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

In [1]:
```
## Import the required Libraries here
import numpy as np
import matplotlib.pyplot as plt
```

Case 1 :

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

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

$$x + y = 5$$

$$2x + 4y = 4$$

In [2]:

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

# Plot the equations
# write your code here
x = np.linspace(-10,10,100)
y1 = -1*x+5
y2 = -0.5*x+1
plt.plot(x, y1,x, y2)
plt.legend(["x+y=5","2x+4y=4"])
plt.show()
# Validate the solution by obtaining the error
error =  b-np.matmul(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.]]
```

For the following equation :

$$x + y + z = 5$$

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

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

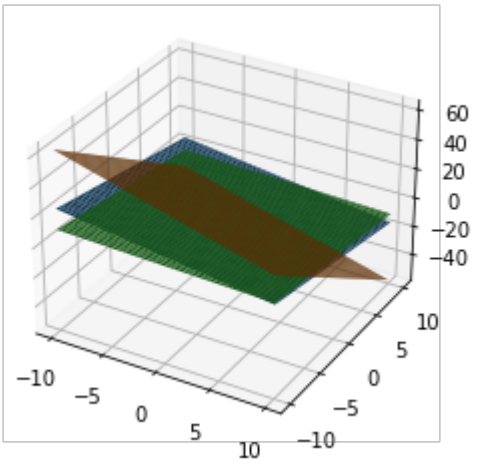
Write the code to :

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

In [3]:

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

# Plot the equations
# write your code here
x = np.linspace(-10,10,100)
y = np.linspace(-10,10,100)
x, y = np.meshgrid(x, y)
z1 = -1*x-1*y+5
z2 = -2*x-4*y+4
z3 = -0.25*x-0.75*y+1
ax = plt.axes(projection ='3d')
ax.plot_surface(x,y,z1)
ax.plot_surface(x,y,z2)
ax.plot_surface(x,y,z3)
plt.show()
# Validate the solution by obtaining the error
error =  b-np.matmul(A,x_op)# write your code here
print('error=',error,'\n')
```

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

b= [[5]
 [4]
 [4]]

Determinant= 7.000000000000001

Matrix rank= 3

A_inverse= [[ 2.         -0.28571429 -0.42857143]
 [-1.          0.42857143  0.14285714]
 [ 0.         -0.14285714  0.28571429]]

x= [[ 7.14285714]
 [-2.71428571]
 [ 0.57142857]]
```




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

Case 2 :

Consider an eqauation $A\mathbf{x}$=**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 $\boldsymbol{x}_u$=$A^{-1}\boldsymbol{b}$ (here A^{-1} is the pseudo inverse of matrix A), where $\boldsymbol{x}_u$ is the unique solution and the error is given as ***b*** - $A\boldsymbol{x}_u$, 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 $\boldsymbol{x}_{ls}$=$A^{-1}\boldsymbol{b}$ (here A^{-1} is the pseudo inverse of matrix A) and the error is given as ***b*** - $A\boldsymbol{x}_{ls}$

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

$$x + z = 0$$

$$x + y + z = 0$$

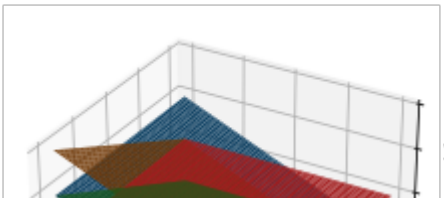
$$y + z = 0$$

$$z = 0$$

In [4]:

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

# Plot the equations
x = np.linspace(-10,10,100)
y = np.linspace(-10,10,100)
x, y = np.meshgrid(x, y)
z1 = -1*x
z2 = -1*x-1*y
z3 = -1*y
z4 = 0*x
ax = plt.axes(projection ='3d')
ax.plot_surface(x,y,z1)
ax.plot_surface(x,y,z2)
ax.plot_surface(x,y,z3)
ax.plot_surface(x,y,z4)
plt.show()
# Validate the solution by computing the error
error =  b-np.matmul(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.]]
```



For the following equation :

$$x + y + z = 35$$

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

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

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

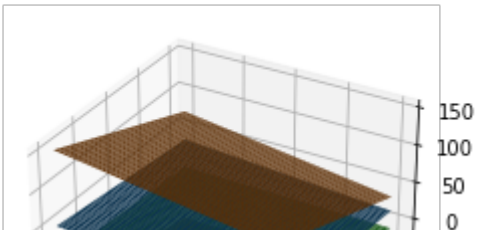
Write the code to :

