Numerical Linear Algebra Homework Project 2: Least Squares, Orthogonalization, and the SVD

Catalano Giuseppe, Cerrato Nunzia

Problem 1

(1) Suppose we are given m pairs of data points, $(x_1, y_1), \ldots, (x_m, y_m)$. We want to find a linear combination of prescribed functions $\phi_1, \ldots, \phi_n$ whose values at the points $x_i \in [a, b]$, $1 \leq i \leq m$, approximate the values $y_1, \ldots, y_m$ as well as possible. More precisely, the problem is to find a function of the form $f(x) = \alpha_1\phi_1(x) + \cdots + \alpha_n\phi_n(x)$ such that

$$\sum_{i=1}^{m} [y_i - f(x_i)]^2 \leq \sum_{i=1}^{m} [y_i - g(x_i)]^2 \quad \forall g \in \text{Span}(\phi_1, \ldots, \phi_n), \tag{1}$$

where, usually, $m > n$. It is possible to rephrase the problem as:

$$f = \arg \min_{f \in \text{Span}(\phi_1, \ldots, \phi_n)} \sum_{i=1}^{m} [y_i - f(x_i)]^2 . \tag{2}$$

Now we can define a column vector $\mathbf{z} \in \mathbb{R}^n$ such that:

$$[\mathbf{z}]_i = \alpha_i \tag{3}$$

and a matrix A such that:

$$[A\mathbf{z}]_i = f(x_i) = \alpha_1\phi_1(x_i) + \cdots + \alpha_n\phi_n(x_i). \tag{4}$$

In this way, the element of the $i-th$ row and $j-th$ column of the matrix A is:

$$[A]_{ij} = a_{ij} = \phi_j(x_i). \tag{5}$$

Finally, defining a column vector $\mathbf{b} \in \mathbb{R}^n$ such that:

$$[\mathbf{b}]_i = y_i \tag{6}$$

we can rewrite the Eq. (2) as follows:

$$\tilde{\mathbf{z}} = \arg \min_{\mathbf{z} \in \mathbb{R}^n} \|\mathbf{b} - A\mathbf{z}\|_2^2 = \arg \min_{\mathbf{z} \in \mathbb{R}^n} \|\mathbf{b} - A\mathbf{z}\|_2. \tag{7}$$

Note that, once having obtained $\tilde{\mathbf{z}}$ it is possible to construct explicitly the function f.

(2) Now we suppose to take $\phi_k = x^{k-1}$, $1 \leq k \leq n$. Under this assumption, the matrix A takes the form:

$$A = \begin{bmatrix} x_1^0 & \cdots & x_1^{n-1} \\ \vdots & \ddots & \vdots \\ x_m^0 & \cdots & x_m^{n-1} \end{bmatrix}. \tag{8}$$

We want to prove that, assuming that $x_i \neq x_j$ for $i \neq j$, A has full rank, namely $\text{rank}(A) = n$.
Proof: Proving that $\text{rank}(A) = n$ is equivalent to prove that $\dim(\ker(A)) = 0$, that means

x_i	8	10	12	16	20	30	40	60	100
y_i	0.88	1.22	1.64	2.72	3.96	7.66	11.96	21.56	43.16

Table 1: Data points (x_i, y_i) given in the text.

that $\nexists\, \mathbf{v} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$, s.t. $\mathbf{v} \in \ker(A)$. We want to prove this statement by contradiction. Therefore, we look for a vector $\mathbf{v} \in \mathbb{R}^n$, with $\mathbf{v} \neq \mathbf{0}$, such that $A\mathbf{v} = \mathbf{0}$, that means:

$$\begin{cases} v_1 x_1^0 + \cdots + v_n x_1^{n-1} = 0 \\ \vdots \\ v_1 x_m^0 + \cdots + v_n x_m^{n-1} = 0 \end{cases} \tag{9}$$

Defining the polynomial

$$p_{\mathbf{v}}^{(n-1)}(x) = \sum_{i=1}^{n} v_i x^{i-1}, \tag{10}$$

we can observe that, for any choice of $\mathbf{v} \neq \mathbf{0}$, $p_{\mathbf{v}}^{(n-1)}(x)$ admits at most $n-1$ different roots, therefore, since $m > n$, $\nexists\, \mathbf{v} \neq \mathbf{0}$ such that $A\mathbf{v} = \mathbf{0}$. This concludes the proof. □

(3) Consider the problem of finding the best fit with a quadratic function $f(x) = \alpha_1 + \alpha_2 x + \alpha_3 x^2$ for the following data:

Below is the code that solves the normal equations:

$$A^T A\mathbf{v} = A^T \mathbf{b} \tag{11}$$

using the Cholesky factorization algorithm and then compares the result with the one found using the QR factorization of the matrix A. To begin with, we report the functions defined in the library `Project_2.py`.

```
def compute_A(points,order):
  r'''This function computes the Vandermonde matrix A taking as inputs an
   ↪  array, containing the coordinates of the points, and an integer
   ↪  representing the order of the polynomial.

  Parameters
  ----------
  points : ndarray
           Input points
  order : int
          Order of the polynomial

  Returns
  -------
  A : ndarray
      Vandermonde matrix
  '''
  m = points.shape[0]
  n = order + 1
  A = np.zeros((m,n))
  for index in range(m):
    A[index,:]=(points[index])**range(n)
```

```
21
22      return A
23
24  def LeastSquare_Cholesky(A, b):
25      r''' This function computes the Cholesky factorization of a matrix
        ↪  :math:`A^{T}A)`, and solves the linear system :math:`A^{T}Ax =
        ↪  A^{T}b)`, giving as output the solution of the system.
26
27      Parameters
28      ----------
29      A : ndarray
30          Input matrix of dimension :math:`(m\times n)`
31      b : ndarray
32          Input vector of dimension :math:`(m)`
33
34      Returns
35      -------
36      x : ndarray
37          Solution of the linear system
38      '''
39      A_transpose = np.transpose(A)
40      C = A_transpose@A
41      d = A_transpose@b
42
43      L, low = scipy.linalg.cho_factor(C)
44      x = scipy.linalg.cho_solve((L, low),d)
45      return x
46
47  def LeastSquare_QR(A, b):
48      r''' This function computes the QR factorization of a matrix :math:`A`,
        ↪  and solves the linear system :math:`Rx = Q^{T}b`, giving as output the
        ↪  solution of the system.
49
50      Parameters
51      ----------
52      A : ndarray
53          Input matrix of dimension :math:`(m\times n)`
54      b : ndarray
55          Input vector of dimension :math:`(m)`
56
57      Returns
58      -------
59      x : ndarray
60          Solution of the linear system
61      '''
62      Q, R = np.linalg.qr(A)
63      c = np.transpose(Q) @ b
64      x = scipy.linalg.solve_triangular(R, c, lower = False)
65      return x
66
```

