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
import plotly.graph_objects as go
import scipy as sp
%matplotlib inline
```

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 :

$$x + y = 5$$

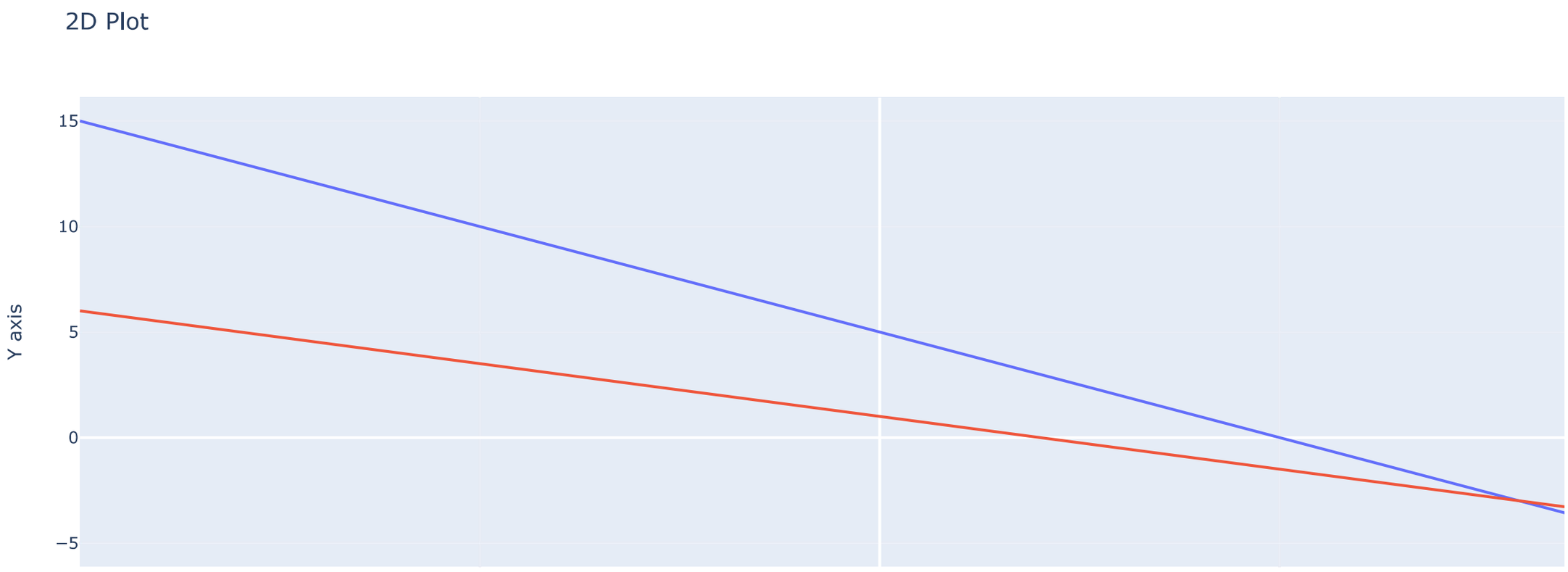
$$2x + 4y = 4$$

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

#Determine the determinant of matrix A
Det = A[0][0]*A[1][1] - A[0][1]*A[1][0]
print('Determinant=',Det,'\n')

#Determine the rank of the matrix A

#Converting into row echelon form
A_rank = np.zeros(A.shape)
A_rank[1] = A[1] - 2*A[0]
A_rank[0] = A[0]
print('The row echelon form of A is',A_rank,'\n')

#creating a function to determine the rank of a matrix
#rank represents the number of non-zero rows in a matrix
def checkRank(matrix):
    MatrixRank = 0
    for i in range(matrix.shape[0]):
```

```
        if np.sum(matrix[i]) != 0:
            MatrixRank = MatrixRank + 1

    return MatrixRank

checkRank(A_rank)
rank = checkRank(A_rank)
print('Matrix rank=',rank,'\n')

# Determine the Inverse of matrix A

#calculate adjoint for a 2x2 matrix
def adjointMatrix(matrix):
    adjoint = np.zeros(matrix.shape)
    adjoint[0][0] = (-1)**(1+1)*(matrix[1][1])
    adjoint[0][1] = (-1)**(1+2)*(matrix[1][0])
    adjoint[1][0] = (-1)**(2+1)*(matrix[0][1])
    adjoint[1][1] = (-1)**(2+2)*(matrix[0][0])

    adjoint = np.transpose(adjoint)

    return adjoint

A_inverse =  adjointMatrix(A)/Det
print('A_inverse =',A_inverse,'\n')

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

# Plot the equations
x = np.linspace(-10,10,1000)

line1 = 5 - x
line2 = (4 - 2*x)/4

fig = go.Figure()
fig.add_trace(go.Scatter(x=x, y=line1, name="Line 1", mode="lines"))
fig.add_trace(go.Scatter(x=x, y=line2, name="Line 2", mode="lines"))
fig.update_layout(
    title="2D Plot", xaxis_title="X axis", yaxis_title="Y axis"
)
fig.show()
# Validate the solution by obtaining the error
error = b - np.matmul(A, x_op)
print('error=',error,'\n')
```

