NUMERICAL ANALYSIS

PROJECT – 6

Name – Mrinal Saini **Roll No. – BS1920** **Semester – II**

QR Decomposition: For any nxp matrix X with $\boldsymbol{n \geq p}$ we have an nxn orthogonal matrix Q and an nxp upper triangular matrix R such that $X = QR$. Such a decomposition is called QR Decomposition.

Householder's Transformation: If A is any orthogonal matrix, then we know that $||Ax|| = ||x||$. In other words, an orthogonal matrix does not change shape or size of an object. It can only rotate and reflect it. Now, the question is whether the reverse is true or not? Or in other words, if $x \neq y$ are two vectors of same length, then does there exist an orthogonal matrix A that takes x to y and vice versa? That is, we are looking for an orthogonal A such that $Ax = y$ and $Ay = x$?

The answer is ''Yes.'' In fact, there may be many. Householder's transformation is one such:

$$\boldsymbol{A = I - 2uu'}, \quad \text{where } u = unit(x - y)$$

Using Householder's Transformation for QR Decomposition: The idea is to shave the columns of X one by one by multiplying with Householder matrices. For any non-zero vector u define H_u as the Householder matrix that reduces u to

$$\boldsymbol{v} = \begin{bmatrix} ||\boldsymbol{u}||^2 \\ \boldsymbol{0} \\ \vdots \\ \boldsymbol{0} \end{bmatrix}$$

Let the first column of X be a. Let H_1 shave its lower $n - 1$ entries. Consider the second column b of H_1X. Let H_2 shave off its last $n - 2$ entries. Next c denote the third column of H_2H_1X and so on. Proceeding in this way we get $H_1, H_2, \dots \dots, H_p$ all of which are orthogonal Householder matrices. Define $Q = (H_1H_2 \dots H_p)'$ and $R = Q'X$ to get a QR decomposition.

Efficient Implementation: Notice that though the Householder matrix

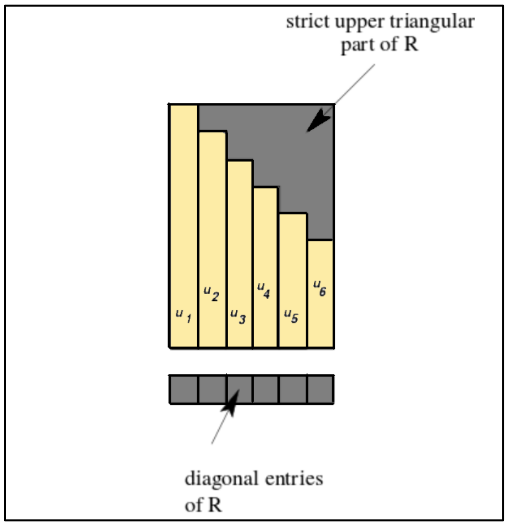
$$\boldsymbol{I - 2uu'}$$

is an nxn matrix, it is actually determined by only n numbers. Thus, we can effectively store the matrix in linear space. In particular, the matrix H_1 needs only n spaces, H_2 needs only $n - 1$ spaces and so on. So, we shall try to store these in the ''shaved'' parts of X. Let $H_1 = I - 2u_1u_1{}'$ and H_1X be partitioned as

$$\begin{bmatrix} \boldsymbol{\alpha} & \boldsymbol{v'} \\ \boldsymbol{0} & \boldsymbol{X_1} \end{bmatrix}$$

We shall try to store u_1 in place of the $0's$. But u_1 is an nx1 vector, while we have only $n - 1$ zeroes. So, the standard practice is to store α (which is the squared norm of the first column) in a separate array, and store u_1 in place of the first column of H_1X. The final output will be a nxp matrix and a p-

dimensional vector. The matrix is packed with the $u's$ and the strictly upper triangular part of R while the p-dimensional vector contains the diagonal entries of R.



It is possible to "'unpack" Q from the $u's$, however, if we need Q only to multiply some x to get Qx, then even this "unpacking" is not necessary as $Q = (H_1H_2 \dots H_p)'$ and $H_i = I - 2u_iu_i'$ and since, $H_i's$ are symmetric, therefore, $Q = H_p \dots H_2H_1$.

Application to Least Squares: An important use of the QR decomposition is in solving least squares problems. Consider a (possibly inconsistent) system

$$\boldsymbol{Ax = b}$$

If A is a full column rank (need not be square) then this system has a unique least square solution given by

$$\boldsymbol{x = (A'A)^{-1}A'b}$$

However, computing the inverse directly and then performing matrix multiplication is not an efficient algorithm. A better way is to first form a QR decomposition of A as $A = QR$. The given system now looks like

$$\boldsymbol{QRx = b}$$

The lower part of R is made of zeroes:

$$\boldsymbol{Q\begin{bmatrix} R_1 \\ O \end{bmatrix} x = b}$$

Or

$$\boldsymbol{\begin{bmatrix} R_1 \\ O \end{bmatrix} x = Q'b}$$

Partition $Q'b$ appropriately to get

$$\boldsymbol{\begin{bmatrix} R_1 \\ O \end{bmatrix} x = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}}$$