Below is the main program.

```python

import numpy as np
import scipy.linalg

# Set the number of significant digits
np.set_printoptions(precision=15, suppress=True)

# Set the order of the polynomial
order = 2

# Write the data points in appropriate arrays
points = np.array([8,10,12,16,20,30,40,60,100])
b = np.array([0.88,1.22,1.64,2.72,3.96,7.66,11.96,21.56,43.16])

# Build the Vandermonde matrix
A = compute_A(points, order)

# Obtain the solution to the minimization problem
x_chol = Least_Square_Cholesky(A, b)
x_qr = Least_Square_QR(A, b)

print(f'x_Cholesky = {format(x_chol)}')
print(f'x_QR = {format(x_qr)}')
```

The results that we obtained are the following:

```
x_Cholesky = [-1.91914925269909,0.278213536291725,0.001739400875055]
x_QR = [-1.91914925269904,0.278213536291722,0.001739400875055]
```

We can observe that, for this problem, both the algorithms perform in a similar way. In fact, the results differ at most in the 15th digit. In Figure 1 we show the input data points, reported in Table 1, and the function $f(x) = \alpha_1 + \alpha_2 x + \alpha_3 x^2$, whose parameters are obtained by solving the least square problem.

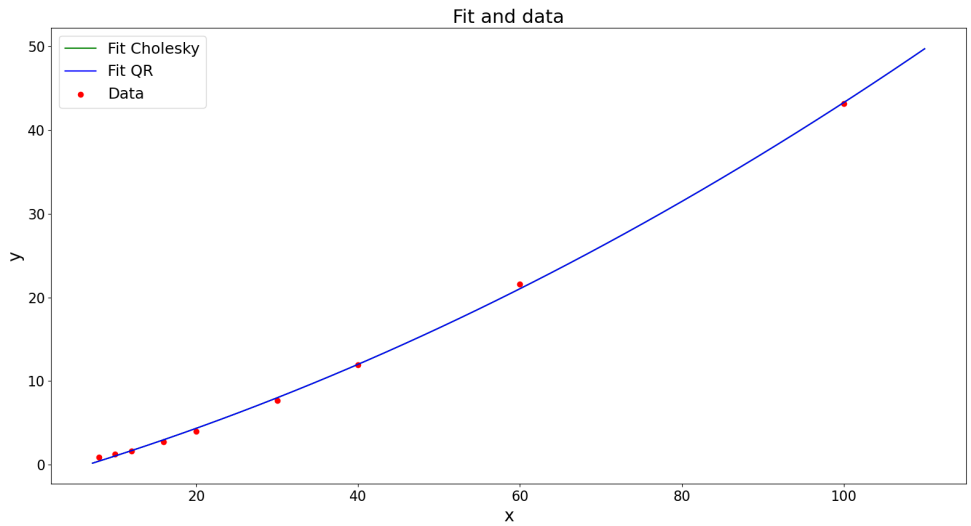


Figure 1: Data points (x_i, y_i), reported in Table 1, and best fit functions $f(x) = \alpha_1 + \alpha_2 x + \alpha_3 x^2$ obtained by using the Cholesky and QR factorization to solve the least square problem.

(4) The following code computes the residual $\mathbf{r} = \mathbf{d} - C\hat{\mathbf{x}}$, where $\hat{\mathbf{x}} = [-1.919,\ 0.2782,\ 0.001739]$ is the approximate solution of the least squares problem, given in the text of the exercise.

```
# Compute A^{T}, C, and d
A_transpose = np.transpose(A)
C = A_transpose@A
d = A_transpose@b

# Consider the approximate solution
approx_solution = np.array([-1.919,0.2782,0.001739])

# Compute the residual
residual = d - C @ approx_solution
residual_norm_2 = np.linalg.norm(residual, ord=2)
print(f'Residual = {format(residual)}')
print(f'Norm 2 of the residual = {format(residual_norm_2)}')
```

The results that we obtain are:

```
Residual = [ 0.009503999999993  0.716895999998997 62.090847999905236]
Norm 2 of the residual = 62.09498720144941
```

We can note that the value of the residual is relatively high, even though, if we compute the relative error, by using as true value the one obtained before by solving the least square problem, we find that it is approximatively

$$\frac{\|\mathbf{x} - \hat{\mathbf{x}}\|_2}{\|\mathbf{x}\|_2} \simeq 7.73\text{e-}05. \tag{12}$$

It can be observed that the relative error assumes a quite small value that may seem incompatible with the value of the residual. However, by manipulating the relation

$$\frac{1}{k_2(C)} \frac{\|\mathbf{r}\|_2}{\|\mathbf{d}\|_2} \leq \frac{\|\mathbf{x} - \hat{\mathbf{x}}\|_2}{\|\mathbf{x}\|_2} \leq k_2(C) \frac{\|\mathbf{r}\|_2}{\|\mathbf{d}\|_2}, \tag{13}$$

we can obtain the following relation for $\mathbf{r}$:

$$\frac{\|\mathbf{d}\|_2}{k_2(C)} \frac{\|\mathbf{x} - \hat{\mathbf{x}}\|_2}{\|\mathbf{x}\|_2} \leq \|\mathbf{r}\|_2 \leq k_2(C) \|\mathbf{d}\|_2 \frac{\|\mathbf{x} - \hat{\mathbf{x}}\|_2}{\|\mathbf{x}\|_2}. \tag{14}$$

