Math 271.1: Exercise 2 (#3)

INSTRUCTION: QR Factorization, Rank, and Nullity

(a) Write your own QR factorization code.
(b) Determine its rank and nullity using factorization code
(c) Verify using a built-in rank computation function.

(a) QR Factorization Code

Importing Libraries needed

```
import numpy as np # for matrix operations
from IPython.display import display # for displaying arrays and matrices nicely
```

Initialize Matrix A

- creates the input matrix (3x3)
- uses float type for numerical stability

```
A = np.array([
    [1, 2, 3],
    [2, 4, 6],
    [1, 1, 0]
], dtype=float)

display(A)
```

```
array([[1., 2., 3.],
       [2., 4., 6.],
       [1., 1., 0.]])
```

Initialize Q and R matrices

Gets dimensions of A: m rows and n columns Creates and initialize zero matrices for Q (for the orthogonal matrix) and R (upper triangular)

```
m, n = A.shape
Q = np.zeros((m, n))
R = np.zeros((n, n))

display(Q, R)
```

```
array([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]])
array([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]])
```

Implement QR Factorization using Gram-Schmidt

Logic:

- For each column j:
  - Take column j of A
  - Subtract its projections onto previous orthonormal vectors
  - Normalize the result
  - Store the normalized vector in Q
  - store coefficients in R

```
for j in range(n):
    v = A[:, j].copy()

    # Subtract projections onto previous Q columns
    for i in range(j):
        R[i, j] = np.dot(Q[:, i], A[:, j])
        v = v - R[i, j] * Q[:, i]
    # Normalize
    R[j, j] = np.linalg.norm(v)

    if R[j, j] > 1e-10:
        Q[:, j] = v / R[j, j]
    else:
        Q[:, j] = v

display(Q, R)
```

```
array([[ 4.08248290e-01,  1.82574186e-01,  1.27675648e-15],
       [ 8.16496581e-01,  3.65148372e-01,  2.55351296e-15],
       [ 4.08248290e-01, -9.12870929e-01, -4.88498131e-15]])
array([[2.44948974e+00, 4.49073120e+00, 6.12372436e+00],
       [0.00000000e+00, 9.12870929e-01, 2.73861279e+00],
       [0.00000000e+00, 0.00000000e+00, 5.65805425e-15]])
```

```
print("Q matrix (orthonormal):\n")
print(Q)
```

```
Q matrix (orthonormal):

[[ 4.08248290e-01  1.82574186e-01  1.27675648e-15]
 [ 8.16496581e-01  3.65148372e-01  2.55351296e-15]
 [ 4.08248290e-01 -9.12870929e-01 -4.88498131e-15]]
```

```
print("\nR matrix (upper triangular):\n")
print(R)
```

```

R matrix (upper triangular):

[[2.44948974e+00 4.49073120e+00 6.12372436e+00]
 [0.00000000e+00 9.12870929e-01 2.73861279e+00]
 [0.00000000e+00 0.00000000e+00 5.65805425e-15]]
```

(b) Calculate Rank and Nullity

- Use diagonal elements of R to determine rank
- Numbers below tolerance (1e-10) are considered zero
    - 1e-10 provides a good balance for double-precision floating-point numbers not too large or too small
- Calculate nullity

```
tol=1e-10

diagonal = np.abs(np.diag(R))
rank = np.sum(diagonal > tol)

print("rank:", rank)
print("nullity:", A.shape[1] - rank)

print(f"NumPy rank: {np.linalg.matrix_rank(A)}")
```

```
rank: 2
nullity: 1
NumPy rank: 2
```

(c) Verify using Numpy

- Compare with the calculated rank

```
print("Verification:", np.allclose(rank, np.linalg.matrix_rank(A)))
```

```
Verification: True
```