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

b= [[5]
 [4]]

Determinant= 2
```

```
The row echelon form of A is [[1. 1.]
 [0. 2.]]

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 :

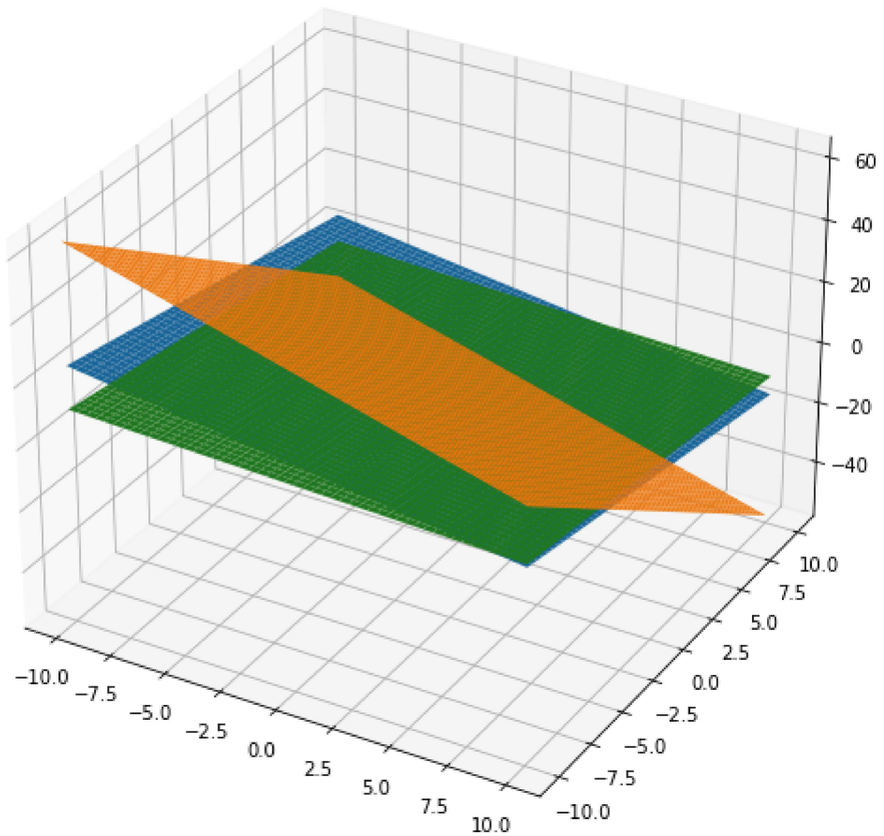
1. Define Matrices A and B
2. Determine the determinant of A
3. Determine the rank of A
4. Determine the Inverse of matrix A
5. Determine the optimal solution
6. Plot the equations
7. Validate the solution by obataining error