Computing the bound for the residual this expression becomes:

$$5 \times 10^{-7} \lesssim \|\mathbf{r}\|_2 \lesssim 3 \times 10^{9}. \tag{15}$$

It emerges that the range in which we can find the 2-norm of the residual is, in this case, of sixteen orders of magnitude. This is due to the high condition number of C, that is $k_2(C) \simeq 8 \times 10^7$. Therefore, this relation suggests not to use the residual as a measure of the accuracy of the solution when the problem is ill-conditioned (in this case, in fact, the approximate solution is quite accurate).

Problem 2

(1) Let $A \in \mathbb{R}^{m \times n}$, with $\text{rank}(A) = n$, let A = QR be the (full) QR factorization of A, with $Q \in \mathbb{R}^{m \times m}$ orthogonal and $R \in \mathbb{R}^{m \times n}$ upper trapezoidal. Also, let $A = Q_1 R_1$ be the reduced QR factorization of A with $Q_1 \in \mathbb{R}^{m \times n}$ having orthonormal columns and $R_1 \in \mathbb{R}^{m \times m}$

upper triangular. Show that R_1 is nonsingular, and that the columns $\mathbf{q}_1 ..., \mathbf{q}_n$ of Q_1 form an orthonormal basis for $\text{Ran}(A)$, the column space of A. Also, find an orthonormal basis for $\text{Null}(A^T)$, the null space of A^T .

We start showing that R_1 is nonsingular. Since $\text{rank}(A) = n$, we know, from the rank-nullity theorem, that

$$A\mathbf{x} = \mathbf{0} \Leftrightarrow \mathbf{x} = \mathbf{0}. \tag{16}$$

Multiplying both sides for Q_1^T, and knowing that $Q_1^T Q_1 = \mathbb{1}_n$, we obtain

$$R_1\mathbf{x} = \mathbf{0} \Leftrightarrow \mathbf{x} = \mathbf{0}, \tag{17}$$

that concludes the proof.
Now we want to show that the columns $\{\mathbf{q}_1, \dots, \mathbf{q}_n\}$ of Q_1 form an orthonormal basis for $\text{Ran}(A)$. We start by observing that, from the definition of range of a matrix:

$$\forall\, \mathbf{y} \in \text{Ran}(A)\, \exists\, \mathbf{x} \in \mathbb{R}^n : \mathbf{y} = A\mathbf{x} = Q_1 R_1 \mathbf{x}. \tag{18}$$

In a similar way, knowing that R_1 is nonsingular and, therefore, it is a bijective map from $\mathbb{R}^n$ to $\mathbb{R}^n$, we can set $\mathbf{x}' = R_i\mathbf{x}$ and say that

$$\forall\, \mathbf{y} \in \text{Ran}(A)\, \exists\, \mathbf{x} \in \mathbb{R}^n : \mathbf{y} = A\mathbf{x} = Q_1 \mathbf{x}', \tag{19}$$

which means that $\text{Ran}(A) = \text{Ran}(Q_1) = \text{Span}\{\mathbf{q}_1, \dots, \mathbf{q}_n\}$.
Finally, we want to find an orthonormal basis for $\text{Null}(A^T)$. We start by proving that $\text{Null}(A^T) = (\text{Ran}(A))^\perp$. This is equivalent to say that $(\text{Ran}(A)) \perp \text{Null}(A^T)$, that is $\forall\, \mathbf{y} \in \text{Ran}(A)$ and $\forall\, \mathbf{z} \in \text{Null}(A^T)$, $\mathbf{y}^T\mathbf{z} = 0$. From the definition of $\text{Ran}(A)$, for $\mathbf{y} \in \text{Ran}(A)$, $\exists\, \mathbf{x} \in \text{Dom}(A)$ s.t. $\mathbf{y} = A\mathbf{x}$. Hence, $\mathbf{y}^T\mathbf{z} = \mathbf{x}^T A^T \mathbf{z} = 0$. This leads immediately to the solution. In fact:

$$(\text{Ran}(A))^\perp = \text{Span}\{\mathbf{q}_1, \dots, \mathbf{q}_n\}^\perp = \text{Span}\{\mathbf{q}_{n+1} \dots, \mathbf{q}_m\} = \text{Null}(A^T), \tag{20}$$

where $\{\mathbf{q}_{n+1} \dots, \mathbf{q}_m\}$ represent the remaining $m-n$ columns of the matrix Q. This concludes the proof. □

(2) Given the full rank matrix

$$A = \begin{bmatrix} 1.07 & 1.10 \\ 1.07 & 1.11 \\ 1.07 & 1.15 \end{bmatrix}, \tag{21}$$

compute $A^T A$ in $\beta = 10$, $t = 3$ digit arithmetic and check if $A^T A$ is positive definite.
We performed the prescribed calculation using the rounding function

$$\text{fl}(x) = \arg\min_{y \in \mathbb{F}} |y - x|, \tag{22}$$

where $\mathbb{F} = \{\pm(0.d_1 d_2 d_3) \times 10^p : d_i \in 0, \dots, 9,\, d_1 \neq 0,\ -2046 \leq p \leq 2046\} \cup \{0\}$. Moreover, when there is ambiguity, we always approximate away from 0. The rounding function has been applied at each step of the calculation, meaning that, if we consider $\mathbf{v}, \mathbf{u} \in \mathbb{R}^n$, then

$$\mathbf{v}^T\mathbf{u} = \text{fl}\left(\sum_{i=1}^{n} \text{fl}(v_i u_i)\right). \tag{23}$$

The result of the calculation is the following:

$$(A^T A)_r = \begin{bmatrix} 3.42 & 3.6 \\ 3.6 & 3.76 \end{bmatrix}, \tag{24}$$

where r stands for *rounded*. We can note that, even if A^TA must be positive definite, with the considered approximation this matrix becomes indefinite, in fact $\det(A^TA) = 3.42 \times 3.76 - 3.6^2 = 12.8592 - 12.96 < 0$.

