Lab 2 : Linear Algebra

Solutions of the system of equations

There are missing fields in the code that you need to fill to get the results but note that you can write you own code to obtain the results

```
## Import the required Libraries here
import numpy as np
import matplotlib.pyplot as plt
```

#Case 1 :

Consider an eqauation $A\mathbf{x}=\mathbf{b}$ where A is a Full rank and square martrix, then the solution is given as $\mathbf{x}_{op}=A^{-1}\boldsymbol{b}$, where $\mathbf{x}_{op}$ is the optimal solution and the error is given as $\boldsymbol{b}$ - $A\mathbf{x}_{op}$

Use the above information to solve the following equatation and compute the error :

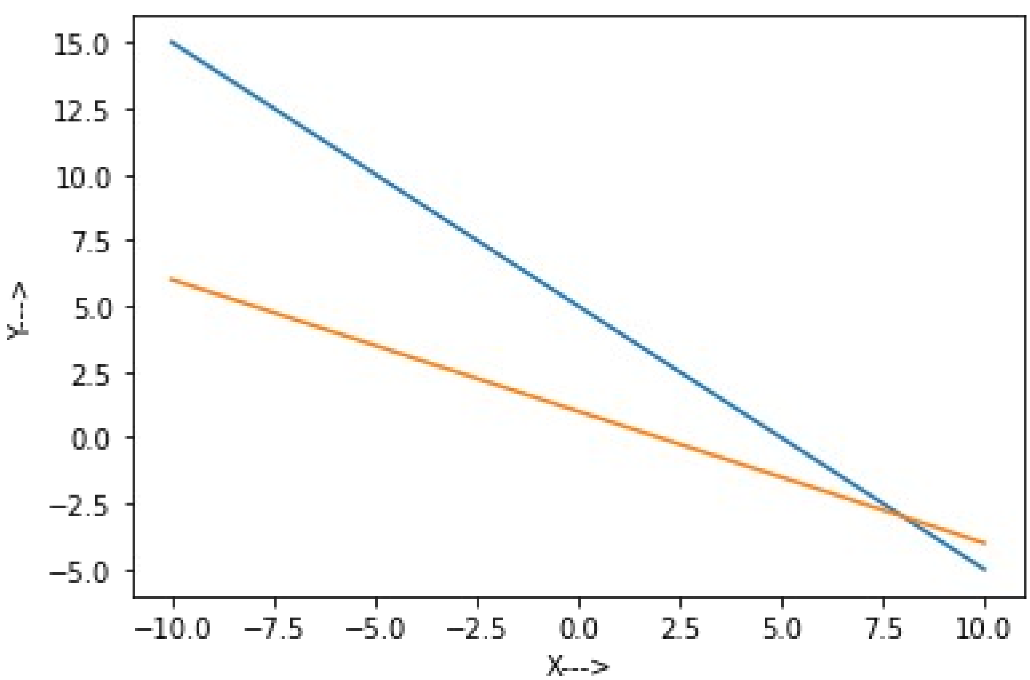
$$x+y=5$$

$$2x+4y=4$$

```
# Define Matrix A and B
A = np.array([[1, 1], [2, 4]]) # write your code here
b = np.array([5, 4]) # write your code here
print('A=',A,'\n')
print('b=',b,'\n')


# Determine the determinant of matrix A
Det = np.linalg.det(A) # write your code here
print('Determinant=',Det,'\n')

# Determine the rank of the matrix A
rank = np.linalg.matrix_rank(A) # write your code here
print('Matrix rank=',rank,'\n')

# Determine the Inverse of matrix A
A_inverse = np.linalg.inv(A) # write your code here
print('A_inverse=',A_inverse,'\n')

# Determine the optimal solution
x_op = A_inverse @ b # write your code here
print('x=',x_op,'\n')

# Plot the equations
# write your code here
x = np.linspace(-10, 10, 1000)
y1 = 5 - x
```

```
y2 = (2-x)/2
plt.xlabel("X--->")
plt.ylabel("Y--->")
plt.plot(x, y1)
plt.plot(x, y2)

# Validate the solution by obtaining the error
error = b - (A @ x_op) # write your code here
print('error=',error,'\n')
```

```
A= [[1 1]
 [2 4]]

b= [5 4]

Determinant= 2.0

Matrix rank= 2

A_inverse= [[ 2.  -0.5]
 [-1.   0.5]]

x= [ 8. -3.]

error= [0. 0.]
```



For the following equation :

$$x+y+z=5$$

$$2x+4y+z=4$$

$$x+3y+4z=4$$

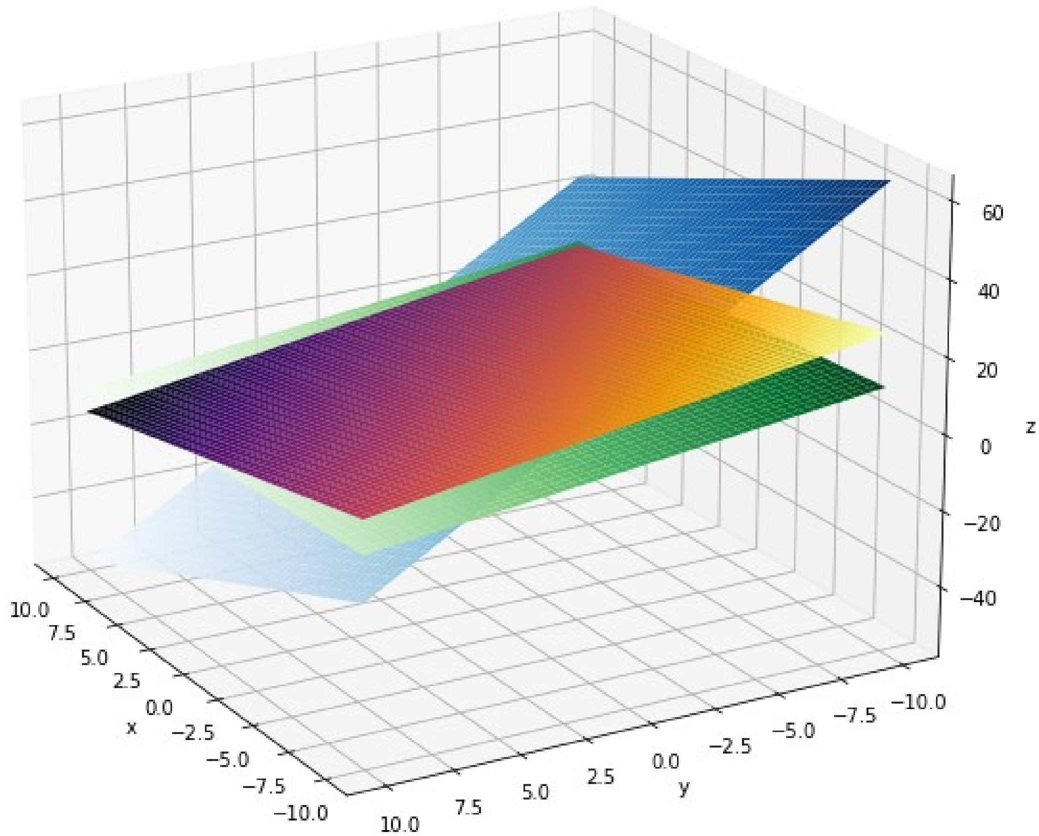
Write the code to :

1. Define Matrices A and B
2. Determine the determinant of A
3. Determine the rank of A
4. Determine the Inverse of matrix A
5. Determine the optimal solution
6. Plot the equations
7. Validate the solution by obataining error

```
## write your code here
# Define Matrix A and B
A = np.array([
    [1, 1, 1],
    [2, 4, 1],
    [1, 3, 4]
])

b = np.array([5, 4, 4])

print('A=',A,'\n')
print('b=',b,'\n')


# Determine the determinant of matrix A
Det = np.linalg.det(A) # write your code here
print('Determinant=',Det,'\n')

# Determine the rank of the matrix A
rank = np.linalg.matrix_rank(A) # write your code here
print('Matrix rank=',rank,'\n')

# Determine the Inverse of matrix A
A_inverse = np.linalg.inv(A) # write your code here
print('A_inverse=',A_inverse,'\n')

# Determine the optimal solution
x_op = A_inverse @ b # write your code here
print('x=',x_op,'\n')

# Plot the equations
# write your code here
x = np.linspace(-10, 10, 1000)
```

```
y = np.linspace(-10, 10, 1000)
X, Y = np.meshgrid(x, y)
z1 = 5 - X - Y
z2 = 4 - 2*X - 4*Y
z3 = (4 - X - 3*Y)/4

fig = plt.figure(figsize=(20, 10))
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, z1, cmap='inferno')
ax.plot_surface(X, Y, z2, cmap='Blues')
ax.plot_surface(X, Y, z3, cmap='Greens')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
ax.view_init(20, 150)

# Validate the solution by obtaining the error
error = b - (A @ x_op) # write your code here
print('error=',error,'\n')
```

```
A= [[1 1 1]
 [2 4 1]
 [1 3 4]]

b= [5 4 4]

Determinant= 7.999999999999998

Matrix rank= 3

A_inverse= [[ 1.625 -0.125 -0.375]
 [-0.875  0.375  0.125]
 [ 0.25  -0.25   0.25 ]]

x= [ 6.125 -2.375  1.25 ]

error= [0. 0. 0.]
```