In [3]:

```
#Defining matrices A and B

A_NEW = np.array([[1, 1, 1], [2, 4, 1], [1, 3, 4]])
B_NEW = np.array([[5], [4], [4]])
print('A=',A_NEW,'\n')
print('b=',B_NEW,'\n')

#Determine the determinant of A

#Using numpy library to compute the determinant of 3x3 matrix
det = np.linalg.det(A_NEW)
print("Determinant of A = ", det,'\n')

#Determine the rank of the matrix A

#Converting into row echelon form
A_rank = np.zeros(A_NEW.shape)
A_rank[0] = A_NEW[0]
A_rank[1] = A_NEW[1] - 2*A_NEW[0]
A_rank[2] = A_NEW[2] - A_NEW[0]
A_rank[2] = A_rank[2] - A_rank[1]
print('The row echelon form of A is',A_rank,'\n')

rank = checkRank(A_rank)
print('Matrix rank=',rank,'\n')

#inverse matrix for A

#using the numpy library to find the inverse of the matrix A
A_inv = np.linalg.inv(A_NEW)
print('A_inverse =',A_inv,'\n')

#Determine the optimal solution

#using the numpy library to perform matrix multiplication
X_optimal = np.matmul(A_inv, B_NEW)
print('X=' ,X_optimal)

# Plot the equations

x = np.outer(np.linspace(-10, 10, 1000), np.ones(1000))
```

```python
y = x.copy().T # transpose
z1 = 5 - x - y
z2 = 4 - 2*x - 4*y
z3 = (4 - x - 3*y)/4

# Creating figure
fig = plt.figure(figsize =(16,9))
ax = plt.axes(projection ='3d')

# Creating plot
ax.plot_surface(x, y, z1)
ax.plot_surface(x, y, z2)
ax.plot_surface(x, y, z3)

# show plot
plt.show()

# Validate the solution by obtaining the error
error = B_NEW - np.matmul(A_NEW,X_optimal)
print('error=',error,'\n')
```

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

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

Determinant of A =  7.999999999999998

The row echelon form of A is [[ 1.  1.  1.]
 [ 0.  2. -1.]
 [ 0.  0.  4.]]

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 $A\mathbf{x}$=**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 :

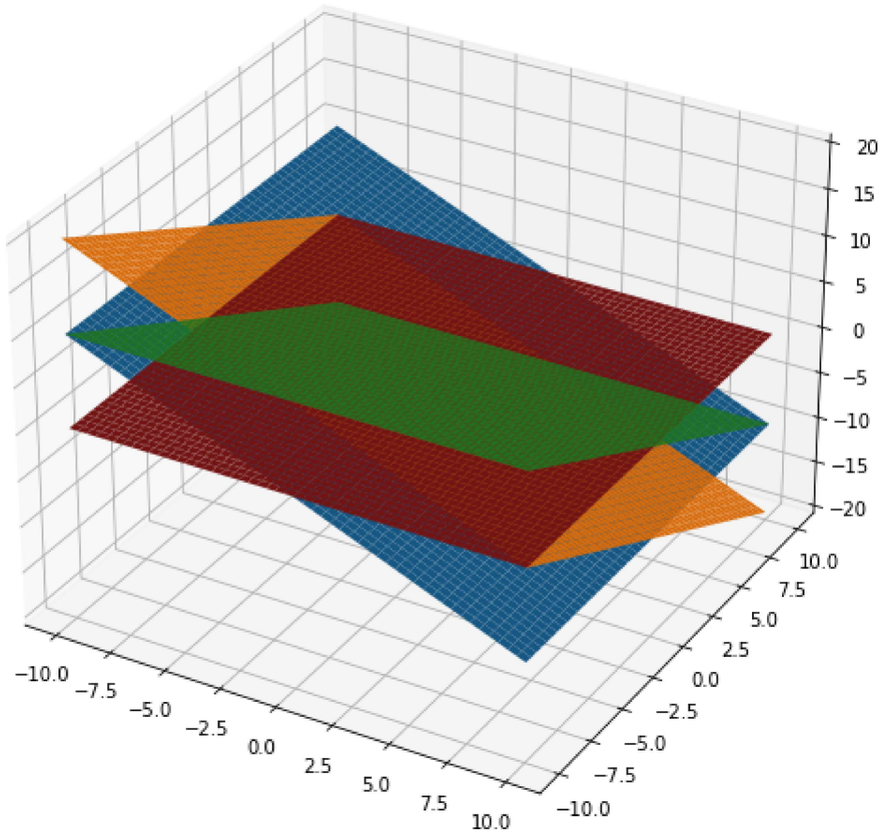
$$x + z = 0$$
$$x + y + z = 0$$
$$y + z = 0$$
$$z = 0$$