(3) Compute the QR factorization of the matrix

$$A = \begin{bmatrix} 1.07 & 1.10 \\ 1.07 & 1.11 \\ 1.07 & 1.15 \end{bmatrix} = [\mathbf{a}_1 \ \mathbf{a}_2] \tag{25}$$

in 3 digit arithmetic, i.e. with $\beta = 10$, $t = 3$. The $\text{fl}(x)$ function, reported in Eq. (22), has been applied at each step, as in the previous exercise.
We want to compute H_1A, which will assume the following form:

$$H_1A = \begin{bmatrix} r_{11} & \mathbf{r}_1^T \\ \mathbf{0} & A_1 \end{bmatrix}, \tag{26}$$

without explicitly computing the matrix H_1. We know that $r_{11} = -\text{sgn}(a_{11})\|\mathbf{a}_1\|_2$, while the other columns of this matrix can be obtained as

$$H_1\mathbf{a}_j = \mathbf{a}_j - 2(\mathbf{u}_1^T\mathbf{a}_j)\mathbf{u}_1,$$

where

$$\hat{\mathbf{u}}_1 \coloneqq \mathbf{a}_1 + (\text{sgn}(a_{11})\|\mathbf{a}_1\|_2)\mathbf{e}_1, \quad \text{and} \quad \mathbf{u}_1 = \hat{\mathbf{u}}_1/\|\hat{\mathbf{u}}_1\|_2.$$

Note that these expressions hold in general, while in this case we only have to compute the column H_1a_2. We will denote with tilde $\sim$ all the rounded numbers and vectors obtained using the $\text{fl}(x)$ function. We start by computing $\tilde{r}_{11}$:

$$\tilde{r}_{11} = \text{fl}\left(\sqrt{\text{fl}\left(\text{fl}(a_{11})^2 + \text{fl}(a_{21})^2 + \text{fl}(a_{31})^2\right)}\right) = -1.85 \tag{27}$$

Then:

$$\tilde{\hat{\mathbf{u}}}_1 = \begin{pmatrix} \text{fl}(a_{11} - \tilde{r}_{11}) \\ a_{21} \\ a_{31} \end{pmatrix} = \begin{pmatrix} 2.92 \\ 1.07 \\ 1.07 \end{pmatrix} \tag{28}$$

By normalizing this vector, with the same approach that we used to compute the normalization of $\mathbf{a}_1$ (see Eq. (27)), one obtains:

$$\tilde{\mathbf{u}}_1 = \begin{pmatrix} 0.888 \\ 0.325 \\ 0.325 \end{pmatrix}. \tag{29}$$

Having obtained all the rounded vectors, it is possible to compute the column $H_1\mathbf{a}_2$. We first compute the rounded scalar product $\mathbf{u}_1^T\mathbf{a}_2$, which we called $\tilde{s}$, as:

$$\tilde{s} = \text{fl}\left(\sum_{i=1}^{3} \text{fl}\Big((\tilde{\mathbf{u}}_1)_i(\mathbf{a}_2)_i\Big)\right) = 1.71. \tag{30}$$

Then:

$$(H_1\mathbf{a}_2)_j = \text{fl}\Big((\mathbf{a}_2)_j - \text{fl}(2\tilde{s}(\tilde{\mathbf{u}}_1)_j\Big) \text{ for } j = 1, 2, 3. \tag{31}$$

Therefore, the full-rounded column will be (we omit the tilde here for clarity):

$$H_1\mathbf{a}_2 = \begin{pmatrix} -1.94 \\ 0 \\ 0.04 \end{pmatrix}. \tag{32}$$

At this stage of the computation, the rounded matrix $H_1 A$ assumes the following form:

$$H_1 A = \begin{pmatrix} -1.85 & -1.94 \\ 0 & 0 \\ 0 & 0.04 \end{pmatrix}. \tag{33}$$

By following the same rounding procedure, we can compute $\tilde{r}_{22}$ and $\tilde{\mathbf{u}_2}$, in order to obtain the rounded matrix $H_2 H_1 A$, which will be equal to the (rounded) matrix $\tilde{R}$ of the QR factorization of A. In particular, we found that $\tilde{r}_{22} = -0.04$ and the normalized vector $\tilde{\mathbf{u}}_2^T$ is equal to $\tilde{\mathbf{u}}_2^T = (0, 0.707, 0.707)^T$. Therefore, the final matrix will be:

$$\tilde{R} = \begin{pmatrix} -1.85 & -1.94 \\ 0 & -0.04 \\ 0 & 0 \end{pmatrix}. \tag{34}$$

Now we want to compute $Q = [\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3]$ and, in order to do it, we use the following relations:

$$\begin{aligned} \mathbf{q}_1 &= H_1 H_2 \mathbf{e}_1 = \mathbf{e}_1 - 2(\mathbf{u}_1^T \mathbf{e}_1)\mathbf{u}_1 \\ \mathbf{q}_2 &= H_1 H_2 \mathbf{e}_2 = H_1(\mathbf{e}_2 - 2(\mathbf{u}_2^T \mathbf{e}_2)\mathbf{u}_2) = \mathbf{e}_2 - 2(\mathbf{u}_2^T \mathbf{e}_2)\mathbf{u}_2 - 2(\mathbf{u}_1^T \mathbf{e}_2)\mathbf{u}_1 + 4(\mathbf{u}_2^T \mathbf{e}_2)(\mathbf{u}_1^T \mathbf{u}_2)\mathbf{u}_1. \\ \mathbf{q}_3 &= H_1 H_2 \mathbf{e}_3 = H_1(\mathbf{e}_3 - 2(\mathbf{u}_2^T \mathbf{e}_3)\mathbf{u}_2) = \mathbf{e}_3 - 2(\mathbf{u}_2^T \mathbf{e}_3)\mathbf{u}_2 - 2(\mathbf{u}_1^T \mathbf{e}_3)\mathbf{u}_1 + 4(\mathbf{u}_2^T \mathbf{e}_3)(\mathbf{u}_1^T \mathbf{u}_2)\mathbf{u}_1 \end{aligned} \tag{35}$$

