200010021_Lab_2

August 19, 2022

1 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

[1]:
```
## Import the required Libraries here
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

#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 $\boldsymbol{x}\ _{op}$=$A^{-1}\boldsymbol{b}$, 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 :

$$x + y = 5$$

$$2x + 4y = 4$$

[2]:
```
# 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
```

```python
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

# 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 :

$$x + y + z = 5$$
$$2x + 4y + z = 4$$
$$x + 3y + 4z = 4$$

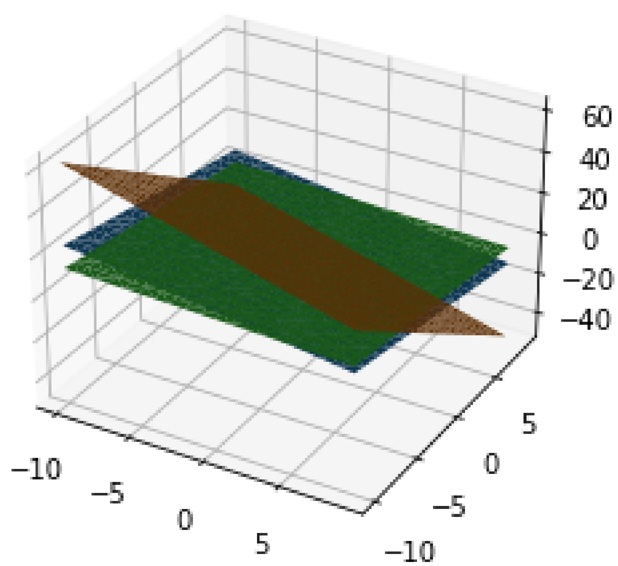
Write the code to : 1. Define Matrices A and B 2. Determine the determinant of A 3. Determine the rank of A 4. Determine the Inverse of matrix A 5. Determine the optimal solution 6. Plot the equations 7. Validate the solution by obataining error

[3]:
```python
def graph(A, B):
    X_Y_points = [[i,j] for i in range(-10, 10) for j in range(-10,10)]

    Z1_points = np.array([B[0] - (np.array(arr) @ A[0][:2]) / A[0][2] for arr␣
 ↪in X_Y_points])
    Z2_points = np.array([B[1] - (np.array(arr) @ A[1][:2]) / A[1][2] for arr␣
 ↪in X_Y_points])
```

```
    Z3_points = np.array([B[2] - (np.array(arr) @ A[2][:2]) / A[2][2] for arr␣
 ↪in X_Y_points])

    X,Y = [], []

    for i in range(len(X_Y_points)):
        X.append(X_Y_points[i][0])
        Y.append(X_Y_points[i][1])

    X = np.array(X)
    Y = np.array(Y)

    Z1_points.shape = (Z1_points.shape[0],)
    Z2_points.shape = (Z2_points.shape[0],)
    Z3_points.shape = (Z3_points.shape[0],)

    fig = plt.figure()
    ax = plt.axes(projection="3d")
    ax.plot_trisurf(X,Y,Z1_points)
    ax.plot_trisurf(X,Y,Z2_points)
    ax.plot_trisurf(X,Y,Z3_points)



## write your code here

A = [[1, 1, 1], [2, 4, 1], [1, 3, 4]]
print(f"A= {A} \n")

B = [[5], [4], [4]]
print(f"B= {B} \n")

Det_A = np.linalg.det(A)
print(f"Determinant= {Det_A} \n")

Rank_A = np.linalg.matrix_rank(A)
print(f"Matrix rank= {Rank_A} \n")

Inverse_A = np.linalg.inv(A)
print(f"A_Inverse= {Inverse_A} \n")

x = Inverse_A @ B
print(f"x= {x} \n")

graph(A,B)

error = B - A @ x
```

```
print(f"error= {error}")
```

```
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 x_u=$A^{-1}\boldsymbol{b}$ (here A^{-1} is the pseudo inverse of matrix A), where x_u is the unique solution and the error is given as $\boldsymbol{b}$ - Ax_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 $x_{ls}= A^{-1}\boldsymbol{b}$ (here A^{-1} is the pseudo inverse of matrix A) and the error is given as $\boldsymbol{b}$ - Ax_{ls}

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

$$x + z = 0$$

$$x + y + z = 0$$

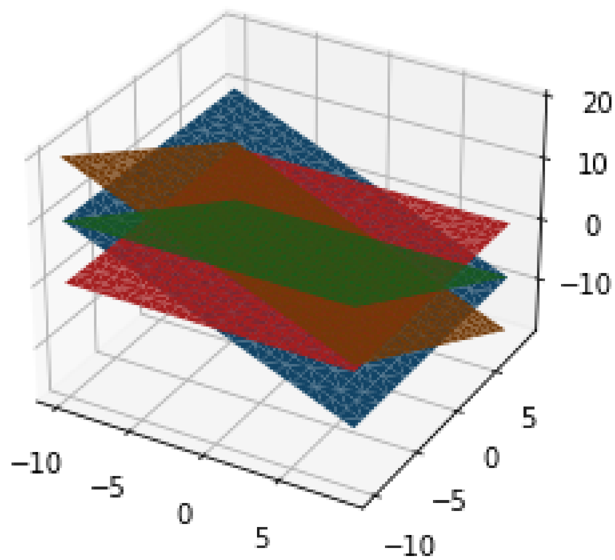
$$y + z = 0$$

$$z = 0$$

[4]:
```python
def graph(A, B):
    X_Y_points = [[i,j] for i in range(-10, 10) for j in range(-10,10)]

