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
```

Case 1 :

Consider an equation A**x**=**b** where A is a Full rank and square matrix, then the solution is given as x_{op}=A^{-1} ***b***, where x_{op} is the optimal solution and the error is given as ***b*** - Ax_{op}

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

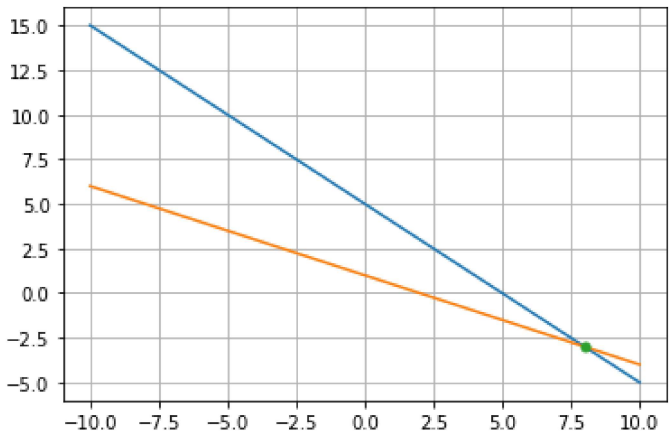
$$x + y = 5$$

$$2x + 4y = 4$$

In [ ]:
```
# 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 =\n',A,'\n')
print('b =\n',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 =\n',A_inverse,'\n')

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

# Plot the equations
# write your code here
fig = plt.figure()
ax = plt.axes()
plt.grid()
X = np.linspace(-10, 10, 100)
```

```
# The lines are of the form ax + by = c
# So, y = (c - a*x) / b
# Consider 2x + 4y = 4
# Here, y = (4 - 2*x) / 4
# To generate Y, we could hard code it for the scenario; but let's make adaptable code
# in the above, c = first ele of b, a = first ele of first row of A, x = X, b = second ele of sec
# effectively getting a transformation of X saved in y
def getY(A_row, b_num, X_in):
    Y_out = (b_num - A_row[0]*X_in)/(A_row[1])        # A_row, X_in should be row vectors for thi
    return Y_out

Y = getY(A[0,:], b[0], X)
Y2 = getY(A[1,:], b[1], X)
plt.plot(X, Y)
plt.plot(X, Y2)
plt.plot(x_op[0], x_op[1], marker="o", markersize=5)

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

plt.show()
```

```
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 :

$$x + y + z = 5$$

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

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

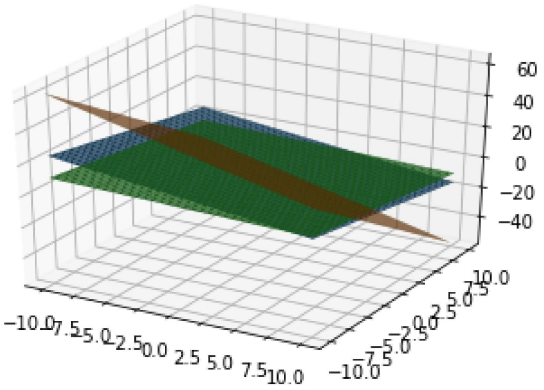
Write the code to :

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 [ ]:
```
## write your code here
# from mpl_toolkits.mplot3d import Axes3D

A = np.array([
    [1, 1, 1],
    [2, 4, 1],
    [1, 3, 4]
])
print("A =\n", A, "\n")

b = np.array([
    [5],
    [4],
    [4]
])
print("b =\n", b, "\n")


Det = np.linalg.det(A)
print('Determinant = ',Det,'\n')
rank = np.linalg.matrix_rank(A)
print('Matrix rank = ',rank,'\n')
A_inverse = np.linalg.inv(A)
print('A_inverse =\n',A_inverse,'\n')
x_op = np.matmul(A_inverse, b)
print('x =\n',x_op,'\n')

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
x, y = np.linspace(-10, 10, 100), np.linspace(-10, 10, 100)
X, Y = np.meshgrid(x, y)

# This time, ax + by + cz = d
# So, z = (d - ax - by)/c
def getZ(A_row, b_num, X_in, Y_in):
    Z_out = (b_num - A_row[0]*X_in - A_row[1]*Y_in)/(A_row[2])
    return Z_out

Z1 = getZ(A[0,:], b[0], X, Y)
Z2 = getZ(A[1,:], b[1], X, Y)
Z3 = getZ(A[2,:], b[2], X, Y)
ax.plot_surface(X, Y, Z1)
ax.plot_surface(X, Y, Z2)
ax.plot_surface(X, Y, Z3)
# ax.scatter(x_op[0], x_op[1], x_op[2])

error = b - np.matmul(A, x_op)
print('error =\n',error,'\n')
```

```
plt.show()
```