Using these relations, we find that the rounded matrix $\tilde{Q}$ is:

$$\tilde{Q} = \begin{bmatrix} -0.578 & 0.572 & 0.572 \\ -0.578 & 0.212 & -0.788 \\ -0.578 & -0.788 & 0.212 \end{bmatrix}. \tag{36}$$

Finally, we can verify that, multiplying $\tilde{Q}$ and $\tilde{R}$ and approximating as before, we recover the original matrix A.

At this point, we can do a final comment. This rounding procedure, when considering only three significant digits, causes problems at different levels. The first one, which appears in the second point of this exercise, is that the matrix $A^T A$, which must be positive definite by construction, becomes indefinite when rounded. This implies that, for example, the Cholesky factorization algorithm, which requires the input matrix to be positive definite, is not applicable, while this problem does not appear for the QR factorization algorithm. However, in this case, a second problem emerges, since the columns of the matrix Q, which must be unitary and, therefore, orthonormal, become not exactly orthonormal when considering the rounded version of Q.

(4) Let $A \in \mathbb{R}^{m \times n}$, with $\text{rank}(A) = n$, and let $\mathbf{b} \in \mathbb{R}^m$. Let $A = Q_1 R_1$ be a reduced QR decomposition of A, where $Q_1 \in \mathbb{R}^{m \times n}$ has orthonormal columns and $R_1 \in \mathbb{R}^{n \times n}$ is upper triangular and nonsingular. Show that a reduced QR factorization of the augmented matrix $A_+ = [A\ \mathbf{b}]$ is given by:

$$A_+ = [Q_1\ \mathbf{q}_{n+1}] \begin{bmatrix} R_1 & \mathbf{z} \\ \mathbf{0}^T & \rho \end{bmatrix} \tag{37}$$

where $\mathbf{z} = Q_1^T \mathbf{b}$. Also, show that $|\rho| = \|\mathbf{b} - A\mathbf{x}^*\|_2$ where $\mathbf{x}^*$ is the solution to the least squares problem $\|\mathbf{b} - A\mathbf{x}\|_2 = \min$.

Since the QR factorization exists for any matrix, we can define the complete and reduced QR factorization of the matrix A_+, respectively, as $A_+ = QR$ and $A_+ = Q_+ R_+$, where Q is an

orthogonal matrix and Q_+ has orthonormal columns, while R is upper trapezoidal and R_+ is upper triangular.
We now focus on the reduced QR factorization of A_+. Given this, and since we know that $A = Q_1 R_1$, we can pose, without loss of generality:

$$Q_+ = [Q_1 \ \mathbf{q}_{n+1}], \qquad \text{and} \qquad R_+ = \begin{bmatrix} R_1 & \mathbf{z} \\ \mathbf{0}^T & \rho \end{bmatrix}, \tag{38}$$

where we know that $\mathbf{q}_{n+1} \in \text{Span}\{\mathbf{q}_1, \ldots, \mathbf{q}_n\}^\perp$, while $\mathbf{z}$ and ρ have to be defined. In order to find the value of $\mathbf{z}$ and ρ, we compute the product $Q_+ R_+$ and impose that it is equal to A_+:

$$[Q_1 \ \mathbf{q}_{n+1}] \begin{bmatrix} R_1 & \mathbf{z} \\ \mathbf{0}^T & \rho \end{bmatrix} = [Q_1 R_1 \ Q_1 \mathbf{z} + \rho \mathbf{q}_{n+1}] = [A \ \mathbf{b}]. \tag{39}$$

Therefore, we must have

$$\mathbf{b} = Q_1 \mathbf{z} + \rho \mathbf{q}_{n+1}. \tag{40}$$

Then, by multiplying both members of the last equation for Q_1^T, we find that:

$$Q_1^T \mathbf{b} = Q_1^T Q_1 \mathbf{z} + \rho Q_1^T \mathbf{q}_{n+1} = \mathbf{z}, \tag{41}$$

where we used that the columns of Q_1 are orthonormal and $\mathbf{q}_{n+1}$ is orthogonal to all the columns of Q_1.
In order to prove that $|\rho| = \|\mathbf{b} - A\mathbf{x}^*\|_2$, where $\mathbf{x}^*$ is the solution to the least squares problem $\|\mathbf{b} - A\mathbf{x}\|_2 = \min$, we consider the complete QR factorization of A_+, namely $A_+ = QR$, where we can define $Q = [Q_1 Q_2]$ and $Q_1 = [\mathbf{q}_1, \ldots, \mathbf{q}_n]$, $Q_2 = [\mathbf{q}_{n+1}, \ldots, \mathbf{q}_m]$.

$$A_+ = [Q_1 \ Q_2] \left[\begin{array}{cc} R_1 & \mathbf{z} \\ \mathbf{0}^T & \rho \\ \hline \multicolumn{2}{c}{\mathbf{0}} \end{array}\right] = [Q_1 R_1 \ \ Q_1 \mathbf{z} + Q_2 (\rho, 0, \ldots, 0)^T]. \tag{42}$$

