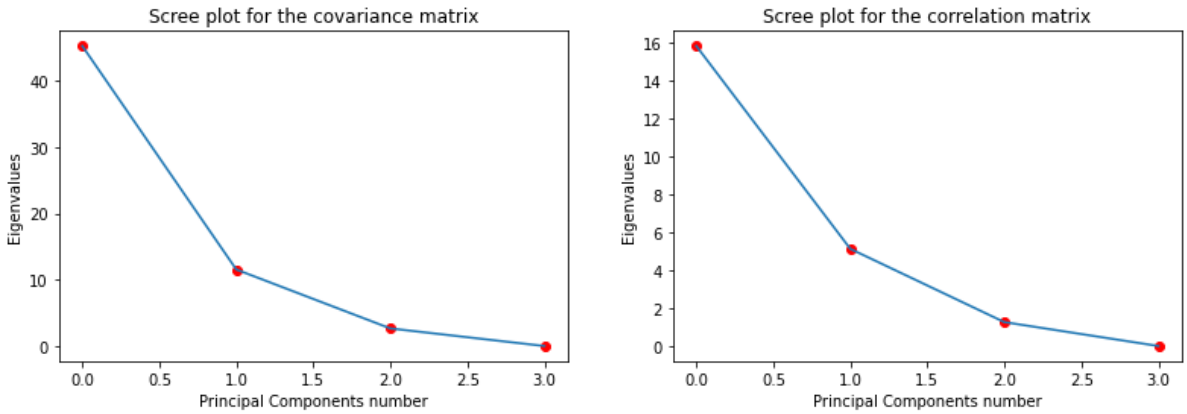
To finish this report, let us focus on the PCA. This final script is composed of five pyhton functions. I have created two functions to read the datasets, one for the *example.dat*, called *read_txt* and another one for the *csv* file, called *read_csv*.[2]. The main function is called *PCA*. It computes either the covariance or correlation matrix and builds the matrix $Y$ explained in the project to compute the SVD factorization and computes 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. Moreover, we have made three more functions in order to implement some rules to discuss about the number of principal components needed to explain the datasets. These functions are *rule_34*, which implement the 3/4 of the total variance rule, *Kasier*, which implements the Kasier rule and finally, *Scree_plot*, which plots the Scree plot with matplotlib. Let us explain these rules:

- The 3/4 of the total variance rule dictates that we will keep components until we reach the 75% of total variance.
- The 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 (see Figure 7 for understanding the "elbow concept").
- The Kasier rule dictates that a component should be retained if it has the eigenvalue greater than or equal to one.

When the program is executed, one can see the following: for the first dataset, the study with the covariance and correlation matrix and for the second dataset, the *csv* file, the study with the covariance matrix. The values that we print are: 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, the Kasier rule, the 3/4 of the total variance rule and the scree plot.

Let us discuss about the number of principal components needed to explain the datasets. For the first dataset, we can observe the followings scree plots:

[2]Notice that we have to transpose the matrices!!




(a) Covariance matrix (b) Correlation matrix

Figure 7: Scree plots for the first dataset

In this example, for both matrices, there are two eigenvalues that lay on the left of the "elbow" of the curve, which are considered relevant, while the rest are considered redundant. Nevertheless, since the scree plot test has been heavily criticized due to its subjectivity, we have also computed the Kasier rule and the 3/4 of the total variance rule. In both cases, the Kasier rule gives us that each matrix would be characterized by three principal components, while the 3/4 of the total variance rule gives us that for the covariance matrix only needs one principal component and the correlation matrix two.

Let us finish with the same study to the second dataset. In this case, we have only studied the covariance matrix, the scree plot of which is as follows:

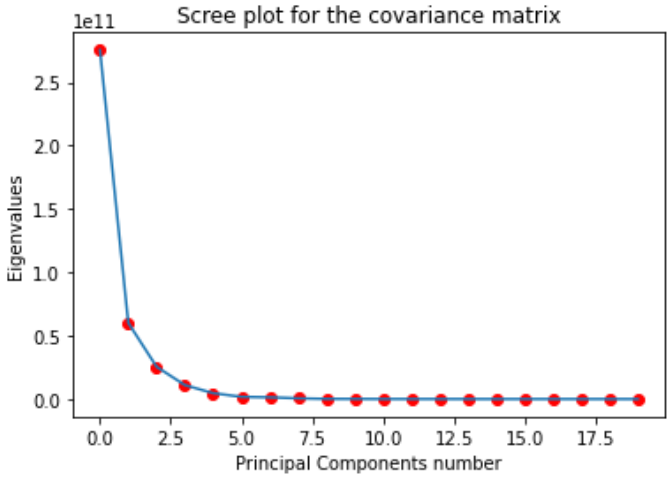



Figure 8: Scree plots for the second dataset

In this case, the scree plot test is highly subjective, as explained before. We could choose three or four principal components, but it is not clear at all. Notice that the Kasier rule detects nineteen principal components, while the 3/4 of the total variance rule detects two, since the total variance of the first two principal components are 72% and 15%.

**Remark 4.1.** *The Kasier rule has been criticized on the grounds that it sometimes results in the selection of too many components, as in the latter case.*