#Case 2 :

Consider an eqauation $A\mathbf{x}=\mathbf{b}$ where A is a Full rank but it is not a square matrix ($m>n$, dimension of A is $m*n$) , Here if b lies in the span of columns of A then there is unique solution and it is given as $\boldsymbol{x}\square_u=A^{-1}\boldsymbol{b}$ (here A^{-1} is the pseudo inverse of matrix A), where $\boldsymbol{x}\square_u$ is the unique solution and the error is given as $\boldsymbol{b}$ - $A\boldsymbol{x}\square_u$, If b does not lie in the span of columns of A then there are no solutions and the least square solution is given as $\boldsymbol{x}\square_{ls}=A^{-1}\boldsymbol{b}$ (here A^{-1} is the pseudo inverse of matrix A) and the error is given as $\boldsymbol{b}$ - $A\boldsymbol{x}\square_{ls}$

Use the above information solve the following equations and compute the error :

$$x+z=0$$

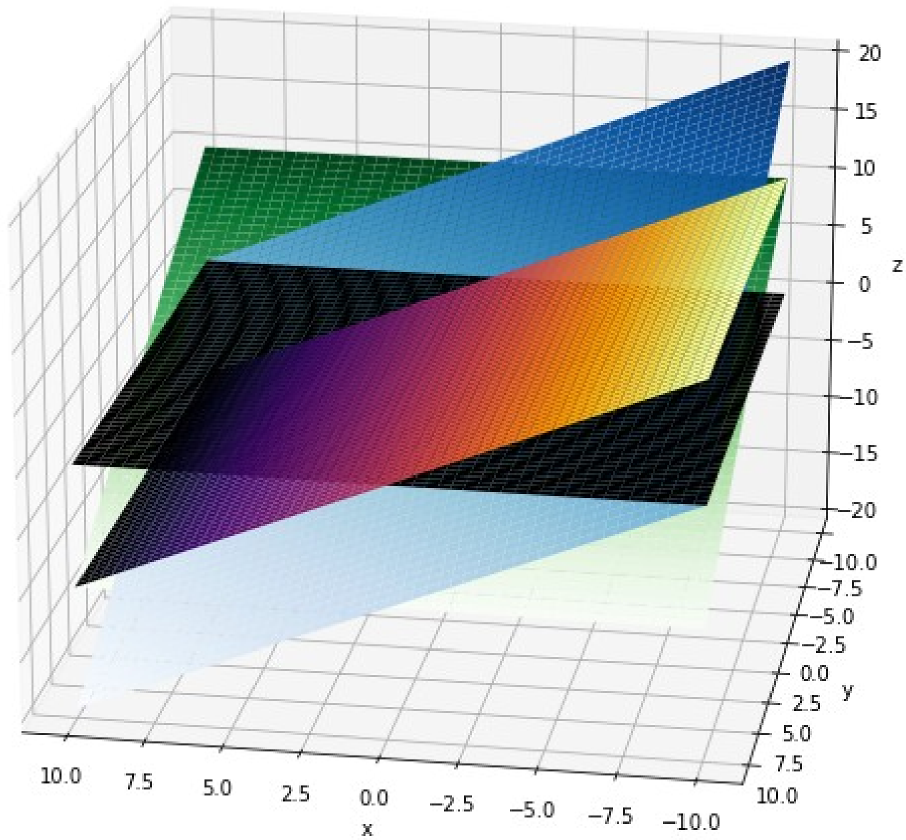
$$x+y+z=0$$

$$y+z=0$$

$$z=0$$

```python
# Define matrix A and B
A = np.array([
        [1, 0, 1],
        [1, 1, 1],
        [0, 1, 1],
        [0, 0, 1],
    ]) # write your code here
b = np.array([0, 0, 0, 0]) # write your code here
print('A=',A,'\n')
print('b=',b,'\n')

# Determine the rank of matrix A
rank = np.linalg.matrix_rank(A) # write your code here
print('Matrix rank=',rank,'\n')

# Determine the pseudo-inverse of A (since A is not Square matrix)
A_inverse = np.linalg.pinv(A) # write your code here
print('A_inverse=',A_inverse,'\n')

# Determine the optimal solution
x_op = A_inverse @ b # write your code here
print('x=',x_op,'\n')

# Plot the equations
x = np.linspace(-10, 10, 1000)
y = np.linspace(-10, 10, 1000)
X, Y = np.meshgrid(x, y)
z1 = -X
z2 = -X -Y
z3 = -Y
z4 = np.zeros((1000, 1000))
print(X.shape)

fig = plt.figure(figsize=(20, 10))
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, z1, cmap='inferno')
ax.plot_surface(X, Y, z2, cmap='Blues')
ax.plot_surface(X, Y, z3, cmap='Greens')
ax.plot_surface(X, Y, z4, cmap='copper')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
ax.view_init(20, 100)


# Validate the solution by computing the error
error = b - (A @ x_op) # write your code here
print('error=',error,'\n')
```

```
A= [[1 0 1]
 [1 1 1]
 [0 1 1]
 [0 0 1]]

b= [0 0 0 0]

Matrix rank= 3

A_inverse= [[ 0.5   0.5  -0.5  -0.5 ]
 [-0.5   0.5   0.5  -0.5 ]
 [ 0.25 -0.25  0.25  0.75]]

x= [0. 0. 0.]

(1000, 1000)
error= [0. 0. 0. 0.]
```



For the following equation :

$$x+y+z=35$$

$$2x+4y+z=94$$

$$x+3y+4z=4$$

$$x+9y+4z=-230$$

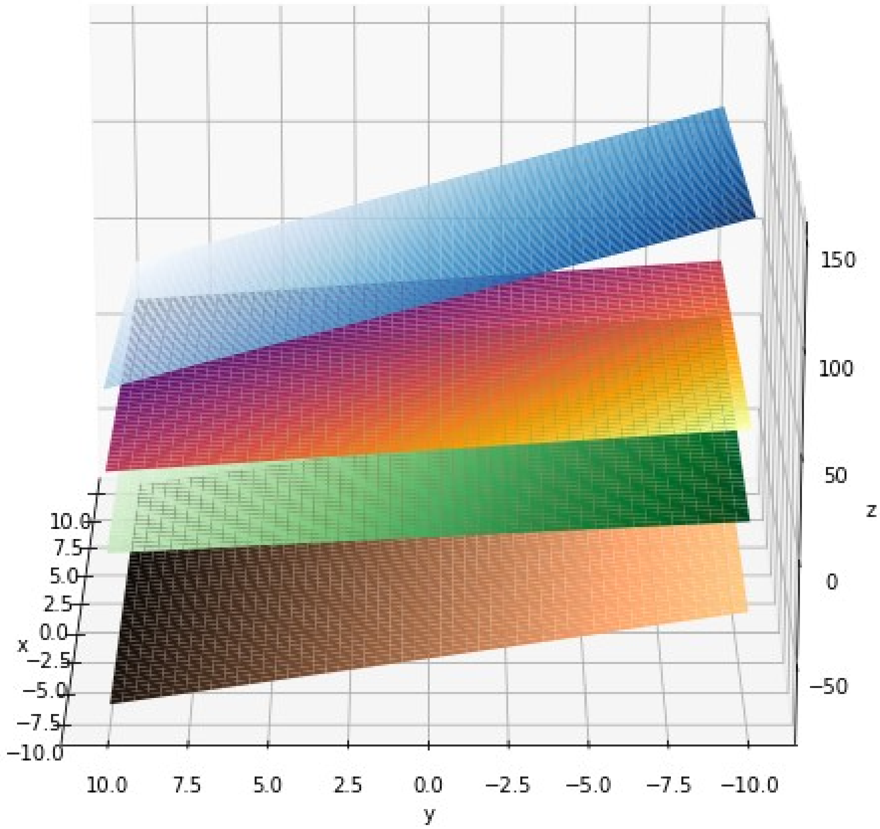
Write the code to :