From this expression, we obtain the expression:

$$\mathbf{b} = Q_1 \mathbf{z} + Q_2 (\rho, 0, \ldots, 0)^T \tag{43}$$

and, therefore, by multiplying both sides of this equation by Q_2^T:

$$Q_2^T \mathbf{b} = \rho (1, 0, \ldots, 0)^T. \tag{44}$$

We can now define the residual $\mathbf{r}$ as $\mathbf{r} = \mathbf{b} - A\mathbf{x}^*$, and compute:

$$\|\mathbf{r}\|_2^2 = \|Q^T \mathbf{r}\|_2^2 = \left\| \begin{bmatrix} Q_1^T \mathbf{b} \\ Q_2^T \mathbf{b} \end{bmatrix} - \begin{bmatrix} Q_1^T A\mathbf{x}^* \\ Q_2^T A\mathbf{x}^* \end{bmatrix} \right\|_2^2 = \|Q_1^T \mathbf{b} - R_1 \mathbf{x}^*\|_2^2 + \|Q_2^T \mathbf{b}\|_2^2 \tag{45}$$

where we used the invariance of the 2-norm with respect to unitary transformations. Knowing that $\mathbf{x}^* = R_1^{-1} Q_1^T \mathbf{b}$, we obtain:

$$\|\mathbf{r}\|_2 = \|Q_2^T \mathbf{b}\|_2 = \|\rho (1, 0, \ldots, 0)^T\|_2 = |\rho|, \tag{46}$$

and this concludes the proof. □

Problem 3

(1) Let $A \in \mathbb{R}^{m\times n}$, with $\text{rank}(A) = n$. The singular value decomposition of A can be written as $A = U\Sigma V^T$ or, equivalently, as $A = \sum_{i=1}^{n} \sigma_i \mathbf{u}_i \mathbf{v}_i^T$, where $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$ represent the singular values of A, while $\{\mathbf{u}_i\}_{i=1,\ldots,n}$ and $\{\mathbf{v}_i\}_{i=1,\ldots,n}$ are, respectively, left and right singular vectors of A.

Express the singular values and singular vectors of the following matrices in terms of those of A.

(a) $(A^TA)^{-1}$

$$(A^TA)^{-1} = (V^T)^{-1}\Sigma^{-2}V^{-1} = V\Sigma^{-2}V^T$$

In order to obtain this expression we used that $U^TU = \mathbb{1}$ and that $V^{-1} = V^T$, since U and V are orthogonal. We can conclude that both left and right singular vectors of $(A^TA)^{-1}$ are equal to the right singular vectors of A, while singular values of the former matrix are equal to those of A raised to the -2 power. Note that, since singular values of A are in increasing order and in this expression they are raised to a negative power, they must appear in reverse order. Moreover, we can also note that, writing A^T and A in terms of their SVDs, we obtain that $(A^TA)^{-1}$ is orthogonally similar to a diagonal matrix and, therefore, diagonalizable.

(b) $(A^TA)^{-1}A^T$

$$(A^TA)^{-1}A^T = V\Sigma^{-2}V^TV\Sigma U^T = V\Sigma^{-1}U^T$$

Analogously to the previous case, we know that $V^TV = \mathbb{1}$. Here we see that left singular vectors of $(A^TA)^{-1}A^T$ are equal to the right singular vectors of A, right singular vectors of $(A^TA)^{-1}A^T$ are equal to the left singular vectors of A, while singular values of this matrix are equal to the inverse of singular values of A.

(c) $A(A^TA)^{-1}$

$$A(A^TA)^{-1} = U\Sigma V^TV\Sigma^{-2}V^{-T} = U\Sigma^{-1}V^T$$

Note that this result can also be obtained from the previous case, by noting that $A(A^TA)^{-1} = ((A^TA)^{-1}A^T)^T$. We can see that left and right singular vectors of $A(A^TA)^{-1}$ coincide with that of A, while singular values of this matrix are equal to the inverse of singular values of A.

(d) $A(A^TA)^{-1}A^T$

$$A(A^TA)^{-1}A^T = U\Sigma^{-1}V^TV\Sigma U^T = UU^T$$

In this case, both left and right singular vectors of $A(A^TA)^{-1}A^T$ are equal to left singular vectors of A, while singular values of this matrix are all equal to 1.

(2) Given the following matrix:

$$A = \begin{bmatrix} 1 & 2 \\ 0 & 2 \end{bmatrix}, \tag{47}$$

its singular values can be computed by considering the square root of the eigenvalues of the matrix A^TA (as emerges from the case **(a)** of the previous point of this exercise), which has the form:

$$A^TA = \begin{bmatrix} 1 & 2 \\ 2 & 8 \end{bmatrix}. \tag{48}$$

Hence, by imposing that $p(\lambda) = \det(A^T A - \lambda \mathbb{1}) = 0$, we find the two eigenvalues of $A^T A$ and, therefore, the two singular values of A, which are:

$$\sigma_{1,2} = \sqrt{\frac{9 \pm \sqrt{65}}{2}}. \tag{49}$$

Finally, knowing that the spectral condition number is defined as $k_2(A) = \frac{\sigma_{\max}}{\sigma_{\min}}$, we can find the spectral condition number of A:

$$k_2(A) = \sqrt{\frac{9 + \sqrt{65}}{9 - \sqrt{65}}} \simeq 4.26. \tag{50}$$

We want now to see how the unit ball is modified by the transformation described by the matrix A. In other words, we want to see what is the image of a vector $\mathbf{x} = (\cos(\theta), \sin(\theta))$, parametrizing the unit ball on a plane, when considering the linear transformation $\mathbf{y} = A\mathbf{x}$. The vector $\mathbf{y}$ under the transformation A assumes the form:

$$\mathbf{y} = (\cos(\theta) + 2\sin(\theta), 2\sin(\theta)) \quad \theta \in [0, 2\pi)\,.$$

Graphically, we have:

