Project 2: SVD applications

1 Least squares problem

Exercise 1: *Write a program to solve the least squares problem (LSP) 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.*

Consider the least squares problem (LSP) $min||Ax-b||_2^2$.
We seek to write a program to solve the LSP by using SVD for the datasets `datafile` and `datafile2.csv` from `pr4: QR factorization and least square problems`.

The corresponding code is presented on the file `C1_1.py`. The code implements the QR-factorization method in order to find the solution for the low rank LSP and computes the algorithm as described on the `Practice 4` theory. It computes the value of c from the linear system $R_1u=c$ in the three possible ways, given that R_1 is non singular, and hence we can apply the normal equations, the regular QR factorization, and the direct computation of the linear system by using the python's function `np.linalg.solve`. Since this is not the goal of this project, any detail on this computation is presented on the Appendixes.

For the matrix given by the file `datafile2.csv`, we have a matrix $A\in\mathbb{R}^{m\times n}$, with $m=15, n=11$ with rank $rank(A)=r=10$. Then we have a rank deficient LSP. Then the corresponding approach is implemented when conducting the SVD factorization. However, since A is actually close to rank deficient, even when dropping out the last singular value, since we have $rank(A)=r=10<n=11$, the result obtained is way far from giving a satisfactory minimization of the least squares problem considered $min||Ax-b||_2^2$. Therefore, a truncated SVD algorithm is implemented, using a tolerance `tol = 1e-4`, that can be changed if wished from the function's parameters, which drops all the singular values smaller than this, and considers the algorithm with the truncated SVD, this is $A^+=V_{r'}\Sigma_{r'}U_{r'}^T$, where r' is the number of singular values remaining, i.e. the number of singular values greater than the tolerance.

Following up, the polynomial fitting for the points obtained from `dataset` is also conducted by solving the corresponding linear system directly and by using the SVD. The algorithm followed in this part is to solve the full-rank LS problem, applied to the fitting polynomial problem, when having the points: $(x_i,y_i)\in\mathbb{R}^2,\ 1\leq i\leq n$. We seek to fit a polynomial $p_m(x)=a_0+a_1x+a_2x^2+\cdots+a_mx^m$, i.e. $p_m(x_i)=y_i$, then the lineal system considered corresponds to:

$$\begin{pmatrix}1 & x_0 & x_0^2 & \cdots & x_0^m\\ 1 & x_1 & x_1^2 & \cdots & x_1^m\\ \vdots & \vdots & \vdots & \cdots & \vdots\\ 1 & x_n & x_n^2 & \cdots & x_n^m\end{pmatrix}\begin{pmatrix}a_0\\ a_1\\ \vdots\\ a_n\end{pmatrix}=\begin{pmatrix}y_0\\ y_1\\ \vdots\\ y_n\end{pmatrix}$$

This system is solved using both methods, and the results are shown in a plot with different values of r for the truncated SVD algorithm.

2 Graphics compression

Exercise 1: *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.*

Theorem 2.1 (Eckart-Young theorem)**.** *The best rank k approximation of $A = U\Sigma V^T$, where V, Σ, U are given by the singular value decomposition, for all $k = 1, \dots, r$, in both the 2-norm and the Frobenius norm is given by the matrix*

$$A_k = \sigma_1 u_1 v_1^T + \cdots + \sigma_k u_k v_k^T = U_k \Sigma_k V_k^T$$

This is, that for both norms we have

$$\|A - B\| \geq \|A - A_k\|,$$

for any other k-rank $m \times n$ matrix B.
Moreover for both norms, due to the orthogonal invariance

$$\|A - A_k\|_2 = \sigma_{k+1}, \quad \|A - A_k\|_F = (\sigma_{k+1}^2 + \cdots + \sigma_r^2)^{\frac{1}{2}}$$

Proof. Note firstly, that for the last part of the theorem, we can see for both norms that given the orthogonal invariance we have that

$$||A - A_k|| = ||\Sigma - \Sigma_k|| = \left\| \begin{pmatrix} 0 & & & & & \\ & \ddots & & & & \\ & & 0 & & & \\ & & & \sigma_{k+1} & & \\ & & & & \ddots & \\ & & & & & \sigma_r \end{pmatrix} \right\|$$

and so, we have for both norms:

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

Therefore, we have to see that for any given k rank matrix B,

$$\|A - A_k\|_2 = \sigma_{k+1} \leq \|A - B\|_2$$
$$\|A - A_k\|_F^2 = \sum_{i=k+1}^{n} \sigma_i^2 \leq \|A - B\|_F^2$$