    Z1_points = np.array([B[0] - (np.array(arr) @ A[0][:2]) / A[0][2] for arr in X_Y_points])
    Z2_points = np.array([B[1] - (np.array(arr) @ A[1][:2]) / A[1][2] for arr in X_Y_points])
    Z3_points = np.array([B[2] - (np.array(arr) @ A[2][:2]) / A[2][2] for arr in X_Y_points])
    Z4_points = np.array([B[3] - (np.array(arr) @ A[3][:2]) / A[3][2] for arr in X_Y_points])

    X,Y = [], []

    for i in range(len(X_Y_points)):
        X.append(X_Y_points[i][0])
        Y.append(X_Y_points[i][1])

    X = np.array(X)
    Y = np.array(Y)

    Z1_points.shape = (Z1_points.shape[0],)
    Z2_points.shape = (Z2_points.shape[0],)
    Z3_points.shape = (Z3_points.shape[0],)
    Z4_points.shape = (Z4_points.shape[0],)

    fig = plt.figure()
    ax = plt.axes(projection="3d")
    ax.plot_trisurf(X,Y,Z1_points)
    ax.plot_trisurf(X,Y,Z2_points)
    ax.plot_trisurf(X,Y,Z3_points)
    ax.plot_trisurf(X,Y,Z4_points)


# Define matrix A and B
A = [[1, 0, 1], [1, 1, 1], [0, 1, 1], [0, 0, 1]] # write your code here
```

```python
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 = A_inverse @ b # write your code here
print('x=',x_op,'\n')

# Plot the equations
graph(A,b)

# 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.]]

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



For the following equation :

$$x + y + z = 35$$
$$2x + 4y + z = 94$$
$$x + 3y + 4z = 4$$
$$x + 9y + 4z = -230$$

