Q1) Gram-Schmidt Algorithm and QR decomposition

i) Write a code to generate a random matrix **A** of size $m \times n$ with $m > n$ and calculate its Frobenius norm, $\|\cdot\|_F$. The entries of **A** must be of the form $r.dddd$ (example 5.4316). The inputs are the positive integers m and n and the output should display the the dimensions and the calculated norm value.
Deliverable(s) : The code with the desired input and output (0.5)

Input:

We take positive integer entries in our case to be 5 and 4 for m and n respectively as

Code:

```
import numpy as np


def rand_rdddd():
 rdddd = str(np.random.randint(low=-9, high=9))+ '.'
 for i in range(4):
   rdddd += str(np.random.randint(low=0, high=9))
 return float(rdddd)

def rand_matrix(m,n):#########check indent for the if m<n: line of code
 if m<0 or n<0:
   raise Exception('error negative dimensions')
   if m < n:
    raise Exception('error, m > n not satisfied')
 A = np.array([rand_rdddd() for i in range(n*m)]).reshape(m,n)
 return A

def frobenius_norm(A):
 frob_norm = A.flatten() @ A.flatten().reshape(-1,1)
 return np.sqrt(frob_norm)[0]

def frobenius_norm_cnt_op(A):
 m,n = A.shape
 plus_cnt = m*n -1
 mul_cnt = m*n
 frob_norm = A.flatten() @ A.flatten().reshape(-1,1)
 return np.sqrt(frob_norm)[0], plus_cnt, mul_cnt, 0

def dim_norm(m,n):
 A = rand_matrix(m,n)
 frob_norm = frobenius_norm(A)
 return f'dimensions: {m ,n} Frobenius norm: {frob_norm}'
```

Testing the code for Output:

```
# output

dim_norm(5,4)
```

```
'dimensions: (5, 4) Frobenius norm: 23.7858489613047'
```

ii) Write a code to decide if Gram-Schmidt Algorithm can be applied to columns of a given matrix $\mathbf{A}$ through calculation of rank. The code should print appropriate messages indicating whether Gram-Schmidt is applicable on columns of the matrix or not.
Deliverable(s) : The code that performs the test. (1)

The input Matrix A be given as:

```
A=rand_matrix(7,5)
```

Code:

```
def linear_indep(A):
 rk = np.linalg.matrix_rank(A)
 nb_col = A.shape[1]

 if nb_col == rk:
   print('Gram-Schmidt Algorithm can be applied \n')
   return True
   print('Gram-Schmidt Algorithm can NOT be applied \n')
 return False
```

Output:

```
linear_indep(A)
```

```
Gram-Schmidt Algorithm can be applied

True
```

iii) Write a code to generate the orthogonal matrix Q from a matrix $\mathbf{A}$ by performing the Gram-Schmidt orthogonalization method. Ensure that $\mathbf{A}$ has linearly independent columns by checking the rank. Keep generating $\mathbf{A}$ until the linear independence is obtained.
Deliverable(s) : The code that produces matrix $\mathbf{Q}$ from A (1)

Input:

```
A=rand_matrix(7,5)
```

Code:

```
def gram_schmidt(A):
 if not linear_indep(A):
   return None
  # othogonal basis of A initialize with 0
 Q = np.zeros(A.shape)

 for i in range(A.shape[1]):
     # vector to orthogonalize
     a = A[:, i]
     sub_ortho = Q[:, :i]
     numerator = a @ sub_ortho
     denominator = np.sum(sub_ortho * sub_ortho, axis =0)
     q = a - np.sum( numerator / denominator  * sub_ortho, axis=1)

     # normalization
     norm = np.sqrt(q @ q)
     q = q/norm
     Q[:, i] = q
 return Q
```