Consider first the 2 norm case. Since B has rank k, then its null space $\mathcal{N}(B) \subset \ltimes$ must be of dimension $n - k$ by the rank nullity theorem. Then, consider the truncated SVD $n \times (k+1)$ matrix $V_{k+1} = [v_1 \cdots v_{k+1}]$, with rank $k+1$ and with column space $\mathcal{C}(V_{k+1}) \subset \mathbb{R}^n$. Note now that $dim\mathcal{N}(B) + dim\mathcal{C}(V_{k+1}) = n - k + 1 + 1 = n + 1$, and therefore they cannot be disjoint spaces, since they are both susbsets of $\mathbb{R}^n$, and hence we can find $w \in \mathcal{N}(B) \cap \mathcal{C}(V_{k+1})$. Without loss of generality we can choose the unitary vector from the intersection, i.e. $\|w\|_2 = 1$. Because $w \in \mathcal{C}(V_{k+1})$ then we can write

$w = \sum_{i=1}^{k+1} w_i v_i$, with $\sum_{i=1}^{k+1} w_i^2 = 1$. Then, we have that by the properties of 2-norm and applying that $w \in \mathcal{N}(B)$ we have

$$\|A - B\|_2^2 \geq \|(A-B)w\|_2^2 = \|Aw\|_2^2 = w^T V \Sigma^2 V^T w = \sum_{i=1}^{k+1} \sigma_i^2 w_i^2 \geq$$

$$\geq \sigma_{k+1}^2 \sum_{i=1}^{k+1} w_i^2 = \sigma_{k+1}^2 = \|A - A_k\|_2^2,$$

as wanted to prove.

Consider now the Frobenius norm. Note that for general X, Y $m \times n$ matrices, due to the triangle inequality with the spectral norm we have that $\sigma_1(X+Y) \leq \sigma_1(X) + \sigma_1(Y)$. Thus suppose that that X_k, Y_k are respectively the rank k approximation to X, Y by SVD factorization, hence we have for $i, j \geq 1$ and denoting $Z = X + Y$,

$$\begin{aligned}\sigma_i(X) + \sigma_j(Y) &= \sigma_1(X - X_{i-1}) + \sigma_1(Y - Y_{j-1}) \geq \sigma_1(X + Y - X_{i-1} - Y_{j-1}) \\ &\geq \sigma_1(Z - Z_{i+j-2}) = \sigma_{i+j-1}(Z),\end{aligned}$$

since $\text{rank}(X_{i-1} + Y_{j-1}) \leq \text{rank}(Z_{i+j-2})$.
Consider now the case of $X = A - B$ and $Y = B$, and therefore $Z = A$. Then, fixing $i \geq 1, j = k+1$, and given that B has rank k we have that $\sigma_{k+1}(B) = 0$, we obtain that $\sigma_i(A - B) \geq \sigma_{k+i}(A)$. Therefore:

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

□

Exercise 2: *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.*

Unfortunately `scipy.ndimage.imread()` has been deprecated, so I will be using instead `matplotlib .pyplot.imread`.
When reading therefore an image data, the function returns an array that has shape

1. (M,N) for grayscale images
2. (M,N,3) for RGB images
3. (M,N,4) for RGBA images

PNG images are returned as floats arrays ranging from 0. to 1., instead to returning arrays ranging from 0 to 255. All other formats, such as .jpeg used in this case, are returned with a bit depth determined by the file's content.

Color images are represented in python as 3 dimensional numpy arrays — the third dimension to represent the color values (red,green blue). However, svd method is applicable to two dimensional matrices. So we have to find a way to convert the 3 dimensional array to 2 dimensional arrays, apply svd and reconstruct it back as a 3 dimensional array . There are two ways to do it, by reshaping the array or by using layers.

The reshape method involves flattening the third dimension of the image array into the second dimension using numpy's reshape method. The layers method consists of treating a colour image as a stack of 3 separate two dimensional images (Red,blue and green layers) and then applying the truncated svd reconstruction on each two dimensional layer separately. Then the layers are put back together.

Both methods are implemented on code `C2_2.py` and the corresponding outputs are stored on the `output_` folders on the `imagesCompression` folder. All the images used are personal pictures that I either took by myself of in which I appear.

3 Principal component Analysis

The correlation matrix is the covariance matrix for the standarized data. So computing the corresponding covariance matrix of the centered data corresponds to computing the correlation matrix. Additionaly, note that we should always work with the centered data, otherwise the PCA study may provide misleading results.

Exercise 1: *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.*

The corresponding code is presented on the file `C3_1.py`, in which a function computes the PCA for the given matrix by calculating the corresponding covariance matrix or the correlation matrix, depending on the parameters given when defining the function. In fact, in the code the corresponding eigenvectors of the correlation matrix of A are printed in order to check the validity of the results obtained.

Exercise 2. *The file `RCsGoff.csv` contains 20 observations of 58581 genes (variables) from an experiment.*

A code preforming the PCA of the covariance matrix is attached on the file `C3_2.py` which also gives as output a file `output_PCA.txt` with the PCA coordinates of each observation and additionally plots the representation of the input dataset in terms of the first two principal components, accounting for the 72.299% and 15.784% of the variance of the data.