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

In [5]:

```
# write your code here
# Define matrix A and B
A =  [[1,1,1],[2,4,1],[1,3,4],[1,9,4]]# write your code here
b =  [[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 =  np.matmul(A_inverse,b)# write your code here
print('x=',x_op,'\n')

# Plot the equations
x = np.linspace(-10,10,100)
y = np.linspace(-10,10,100)
x, y = np.meshgrid(x, y)
z1 = 35-x-y
z2 = 94-2*x-4*y
z3 = 1-0.75*y-0.25*x
z4 = -57.5-0.25*x-2.25*y
ax = plt.axes(projection ='3d')
ax.plot_surface(x,y,z1)
ax.plot_surface(x,y,z2)
ax.plot_surface(x,y,z3)
ax.plot_surface(x,y,z4)
plt.show()
# Validate the solution by computing the error
error =  b-np.matmul(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]]
```



Case 3 :

Consider an eqauation $A\mathbf{x}$=**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 $\boldsymbol{x}_u$=$A^{-1}\boldsymbol{b}$ (here A^{-1} is the pseudo inverse of matrix A), the error is given as ***b*** - $A\boldsymbol{x}_u$, 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 $\boldsymbol{x}_{ls}$=$A^{-1}\boldsymbol{b}$ (here A^{-1} is the pseudo inverse of matrix A) and the error is given as ***b*** - $A\boldsymbol{x}_{ls}$

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

$$x + y + z = 0$$

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

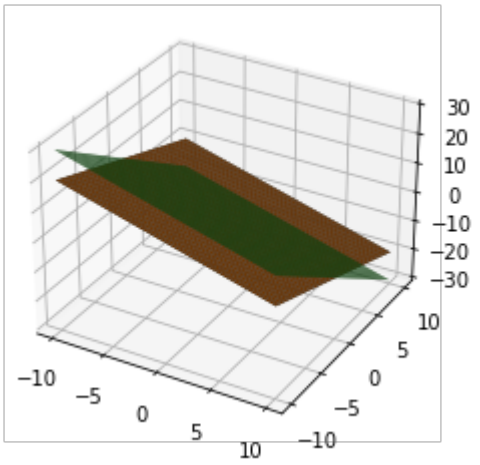
$$x + 2y + z = 0$$

In [6]:

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

# Plot the equations
x = np.linspace(-10,10,100)
y = np.linspace(-10,10,100)
x, y = np.meshgrid(x, y)
z1 = -x-y
z2 = -x-y
z3 = -2*y-x
ax = plt.axes(projection ='3d')
ax.plot_surface(x,y,z1)
ax.plot_surface(x,y,z2)
ax.plot_surface(x,y,z3)
plt.show()
# Validate the solution by computing the error
error =  b-np.matmul(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

In [7]:

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

# Plot the equations
x = np.linspace(-10,10,100)
y = np.linspace(-10,10,100)
x, y = np.meshgrid(x, y)
z1 = -x-y
z2 = 2/3-x-y
z3 = -2*y-x
ax = plt.axes(projection ='3d')
ax.plot_surface(x,y,z1)
ax.plot_surface(x,y,z2)
ax.plot_surface(x,y,z3)
plt.show()
# Validate the solution by computing the error
error =  b-np.matmul(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]]
```

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

In [8]:

```
# write your code here
# declaring all the A,b matrices for the above questions
A1=[[2,3,5],[9,3,2],[5,9,1]]
b1=[[2],[5],[7]]
A2=[[2,3],[5,9],[1,1]]
b2=[[1],[0]]
A3=[[2,5,10],[9,2,1],[4,10,20]]
b3=[[0],[2],[-2]]
A4=[[2,3],[5,9],[1,1]]
b4=[[0],[2],[-2]]
A5=[[2,5,3],[9,2,1],[4,10,6]]
b5=[[0],[0],[0]]

