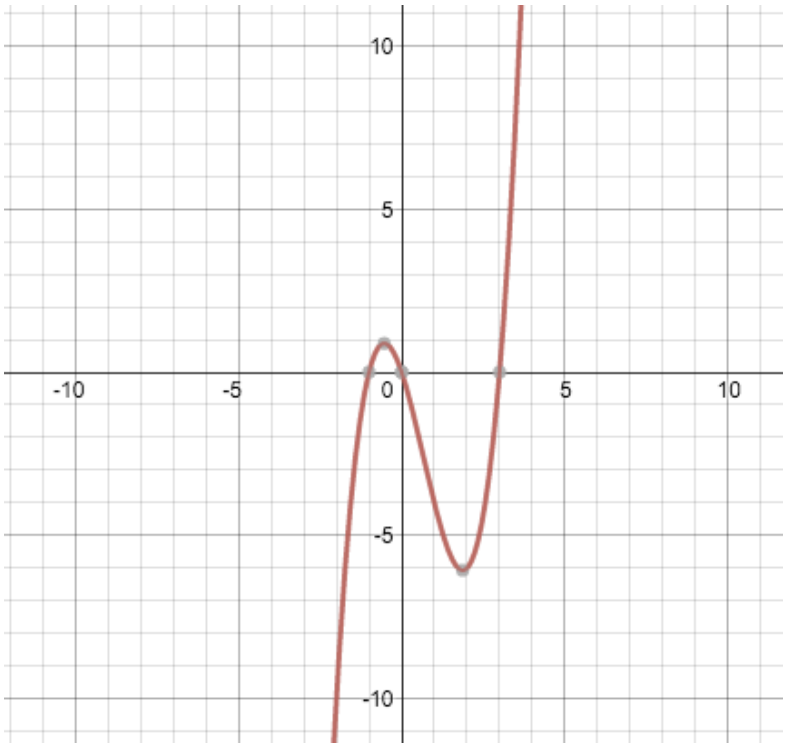
Project 2: Nonlinear Solvers

Erik Gabrielsen

October 7, 2015

Part I: Newton's Method

Newton's method is a procedure that is used to find roots of systems of equations provided the equation is differentiable. In Part I of this project, I implemented Newton's method to approximate the roots of the residual function f(x) = x(x-3)(x+1) given a specified tolerance of Epsilon = {10^{-1}, 10^{-5} , 10^{-9} }. To approximate these roots, I will be giving my newton method initial guesses of x_0 = {-2, -1, 2}. I will also be doing a maximum number of iterations of 15 to terminate the computation if the loop carries on forever. The graph for f(x) = x(x-3)(x+1) is as follows:



Based Upon this graph and just looking at the equation itself, I should expect that Newton's method should correctly Identify the roots of this function at x = {-1, 0, 3}. To compute these roots, I created a file (test_newton.cpp) that would iterate through newton's method using different tolerances of {10^{-1}, 10^{-5} , 10^{-9} } and different initial guesses of {-2, -1, 2} for a total of 9 iterations.

test_newton.cpp

```cpp
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
#include <iomanip>
#include <math.h>
#include "fcn.hpp"
#include "newton.cpp"

using namespace std;

// -- f(x)
class fcn : public Fcn {
public:
  double operator()(double x) {
    // f(x) = x(x-3)(x+1)
    return x * (x-3) * (x+1);
  }
};

// -- f'(x)
class fdx : public Fcn {
public:
  double operator()(double x) {
    // f'(x) = 3x^2 - 4x - 3
    return ((3*x*x) - 4*x - 3);
  }
};

int main(int argc, char* argv[]) {
    // -- set the initial guesses
    double guesses[3] = { -2, 1, 2 };

    // -- set the tolerances
    double tols[3] = { 10e-1, 10e-5, 10e-9 };
    double ans[9];
    int maxit = 15;

    fcn f; //  f(x)
    fdx df; //  f'(x)
    int index = 0;

    // -- loop through each guess
    for(unsigned int i = 0; i < 3; i++) {
      // -- inner loop: loops through each tolerance and displays the root as approximated by newtons method
        for(unsigned int j = 0; j < 3; j++) {
            double guess = guesses[i];
            double tol = tols[j];

            // -- newtons method, storing in an array for display purposes
            ans[index] = newton(f, df, guess, maxit, tol, true);

            // -- display the final approximation for tol[j] and guess[i]
            cout << "The Approximate Root is --->  " << ans[index] << "\n" << endl;
            index++;
        }
    }

    // -- ouput the approximations for each iteration of the newtons method from above in order so that
    // I can compare root guesses.
    for (int i =0; i < 9; i++) {
      cout << "ANS-> " << ans[i] << endl;
    }
    return 0;
}
```