Note that, just applying our intuition, we could retain the 3/4 of the total variance by using the first two principal components, which together contain the 78.084% of the total variance. This is known as the 3/4 of total variance method. However, there are other methods to decide which principal components to keep to explain the information given by the data.
A common method for determining the number of PCs to be retained is a graphical representation known as a scree plot. A Scree Plot is a simple line segment plot that shows the eigenvalues for each individual PC. It shows the eigenvalues on the y-axis and the number of factors on the x-axis. It always displays a downward curve. The scree plot criterion looks for the "elbow" in the curve and selects all components just before the line flattens out. Checking out the plot, we can see that there is an elbow around 17 features, which takes a lot of features to represent enough signficant information of the dataset.

Finally, we could consider the Kaiser rule: pick PCs with eigenvalues of at least 1. Those correspond to the first 19 features, since the only eigenvalue lower than one is the last one. Note that this rule does not represent an accurate method to choose the number of principal components used.

Therefore, I would stand by the first role, taking the amount of principal components by looking at the proportion of variance they represent. If we wish to have more than 3/4 of the variance, we would take 2 principal components. If we seek to have at least 4/5 then it is also enough to take these two components. In fact, by just taking the 5 first principal components we already have more than 99% of the total variance.

Appendix

The rank deficient LS problem using QR

Consider the optimization problem $min\ ||Ax - b||_2^2$, $A \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^m, m \geq n$, with $r := rank(A) < n$, then the solution of the least squares problem is not unique. Consider the following procedure:

1. If $rank(A) < n$, then considering the QR factorization we have $A = QR$ ill-conditioned.

2. There exsits an inifite number of solutions. So, there is freedom on the choice of the solution. Maybe we choose the solution that has minimum norm. From a numerical point of view, having freedom is something we can take advantage of. We have then $n - r$ dimensional vector space of solutions.

3. We can look for the solution such that $||x||_2$ is minimum, by considering the SVD.

To apply the QR factorization to solve the LS rank-deficient problem one needs column pivoting. Let's consider $AP = QR$, $A \in \mathbb{R}^{m\times n}, R \in \mathbb{R}^{m\times n}, Q \in \mathbb{R}^{m\times m}$, $rank(A) = r < n$, then we have the block matrix

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

with $R_1 \in \mathbb{R}^{r\times r}$ being a non-singular and upper triangular matrix.

So going back to our minimization problem, note that we want to minimize the norm. So let's apply the QR factorization conducted $AP = QR \iff A = QRP^T$:

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

Consider the decomposition of the vector $P^T x = (u, v)^T$, where $u \in \mathbb{R}^r, v \in \mathbb{R}^{n-r}$, so recalling that Q is orthogonal, we have:

$$||b - Ax||_2^2 = ||b - Q \begin{pmatrix} R_1 & S \\ 0 & 0 \end{pmatrix} ||b - Ax||_2^2 = \left|\left| Q^T b - \begin{pmatrix} u \\ v \end{pmatrix} \right|\right|_2^2$$

now by considering $Q = (c, d)^T$ we obtain:

$$||b - Ax||_2^2 = ||b - Q \begin{pmatrix} R_1 & S \\ 0 & 0 \end{pmatrix} ||b - Ax||_2^2 = \left|\left| \begin{pmatrix} c - R_1 u - Sv \\ d \end{pmatrix} \right|\right|_2^2 = \underbrace{||c - R_1 u - Sv||_2^2}_{=0} + \underbrace{||d||_2^2}_{error}$$

Since we consider (u,v) such that the first condition holds. Thus, we have that $min||b - Ax||_2^2 = ||d||_2^2$ provided by the conditions: $R_1 u + Sv = c$ given considering the strategy: - $v = 0 \Rightarrow R_1 u = c$ - (u, v) such that $||(u, v)||_2^2$ is minimum by considering the SVD.

In order to find the solution we take the basic solution of the LS problem by setting $v = 0$ on the linear system $R_1 u + Sv = c$, then we have the linear system left $R_1 u = c$, which can be solved since it is full rank $R_1 \in \mathbb{R}_{\not k}{}^{r\times r}$ and rank(R_1)= r, then we could apply the normal equations to solve the full rank LS problem $R_1 u = c$:

1. Compute $R_1^T R_1$

2. Compute $\alpha = R_1^T c$

3. Compute the Cholesky factorization of $R_1^T R_1 = GG^T$

4. Solve $Gy = \alpha$, and $G^T u = y$ to obtain u_{LS}

Now we would have the solution u_{LS}, and we can therefore compute the value x_{LS} we are looking for from the expression $P^T x = (u, v)^T$. Hence $x = P(u, v)^T$, with $v = 0, u_{LS}$ obtained.

Since the value obtained provided a huge norm order, the last step is conducted by implementing the regular QR factorization over this full rank LSP, since R_1 is nonsingular. Then the algorithm implemented is:

1. Compute QR factorization of $R_1 = Q_2 R_2$
2. Write $Q^T c = (y_1, y_2)^T \in \mathbb{R}^m$, with $y_1 \in \mathbb{R}^n$
3. We solve $R_2 u = Q_2^T c \iff R_2^1 u = y_1$ and fix $y_1 = 0$