proj2_b

October 5, 2016

```
In [1]: %pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
In [2]: import matplotlib.pyplot as plt
```

1 Newton's Method

For Newton's Method portion of this project I created a C++ program called newton.cpp to perform Newton's method of root finding for a funciton $f(x)$. This class contained the method # double newton(Fcn& f, Fcn& df, double x, int maxit, double tol, bool show_iterates); In this function definitionm f and df are functions that represent the function for which the root should be found and the derivative of that function. X represents the first inital guess and maxit limits newtons method to only $[0, maxit]$ iterations. Finally tol is the user defined tolerance for the accuracy of the root and show_iterates prints statistics as each iteration finishes.

```
In [3]: %cat newton.cpp
```

```
/* Project - Project 2_Part b
 * Prof - Dr Xu
 * Name - Jake Rowland
 * Date - 10/6/16
 * Purpuse - Use Newton's method of root finding
*/

#include <iostream>
#include <cmath>
#include "fcn.hpp"

#ifndef NEWTON
#define NEWTON

class Newton{
public:
        //Static as to not require class
        static double newton(Fcn& f, Fcn& df, double xStart, int maxit, double tol, bool show_it
```

```
    {
        //Not to override xStart
        double x = xStart;
        //Find f(x)
        double fx = f(x);

        //For total iterations wanted
        for(int i = 0; i < maxit; i++)
        {
            //Find derivative at f(x)
            double fp = df(x);

            //If derivative to small approcing horizontal asymptote or double root.
            if(std::abs(fp) < pow(10, -5))
            {
                if(show_iterates)
                    std::cout << "Too Small Derivative\n";
                break;
            }

            //Value to reduce x by
            double d = (double)fx/(double)fp;

            //Reduce x and find new f(x) value
            x = x - d;
            fx = f(x);

            //If distance is withing tolerance root found
            if(std::abs(d) < tol)
            {
                if(show_iterates)
                {
                    std::cout << "Convergence\n";
                }
                break;
            }

            //Print each value if wanted
            if(show_iterates)
            {
                std::cout << "Iter=" << i << "  ::  x=" << x << "  ::  d=" << st
            }
        }

        //Return root
        return x;
    }
};
```

```
#endif
```