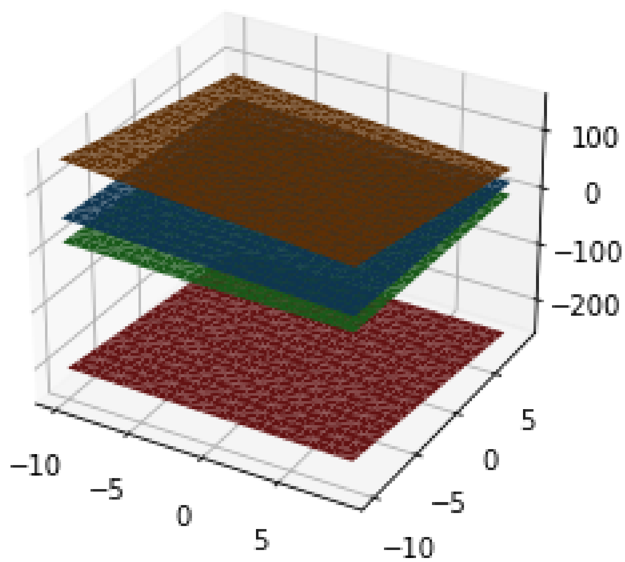
Write the code to : 1. Define Matrices A and B 2. Determine the rank of A 3. Determine the Pseudo Inverse of matrix A 4. Determine the optimal solution 5. Plot the equations 6. Validate the solution by obataining error

[5]:
```
def graph(A, B):
    X_Y_points = [[i,j] for i in range(-10, 10) for j in range(-10,10)]

    Z1_points = np.array([B[0] - (np.array(arr) @ A[0][:2]) / A[0][2] for arr␣
 ↪in X_Y_points])
    Z2_points = np.array([B[1] - (np.array(arr) @ A[1][:2]) / A[1][2] for arr␣
 ↪in X_Y_points])
    Z3_points = np.array([B[2] - (np.array(arr) @ A[2][:2]) / A[2][2] for arr␣
 ↪in X_Y_points])
    Z4_points = np.array([B[3] - (np.array(arr) @ A[3][:2]) / A[3][2] for arr␣
 ↪in X_Y_points])

    X,Y = [], []
```

```python
    for i in range(len(X_Y_points)):
        X.append(X_Y_points[i][0])
        Y.append(X_Y_points[i][1])

    X = np.array(X)
    Y = np.array(Y)

    Z1_points.shape = (Z1_points.shape[0],)
    Z2_points.shape = (Z2_points.shape[0],)
    Z3_points.shape = (Z3_points.shape[0],)
    Z4_points.shape = (Z4_points.shape[0],)

    fig = plt.figure()
    ax = plt.axes(projection="3d")
    ax.plot_trisurf(X,Y,Z1_points)
    ax.plot_trisurf(X,Y,Z2_points)
    ax.plot_trisurf(X,Y,Z3_points)
    ax.plot_trisurf(X,Y,Z4_points)

# write your code here

A = [[1, 1, 1], [2, 4, 1], [1, 3, 4], [1, 9, 4]]
print(f"A= {A} \n")

B = [[35], [94], [4], [-230]]
print(f"B= {B} \n")


Rank_A = np.linalg.matrix_rank(A)
print(f"Matrix rank= {Rank_A} \n")

Inverse_A = np.linalg.pinv(A)
print(f"Psuedo inverse = {Inverse_A} \n")

x_op = Inverse_A @ B
print(f"x_op is {x_op} \n")

graph(A,B)

error = b - A @ x_op
print(f"The error is {error}")
```

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

B= [[35], [94], [4], [-230]]

Matrix rank= 3
```

```
Psuedo 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_op is [[111.9548552 ]
 [-35.69250426]
 [ -3.37649063]]