Output:

```
gram_schmidt(A)
```

```
Gram-Schmidt Algorithm can be applied

array([[ 0.18316997,  0.10038829, -0.6556617 , -0.12861967,  0.07898959],
       [ 0.35937802,  0.00551042,  0.65991909,  0.19099305,  0.35840471],
       [-0.61994772,  0.4494283 ,  0.10335339, -0.12505821,  0.43488096],
       [ 0.24525623,  0.65116989,  0.14704864, -0.58252563, -0.15606963],
       [-0.28636732, -0.17875093,  0.31715037, -0.25768097, -0.70242475],
       [ 0.36636057, -0.37369235,  0.03329783, -0.60475089,  0.25284748],
       [ 0.42021912,  0.4385042 , -0.02482345,  0.39984978, -0.30740063]])
```

iv) Write a code to create a QR decomposition of the matrix $\mathbf{A}$ by utilizing the code developed in the previous sub-parts of this question. Find the matrices $\mathbf{Q}$ and $\mathbf{R}$ and then display the value $\|\mathbf{A} - (\mathbf{Q}.\mathbf{R})\|_F$, where $\|\cdot\|_F$ is the Frobenius norm. The code should also display the total number of additions, multiplications and divisions to find the result. Deliverable(s) : The code with the said input and output. The results obtained for $\mathbf{A}$ generated with $m = 7$ and $n = 5$ with random entries described above. (2.5)

Input:

```
A=rand_matrix(7,5)
```

Code:

```
def gram_schmidt_cnt_op(A):
 if not linear_indep(A):
   return None

 plus_cnt = 0
 mul_cnt = 0
 div_cnt = 0
 m,n = A.shape

 # othogonal basis of A initialize with 0
 Q = np.zeros(A.shape)

 for i in range(n):
     # vector to orthogonalize
     a = A[:, i]
     sub_ortho = Q[:, :i]
     numerator = a @ sub_ortho

     mul_cnt += m*i
     plus_cnt += (m-1)*i

     denominator = np.sum(sub_ortho * sub_ortho, axis =0)

     mul_cnt += m*i
     plus_cnt += (m-1)*i

     q = a - np.sum( numerator / denominator  * sub_ortho, axis=1)

     div_cnt += i
     mul_cnt += i
     plus_cnt += i

     # normalization
     norm = np.sqrt(q @ q)
     a = q/norm

     mul_cnt += m
     plus_cnt += (m-1)
     div_cnt += m
```

```
    Q[:, i] = q
return Q, plus_cnt,mul_cnt ,div_cnt


def question_4(A):
 m,n = A.shape
 Q, plus_cnt, mul_cnt, div_cnt = gram_schmidt_cnt_op(A)
 R = Q.T @ A

 mul_cnt += m*n*n
 plus_cnt += (m-1)

 return frobenius_norm(A-Q @ R), plus_cnt, mul_cnt, div_cnt


x=question_4(A)
```

Output:

```
x=question_4(A)
print(f'Frobenious Norm: {x[0]}  \n Plus cnt : {x[1]} \n Mult Cnt : {x[2]} \n Div_cnt {x[3]} ')
```

```
Gram-Schmidt Algorithm can be applied

Frobenious Norm: 6748.387652670247
 Plus cnt : 166
 Mult Cnt : 360
 Div_cnt 45
```

Q2) Gradient Descent Algorithm

i) Consider the last 4 digits of your mobile number (Note : In case there is a 0 in one of the digits replace it by 3). Let it be $n_1n_2n_3n_4$. Generate a random matrix A of size $n_1n_2 \times n_3n_4$. For example, if the last four digits are 2311, generate a random matrix of size 23×11. Write a code to calculate the l_∞ norm of this matrix.
Deliverable(s) : The code that generates the results. (0.5)

Input:

```
A=rand_matrix(7,5)
```

Code:

```
#  maximum row sum
def infinite_norm(A):
 return np.max(np.sum(np.absolute(A), axis = 1))

A = np.random.randint(5, size=(26, 11))
infinite_norm(A)
```

Output:

```
x1=infinite_norm(A)
print(f'The l∞ norm of this matrix is : {x1}')
```

```
The l∞ norm of this matrix is : 30
```

ii) Generate a random vector b of size $n_1 n_2 \times 1$ and consider the function $f(\mathbf{x}) = \frac{1}{2}\|\mathbf{Ax} - \mathbf{b}\|_2^2$ where $\|\cdot\|_2$ is the vector ℓ_2 norm. Its gradient is given to be $\nabla f(\mathbf{x}) = \mathbf{A}^\top\mathbf{Ax} - \mathbf{A}^\top\mathbf{b}$. Write a code to find the local minima of this function by using the gradient descent algorithm (by using the gradient expression given to you). The step size τ in the iteration $\mathbf{x}_{k+1} = \mathbf{x}_k - \tau\nabla f(\mathbf{x}_k)$ should be chosen by the formula