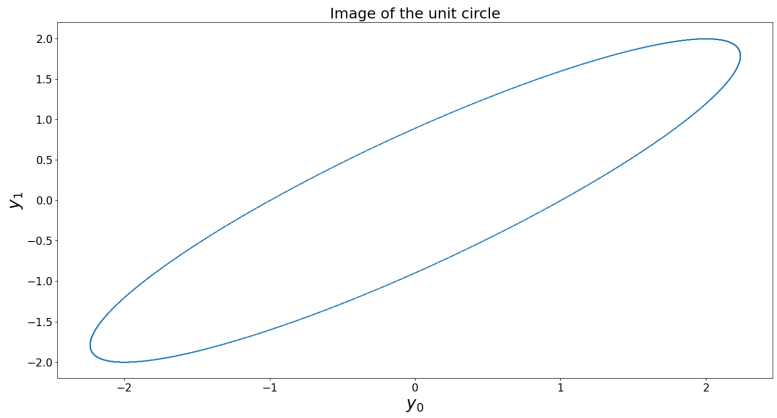


Figure 2: Image of the unit circle under the transformation defined by the matrix A, reported in Eq. (47).

(3) We found the solution $\mathbf{x} = [1.11 \times 10^{-16}, 1, 1]^T$ by computing the pseudoinverse matrix of A, A^+, and by applying it to $\mathbf{b}$, as follows:

$$\mathbf{x} = A^+ \mathbf{b}. \tag{51}$$

In the following there is the code that implements this operation.

```
A = np.array([[1,1,0],[0,1,1]])
b = [1,2]
x = np.linalg.pinv(A) @ b
print(f'Minimum norm solution x = {x}')
```

(4) Given the matrix

$$A = \begin{bmatrix} -4 & -2 & -4 & -2 \\ 2 & -2 & 2 & 1 \\ -800 & 200 & -800 & -401 \end{bmatrix}, \tag{52}$$

we want to compute the singular values of A, the Moore-Penrose pseudoinverse of A, and its spectral condition number. We report in the following the Python script that we used to do it.

```python
import numpy as np

A = np.array([[-4.,-2.,-4.,-2.],[2.,-2.,2.,1.],[-800,200,-800,-401]])
U, singular_values, V_transpose = np.linalg.svd(A, compute_uv=True)
pseudoinverse = np.linalg.pinv(A)
spectral_cond_num = np.linalg.cond(A)
print(f'singular values of A = {singular_values}')
print(f'pseudoinverse of A = {pseudoinverse}')
print(f'spectral condition number of A = {spectral_cond_num}')
```

The results we obtained are reported below:

```
singular values of A = [1.21689895e+03 3.30829410e+00 4.21538860e-03]
pseudoinverse of A = [[-2.50833333e+01  5.00833333e+01  2.50000000e-01]
                      [-1.66666667e-01 -3.33333333e-01  7.57226996e-16]
                      [-2.50833333e+01  5.00833333e+01  2.50000000e-01]
                      [ 1.00000000e+02 -2.00000000e+02 -1.00000000e+00]]
spectral condition number of A = 288680.1350686104
```

Given the obtained results, we can say that $\text{rank}(A) = 3$. In fact, if we compute $(k_2(A))^{-1} = \frac{\sigma_3}{\sigma_1} \simeq 3.5 \times 10^{-6}$, we can see that $(k_2(A))^{-1} \gg u$, where $u = 2^{-53} \simeq 10^{-16}$.

(5) The best rank-k approximation of a matrix A, both in the Frobenius norm and in the 2-norm, is obtained by computing the singular value decomposition of A and then by considering only the first k terms in that expression. In other words, given a matrix A written in terms of its singular value decomposition as $A = \sum_{i=1}^{n} \sigma_i \mathbf{u}_i \mathbf{v}_i^T$, its best rank-k approximation is given by $\tilde{A}_k = \sum_{i=1}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^T$. Below is reported the Python code that we used to obtain the best rank-1 and rank-2 approximations of the matrix A, mentioned in the previous point of this exercise.

```python
A_rank_1 = singular_values[0]*np.outer(U[:,0],V_transpose[0])
A_rank_2 = A_rank_1 + singular_values[1]*np.outer(U[:,1],V_transpose[1])
spectral_cond_num_A_rank_2 = singular_values[0]/singular_values[1]
print(f'A_rank_1 = {A_rank_1}')
print(f'A_rank_2 = {A_rank_2}')
print(f'Spectral condition number of A_rank_2 =
↪  {spectral_cond_num_A_rank_2}')
```

We obtained the following two matrices:

```
A_rank_1 = [[-3.67478811e+00  9.18652344e-01 -3.67478811e+00 -1.84198740e+00]
            [ 2.16152803e+00 -5.40355726e-01  2.16152803e+00  1.08346585e+00]
            [-8.00001057e+02  1.99990537e+02 -8.00001057e+02 -4.01000500e+02]]

A_rank_2 = [[  -3.99955481   -2.00000178   -3.99955481   -2.00177721]
            [   1.99910978   -1.99999645    1.99910978    1.00355377]
            [-800.00000445  200.00000002 -800.00000445 -400.99998223]]

Spectral condition number of A_rank_2 = 367.8327598758788
```

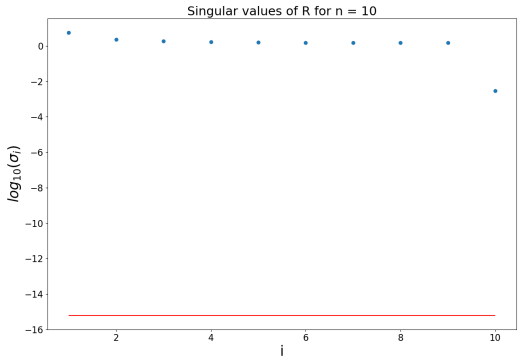
As we can see, the best rank-1 approximation of A returns a matrix that is already quite similar to the initial one, and the approximation improves when considering the best rank-2 approximation of A. This can be seen, in a more quantitative way, by considering the Frobenius norm of the difference between the approximated matrix and the initial matrix, both for the rank-1 and rank-2 approximation. By doing this, one obtains:

```
Frobenious norm between A and A_rank_1 = 3.308296788282531
Frobenious norm between A and A_rank_2 = 0.004215388599497665
```