The error is [[-72.88586031]
 [-77.76320273]
 [  8.6286201 ]
 [222.78364566]]
```



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

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

$$x + y + z = 0$$

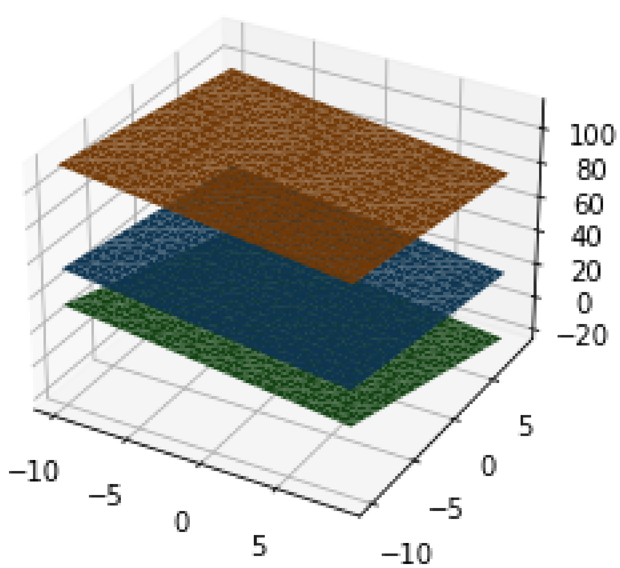
$$3x + 3y + 3z = 0$$

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

[6]:
```python
def graph(A, B):
    X_Y_points = [[i,j] for i in range(-10, 10) for j in range(-10,10)]

    Z1_points = np.array([B[0] - (np.array(arr) @ A[0][:2]) / A[0][2] for arr␣
 ↪in X_Y_points])
    Z2_points = np.array([B[1] - (np.array(arr) @ A[1][:2]) / A[1][2] for arr␣
 ↪in X_Y_points])
    Z3_points = np.array([B[2] - (np.array(arr) @ A[2][:2]) / A[2][2] for arr␣
 ↪in X_Y_points])

    X,Y = [], []

    for i in range(len(X_Y_points)):
        X.append(X_Y_points[i][0])
        Y.append(X_Y_points[i][1])

    X = np.array(X)
    Y = np.array(Y)

    Z1_points.shape = (Z1_points.shape[0],)
    Z2_points.shape = (Z2_points.shape[0],)
    Z3_points.shape = (Z3_points.shape[0],)

    fig = plt.figure()
    ax = plt.axes(projection="3d")
    ax.plot_trisurf(X,Y,Z1_points)
    ax.plot_trisurf(X,Y,Z2_points)
    ax.plot_trisurf(X,Y,Z3_points)


# 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 = A_inverse @ b # write your code here
print('x=',x_op,'\n')
```

```
# Plot the equations
graph(A,B)
# 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 :