In [4]:

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

#Determine the rank of the matrix A

#Converting into row echelon form
A_rank = np.array([[1,0,1], [1,1,1],[0,1,1],[0,0,1]])

A_rank[1] = A_rank[1] - A_rank[0]
A_rank[2] = A_rank[2] - A_rank[1]
A_rank[3] = A_rank[3] - A_rank[2]
print('The row echelon form of A is',A_rank,'\n')

rank = checkRank(A_rank)
print('Matrix rank=',rank,'\n')

#Determine the pseudo-inverse of A (since A is not Square matrix)

#using the numpy library to find the psuedo-inverse of the matrix A
A_inverse =  np.linalg.pinv(A)
print('A_inverse=',A_inverse,'\n')

# Determine the optimal solution

#using the numpy library to perform matrix multiplication
x_op =  np.matmul(A_inverse,b)
print('x=',x_op,'\n')

# Plot the equations

x = np.outer(np.linspace(-10, 10, 1000), np.ones(1000))
y = x.copy().T # transpose
z1 = -x
z2 = -x-y
z3 = -y
z4 = 0*x

# Creating figure
fig = plt.figure(figsize =(16,9))
ax = plt.axes(projection ='3d')

# Creating plot
ax.plot_surface(x, y, z1)
ax.plot_surface(x, y, z2)
ax.plot_surface(x, y, z3)
ax.plot_surface(x, y, z4)

# show plot
plt.show()

# Validate the solution by obtaining the error
```

```
error = b - np.matmul(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]]

The row echelon form of A is [[1 0 1]
 [0 1 0]
 [0 0 1]
 [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

