NMAssignment1

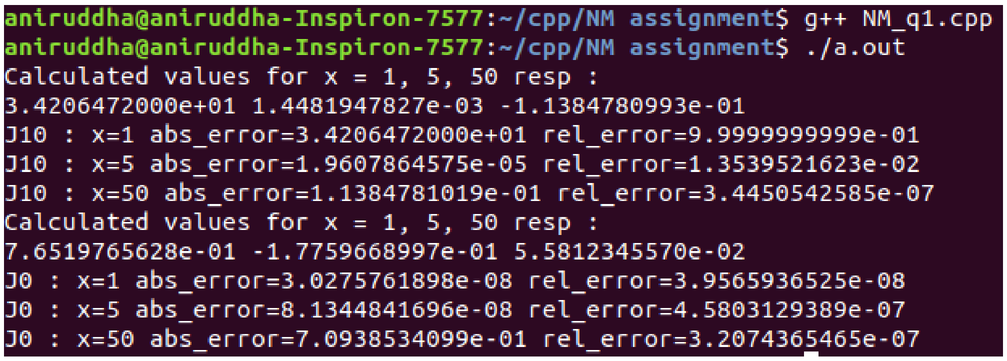
By Aniruddha Bala
Roll No: 06-02-01-10-51-18-1-15655

September 16, 2018

1 Numerical Methods Assignment-1

1.1 Question 1

Please refer the file NM_q1.cpp for the code. On running the code we get the following output.



```
aniruddha@aniruddha-Inspiron-7577:~/cpp/NM assignment$ g++ NM_q1.cpp
aniruddha@aniruddha-Inspiron-7577:~/cpp/NM assignment$ ./a.out
Calculated values for x = 1, 5, 50 resp :
3.4206472000e+01 1.4481947827e-03 -1.1384780993e-01
J10 : x=1 abs_error=3.4206472000e+01 rel_error=9.9999999999e-01
J10 : x=5 abs_error=1.9607864575e-05 rel_error=1.3539521623e-02
J10 : x=50 abs_error=1.1384781019e-01 rel_error=3.4450542585e-07
Calculated values for x = 1, 5, 50 resp :
7.6519765628e-01 -1.7759668997e-01 5.5812345570e-02
J0 : x=1 abs_error=3.0275761898e-08 rel_error=3.9565936525e-08
J0 : x=5 abs_error=8.1344841696e-08 rel_error=4.5803129389e-07
J0 : x=50 abs_error=7.0938534099e-01 rel_error=3.2074365465e-07
```

Q1 Output

1.1.1 Observation

From the output we can see that when recursion is computed in forward direction for $x = 1$, the value of J_{10} diverges and the relative error is 1, the values of J_{10} for $x = 5$ and 50 converges and is hence stable. But when the recursion is computed in backward direction the value of J_0 converges for all values of x. The error in last value computed by recurrence relation in case of backward approach is lesser than that in forward approach. The reason for that has been stated below.

$$J_n(x) = \frac{2(n-1)}{x} J_{n-1}(x) - J_{n-2}(x) \quad \text{---} (i)$$ Forward case

Let J^* be the approximation & J be the true value

$$J_n^*(x) = \frac{2(n-1)}{x} J_{n-1}^*(x) - J_{n-2}^*(x) \quad \text{---} (ii)$$

$$\xi_n = |J - J_n^*|$$

(i) - (ii)

$$\xi_n = \frac{2(n-1)}{x} \xi_{n-1} - \xi_{n-2}$$

$$\xi_2 = \frac{2.1}{x} \xi_1 - \xi_0 .$$

$$\xi_3 = \frac{2.2}{x} \xi_2 - \xi_1$$

$$= \frac{2.2}{x} \left[\frac{2.1}{x} \xi_1 - \xi_0 \right] - \xi_1$$

$$= \frac{2^2 \, 2!}{x^2} \xi_1 - \frac{2.2}{x} \xi_0 - \xi_1$$

From the above pattern it is clear that the major term contributing to the growth of the error at nth iteration would be 1st term

$$\xi_n \propto \frac{2^{n-1}(n-1)!}{x^{n-1}} \xi_1$$

Now if $x = 1$ Then $\xi_n \propto 2^{n-1}(n-1)! \, \xi_1$

$\Rightarrow$ as $n \uparrow$ error grows rapidly also. error due to truncation is greater considering the error in the forward case as we are approximating to first 6 digits & J_0 is of order 10^{-1}.

$$|7.6519768656 e{-01} - 7.65198 e{-01}|$$

$$= 3.1344 \times 10^{-7}$$

And since the denominator $x = 1$, there is no term to balance off the error as n grows.

For $n = 5$ & 50 it converges as ~~as~~ n grows $2^{n-1}(n-1)!$ grows but at the same time denominator grows as 5^{n-1} or 50^{n-1} & hence the error is limited.

<u>For backward case</u>

$$J_n = \frac{2(n+1)}{x} J_{n+1} - J_{n+2}$$ substituting $n = n+1$

$$\therefore \quad J_8 = \frac{2 \cdot 9}{x} J_9 - J_{10}$$

$$J_7 = \frac{2 \cdot 8}{x} J_8 - J_9$$

$$= \frac{2 \cdot 8}{x} \left[\frac{2 \cdot 9}{x} J_9 - J_{10} \right.$$

$$\xi_n = \frac{2(n+1)}{x} \xi_{n+1} - \xi_{n+2}$$

$$\xi_8 = \frac{2 \cdot 9}{x} \xi_9 - \xi_{10}$$

$$\xi_7 = \frac{2 \cdot 8}{x} \xi_8 - \xi_9$$

$$= \frac{2 \cdot 8}{x} \left[\frac{2 \cdot 9}{x} \xi_9 - \xi_{10} \right] - \xi_9$$

$$= \frac{2^2 \cdot 9 \cdot 8}{x^2} \xi_9 - \frac{2 \cdot 8}{x} \xi_{10} - \xi_9$$

Again from the above pattern we can observe that

$$\xi_n \propto \frac{2^{9-n} \cdot 9 \cdot 8 \ldots (n+1)}{x^{9-n}} \xi_9$$

as . $n \to 0$.

$$\xi_0 \propto \frac{2^9 \; 9!}{x^9} \xi_9$$

But since we are starting with a small error here ξ_0 converges.

$$\xi_9 = |\, 5.2492501799 \times 10^{-9} - 5.24925 \times 10^{-9} |$$

for $x = 1$

$$= 1.799 \times 10^{-16}.$$

where as $2^9 \times 9! = 185794560$.

order of 10^8.

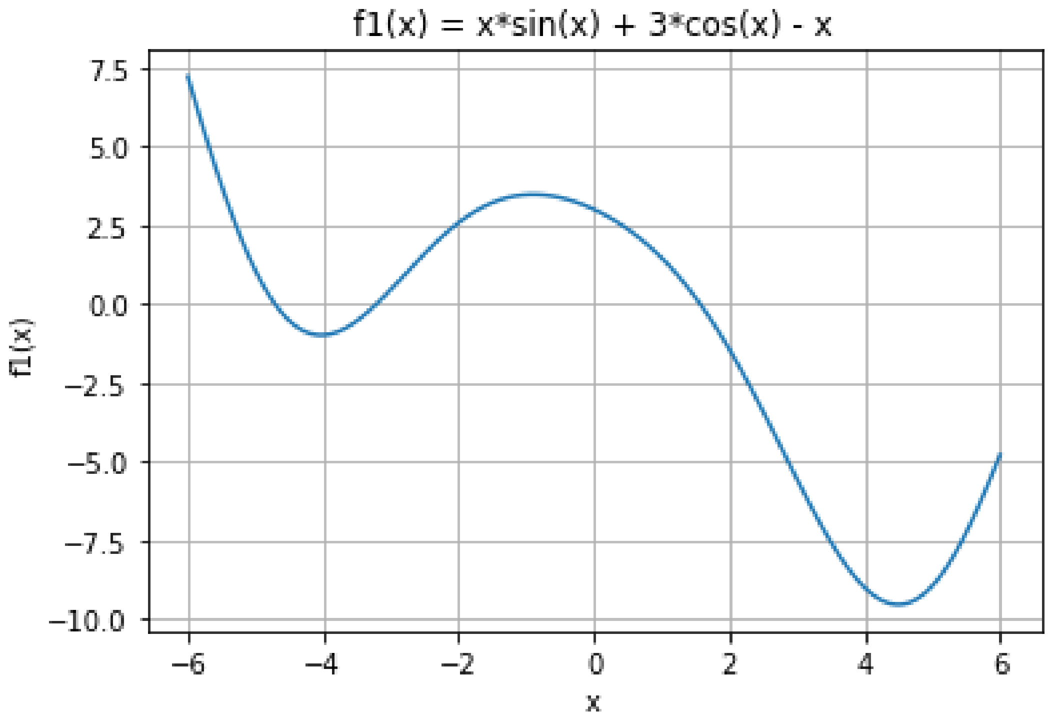
1.2 Question 2

Below are the plots for the two given functions. It is clear from the plots that $x * sin(x) + 3 * cos(x) - x$ has 3 roots between $(-6, 6)$ and also $sin(x) - 0.1 * x$ has 3 positive roots. Therefore we should expected to get these roots using the given root finding methods.

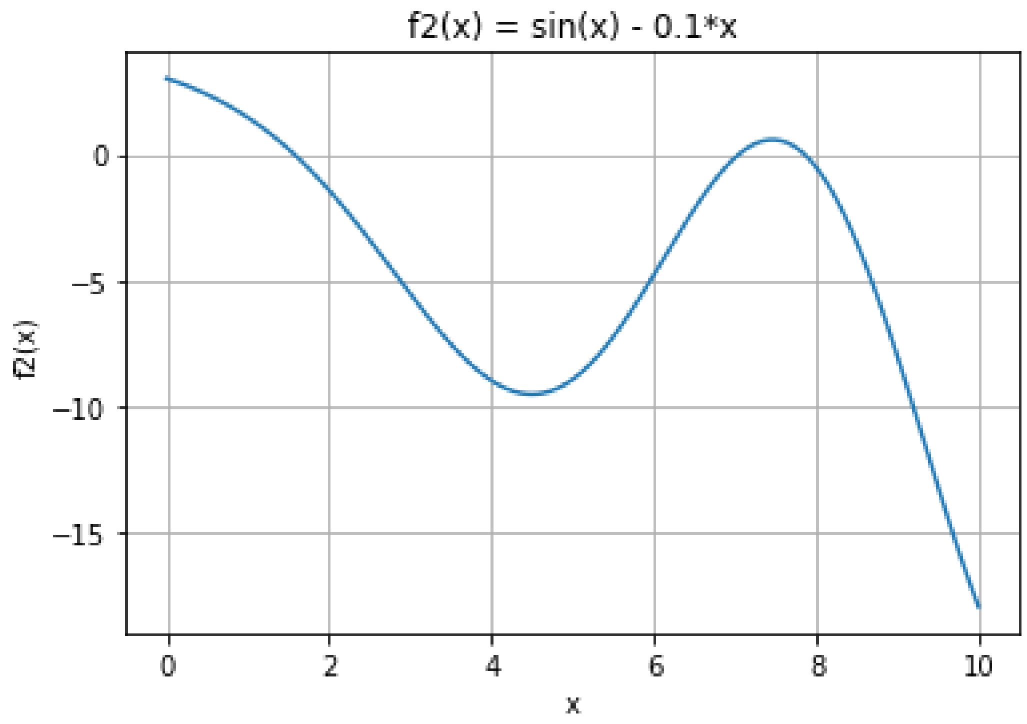
```
In [4]: import numpy as np

        def f1(x):
            return x*np.sin(x) + 3*np.cos(x) -x

        def f2(x):
            return np.sin(x) - 0.1*x
```

```
In [7]: import matplotlib.pyplot as plt
        %matplotlib inline

        x = np.linspace(-6, 6, 1000)
        plt.plot(x , f1(x))
        plt.xlabel('x')
        plt.ylabel('f1(x)')
        plt.grid()
        plt.title('f1(x) = x*sin(x) + 3*cos(x) - x')
```

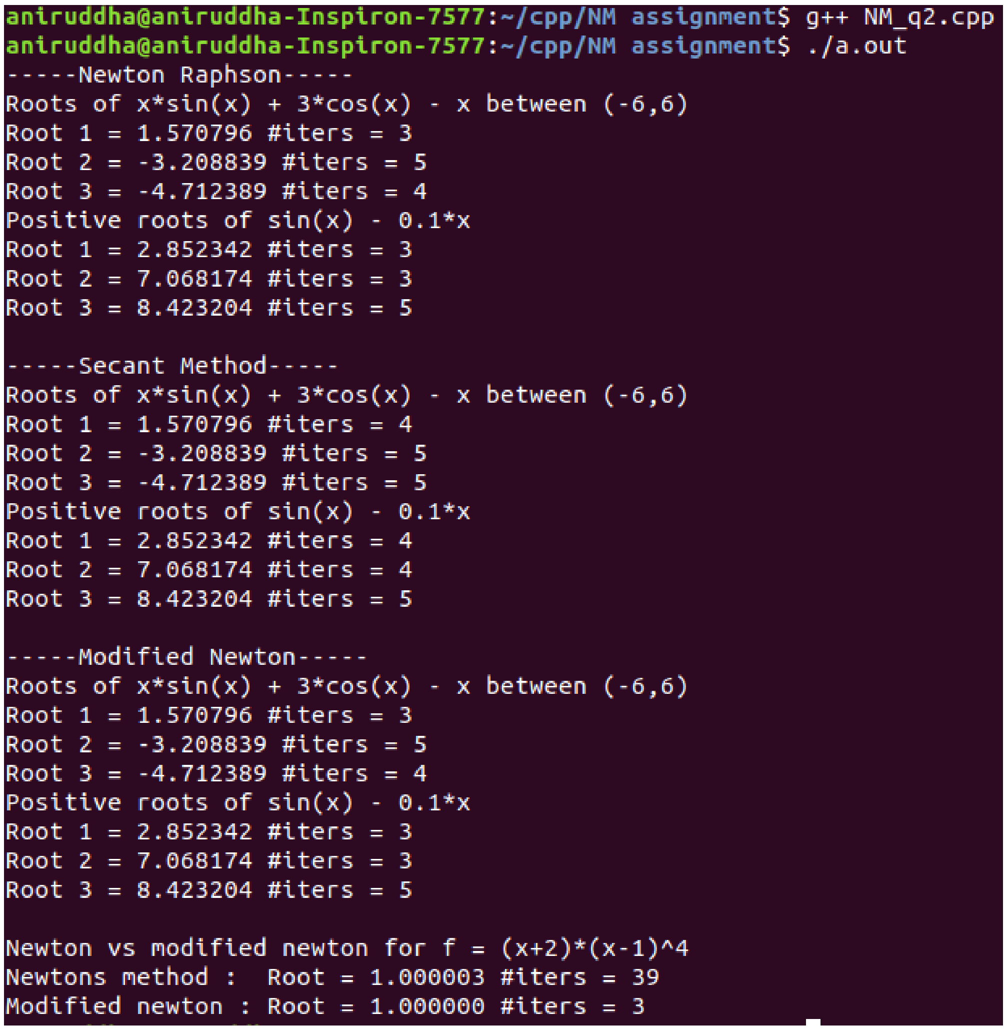
```
Out[7]: Text(0.5,1,'f1(x) = x*sin(x) + 3*cos(x) - x')
```



```
In [8]: x = np.linspace(0, 10, 5000)
        plt.plot(x , f1(x))
        plt.xlabel('x')
        plt.ylabel('f2(x)')
        plt.grid()
        plt.title('f2(x) = sin(x) - 0.1*x')

Out[8]: Text(0.5,1,'f2(x) = sin(x) - 0.1*x')
```



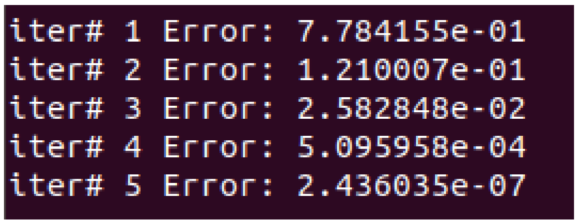
Please refer file NM_q2.cpp for code. On running the code we get the following output.



```
aniruddha@aniruddha-Inspiron-7577:~/cpp/NM assignment$ g++ NM_q2.cpp
aniruddha@aniruddha-Inspiron-7577:~/cpp/NM assignment$ ./a.out
-----Newton Raphson-----
Roots of x*sin(x) + 3*cos(x) - x between (-6,6)
Root 1 = 1.570796 #iters = 3
Root 2 = -3.208839 #iters = 5
Root 3 = -4.712389 #iters = 4
Positive roots of sin(x) - 0.1*x
Root 1 = 2.852342 #iters = 3
Root 2 = 7.068174 #iters = 3
Root 3 = 8.423204 #iters = 5

-----Secant Method-----
Roots of x*sin(x) + 3*cos(x) - x between (-6,6)
Root 1 = 1.570796 #iters = 4
Root 2 = -3.208839 #iters = 5
Root 3 = -4.712389 #iters = 5
Positive roots of sin(x) - 0.1*x
Root 1 = 2.852342 #iters = 4
Root 2 = 7.068174 #iters = 4
Root 3 = 8.423204 #iters = 5

-----Modified Newton-----
Roots of x*sin(x) + 3*cos(x) - x between (-6,6)
Root 1 = 1.570796 #iters = 3
Root 2 = -3.208839 #iters = 5
Root 3 = -4.712389 #iters = 4
Positive roots of sin(x) - 0.1*x
Root 1 = 2.852342 #iters = 3
Root 2 = 7.068174 #iters = 3
Root 3 = 8.423204 #iters = 5

Newton vs modified newton for f = (x+2)*(x-1)^4
Newtons method :  Root = 1.000003 #iters = 39
Modified newton : Root = 1.000000 #iters = 3
```

Q2 Output

To see the rate of convergence I have printed the errors obtained for Newtons method separately for $f_1(x)$, for root -3.208839



```
iter# 1 Error: 7.784155e-01
iter# 2 Error: 1.210007e-01
iter# 3 Error: 2.582848e-02
iter# 4 Error: 5.095958e-04
iter# 5 Error: 2.436035e-07
```

Errors for f1(x) while finding root -3.208839

Below analysis proves the quadratic convergence of Newton's method for the above obtained error values.

```
In [15]: eps = np.array([7.784155e-01,1.210007e-01,2.582848e-02,5.095958e-04,2.436035e-07])
         epsn = eps[1:]
```

```
        epsn_minus_one = eps[:-1]
        epsn/epsn_minus_one**2

Out[15]: array([0.19969396, 1.76409953, 0.76388528, 0.93806265])
```

The above result prints out the values of $\epsilon_n/\epsilon_{n-1}^2$ for which we can see that it gives values $K < 1$ with one exception. Therefore Newton's method shows quadratic convergence.

Newton's method converges relatively faster as compared to secant method in this case, but it requires evaluation of $f'(x)$ at every step. However here the derivative of the function is already passed to newtons method and hence it is equivalent to evaluation of function value $f'(x)$ at $x = x_{n-1}$ at any step.

However difficulty arises in cases where function has multiple roots as Newton's method drops to linear convergence in such cases. In such cases Modified Newton's method can be useful. For this assignment the given functions do not have any root of multiplicity, therefore we donot see any significant performance improvement in case of Modified Newton's method.

Just to compare the performance of Newton's and Modified Newton's method I have taken a function with multiple roots $f_3(x) = (x+2)*(x-1)^4$. This function has a root of multiplicity 4 at $x = 1$. In this case we can see that Modified Newton's method outperforms Newton's method taking only 3 iterations to converge while Newton's method takes 39 iterations with the same initial guess.

1.3 Question 3

Q3) Derivation . Cubit Newton's Method :

$$g(x) = x - \phi(x) f(x) - \psi(x) f^2(x)$$

For 3rd order convergence $g'(P) = 0$ & $g''(P) = 0$.

$$g'(x) = 1 - [\phi'(x) f(x) + \phi(x) f'(x)] - [\psi'(x) f^2(x) + 2\psi(x) f(x) f'(x)]$$

$$g'(P) = 1 - [\cancel{\phi(P) f(P)}^{0} + \phi(P) f'(P)] - [\cancel{\psi'(P) f^2(P)}^{0} + \cancel{2\psi(P) f(P) f'(P)}^{0}]$$

as $f(P) = 0$

$$\Rightarrow 1 - \phi(P) f'(P) \cancel{f(P)} = 0.$$

$$\Rightarrow \phi(P) = 1/f'(P)$$

$$g''(P) = 0.$$

$$g''(x) = -[\phi''(x) f(x) + \phi'(x) f'(x) + \phi'(x) f'(x) + \phi(x) f''(x)]$$
$$- [\psi''(x) f^2(x) + 2\psi'(x) f(x) f'(x) + 2\psi'(x) f(x) f'(x) + 2\psi(x)(f'(x))^2 \cancel{+ 2\psi(x) f(x) f''} + 2\psi(x) f(x) f''(x)]$$

$$g''(P) = -2\phi'(P) f'(P) + \phi(P) f''(P) + 2\psi(P)(f'(P))^2 = 0.$$

$$\psi(P) = \frac{1}{2(f'(P))^2} [2\phi'(P) f'(P) - \phi(P) f''(P)]$$

$$\psi(P) = \frac{1}{2(f'(P))^2}\left[\frac{2f''(P)}{(f'(P))^2}\cancel{f'(P)} - \frac{f''(P)}{f'(P)}\right]$$

$$= \frac{1}{2(f'(P))^2} \cdot \frac{f''(P)}{f'(P)} = \frac{f''(P)}{2(f'(P))^3}$$

$$\Rightarrow \quad g(x) = x - \frac{f(x)}{f'(x)} - \frac{f^2(x)}{2(f'(P))^3} \cdot f''(P)$$

$$P = g(P) \quad - \text{I}$$

$$P_{n+1} = g(P_n) \quad - \text{II}$$

Expand with Taylor series about P.

$$\text{(II)} - \text{(I)} \qquad P_{n+1} - P = g(P_n) - g(P)$$

$$\xi_{n+1} = (P_n - P)\,g'(P) + \frac{(P_n-P)^2}{2!}\,g''(P) + \frac{(P_n-P)^3}{3!}\,g'''(P) + \cdots$$

For ~~2~~ 3rd order convergence

$$g'(P) = 0 \;\; \& \;\; g''(P) = 0$$

$$\xi_{n+1} = \xi_n^3 \times \frac{g'''(P)}{3!}$$

asymptotic order of convergence $\alpha = 3$.

$$g'''(x) = -\left[\phi'''(x) f(x) + 3\phi''(x) f'(x) + 3\phi'(x) f''(x) + \phi(x) f'''(x)\right]$$
$$-\left[\psi'''(x) f^2(x) + 6\psi''(x) f(x) f'(x) + 6\psi'(x) f'^2(x) + 6\psi'(x) f(x) f''(x) + 6\psi(x) f'(x) f''(x) + 2\psi(x) f(x) f'''\right]$$

$$g'''(P) = -\left[3\phi''(P) f'(P) + 3\phi'(P) f''(P) + \phi(P) f'''(P)\right]$$
$$-\left[6\psi'(P) f'^2(P) + 6\psi(P) f'(P) f''(P)\right]$$

$$\phi = 1/f'$$

Now $\phi' = \dfrac{-f''}{(f')^2} \qquad \psi = \dfrac{f''}{2(f')^3}$

$$\psi' = \frac{f'''}{2(f')^3} - \frac{3}{2}\frac{(f'')^2}{(f')^4}$$

$$\phi'' = \frac{-f'''}{(f')^2} + 2\frac{(f'')^2}{(f')^3}$$

$$\therefore g'''(p) = -\left[-3\frac{f'''\cdot f'}{(f')^2} + \frac{6(f'')^2 f'}{(f')^3} + 3\times\frac{-f''}{(f')^2}\cdot f'' + \frac{f'''}{f'}\right]$$

$$- \left[\frac{6f'''\cancel{(f')^2}}{2\cancel{(f')^3}(f')} - 9\frac{(f'')^2\cdot\cancel{(f')^2}}{(f')^{\cancel{4}2}} + \frac{6(f'')^2 f'}{2(f')^3}\right]$$

$$= \cancel{\frac{3f'''}{(f')}} - \frac{6(f'')^2}{(f')^2} + \cancel{3\frac{(f'')^2}{(f')^2}} - \frac{f'''}{f'}$$

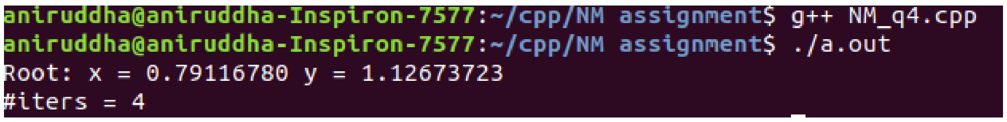
$$- \cancel{\frac{3f'''}{f'}} + 9\frac{(f'')^2}{(f')^2} \cancel{+0} - \cancel{3\frac{(f'')^2}{(f')^2}}$$

$$= \frac{3(f'')^2}{(f')^2} - \frac{f'''}{f'}$$

$$\therefore \lambda = \frac{g'''(p)}{6} = \frac{1}{2}\left(\frac{f''(p)}{f'(p)}\right)^2 - \frac{1}{6}\frac{f'''(p)}{f'(p)}$$

1.4 Question 4

Please refer file NM_q4.cpp for the code. On running the code we get the following output.



```
aniruddha@aniruddha-Inspiron-7577:~/cpp/NM assignment$ g++ NM_q4.cpp
aniruddha@aniruddha-Inspiron-7577:~/cpp/NM assignment$ ./a.out
Root: x = 0.79116780 y = 1.12673723
#iters = 4
```

Q4 Output

Therefore using Newton's method we could find the solution to given system of multi variable non linear equations in the vicinity of point $(1, 1)$. The solution is $x = 0.79116780, y = 1.12673723$.