$$x + y + z = 0$$

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

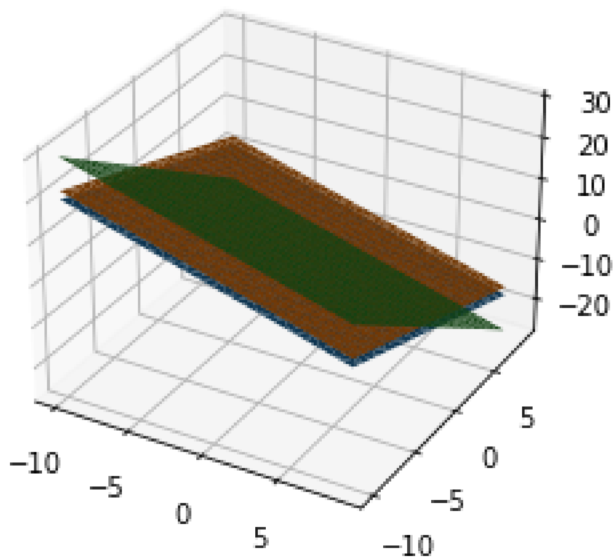
$$x + 2y + z = 0$$

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

[7]:
```python
# write your code here



def graph(A, B):
    X_Y_points = [[i,j] for i in range(-10, 10) for j in range(-10,10)]

    Z1_points = np.array([B[0] - (np.array(arr) @ A[0][:2]) / A[0][2] for arr in X_Y_points])
    Z2_points = np.array([B[1] - (np.array(arr) @ A[1][:2]) / A[1][2] for arr in X_Y_points])
    Z3_points = np.array([B[2] - (np.array(arr) @ A[2][:2]) / A[2][2] for arr in X_Y_points])

    X,Y = [], []

    for i in range(len(X_Y_points)):
        X.append(X_Y_points[i][0])
        Y.append(X_Y_points[i][1])

    X = np.array(X)
    Y = np.array(Y)

    Z1_points.shape = (Z1_points.shape[0],)
    Z2_points.shape = (Z2_points.shape[0],)
    Z3_points.shape = (Z3_points.shape[0],)

    fig = plt.figure()
    ax = plt.axes(projection="3d")
    ax.plot_trisurf(X,Y,Z1_points)
    ax.plot_trisurf(X,Y,Z2_points)
    ax.plot_trisurf(X,Y,Z3_points)



A = [[1, 1, 1], [3, 3, 3], [1, 2, 1]]
print(f"A= {A} \n")

B = [[0], [2], [0]]
```

```python
print(f"B= {B} \n")

Rank_A = np.linalg.matrix_rank(A)
print(f"Matrix rank= {Rank_A} \n")

Inverse_A = np.linalg.pinv(A)
print(f"A_Inverse= {Inverse_A} \n")

x = Inverse_A @ B
print(f"x_op = {x} \n")

graph(A,B)

error = B - A @ x
print(f"error= {error}")
```

```
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_op = [[ 0.6]
 [-0.6]
 [ 0.6]]

error= [[-6.00000000e-01]
 [ 2.00000000e-01]
 [-7.77156117e-16]]
```



2 Examples

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

$1.2x + 3y + 5z = 2, 9x + 3y + 2z = 5, 5x + 9y + z = 7$

$2.2x + 3y = 1, 5x + 9y = 4, x + y = 0$

$3.2x + 5y + 10z = 0, 9x + 2y + z = 1, 4x + 10y + 20z = 5$

$4.2x + 3y = 0, 5x + 9y = 2, x + y = -2$

$5.2x + 5y + 3z = 0, 9x + 2y + z = 0, 4x + 10y + 6z = 0$

3 First We find which cases the questions belong to by finding their rank

[8]:
```
# We find which cases they belong to

# In all the above cases A is 3x3 so it is a square matrix

# Now we find the Ranks

# 1.)
A1 = [[2, 3, 5], [9, 3, 2], [5, 9, 1]]
b1 = [[2], [5], [7]]

Rank_A1 = np.linalg.matrix_rank(A1)
```

```
print(f"Rank of Q1 is {Rank_A1}")

# 2.)
A2 = [[2, 3, 0], [5, 9, 0], [1, 1, 0]]
b2 = [[1], [4], [0]]

Rank_A2 = np.linalg.matrix_rank(A2)
print(f"Rank of Q2 is {Rank_A2}")

# 3.)
A3 = [[2, 5, 10], [9, 2, 1], [4, 10, 20]]
b3 = [[0], [1], [5]]

Rank_A3 = np.linalg.matrix_rank(A3)
print(f"Rank of Q3 is {Rank_A3}")

# 4.)
A4 = [[2, 3, 0], [5, 9, 0], [1, 1, 0]]
b4 = [[0], [2], [-2]]

Rank_A4 = np.linalg.matrix_rank(A4)
print(f"Rank of Q4 is {Rank_A4}")

# 5.)
A5 = [[2, 5, 3], [9, 2, 1], [4, 10, 6]]
b5 = [[0], [0], [0]]

Rank_A5 = np.linalg.matrix_rank(A5)
print(f"Rank of Q5 is {Rank_A5}")
```

```
Rank of Q1 is 3
Rank of Q2 is 2
Rank of Q3 is 2
Rank of Q4 is 2
Rank of Q5 is 2
```

4 Hence Q1 is #case 1 rest are #case 3

4.0.1 Defining graph function

[9]:
```
def graph(A, B):
    X_Y_points = [[i,j] for i in range(-10, 10) for j in range(-10,10)]

    Z1_points = np.array([B[0] - (np.array(arr) @ A[0][:2]) / A[0][2] for arr␣
 ↪in X_Y_points])
    Z2_points = np.array([B[1] - (np.array(arr) @ A[1][:2]) / A[1][2] for arr␣
 ↪in X_Y_points])
```

