Higher Institute of Engineering & Technology, El-Beheira

Computer Engineering Department

Third assignment in numerical analysis

The numerical solution of a system of non-linear equations using Newton's Method.)

Under supervision of Dr.Mahmoud Gamal

Team	**ID**
Mohamed Yosry El-Zarka	19100
Youssef Mohamed El-Sheheimy	19124
Omar Abd Al-Halim Khalil	19138

Source code in python: -

```
import math
from sympy import * #for differentiation & mathematical functions
import numpy as np #matrix operations

pi=3.141592653589793
e=2.718281828459045

print('Project for "Numerical analysis". under the supervision of Dr. Mahmoud Gamal')
print('by:')
print('\t\tMohamed Yosry ElZarka 19100')
print('\t\tYoussef Mohamed Elsheheimy 19124')
print('\t\tOmar Abd Al-Halim Khalil 19138\n')

print("This is a program to calculate the numerical solution of a system of non-linear equations using (Newton's method).\n")

print("""
you can use parentheses () in addition to the following mathematical operators:
(+ Add), (- Subtract), (* Multiply), (/ Divide), (% Modulus), (// Floor division), (** Exponent)
you can also use the following constants:
\t pi=3.141592653589793
\t e=2.718281828459045
note: Trigonometric functions sin(x), asin(x), cos(x), acos(x), tan(x), atan(x) 'equivalent of tan-1(x)' use radian values.
      log(x,y)= log(x) / log(y) ,,, ln(x)
""")

def check_equalization(recent_x,previous_x):
    for i in range (0, len(recent_x)):
        if recent_x[i] != previous_x[i]:
            return False
    return True

decimal_point_precision=4

while True:
    n=int( input('Enter the number of equations: ') )
    equations, equations_values, jacobian_matrix, jacobian_values= [] , [0]*n , [] , []

    for i in range(0,n):
```

```python
        equations.append( str(input("Enter the equation #{}: 0 = ".format(i+1))) )
        jacobian_matrix.append([])
        jacobian_values.append([0]*n)
    last_x , current_x= [] , []
    for i in range(0,n):
        last_x.append( float(input("Enter the initial x{} = ".format(i+1))) )

    for i in range(0,n): #partial differntiaion matrix
        for j in range(0,n):
            jacobian_matrix[i].append(  diff( equations[i] , 'x'+str(j+1) ) )

    print("\njacobian matrix=",jacobian_matrix)

    print("\n        ",end="")
    for i in range(0,n):
        print("x{}          ".format(i+1),end="")
    print("")
    print("i=0",end="")
    for i in range(0,n):
        last_x[i]=round( last_x[i] , decimal_point_precision )
        print(" | %.4f | "%last_x[i],end="")
    print("")

    dictionary_of_last_x={}

    for iterations in range(1,500): #maximum number of iterations is 500
        for i in range(0,n): #updating the values of matrices
            dictionary_of_last_x['x'+str(i+1)]=last_x[i]
        for i in range(0,n):
            equations_values[i]=round( eval(equations[i],dictionary_of_last_x) , decimal_point_precision)
        for i in range(0,n):
            for j in range(0,n):
                jacobian_values[i][j]=round( eval(str(jacobian_matrix[i][j]),dictionary_of_last_x) , decimal_point_precision )

        A = np.array(last_x) #matrix declarations
        B = np.array(jacobian_values)
        C = np.array(equations_values)

        current_x=np.subtract(A, np.dot( np.linalg.inv(B)  , C ) ) #matrix operations
```

```
        print("i={}".format(iterations),end="")
        for i in range(0,n):
            current_x[i]=round( current_x[i] , decimal_point_precision )
            print(" | %.4f | "%current_x[i],end="")
        print("")
        if check_equalization(current_x,last_x):
            break
        last_x=current_x

    print("\nAfter",iterations,"iterations, The solution is at")
    for i in range(0,n):
        print("\t\tx{} =".format(i+1),last_x[i])
    print("\n____________________________________________")
    print("Try another system of non-linear equations.")
```

The program

C:\WINDOWS\py.exe

```
Project for "Numerical analysis". under the supervision of Dr. Mahmoud Gamal
by:
                Mohamed Yosry ElZarka 19100
                Youssef Mohamed Elsheheimy 19124
                Omar Abd Al-Halim Khalil 19138

This is a program to calculate the numerical solution of a system of non-linear equations using (Newton's method).


you can use parentheses () in addition to the following mathematical operators:
(+ Add), (- Subtract), (* Multiply), (/ Divide), (% Modulus), (// Floor division), (** Exponent)
you can also use the following constants:
         pi=3.141592653589793
         e=2.718281828459045
note: Trigonometric functions sin(x), asin(x), cos(x), acos(x), tan(x), atan(x) 'equivalent of tan-1(x)' use radian values.
      log(x,y)= log(x) / log(y) ,,, ln(x)

Enter the number of equations: 2
Enter the equation #1:    0 = x1**2+x2**2-4
Enter the equation #2:    0 = 2*x1-x2**2
Enter the initial x1 = 1
Enter the initial x2 = 1

jacobian matrix= [[2*x1, 2*x2], [2, -2*x2]]

        x1          x2
i=0 | 1.0000 |  | 1.0000 |
i=1 | 1.2500 |  | 1.7500 |
i=2 | 1.2361 |  | 1.5813 |
i=3 | 1.2361 |  | 1.5723 |
i=4 | 1.2361 |  | 1.5723 |

After 4 iterations, The solution is at
                x1 = 1.2361
                x2 = 1.5723


____________________________________________
Try another system of non-linear equations.
Enter the number of equations: 3
Enter the equation #1:    0 = x1**2+x2**2+x3**2-6
```

C:\WINDOWS\py.exe

```
i=1 | 1.2500 |  | 1.7500 |
i=2 | 1.2361 |  | 1.5813 |
i=3 | 1.2361 |  | 1.5723 |
i=4 | 1.2361 |  | 1.5723 |

After 4 iterations, The solution is at
                x1 = 1.2361
                x2 = 1.5723


____________________________________________
Try another system of non-linear equations.
Enter the number of equations: 3
Enter the equation #1:    0 = x1**2+x2**2+x3**2-6
Enter the equation #2:    0 = x1**2-x2**2+2*x3**2-2
Enter the equation #3:    0 = 2*x1**2+x2**2-x3**2-3
Enter the initial x1 = 4
Enter the initial x2 = 4
Enter the initial x3 = 4

jacobian matrix= [[2*x1, 2*x2, 2*x3], [2*x1, -2*x2, 4*x3], [4*x1, 2*x2, -2*x3]]

        x1          x2          x3
i=0 | 4.0000 |  | 4.0000 |  | 4.0000 |
i=1 | 2.1250 |  | 2.3750 |  | 2.2500 |
i=2 | 1.2978 |  | 1.8191 |  | 1.5694 |
i=3 | 1.0342 |  | 1.7341 |  | 1.4219 |
i=4 | 1.0006 |  | 1.7320 |  | 1.4142 |
i=5 | 1.0000 |  | 1.7320 |  | 1.4142 |
i=6 | 1.0000 |  | 1.7320 |  | 1.4142 |

After 6 iterations, The solution is at
                x1 = 1.0
                x2 = 1.732
                x3 = 1.4142


____________________________________________
Try another system of non-linear equations.
Enter the number of equations:
```