QR Decomposition with Gram-Schmidt

The **QR decomposition** (also called the QR factorization) of a matrix is a decomposition of the matrix into an orthogonal matrix and a triangular matrix. A QR decomposition of a real square matrix A is a decomposition of A as

$$A = QR,$$

where Q is an orthogonal matrix (i.e. QTQ = I) and R is an upper triangular matrix. If A is nonsingular, then this factorization is unique. There are several methods for computing the QR decomposition. One of such method is the Gram-Schmidt process.

QR Decomposition Applications:

1. Used in computer codes to find eigen values.
2. Used to solve linear systems.
3. solution of the linear least squares

QR Decomposition is widely used in quantitative finance as the basis for the solution of the linear least squares problem, which itself is used for statistical regression analysis.

Key benefits of using QR Decomposition over other methods for solving linear least squares is that it is more numerically stable

*Below is the program and its output files attached. This code I have tested with 2*2, 3*3, 4*4*

```
import numpy as np

def QR_Decomposition(A):
    n, m = A.shape  # get the shape of A
    print("--------------------------------------------------------")
    print(f"Given matrix shape is : {n}x{m}")

    # init Q and u
    Q = np.empty((n, n))
    u = np.empty((n, n))

    # To begin, we set u=a and then normalize:
    u[:, 0] = A[:, 0]
    Q[:, 0] = u[:, 0] / np.linalg.norm(u[:, 0])
    print("--------------------------------------------------------")
    # repeat the process as per
    for i in range(1, n):
        print(f"STEP : {i}")
        print("--------------------------------------------------------")
        print(f"\n==> u{i}")
        u[:, i] = A[:, i]
        print(u)

        for j in range(i):
            u[:, i] -= (A[:, i] @ Q[:, j]) * Q[:, j]  # get each u vector

        Q[:, i] = u[:, i] / np.linalg.norm(u[:, i])  # compute each e vetor
        print(f"\n==> e{i}")
        print(Q)
        print(f"\nEND OF STEP {i}");
        print("--------------------------------------------------------\n")
```

```
    R = np.zeros((n, m))
    for i in range(n):
        for j in range(i, m):
            R[i, j] = A[:, j] @ Q[:, i]
    return Q, R


def diag_sign(A):
    "Compute the signs of the diagonal of matrix A"

    D = np.diag(np.sign(np.diag(A)))

    return D


def adjust_sign(Q, R):
    #Adjust the signs of the columns in Q and rows in R to impose positive diagonal of Q

    D = diag_sign(Q)

    Q[:, :] = Q @ D
    R[:, :] = D @ R

    return Q, R


# A = np.array([[1, -1], [4, 2]])
A = np.array([[0, -1, 1], [4, 2, 0], [3, 4, 0]])
# A = np.array([[2, 1, 3, 3], [2, 1, -1,1], [2, -1, 3, -3], [2, -1, -1, -1]])
# A = np.array([[1.0, 1.0, 0.0], [1.0, 0.0, 1.0], [0.0, 1.0, 1.0]])
# A = np.array([[1.0, 0.5, 0.2], [0.5, 0.5, 1.0], [0.0, 1.0, 1.0]])
# A = np.array([[1.0, 0.5, 0.2], [0.5, 0.5, 1.0]])

print("Input array")
print("[A] = \n", A)

Q, R = adjust_sign(*QR_Decomposition(A))

print('\n==> Q: \n', Q)
print('\n')
print('==> R: \n', R)
print('\n')
```

Output files.

2_matrix.txt 3_matrix.txt 4_matrix.txt

Out file content for 3*3 matrix

```
Input array
[A] =
 [[ 0 -1  1]
 [ 4  2  0]
 [ 3  4  0]]
------------------------------------------------------------
Given matrix shape is : 3x3
------------------------------------------------------------
STEP : 1
------------------------------------------------------------

==> u1
[[ 0.00000000e+000 -1.00000000e+000  9.07999055e-312]
 [ 4.00000000e+000  2.00000000e+000  0.00000000e+000]
 [ 3.00000000e+000  4.00000000e+000  0.00000000e+000]]

==> e1
[[ 0.00000000e+000 -4.47213595e-001  9.08001806e-312]
 [ 8.00000000e-001 -5.36656315e-001  0.00000000e+000]
 [ 6.00000000e-001  7.15541753e-001  0.00000000e+000]]

END OF STEP 1
------------------------------------------------------------

STEP : 2
------------------------------------------------------------

==> u2
[[ 0.  -1.   1. ]
 [ 4.  -1.2  0. ]
 [ 3.   1.6  0. ]]

==> e2
[[ 0.         -0.4472136   0.89442719]
 [ 0.8        -0.53665631 -0.26832816]
 [ 0.6         0.71554175  0.35777088]]

END OF STEP 2
------------------------------------------------------------


==> Q:
 [[ 0.          0.4472136   0.89442719]
 [ 0.          0.53665631 -0.26832816]
 [ 0.         -0.71554175  0.35777088]]

==> R:
 [[ 0.          0.          0.        ]
 [ 0.         -2.23606798  0.4472136 ]
 [ 0.          0.          0.89442719]]
```