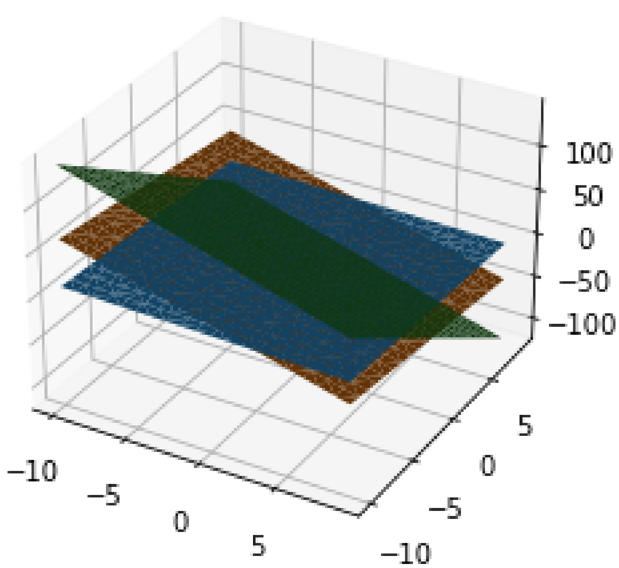
```
    Z3_points = np.array([B[2] - (np.array(arr) @ A[2][:2]) / A[2][2] for arr in X_Y_points])

    X,Y = [], []

    for i in range(len(X_Y_points)):
        X.append(X_Y_points[i][0])
        Y.append(X_Y_points[i][1])

    X = np.array(X)
    Y = np.array(Y)

    Z1_points.shape = (Z1_points.shape[0],)
    Z2_points.shape = (Z2_points.shape[0],)
    Z3_points.shape = (Z3_points.shape[0],)

    fig = plt.figure()
    ax = plt.axes(projection="3d")
    ax.plot_trisurf(X,Y,Z1_points)
    ax.plot_trisurf(X,Y,Z2_points)
    ax.plot_trisurf(X,Y,Z3_points)


def graph2d(A, B):
    X = [i for i in range(-10,10)]

    y1 = np.array([B[0][0] - (A[0][0] * _ )/ A[0][1] for _ in X])
    y2 = np.array([B[1][0] - (A[1][0] * _ )/ A[1][1] for _ in X])
    y3 = np.array([B[2][0] - (A[2][0] * _ )/ A[2][1] for _ in X])

    y1.shape = (y1.shape[0],)
    y2.shape = (y2.shape[0],)
    y3.shape = (y3.shape[0],)

    Z1_points = np.zeros(y1.shape[0])
    Z2_points = np.zeros(y1.shape[0])
    Z3_points = np.zeros(y1.shape[0])

    fig = plt.figure()
    plt.plot(X,y1)
    plt.plot(X,y2)
    plt.plot(X,y3)
```

5 Solving the questions

5.1 Q1 –> Case 1

[10]:
```
A1_inverse =  np.linalg.inv(A1)
x1_op = A1_inverse @ b1
graph(A1,b1)
error1 = b1 - A1 @ x1_op

print(f"A1 is {A1} \n")
print(f"b is {b1}\n")
print(f"x_op is {x1_op}\n")
print(f"The error is {error1}")
```

