HAMZA BIN ZAHID Mat No: 3405458

Code Algorithem

Solution :-

Given is non Linear System

$$f_1(x,y) = x^2 + y^2 - 9 \quad \text{—— ①}$$
$$f_2(x,y) = x + 2y - 6 \quad \text{—— ②}$$

① Calculating Jacobi Matrix using partial derivatives.

$$\frac{\partial}{\partial x} f_1(x,y) = 2x$$

$$\frac{\partial}{\partial y} f_1(x,y) = 2y$$

$$\frac{\partial}{\partial x} f_2(x,y) = 1$$

$$\frac{\partial}{\partial y} f_2(x,y) = 2$$

$$J = \begin{bmatrix} 2x & 2y \\ 1 & 2 \end{bmatrix}$$

② C++ code will be done in the following flow & .cpp file will be submitted.

③ Initial guess is

$$x_0 = 0.1, \quad y_0 = 0.1$$

(HAMZA BIN ZAHID / Mat No: 340595 8) ① back

④ Input from user not required in C++ code.

⑤ Run the program for "11" Iterations.

Newton Method Iterations formula:

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} x_n \\ y_n \end{bmatrix} - J^{-1} * F(x_n)$$ ← Main Eq

Iteration # 01 i.e (initial guess $(x_0 = 0.1, y_0 = 0.1)$)

$$F(x_0, y_0) = \begin{bmatrix} x_0^2 + y_0^2 - 9 \\ x_0 + 2y_0 - 6 \end{bmatrix} = \begin{bmatrix} (0.1)^2 + (0.1)^2 - 9 \\ (0.1) + 2(0.1) - 6 \end{bmatrix} = \begin{bmatrix} -8.98 \\ -5.7 \end{bmatrix}$$

Calculating J^{-1}

$$J^{-1} = \frac{\begin{bmatrix} 2 & -2y_0 \\ -1 & 2x_0 \end{bmatrix}}{4x - 2y}$$

Now guess

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} - \left\{ \frac{\begin{bmatrix} 2 & -2y_0 \\ -1 & 2x_0 \end{bmatrix}}{4x_0 - 2y_0} \right\} * \begin{bmatrix} -8.98 \\ -5.7 \end{bmatrix}$$

(HAMZA BIN ZAHID 3405958)

(2) Point

New guess will be

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix} - \left\{ \frac{\begin{bmatrix} 2 & -2(0.1) \\ -1 & 2(0.1) \end{bmatrix}}{4(0.1) - 2(0.1)} \right\} * \begin{bmatrix} -8.98 \\ -5.7 \end{bmatrix}$$

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix} - \begin{bmatrix} 2/0.2 & \frac{-0.2}{0.2} \\ -1/0.2 & \frac{0.2}{0.2} \end{bmatrix} * \begin{bmatrix} -8.98 \\ -5.7 \end{bmatrix}$$

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix} - \begin{bmatrix} 10 & -1 \\ -5 & 1 \end{bmatrix} * \begin{bmatrix} -8.98 \\ -5.7 \end{bmatrix}$$

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix} - \begin{bmatrix} -84.1 \\ 39.2 \end{bmatrix}$$

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} 84.2 \\ -39.1 \end{bmatrix}$$

Notes

New guess for x_1 & y_1 willbe '84.2 & -39.1'

following the values of x_1, y_1 & updating x & y values each time we will get results repeat this for 11 iterations.

From all 11 iterations we will get the following results (Code in .ipynb file)

(HAMZA BIN ZAHID 340595 8)

(2) back

Iteration	x	y	Norm
1	84.2	-39.1	92.7871
2	42.7087	-18.3543	46.3887
3	21.9717	-7.98584	23.1847
4	11.6205	-2.81025	11.573
5	6.47935	-0.239673	5.74799
6	3.97605	1.01197	2.79877
7	2.84739	1.57631	1.26189
8	2.86075	1.76963	0.193275
9	2.40146	1.79927	0.0662836
10	2.4	1.8	0.00163537
11	2.4	1.8	9.96706e-07

decrease in Norm value can be seen w.r.t increase in Iterations

Norm Calculation: for (1st Iteration)

$$Norm = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}$$

$$Norm = \sqrt{(84.2 - 0.1)^2 + (-39.1 - 0.1)^2}$$

$$Norm = 92.7871$$

Note: for each iteration we will calculate Norm using updated values of x & y & then subtracting it from previous value. Norms results are mentioned in above table (C++ Code in .cpp file).

"The Norm decreases to a very small value indicating convergence to exact solution"

(8) (HAMZA BIN ZAHID 8405958)

(3) Front

Now calculating convergence order using the given formula.

$$P_n = \frac{\log(Norm_n / Norm_{n+1})}{\log(Norm_{n+1} / Norm_{n+2})}$$

Using Norm value from the table as we get.

[92.7871, 46.3887, 23.1847, 11.573, 5.74799, 2.79877
1.26189, 0.432275, 0.0662836, 0.00163537, 9.96706×10^{-7}]

Now for

n = 1

$$P_1 = \frac{\log(92.7871/46.3887)}{\log(46.3887/23.1847)}$$

$P_1 = 0.999960$

n = 2

$$P_2 = \frac{\log(46.3887/23.1847)}{\log(23.1847/11.573)}$$

$P_2 = 0.998141$

n = 3

$$P_3 = \frac{\log(23.1847/11.573)}{\log(11.573/5.74799)}$$

$P_3 = 0.992823$

n = 4

$$P_4 = \frac{\log(11.573/5.74799)}{\log(5.74799/2.79877)}$$

$P_4 = 0.971513$

n = 5

$$P_5 = \frac{\log(5.74799/2.79877)}{\log(2.79877/1.26189)}$$

$P_5 = 0.903502$

n = 6

$$P_6 = \frac{\log(2.79877/1.26189)}{\log(1.26189/0.432275)}$$

$P_6 = 0.743551$

n = 7

$$P_7 = \frac{\log(1.26189/0.432275)}{\log(0.432275/0.0662836)}$$

$P_7 = 0.571325$

n = 8

$$P_8 = \frac{\log(0.432275/0.0662836)}{\log(0.0662836/0.00163537)}$$

$P_8 = 0.506505$

(HAMZA BIN ZAHID 340595 8)

(3) back

n = 9

$$P_9 = \frac{\log(0.0662836/0.00163537)}{\log(0.00163537/9.96706\times10^{57})}$$

$P_9 = 0.500082$

Now Taking Av. of convagence order =>

$$\frac{0.99960 + 0.998141 + 0.992823 + 0.971513 + 0.903502 + 0.743551 + 0.571325 + 0.506505 + 0.500082}{9}$$

Av. Convagence Order => 0.798560222 2

Note To round of this number to nearest 0.5 precision we will consider figure after point i.e, 7 if it is greater than 0.25 then we will round up to nearest 0.5 if less than 0.25 we will round down to nearest 0.5

In this case number after point is 7 80; greater than 0.25 round up to nearest 0.5 number will became 1.

Hamza Bin Zahid
Matriculation Number: 3405958

Iteration vs Norm Graphical Representation using (Matlab):

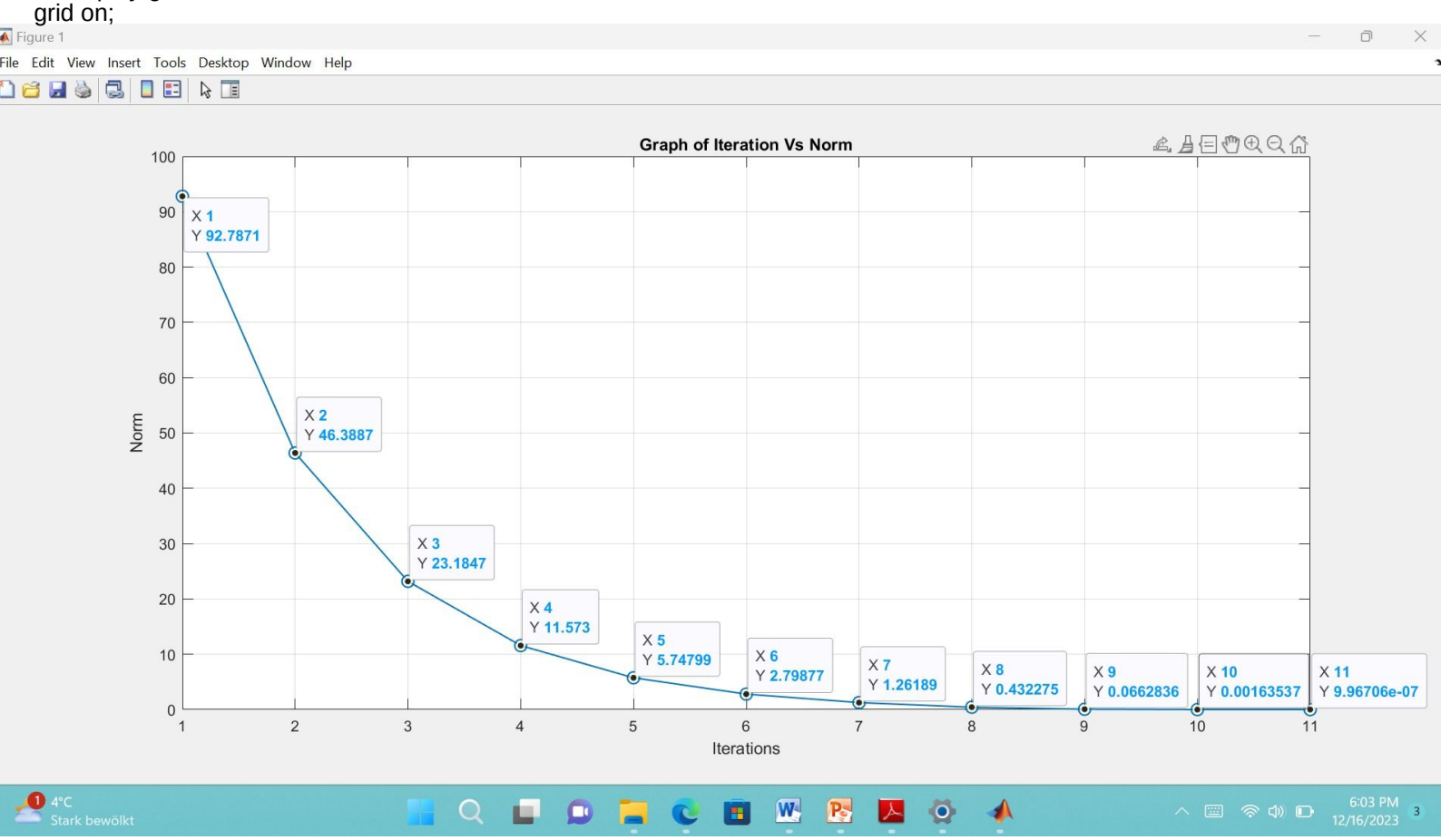
//Norm values are taken from C++ code results whereby matlab used for plotting against each iteration as instructed in question

Matlab Code:

```
% Given points
Iterations = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11];
Norm = [92.7871, 46.3887, 23.1847, 11.573, 5.74799, 2.79877, 1.26189, 0.432275, 0.0662836, 0.00163537, 9.96706*10^-7];

% Plot the given points
plot(Iterations, Norm, 'o-', 'LineWidth', 1, 'MarkerSize', 8);

% Add labels and title
xlabel('Iterations');
ylabel('Norm');
title('Graph of Iteration Vs Norm');
% Display grid
grid on;
```



Additional Comment:

"The Norm values is decreasing to a very small value indicating convergence to exact solution" the above graph depicts the same

Hamza Bin Zahid
Matriculation Number: 3405958

C++ Code Output:

```
Initial Values for x is 0.1
Initial Values for y is 0.1
Iteration x y Norm
   1      84.2      -39.1      92.7871
   2      42.7087      -18.3543      46.3887
   3      21.9717      -7.98584      23.1847
   4      11.6205      -2.81025      11.573
   5      6.47935      -0.239673      5.74799
   6      3.97605      1.01197      2.79877
   7      2.84739      1.57631      1.26189
   8      2.46075      1.76963      0.432275
   9      2.40146      1.79927      0.0662836
   10      2.4      1.8      0.00163537
   11      2.4      1.8      9.96706e-07
Average Convergence Order: 0.79866
Rounded Convergence Order (0.5 precision): 1
```

Hamza Bin Zahid
Matriculation No 3405958
C++ Code with detailed comments:
//Hamza Bin Zahid
//Matriculation Number: 3405958

```
#include <iostream>
#include <cmath>
using namespace std;

int main()

{
    // Initial guess as given in problem
    double x = 0.1;
    double y = 0.1;
    cout << "Initial Values for x is " <<   x <<  endl;
    cout << "Initial Values for y is " <<   y <<  endl;

    // Defining maxIterations as given is 11 in question
    const int maxIterations = 11;
    cout << "Iteration x y Norm" << endl;

    // Loop for iterations till maximum 11
    for (int i = 0; i < maxIterations; ++i)
    {
        // Equations and their derivatives
        double f1 = x * x + y * y - 9; //defining funcion "F1(x)""
        double f2 = x + 2 * y - 6; //defining funcion "F2(x)""
        double df1dx = 2 * x; //defining partial derivative of funcion "F1(x)"" wrt "x"
        double df1dy = 2 * y; //defining partial derivative of funcion "F1(x)"" wrt "y"
        double df2dx = 1; //defining partial derivative of funcion "F2(x)" wrt "x"
        double df2dy = 2; //defining partial derivative of funcion "F2(x)" wrt "y"

        //This part of the loop is where the Newton method is applied.
        //It calculates the Jacobian matrix determinant, checks for division by zero, and then updates
        //the variables (x and y) based on the Newton method's formula.
        //The loop iterates until convergence is achieved or the maximum number of iterations is reached
        // Check for division by zero error "safety check" to check for infinity situation not created in code

        double det = df1dx * df2dy - df1dy * df2dx;// defining and finding Jacobian matrix determinant

        // Check for division by zero error "safety check" to check for infinity situation not created in code
        if (fabs(det) < 1e-10)
        {
            cerr << "Error: Division by zero." << endl;
            return 1;
        }

        // Calculate the change in variables
        double deltaX = (f1 * df2dy - f2 * df1dy) / det;
        double deltaY = (f2 * df1dx - f1 * df2dx) / det;

        // Update variables
        x -= deltaX;
        y -= deltaY;

        // Calculate the Euclidean norm
        double norm = sqrt(deltaX * deltaX + deltaY * deltaY);

        // Print current solution and norm
        cout << "   " << i + 1 << "     " << x << "     " << y << "     " << norm << endl;

        // Check for convergence (convergence vs Iteration graph & code made in Matlab)
        if (norm < 1e-8)
        {
            cout << "Converged to a solution." << endl;
            break;
        }
    }
    //Generated norm value arrays to futhur calculate the covergance order
    double Norm[] = {92.7871, 46.3887, 23.1847, 11.573, 5.74799, 2.79877, 1.26189, 0.432275, 0.0662836, 0.00163537, 9.96706e-7};

    double convergenceOrders[10];
    //Convergance order loop just like in notes I have calculated P1,p2,p3 till P9 convergence convergenceOrders
    for (int i = 0; i < 9; ++i)
    {
        convergenceOrders[i] = log(Norm[i] / Norm[i + 1]) / log(Norm[i + 1] / Norm[i + 2]);
    }

    // Calculate the average convergence order
    double averageConvergenceOrder = 0.0;
    for (int i = 0; i < 9; ++i)
    {
        averageConvergenceOrder += convergenceOrders[i];
    }
    averageConvergenceOrder /= 9;

    // Round to the nearest 0.5 precision
    double roundedConvergenceOrder = round(averageConvergenceOrder * 2) / 2;

    cout << "Average Convergence Order: " << averageConvergenceOrder << endl;
    cout << "Rounded Convergence Order (0.5 precision): " << roundedConvergenceOrder << endl;

    return 0;

}
```