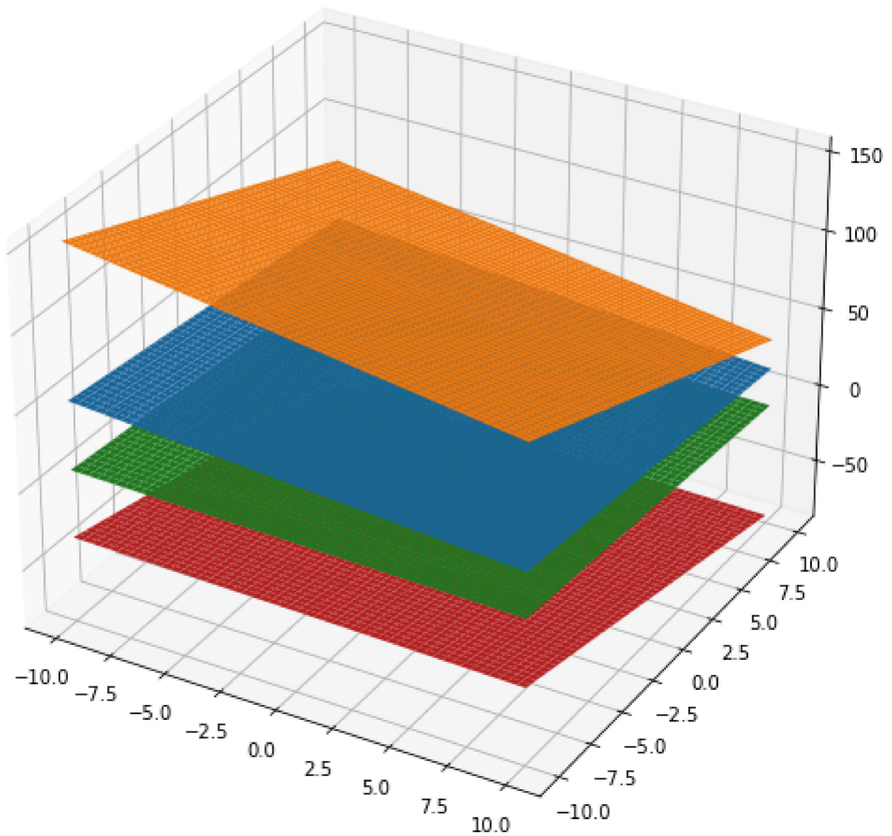
5. Plot the equations
6. Validate the solution by obataining error

In [5]:

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

#Determine the rank of matrix A

#Converting into row echelon form
A_rank = np.array([[1,1,1], [2,4,1],[1,3,4],[1,9,4]])

A_rank[1] = A_rank[1] - 2*A_rank[0]
A_rank[2] = A_rank[2] - A_rank[0]
A_rank[3] = A_rank[3] - A_rank[0]
A_rank[2] = A_rank[2] - A_rank[1]
A_rank[3] = A_rank[3] - 2*A_rank[1]
A_rank[3] = A_rank[3] - (7/4)*A_rank[2]

rank = np.linalg.matrix_rank(A_rank)
print('The row echelon form of A is',A_rank,'\n')

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

#Determine the pseudo-inverse of A (since A is not Square matrix)

#using the numpy library to find the pseudo-inverse of the matrix A
A_inverse =  np.linalg.pinv(A)
print('A_inverse=',A_inverse,'\n')

# Determine the optimal solution

#using the numpy library to perform matrix multiplication
x_op =  np.matmul(A_inverse,b)
print('x=',x_op,'\n')

# Plot the equations

x = np.outer(np.linspace(-10, 10, 1000), np.ones(1000))
y = x.copy().T #transpose
z1 = 35-x-y
z2 = 94-2*x-4*y
z3 = (4-x-3*y)/4
z4 = (-230-x-9*y)/4

# Creating figure
fig = plt.figure(figsize =(16,9))
ax = plt.axes(projection ='3d')

# Creating plot
ax.plot_surface(x, y, z1)
```

```
ax.plot_surface(x, y, z2)
ax.plot_surface(x, y, z3)
ax.plot_surface(x, y, z4)

# show plot
plt.show()

# Validate the solution by obtaining the error
error = b - np.matmul(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]]

The row echelon form of A is [[ 1  1  1]
 [ 0  2 -1]
 [ 0  0  4]
 [ 0  4 -2]]

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 $A\mathbf{x}$=**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 :

