Logan Wright

Brother Sinkovic

Python Assignment 3

Part I

Question 1

I compute Q^TQ to show that Q is orthonormal.

In [9]:
```
from numpy import array, matmul
from numpy.linalg import qr

# Define the matrix A
A = array([[-10, 13, 7], [2, 1, -5], [-6, 3, 13], [2, 1, -5]], dtype = float)

# Build the qr matrices from A
Q, R = qr(A)

# Multiply the transposed matrix by the untransposed matrix Q
B = matmul(Q.transpose(), Q)
B[abs(B) < 1e-12] = 0.0

# Print out B where all small values become 0 to eliminate the computer error
print(B)
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2 (Intel(R) SSE4.2) enabled only processors has been deprecated. Intel oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced Vector Extensions (Intel(R) AVX) instructions.
```

Question 2

Now, we do the same thing again but in the opposite order as in QQ^T.

In [10]:
```
# Multiply the untransposed matrix by the transposed matrix Q
B = matmul(Q, Q.transpose())
B[abs(B) < 1e-12] = 0.0

# Print out B where all small values become 0 to eliminate the computer error
print(B)
```

```
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2 (Intel(R) SSE4.2) enabled only processors has been deprecated. Intel oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced Vector Extensions (Intel(R) AVX) instructions.
[[1.  0.  0.  0. ]
 [0.  0.5 0.  0.5]
 [0.  0.  1.  0. ]
 [0.  0.5 0.  0.5]]
```

Question 3

Now, I will use an iterative method to solve for the eigenvalues of a matrix, A, by using QR factorization over 100 steps to force A_n to become a triangular matrix where, simply, its eigenvalues are the diagonal entries. The process looks like

$$A_n = Q_{n-1}^T A_{n-1} Q_{n-1}$$

And this repeats 100 times.

In [12]:
```
# Build a function to do the qr factorization and return a new matrix A_n
def EigenValues(A):
    Q, _ = qr(A)
    A = matmul(Q.transpose(), matmul(A, Q))
    return A

# Define the matrix A for operation
A = array([[-10, 13, 7], [2, 1, -5], [-6, 3, 13]], dtype = float)

# Iterate over 100 steps to converge to eigenvalues
for _ in range(1, 101):
    A = EigenValues(A)

A[abs(A) < 1e-12] = 0.0

print(A)
```

```
[[ 1.13437569e+01 -6.40729937e+00  8.98940335e+00]
 [ 5.82899411e-10 -8.78821488e+00  1.52360475e+01]
 [ 0.00000000e+00  0.00000000e+00  1.44445799e+00]]
```

Question 4

According to the convergence of the matrix, A, I see that the eigenvalues are approximately

$$\lambda = 11.3438, -8.78821, 1.44446$$

Part II

Question 5

Now, I will use the NumPy function, eig, to find the more accurate eigenvalues from the same original matrix as above A.

In [13]:
```
from numpy.linalg import eig

# Redefine the matrix A for operation since the last one was overwritten
A = array([[-10, 13, 7], [2, 1, -5], [-6, 3, 13]], dtype = float)

E, P = eig(A)

print(E)
```

```
[-8.78821488 11.34375689  1.44445799]
```

Part III

Question 6

I will create an orthonormal matrix, U, which agrees on the span of H which was given in the assignment document. To generate the orthonormal columns of U, all that I will have to do is take the dot product between each column vector and divide each entry in the corresponding column by the square root of the sum of the squares (the dot product). In mathematical notation,

$$U = \left[\frac{\vec{v_1}}{||\vec{v_1}||}, \frac{\vec{v_2}}{||\vec{v_2}||}, \frac{\vec{v_3}}{||\vec{v_3}||} \right]$$

$$H = \text{Span}\ \{U\}$$

After finding the orthonormal projection matrix, $P = UU^T$, I will project the vector $\vec{y}$ onto $\text{Col}\ H$.

In [22]:
```
from numpy import sum, sqrt

# Define U as the matrix with columns v1, v2, v3
U = array([[1, 1, 0], [1, -1, 0], [0, 1, 1], [0, -1, 1], [1, 0, 1], [1, 0, -1]], dtype = float)

# Divide magnitudes from the column vectors to make orthonormal
for i in range(len(U[0])):
    U[:, i] /= sqrt(sum(U[:, i] * U[:, i]))

print("The new matrix U is orthonormal as demonstrated below.\n")
print(U)
```

```
The new matrix U is orthonormal as demonstrated below.

[[ 0.5  0.5  0. ]
 [ 0.5 -0.5  0. ]
 [ 0.   0.5  0.5]
 [ 0.  -0.5  0.5]
 [ 0.5  0.   0.5]
 [ 0.5  0.  -0.5]]
```

In [23]:
```
# Define the new matrix P
P = matmul(U, U.transpose())

print(P)
```

```
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2 (Intel(R) SSE4.2) enabled only processors has been deprecated. Intel oneAPI Math Kernel Library 2025.0 will require Intel(R) Advanced Vector Extensions (Intel(R) AVX) instructions.
[[ 0.5   0.    0.25 -0.25  0.25  0.25]
 [ 0.    0.5  -0.25  0.25  0.25  0.25]
 [ 0.25 -0.25  0.5   0.    0.25 -0.25]
 [-0.25  0.25  0.    0.5   0.25 -0.25]
 [ 0.25  0.25  0.25  0.25  0.5   0.  ]
 [ 0.25  0.25 -0.25 -0.25  0.    0.5 ]]
```

In [26]:
```
# Create the vector y
y = array([[10], [5], [9], [-1], [4], [3]], dtype = float)

# Project y onto H
projH_y = matmul(P, y)

print("The projection of y onto H\n")
print(projH_y)
```

```
The projection of y onto H

[[ 9.25]
 [ 1.75]
 [ 6.  ]
 [-1.5 ]
 [ 7.75]
 [ 3.25]]
```