Linear Algebra and Matrix Theory

Assignment III

Submitted By:

Ankit Shrestha

Submitted To:

Samriddha Pathak

Deadline: July 21, 2025

Table of Contents

1. Define a linear transformation. Show whether the function T(x, y) = (2x, 3y) is a linear transformation by checking the two required properties. 4

2. Write a NumPy function that checks if a given transformation matrix satisfies additivity and scalar multiplication preservation (i.e., linearity).....5

3. Classify the following system as consistent/inconsistent and dependent/independent: (x + 2y = 3 , 2x + 4y = 6 6

4. Solve the system of equations using both NumPy's np.linalg.solve and np.linalg.lstsq. Compare the results: (2x + 3y = 8 5x + 4y = 13 7

Given the matrix $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, compute the following using NumPy:

- Transpose
- Inverse
- Determinant
- Verify that $AA^{-1} \approx I$

5. 8

6. Perform LU decomposition on a 3×3 matrix using SciPy. Interpret the resulting matrices L, U, P and describe their utility in solving linear systems. 9

Solve the following overdetermined system using QR decomposition:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

7. Use NumPy to perform decomposition and back-substitution. 10

8. Explain the geometric interpretation of consistent and inconsistent linear systems. Create and solve one example of each using NumPy, then visualize in 2D using matplotlib.....10

9. Why is matrix invertibility important in solving linear systems? Give an example of a non-invertible matrix and interpret the result in terms of system solutions. 11

10. Write a Python script using NumPy that classifies a given system AX = b as: • Consistent with a unique solution • Consistent with infinite solutions • Inconsistent 12

11. Explain the difference between "basis of a vector space" and "basis of a column space" with a concrete example.....12

12. Use NumPy to check whether the following vectors form a basis for R 3: v1 = [1, 0, 0] v2 = [0, 1, 0] v3 = [0, 0, 1].....13

13. State and explain the rank-nullity theorem. Provide a matrix example with full explanation. 13

14. Compute the rank of the following matrix using NumPy: A = [[1, 2, 3], [2, 4, 6], [1, 1, 1]] Explain why the rank is less than 3. 15

15. Prove that the rank of a matrix equals the number of pivot columns in its row echelon form. Illustrate with an example matrix. 16

16. Construct a 3×3 matrix of rank 1. Use NumPy to verify that it has only one linearly independent column. 17

17. Write a Python function to generate 10 random 4×4 matrices. For each, compute its rank and determine how many are full rank. Report the percentage. ..18

18. Prove that any n linearly independent vectors in Rn form a basis. Verify this numerically with three vectors in R3. 19

19.

Consider the matrix:

$$A = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

Find the rank of A and determine which columns form a basis for its column space. ..20

1. Define a linear transformation. Show whether the function T(x, y) = (2x, 3y) is a linear transformation by checking the two required properties.

Ans: A linear transformation is a function between two vector spaces that preserves two properties:

1. Additivity: T(u + v) = T(u)+T(v)
2. Homogeneity (scalar multiplication): T(cu)=c*T(u)

If both hold for all vectors u, v and all scalars c, the transformation is linear.

Let u = (x1, y1) and v = (x2, y2) and scalar c.

1. Additivity

- T(u + v) = T(x1+x2, y1+y2) = (2(x1+x2), 3(y1+y2)) = (2x1+2x2, 3y1+3y2)
- T(u)+T(v) = (2x1,3y1) + (2x2,3y2) = (2x1+2x2, 3y1+3y2)
  Thus, additivity holds true.

2. Homogeneity:

- T(cu) = T(cx1, cy1) = (2cx1, 3cy1)
- cT(u) = c(2x1, 3y1) = (2cx1, 3cy1)
  Thus, homogeneity holds true.

