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 [ ]:
```
## Import the required Libraries here
import numpy as np
import matplotlib.pyplot as plt
from numpy import linalg as la
```

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 :

$$x + y = 5$$

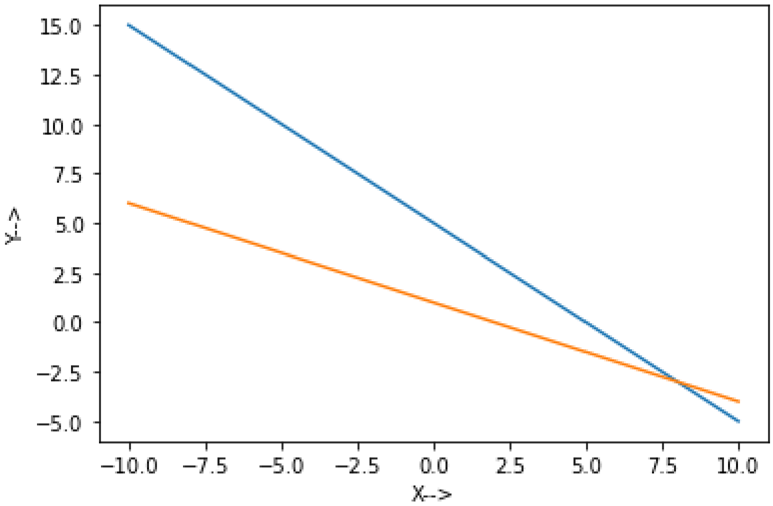
$$2x + 4y = 4$$

In [ ]:
```
# Function to plot 2D equations
def plot2D(A, b):
  # Plot the equations
  x = np.linspace(-10, 10)
  y = []

  A = np.array(A)
  b = np.array(b)
  for i in range(len(A)):
    y.append((b[i][0] - A[i][0]*x) / A[i][1])

  for i in range(len(A)):
    plt.plot(x, y[i])

  plt.xlabel("X-->")
  plt.ylabel("Y-->")
  plt.show()
```

In [ ]:
```
# Define Matrix A and B
A = np.matrix([[1,1], [2,4]])
b = np.matrix([[5], [4]])
print('A=',A,'\n')
print('b=',b,'\n')


# Determine the determinant of matrix A
Det = la.det(A)
print('Determinant=',Det,'\n')

# Determine the rank of the matrix A
rank = la.matrix_rank(A)
```

```
print('Matrix rank=',rank,'\n')

# Determine the Inverse of matrix A
A_inverse = la.inv(A)
print('A_inverse=',A_inverse,'\n')

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

# Plot the equations
plot2D(A, b)

# Validate the solution by obtaining the error
error = b - A @ x_op
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 :

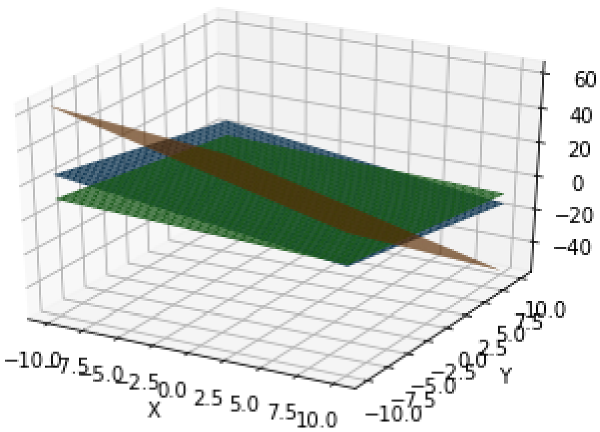
1. Define Matrices A and B

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

In [ ]:
```
# Function to plot 3D equations
def plot3D(A, b):
  # Plot the equations
  x = np.linspace(-10, 10)
  y = np.linspace(-10, 10)
  x, y = np.meshgrid(x, y)
  z = []

  A = np.array(A)
  b = np.array(b)
  for i in range(len(A)):
    z.append((b[i][0] - A[i][0]*x - A[i][1]*y) / A[i][2])

  fig, ax = plt.subplots(subplot_kw={"projection": "3d"})

  for i in range(len(A)):
    ax.plot_surface(x, y, z[i])

  plt.xlabel("X")
  plt.ylabel("Y")
  plt.show()
```

In [ ]:
```
# Define Matrix A and B
A = np.matrix([[1,1,1], [2,4,1], [1,3,4]])
b = np.matrix([[5], [4], [4]])
print('A=',A,'\n')
print('b=',b,'\n')