The second part of Newton's Method had me creating the C++ program test_newton.cpp. In this program I use my previously implemented program newton.cpp to find the roots for the function $f(x) = x^2(x-3)(x+2)$. Given the initial guesses $x_0 = [-3, 1, 2]$ and the tolerances $e = [10^{-1}, 10^{-5}, 10^{-9}]$ I ran each initial guess with each tolerance and, with show_iterates active, concluded the result into a file named proj2_b.txt

```
In [4]: proj2 = open("proj2_b.txt").read()
        print (proj2)
```

```
Using -3 to an error of 0.1
Iter=0  ::  x=-2.45455  ::  d=0.545455  ::  fx=14.9375
Iter=1  ::  x=-2.14186  ::  d=0.312681  ::  fx=3.3464
Iter=2  ::  x=-2.01957  ::  d=0.122291  ::  fx=0.400736
Convergence


Using -3 to an error of 1e-05
Iter=0  ::  x=-2.45455  ::  d=0.545455  ::  fx=14.9375
Iter=1  ::  x=-2.14186  ::  d=0.312681  ::  fx=3.3464
Iter=2  ::  x=-2.01957  ::  d=0.122291  ::  fx=0.400736
Iter=3  ::  x=-2.00045  ::  d=0.0191283  ::  fx=0.00891221
Iter=4  ::  x=-2  ::  d=0.000445134  ::  fx=4.75706e-06
Convergence


Using -3 to an error of 1e-09
Iter=0  ::  x=-2.45455  ::  d=0.545455  ::  fx=14.9375
Iter=1  ::  x=-2.14186  ::  d=0.312681  ::  fx=3.3464
Iter=2  ::  x=-2.01957  ::  d=0.122291  ::  fx=0.400736
Iter=3  ::  x=-2.00045  ::  d=0.0191283  ::  fx=0.00891221
Iter=4  ::  x=-2  ::  d=0.000445134  ::  fx=4.75706e-06
Iter=5  ::  x=-2  ::  d=2.37853e-07  ::  fx=1.35891e-12
Convergence


Using 1 to an error of 0.1
Iter=0  ::  x=0.454545  ::  d=0.545455  ::  fx=1.2909
Iter=1  ::  x=0.228022  ::  d=0.226524  ::  fx=0.321116
Iter=2  ::  x=0.115144  ::  d=0.112877  ::  fx=0.0809002
Convergence
```

```
Using 1 to an error of 1e-05
Iter=0  ::  x=0.454545  ::  d=0.545455  ::  fx=1.2909
Iter=1  ::  x=0.228022  ::  d=0.226524  ::  fx=0.321116
Iter=2  ::  x=0.115144  ::  d=0.112877  ::  fx=0.0809002
Iter=3  ::  x=0.0579873  ::  d=0.0571571  ::  fx=0.0203588
Iter=4  ::  x=0.0291159  ::  d=0.0288714  ::  fx=0.00511036
Iter=5  ::  x=0.014591  ::  d=0.0145249  ::  fx=0.00128044
Iter=6  ::  x=0.00730406  ::  d=0.0072869  ::  fx=0.000320483
Iter=7  ::  x=0.00365422  ::  d=0.00364985  ::  fx=8.01685e-05
Iter=8  ::  x=0.00182766  ::  d=0.00182656  ::  fx=2.00482e-05
Iter=9  ::  x=0.000913969  ::  d=0.000913692  ::  fx=5.0128e-06
Iter=10  ::  x=0.000457019  ::  d=0.00045695  ::  fx=1.2533e-06
Iter=11  ::  x=0.000228518  ::  d=0.000228501  ::  fx=3.13336e-07
Iter=12  ::  x=0.000114261  ::  d=0.000114257  ::  fx=7.83354e-08
Iter=13  ::  x=5.71312e-05  ::  d=5.71301e-05  ::  fx=1.9584e-08
Iter=14  ::  x=2.85657e-05  ::  d=2.85655e-05  ::  fx=4.89603e-09
Iter=15  ::  x=1.42829e-05  ::  d=1.42828e-05  ::  fx=1.22401e-09
Convergence


Using 1 to an error of 1e-09
Iter=0  ::  x=0.454545  ::  d=0.545455  ::  fx=1.2909
Iter=1  ::  x=0.228022  ::  d=0.226524  ::  fx=0.321116
Iter=2  ::  x=0.115144  ::  d=0.112877  ::  fx=0.0809002
Iter=3  ::  x=0.0579873  ::  d=0.0571571  ::  fx=0.0203588
Iter=4  ::  x=0.0291159  ::  d=0.0288714  ::  fx=0.00511036
Iter=5  ::  x=0.014591  ::  d=0.0145249  ::  fx=0.00128044
Iter=6  ::  x=0.00730406  ::  d=0.0072869  ::  fx=0.000320483
Iter=7  ::  x=0.00365422  ::  d=0.00364985  ::  fx=8.01685e-05
Iter=8  ::  x=0.00182766  ::  d=0.00182656  ::  fx=2.00482e-05
Iter=9  ::  x=0.000913969  ::  d=0.000913692  ::  fx=5.0128e-06
Iter=10  ::  x=0.000457019  ::  d=0.00045695  ::  fx=1.2533e-06
Iter=11  ::  x=0.000228518  ::  d=0.000228501  ::  fx=3.13336e-07
Iter=12  ::  x=0.000114261  ::  d=0.000114257  ::  fx=7.83354e-08
Iter=13  ::  x=5.71312e-05  ::  d=5.71301e-05  ::  fx=1.9584e-08
Iter=14  ::  x=2.85657e-05  ::  d=2.85655e-05  ::  fx=4.89603e-09
Iter=15  ::  x=1.42829e-05  ::  d=1.42828e-05  ::  fx=1.22401e-09
Iter=16  ::  x=7.14146e-06  ::  d=7.14144e-06  ::  fx=3.06003e-10
Iter=17  ::  x=3.57073e-06  ::  d=3.57073e-06  ::  fx=7.65008e-11
Iter=18  ::  x=1.78537e-06  ::  d=1.78537e-06  ::  fx=1.91252e-11
Iter=19  ::  x=8.92684e-07  ::  d=8.92683e-07  ::  fx=4.7813e-12
Iter=20  ::  x=4.46342e-07  ::  d=4.46342e-07  ::  fx=1.19533e-12
Too Small Derivative


Using 2 to an error of 0.1
Iter=0  ::  x=-2  ::  d=4  ::  fx=0
Convergence
```

```
Using 2 to an error of 1e-05
Iter=0  ::  x=-2  ::  d=4  ::  fx=0
Convergence


Using 2 to an error of 1e-09
Iter=0  ::  x=-2  ::  d=4  ::  fx=0
Convergence
```