$$x + y + z = 0$$

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

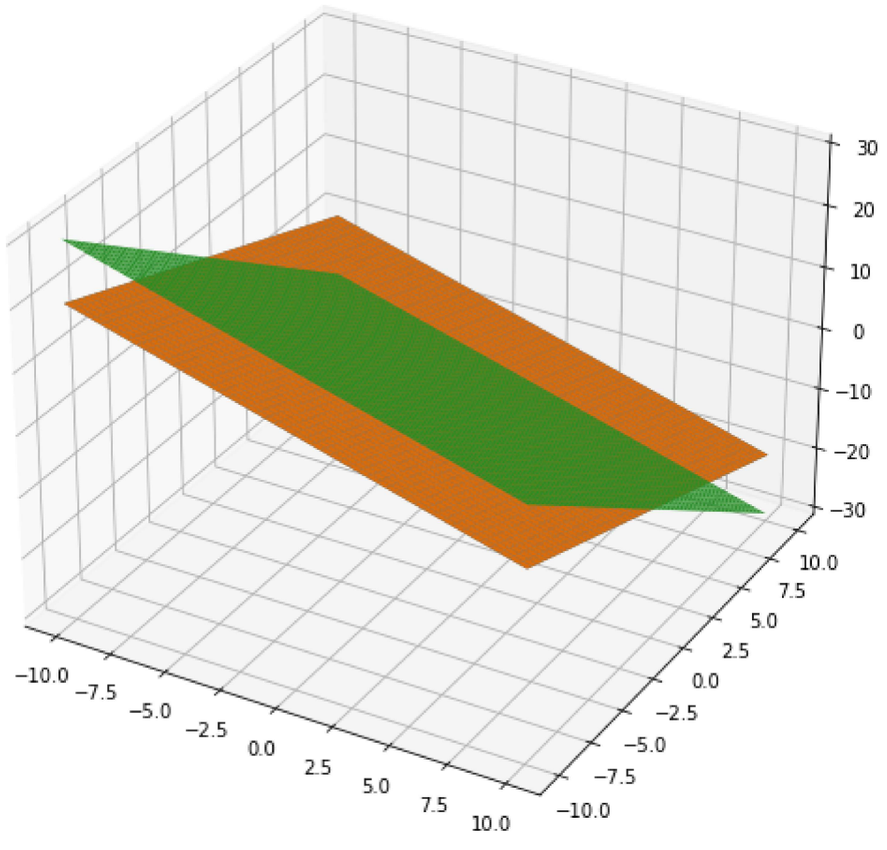
$$x + 2y + z = 0$$

In [6]:
```
#Define matrix A and B
```

```
A =  np.array([[1,1,1], [3,3,3],[1,2,1]])
b =  np.array([[0],[0],[0]])
print('A=',A,'\n')
print('b=',b,'\n')

#Determine the rank of matrix A

#Converting into row echelon form
A_rank = np.array([[1,1,1], [3,3,3],[1,2,1]])

A_rank[1] = A_rank[1] - 3*A_rank[0]
A_rank[2] = A_rank[2] - A_rank[0]
print('The row echelon form of A is',A_rank,'\n')

rank = checkRank(A_rank)
print('Matrix rank=',rank,'\n')

#Determine the inverse of A

#using the numpy library to find the pseudo-inverse of the matrix A
A_inverse =  np.linalg.pinv(A)
print('A_inverse=',A_inverse,'\n')

# Determine the optimal solution

#using the numpy library to perform matrix multiplication
x_op =  np.matmul(A_inverse,b)
print('x=',x_op,'\n')

#Plot the equations

x = np.outer(np.linspace(-10, 10, 1000), np.ones(1000))
y = x.copy().T # transpose
z1 = -x-y
z2 = (-3*x-3*y)/3
z3 = -x-2*y

# Creating figure
fig = plt.figure(figsize =(16,9))
ax = plt.axes(projection ='3d')

# Creating plot
ax.plot_surface(x, y, z1)
ax.plot_surface(x, y, z2)
ax.plot_surface(x, y, z3)

# show plot
plt.show()

# Validate the solution by obtaining the error
error = b - np.matmul(A,x_op)
print('error=',error,'\n')
```