newton.cpp

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

using namespace std;

double newton(Fcn& f, Fcn& df, double x, int maxit, double tol, bool show_iterates) {
  if (show_iterates) {
    cout << endl;
    // -- show the initial guess
    cout << "guess " << x << " tol: " << tol << endl;
  }

  int iteration = 0;
  double solutionUpdate = 10;

  // -- go to max iterations or until solution update is less than the tolerance
  while(iteration <= maxit && solutionUpdate > tol) {
    iteration++;
    double previous = x;

    // -- x = x(n+1) by (x - f(x)/f'(x))
    x = x - (f(x)/df(x));

    // -- solution update = |x(n+1) - x|
    solutionUpdate = fabs(x - previous);

    if (show_iterates) {
      //-- output the iteration, guess, solution update and residual for each iteration
      cout << "iter " << iteration << "; guess " << x << "; solution update " <<
          solutionUpdate << "; residual: " << fabs(f(x)) << endl;
    }
  }
  return x;
}

```

RESULTS:

```
guess -2 tol: 1
iter 1; guess -1.41176; solution update 0.588235; residual: 2.56462
The Approximate Root is --->  -1.41176


guess -2 tol: 0.0001
iter 1; guess -1.41176; solution update 0.588235; residual: 2.56462
iter 2; guess -1.11446; solution update 0.297303; residual: 0.524854
iter 3; guess -1.01322; solution update 0.101246; residual: 0.0537364
iter 4; guess -1.00021; solution update 0.0130028; residual: 0.000849868
iter 5; guess -1; solution update 0.000212354; residual: 2.25491e-07
```

```
iter 6; guess -1; solution update 5.63727e-08; residual: 1.59872e-14
The Approximate Root is --->  -1


guess -2 tol: 1e-08
iter 1; guess -1.41176; solution update 0.588235; residual: 2.56462
iter 2; guess -1.11446; solution update 0.297303; residual: 0.524854
iter 3; guess -1.01322; solution update 0.101246; residual: 0.0537364
iter 4; guess -1.00021; solution update 0.0130028; residual: 0.000849868
iter 5; guess -1; solution update 0.000212354; residual: 2.25491e-07
iter 6; guess -1; solution update 5.63727e-08; residual: 1.59872e-14
iter 7; guess -1; solution update 3.9968e-15; residual: 0
The Approximate Root is --->  -1


guess 1 tol: 1
iter 1; guess 0; solution update 1; residual: 0
The Approximate Root is --->  0


guess 1 tol: 0.0001
iter 1; guess 0; solution update 1; residual: 0
iter 2; guess 0; solution update 0; residual: 0
The Approximate Root is --->  0


guess 1 tol: 1e-08
iter 1; guess 0; solution update 1; residual: 0
iter 2; guess 0; solution update 0; residual: 0
The Approximate Root is --->  0


guess 2 tol: 1
iter 1; guess 8; solution update 6; residual: 360
iter 2; guess 5.70701; solution update 2.29299; residual: 103.616
iter 3; guess 4.26553; solution update 1.44148; residual: 28.4241
iter 4; guess 3.44217; solution update 0.82336; residual: 6.76107
The Approximate Root is --->  3.44217


guess 2 tol: 0.0001
iter 1; guess 8; solution update 6; residual: 360
iter 2; guess 5.70701; solution update 2.29299; residual: 103.616
iter 3; guess 4.26553; solution update 1.44148; residual: 28.4241
iter 4; guess 3.44217; solution update 0.82336; residual: 6.76107
iter 5; guess 3.0821; solution update 0.360074; residual: 1.03287
iter 6; guess 3.00367; solution update 0.0784289; residual: 0.0440901
iter 7; guess 3.00001; solution update 0.00365851; residual: 9.37913e-05
iter 8; guess 3; solution update 7.81587e-06; residual: 4.27615e-10
The Approximate Root is --->  3


guess 2 tol: 1e-08
iter 1; guess 8; solution update 6; residual: 360
iter 2; guess 5.70701; solution update 2.29299; residual: 103.616
iter 3; guess 4.26553; solution update 1.44148; residual: 28.4241
iter 4; guess 3.44217; solution update 0.82336; residual: 6.76107
iter 5; guess 3.0821; solution update 0.360074; residual: 1.03287
iter 6; guess 3.00367; solution update 0.0784289; residual: 0.0440901
iter 7; guess 3.00001; solution update 0.00365851; residual: 9.37913e-05
```