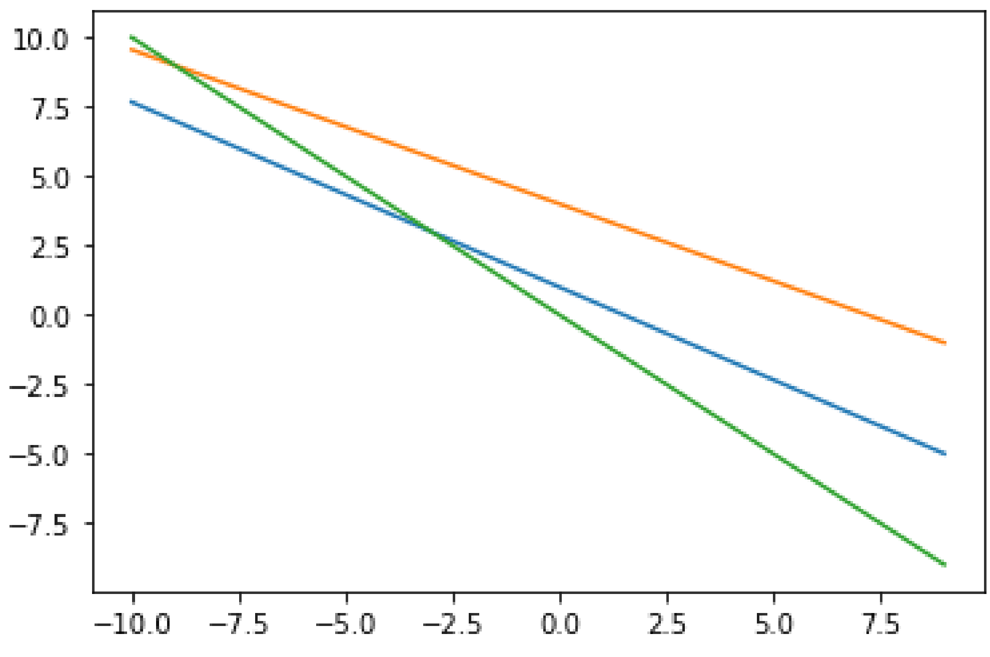
```
A1 is [[2, 3, 5], [9, 3, 2], [5, 9, 1]]

b is [[2], [5], [7]]

x_op is [[ 0.38613861]
 [ 0.57425743]
 [-0.0990099 ]]

The error is [[-4.44089210e-16]
 [ 0.00000000e+00]
 [-1.77635684e-15]]
```



5.2 Q2 –> Case 3

[11]:
```
A2_pinverse =  np.linalg.pinv(A2)
x2_op = A2_pinverse @ b2
graph2d(A2,b2)
error2 = b2 - A2 @ x2_op

print(f"A2 is {A2} \n")
print(f"b is {b2}\n")
print(f"x_op is {x2_op}\n")
print(f"The error is {error2}")
```

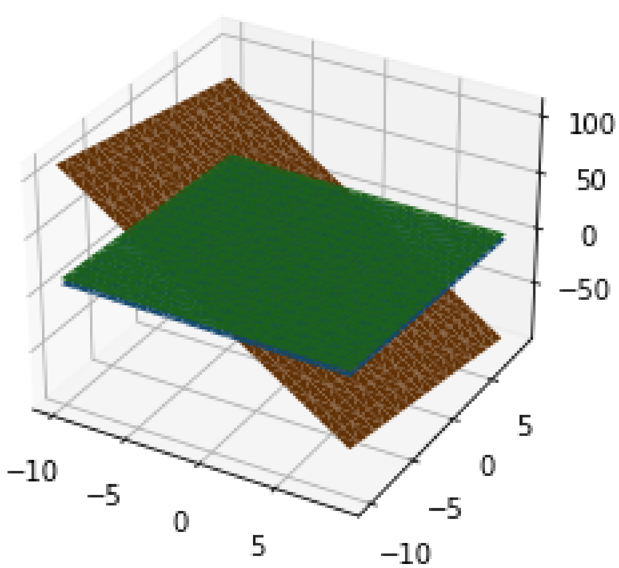
```
A2 is [[2, 3, 0], [5, 9, 0], [1, 1, 0]]

b is [[1], [4], [0]]

x_op is [[-1.]
 [ 1.]
 [ 0.]]

The error is [[1.77635684e-15]
 [5.32907052e-15]
 [1.11022302e-15]]
```