# Determine the determinant of matrix A
Det = la.det(A)
print('Determinant=',Det,'\n')

# Determine the rank of the matrix A
rank = la.matrix_rank(A)
print('Matrix rank=',rank,'\n')

# Determine the Inverse of matrix A
A_inverse = la.inv(A)
print('A_inverse=',A_inverse,'\n')

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

# Plot the equations
plot3D(A, b)

# Validate the solution by obtaining the error
error = b - A @ x_op
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**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}***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} ***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 :

$$x + z = 0$$

$$x + y + z = 0$$

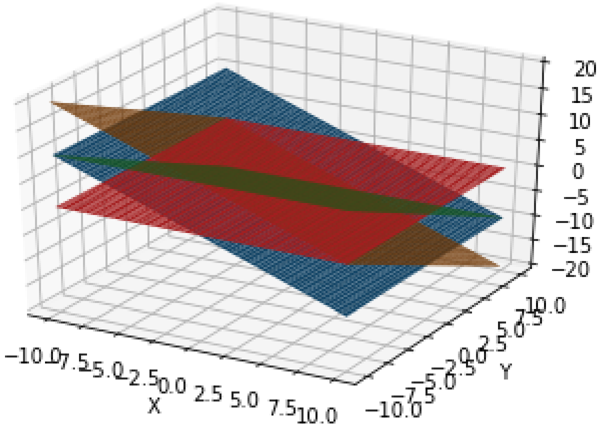
$$y + z = 0$$

$$z = 0$$

In [ ]:
```
# Define matrix A and B
```

```
A = np.matrix([[1, 0, 1], [1, 1, 1], [0, 1, 1], [0, 0, 1]])
b = np.matrix([[0], [0], [0], [0]])
print('A=',A,'\n')
print('b=',b,'\n')

# Determine the rank of matrix A
rank = la.matrix_rank(A)
print('Matrix rank=',rank,'\n')

# Determine the pseudo-inverse of A (since A is not Square matrix)
A_inverse = la.pinv(A)
print('A_inverse=',A_inverse,'\n')

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

# Plot the equations
plot3D(A, b)

# Validate the solution by computing the error
error = b - A @ x_op
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 :

$$x + y + z = 35$$

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

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

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

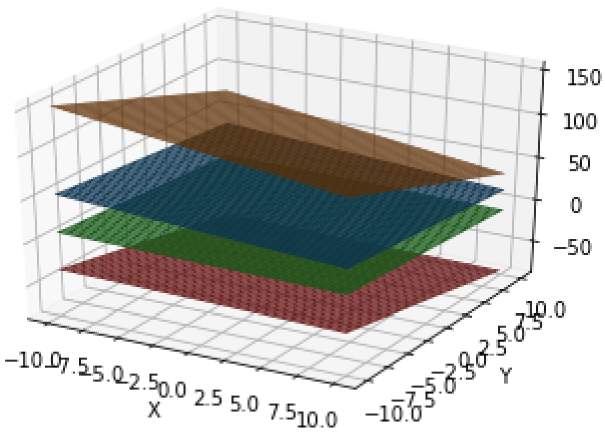
Write the code to :

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

In [ ]:

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

# Determine the rank of matrix A
rank = la.matrix_rank(A)
print('Matrix rank=',rank,'\n')

# Determine the pseudo-inverse of A (since A is not Square matrix)
A_inverse = la.pinv(A)
print('A_inverse=',A_inverse,'\n')

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

# Plot the equations
plot3D(A, b)

# Validate the solution by computing the error
error = b - A @ x_op
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**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 $\mathbf{x}_u$=A^{-1}***b*** (here A^{-1} is the pseudo inverse of matrix A), the error is given as ***b*** - $A\mathbf{x}_u$, 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 $\mathbf{x}_{ls}$=A^{-1}***b*** (here A^{-1} is the pseudo inverse of matrix A) and the error is given as ***b*** - $A\mathbf{x}_{ls}$

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

$$x + y + z = 0$$

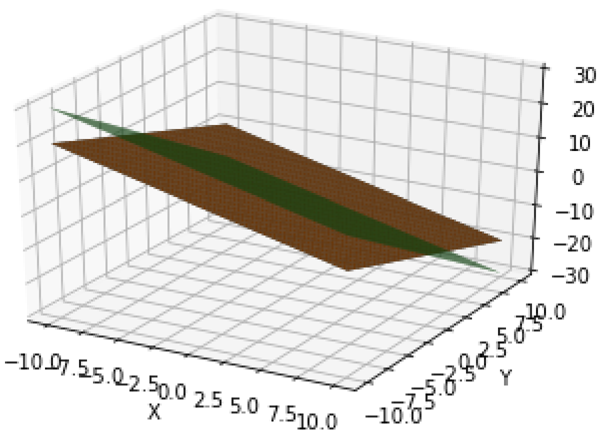
$$3x + 3y + 3z = 0$$

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

In [ ]:
```
# Define matrix A and B
A = np.matrix([[1, 1, 1], [3, 3, 3], [1, 2, 1]])
```

