1

```
import numpy as np

# Create a row vector
row_vector = np.array([1, 2, 3])
print("Row vector: ", row_vector)

# Create a column vector
col_vector = np.array([[1], [2], [3]])
print("Column vector: \n", col_vector)


Row vector:  [1 2 3]
Column vector: 
 [[1]
  [2]
  [3]]

matrix = np.array([[1, 2], [3, 4]])
print("Matrix: \n", matrix)

Matrix: 
 [[1 2]
  [3 4]]

# Transpose a row vector
transpose_row_vector = row_vector.T
print("Transpose of row vector: \n", transpose_row_vector)

# Transpose a matrix
transpose_matrix = matrix.T
print("Transpose of matrix: \n", transpose_matrix)

Transpose of row vector: 
 [1 2 3]
Transpose of matrix: 
 [[1 3]
  [2 4]]


# Conjugate transpose a row vector
```

```
conj_trans_row_vector = row_vector.conj().T
print("Conjugate transpose of row vector: \n", conj_trans_row_vector)

# Conjugate transpose a matrix
conj_trans_matrix = matrix.conj().T
print("Conjugate transpose of matrix: \n", conj_trans_matrix)
```

```
Conjugate transpose of row vector:
 [1 2 3]
Conjugate transpose of matrix:
 [[1 3]
  [2 4]]
```

2

```
import numpy as np

A = np.array([
    [1, 2, 3, 4],
    [5, 6, 7, 8],
    [9, 10, 11, 12],
    [13, 14, 15, 16]
])


# Generate the matrix into echelon form
# by solving for the matrix using np.linalg.solve
# with the augmented matrix as input
n_rows, n_cols = A.shape
for i in range(min(n_rows, n_cols)):
    # Find the row with the largest absolute value in the i-th column
    max_row = np.argmax(np.abs(A[i:, i])) + i

    # Swap the max row with the current row
    if i != max_row:
        A[[i, max_row]] = A[[max_row, i]]

    # Perform row operations to get zeros below the leading coefficient
    for j in range(i+1, n_rows):
```

```
        c = A[j, i] / A[i, i]
        A[j, i:] -= c * A[i, i:]

# Find the rank of the matrix by counting the number of non-zero rows
rank = np.sum([np.any(A[i]) for i in range(n_rows)])

print("Echelon form of A:\n", A)
print("Rank of A:", rank)
```

```
Echelon form of A:
 [[13 14 15 16]
  [ 0 -1 -2 -3]
  [ 0  0  1  2]
  [ 0  0  0 -1]]
Rank of A: 4
```

3

```
import numpy as np

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

# Find the cofactors of A
n_rows, n_cols = A.shape
cofactors = np.zeros((n_rows, n_cols))

for i in range(n_rows):
    for j in range(n_cols):
        submatrix = np.delete(np.delete(A, i, axis=0), j, axis=1)
        minor = np.linalg.det(submatrix)
        cofactors[i, j] = (-1) ** (i+j) * minor

print("Cofactors of A:\n", cofactors)
```

```
Cofactors of A:
 [[ 1.  2. -1.]
```

```
 [ 2.  3. -2.]
 [-1. -2.  1.]]
# Find the determinant of A
det = np.linalg.det(A)

print("Determinant of A:", det)
```

```
Determinant of A: 5.0
```

```
# Find the adjoint of A
adj = cofactors.T

print("Adjoint of A:\n", adj)
```

```
Adjoint of A:
 [[ 1.  2. -1.]
 [ 2.  3. -2.]
 [-1. -2.  1.]]
```

```
# Find the inverse of A
inv = adj / det

print("Inverse of A:\n", inv)
```

```
Inverse of A:
 [[ 0.2  0.4 -0.2]
 [ 0.4  0.6 -0.4]
 [-0.2 -0.4  0.2]]
```

4

2x + 3y + z = 9
x - y + 2z = 1

3x + 4y + 2z = 15

```
import numpy as np

# Matrix of coefficients
A = np.array([
    [2, 3, 1],
    [1, -1, 2],
    [3, 4, 2]
])

# Column vector of constants
b = np.array([
    [9],
    [1],
    [15]
])


def gauss_elimination(A, b):
    n = A.shape[0]
    aug = np.concatenate((A, b), axis=1)

    # Forward elimination
    for i in range(n):
        pivot = aug[i, i]
        for j in range(i+1, n):
            factor = aug[j, i] / pivot
            aug[j, :] -= factor * aug[i, :]

    # Back substitution
    x = np.zeros((n, 1))
    for i in range(n-1, -1, -1):
        x[i] = (aug[i, -1] - aug[i, :-1] @ x) / aug[i, i]

    return x



# Solve the system of equations
x = gauss_elimination(A, b)

print("Solution:")
print("x =", x[0])
```

```
print("y =", x[1])
print("z =", x[2])
```

Solution:
x = [3.]
y = [-2.]
z = [4.]

5

2x + 3y + z = 0
x - y + 2z = 0
3x + 4y + 2z = 0

```
import numpy as np

# Matrix of coefficients
A = np.array([
    [2, 3, 1],
    [1, -1, 2],
    [3, 4, 2]
])

# Column vector of zeros
b = np.zeros((3, 1))

def gauss_jordan(A):
    n = A.shape[0]
    aug = np.concatenate((A, np.eye(n)), axis=1)

    # Forward elimination
    for i in range(n):
        pivot = aug[i, i]
        aug[i, :] /= pivot
        for j in range(i+1, n):
            factor = aug[j, i]
            aug[j, :] -= factor * aug[i, :]

    # Back substitution
    for i in range(n-1, -1, -1):
        for j in range(i-1, -1, -1):
            factor = aug[j, i]
```

```
            aug[j, :] -= factor * aug[i, :]

    return aug[:, n:]


# Solve the system of homogeneous equations
x = gauss_jordan(A)

print("Solutions:")
print("x =", x[0])
print("y =", x[1])
print("z =", x[2])
```

```
Solutions:
x = [ 2.]
y = [-1.]
z = [ 1.]
```

6

```
import numpy as np

A = np.array([
    [1, 2, 1, 4],
    [2, 4, 2, 8],
    [3, 6, 3, 12]
])

# Find the basis of the column space
rref_A = np.around(np.linalg.inv(A.T @ A) @ A.T @ A, decimals=6)
basis_col_space = A[:, np.where(np.abs(rref_A) > 0)[1]]

print("Basis of column space:")
print(basis_col_space)
```

```
Basis of column space:
[[1 2]
 [2 4]
 [3 6]]
```

```
from scipy.linalg import null_space

# Find the basis of the null space
basis_null_space = null_space(A)

print("Basis of null space:")
print(basis_null_space)
```

```
Basis of null space:
[[ 0.40824829 -0.81649658  0.40824829]
 [-0.70710678  0.          0.70710678]
 [ 0.          0.          0.        ]
 [ 0.          0.          0.        ]]
```

```
# Find the basis of the row space
rref_A = np.around(np.linalg.inv(A.T @ A) @ A.T @ A, decimals=6)
basis_row_space = A[np.where(np.abs(rref_A) > 0)[0], :]

print("Basis of row space:")
print(basis_row_space)
```

```
Basis of row space:
[[1 2 1 4]
 [0 0 0 0]]
```

```
# Find the basis of the left null space
basis_left_null_space = null_space(A.T)

print("Basis of left null space:")
print(basis_left_null_space)
```

Basis of left null space:
```
[[ 0.          0.70710678]
 [-0.89442719 -0.35355339]
 [ 0.4472136  -0.53033009]
 [ 0.          0.        ]]
```