2. Write a NumPy function that checks if a given transformation matrix satisfies additivity and scalar multiplication preservation (i.e., linearity).

```python
import numpy as np
def check_linearity(A):
    x = np.array([4, 6])
    y = np.array([5, 2])
    c = 3

    T_x_plus_y = A @ (x + y)
    T_x_plus_T_y = (A @ x) + (A @ y)

    T_cx = A @ (c * x)
    c_T_x = c * (A @ x)

    additivity = np.allclose(T_x_plus_y, T_x_plus_T_y)
    homogeneity = np.allclose(T_cx, c_T_x)
    return additivity and homogeneity


A = np.array([[2, 0],
              [0, 3]])

print(f"Is linear?: {check_linearity(A)}")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\DELL\Desktop\AIML class> python k.py
Is linear?: True
PS C:\Users\DELL\Desktop\AIML class>
```

3. Classify the following system as consistent/inconsistent and dependent/independent: (x + 2y = 3 , 2x + 4y = 6

k.py > ...

```python
import numpy as np

A = np.array([[1, 2],
              [2, 4]])
b = np.array([3, 6])

rA = np.linalg.matrix_rank(A)
rAb = np.linalg.matrix_rank(np.c_[A, b])

if rA == rAb:
    if rA == A.shape[1]:
        print("Unique solution")
    else:
        print("Infinite solutions")
else:
    print("No solution")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\DELL\Desktop\AIML class> python k.py
Infinite solutions
```

4. Solve the system of equations using both NumPy's np.linalg.solve and np.linalg.lstsq. Compare the results: (2x + 3y = 8 5x + 4y = 13

k.py > ...

```python
import numpy as np

A = np.array([[2, 3],
              [5, 4]])
b = np.array([8, 13])

# Direct solve
sol = np.linalg.solve(A, b)
print("Solution using solve:", sol)

# Least squares
sol2, _, _, _ = np.linalg.lstsq(A, b, rcond=None)
print("Solution using lstsq:", sol2)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\DELL\Desktop\AIML class> python k.py
Solution using solve: [1. 2.]
Solution using lstsq: [1. 2.]
PS C:\Users\DELL\Desktop\AIML class>
```

5. Given the matrix $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, compute the following using NumPy:

- Transpose
- Inverse
- Determinant
- Verify that $AA^{-1} \approx I$

```python
import numpy as np
A = np.array([[1, 2],
              [3, 4]])

print(f"Transpose:\n {A.T}")
print(f"Inverse:\n {np.linalg.inv(A)}")
print(f"Determinant: {np.linalg.det(A)}")

I = A @ np.linalg.inv(A)
print(f"A * A^-1:\n {I}")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\DELL\Desktop\AIML class> python k.py
Transpose:
 [[1 3]
 [2 4]]
Inverse:
 [[-2.   1. ]
 [ 1.5 -0.5]]
Determinant: -2.0000000000000004
A * A^-1:
 [[1.0000000e+00 0.0000000e+00]
 [8.8817842e-16 1.0000000e+00]]
PS C:\Users\DELL\Desktop\AIML class>
```

6. Perform LU decomposition on a 3×3 matrix using SciPy. Interpret the resulting matrices L, U, P and describe their utility in solving linear systems.

k.py > ...

```python
import numpy as np
from scipy.linalg import lu

A = np.array([[2, 3, 1],
              [4, 7, 7],
              [6, 18, 22]])

P, L, U = lu(A)

print("P (Permutation matrix):\n", P)
print("L (Lower triangular):\n", L)
print("U (Upper triangular):\n", U)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\DELL\Desktop\AIML class> python k.py
P (Permutation matrix):
 [[0. 0. 1.]
 [0. 1. 0.]
 [1. 0. 0.]]
L (Lower triangular):
 [[1.         0.         0.        ]
 [0.66666667 1.         0.        ]
 [0.33333333 0.6        1.        ]]
U (Upper triangular):
 [[ 6.         18.         22.        ]
 [ 0.         -5.         -7.66666667]
 [ 0.          0.         -1.73333333]]
PS C:\Users\DELL\Desktop\AIML class>
```

Solve the following overdetermined system using QR decomposition:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

7. Use NumPy to perform decomposition and back-substitution.

```
k.py > ...
1   import numpy as np
2
3   A = np.array([[1, 2],
4                 [3, 4],
5                 [5, 6]])
6   b = np.array([1, 2, 3])
7
8   Q, R = np.linalg.qr(A)
9
10  x = np.linalg.solve(R, Q.T @ b)
11  print("Solution:", x)
12
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\DELL\Desktop\AIML class> python k.py
Solution: [1.50129554e-16 5.00000000e-01]
PS C:\Users\DELL\Desktop\AIML class>
```

8. Explain the geometric interpretation of consistent and inconsistent linear systems. Create and solve one example of each using NumPy, then visualize in 2D using matplotlib.

9. Why is matrix invertibility important in solving linear systems? Give an example of a non-invertible matrix and interpret the result in terms of system solutions.

If a matrix is invertible, we can find a unique solution for AX=b as X=A−1b.
If not invertible (singular), the system has no solution or infinite solutions.

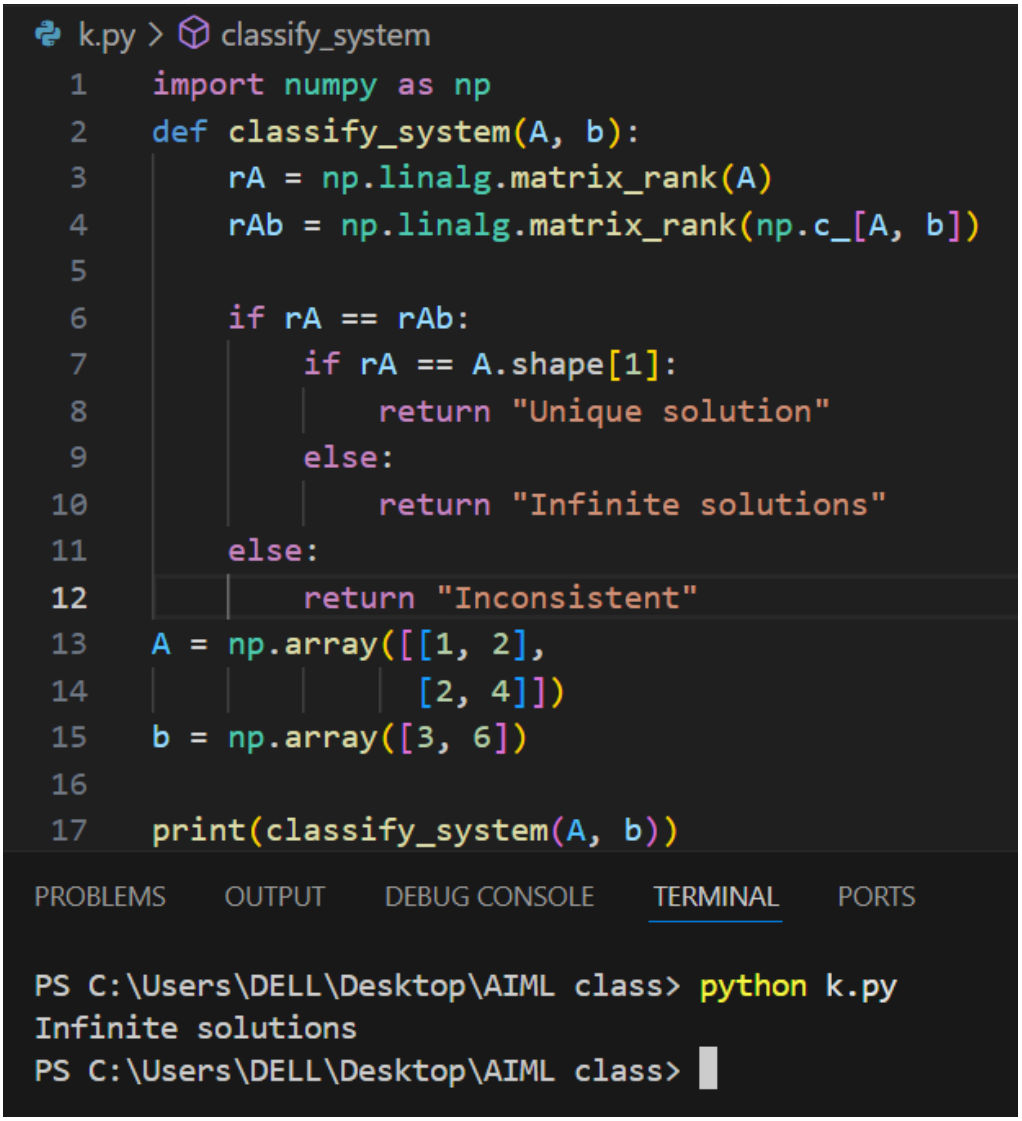
k.py > ...

```python
import numpy as np
A = np.array([[1, 2],
              [2, 4]])
b = np.array([3, 6])

print("Determinant:", np.linalg.det(A))
try:
    sol = np.linalg.solve(A, b)
    print("Solution:", sol)
except np.linalg.LinAlgError:
    print("Matrix not invertible (no unique solution)")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\DELL\Desktop\AIML class> python k.py
Determinant: 0.0
Matrix not invertible (no unique solution)
PS C:\Users\DELL\Desktop\AIML class>
```

10. Write a Python script using NumPy that classifies a given system AX = b as:
    • Consistent with a unique solution • Consistent with infinite solutions • Inconsistent



11. Explain the difference between “basis of a vector space” and “basis of a column space” with a concrete example.

- Basis of a vector space: A set of independent vectors that can make every vector in that space.
  Example: (1, 0) and (0, 1) make all vectors in R2.

- Basis of a column space: A set of independent columns of a matrix that can make all other columns by combination.
  Example: In A = [[1, 2] , [0, 0]], only the first column is needed as a basis.

12. Use NumPy to check whether the following vectors form a basis for R 3: v1 = [1, 0, 0] v2 = [0, 1, 0] v3 = [0, 0, 1]

```
k.py > ...
1   import numpy as np
2
3   A = np.array([[1,0,0],
4                 [0,1,0],
5                 [0,0,1]])
6
7   if np.linalg.matrix_rank(A) == 3:
8       print("Yes, they form a basis for R3")
9   else:
10      print("No, they don't")
11
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\DELL\Desktop\AIML class> python k.py
Yes, they form a basis for R3
PS C:\Users\DELL\Desktop\AIML class>
```

13. State and explain the rank-nullity theorem. Provide a matrix example with full explanation.

The Rank-Nullity Theorem says:

Rank(A)+Nullity(A)=Number of columns of A

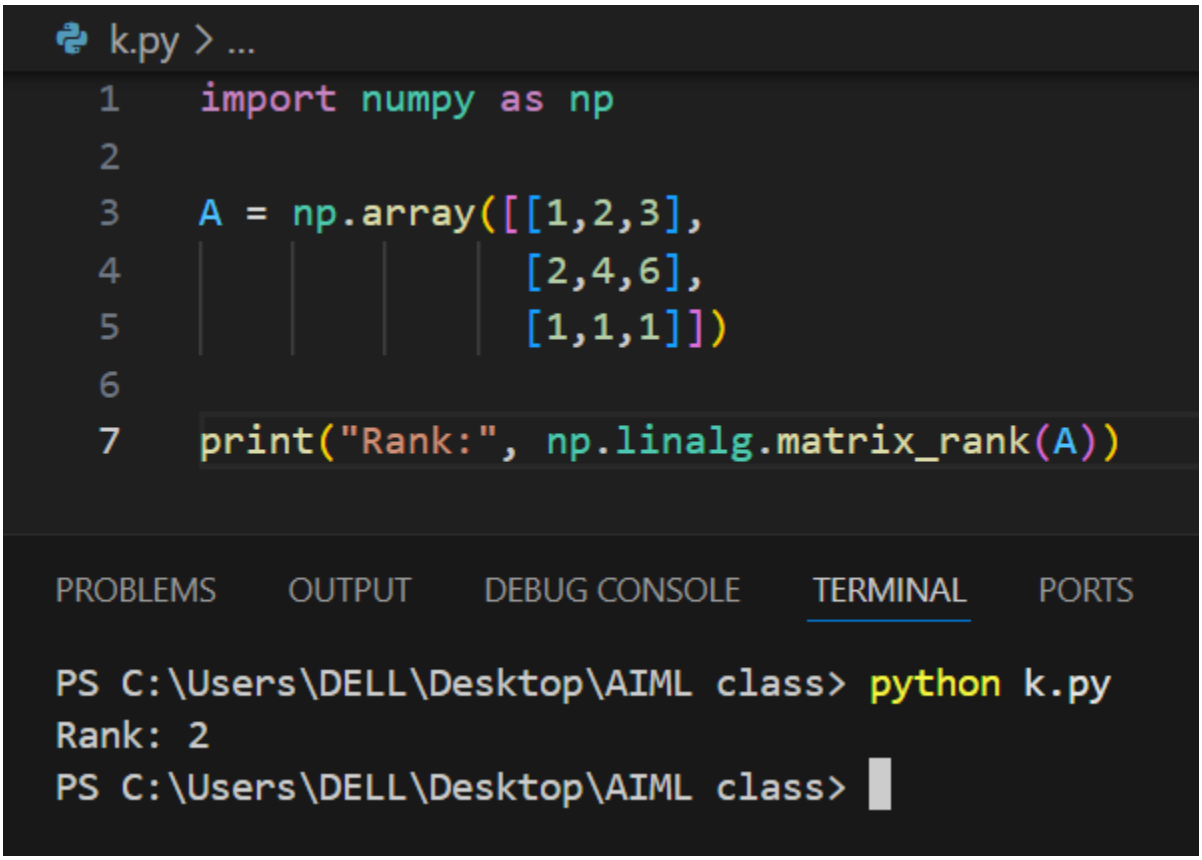
- Rank(A) = number of linearly independent columns.
- Nullity(A) = dimension of the null space (solutions to Ax = 0).

```python
import numpy as np

A = np.array([[1,2,3],
              [0,1,4]])

rank = np.linalg.matrix_rank(A)
nullity = A.shape[1] - rank
print("Rank:", rank)
print("Nullity:", nullity)
print("Check:", rank + nullity == A.shape[1])

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\DELL\Desktop\AIML class> python k.py
Rank: 2
Nullity: 1
Check: True
PS C:\Users\DELL\Desktop\AIML class>
```

14. Compute the rank of the following matrix using NumPy: A = [[1, 2, 3], [2, 4, 6], [1, 1, 1]] Explain why the rank is less than 3.



As we know, rank is the number of rows or columns who are independent. Here the second row is just the multiple of 2 of the first row. Thus, the rank is 2 and not 3.

15. Prove that the rank of a matrix equals the number of pivot columns in its row echelon form. Illustrate with an example matrix.