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

```
 [1 2 1]]

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

The row echelon form of A is [[1 1 1]
 [0 0 0]
 [0 1 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 :

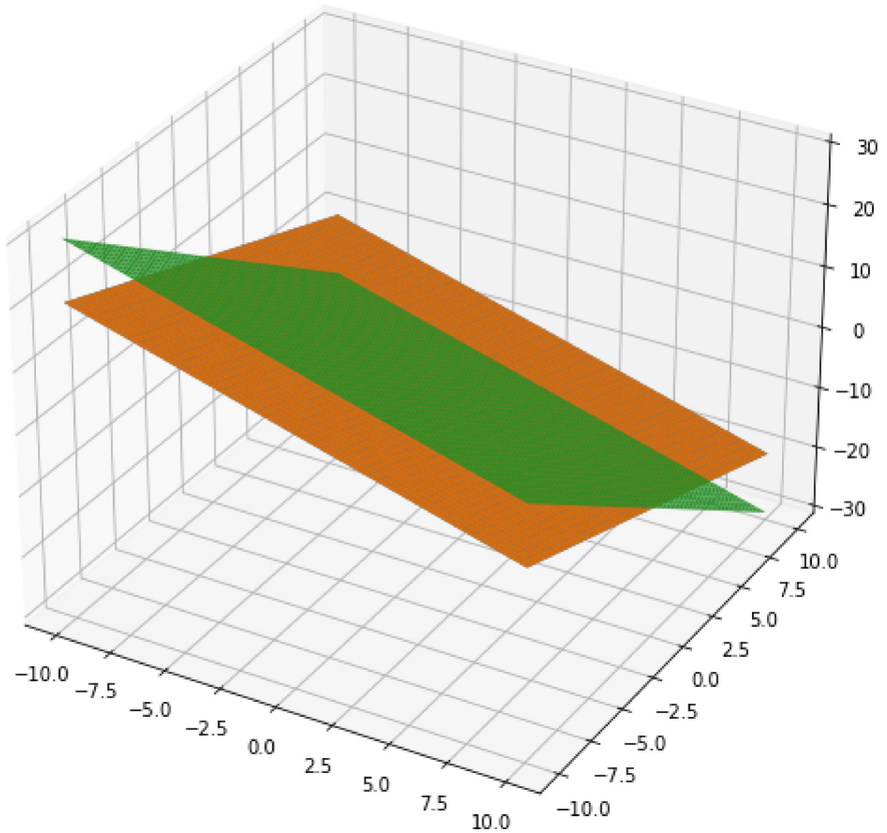
1. Define Matrices A and B
2. Determine the rank of A
3. Determine the Pseudo Inverse of matrix A
4. Determine the optimal solution
5. Plot the equations
6. Validate the solution by obataining error

In [7]:

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

#Determine the rank of matrix A

#Converting into row echelon form
A_rank = np.array([[1,1,1], [3,3,3],[1,2,1]])

A_rank[1] = A_rank[1] - 3*A_rank[0]
A_rank[2] = A_rank[2] - A_rank[0]
print('The row echelon form of A is',A_rank,'\n')

rank = checkRank(A_rank)
print('Matrix rank=',rank,'\n')

#Determine the inverse of A

#using the numpy library to find the psuedo-inverse of the matrix A
A_inverse =  np.linalg.pinv(A)
print('A_inverse=',A_inverse,'\n')

# Determine the optimal solution

#using the numpy library to perform matrix multiplication
x_op =  np.matmul(A_inverse,b)
print('x=',x_op,'\n')

# Plot the equations

x = np.outer(np.linspace(-10, 10, 1000), np.ones(1000))
y = x.copy().T # transpose
z1 = -x-y
z2 = (-3*x-3*y)/3
z3 = -x-2*y
```

```
# Creating figure
fig = plt.figure(figsize =(16,9))
ax = plt.axes(projection ='3d')

# Creating plot
ax.plot_surface(x, y, z1)
ax.plot_surface(x, y, z2)
ax.plot_surface(x, y, z3)

# show plot
plt.show()

# Validate the solution by obtaining the error
error = b - np.matmul(A,x_op)
print('error=',error,'\n')
```