5.3 Q3 –> Case 3

[12]:
```
A3_inverse =  np.linalg.pinv(A3)
x3_op = A3_inverse @ b3
graph(A3,b3)
error3 = b3 - A3 @ x3_op

print(f"A3 is {A3} \n")
print(f"b is {b3}\n")
print(f"x_op is {x3_op}\n")
print(f"The error is {error3}")
```

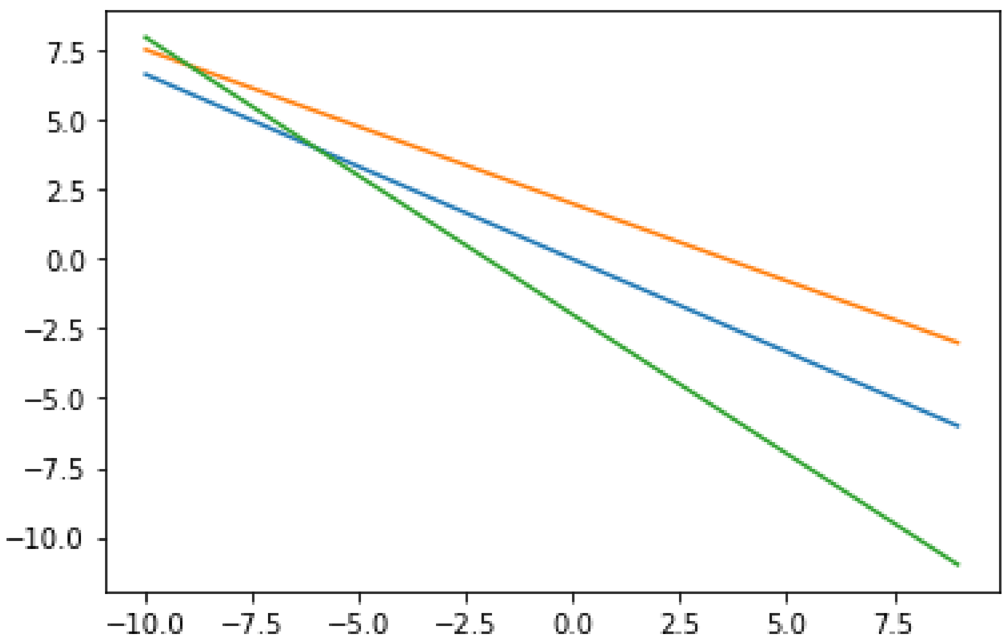
```
A3 is [[2, 5, 10], [9, 2, 1], [4, 10, 20]]

b is [[0], [1], [5]]

x_op is [[0.07720207]
 [0.08041451]
 [0.14435233]]

The error is [[-2.00000000e+00]
 [-1.11022302e-15]
 [ 1.00000000e+00]]
```



5.4 Q4 —> Case 3

[13]:
```
A4_inverse =  np.linalg.pinv(A4)
x4_op = A4_inverse @ b4
graph2d(A4,b4)
error4 = b4 - A4 @ x4_op

print(f"A4 is {A4} \n")
print(f"b is {b4}\n")
print(f"x_op is {x4_op}\n")
print(f"The error is {error4}")
```

```
A4 is [[2, 3, 0], [5, 9, 0], [1, 1, 0]]

b is [[0], [2], [-2]]

x_op is [[-4.        ]
 [ 2.46153846]
 [ 0.        ]]

The error is [[ 0.61538462]
 [-0.15384615]
 [-0.46153846]]
```



5.5 Q5 —> Case 3

[14]:
```
A5_inverse =  np.linalg.pinv(A5)
x5_op = A5_inverse @ b5
graph(A5,b5)
error5 = b5 - A5 @ x5_op

print(f"A5 is {A5} \n")
print(f"b is {b5}\n")
print(f"x_op is {x5_op}\n")
print(f"The error is {error5}")
```

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

b is [[0], [0], [0]]

x_op is [[0.]
 [0.]
 [0.]]

The error is [[0.]
 [0.]
 [0.]]
```