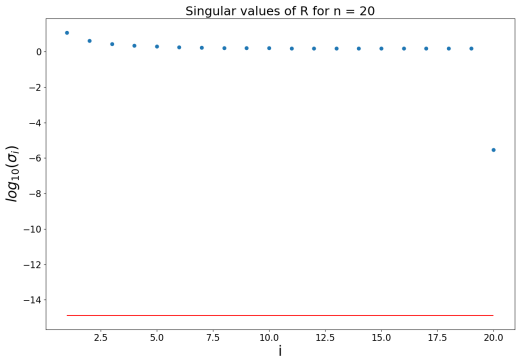
(6) Consider an upper triangular matrix matrix $R = (r_{ij})$, whose entries are given by $r_{ii} = 1$ and $r_{ij} = -1$ for $j > i$. We defined a function `R_matrix(n)` that allows us to compute this matrix for a given dimension n:

```
def R_matrix(n):
 R = np.triu(-np.ones((n,n)),k=1) + np.eye(n)
 return R
```

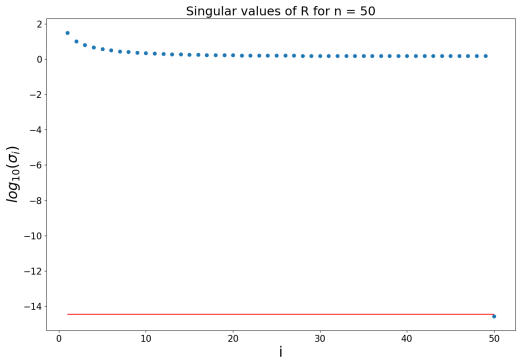
Note that we used an upper triangular mask to obtain the entries r_{ij}. Once derived R, we computed its singular values for $n = 10, 20, 50, 100$, as required. The plot of the singular values is reported below.



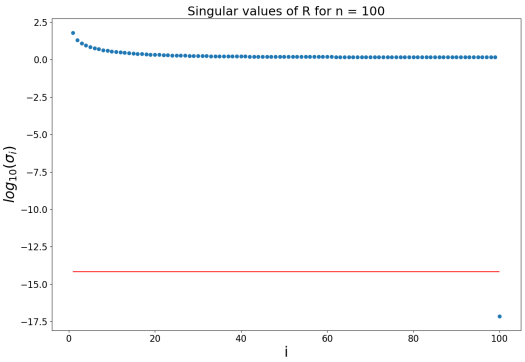
(a) n=10



(b) n=20



(c) n=50



(d) n=100

Figure 3: Singular values of the matrix R defined at the beginning of this exercise for $n = 10, 20, 50, 100$. The logarithm of the singular values is reported on the ordinate axes.

In order to understand when the matrix R becomes numerically singular, we put on the graphs a threshold line given by $u\sigma_1$, where u is the machine precision ($u \simeq 10^{-16}$). It can be observed that for $n = 50$ and for $n = 100$ the last singular value becomes smaller than the threshold value. This means that the matrix R becomes numerically singular for these values of its dimension.

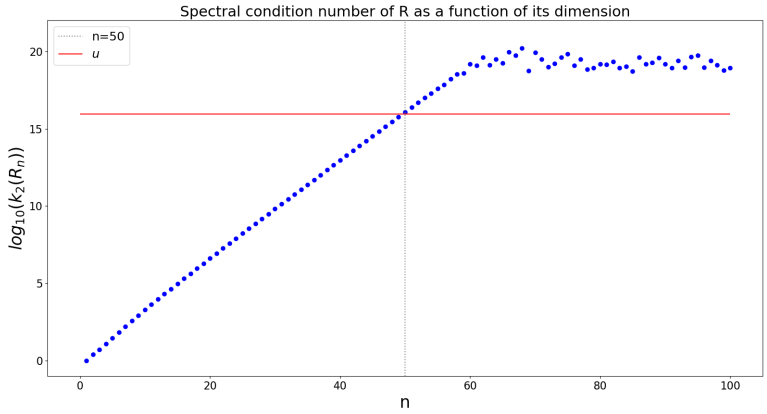


Figure 4: Spectral condition number of R as a function of its dimension. Gray dotted line is reported for $n = 50$, while the red line represents the threshold value, fixed by the machine precision u.

We can see that for $n = 50$ the matrix becomes numerically singular, since the condition number becomes greater than the inverse of the machine precision u.

Finally, we report in the following plot the number of the singular values that are smaller than $u\sigma_1$.

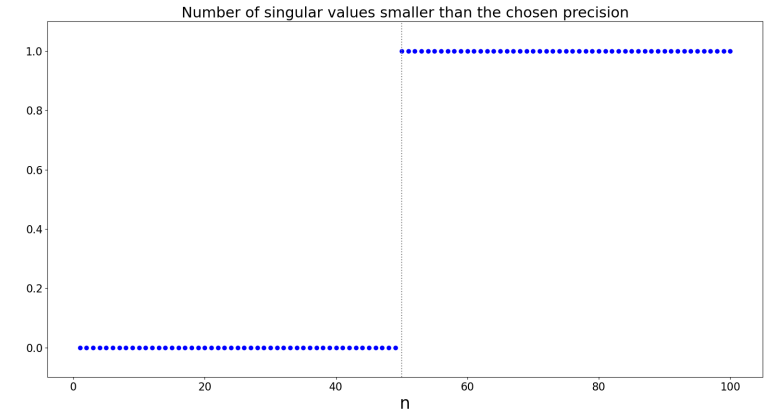


Figure 5: Number of the singular values smaller than $u\sigma_1$. Gray dotted line is reported for $n = 50$.

From this result, we can observe that the numerical rank of the matrix A becomes $n-1$ when $n \geq 50$, otherwise it is equal to n.