1. Define Matrices A and B
2. Determine the rank of A
3. Determine the Pseudo Inverse of matrix A
4. Determine the optimal solution
5. Plot the equations

6. Validate the solution by obataining error

```
# Define matrix A and B
A = np.array([
        [1, 1, 1],
        [2, 4, 1],
        [1, 3, 4],
        [1, 9, 4],
    ]) # write your code here
b = np.array([35, 94, 4, -230]) # write your code here
print('A=',A,'\n')
print('b=',b,'\n')

# Determine the rank of matrix A
rank = np.linalg.matrix_rank(A) # write your code here
print('Matrix rank=',rank,'\n')

# Determine the pseudo-inverse of A (since A is not Square matrix)
A_inverse = np.linalg.pinv(A) # write your code here
print('A_inverse=',A_inverse,'\n')

# Determine the optimal solution
x_op = A_inverse @ b # write your code here
print('x=',x_op,'\n')

# Plot the equations
x = np.linspace(-10, 10, 1000)
y = np.linspace(-10, 10, 1000)
X, Y = np.meshgrid(x, y)
z1 = 35 -X -Y
z2 = 94 -2*X -4*Y
z3 = (4 -X -3*Y)/4
z4 = (-230 -X -9*Y)/4

fig = plt.figure(figsize=(20, 10))
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, z1, cmap='inferno')
ax.plot_surface(X, Y, z2, cmap='Blues')
ax.plot_surface(X, Y, z3, cmap='Greens')
ax.plot_surface(X, Y, z4, cmap='copper')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
ax.view_init(20, 180)


# Validate the solution by computing the error
error = b - (A @ x_op) # write your code here
print('error=',error,'\n')
```

```
A= [[1 1 1]
 [2 4 1]
 [1 3 4]
 [1 9 4]]

b= [  35   94    4 -230]

Matrix rank= 3

A_inverse= [[ 0.27001704  0.45570698  0.07666099 -0.25809199]
 [-0.06558773  0.02810903 -0.14480409  0.15417376]
 [ 0.04429302 -0.16183986  0.31856899 -0.03918228]]

x= [111.9548552  -35.69250426  -3.37649063]

error= [-37.88586031  16.23679727  12.6286201   -7.21635434]
```



#Case 3 :

Consider an eqauation $A\mathbf{x}=\mathbf{b}$ where A is not a Full rank matrix, Here if b lies in the span of columns of A then there are multiple solutions and one of the solution is given as $\mathbf{x}^{\square}_{u}=A^{-1}\boldsymbol{b}$ (here A^{-1} is the pseudo inverse of matrix A), the error is given as $\boldsymbol{b}$ - $A\mathbf{x}^{\square}_{u}$, If b does not lie in the span of columns of A then there are no solutions and the least square solution is given as $\mathbf{x}^{\square}_{ls}=A^{-1}\boldsymbol{b}$ (here A^{-1} is the pseudo inverse of matrix A) and the error is given as $\boldsymbol{b}$ - $A\boldsymbol{x}^{\square}_{ls}$

Use the above information solve the following equations and compute the error :