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

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

$$x + z = 0$$

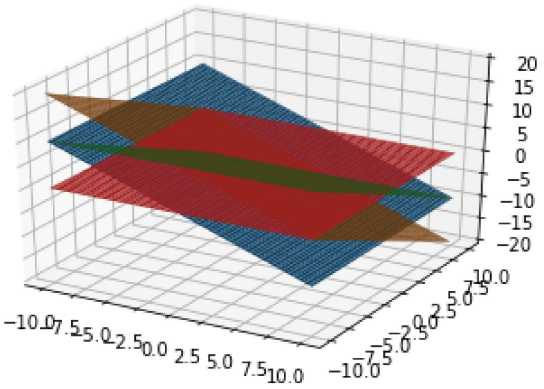
$$x + y + z = 0$$

$$y + z = 0$$

$$z = 0$$

In [ ]:
```
# 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 =\n',A,'\n')
print('b =\n',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 =\n',A_inverse,'\n')

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

# Plot the equations
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
x, y = np.linspace(-10, 10, 100), np.linspace(-10, 10, 100)
X, Y = np.meshgrid(x, y)

# This time, ax + by + cz = d
# So, z = (d - ax - by)/c
def getZ(A_row, b_num, X_in, Y_in):
    Z_out = (b_num - A_row[0]*X_in - A_row[1]*Y_in)/(A_row[2])
    return Z_out

Z1 = getZ(A[0,:], b[0], X, Y)
Z2 = getZ(A[1,:], b[1], X, Y)
Z3 = getZ(A[2,:], b[2], X, Y)
Z4 = getZ(A[3,:], b[3], X, Y)
ax.plot_surface(X, Y, Z1)
ax.plot_surface(X, Y, Z2)
ax.plot_surface(X, Y, Z3)
ax.plot_surface(X, Y, Z4)
# ax.scatter(x_op[0], x_op[1], x_op[2])

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

plt.show()
```

```
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 :

$$x + y + z = 35$$

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

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

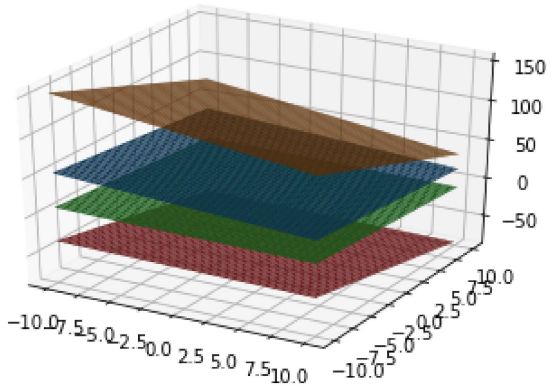
$$x + 9y + 4z = -230$$

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

rank = np.linalg.matrix_rank(A)
print('Matrix rank = ',rank,'\n')
A_inverse = np.linalg.pinv(A)
print('A_inverse =\n',A_inverse,'\n')
x_op = np.matmul(A_inverse, b)
print('x =\n',x_op,'\n')

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
x, y = np.linspace(-10, 10, 100), np.linspace(-10, 10, 100)
X, Y = np.meshgrid(x, y)

def getZ(A_row, b_num, X_in, Y_in):
    Z_out = (b_num - A_row[0]*X_in - A_row[1]*Y_in)/(A_row[2])
    return Z_out

Z1 = getZ(A[0,:], b[0], X, Y)
Z2 = getZ(A[1,:], b[1], X, Y)
Z3 = getZ(A[2,:], b[2], X, Y)
Z4 = getZ(A[3,:], b[3], X, Y)
ax.plot_surface(X, Y, Z1)
ax.plot_surface(X, Y, Z2)
ax.plot_surface(X, Y, Z3)
ax.plot_surface(X, Y, Z4)

# ax.scatter(x_op[0], x_op[1], x_op[2])

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

plt.show()
```

```
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 equation 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 x_u=A^{-1}***b*** (here A^{-1} is the pseudo inverse of matrix A), the error is given as ***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}***b*** (here A^{-1} is the pseudo inverse of matrix A) and the error is given as ***b*** - Ax_{ls}

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

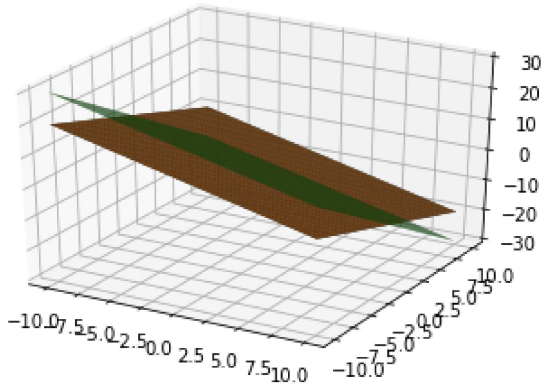
$$x + y + z = 0$$

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

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

In [ ]:
```
# Define matrix A and B
A = np.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 =\n',A,'\n')
print('b =\n',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 =\n',A_inverse,'\n')

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

# Plot the equations

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
x, y = np.linspace(-10, 10, 100), np.linspace(-10, 10, 100)
X, Y = np.meshgrid(x, y)

def getZ(A_row, b_num, X_in, Y_in):
    Z_out = (b_num - A_row[0]*X_in - A_row[1]*Y_in)/(A_row[2])
    return Z_out

Z1 = getZ(A[0,:], b[0], X, Y)
Z2 = getZ(A[1,:], b[1], X, Y)
Z3 = getZ(A[2,:], b[2], X, Y)
ax.plot_surface(X, Y, Z1)
ax.plot_surface(X, Y, Z2)
ax.plot_surface(X, Y, Z3)

# Validate the solution by computing the error
error = b - np.matmul(A, x_op) # write your code here
print('error =\n',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 :

$$x + y + z = 0$$

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

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

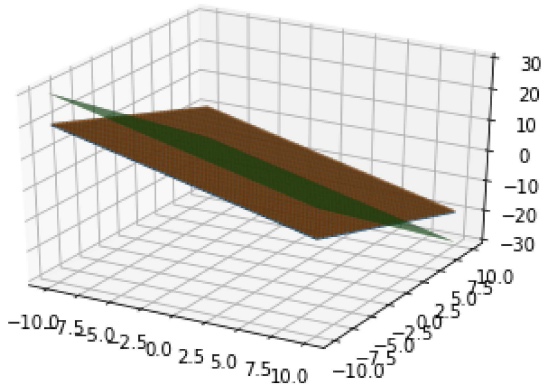
Write the code to :

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 [ ]:
```
# write your code here
A = np.array([
    [1, 1, 1],
    [3, 3, 3],
```

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

rank = np.linalg.matrix_rank(A)
print('Matrix rank = ',rank,'\n')
A_inverse = np.linalg.pinv(A)
print('A_inverse =\n',A_inverse,'\n')
x_op = np.matmul(A_inverse, b)
print('x =\n',x_op,'\n')

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
x, y = np.linspace(-10, 10, 100), np.linspace(-10, 10, 100)
X, Y = np.meshgrid(x, y)

def getZ(A_row, b_num, X_in, Y_in):
    Z_out = (b_num - A_row[0]*X_in - A_row[1]*Y_in)/(A_row[2])
    return Z_out

Z1 = getZ(A[0,:], b[0], X, Y)
Z2 = getZ(A[1,:], b[1], X, Y)
Z3 = getZ(A[2,:], b[2], X, Y)
ax.plot_surface(X, Y, Z1)
ax.plot_surface(X, Y, Z2)
ax.plot_surface(X, Y, Z3)

error = b - np.matmul(A, x_op)
print('error =\n',error,'\n')
plt.show()
```

```
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]]
```



Summary of all cases

1. A is a full-rank square matrix. Solution is given as x_{op}=A^{-1}***b***, where x_{op} is the optimal solution and the error is given as ***b*** - Ax_{op}
2. A is a full-rank non-square matrix. ($m > n$)
   - If b lies in the span of columns of A, then there is unique solution and it is given as x_u=A^{-1}***b*** (here A^{-1} is the pseudo inverse of matrix A), where x_u is the unique solution and the error is given as ***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}***b*** (here A^{-1} is the pseudo inverse of matrix A) and the error is given as ***b*** - Ax_{ls}
3. A is not a full-rank matrix.
   - If b lies in the span of columns of A, then there are multiple solutions and one of the solution is given as x_u=A^{-1}***b*** (here A^{-1} is the pseudo inverse of matrix A), the error is given as ***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}***b*** (here A^{-1} is the pseudo inverse of matrix A) and the error is given as ***b*** - Ax_{ls}

Examples

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

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

In [ ]:
```
# write your code here
A = np.array([
    [2, 3, 5],
    [9, 3, 2],
    [5, 9, 1]
])
b = np.array([
    [2],
    [5],
    [7]
])
print('A =\n',A,'\n')
print('b =\n',b,'\n')
```

```
Det = np.linalg.det(A)
print('Determinant = ',Det,'\n')
rank = np.linalg.matrix_rank(A)
print('Matrix rank = ',rank,'\n')

print("_____________________________________________________________________________")
print("Here, A is a full rank square matrix. So, this set of equations belongs to Case 1.")
print("The given system of equations has a unique optimal solution.")
print("_____________________________________________________________________________")

A_inverse = np.linalg.inv(A)
print('A_inverse =\n',A_inverse,'\n')
x_op = np.matmul(A_inverse, b)
print('x =\n',x_op,'\n')
error = b - np.matmul(A, x_op)
print('error =\n',error,'\n')
```

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

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

Determinant =  302.9999999999999

Matrix rank =  3

_____________________________________________________________________________
Here, A is a full rank square matrix. So, this set of equations belongs to Case 1.
The given system of equations has a unique optimal solution.
_____________________________________________________________________________
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 =
 [[4.4408921e-16]
 [0.0000000e+00]
 [8.8817842e-16]]
```

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

In [ ]:
```
# write your code here
A = np.array([
    [2, 3],
    [5, 9],
    [1, 1]
])
b = np.array([
    [1],
    [4],
```

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

rank = np.linalg.matrix_rank(A)
print('Matrix rank of A = ',rank,'\n')

print("_____________________________________________________________________________________")
print("Here, A is a full rank non-square matrix. So, this set of equations belongs to Case 2.")
# to deal with span part; we can augment b to A, and check if rank has increased
Aaugb = np.column_stack((A, b))
print('(A|b) =\n',Aaugb,'\n')
rank2 = np.linalg.matrix_rank(Aaugb)
print('Matrix rank of (A aug b) = ',rank2,'\n')
print("The augmented matrix is of dimensions ",Aaugb.shape, " but rank is EQUAL to that of A")
print("So, b lies in span of columns of A. This implies that the system of equations has a unique
print("This unique solution could be found out using pseudo-inverse.")
print("_____________________________________________________________________________________")

A_inverse = np.linalg.pinv(A)
print('A_inverse =\n',A_inverse,'\n')
x_op = np.matmul(A_inverse, b)
print('x =\n',x_op,'\n')
error = b - np.matmul(A, x_op)
print('error =\n',error,'\n')
```

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

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

Matrix rank of A =  2
```

```
Here, A is a full rank non-square matrix. So, this set of equations belongs to Case 2.
(A|b) =
 [[2 3 1]
 [5 9 4]
 [1 1 0]]

Matrix rank of (A aug b) =  2

The augmented matrix is of dimensions  (3, 3)  but rank is EQUAL to that of A
So, b lies in span of columns of A. This implies that the system of equations has a unique soluti
on.
This unique solution could be found out using pseudo-inverse.
```

```
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]]
```

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

In [ ]:
```
# write your code here
A = np.array([
    [2, 5, 10],
    [9, 2, 1],
    [4, 10, 20]
])
b = np.array([
    [0],
    [1],
    [5]
])
print('A =\n',A,'\n')
print('b =\n',b,'\n')

Det = np.linalg.det(A)
print('Determinant = ',Det,'\n')
rank = np.linalg.matrix_rank(A)
print('Matrix rank = ',rank,'\n')
```

```
print("_____")
print("Here, A is a non-full rank square matrix. So, this set of equations belongs to Case 3.")
# to deal with span part; we can augment b to A, and check if rank has increased
Aaugb = np.column_stack((A, b))
print('(A|b) =\n',Aaugb,'\n')
rank2 = np.linalg.matrix_rank(Aaugb)
print('Matrix rank of (A aug b) = ',rank2,'\n')
print("The augmented matrix is of dimensions ",Aaugb.shape, " but has rank that is ONE more than
print("So, b DOES NOT lie in span of columns of A. This implies that the system of equations has
print("Hence, we calculate least squares solution with the help of pseudo-inverse.")
print("_____")

A_inverse = np.linalg.pinv(A)
print('A_inverse =\n',A_inverse,'\n')
x_op = np.matmul(A_inverse, b)
print('x =\n',x_op,'\n')
error = b - np.matmul(A, x_op)
print('error =\n',error,'\n')
```

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

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

Determinant =  0.0

Matrix rank =  2

_____
Here, A is a non-full rank square matrix. So, this set of equations belongs to Case 3.
(A|b) =
 [[ 2  5 10  0]
 [ 9  2  1  1]
 [ 4 10 20  5]]

Matrix rank of (A aug b) =  3

The augmented matrix is of dimensions  (3, 4)  but has rank that is ONE more than A.
So, b DOES NOT lie in span of columns of A. This implies that the system of equations has no solu
tion.
Hence, we calculate least squares solution with the help of pseudo-inverse.
_____
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]]
```

1.

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

In [ ]:
```
# write your code here
A = np.array([
    [2, 3],
    [5, 9],
    [1, 1]
])
b = np.array([
    [0],
    [2],
    [-2]
])
print('A =\n',A,'\n')
print('b =\n',b,'\n')

rank = np.linalg.matrix_rank(A)
print('Matrix rank = ',rank,'\n')

print("_____")
print("Here, A is a full rank non-square matrix. So, this set of equations belongs to Case 2.")
# to deal with span part; we can augment b to A, and check if rank has increased
Aaugb = np.column_stack((A, b))
print('(A|b) =\n',Aaugb,'\n')
rank2 = np.linalg.matrix_rank(Aaugb)
print('Matrix rank of (A aug b) = ',rank2,'\n')
print("The augmented matrix is of dimensions ",Aaugb.shape, " but has rank that is ONE more than
print("So, b DOES NOT lie in span of columns of A. This implies that the system of equations has
print("Hence, we calculate least squares solution with the help of pseudo-inverse.")
print("_____")

A_inverse = np.linalg.pinv(A)
print('A_inverse =\n',A_inverse,'\n')
x_op = np.matmul(A_inverse, b)
print('x =\n',x_op,'\n')
error = b - np.matmul(A, x_op)
print('error =\n',error,'\n')
```

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

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

Matrix rank =  2
```

```
Here, A is a full rank non-square matrix. So, this set of equations belongs to Case 2.
(A|b) =
 [[ 2  3  0]
 [ 5  9  2]
 [ 1  1 -2]]

Matrix rank of (A aug b) =  3

The augmented matrix is of dimensions  (3, 3)  but has rank that is ONE more than A.
So, b DOES NOT lie in span of columns of A. This implies that the system of equations has no solu
tion.
Hence, we calculate least squares solution with the help of pseudo-inverse.
```

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

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

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

1. $2x + 5y + 3z = 0,$
   $9x + 2y + z = 0,$
   $4x + 10y + 6z = 0$

In [ ]:
```
# write your code here
A = np.array([
    [2, 5, 3],
    [9, 2, 1],
    [4, 10, 6]
])
b = np.array([
    [0],
    [0],
    [0]
])
print('A =\n',A,'\n')
print('b =\n',b,'\n')

Det = np.linalg.det(A)
print('Determinant = ',Det,'\n')
rank = np.linalg.matrix_rank(A)
print('Matrix rank = ',rank,'\n')
```

```
print("_____")
print("Here, A is a non-full rank square matrix. So, this set of equations belongs to Case 3.")
# to deal with span part; we can augment b to A, and check if rank has increased
Aaugb = np.column_stack((A, b))
print('(A|b) =\n',Aaugb,'\n')
rank2 = np.linalg.matrix_rank(Aaugb)
print('Matrix rank of (A aug b) = ',rank2,'\n')
print("The augmented matrix is of dimensions ",Aaugb.shape, " but has rank that is EQUAL to that
print("So, b lies in the span of columns of A. This implies that the system of equations has mult
print("The given system of equations has multiple solutions, one of which we will calculate, with
print("_____")

A_inverse = np.linalg.pinv(A)
print('A_inverse =\n',A_inverse,'\n')
x_op = np.matmul(A_inverse, b)
print('x =\n',x_op,'\n')
error = b - np.matmul(A, x_op)
print('error =\n',error,'\n')
```

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

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

Determinant =  0.0

Matrix rank =  2

--------------------------------------------------------------------------------
_____
Here, A is a non-full rank square matrix. So, this set of equations belongs to Case 3.
(A|b) =
 [[ 2  5  3  0]
 [ 9  2  1  0]
 [ 4 10  6  0]]

Matrix rank of (A aug b) =  2

The augmented matrix is of dimensions  (3, 4)  but has rank that is EQUAL to that of A.
So, b lies in the span of columns of A. This implies that the system of equations has multiple so
lutions.
The given system of equations has multiple solutions, one of which we will calculate, with the he
lp of pseudoinverse.
--------------------------------------------------------------------------------
_____
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.]]
```

In [ ]: