AIN SHAMS UNIVERSITY

FACULTY OF ENGINEERING

SENIOR-2 MECHATRONICS ENGINEERING

(CSE 473s): Computational Intelligence

Major Task Submission

Milestone (1)

NAME	ID
Elsayed Ayman Elsayed Ali Habib	1804765
Mahmoud Ayman Abd elaziz	1802936
Anas Ahmed Talaat Khaled	1806766
Abdelrhman Motawea Ali	1801446

Submitted to:

Dr. Hossam El-Din Hassan Abd El Munim

Table of Content

- Problem Definition & Importance……………………………………………6
- Non-linear equations…………………………………………….……..7
- Formulated as a minimization…………………………………..……7
- Gradient Vector……………………………………………………….….8
- Hessian matrix……………………………………………………….…..8
- Gradient Descent Method……………………………………..……....9
- Newton-Raphson Method…………………………….………..9
- Steepest Gradient Descent……………………………..………10
- Gradient Descent experimental results…………………….………..12
    - First point…………………………………………..………..12
    - Second Point…………………………………………..……...13
    - Third Point………………………………….……………...14
- Newton-Raphson experimental results………………………………..15
    - First point……………………………………….………...15
    - Second point ……………………………………………...16
    - Third point………………………………………….……...17
- Steepest Gradient Descent experimental results……………………....18
    - First point ……………………………………..……......18
    - Second point………………………………………………....19
    - Third point ……………………………………………......20
- Gradient Descent Method Code..21
- Newton-Raphson Method Code…………………………………………22
- Steepest Gradient Descent Method Code…………………….……...23
- References…………………………………………………...……24

Table of Figures

- Figure 1 Gradient Vector ……………………………………………..……..8
- Figure 2 Hessian matrix………………………………..………....……..8
- Gradient Descent experimental results
    - First Point
        - Figure 3 Number of iterations & f(x)……………………….………12
        - Figure 4 Number of iterations & Amplitude $|\nabla f(Xn)|$......................12
        - Figure 5 Number of iterations & First Element of Xn…………..……..12
        - Figure 6 Number of iterations & Second Element of Xn..................12
        - Figure 7 Number of iterations & Third Element of Xn.....................12
        - Figure 8 (3-D) First point of Gradient descent
    - Second Point
        - Figure 3 Number of iterations & f(x) 13
        - Figure 4 Number of iterations & Amplitude $|\nabla f(Xn)|$ 13
        - Figure 5 Number of iterations & First Element of Xn.....................13
        - Figure 6 Number of iterations & Second Element of Xn.................13
        - Figure 7 Number of iterations & Third Element of Xn....................13
        - Figure 14 (3-D) Second point of Gradient descent..........................13
    - Third Point
        - Figure 3 Number of iterations & f(x) 14
        - Figure 4 Number of iterations & Amplitude $|\nabla f(Xn)|$ 14
        - Figure 5 Number of iterations & First Element of Xn.....................14
        - Figure 6 Number of iterations & Second Element of Xn..................14

- Figure 7 Number of iterations & Third Element of Xn…………….....14
- Figure 20 (3-D) Third point of Gradient descent……………………..14

- Newton-Raphson experimental results
  - First Point
    - Figure 3 Number of iterations & f(x)……………………………….…15
    - Figure 4 Number of iterations & Amplitude $|\nabla f(Xn)|$ …………….…15
    - Figure 5 Number of iterations & First Element of Xn……………...15
    - Figure 6 Number of iterations & Second Element Xn……………...15
    - Figure 7 Number of iterations & Third Element of Xn……………..15
    - Figure 26 (3-D) First point of Newton-Raphson……………………15
  - Second Point
    - Figure 3 Number of iterations & f(x) ………………………………16
    - Figure 4 Number of iterations & Amplitude $|\nabla f(Xn)|$ …………….…16
    - Figure 5 Number of iterations & First Element of Xn……………...16
    - Figure 6 Number of iterations & Second Element of Xn…….……..16
    - Figure 7 Number of iterations & Third Element of Xn…………..…16
    - Figure 26 (3-D) First point of Newton-Raphson…………………....16
  - Third Point
    - Figure 3 Number of iterations & f(x) ………………………………17
    - Figure 4 Number of iterations & Amplitude $|\nabla f(Xn)|$ …………….17
    - Figure 5 Number of iterations & First Element of Xn……………...17
    - Figure 6 Number of iterations & Second Element of Xn…….……..17
    - Figure 7 Number of iterations & Third Element of Xn…………..…17
    - Figure 26 (3-D) First point of Newton-Raphson…………………....17

- Steepest Gradient Descent experimental results
  - First Point
    - Figure 3 Number of iterations & f(x) ………………………………18
    - Figure 4 Number of iterations & Amplitude $|\nabla f(Xn)|$ ………….…18
    - Figure 5 Number of iterations & First Element of Xn……………...18
    - Figure 6 Number of iterations & Second Element of Xn…………..18
    - Figure 7 Number of iterations & Third Element of Xn………….……18
    - Figure 44 (3-D) First point of Steepest Gradient Descent………….…18
  - Second Point
    - Figure 3 Number of iterations & f(x) ………………………………19
    - Figure 4 Number of iterations & Amplitude $|\nabla f(Xn)|$ ………….……19
    - Figure 5 Number of iterations & First Element of Xn……………..19
    - Figure 6 Number of iterations & Second Element of Xn…………..19
    - Figure 7 Number of iterations & Third Element of Xn………….……19
    - Figure 44 (3-D) First point of Steepest Gradient Descent…………….19
  - Third Point
    - Figure 3 Number of iterations & f(x) ………………………………20
    - Figure 4 Number of iterations & Amplitude $|\nabla f(Xn)|$ ……………..20
    - Figure 5 Number of iterations & First Element of Xn……………..20
    - Figure 6 Number of iterations & Second Element of Xn…….……20
    - Figure 7 Number of iterations & Third Element of Xn…………..…20
    - Figure 44 (3-D) First point of Steepest Gradient Descent………….…20

Problem Definition & Importance

Mathematical Optimization, also known as Mathematical Programming, Operations Research is a discipline that solves a great variety of applied problems in diverse areas: medicine, manufacturing, transportation, supply chain, finance, government,physics, economics, artificial intelligence, etc.
In our daily lives, we benefit from the application of Mathematical Optimization algorithms. They are used, for example, by GPS systems, by shipping companies delivering packages to our homes, by financial companies, airline reservations systems, etc.

Some of importance benefits:

- **Investigate thousands of possible solutions and find the best ones: There are decisions that involve so many variables and possibilities that it is just impossible for a human toobserve all the possible solutions and identify the best one. Some examples are problems such as vehicle routing, production scheduling, and packing - their complexity increases exponentially as the size of the problem increases, and millions (or even billions and trillions) of solutions can exist.**
- **Increase responsiveness and time efficiency reduce the time of the decision-makingprocess and allow the users to focus their time on the analyses.**

In mathematics, computer science and economics, an optimization problem is the problem of finding the best solution from all feasible solutions.
Optimization problems can be divided into categories, depending on whether the variables are continuous or discrete: An optimization problem with discrete variables is known as a discrete optimization, in which an object such as an integer,permutation or graph must be found from a countable set. A problem with continuous variables is known as a continuous optimization, in which an optimal value from a continuous function must be found. They can include constrained problems and multimodal problems.

we are required to optimize a given set of non-linear equations using different optimization techniques (gradient decent – newton Raphson – steepest gradient decent) & get gradient magnitude & function value and plotting them with different Initial points to see theeffectiveness and performance of each Algorithm.

The main required is to get the variables valuesthat give the minimum value of the main function using different ways (global minimum point).

- **The objective of this part is to find a solution for the following set of non-linear equations using different optimization techniques:**

$$g_1(x_1, x_2, x_3) = 3x_1 - \cos(x_2 x_3) - 0.5 = 0$$

$$g_2(x_1, x_2, x_3) = x_1^2 - 81(x_2 + 0.1)^2 + \sin(x_3) + 1.06 = 0$$

$$g_3(x_1, x_2, x_3) = e^{-x_2 x_3} + 20x_3 + \frac{(10\pi - 3)}{3} = 0$$

- **The problem can be formulated as a minimization of the following suggested objective function:**

$$F(x_1, x_2, x_3) = \frac{1}{2}[g_1(x_1, x_2, x_3)\ ]^2 + \frac{1}{2}[g_2(x_1, x_2, x_3)\ ]^2 + \frac{1}{2}[g_3(x_1, x_2, x_3)\ ]^2$$

$$F(x_1, x_2, x_3) = \frac{1}{2}\ [3x_1 - \cos(x_2 x_3) - 0.5]^2 + \frac{1}{2}\ [{x_1}^2 - 81(x_2 + 0.1)^2 + \sin(x_3) + 1.06]^2 + \frac{1}{2}\ [e^{-x_1 x_2} + 20x_3 + \frac{(10\pi - 3)}{3}]^2$$

❖ We’ll discuss the problem definition, the importance, formula & how to solve analytically of the following cases:

Gradient Vector: The gradient of a scalar-valued multivariable function f(x,y,…),denoted ∇f, packages all its partial derivative information into a vector:

$$\nabla f = \begin{pmatrix} \dfrac{\partial f}{\partial x_1} \\ \dfrac{\partial f}{\partial x_2} \\ \vdots \\ \dfrac{\partial f}{\partial x_n} \end{pmatrix}$$

```python
#
# Gradiant Vector = [df/dx1]
#                   |df/dx2|
#                   [df/dx3]
#                          3*1
#
Gradiant_Vector = np.matrix(
                            [
                                [dfdx_1],
                                [dfdx_2],
                                [dfdx_3]
                            ]
)
```

Figure 1 Gradient Vector

Hessian matrix: It is a square matrix of second partial derivatives of a function. It is often used in machine learning & data science algorithms for optimizing a function.

```python
#
# Hessian matrix of the function f
#
H = np.matrix(
    [
        [df2dx2_1,df2dx_1dx_2,df2dx_1dx_3],
        [df2dx_2dx_1,df2dx2_2,df2dx_2dx_3],
        [df2dx_3dx_1,df2dx_3dx_2,df2dx2_3]
    ]
)
```

$$H_{(3\times 3)} = \begin{bmatrix} \dfrac{\partial^2 F}{\partial x_1^2} & \dfrac{\partial^2 F}{\partial x_1 \partial x_2} & \dfrac{\partial^2 F}{\partial x_1 \partial x_3} \\ \dfrac{\partial^2 F}{\partial x_2 \partial x_1} & \dfrac{\partial^2 F}{\partial x_2^2} & \dfrac{\partial^2 F}{\partial x_2 \partial x_3} \\ \dfrac{\partial^2 F}{\partial x_3 \partial x_1} & \dfrac{\partial^2 F}{\partial x_3 \partial x_2} & \dfrac{\partial^2 F}{\partial x_3^2} \end{bmatrix}$$

Figure 2 Hessian matrix

- **Gradient Descent Method:** is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function. To find a local minimum of a function using gradient descent, we take steps proportional to the negative of the gradient (or approximate gradient) of the function at the current point. But if we instead take steps proportional to the positive of the gradient, we approach a local maximum of that function; the procedure is then known as gradient ascent.

- The main idea of the Gradient Descent Algorism:

<u>*The algorism keeps running until the gradient amplitude is smaller than the stop value.*</u>

  - Calculate the amplitude of the gradient descent as following Equation:

$$|\nabla f(X_n)| = \sqrt{\left(\frac{df}{dx_1}\right)^2 + \left(\frac{df}{dx_2}\right)^2 + \left(\frac{df}{dx_3}\right)^2}$$

  - Calculate the New Minimum Point The equation of gradient descent:

$$\underbrace{\boxed{X_{n+1}}}_{\text{New position}} \boxed{=} \underbrace{\boxed{X_n}}_{\text{Old position}} \boxed{-} \underbrace{\boxed{\eta}}_{\text{Learning rate}} \underbrace{\boxed{\nabla f(X_n)}}_{\text{Gradient matrix}}$$

  - Update old position with the new position in next iteration.

- **Newton-Raphson Method**: aims at estimating the roots of a function. For this purpose, an initial approximation is chosen, after this, the equation of the tangent line of the function at this point and the intersection of it with the axis of the abscissa, to find a better approximation for the root, is calculated.
  By repeating the process, an iterative method is created to find the root of the function.

- The main idea of the Newton-Raphson Algorism:

  The algorism keeps running until the gradient amplitude is smaller than the stop value.

  - Calculate the amplitude of the gradient:

$$|\nabla f(X_n)| = \sqrt{\left(\frac{df}{dx_1}\right)^2 + \left(\frac{df}{dx_2}\right)^2 + \left(\frac{df}{dx_3}\right)^2}$$

  - The equation of Newton-Raphson:

$$\underbrace{X_{n+1}}_{\text{New position}} = \underbrace{X_n}_{\text{Old position}} - \underbrace{H^{-1}}_{\text{Hessian inverse}} * \underbrace{\nabla f(X_n)}_{\text{Gradient matrix}}$$

  - Update old position with the new position in next iteration.

- **Steepest Gradient Descent Method**: is a special case of gradient descent where the step length is chosen to minimize the objective function value. Gradient descent refers to any of a class of algorithms that calculate the gradient of the objective function, then move "downhill" in the indicated direction; the step length can be fixed, estimated by Newton-Raphson technique to get desired Minimum Step Length.

- The main idea of the Steepest descent Algorism:

The algorism keeps running until the gradient amplitude is smaller than the stop value.