```
b = np.matrix([[0], [0], [0]])
print('A=',A,'\n')
print('b=',b,'\n')

# Determine the rank of matrix A
rank = la.matrix_rank(A)
print('Matrix rank=',rank,'\n')

# Determine the pseudo-inverse of A (since A is not Square matrix)
A_inverse = la.pinv(A)
print('A_inverse=',A_inverse,'\n')

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

# Plot the equations
plot3D(A, b)

# Validate the solution by computing the error
error = b - A @ x_op
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 :

$$x + y + z = 0$$

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

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

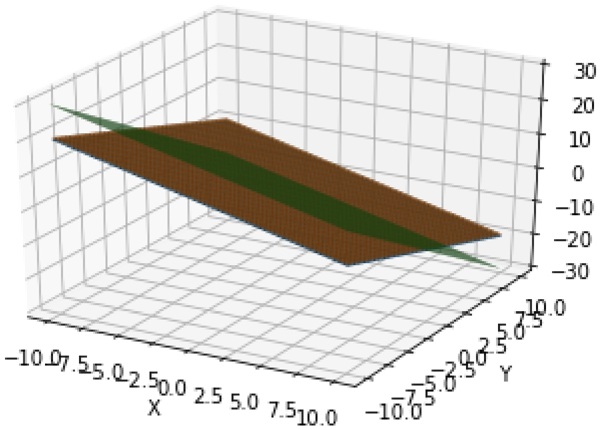
Write the code to :

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 [ ]:

```
# Define matrix A and B
A = np.matrix([[1, 1, 1], [3, 3, 3], [1, 2, 1]])
b = np.matrix([[0], [2], [0]])

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

# Determine the rank of matrix A
rank = la.matrix_rank(A)
print('Matrix rank=',rank,'\n')

# Determine the pseudo-inverse of A (since A is not Square matrix)
A_inverse = la.pinv(A)
print('A_inverse=',A_inverse,'\n')

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

# Plot the equations
plot3D(A, b)

# Validate the solution by computing the error
error = b - A @ x_op
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.$2x + 5y + 10z = 0, 9x + 2y + z = 1, 4x + 10y + 20z = 5$

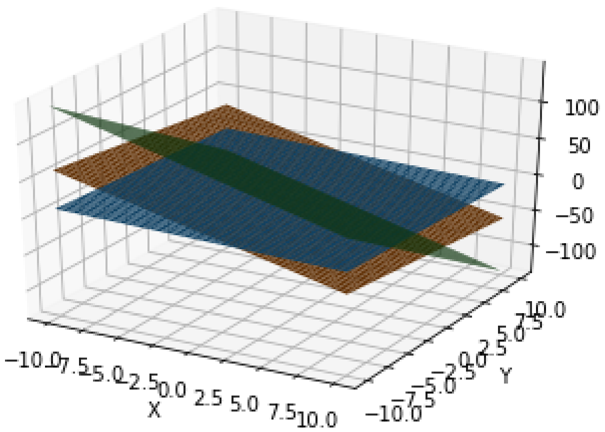
4.$2x + 3y = 0, 5x + 9y = 2, x + y = -2$

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

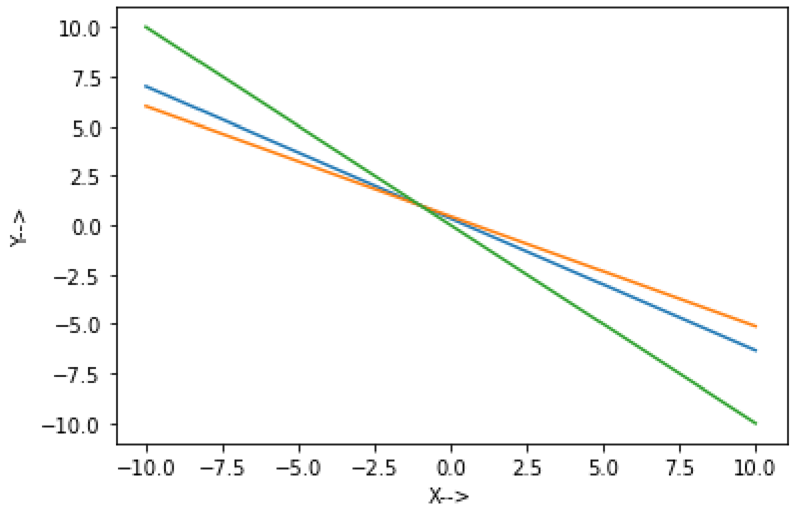
In [ ]:
```
# Matrices A & b
As = []
bs = []
# 1
As.append(np.matrix([[2, 3, 5], [9, 3, 2], [5, 9, 1]]))
bs.append(np.matrix([[2], [5], [7]]))
# 2
As.append(np.matrix([[2, 3], [5, 9], [1, 1]]))
bs.append(np.matrix([[1], [4], [0]]))
# 3
As.append(np.matrix([[2, 5, 10], [9, 2, 1], [4, 10, 20]]))
bs.append(np.matrix([[0], [1], [5]]))
# 4
As.append(np.matrix([[2, 3], [5, 9], [1, 1]]))
bs.append(np.matrix([[0], [2], [-2]]))
# 5
As.append(np.matrix([[2, 5, 3], [9, 2, 1], [4, 10, 6]]))
bs.append(np.matrix([[0], [0], [0]]))