$$x+y+z=0$$

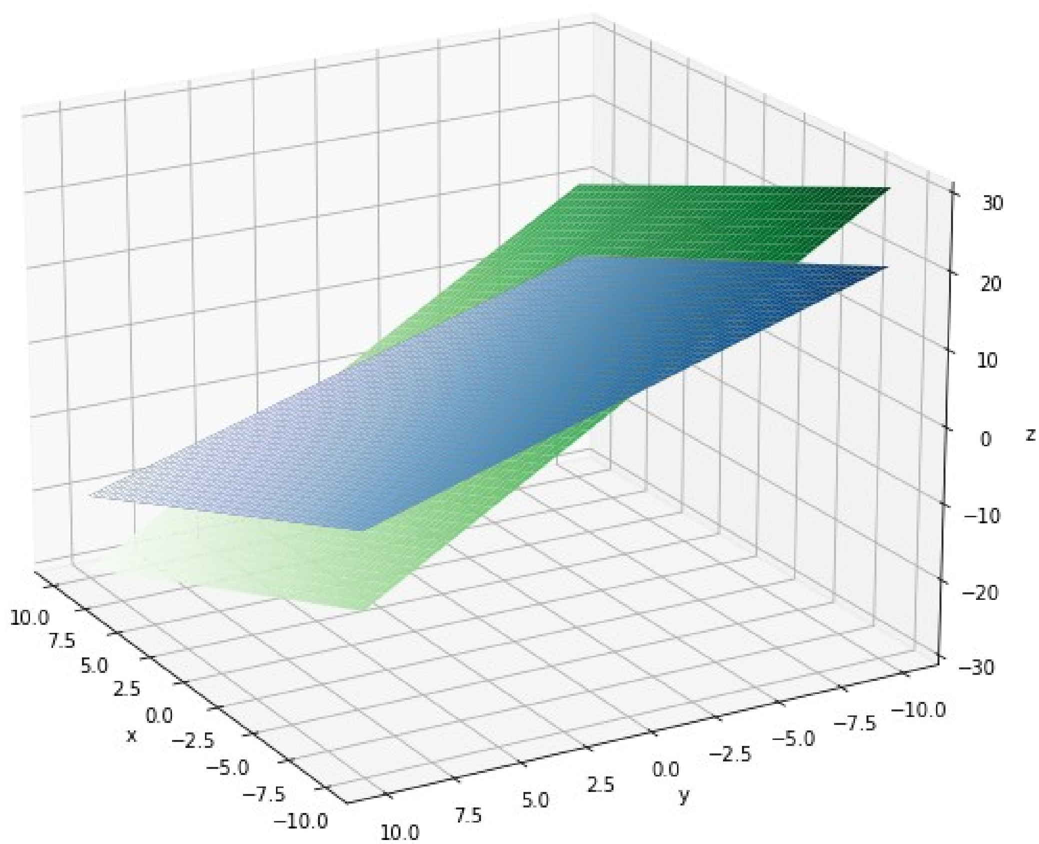
$$3x+3y+3z=0$$

$$x+2y+z=0$$

```python
# Define matrix A and B
A = np.array([
        [1, 1, 1],
        [3, 3, 3],
        [1, 2, 1]
    ]) # write your code here
b = np.array([0, 0, 0]) # write your code here
print('A=',A,'\n')
print('b=',b,'\n')

# Determine the rank of matrix A
rank = np.linalg.matrix_rank(A) # write your code here
print('Matrix rank=',rank,'\n')

# Determine the pseudo-inverse of A (since A is not Square matrix)
A_inverse = np.linalg.pinv(A) # write your code here
print('A_inverse=',A_inverse,'\n')

# Determine the optimal solution
x_op = A_inverse @ b # write your code here
print('x=',x_op,'\n')

# Plot the equations
x = np.linspace(-10, 10, 1000)
y = np.linspace(-10, 10, 1000)
X, Y = np.meshgrid(x, y)
z1 = -X -Y
z2 = (-3*X -3*Y)/3
z3 = -X -2*Y

fig = plt.figure(figsize=(20, 10))
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, z1, cmap='inferno')
ax.plot_surface(X, Y, z2, cmap='Blues')
ax.plot_surface(X, Y, z3, cmap='Greens')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
ax.view_init(20, 150)


# Validate the solution by computing the error
error = b - (A @ x_op) # write your code here
print('error=',error,'\n')
```

```
A= [[1 1 1]
 [3 3 3]
 [1 2 1]]

b= [0 0 0]
```

```
Matrix rank= 2

A_inverse= [[ 0.1  0.3 -0.5]
 [-0.1 -0.3  1. ]
 [ 0.1  0.3 -0.5]]

x= [0. 0. 0.]

error= [0. 0. 0.]
```



For the following equation :

