Name : Mudit Sand

Roll No. : 203100068

```
#Importing the libraries
import numpy as np
```

```
#DEFINING HARD CODED FUNCTIONS and GUESSED ROOTS FROM GIVEN IMAGE FOR N = 3, here x2 and x3 are taken as y and z for convinience
func1 = ['3*x - np.cos(y*z) - 0.5', '4*x**2 - 625*y**2 + 2*y - 1',
          'np.exp(-x*y) + 20*z + (10*(np.pi)-3)/3']
gue1 = [0.1, 0.1, -0.1]

func2 = ['x**2 + y - 37', 'x - y**2 - 5', 'x+y+z-3']
gue2 = [0.1, 0.1, -0.1]

func3 = ['15*x + y**2 - 4*z -13', 'x**2 + 10 * y - z -11', 'y**3 - 25*z + 22']
gue3 = [0.1, 0.1, -0.1]
```

```
## Subroutine for the root finding for function 1
def newton_multi(function, guess_vector):
  # Initialization of guesses

  x = guess_vector[0]
  y = guess_vector[1]
  z = guess_vector[2]
  f1= function[0] # Getting the function from strings
  f2= function[1]
  f3= function[2]
  x0 = np.array([[x], [y], [z]])
  j = 1
  N = 100  # Number of max iterations
  TOL = 1e-5

  while j <= N:

      d11 = 3
      d12 = z * np.sin(y * z)
      d13 = y * np.sin(y * z)
      d21 = 8 * x
      d22 = -1250*y + 2
      d23 = 0
      d31 = -y * np.exp(-x * y)
      d32 = -x * np.exp(-x * y)
      d33 = 20

      v1 = eval(f1)
      v2 = eval(f2)
      v3 = eval(f3) # Eval function evaluate the value of mathematical function

      J = np.array([[d11, d12, d13], [d21, d22, d23], [d31, d32, d33]])
      F = np.array([[v1], [v2], [v3]])
      Y = np.linalg.solve(J, -F)
      x1 = x0 + Y
      if abs(max(Y)) < TOL:
          print('Iteration number : ', j, '\nRoots-\n', x1, '\nand x1 - x0 is : \n', abs(x1 - x0))
          return
      print('Iteration number : ', j, '\nRoots-\n', x1)
      x, y, z = x1[0][0], x1[1][0], x1[2][0]
      x0 = x1
      j = j + 1
  else:
    print('Not converges or change the initial guess or look into the above iterations if solution is there ')
    print('Increase Iteration\n', 'Iteration no-', k, '\nRoots-\n', x1, )
    return

newton_multi(func1, gue1)
```

```
Iteration number :  1
Roots-
 [[ 0.499861  ]
 [ 0.04560885]
 [-0.52139111]]
Iteration number :  2
```

```
Roots-
 [[ 0.49999599]
 [ 0.02363322]
 [-0.52300832]]
Iteration number :  3
Roots-
 [[ 0.49999812]
 [ 0.01267443]
 [-0.52328217]]
and x1 - x0 is :
 [[2.13862082e-06]
 [1.09587872e-02]
 [2.73849404e-04]]
```

```
## Subroutine for the root finding for function 2
def newton_multi(function, guess_vector):
  # Initialization of guesses

  x = guess_vector[0]
  y = guess_vector[1]
  z = guess_vector[2]
  f1= function[0] # Getting the function from strings
  f2= function[1]
  f3= function[2]
  x0 = np.array([[x], [y], [z]])
  j = 1
  N = 100  # Number of max iterations
  TOL = 1e-5

  while j <= N:

      d11 = 2
      d12 = 1
      d13 = 0
      d21 = 1
      d22 = -2*x
      d23 = 0
      d31 = 1
      d32 = 1
      d33 = 1

      v1 = eval(f1)
      v2 = eval(f2)
      v3 = eval(f3) # Eval function evaluate the value of mathematical function

      J = np.array([[d11, d12, d13], [d21, d22, d23], [d31, d32, d33]])
      F = np.array([[v1], [v2], [v3]])
      Y = np.linalg.solve(J, -F)
      x1 = x0 + Y
      if abs(max(Y)) < TOL:
          print('Iteration number : ', j, '\nRoots-\n', x1, '\nand x1 - x0 is : \n', abs(x1 - x0))
          return
      print('Iteration number : ', j, '\nRoots-\n', x1)
      x, y, z = x1[0][0], x1[1][0], x1[2][0]
      x0 = x1
      j = j + 1
  else:
    print('Not converges or change the initial guess or look into the above iterations if solution is there ')
    print('Increase Iteration\n', 'Iteration no-', j, '\nRoots-\n', x1, )
    return

newton_multi(func2, gue2)
```

```
 [[nan]
 [nan]
 [nan]]
Iteration number :  82
Roots-
 [[nan]
 [nan]
 [nan]]
Iteration number :  83
Roots-
 [[nan]
 [nan]
 [nan]]
Iteration number :  84
Roots-
 [[nan]
 [nan]
 [nan]]
```

```
 [nan]]
Iteration number :  85
Roots-
 [[nan]
 [nan]
 [nan]]
Iteration number :  86
Roots-
 [[nan]
 [nan]
 [nan]]
Iteration number :  87
Roots-
 [[nan]
 [nan]
 [nan]]
Iteration number :  88
Roots-
 [[nan]
 [nan]
 [nan]]
Iteration number :  89
Roots-
 [[nan]
 [nan]
 [nan]]
Iteration number :  90
Roots-

 [[nan]
 [nan]
 [nan]]
Iteration number :  91
Roots-
 [[nan]
 [nan]
 [nan]]
Iteration number :  92
Roots-
 [[nan]
 [nan]
 [nan]]
```

```
## Subroutine for the root finding for function 2
def newton_multi(function, guess_vector):
  # Initialization of guesses

  x = guess_vector[0]
  y = guess_vector[1]
  z = guess_vector[2]
  f1= function[0] # Getting the function from strings
  f2= function[1]
  f3= function[2]
  x0 = np.array([[x], [y], [z]])
  j = 1
  N = 100  # Number of max iterations
  TOL = 1e-5

  while j <= N:

      d11 = 15
      d12 = 2*y
      d13 = -4
      d21 = 2*x
      d22 = 10
      d23 = -1
      d31 = 0
      d32 = 3*y**2
      d33 = -25

      v1 = eval(f1)
      v2 = eval(f2)
      v3 = eval(f3) # Eval function evaluate the value of mathematical function

      J = np.array([[d11, d12, d13], [d21, d22, d23], [d31, d32, d33]])
      F = np.array([[v1], [v2], [v3]])
      Y = np.linalg.solve(J, -F)
      x1 = x0 + Y
      if abs(max(Y)) < TOL:
          print('Iteration number : ', j, '\nRoots-\n', x1, '\nand x1 - x0 is : \n', abs(x1 - x0))
          return
      print('Iteration number : ', j, '\nRoots-\n', x1)
      x, y, z = x1[0][0], x1[1][0], x1[2][0]
      x0 = x1
```

```
      j = j + 1
  else:
    print('Not converges or change the initial guess or look into the above iterations if solution is there ')
    print('Increase Iteration\n', 'Iteration no-', j, '\nRoots-\n', x1, )
    return

newton_multi(func3, gue3)
```

```
Iteration number :  1
Roots-
 [[1.08678695]
 [1.16739635]
 [0.88132088]]
Iteration number :  2
Roots-
 [[1.036588  ]
 [1.08583039]
 [0.93029866]]
Iteration number :  3
Roots-
 [[1.03640047]
 [1.08570655]
 [0.93119144]]
Iteration number :  4
Roots-
 [[1.03640047]
 [1.08570655]
 [0.93119144]]
and x1 - x0 is :
 [[1.32369449e-10]
 [3.33674199e-09]
 [1.52617163e-09]]
```