# Function to find solution and case for given equations
def findSol(A, b):
  print('A=',A,'\n')
  print('b=',b,'\n')

  # Determine the rank of matrix A
  rank = la.matrix_rank(A)
  print('Matrix rank=',rank,'\n')
```

```
  # Determine the pseudo-inverse of A (since A is not Square matrix)
  A_inverse = la.pinv(A)
  print('A_inverse=',A_inverse,'\n')

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

  # Plot the equations
  dim = len(np.array(A)[0])
  if dim == 3:
    plot3D(A, b)
  elif dim == 2:
    plot2D(A, b)

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

  # Check if it is square matrix and/or full matrix
  rows = len(np.array(A))
  cols = len(np.array(A)[0])
  isSquare = (rows == cols)
  if rows > cols:
    isFullRank = (rank == cols)
  else:
    isFullRank = (rank == rows)

  # Choose the case
  if isFullRank and isSquare:
    # Must be case 1
    print("Answer - Case 1 \nJustification - The matrix A is a full rank squ
  elif isFullRank and not isSquare:
    # Must be case 2
    print("Answer - Case 2 \nJustification - The matrix A is a full rank non
  elif not isFullRank:
    # Must be case 3
    print("Answer - Case 3 \nJustification - The matrix A is not a full rank

for i, (A, b) in enumerate(zip(As, bs)):
  print(f"Question {i+1}:\n")
  findSol(A, b)
  print("\n\n\n")
```

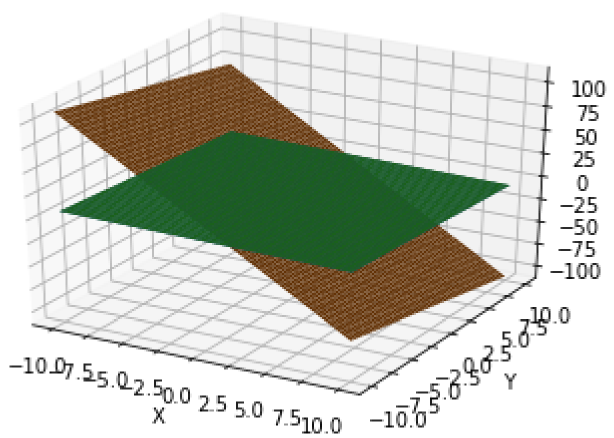
```
Question 1:

A= [[2 3 5]
 [9 3 2]
 [5 9 1]]

b= [[2]
 [5]
 [7]]

Matrix rank= 3

A_inverse= [[-0.04950495  0.13861386 -0.02970297]
 [ 0.00330033 -0.07590759  0.13531353]
 [ 0.21782178 -0.00990099 -0.06930693]]

x= [[ 0.38613861]
 [ 0.57425743]
 [-0.0990099 ]]
```



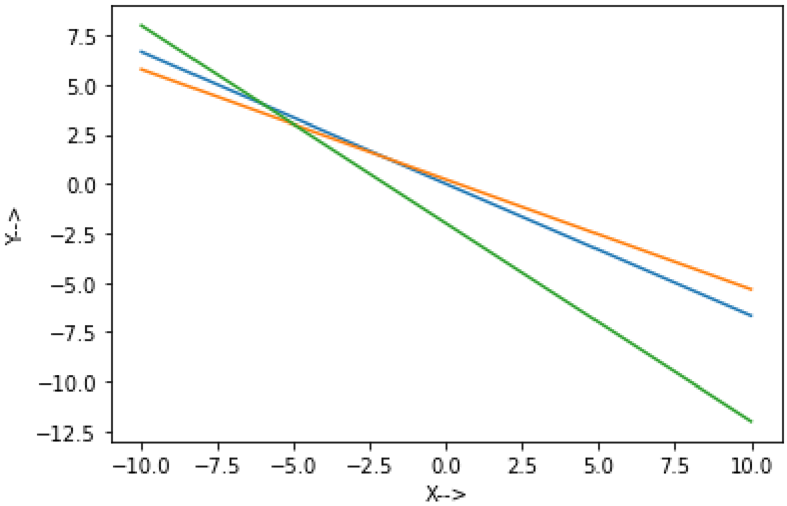

```
error= [[3.55271368e-15]
 [3.55271368e-15]
 [8.88178420e-16]]

Answer - Case 1
Justification - The matrix A is a full rank square matrix.




Question 2:

A= [[2 3]
 [5 9]
 [1 1]]

b= [[1]
 [4]
 [0]]

Matrix rank= 2

A_inverse= [[ 1.         -0.5         1.5       ]
 [-0.53846154  0.38461538 -0.84615385]]

x= [[-1.]
 [ 1.]]
```



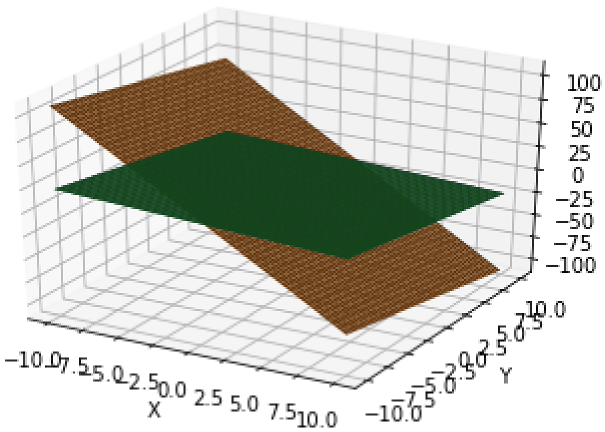

```
error= [[2.22044605e-15]
 [5.32907052e-15]
 [1.33226763e-15]]

Answer - Case 2
Justification - The matrix A is a full rank non-square matrix.




Question 3:

A= [[ 2  5 10]
 [ 9  2  1]
 [ 4 10 20]]

b= [[0]
 [1]
 [5]]

Matrix rank= 2

A_inverse= [[-0.00352332  0.11243523 -0.00704663]
 [ 0.00733679  0.00704663  0.01467358]
 [ 0.01703627 -0.02601036  0.03407254]]

x= [[0.07720207]
 [0.08041451]
 [0.14435233]]
```



```
error= [[-2.00000000e+00]
 [-1.33226763e-15]
 [ 1.00000000e+00]]

Answer - Case 3
Justification - The matrix A is not a full rank matrix.




Question 4:

A= [[2 3]
 [5 9]
 [1 1]]

b= [[ 0]
 [ 2]
 [-2]]

Matrix rank= 2

A_inverse= [[ 1.         -0.5         1.5       ]
 [-0.53846154  0.38461538 -0.84615385]]

x= [[-4.        ]
 [ 2.46153846]]
```



```
error= [[ 0.61538462]
 [-0.15384615]
 [-0.46153846]]

Answer - Case 2
Justification - The matrix A is a full rank non-square matrix.




Question 5:

A= [[ 2  5  3]
 [ 9  2  1]
 [ 4 10  6]]

b= [[0]
 [0]
 [0]]

Matrix rank= 2

A_inverse= [[-0.00927612  0.12136974 -0.01855223]
 [ 0.0319029  -0.03424361  0.06380581]
 [ 0.01967924 -0.02384049  0.03935847]]

x= [[0.]
 [0.]
 [0.]]
```




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

Answer - Case 3
Justification - The matrix A is not a full rank matrix.
```