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

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

The row echelon form of A is [[1 1 1]
 [0 0 0]
 [0 1 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 [8]:
```
def solveEquations(A, b):
    #conversion into numpy arrays
```

```
    A = np.array(A)
    b = np.array(b)

    augmentedMatrix = np.c_[A, b]

    rank_Aug = np.linalg.matrix_rank(augmentedMatrix)
    rank_matrix = np.linalg.matrix_rank(A)

    case = 0
    #unique - 0, infinite - 1, inconsistent - 2
    if rank_Aug == rank_matrix:
        if rank_Aug != A.shape[0]:
            case = 1
    else:
        case = 2

    InverseA = np.linalg.pinv(A)
    OptimalSolution = np.matmul(InverseA, b)

    error = b - np.matmul(A, OptimalSolution)

    return case, OptimalSolution, error
```

In [9]:

```
#problem 1

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

case, optimal, error = solveEquations(A1, b1)

print("Optimal solution is ", optimal, "\n")
print("Error is ", error, "\n")

if case == 0:
    print("The system of equations is unique")

if case == 1:
    print("The system of equations has infinite solutions")

if case == 2:
    print("The system of equations is inconsistent")
```

```
Optimal solution is  [[ 0.38613861]
 [ 0.57425743]
 [-0.0990099 ]]

Error is  [[3.55271368e-15]
 [3.55271368e-15]
 [8.88178420e-16]]

The system of equations is unique
```

In [10]:

```
#problem 2
```

```
A1 = [[2,3], [5,9], [1,1]]
b1 = [[1], [4], [0]]

case, optimal, error = solveEquations(A1, b1)
print("Optimal solution is ", optimal, "\n")
print("Error is ", error, "\n")

if np.linalg.norm(error) < 10**-10:
    print("The system of equations is unique")
else:
    print("The system of equations is inconsistent")
```

```
Optimal solution is  [[-1.]
 [ 1.]]

Error is  [[2.22044605e-15]
 [5.32907052e-15]
 [1.33226763e-15]]

The system of equations is unique
```

In [11]:

```
#problem 3

A1 = [[2,5,10], [9,2,1], [4,10,20]]
b1 = [[0], [1], [5]]

case, optimal, error = solveEquations(A1, b1)
print("Optimal solution is ", optimal, "\n")
print("Error is ", error, "\n")

if case == 0:
    print("The system of equations is unique")

if case == 1:
    print("The system of equations has infinite solutions")

if case == 2:
    print("The system of equations is inconsistent")
```

```
Optimal solution is  [[0.07720207]
 [0.08041451]
 [0.14435233]]

Error is  [[-2.00000000e+00]
 [-1.33226763e-15]
 [ 1.00000000e+00]]

The system of equations is inconsistent
```

In [12]:

```
#problem 4

A1 = [[2,3], [5,9], [1,1]]
b1 = [[0], [2], [-2]]
```

```
case, optimal, error = solveEquations(A1, b1)
print("Optimal solution is ", optimal, "\n")
print("Error is ", error, "\n")

if np.linalg.norm(error) < 10**-10:
    print("The system of equations is unique")
else:
    print("The system of equations is inconsistent")
```

```
Optimal solution is  [[-4.        ]
 [ 2.46153846]]

Error is  [[ 0.61538462]
 [-0.15384615]
 [-0.46153846]]

The system of equations is inconsistent
```

In [13]:

```
#problem 5

A1 = [[2,5,3], [9,2,1], [4,10,6]]
b1 = [[0], [0], [0]]

case, optimal, error = solveEquations(A1, b1)
print("Optimal solution is ", optimal, "\n")
print("Error is ", error, "\n")

if case == 0:
    print("The system of equations is unique")

if case == 1:
    print("The system of equations has infinite solutions")

if case == 2:
    print("The system of equations is inconsistent")
```

```
Optimal solution is  [[0.]
 [0.]
 [0.]]

Error is  [[0.]
 [0.]
 [0.]]

The system of equations has infinite solutions
```