```
iter 8; guess 3; solution update 7.81587e-06; residual: 4.27615e-10
iter 9; guess 3; solution update 3.56346e-11; residual: 0
The Approximate Root is --->  3

ANS-> -1.41176
ANS-> -1
ANS-> -1
ANS-> 0
ANS-> 0
ANS-> 0
ANS-> 3.44217
ANS-> 3
ANS-> 3
```

After inspecting these results, I can see that depending on the guess provided for newton's method, the procedure will find a different root. For instance, with a guess of -2, newton's method approximates the root to be -1, which makes sense as this is where the nearest root is located to the original guess. With a guess of 1, the root approximated was 0 and with a guess of 2, the approximated root is 3. The best guess was 1 as it was able to find the root of the function in 1 iteration. For the other two guesses, as the tolerance decreased, the root guess got closer and closer to the correct number until it reached the root in 7 or 8 iterations. Therefore, I can conclude that in some cases (like with a guess of 2) the smaller the tolerance for error, newton will find a more accurate approximate root.

Part II: Finite-Difference Newton's Method

For Part II, implemented a Finite-Difference Newton Method to approximate the root of a given function. In Part I, Newton's Method uses the derivative of the function in order to determine the root of the function. However, if the derivative is 0 then newton's method will fail because newton's has the derivative in the denominator. Therefore, the finite-difference method finds an approximation for the derivative using a forward finite difference:

$$F'(x) \approx [f(x + \text{alpha}) - f(x)] / \text{alpha}$$, where alpha is an approximate increment

fd_newton.cpp

```cpp
#include <iostream>
#include <cmath>

using namespace std;

double fd_newton(Fcn& f, double x, int maxit, double tol, double alpha, bool show_iterates) {
  cout << "guess " << x << " tol: " << tol << " alpha: " << alpha << endl;
  int iteration = 0;
  double solutionUpdate = 10;

  // -- while loop till max iterations or until solution update is less than the tolerance
  while(iteration <= maxit && solutionUpdate > tol) {
    iteration++;
    double previous = x;

    // --  dpF which is an approximation of the derivative.
    double dpF = (f(x + alpha) - f(x)) / alpha;

    // -- x = x(n+1) by (x - f(x)/dpF(x))
    x = x - (f(x)/dpF);

    // -- solution update = |x(n+1) - x|
    solutionUpdate = fabs(x - previous);

    if (show_iterates) {
      //-- output the iteration, guess, solution update and residual for each iteration
      cout << "iter " << iteration << ", x = " << x << ", |h| = " << solutionUpdate << "; |f(x)| =  " << fabs(f(x)) << endl;
    }
  }
  cout << endl;
  return x;
}

```

To test its functionality, I will use the same guesses and tolerances as before, but I will also be using differencing parameters for alpha at { 2^{-4} , 2^{-26} , 2^{-50} }. I will Perform these tests for a total of 27 times. Instead of looping through 20 times and then having a break after an if statement if the solution Update was less then the tolerance I chose to just include it in a while loop. This reduces the amount of code in the function but does the same number of iterations. My code for the test driver is on the following page.

test_fd_newton.cpp

```cpp
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
#include <iomanip>
#include <math.h>
#include "fcn.hpp"
#include "fd_newton.cpp"

using namespace std;

// -- f(x)
class fcn : public Fcn {
public:
  double operator()(double x) {
    // f(x) = x(x - 3)(x + 1)
    return x * (x-3) * (x+1);
  }
};

int main(int argc, char* argv[]) {
    // setting up alpha = {2^-4, 2^-26, and 2^-50}
    double alpha[3] = { pow(2.0, -4), pow(2.0, -26), pow(2.0, -50) };
    double guess[3] = { -2, 1, 2 };  // initial guesses
    double tols[3] = { 1e-1, 1e-5, 1e-9 }; // tolerances
    double ans[27]; // array to store results and print to screen following fp_newtons output
    int maxit = 20;  // max set to 20 iterations of fp_newton

    fcn f;
    int index = 0;

    // iterate through tolerances
    for(unsigned int i = 0; i < 3; i++) {
      // iterate through the guesses
      for(unsigned int j = 0; j < 3; j++) {
        // iterate through alpha
        for (int k = 0; k < 3; k++) {
          ans[index] = fd_newton(f, guess[j], maxit, tols[i], alpha[k], true);
          index++;
        }
      }
    }

    // -- output all approximation answers
    for (int i =0; i < 27; i++) {
      cout << "ANS-> " << ans[i] << endl;
    }
    return 0;
}
```

RESULTS:

```
guess -2 tol: 0.1 alpha: 0.0625
iter 1, x = -1.39408, |h| = 0.605917; |f(x)| =  2.41404
iter 2, x = -1.09324, |h| = 0.300843; |f(x)| =  0.417238
iter 3, x = -1.00317, |h| = 0.0900695; |f(x)| =  0.0127316

guess -2 tol: 0.1 alpha: 1.49012e-08
iter 1, x = -1.41176, |h| = 0.588235; |f(x)| =  2.56462
iter 2, x = -1.11446, |h| = 0.297303; |f(x)| =  0.524854
iter 3, x = -1.01322, |h| = 0.101246; |f(x)| =  0.0537363
iter 4, x = -1.00021, |h| = 0.0130028; |f(x)| =  0.000849867

guess -2 tol: 0.1 alpha: 8.88178e-16
iter 1, x = -1.375, |h| = 0.625; |f(x)| =  2.25586
iter 2, x = -1.1096, |h| = 0.265395; |f(x)| =  0.499802
iter 3, x = -1.01208, |h| = 0.0975223; |f(x)| =  0.0490616

guess 1 tol: 0.1 alpha: 0.0625
iter 1, x = -0.0168818, |h| = 1.01688; |f(x)| =  0.0500707
iter 2, x = -0.000496782, |h| = 0.016385; |f(x)| =  0.00148985

guess 1 tol: 0.1 alpha: 1.49012e-08
iter 1, x = 0, |h| = 1; |f(x)| =  0
iter 2, x = 0, |h| = 0; |f(x)| =  0

guess 1 tol: 0.1 alpha: 8.88178e-16
iter 1, x = 0, |h| = 1; |f(x)| =  0
iter 2, x = 0, |h| = 0; |f(x)| =  0

guess 2 tol: 0.1 alpha: 0.0625
iter 1, x = 6.78505, |h| = 4.78505; |f(x)| =  199.933
iter 2, x = 4.95284, |h| = 1.83221; |f(x)| =  57.5764
iter 3, x = 3.83676, |h| = 1.11608; |f(x)| =  15.5281
iter 4, x = 3.24887, |h| = 0.587887; |f(x)| =  3.43541
iter 5, x = 3.03626, |h| = 0.212613; |f(x)| =  0.444338
iter 6, x = 3.00197, |h| = 0.034286; |f(x)| =  0.0236834

guess 2 tol: 0.1 alpha: 1.49012e-08
iter 1, x = 8, |h| = 6; |f(x)| =  360
iter 2, x = 5.70701, |h| = 2.29299; |f(x)| =  103.616
iter 3, x = 4.26553, |h| = 1.44148; |f(x)| =  28.4241
iter 4, x = 3.44217, |h| = 0.82336; |f(x)| =  6.76107
iter 5, x = 3.0821, |h| = 0.360074; |f(x)| =  1.03287
iter 6, x = 3.00367, |h| = 0.0784288; |f(x)| =  0.0440901

guess 2 tol: 0.1 alpha: 8.88178e-16
iter 1, x = 8, |h| = 6; |f(x)| =  360
iter 2, x = -inf, |h| = inf; |f(x)| =  inf
iter 3, x = nan, |h| = nan; |f(x)| =  nan

guess -2 tol: 1e-05 alpha: 0.0625
iter 1, x = -1.39408, |h| = 0.605917; |f(x)| =  2.41404
iter 2, x = -1.09324, |h| = 0.300843; |f(x)| =  0.417238
iter 3, x = -1.00317, |h| = 0.0900695; |f(x)| =  0.0127316
iter 4, x = -0.99975, |h| = 0.00342012; |f(x)| =  0.000998904
iter 5, x = -1.00002, |h| = 0.000270782; |f(x)| =  8.39157e-05
iter 6, x = -0.999998, |h| = 2.27314e-05; |f(x)| =  7.01229e-06
```

```
iter 7, x = -1, |h| = 1.89963e-06; |f(x)| =  5.86232e-07

guess -2 tol: 1e-05 alpha: 1.49012e-08
iter 1, x = -1.41176, |h| = 0.588235; |f(x)| =  2.56462
iter 2, x = -1.11446, |h| = 0.297303; |f(x)| =  0.524854
iter 3, x = -1.01322, |h| = 0.101246; |f(x)| =  0.0537363
iter 4, x = -1.00021, |h| = 0.0130028; |f(x)| =  0.000849867
iter 5, x = -1, |h| = 0.000212354; |f(x)| =  2.25474e-07
iter 6, x = -1, |h| = 5.63686e-08; |f(x)| =  1.15463e-14

guess -2 tol: 1e-05 alpha: 8.88178e-16
iter 1, x = -1.375, |h| = 0.625; |f(x)| =  2.25586
iter 2, x = -1.1096, |h| = 0.265395; |f(x)| =  0.499802
iter 3, x = -1.01208, |h| = 0.0975223; |f(x)| =  0.0490616
iter 4, x = -1.00017, |h| = 0.0119163; |f(x)| =  0.000664883
iter 5, x = -1, |h| = 0.00016615; |f(x)| =  1.45853e-07
iter 6, x = -1, |h| = 3.64634e-08; |f(x)| =  7.10543e-15

guess 1 tol: 1e-05 alpha: 0.0625
iter 1, x = -0.0168818, |h| = 1.01688; |f(x)| =  0.0500707
iter 2, x = -0.000496782, |h| = 0.016385; |f(x)| =  0.00148985
iter 3, x = -1.91426e-05, |h| = 0.00047764; |f(x)| =  5.74272e-05
iter 4, x = -7.4251e-07, |h| = 1.84001e-05; |f(x)| =  2.22753e-06
iter 5, x = -2.8808e-08, |h| = 7.13702e-07; |f(x)| =  8.64239e-08

guess 1 tol: 1e-05 alpha: 1.49012e-08
iter 1, x = 0, |h| = 1; |f(x)| =  0
iter 2, x = 0, |h| = 0; |f(x)| =  0

guess 1 tol: 1e-05 alpha: 8.88178e-16
iter 1, x = 0, |h| = 1; |f(x)| =  0
iter 2, x = 0, |h| = 0; |f(x)| =  0

guess 2 tol: 1e-05 alpha: 0.0625
iter 1, x = 6.78505, |h| = 4.78505; |f(x)| =  199.933
iter 2, x = 4.95284, |h| = 1.83221; |f(x)| =  57.5764
iter 3, x = 3.83676, |h| = 1.11608; |f(x)| =  15.5281
iter 4, x = 3.24887, |h| = 0.587887; |f(x)| =  3.43541
iter 5, x = 3.03626, |h| = 0.212613; |f(x)| =  0.444338
iter 6, x = 3.00197, |h| = 0.034286; |f(x)| =  0.0236834
iter 7, x = 3.00007, |h| = 0.00189932; |f(x)| =  0.000864342
iter 8, x = 3, |h| = 6.94673e-05; |f(x)| =  3.0698e-05
iter 9, x = 3, |h| = 2.4674e-06; |f(x)| =  1.08917e-06

guess 2 tol: 1e-05 alpha: 1.49012e-08
iter 1, x = 8, |h| = 6; |f(x)| =  360
iter 2, x = 5.70701, |h| = 2.29299; |f(x)| =  103.616
iter 3, x = 4.26553, |h| = 1.44148; |f(x)| =  28.4241
iter 4, x = 3.44217, |h| = 0.82336; |f(x)| =  6.76107
iter 5, x = 3.0821, |h| = 0.360074; |f(x)| =  1.03287
iter 6, x = 3.00367, |h| = 0.0784288; |f(x)| =  0.0440901
iter 7, x = 3.00001, |h| = 0.00365851; |f(x)| =  9.37917e-05
iter 8, x = 3, |h| = 7.8159e-06; |f(x)| =  4.28436e-10