Where c_1 is px1. This system is made of two systems:

$$\boldsymbol{R_1x = c_1}$$

And

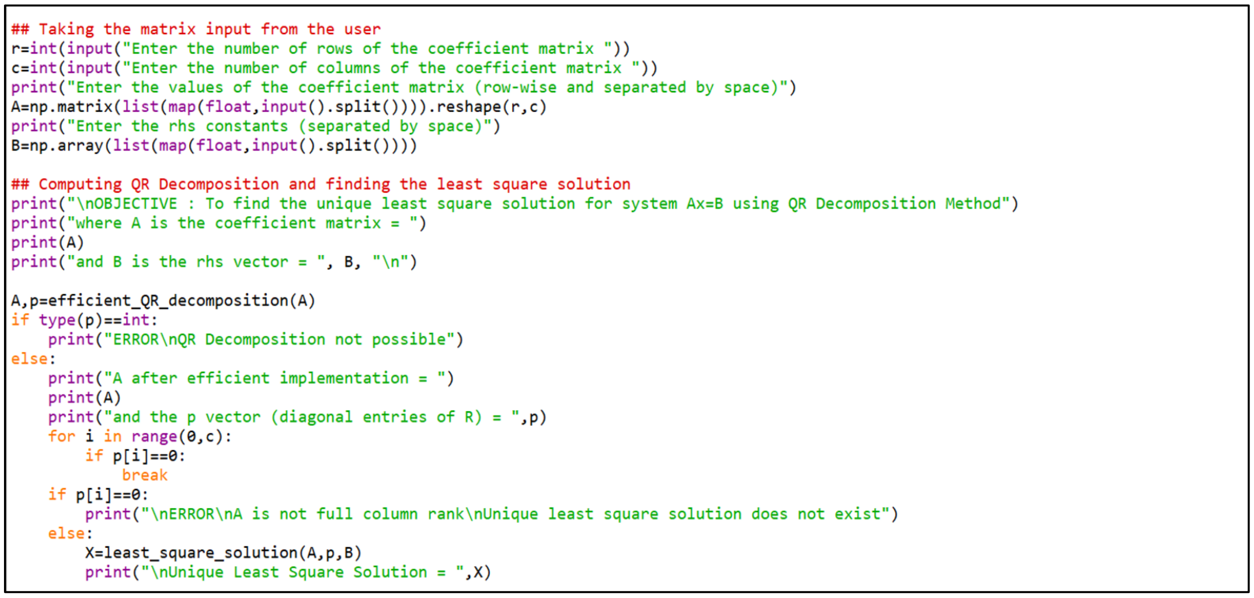
$$Ox = c_2$$

The first system is always consistent and can be solved by back-substitution to find the least square solution. The second system is trivial.

Project: Write a program to find a least squares solution to the system $AX = B$. Your program should first implement the efficient version of QR decomposition of A. Your program should be able to detect if A is not full column rank, in which case it should stop with an error message. If A is full column rank, then the program should output the unique least squares solution. Your program must never compute any Householder matrix explicitly.