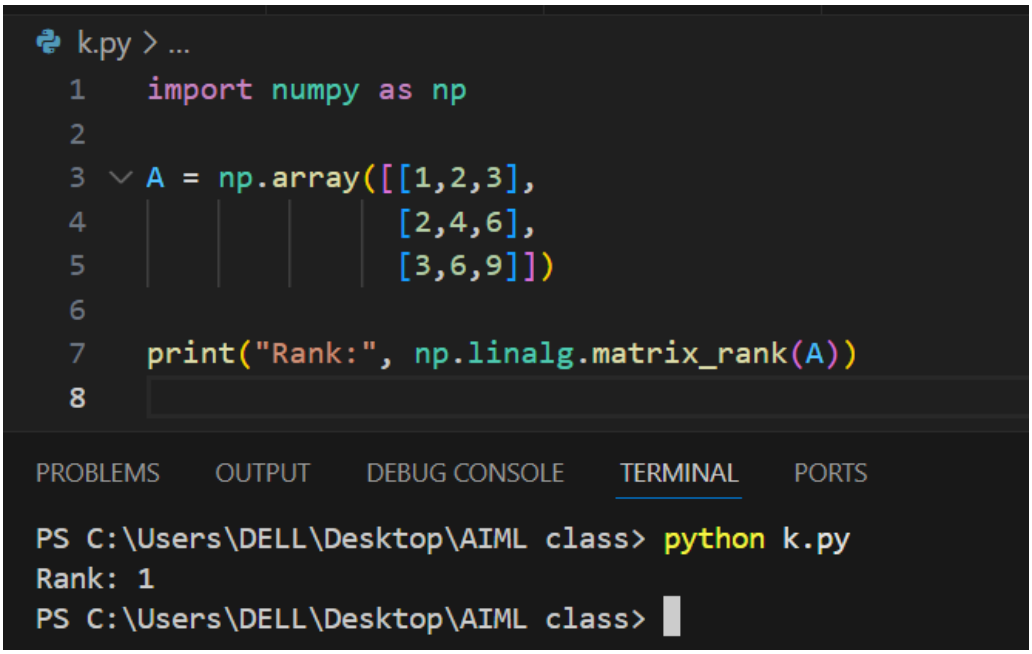
k.py > ...

```python
import numpy as np
from scipy.linalg import lu

A = np.array([[1,2,3],
              [2,4,6],
              [1,1,1]])

P, L, U = lu(A)
print("Row Echelon Form (U):\n", U)
print("Rank:", np.linalg.matrix_rank(A))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\DELL\Desktop\AIML class> python k.py
Row Echelon Form (U):
 [[ 2.  4.  6.]
 [ 0. -1. -2.]
 [ 0.  0.  0.]]
Rank: 2
PS C:\Users\DELL\Desktop\AIML class>
```

16. Construct a 3×3 matrix of rank 1. Use NumPy to verify that it has only one linearly independent column.



17. Write a Python function to generate 10 random 4×4 matrices. For each, compute its rank and determine how many are full rank. Report the percentage.

k.py > ...

```python
import numpy as np

full_rank_count = 0
total = 10

for i in range(total):
    A = np.random.randint(1, 10, (4, 4))
    r = np.linalg.matrix_rank(A)
    print(f"Matrix {i+1} Rank:", r)
    if r == 4:
        full_rank_count += 1

percentage = (full_rank_count / total) * 100
print("Full-rank matrices:", full_rank_count)
print("Percentage:", percentage, "%")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\DELL\Desktop\AIML class> python k.py
Matrix 1 Rank: 4
Matrix 2 Rank: 4
Matrix 3 Rank: 4
Matrix 4 Rank: 4
Matrix 5 Rank: 4
Matrix 6 Rank: 4
Matrix 7 Rank: 4
Matrix 8 Rank: 4
Matrix 9 Rank: 4
Matrix 10 Rank: 4
Full-rank matrices: 10
Percentage: 100.0 %
```

18. Prove that any n linearly independent vectors in Rn form a basis. Verify this numerically with three vectors in R3.

k.py > ...

```python
import numpy as np

A = np.array([[1,0,0],
              [0,1,0],
              [0,0,1]])

if np.linalg.matrix_rank(A) == 3:
    print("Yes, they form a basis for R3")
else:
    print("No, they don't")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\DELL\Desktop\AIML class> python k.py
Yes, they form a basis for R3
PS C:\Users\DELL\Desktop\AIML class>
```

Consider the matrix:

$$A = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

19. Find the rank of A and determine which columns form a basis for its column space.

k.py > ...

```python
import numpy as np

A = np.array([[1,0,1],
              [0,1,1],
              [0,1,1]])

print("Rank:", np.linalg.matrix_rank(A))

basis = []
B = []

for i in range(A.shape[1]):
    test = B + [A[:,i]]
    if np.linalg.matrix_rank(np.column_stack(test)) > len(B):
        B.append(A[:,i])
        basis.append(i+1)

print("Basis columns are:", basis)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\DELL\Desktop\AIML class> python k.py
Rank: 2
Basis columns are: [1, 2]
PS C:\Users\DELL\Desktop\AIML class>
```