guess 2 tol: 1e-05 alpha: 8.88178e-16
iter 1, x = 8, |h| = 6; |f(x)| =  360
iter 2, x = -inf, |h| = inf; |f(x)| =  inf
iter 3, x = nan, |h| = nan; |f(x)| =  nan

guess -2 tol: 1e-09 alpha: 0.0625
```

```
iter 1, x = -1.39408, |h| = 0.605917; |f(x)| =  2.41404
iter 2, x = -1.09324, |h| = 0.300843; |f(x)| =  0.417238
iter 3, x = -1.00317, |h| = 0.0900695; |f(x)| =  0.0127316
iter 4, x = -0.99975, |h| = 0.00342012; |f(x)| =  0.000998904
iter 5, x = -1.00002, |h| = 0.000270782; |f(x)| =  8.39157e-05
iter 6, x = -0.999998, |h| = 2.27314e-05; |f(x)| =  7.01229e-06
iter 7, x = -1, |h| = 1.89963e-06; |f(x)| =  5.86232e-07
iter 8, x = -1, |h| = 1.5881e-07; |f(x)| =  4.90077e-08
iter 9, x = -1, |h| = 1.32761e-08; |f(x)| =  4.09694e-09
iter 10, x = -1, |h| = 1.10986e-09; |f(x)| =  3.42495e-10
iter 11, x = -1, |h| = 9.27819e-11; |f(x)| =  2.86322e-11

guess -2 tol: 1e-09 alpha: 1.49012e-08
iter 1, x = -1.41176, |h| = 0.588235; |f(x)| =  2.56462
iter 2, x = -1.11446, |h| = 0.297303; |f(x)| =  0.524854
iter 3, x = -1.01322, |h| = 0.101246; |f(x)| =  0.0537363
iter 4, x = -1.00021, |h| = 0.0130028; |f(x)| =  0.000849867
iter 5, x = -1, |h| = 0.000212354; |f(x)| =  2.25474e-07
iter 6, x = -1, |h| = 5.63686e-08; |f(x)| =  1.15463e-14
iter 7, x = -1, |h| = 2.88658e-15; |f(x)| =  0

guess -2 tol: 1e-09 alpha: 8.88178e-16
iter 1, x = -1.375, |h| = 0.625; |f(x)| =  2.25586
iter 2, x = -1.1096, |h| = 0.265395; |f(x)| =  0.499802
iter 3, x = -1.01208, |h| = 0.0975223; |f(x)| =  0.0490616
iter 4, x = -1.00017, |h| = 0.0119163; |f(x)| =  0.000664883
iter 5, x = -1, |h| = 0.00016615; |f(x)| =  1.45853e-07
iter 6, x = -1, |h| = 3.64634e-08; |f(x)| =  7.10543e-15
iter 7, x = -1, |h| = 1.77636e-15; |f(x)| =  0

guess 1 tol: 1e-09 alpha: 0.0625
iter 1, x = -0.0168818, |h| = 1.01688; |f(x)| =  0.0500707
iter 2, x = -0.000496782, |h| = 0.016385; |f(x)| =  0.00148985
iter 3, x = -1.91426e-05, |h| = 0.00047764; |f(x)| =  5.74272e-05
iter 4, x = -7.4251e-07, |h| = 1.84001e-05; |f(x)| =  2.22753e-06
iter 5, x = -2.8808e-08, |h| = 7.13702e-07; |f(x)| =  8.64239e-08
iter 6, x = -1.11771e-09, |h| = 2.76903e-08; |f(x)| =  3.35312e-09
iter 7, x = -4.33653e-11, |h| = 1.07434e-09; |f(x)| =  1.30096e-10
iter 8, x = -1.68251e-12, |h| = 4.16828e-11; |f(x)| =  5.04753e-12

guess 1 tol: 1e-09 alpha: 1.49012e-08
iter 1, x = 0, |h| = 1; |f(x)| =  0
iter 2, x = 0, |h| = 0; |f(x)| =  0

guess 1 tol: 1e-09 alpha: 8.88178e-16
iter 1, x = 0, |h| = 1; |f(x)| =  0
iter 2, x = 0, |h| = 0; |f(x)| =  0

guess 2 tol: 1e-09 alpha: 0.0625
iter 1, x = 6.78505, |h| = 4.78505; |f(x)| =  199.933
iter 2, x = 4.95284, |h| = 1.83221; |f(x)| =  57.5764
iter 3, x = 3.83676, |h| = 1.11608; |f(x)| =  15.5281
iter 4, x = 3.24887, |h| = 0.587887; |f(x)| =  3.43541
iter 5, x = 3.03626, |h| = 0.212613; |f(x)| =  0.444338
iter 6, x = 3.00197, |h| = 0.034286; |f(x)| =  0.0236834
iter 7, x = 3.00007, |h| = 0.00189932; |f(x)| =  0.000864342
iter 8, x = 3, |h| = 6.94673e-05; |f(x)| =  3.0698e-05
iter 9, x = 3, |h| = 2.4674e-06; |f(x)| =  1.08917e-06
iter 10, x = 3, |h| = 8.7544e-08; |f(x)| =  3.86425e-08
iter 11, x = 3, |h| = 3.10596e-09; |f(x)| =  1.37099e-09
```

```
iter 12, x = 3, |h| = 1.10196e-10; |f(x)| =  4.86384e-11

guess 2 tol: 1e-09 alpha: 1.49012e-08
iter 1, x = 8, |h| = 6; |f(x)| =  360
iter 2, x = 5.70701, |h| = 2.29299; |f(x)| =  103.616
iter 3, x = 4.26553, |h| = 1.44148; |f(x)| =  28.4241
iter 4, x = 3.44217, |h| = 0.82336; |f(x)| =  6.76107
iter 5, x = 3.0821, |h| = 0.360074; |f(x)| =  1.03287
iter 6, x = 3.00367, |h| = 0.0784288; |f(x)| =  0.0440901
iter 7, x = 3.00001, |h| = 0.00365851; |f(x)| =  9.37917e-05
iter 8, x = 3, |h| = 7.8159e-06; |f(x)| =  4.28436e-10
iter 9, x = 3, |h| = 3.5703e-11; |f(x)| =  0

guess 2 tol: 1e-09 alpha: 8.88178e-16
iter 1, x = 8, |h| = 6; |f(x)| =  360
iter 2, x = -inf, |h| = inf; |f(x)| =  inf
iter 3, x = nan, |h| = nan; |f(x)| =  nan
```

Based on these results I can see that fd_newton does not work in some instances. When the initial guess is 2 and alpha is 2^{-50} the approximate root becomes becomes nan, regardless of what the tolerance is. This is most likely caused by the alpha being too small and the computer representing 2^{-50} as 0, causing the substitute of f'(x) to equal an invalid number.

For the guess of -2 I noticed that as the tolerance decreased it took more iterations in order to find the root of f(x). Also the fd_newton found the root as -1 during iteration 5 but it would continue for 2 or 3 more iterations depending on the tolerance given. The computer cannot represent the value of x that is actually being used as the guess because the number 1.0…001 cannot be represented in the 16-bit number that the computer uses.

Another observation is that, like newton, fd_newton works best when the initial guess is 1. Fd_newton finds the root 0 in 2 iterations just like newton. fd_newton does, however, take up to 12 iterations in some cases when newton's most iterations were 9. It also takes a higher tolerance in order to guess the root -1 whereas in newton's it could find the root with the tolerance of 1.

Part III: Kepler

Part III, I used the newton function from Part I in order to solve Kepler's equation for w at various times t. Kepler's equation is used to determine the position of an object in an elliptical orbit and its nonlinear equation is Esin(w) – w = t, where E is the objects orbital eccentricity and w is the angle of the object in orbit. E can be found by taking the square root of $(1 - b^2 / a^2)$. In this problem we will use values a = 2.0 and b = 1.25. My task is to solve for w(t) at the times t = {0, 0.001, 0.002, …, 10} with an initial guess of w = 0. After using newton to solve for the next guess, I will update the time, t, to the next iteration and find the Cartesian coordinates: x(t) = rcos(w) and y(t) = rsin(w) and save them to files so that I may upload and plot these coordinates vs time.

kepler.cpp - main

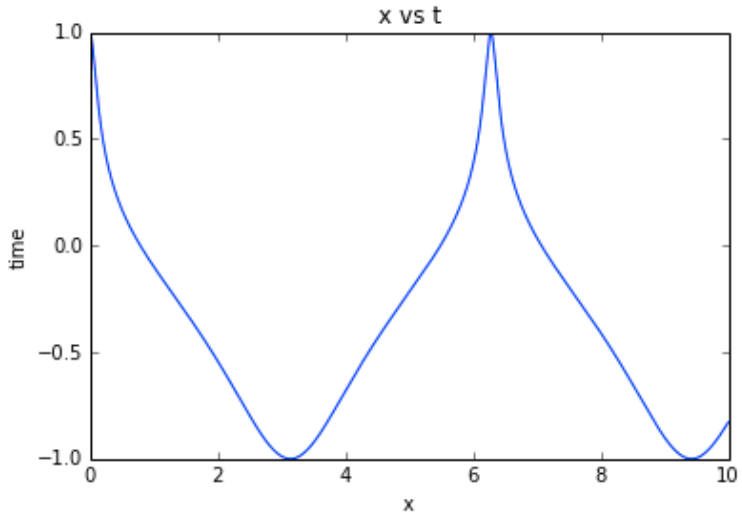
```
int main(int argc, char* argv[]) {
  // -- Matrix of t for times.
  Matrix t = Linspace(0, 10, 10001);
  t.Write("t.txt");
  Matrix w = Matrix(10001);
  Matrix x = Matrix(10001);
  Matrix y = Matrix(10001);

  fcn f; // f(w)
  fdx df; // df(w)
  fr r; // r(w)

  for (int i = 0; i < 10001; i++) {
    double guess;
    if (i == 0) {
      guess = 0; // for first iteration guess is 0
    } else {
      guess = w(i - 1); // get the last approximation and use for the guess
    }
    f.setTime(t(i)); // update t with next time

    // -- solve for w(t)
    w(i) = newton(f, df, guess, 6, 1e-5, false);

    // -- solve the Cartesian Coordinates
    x(i) = r(w(i))*cos(w(i));
    y(i) = r(w(i))*sin(w(i));
  }
  w.Write("w.txt");
  x.Write("x.txt");
  y.Write("y.txt");


  return 0;
}
```

The functions f, df, and r were made into classes:

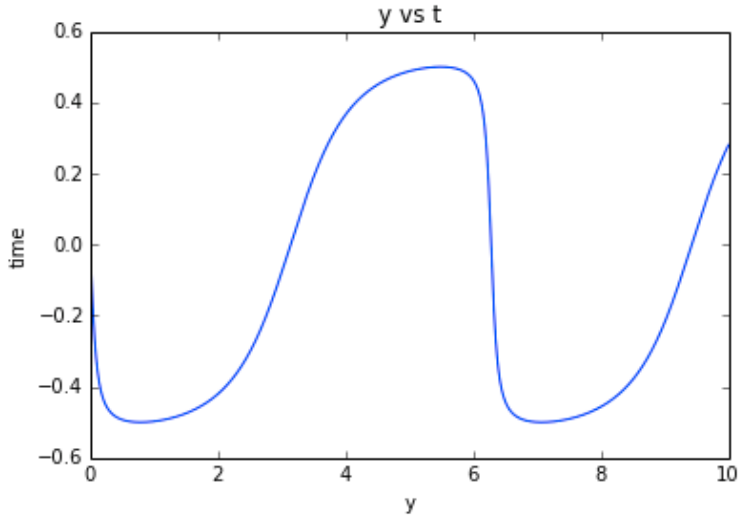
```
class fcn : public Fcn {
private:
  // epsilon
  double eps = sqrt(1 - (pow(1.25, 2) / pow(2.0, 2)));
  // the initial time of t = 0;
  double t = 0;
public:
  double operator()(double x) {
    // return f(w)
    return eps * sin(x) - x - t;
  }
  void setTime(double x) {
    // update the time for each iteration
    t = x;
  }
};

// -- f'(x)
class fdx : public Fcn {
private:
  // epsilon
  double eps = sqrt(1 - (pow(1.25, 2) / pow(2.0, 2)));
public:
  double operator()(double x) {
    // f'(w)
    return eps * cos(x) - 1;
  }
};

class fr : public Fcn {
public:
  double operator()(double x) {
    // r(w)
    return .5/sqrt(pow(.5*cos(x), 2) + sin(x)*sin(x));
  }
};
```

After running my code I graphed x vs t, y vs t, and x vs y and found that my newton function correctly depicted the elliptical orbit of the object defined by Esin(w) – w = t.

x_vs_t.png



y_vs_t.png



X_vs_y.png