$$x + y + z = 0$$

$$3x + 3y + 3z = 2$$

$$x+2y+z=0$$

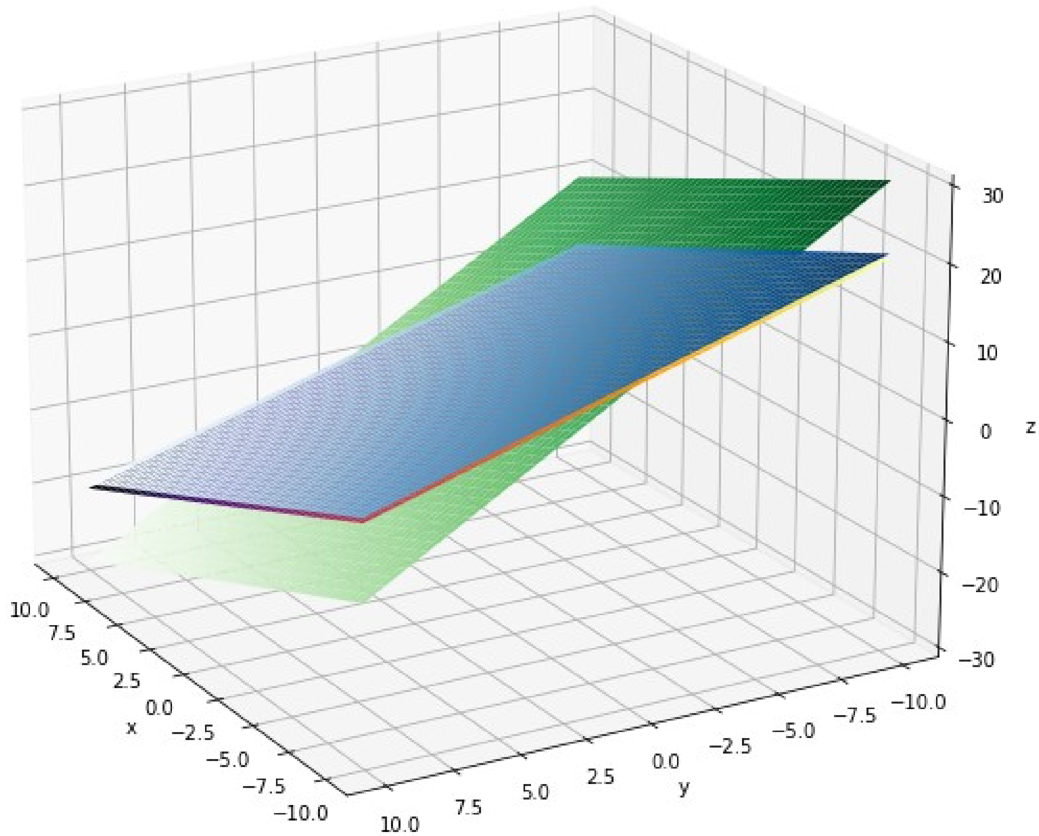
Write the code to :

1. Define Matrices A and B
2. Determine the rank of A
3. Determine the Pseudo Inverse of matrix A
4. Determine the optimal solution
5. Plot the equations
6. Validate the solution by obataining error

```
# write your code here
# Define matrix A and B
A = np.array([
        [1, 1, 1],
        [3, 3, 3],
        [1, 2, 1]
    ]) # write your code here
b = np.array([0, 2, 0]) # write your code here
print('A=',A,'\n')
print('b=',b,'\n')

# Determine the rank of matrix A
rank = np.linalg.matrix_rank(A) # write your code here
print('Matrix rank=',rank,'\n')

# Determine the pseudo-inverse of A (since A is not Square matrix)
A_inverse = np.linalg.pinv(A) # write your code here
print('A_inverse=',A_inverse,'\n')

# Determine the optimal solution
x_op = A_inverse @ b # write your code here
print('x=',x_op,'\n')

# Plot the equations
x = np.linspace(-10, 10, 1000)
y = np.linspace(-10, 10, 1000)
X, Y = np.meshgrid(x, y)
z1 = -X -Y
z2 = (2-3*X -3*Y)/3
z3 = -X -2*Y

fig = plt.figure(figsize=(20, 10))
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, z1, cmap='inferno')
ax.plot_surface(X, Y, z2, cmap='Blues')
ax.plot_surface(X, Y, z3, cmap='Greens')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
```

```python
ax.view_init(20, 150)


# Validate the solution by computing the error
error = b - (A @ x_op) # write your code here
print('error=',error,'\n')
```

```
A= [[1 1 1]
 [3 3 3]
 [1 2 1]]

b= [0 2 0]

Matrix rank= 2

A_inverse= [[ 0.1  0.3 -0.5]
 [-0.1 -0.3  1. ]
 [ 0.1  0.3 -0.5]]

x= [ 0.6 -0.6  0.6]

error= [-6.0000000e-01  2.0000000e-01 -8.8817842e-16]
```



Examples

Find the solution for the below equations and justify the case that they belong to