- Calculate the amplitude of the gradient descent:

$$|\nabla f(X_n)| = \sqrt{\left(\frac{df}{dx_1}\right)^2 + \left(\frac{df}{dx_2}\right)^2 + \left(\frac{df}{dx_3}\right)^2}$$

- The equation of steepest gradient descent:

$$\underbrace{\boxed{X_{n+1}}}_{\text{New position}} \; \boxed{=} \; \underbrace{\boxed{X_n}}_{\text{Old position}} \; \boxed{-} \; \underbrace{\boxed{\eta}}_{\text{Learning rate}} \; \underbrace{\boxed{\nabla f(X_n)}}_{\text{Gradient matrix}}$$

- The algorism keeps running until the gradient amplitude (learning rate) is smaller than the stop value (learning rate).

$$\underbrace{\boxed{\eta_{n+1}}}_{\text{New learning rate}} \; \boxed{=} \; \underbrace{\boxed{\eta_n}}_{\text{Old learning rate}} \; \boxed{-} \; \underbrace{\boxed{\eta'}}_{\text{First derivative}} \; \boxed{/} \; \underbrace{\boxed{\eta``}}_{\text{Second derivative}}$$

- Update old Learning Rate with the new Learning Rate in next iteration.
- Calculate the New Position and Update The old Position with this Value.

Experimental Results (samples of your trails) and discussions.

1. Gradient Descent.

1. First point:

```
previous = ([[1.00],
             [0.00],
             [0.00]])
```

minimum Point →

```
[[ 0.6188]
 [ 0.0112]
 [-0.5207]]
```

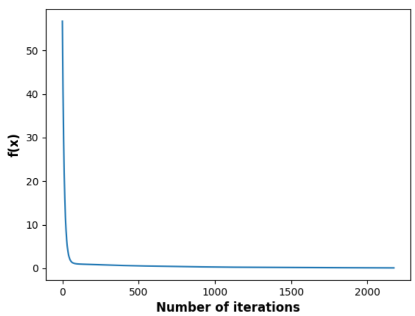

Figure 3 Number of iterations & f(x)

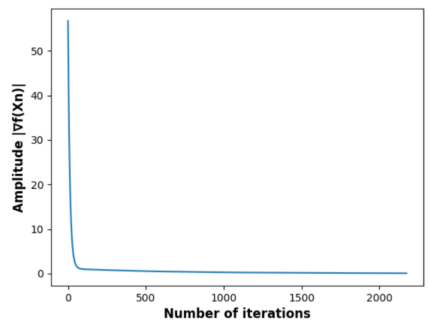

Figure 4 Number of iterations & Amplitude $|\nabla f(X_n)|$

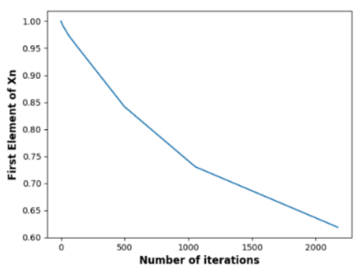

Figure 5 Number of iterations & First Element of Xn

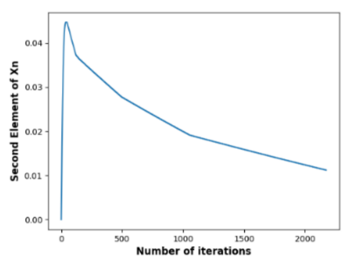

Figure 6 Number of iterations & Second Element of Xn

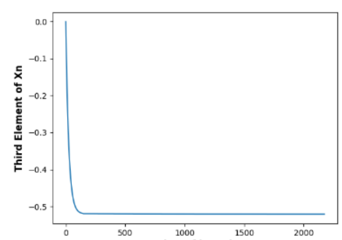

Figure 7 Number of iterations & Third Element of Xn

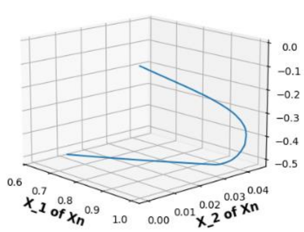

Figure 8 (3-D) First point of Gradient descent

2. Second Point:

```
previous = ([[-1.00],
             [0.00],
             [2.00]])
```

minimum Point:

```
[[ 0.3831]
 [-0.0111]
 [-0.5219]]
```

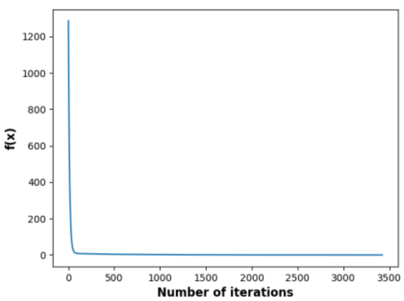

Figure 9 Number of iterations & f(x)

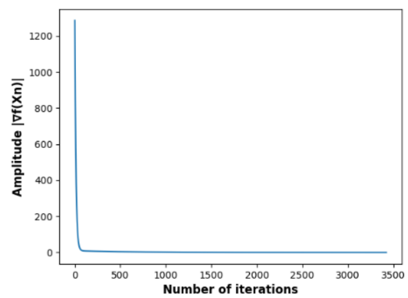

Figure 10 Number of iterations & Amplitude $|\nabla f(X_n)|$

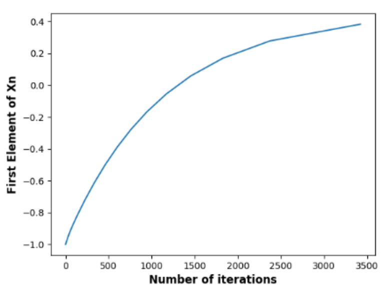

Figure 11 Number of iterations & First Element of Xn

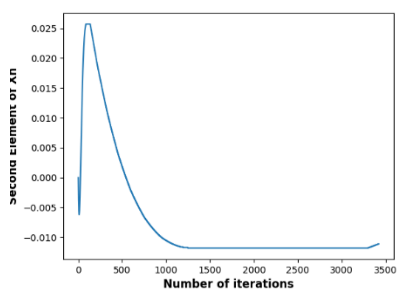

Figure 12 Number of iterations & Second Element of Xn

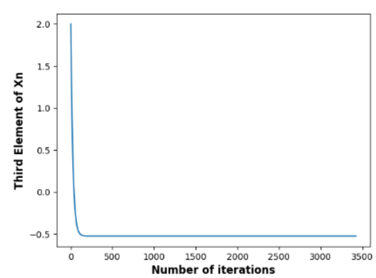

Figure 13 Number of iterations & Third Element of Xn

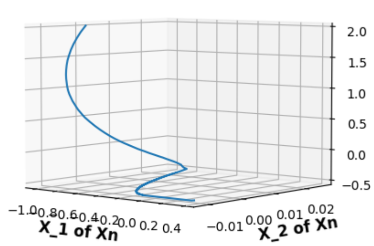

Figure 14 (3-D) Second point of Gradient descent

3. **Third Point:**

```
previous = ([[5.00],
             [1.00],
             [-1.00]])
```

minimum Point:

```
[[ 0.6178]
 [-0.2105]
 [-0.5329]]
```

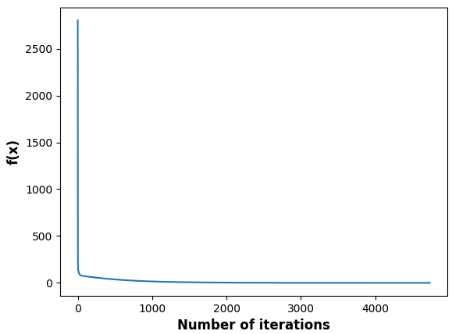

Figure 15 Number of iterations & f(x)

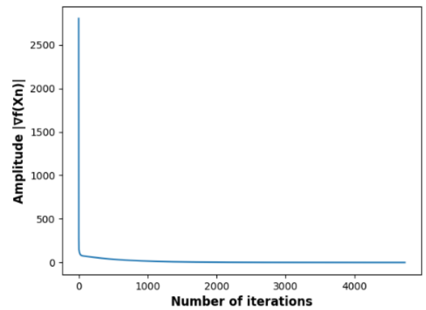

Figure 16 Number of iterations & Amplitude $|\nabla f(X_n)|$

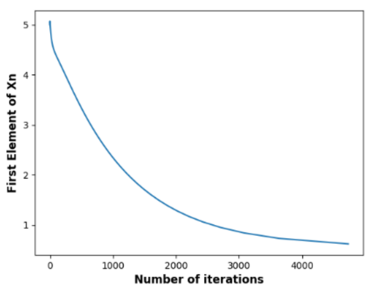

Figure 17 Number of iterations & First Element of Xn

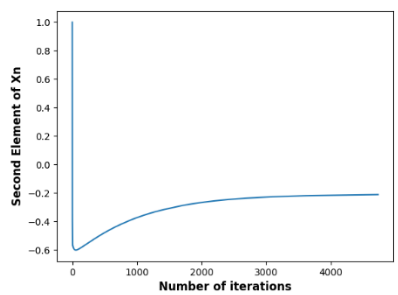

Figure 18 Number of iterations & Second Element of Xn

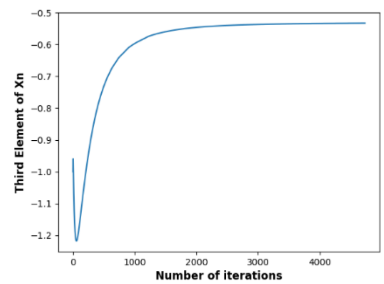

Figure 19 Number of iterations & Third Element of Xn

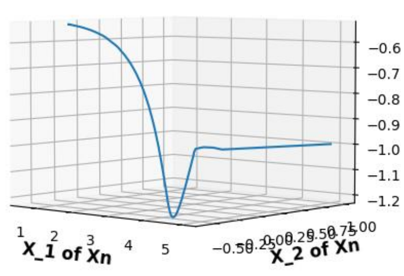

Figure 20 (3-D) Third point of Gradient descent

2. Newton-Raphson.

1. First point:

```
previous = ([[1.00],
             [0.00],
             [0.00]])
```

minimum Point →

```
[[ 0.46931602]
 [ 0.00175519]
 [-0.5240503 ]]
```

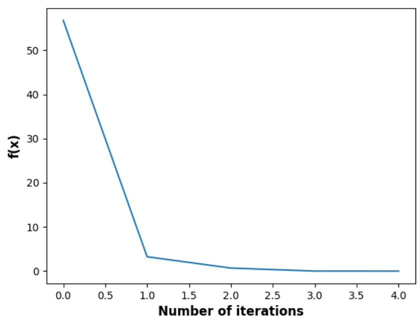

Figure 21 Number of iterations & f(x)

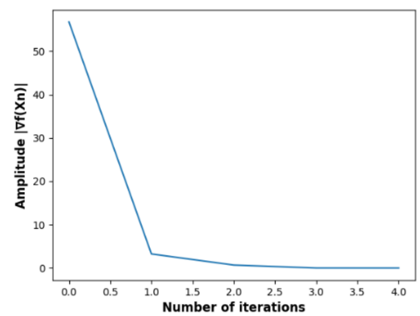

Figure 22 Number of iterations & Amplitude $|\nabla f(X_n)|$

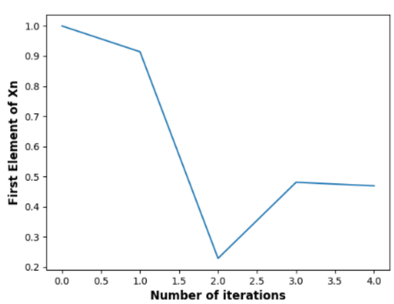

Figure 23 Number of iterations & First Element of Xn

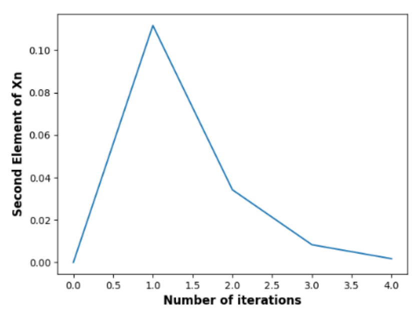

Figure 24 Number of iterations & Second Element of Xn

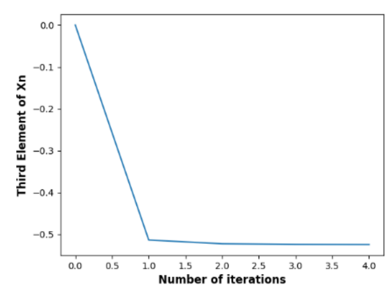

Figure 25 Number of iterations & Third Element of Xn

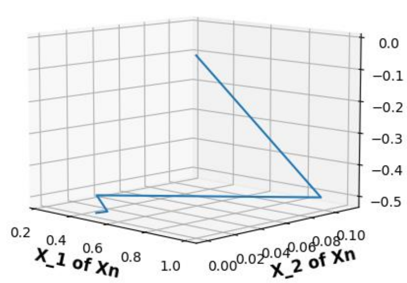

Figure 26 (3-D) First point of Newton-Raphson

2. Second Point:

```
previous = np.array([[1.00],
                     [5.00],
                     [2.00]])
```

minimum Point:

```
[[ 0.44063309]
 [-0.09901937]
 [-0.52533366]]
```

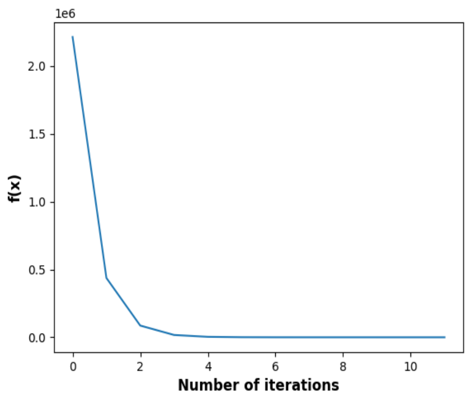

Figure 27 Number of iterations & f(x)

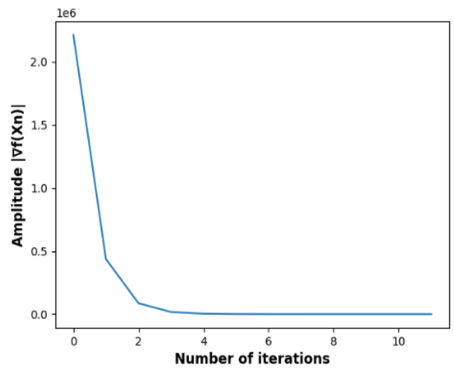

Figure 28 Number of iterations & Amplitude $|\nabla f(X_n)|$

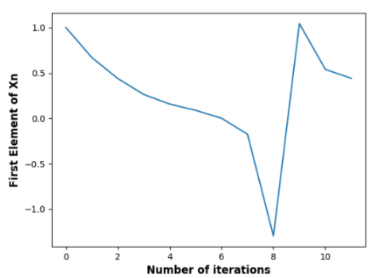

Figure 29 Number of iterations & First Element of Xn

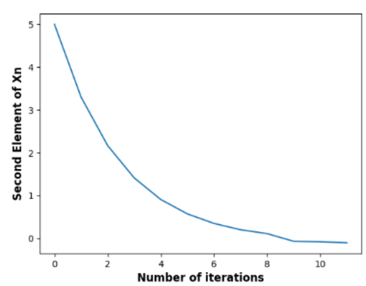

Figure 30 Number of iterations & Second Element of Xn

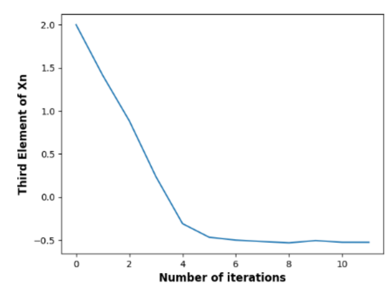

Figure 31 Number of iterations & Third Element of Xn

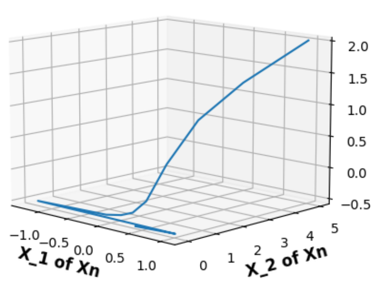

Figure 32 (3-D) Second point of Newton-Raphson

3. Third Point:

```
previous = np.array([[4.00],
                     [2.00],
                     [1.00]])
```

minimum Point:

```
[[ 0.42893864]
 [-0.00097304]
 [-0.52457821]]
```

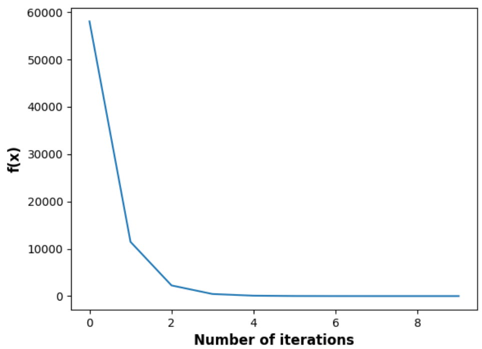

Figure 33 Number of iterations & f(x)

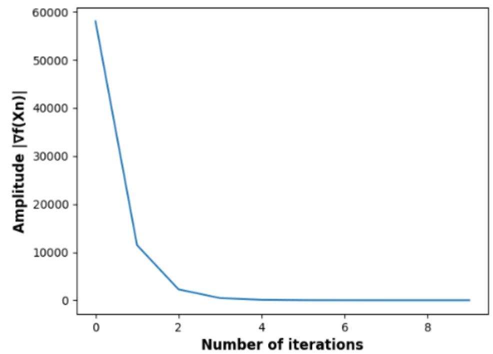

Figure 34 Number of iterations & Amplitude $|\nabla f(X_n)|$

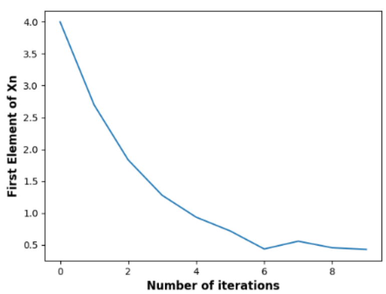

Figure 35 Number of iterations & First Element of Xn

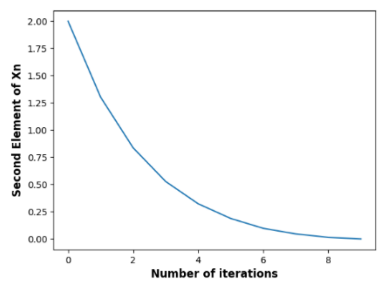

Figure 36 Number of iterations & Second Element of Xn

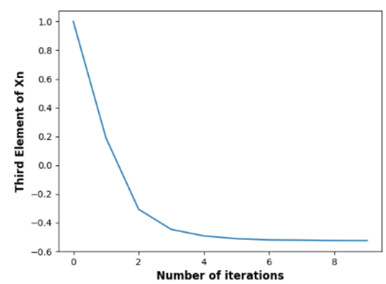

Figure 37 Number of iterations & Third Element of Xn

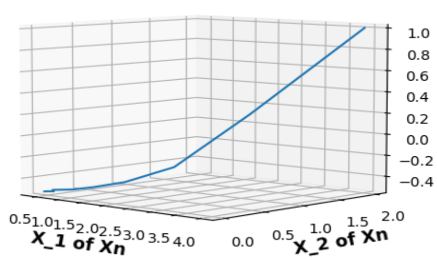

Figure 38 (3-D) Third point of Newton-Raphson

3. Steepest Gradient Descent

1. First Point:

```
previous = ([[1.00],
             [0.00],
             [0.00]])
```

minimum Point:

```
[[ 0.60650525]
 [-0.20776506]
 [-0.53195487]]
```

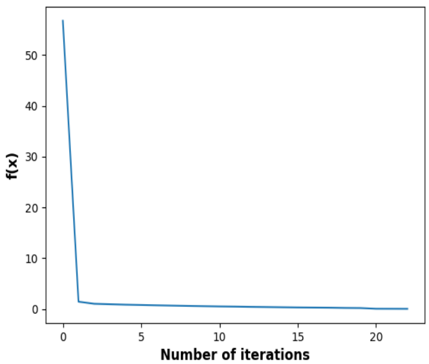

Figure 39 Number of iterations & f(x)

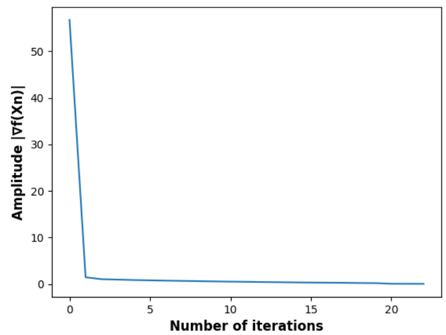

Figure 40 Number of iterations & Amplitude $|\nabla f(X_n)|$

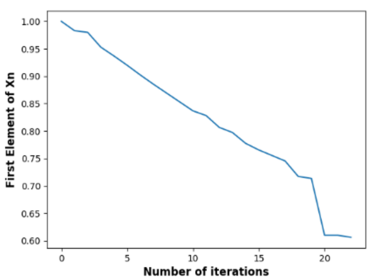

Figure 41 Number of iterations & First Element of Xn

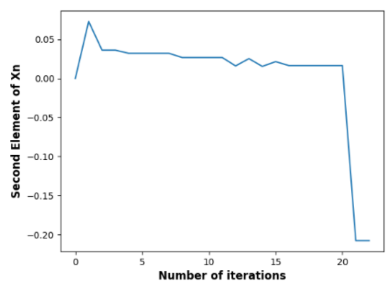

Figure 42 Number of iterations & Second Element of Xn

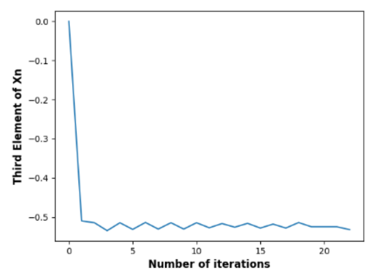

Figure 43 Number of iterations & Third Element of Xn

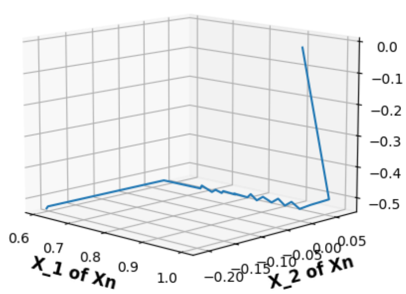

Figure 44 (3-D) First point of Steepest Gradient Descent

2. second Point:

```
previous = ([[5.00],
             [3.00],
             [4.00]])
```

minimum Point:

```
[[ 0.51275815]
 [-0.09556141]
 [-0.52931706]]
```

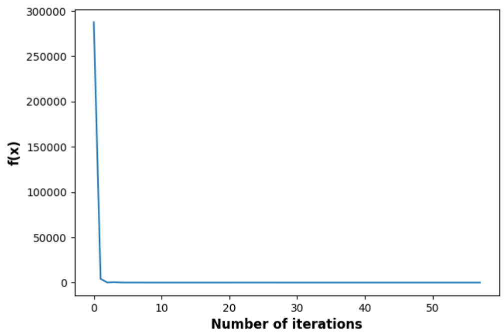

Figure 45 Number of iterations & f(x)

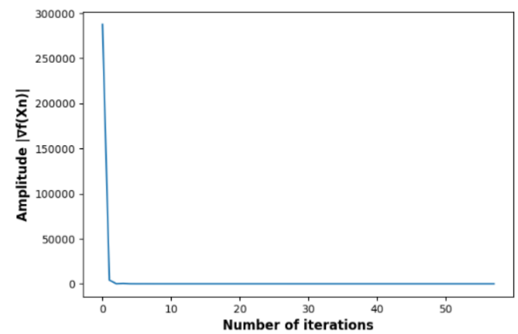

Figure 46 Number of iterations & Amplitude $|\nabla f(X_n)|$

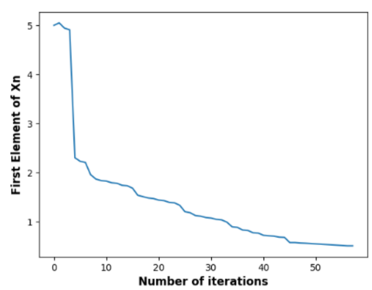

Figure 47 Number of iterations & First Element of Xn

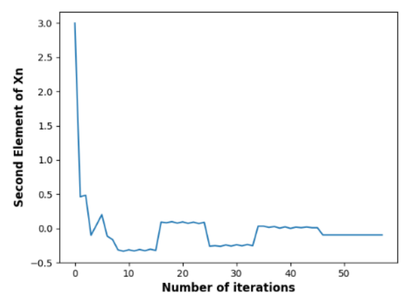

Figure 48 Number of iterations & Second Element of Xn

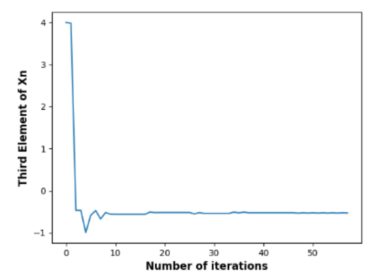

Figure 49 Number of iterations & Third Element of Xn

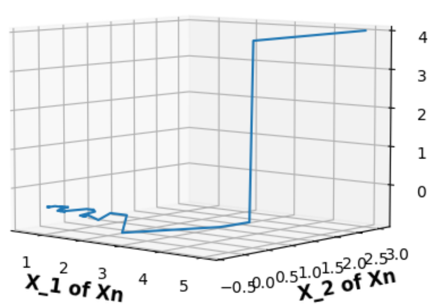

Figure 50 (3-D) Second point of Steepest Gradient Descent

3. Third Point:

```
previous = ([[1.00],
             [3.00],
             [2.00]])
```

minimum Point:

```
[[ 0.51259146]
 [-0.10175761]
 [-0.52933225]]
```

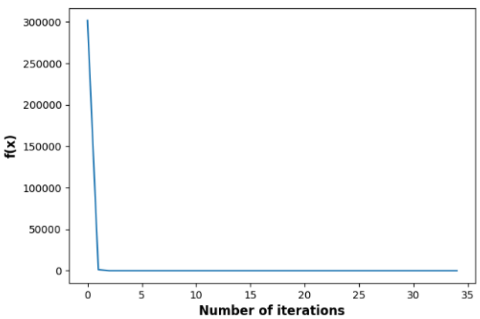

Figure 51 Number of iterations & f(x)

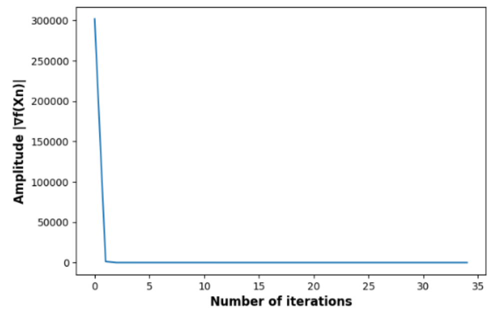

Figure 52 Number of iterations & Amplitude $|\nabla f(X_n)|$

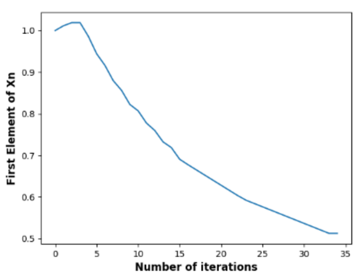

Figure 53 Number of iterations & First Element of Xn

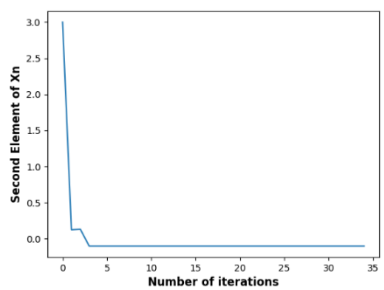

Figure 54 Number of iterations & Second Element of Xn

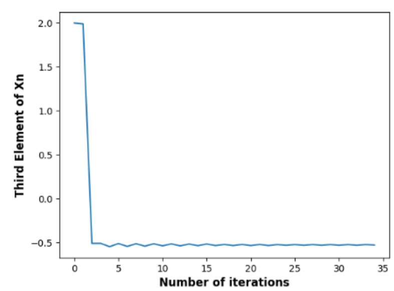

Figure 55 Number of iterations & Third Element of Xn

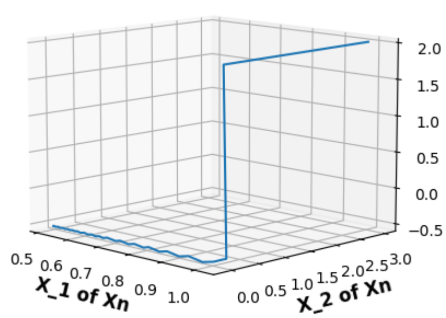

Figure 56 (3-D) Third point of Steepest Gradient Descent

An important note to run the program:

You must choose the algorithm according to the following

1- gradient descent
2- Newton-Raphson
3- steepest gradient descent

default in the Choose_Algorism Variable is (gradient descent)

```python

#
# You must choose the algorithm according to the following
# 1- gradient descent
# 2- Newton-Raphson
# 3- steepest gradient descent
# defult in the Choose_Algorism Variable is (gradient descent)
#
Choose_Algorism = "steepest gradient descent"
```

➢ Gradient Descent Method Code:

```python
#
# optimize the above function using the conventional gradient descent technique.
#
if Choose_Algorism == "gradient descent" :
    learning_rate = 0.0001
    Stop = 0.01
    Amp_df = 0.1
    iteration = 0
    #
    # Xn+1
    #
    current = ([[0.00],
                [0.00],
                [0.00]])
    #
    # Xn
    #
    previous = ([[5.00],
                 [1.00],
                 [-1.00]])
    while Amp_df > Stop :
        iteration += 1
        f_Value = float(f.subs([(x_1 , previous[0][0]) , (x_2 , previous[1][0]) , (x_3 , previous[2][0])]))
        dx_1 = int(dfdx_1.subs([(x_1 , previous[0][0]) , (x_2 , previous[1][0]) , (x_3 , previous[2][0])]))
        dx_2 = int(dfdx_2.subs([(x_1 , previous[0][0]) , (x_2 , previous[1][0]) , (x_3 , previous[2][0])]))
        dx_3 = int(dfdx_3.subs([(x_1 , previous[0][0]) , (x_2 , previous[1][0]) , (x_3 , previous[2][0])]))

        #
        #             ________________________________________
        # |∇f(Xn)| =|(df/dx1)^2 + (df/dx2)^2 + (df/dx3)^2
        #
        Amp_df = math.sqrt((((dx_1)**2)+((dx_2)**2)+((dx_3)**2)))
        #
        # g_df = [df/dx1]
        #        |df/dx2|
        #        [df/dx3]
        #               3*1
        #
        g_df = ([[dx_1],
                 [dx_2],
                 [dx_3]])
        #
        # Xn+1 = Xn - η∇f(Xn)
        #
        current = np.subtract(previous,(np.multiply(learning_rate,g_df)))
        arr_fplot = np.append(arr_fplot , f_Value)
        arr_Aplot = np.append(arr_Aplot , Amp_df)

```

```python
        arr_xplot = np.append(arr_xplot , previous[0][0])
        arr_yplot = np.append(arr_yplot , previous[1][0])
        arr_zplot = np.append(arr_zplot , previous[2][0])

        arr_Xplot = np.append(arr_Xplot , previous[0][0])
        arr_Yplot = np.append(arr_Yplot , previous[1][0])
        arr_Zplot = np.append(arr_Zplot , previous[2][0])

        #
        # Xn = Xn+1
        #
        previous = np.array(current)
        print (Amp_df)

```

➢ Newton-Raphson Method Code:

```python
elif Choose_Algorism == "Newton-Raphson" :
    iteration = 0
    Stop = 0.01
    f_dash_Amp = 0.02

    current = np.array([[0.00],
                        [0.00],
                        [0.00]])
    previous = np.array([[4.00],
                         [2.00],
                         [1.00]])
    while f_dash_Amp > Stop :
        iteration += 1
        f_Value = float(f.subs([(x_1 , previous[0][0]) , (x_2 , previous[1][0]) , (x_3 , previous[2][0])]))
        dx_1 = int(dfdx_1.subs([(x_1 , previous[0][0]) , (x_2 , previous[1][0]) , (x_3 , previous[2][0])]))
        dx_2 = int(dfdx_2.subs([(x_1 , previous[0][0]) , (x_2 , previous[1][0]) , (x_3 , previous[2][0])]))
        dx_3 = int(dfdx_3.subs([(x_1 , previous[0][0]) , (x_2 , previous[1][0]) , (x_3 , previous[2][0])]))

        #
        # f_dash = [df/dx1]
        #          |df/dx2|
        #          [df/dx3]
        #                 3*1
        f_dash = ([[dx_1],
                   [dx_2],
                   [dx_3]])

        #          ______________________________________
        # |∇f(Xn)| =|(df/dx1)^2 + (df/dx2)^2 + (df/dx3)^2
        #
        f_dash_Amp = math.sqrt((((dx_1)**2)+((dx_2)**2)+((dx_3)**2)))

        dx1_2  = int(df2dx2_1.subs([(x_1 , previous[0][0]) , (x_2 , previous[1][0]) , (x_3 , previous[2][0])]))
        dx1_x2 = int(df2dx_1dx_2.subs([(x_1 , previous[0][0]) , (x_2 , previous[1][0]) , (x_3 , previous[2][0])]))
        dx1_x3 = int(df2dx_1dx_3.subs([(x_1 , previous[0][0]) , (x_2 , previous[1][0]) , (x_3 , previous[2][0])]))

        dx2_2  = int(df2dx2_2.subs([(x_1 , previous[0][0]) , (x_2 , previous[1][0]) , (x_3 , previous[2][0])]))
        dx2_x1 = int(df2dx_2dx_1.subs([(x_1 , previous[0][0]) , (x_2 , previous[1][0]) , (x_3 , previous[2][0])]))
        dx2_x3 = int(df2dx_2dx_3.subs([(x_1 , previous[0][0]) , (x_2 , previous[1][0]) , (x_3 , previous[2][0])]))
```

```python
        dx3_2  = int(df2dx2_3.subs([(x_1 , previous[0][0]) , (x_2 , previous[1][0]) , (x_3 , previous[2][0])]))
        dx3_x1 = int(df2dx_3dx_1.subs([(x_1 , previous[0][0]) , (x_2 , previous[1][0]) , (x_3 , previous[2][0])]))
        dx3_x2 = int(df2dx_3dx_2.subs([(x_1 , previous[0][0]) , (x_2 , previous[1][0]) , (x_3 , previous[2][0])]))

        #
        # H = [d2f/dx1^2   d2f/dx1dx2  d2f/dx1dx3]
        #     |d2f/dx2dx1  d2f/dx2^2   d2f/dx2dx3|
        #     [d2f/dx3dx1  d2f/dx3dx2  d2f/dx3^2 ]
        #                                         3*3
        f_double_dash = np.matrix(
                                [
                                    [dx1_2,dx1_x2,dx1_x3],
                                    [dx2_x1,dx2_2,dx2_x3],
                                    [dx3_x1,dx3_x2,dx3_2]
                                ]
        )

        #
        # Invese of H
        #
        f_double_dash_inv = np.linalg.inv(f_double_dash)

        #
        # Xn+1 = Xn - (1/H) * f_dash
        #
        current = np.subtract(previous,(np.dot(f_double_dash_inv,f_dash)))
        arr_fplot = np.append(arr_fplot , f_Value)
        arr_Aplot = np.append(arr_Aplot , f_dash_Amp)

        arr_xplot = np.append(arr_xplot , previous[0][0])
        arr_yplot = np.append(arr_yplot , previous[1][0])
        arr_zplot = np.append(arr_zplot , previous[2][0])

        arr_Xplot = np.append(arr_Xplot , previous[0][0])
        arr_Yplot = np.append(arr_Yplot , previous[1][0])
        arr_Zplot = np.append(arr_Zplot , previous[2][0])
        previous = np.array(current)
        print (f_dash_Amp)
```

- **Steepest Gradient Descent Method Code:**

```python
elif Choose_Algorism == "steepest gradient descent" :
    iteration = 0
    Stop = 0.01
    Stop_η = 0.01
    Amp_df = 0.1
    current_η = 0.00
    previous_η = 0.00
    η_dash = 0.02
    η_double_dash = 0.02
    current = ([[0.00],
                [0.00],
                [0.00]])
    previous = ([[1.00],
                 [3.00],
                 [2.00]])
    while Amp_df > Stop:
        iteration += 1
        f_Value = float(f.subs([(x_1 , previous[0][0]) , (x_2 , previous[1][0]) , (x_3 , previous[2][0])]))
        dx_1 = int(dfdx_1.subs([(x_1 , previous[0][0]) , (x_2 , previous[1][0]) , (x_3 , previous[2][0])]))
        dx_2 = int(dfdx_2.subs([(x_1 , previous[0][0]) , (x_2 , previous[1][0]) , (x_3 , previous[2][0])]))
        dx_3 = int(dfdx_3.subs([(x_1 , previous[0][0]) , (x_2 , previous[1][0]) , (x_3 , previous[2][0])]))

        #
        # g_df = [df/dx1]
        #        |df/dx2|
        #        [df/dx3]
        #              3*1
        #
        g_df = ([[dx_1],
                 [dx_2],
                 [dx_3]])

        #       ______________________________________
        # |∇f(Xn)| =|(df/dx1)^2 + (df/dx2)^2 + (df/dx3)^2
        #
        Amp_df = math.sqrt((((dx_1)**2)+((dx_2)**2)+((dx_3)**2)))
```

```
        #
        # Xn+1 = Xn - η∇f(Xn)
        #
        current = np.subtract(previous,(np.multiply(η,g_df)))

        g1_η = g_1.subs([(x_1 , current[0][0]) , (x_2 , current[1][0]) , (x_3 , current[2][0])])
        g2_η = g_2.subs([(x_1 , current[0][0]) , (x_2 , current[1][0]) , (x_3 , current[2][0])])
        g3_η = g_3.subs([(x_1 , current[0][0]) , (x_2 , current[1][0]) , (x_3 , current[2][0])])

        #
        # F(η)=1/2[g1(η)]^2+1/2[g2(η))]^2+1/2[g3(η)]^2
        #
        f_η = (1/2)*(g1_η**2) + (1/2)*(g2_η**2) + (1/2)*(g3_η**2)

        df_dη = smp.diff(f_η, η)
        df2dη2 = smp.diff(f_η, η , 2)

        if iteration >1:
            η_dash = int(df_dη.subs(η , previous_η))

        while abs(η_dash) > Stop_η :
            η_dash = int(df_dη.subs(η , previous_η))
            η_double_dash = int(df2dη2.subs(η , previous_η))

            #
            # ηn+1 = ηn _ η'/η``
            #
            current_η = previous_η - (η_dash/η_double_dash)
            previous_η = current_η

        #
        # Xn+1 = Xn - η∇f(xn)
        #
        current = np.subtract(previous,(np.multiply(current_η,g_df)))
        arr_fplot = np.append(arr_fplot , f_Value)
        arr_Aplot = np.append(arr_Aplot , Amp_df)
        arr_xplot = np.append(arr_xplot , previous[0][0])
        arr_yplot = np.append(arr_yplot , previous[1][0])
        arr_zplot = np.append(arr_zplot , previous[2][0])

        arr_Xplot = np.append(arr_Xplot , previous[0][0])
        arr_Yplot = np.append(arr_Yplot , previous[1][0])
        arr_Zplot = np.append(arr_Zplot , previous[2][0])
        previous = current
        print (Amp_df)
```

➢ You can get the code through this link:

https://drive.google.com/file/d/1efHkIlwpjxh0cqkc4iUZNXOTOB5dN7eV/view?usp=share_link

➢ references:

- Doctor Lectures
- Section Notes