A few things of interest arise after looking at the results. Algebra tells us that the roots for $f(x) = x^2(x-3)(x+2)$ are $[-2, 0^2, +3]$ with zero being a double root. When the initial guess of $x_0 = 3$ is applied to newtons method, the method works marvelously. Very little iterations are required to find the root -2 to tolerances $[10^{-1}, 10^{-5}, 10^{-9}]$. Things don't go as smoothly for the next two guesses $x_0 = 1, 2$. $x_0 = 1$ degrades to linearly convergent because the root at $x = 0$ has a derivative of $f'(0) = 0$. This causes the change factor in newtons method to approach zero slowly because $lim_{x->0}\ f'(x) = 0$. This however cannot be avoided in newton's method. And, with the final tolerance of 10^{-9} results in a stop case because the derivative becomes to small signaling a horizontal asymptote or double root. The final result is the most interesting however. The first two results found or closely approximated the root nearest to them $x_0 = -3$ found the root $f(-2) = 0$, $x_0 = 1$ found the root $f(0) = 0$ or closely approximated it. However $x_0 = 2$ did not find the root $f(3) = 0$, but it did find a root. This is considered an ill-chosen starting point because the method worked as intended $d = f(2)/fd(2)$ and $ x = x - d, *sowhatstheproblem?Solvingforthevaluesshowsthat*d = -16/-4 = 4$ then the new x is $x = 2 - 4 = -2$. As stated above $f(-2) = 0$. Because of the shape of this graph and because of the ill-chosen point $x_0 = 2$ newton's method believes to have found a root and in fact is has, $f(-2) = 0$ but it found the wrong root, not $f(3) = 0$

```
In [5]: %cat test_newton.cpp

/* Project - Project 2_Part b
 * Prof - Dr Xu
 * Name - Jake Rowland
 * Date - 10/6/16
 * Purpuse - Test the newton class
*/

#include <iostream>
#include <fstream>
#include <cmath>
#include "newton.cpp"
#include "fcn.hpp"
```

```
//Creating the fuction f(x) = x^2 * (x - 3) * (x + 2)
class fnorm: public Fcn
{
        double operator()(double x)
        {
                return pow(x, 2) * (x - 3) * (x + 2);
        }
};

//Create the derivative of the function f(x)
class fderv: public Fcn
{
        double operator()(double x)
        {
                return x * ((4 * pow(x, 2)) - (3 * x) - 12);
        }
};

int main ()
{
        //Creating the starting points and the tolerances
        double x0[3] = {-3, 1, 2};
        double e[3] = {pow(10, -1), pow(10, -5), pow (10, -9)};

        //Create both of the functions
        fnorm f1;
        fderv f2;

        //Loop throught the different starting points
        for(int i = 0; i < 3; i++)
        {
                //Loop throught the different tolerances
                for(int j = 0; j < 3; j++)
                {
                        //States what starting point and tolerance used and calls Newton::newton
                        std::cout << "Using " << x0[i] << " to an error of " << e[j] << "\n";
                        Newton::newton(f1, f2, x0[i], 50, e[j], true);
                        std::cout << "\n\n";
                }
        }
}
```