1. $2x+3y+5z=2, 9x+3y+2z=5, 5x+9y+z=7$

2. $2x+3y=1, 5x+9y=4, x+y=0$

3. $2x+5y+10z=0, 9x+2y+z=1, 4x+10y+20z=5$

4. $2x+3y=0, 5x+9y=2, x+y=-2$

5. $2x+5y+3z=0, 9x+2y+z=0, 4x+10y+6z=0$

1: $2x+3y+5z=2, 9x+3y+2z=5, 5x+9y+z=7$

```
A = np.array([
    [2, 3, 5],
```

```
    [9, 3, 2],
    [5, 9, 1]
])
print("A: ", A)
print()

b = np.array([2, 5, 7])
print("b: ", b)
print()

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

A_inverse = np.linalg.pinv(A)
x_op = A_inverse @ b

print("Optimal x: ", x_op)
print("Error: ", b - (A @ x_op))
print()
print("This is of the form CASE I: A is full rank and m=n")
```

```
A:  [[2 3 5]
 [9 3 2]
 [5 9 1]]

b:  [2 5 7]

Rank of A:  3
Optimal x:  [ 0.38613861  0.57425743 -0.0990099 ]
Error:  [3.55271368e-15 3.55271368e-15 8.88178420e-16]

This is of the form CASE I: A is full rank and m=n
```

2. $2x+3y=1, 5x+9y=4, x+y=0$

```
A = np.array([
    [2, 3],
    [5, 9],
    [1, 1]
])
print("A: ", A)
print()

b = np.array([1, 4, 0])
print("b: ", b)
print()

rank_A = np.linalg.matrix_rank(A)
print("Rank of A: ", rank_A)
```

```
A_inverse = np.linalg.pinv(A)
x_op = A_inverse @ b

print("Optimal x: ", x_op)
print("Error: ", b - (A @ x_op))
print()
print("This is of the form CASE II: A is full rank but m>n")
```

```
A:  [[2 3]
 [5 9]
 [1 1]]

b:  [1 4 0]

Rank of A:  2
Optimal x:  [-1.  1.]
Error:  [2.22044605e-15 5.32907052e-15 1.33226763e-15]

This is of the form CASE II: A is full rank but m>n
```

3. $2x+5y+10z=0, 9x+2y+z=1, 4x+10y+20z=5$

```
A = np.array([
    [2, 5, 10],
    [9, 2, 1],
    [4, 10, 20]
])
print("A: ", A)
print()

b = np.array([0, 1, 5])
print("b: ", b)
print()

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

A_inverse = np.linalg.pinv(A)
x_op = A_inverse @ b

print("Optimal x: ", x_op)
print("Error: ", b - (A @ x_op))
print()
print("This is of the form CASE III: A is not full rank")
```

```
A:  [[ 2  5 10]
 [ 9  2  1]
 [ 4 10 20]]

b:  [0 1 5]
```

```
Rank of A:  2
Optimal x:  [0.07720207 0.08041451 0.14435233]
Error:  [-2.00000000e+00 -1.11022302e-15  1.00000000e+00]

This is of the form CASE III: A is not full rank
```

4. $2x+3y=0, 5x+9y=2, x+y=-2$

```
A = np.array([
    [2, 3],
    [5, 9],
    [1, 1]
])
print("A: ", A)
print()

b = np.array([0, 2, -2])
print("b: ", b)
print()

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

A_inverse = np.linalg.pinv(A)
x_op = A_inverse @ b

print("Optimal x: ", x_op)
print("Error: ", b - (A @ x_op))
print()
print("This is of the form CASE II: A is full rank but m>n")
```

```
A:  [[2 3]
 [5 9]
 [1 1]]

b:  [ 0  2 -2]

Rank of A:  2
Optimal x:  [-4.          2.46153846]
Error:  [ 0.61538462 -0.15384615 -0.46153846]

This is of the form CASE II: A is full rank but m>n
```

5. $2x+5y+3z=0, 9x+2y+z=0, 4x+10y+6z=0$

```
A = np.array([
    [2, 5, 3],
    [9, 2, 1],
    [4, 10, 6]
```

```python
])
print("A: ", A)
print()

b = np.array([0, 0, 0])
print("b: ", b)
print()

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

A_inverse = np.linalg.pinv(A)
x_op = A_inverse @ b

print("Optimal x: ", x_op)
print("Error: ", b - (A @ x_op))
print()
print("This is of the form CASE III: A is not full rank.")
```

```
A:  [[ 2  5  3]
 [ 9  2  1]
 [ 4 10  6]]

b:  [0 0 0]

Rank of A:  2
Optimal x:  [0. 0. 0.]
Error:  [0. 0. 0.]

This is of the form CASE III: A is not full rank.
```