$$\tau = \frac{\mathbf{g_k^T g_k}}{\mathbf{g_k^T A^T A g_k}}$$

where $\mathbf{g}_k = \nabla f(\mathbf{x_k}) = \mathbf{A}^\top\mathbf{Ax_k} - \mathbf{A}^\top\mathbf{b}$. The algorithm should execute until $\|\mathbf{x_k} - \mathbf{x_{k-1}}\|_2 < \mathbf{10^{-4}}$.
Deliverable(s) : The code that finds the minimum of the given function

Input:

```
A=rand_matrix(7,5)
```

```
b = np.random.randint(5, size=(26))
```

Code:

```
import pandas as pd#####state the libraries used and indent issue on th
e x_prev = x_new line of code
b = np.random.randint(5, size=(26))

def gradient(A, x, b):
 return A.T @ A @ x - A.T @ b

def step(A, grad):
 return (grad.T @ grad) / (grad.T @ A.T @ A @ grad)

def f(A, x, b): #####state the libraries used and indent issue on the x
_prev = x_new line of code
 norm = frobenius_norm(A@x -b)
 return 0.5 * norm * norm



x_prev = np.random.randint(5, size=(11))

x = [x_prev]
fx = [f(A, x_prev, b)]

while True:
 grad = gradient(A, x_prev, b)
 stp = step(A, grad)

 x_new = x_prev - stp * grad
 x.append(x_new)
 fx.append(f(A, x_new, b))

 if frobenius_norm(x_new - x_prev) < 10**-4:
   break
   x_prev = x_new

print('Local min: ', x_new)


Output:
```

iii) Generate the graph of $f(\mathbf{x_k})$ vs k where k is the iteration number and $\mathbf{x_k}$ is the current estimate of x at iteration k. This graph should convey the decreasing nature of function values.
Deliverable(s) : The graph that is generated. (0.5)

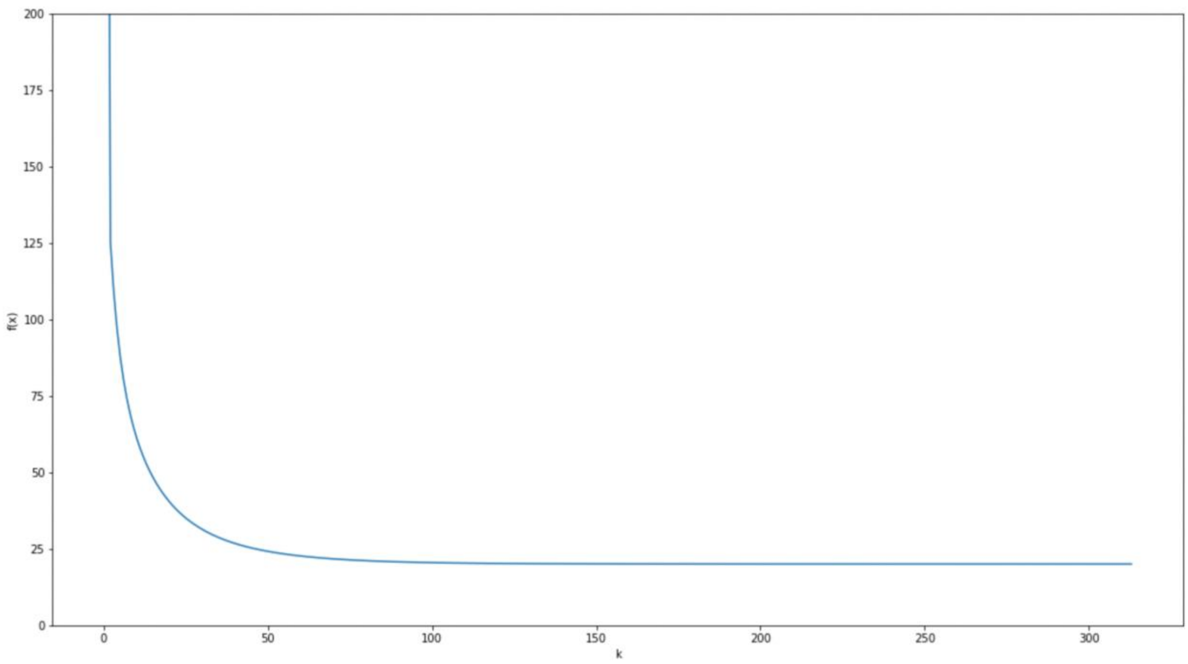
Input:

```
x_prev = np.random.randint(5, size=(11))
x = [x_prev]
```