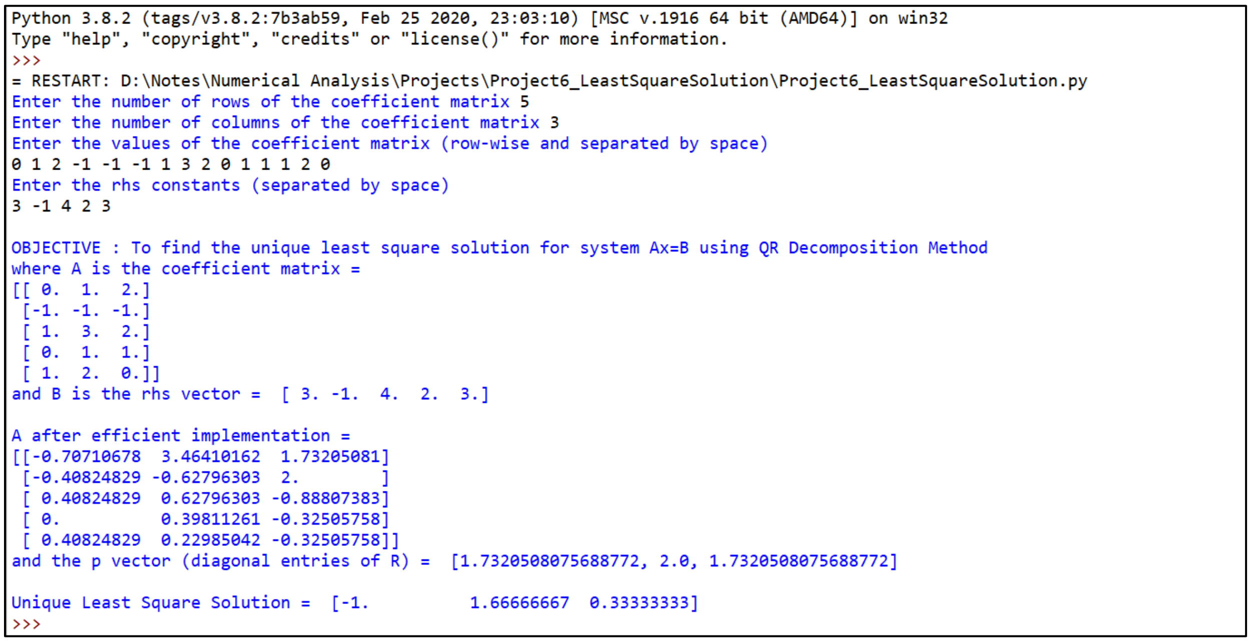
Python Code:

```
# PROJECT - 6

## Importing Libraries
import numpy as np

## Efficient QR Decomposition
def efficient_QR_decomposition(A):
    r,c=A.shape
    if r<c:
        return A,0
    else:
        p=[]
        for j in range(0,c):
            s=0
            for i in range(j,r):
                s = s + A[i,j]**2
            s=round(s,10)
            p.append(s**0.5)
            if s!=0:
                A[j,j] = A[j,j] - p[j]
                s=0
                for i in range(j,r):
                    s = s + A[i,j]**2
                s=s**0.5
                for i in range(j,r):
                    A[i,j] = A[i,j]/s

                for k in range(j+1,c):
                    s=0
                    for i in range(j,r):
                       s = s + A[i,j]*A[i,k]
                    for i in range(j,r):
                        A[i,k] = A[i,k] - 2*s*A[i,j]
        for i in range(0,r):
            for j in range(0,c):
                A[i,j]=round(A[i,j],10)

        return A,p

## Unique Least Square solution
def least_square_solution(A,p,B):
    r,c=A.shape
    X=np.array([])
    for j in range(0,c):
        s=0
        for i in range(j,r):
            s = s + A[i,j]*B[i]
        for i in range(j,r):
            B[i] = B[i] - 2*s*A[i,j]

    for i in range(0,c):
        X=np.append(X,[B[i]])

    for i in range(c-1,-1,-1):
        for k in range(c-1,i,-1):
            X[i] = X[i] - X[k]*A[i,k]
        X[i] = X[i]/p[i]

    return X
```



```python
## Taking the matrix input from the user
r=int(input("Enter the number of rows of the coefficient matrix "))
c=int(input("Enter the number of columns of the coefficient matrix "))
print("Enter the values of the coefficient matrix (row-wise and separated by space)")
A=np.matrix(list(map(float,input().split()))).reshape(r,c)
print("Enter the rhs constants (separated by space)")
B=np.array(list(map(float,input().split())))

## Computing QR Decomposition and finding the least square solution
print("\nOBJECTIVE : To find the unique least square solution for system Ax=B using QR Decomposition Method")
print("where A is the coefficient matrix = ")
print(A)
print("and B is the rhs vector = ", B, "\n")

A,p=efficient_QR_decomposition(A)
if type(p)==int:
    print("ERROR\nQR Decomposition not possible")
else:
    print("A after efficient implementation = ")
    print(A)
    print("and the p vector (diagonal entries of R) = ",p)
    for i in range(0,c):
        if p[i]==0:
            break
    if p[i]==0:
        print("\nERROR\nA is not full column rank\nUnique least square solution does not exist")
    else:
        X=least_square_solution(A,p,B)
        print("\nUnique Least Square Solution = ",X)
```

Output:



```
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 23:03:10) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: D:\Notes\Numerical Analysis\Projects\Project6_LeastSquareSolution\Project6_LeastSquareSolution.py
Enter the number of rows of the coefficient matrix 5
Enter the number of columns of the coefficient matrix 3
Enter the values of the coefficient matrix (row-wise and separated by space)
0 1 2 -1 -1 -1 1 3 2 0 1 1 1 2 0
Enter the rhs constants (separated by space)
3 -1 4 2 3

OBJECTIVE : To find the unique least square solution for system Ax=B using QR Decomposition Method
where A is the coefficient matrix =
[[ 0.  1.  2.]
 [-1. -1. -1.]
 [ 1.  3.  2.]
 [ 0.  1.  1.]
 [ 1.  2.  0.]]
and B is the rhs vector =  [ 3. -1.  4.  2.  3.]

A after efficient implementation =
[[-0.70710678  3.46410162  1.73205081]
 [-0.40824829 -0.62796303  2.        ]
 [ 0.40824829  0.62796303 -0.88807383]
 [ 0.          0.39811261 -0.32505758]
 [ 0.40824829  0.22985042 -0.32505758]]
and the p vector (diagonal entries of R) =  [1.7320508075688772, 2.0, 1.7320508075688772]

Unique Least Square Solution =  [-1.          1.66666667  0.33333333]
>>>
```