#solving the 1st question
#printing the matrices
print('A1=',A1)
print('b1=',b1,'\n')
#checking which case the question corresponds to
#detailedlogic for this can be found below
if (np.linalg.matrix_rank(A1)==len(A1[0]) and len(A1)==len(A1[0])):
    print("Case 1 \n")
elif(np.linalg.matrix_rank(A1)==len(A1[0]) and len(A1)!=len(A1[0])):
    print("Case 2 \n")
else:
    print("Case 3")
#printing the optimal solution
x_op1 =  np.matmul(np.linalg.pinv(A1),b)
print('x1=',x_op1,'\n')
print("=========================================================")
#solving the 2nd question
#printing the matrices
print('A2=',A2,'\n')
print('b2=',b2,'\n')
#checking which case the question corresponds to
#detailedlogic for this can be found below
if (np.linalg.matrix_rank(A2)==len(A2[0]) and len(A2)==len(A2[0])):
    print("Case 1 \n")
elif(np.linalg.matrix_rank(A2)==len(A2[0]) and len(A2)!=len(A2[0])):
    print("Case 2 \n")
else:
    print("Case 3")
x_op2 =  np.matmul(np.linalg.pinv(A2),b)
print('x2=',x_op2,'\n')
print("=========================================================")
#solving the 3rd question
#printing the matrices
print('A3=',A3,'\n')
print('b3=',b3,'\n')
#checking which case the question corresponds to
#detailedlogic for this can be found below
if (np.linalg.matrix_rank(A3)==len(A3[0]) and len(A3)==len(A3[0])):
    print("Case 1 \n")
elif(np.linalg.matrix_rank(A3)==len(A3[0]) and len(A3)!=len(A3[0])):
    print("Case 2 \n")
else:
    print("Case 3")
#printing the optimal solution
x_op3 =  np.matmul(np.linalg.pinv(A3),b)
print('x3=',x_op3,'\n')
print("=========================================================")
#solving the 4th question
#printing the matrices
```

```
    print( Case 2 \n )
else:
    print("Case 3")
#printing the optimal solution
x_op4 =  np.matmul(np.linalg.pinv(A4),b)
print('x4=',x_op4,'\n')
print("==========================================================")
#solving the 5th question
#printing the matrices
print('A5=',A5,'\n')
print('b5=',b5,'\n')
#checking which case the question corresponds to
#detailedlogic for this can be found below
if (np.linalg.matrix_rank(A5)==len(A5[0]) and len(A5)==len(A5[0])):
    print("Case 1 \n")
elif(np.linalg.matrix_rank(A5)==len(A5[0]) and len(A5)!=len(A5[0])):
    print("Case 2 \n")
else:
    print("Case 3")
#printing the optimal solution
x_op5 =  np.matmul(np.linalg.pinv(A5),b)
print('x5=',x_op5,'\n')
```

```
A1= [[2, 3, 5], [9, 3, 2], [5, 9, 1]]
b1= [[2], [5], [7]]

Case 1

x1= [[ 0.27722772]
 [-0.15181518]
 [-0.01980198]]

==========================================================
A2= [[2, 3], [5, 9], [1, 1]]

b2= [[1], [0]]

Case 2

x2= [[-1.        ]
 [ 0.76923077]]

==========================================================
A3= [[2, 5, 10], [9, 2, 1], [4, 10, 20]]

b3= [[0], [2], [-2]]

Case 3
x3= [[ 0.22487047]
 [ 0.01409326]
 [-0.05202073]]

==========================================================
A4= [[2, 3], [5, 9], [1, 1]]

b4= [[0], [2], [-2]]

Case 2
```

```
x4= [[-1.        ]
 [ 0.76923077]]

=====================================================
A5= [[2, 5, 3], [9, 2, 1], [4, 10, 6]]

b5= [[0], [0], [0]]

Case 3
x5= [[ 0.24273949]
 [-0.06848721]
 [-0.04768097]]
```

In the if-then-else statements, the logic used is as follows:

- The first if statement checks condition for case 1: >

  > we compare the rank with the no of columns to check if it is a full rank matrix
  >
  > and
  >
  > we compared the no of rows and columns to check if it is a square matrix
  >
  > If it is full rank and square, case 1 is printed,

- The second elif statement checks condition for case 2: >

  > we compare the rank with the no of columns to check if it is a full rank matrix
  >
  > and
  >
  > we compared the no of rows and columns to check if it is a square matrix
  >
  > If it is full rank but not square, case 2 is printed,

- Lastly we are left with case 3: >

  > If all other cases fail, matrix is not full rank and so case 3 is printed

In [ ]: