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

```
## Import the required Libraries here

import numpy as np
import matplotlib.pyplot as plt
%matplotlib widget
```

#Case 1 :

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

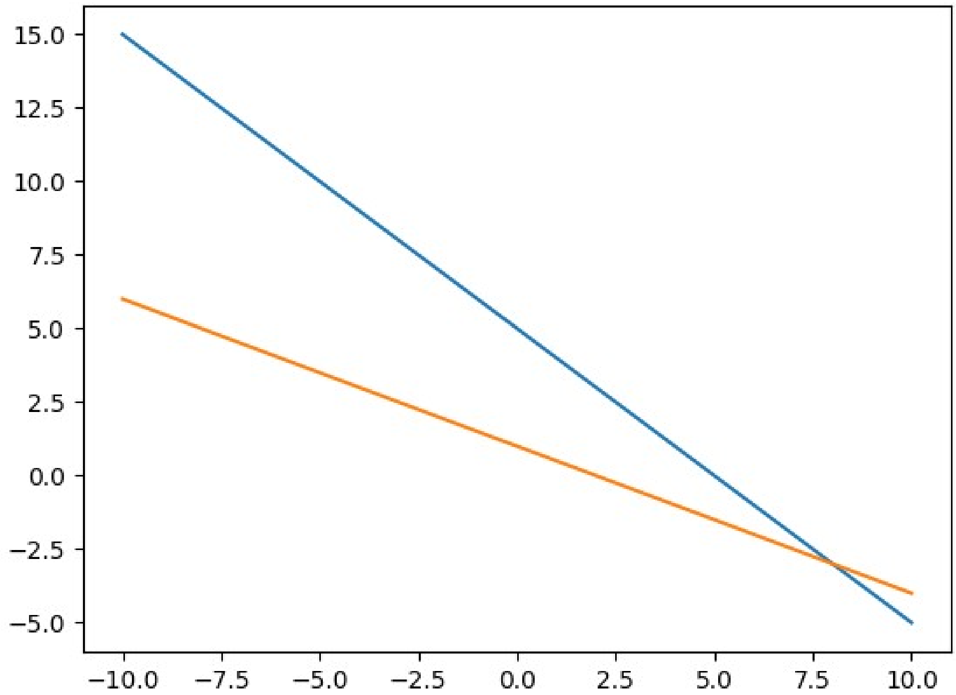
$$x+y=5$$

$$2x+4y=4$$

```
# Define Matrix A and B
from operator import matmul


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 = np.linalg.det(A)
print('Determinant=',Det,'\n')

# Determine the rank of the matrix A
rank =  np.linalg.matrix_rank(A)
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 = A_inverse @ b# write your code here
print('x=',x_op,'\n')
```

```python
# Plot the equations
x = np.linspace(-10,10)
plt.plot(x,5-x)
plt.plot(x,(4-2*x)/4)
plt.show()

# 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 :

$$x+y+z=5$$

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

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

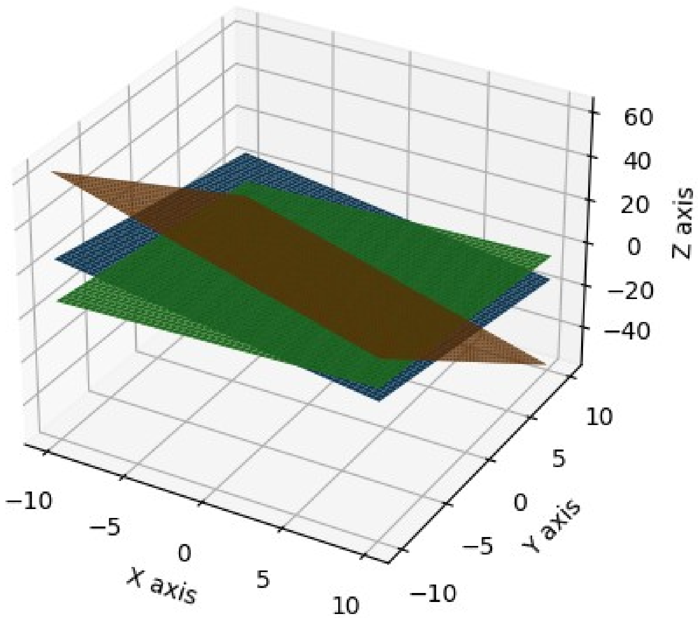
Write the code to :

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

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

```python
print(A)
print(b)
print("Deteminant = ",np.linalg.det(A)) #Determinant of A
print("Matrix rank = ",np.linalg.matrix_rank(A)) #rank of A
print("A_inverse = ",np.linalg.inv(A)) #inverse of A
print("x = ",np.linalg.inv(A) @ b) #optimal solution
print("error = ",b - A @ (np.linalg.inv(A) @ b))
#plotting
x_axis, y_axis = np.linspace(-10, 10), np.linspace(-10, 10)
X , Y = np.meshgrid(x_axis, y_axis)
Z1 = (5-X-Y)
Z2 = (4-2*X-4*Y)
Z3 = (4+X-3*Y)/4
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, Z1)
ax.plot_surface(X, Y, Z2)
ax.plot_surface(X, Y, Z3)

ax.set_xlabel('X axis')
ax.set_ylabel('Y axis')
ax.set_zlabel('Z axis')

plt.show()
```

```
[[1 1 1]
 [2 4 1]
 [1 3 4]]
[[5]
 [4]
 [4]]
Deteminant =  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 $\mathbf{x}\square_u=A^{-1}\boldsymbol{b}$ (here A^{-1} is the pseudo inverse of matrix A), where $\boldsymbol{x}\square_u$ is the unique solution and the error is given as $\boldsymbol{b}$ - $A\mathbf{x}\square_u$, 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 $\mathbf{x}\square_{ls}=A^{-1}\boldsymbol{b}$ (here A^{-1} is the pseudo inverse of matrix A) and the error is given as $\boldsymbol{b}$ - $A\mathbf{x}\square_{ls}$

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

$$x+z=0$$

$$x+y+z=0$$

$$y+z=0$$

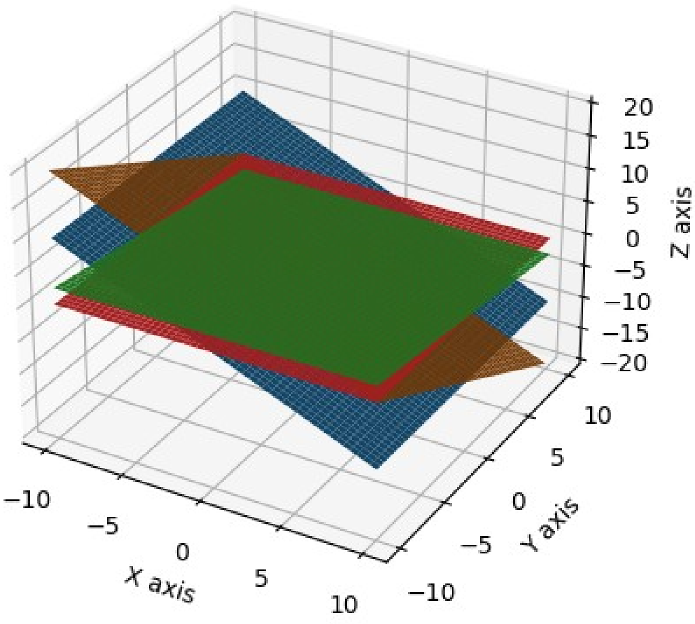
$$z=0$$

```
# 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 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
x_axis, y_axis = np.linspace(-10, 10), np.linspace(-10, 10)
X , Y = np.meshgrid(x_axis, y_axis)
Z1 = (-X)
Z2 = (-X-Y)
Z3 = (-Y)/4
Z4 = 0*X
fig = plt.figure()
ax = fig.add_subplot(111, 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)


ax.set_xlabel('X axis')
ax.set_ylabel('Y axis')
ax.set_zlabel('Z axis')

plt.show()
# 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 :

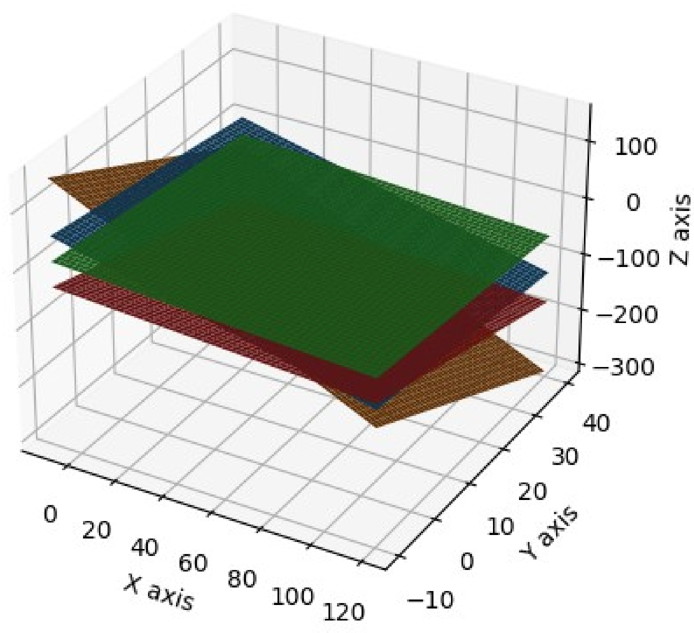
1. Define Matrices A and B

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

```
# 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
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
x_axis, y_axis = np.linspace(-10, 120), np.linspace(-10, 40)
X , Y = np.meshgrid(x_axis, y_axis)
Z1 = (35-X-Y)
Z2 = (94-2*X-4*Y)
Z3 = (4-X-3*Y)/4
Z4 = (-230-X-9*Y)/4
fig = plt.figure()
ax = fig.add_subplot(111, 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)


ax.set_xlabel('X axis')
ax.set_ylabel('Y axis')
ax.set_zlabel('Z axis')

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

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

$$x+y+z=0$$

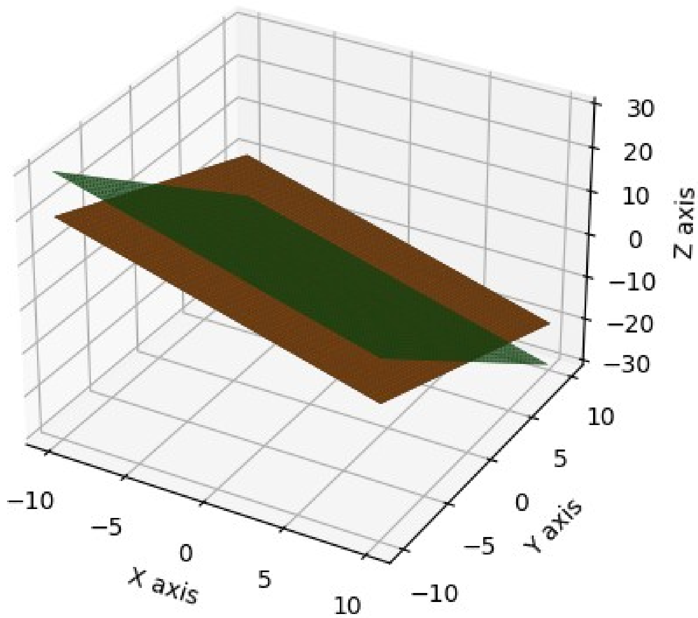
$$3x+3y+3z=0$$

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

```
# 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
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
x_axis, y_axis = np.linspace(-10, 10), np.linspace(-10, 10)
X , Y = np.meshgrid(x_axis, y_axis)
Z1 = (-X-Y)
Z2 = (-X-Y)
Z3 = (-X-2*Y)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, Z1)
ax.plot_surface(X, Y, Z2)
ax.plot_surface(X, Y, Z3)
```

```
ax.set_xlabel('X axis')
ax.set_ylabel('Y axis')
ax.set_zlabel('Z axis')

plt.show()
# 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 :

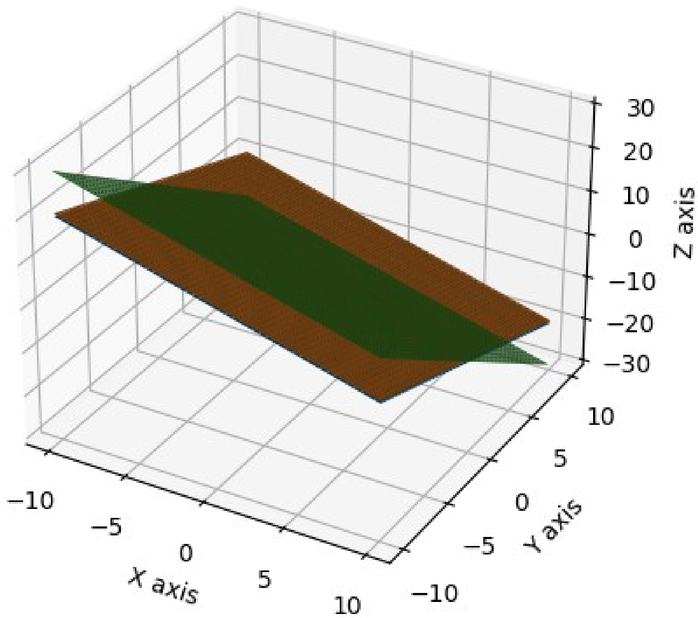
1. Define Matrices A and B
2. Determine the rank of A
3. Determine the Pseudo Inverse of matrix A
4. Determine the optimal solution
5. Plot the equations
6. Validate the solution by obataining error

```
# 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
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
x_axis, y_axis = np.linspace(-10, 10), np.linspace(-10, 10)
X , Y = np.meshgrid(x_axis, y_axis)
Z1 = (-X-Y)
Z2 = (2-3*X-3*Y)/3
Z3 = (-X-2*Y)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, Z1)
ax.plot_surface(X, Y, Z2)
ax.plot_surface(X, Y, Z3)



ax.set_xlabel('X axis')
ax.set_ylabel('Y axis')
ax.set_zlabel('Z axis')

plt.show()
# 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. $2x+5y+3z=0, 9x+2y+z=0, 4x+10y+6z=0$

```
def Solution(A,b):
    rank =  np.linalg.matrix_rank(A)
    dim = np.shape(A)
    if(dim[0] == dim[1]):
        if(rank==dim[0]):
            print("it's case-1")
            A_inverse = np.linalg.inv(A)
            x = A_inverse @ b
        elif(rank!=dim[0]):
            print("it's case-3")
            A_inverse = np.linalg.pinv(A)
            x = A_inverse @ b

    else:
            print("it's case-2")
            A_inverse = np.linalg.pinv(A)
            x = A_inverse @ b
    return x

print("1.")
print(Solution([[2,3,5],[9,3,2],[5,9,1]],[[2],[5],[7]]))

print("2. ")
print(Solution([[2,3],[5,9],[1,1]],[[1],[4],[0]]))

print("3. ")
print(Solution([[2,5,10],[9,2,1],[4,10,20]],[[0],[1],[5]]))

print("4.")
print(Solution([[2,3],[5,9],[1,1]],[[0],[2],[-2]]))

print("5. ")
print(Solution([[2,5,3],[9,2,1],[4,10,6]],[[0],[0],[0]]))
```

```
1.
it's case-1
[[ 0.38613861]
 [ 0.57425743]
 [-0.0990099 ]]
2.
it's case-2
[[-1.]
 [ 1.]]
3.
```

```
it's case-3
[[0.07720207]
 [0.08041451]
 [0.14435233]]
4.
it's case-2
[[-4.        ]
 [ 2.46153846]]
5.
it's case-3
[[0.]
 [0.]
 [0.]]
```