Code:

```
# save x and f(x)to file
df = pd.DataFrame(x)
df['fx'] = fx
df.to_csv('x_f(x).csv', index=False)
```

Output:



Q3) Critical Points of a function

i) Generate a third degree polynomial in x and y named $g(x, y)$ that is based on your mobile number (Note : In case there is a 0 in one of the digits replace it by 3). Suppose your mobile number is 9412821233, then the polynomial would be $g(x, y) = 9x^3 - 4x^2y + 1xy^2 - 2y^3 + 8x^2 - 2xy + y^2 - 2x + 3y - 3$, where alternate positive and negative sign are used.
Deliverable(s) : The polynomial constructed should be reported. (0.5)

solution

%so the given polynomial will depend on the specific no.

%so we write the polynomial as follows with 0 replaced by 3

syms x y

writing the polynomial function itself

f=(9*x^3 - 9*x^2*y + 3*x*y^2 - 3*y^3 + 2*x^2 - 5*x*y + 8*y^2 - 3*x + 2*y-8) ;

Output :

```
f =

9*x^3 - 9*x^2*y + 2*x^2 + 3*x*y^2 - 5*x*y - 3*x - 3*y^3 + 8*y^2 + 2*y - 8
```

ii) Write a code to find all critical points of $g(x, y)$. You may use built in functions like 'solve' (or other similar functions) in Octave/Matlab to find the critical points .
Deliverable(s) : The code that finds the critical points along with the display of all the calculated critical points. (1)

Input:

f=(9*x^3 - 9*x^2*y + 3*x*y^2 - 3*y^3 + 2*x^2 - 5*x*y + 8*y^2 - 3*x + 2*y-8) ;

Code:

```
gradientf2 = jacobian(f,[x,y]);
% Hessian matrix (the square matrix of second partial derivatives)
Hessian_Matrix = jacobian(gradientf2, [x,y])
```

```
Hessian_Matrix =


[54*x - 18*y + 4,  6*y - 18*x - 5]
[ 6*y - 18*x - 5, 6*x - 18*y + 16]
```

```
% calculate the first partial derivatives
[xcr2, ycr2] = solve(gradientf2(1),gradientf2(2));
[xcr2, ycr2]
```

$$\begin{pmatrix} -\frac{24\,\mathrm{root}(\sigma_1, z, 1)^2}{11} + \frac{43\,\mathrm{root}(\sigma_1, z, 1)}{11} + \frac{3}{11} & \mathrm{root}(\sigma_1, z, 1) \\ -\frac{24\,\mathrm{root}(\sigma_1, z, 2)^2}{11} + \frac{43\,\mathrm{root}(\sigma_1, z, 2)}{11} + \frac{3}{11} & \mathrm{root}(\sigma_1, z, 2) \\ -\frac{24\,\mathrm{root}(\sigma_1, z, 3)^2}{11} + \frac{43\,\mathrm{root}(\sigma_1, z, 3)}{11} + \frac{3}{11} & \mathrm{root}(\sigma_1, z, 3) \\ -\frac{24\,\mathrm{root}(\sigma_1, z, 4)^2}{11} + \frac{43\,\mathrm{root}(\sigma_1, z, 4)}{11} + \frac{3}{11} & \mathrm{root}(\sigma_1, z, 4) \end{pmatrix}$$

where

$$\sigma_1 = z^4 - \frac{59\,z^3}{18} + \frac{341\,z^2}{144} + \frac{851\,z}{1728} + \frac{1}{1296}$$

```
% evaluate the Hessian matrix at the critical points
% and compute the eigenvalues of the matrix
H1=subs(Hessian_Matrix,[x,y],[xcr2(1), ycr2(1)])
```

$$\begin{pmatrix} -\frac{1296\,{\sigma_1}^2}{11} + \frac{2124\,\sigma_1}{11} + \frac{206}{11} & \frac{432\,{\sigma_1}^2}{11} - \frac{708\,\sigma_1}{11} - \frac{109}{11} \\ \frac{432\,{\sigma_1}^2}{11} - \frac{708\,\sigma_1}{11} - \frac{109}{11} & -\frac{144\,{\sigma_1}^2}{11} + \frac{60\,\sigma_1}{11} + \frac{194}{11} \end{pmatrix}$$

where

$$\sigma_1 = \mathrm{root}\left(z^4 - \frac{59\,z^3}{18} + \frac{341\,z^2}{144} + \frac{851\,z}{1728} + \frac{1}{1296}, z, 1\right)$$

```
H2=subs(Hessian_Matrix,[x,y],[xcr2(2), ycr2(2)]);
```

H2 =

```
[(2124*root(z^4 - (59*z^3)/18 + (341*z^2)/144 + (851*z)/1728 + 1/1296, z, 2))/11 - (1296*root(z^4 - (59*z^3)/18 + (341*z^2)/144 + (851*z)/1728 + 1/1296,
[  (432*root(z^4 - (59*z^3)/18 + (341*z^2)/144 + (851*z)/1728 + 1/1296, z, 2)^2)/11 - (708*root(z^4 - (59*z^3)/18 + (341*z^2)/144 + (851*z)/1728 + 1/1296
```

```
H3=subs(Hessian_Matrix,[x,y],[xcr2(3), ycr2(3)])
```

$$\begin{pmatrix} -\frac{1296\,\sigma_1{}^2}{11}+\frac{2124\,\sigma_1}{11}+\frac{206}{11} & \frac{432\,\sigma_1{}^2}{11}-\frac{708\,\sigma_1}{11}-\frac{109}{11} \\ \frac{432\,\sigma_1{}^2}{11}-\frac{708\,\sigma_1}{11}-\frac{109}{11} & -\frac{144\,\sigma_1{}^2}{11}+\frac{60\,\sigma_1}{11}+\frac{194}{11} \end{pmatrix}$$

where

$$\sigma_1 = \mathrm{root}\left(z^4 - \frac{59\,z^3}{18} + \frac{341\,z^2}{144} + \frac{851\,z}{1728} + \frac{1}{1296}, z, 3\right)$$

H4=subs(Hessian_Matrix,[x,y],[xcr2(4), ycr2(4)])

$$\begin{pmatrix} -\frac{1296\,\sigma_1{}^2}{11}+\frac{2124\,\sigma_1}{11}+\frac{206}{11} & \frac{432\,\sigma_1{}^2}{11}-\frac{708\,\sigma_1}{11}-\frac{109}{11} \\ \frac{432\,\sigma_1{}^2}{11}-\frac{708\,\sigma_1}{11}-\frac{109}{11} & -\frac{144\,\sigma_1{}^2}{11}+\frac{60\,\sigma_1}{11}+\frac{194}{11} \end{pmatrix}$$

where

$$\sigma_1 = \mathrm{root}\left(z^4 - \frac{59\,z^3}{18} + \frac{341\,z^2}{144} + \frac{851\,z}{1728} + \frac{1}{1296}, z, 4\right)$$

Output:

```
%for properly substituting the symbolic values of the coordinates
% converted them to numerical values, using double.

Eval1=double(eig(H1))
```

```
Eval1 =

    8.2095
   27.8403
```

```
Eval2=double(eig(H2))
Eval3=double(eig(H3))
Eval4=double(eig(H4))
```

```
Eval2 =

  -16.6655
   16.4747


Eval3 =

  -10.8440
   35.7147


Eval4 =

  -35.8766
  -15.0753
```

iii) Write a code to determine whether they correspond to a maximum, minimum or a saddle point.

Deliverable(s) : The code that identifies the type of critical points. The critical points and their type must be presented in the form of the table generated by code for the above polynomial. (1.5 marks)

Code:

```
% values of the eigenvalues will determine the type of critical points
% If the eigenvalues of the Hessian matrix all of them are positive at a
% critical point, the function has a local minimum there;
% if all are negative, the function has a local maximum;
% if they have mixed signs, the function has a saddle point;
% and if at least one of them is 0, the critical point is degenerate

CriticalPoints_Coordinates = [8.2095,27.8;-16.6655,16.4747;-10.8440,35.7147;-35.8766,-15.0753]
```

```
CriticalPoints_Coordinates =

    8.2095   27.8000
  -16.6655   16.4747
  -10.8440   35.7147
  -35.8766  -15.0753
```

```
Type_Of_Critical_Points = {};
p1 = double(eig(H1))
```

```
p1 =

    8.2095
   27.8403
```

```
if p1(1) ==0 || p1(2)==0
Type_Of_Critical_Points{1,1} = 'Critical point is degenerate'
elseif p1(1) > 0 && p1(2) >0
Type_Of_Critical_Points{1,1} = 'Critical Point is Minimum'
elseif p1(1) <0 && p1(2) <0
Type_Of_Critical_Points{1,1} = 'Critical Point is Maximum'
else
Type_Of_Critical_Points{1,1} = 'Critical Point is Saddle'
end
```

```
Type_Of_Critical_Points =

  1×1 cell array

    {'Critical Point is Minimum'}
```

```
p2= double(eig(H2))
```

```
p2 =

  -16.6655
   16.4747
```

```
if p2(1) ==0 || p2(2)==0
Type_Of_Critical_Points{2,1} = 'Critical point is degenerate'
elseif p2(1) > 0 && p2(2) >0
Type_Of_Critical_Points{2,1} = 'Critical Point is Minimum'
elseif p2(1) <0 && p2(2) <0
Type_Of_Critical_Points{2,1} = 'Critical Point is Maximum'
else
Type_Of_Critical_Points{2,1} = 'Critical Point is Saddle'
end
```

```
Type_Of_Critical_Points =

  2x1 cell array

    {'Critical Point is Minimum'}
    {'Critical Point is Saddle' }
```

```
p3 = double(eig(H3));
if p3(1) ==0 || p3(2)==0
Type_Of_Critical_Points{3,1} = 'Critical point is degenerate'
elseif p3(1) > 0 && p3(2) >0
Type_Of_Critical_Points{3,1} = 'Critical Point is Minimum'
elseif p3(1) <0 && p3(2) <0
Type_Of_Critical_Points{3,1} = 'Critical Point is Maximum'
else
Type_Of_Critical_Points{3,1} = 'Critical Point is Saddle'
end
```

```
p3 =

  -10.8440
   35.7147


Type_Of_Critical_Points =

  3x1 cell array

    {'Critical Point is Minimum'}
    {'Critical Point is Saddle' }
    {'Critical Point is Saddle' }
```

```
p4 = double(eig(H4));
if p4(1)==0 || p4(2)==0
Type_Of_Critical_Points{4,1} = 'Critical point is degenerate';
elseif p4(1) > 0 && p4(2) >0
Type_Of_Critical_Points{4,1} = 'Critical Point is Minimum';
elseif p4(1) <0 && p4(2) <0
Type_Of_Critical_Points{4,1} = 'Critical Point is Maximum';
else
Type_Of_Critical_Points{4,1} = 'Critical Point is Saddle';
end
```

```
p4 =

  -35.8766
  -15.0753


Type_Of_Critical_Points =

  4x1 cell array

    {'Critical Point is Minimum'}
    {'Critical Point is Saddle' }
    {'Critical Point is Saddle' }
    {'Critical Point is Maximum'}
```

```
Critical_Points_Cordinate_x = [p1(1);p2(1);p3(1);p4(1)];
Critical_Points_Cordinate_y = [p1(2);p2(2);p3(2);p4(2)];
Type_Of_Critical_Points = cellstr(Type_Of_Critical_Points);

Critical_Points_of_function=table(Critical_Points_Cordinate_x,Critical_Points_Cord
inate_y,Type_Of_Critical_Points)
```

Output:

Critical_Points_Cordinate_x	Critical_Points_Cordinate_y	Type_Of_Critical_Points
8.2095	27.84	{'Critical Point is Minimum'}
-16.666	16.475	{'Critical Point is Saddle' }
-10.844	35.715	{'Critical Point is Saddle' }
-35.877	-15.075	{'Critical Point is Maximum'}