proj2_report

October 8, 2015

1 Project 2: Non Linear Solvers

1.1 by Arslan Memon

1.1.1 Part 1: Newton's Method

For part 1 of the project, we were required to create a c++ method that performs the Newton root finding function. This method had the following parameters: a function, an initial guess, the derivative of the function, the tolerance level, a max number of iterations, and whether or not to show data from each iteration. I started writing this method by looking at the psuedo code in the book and converting it to c++. Then I added restraints to the method, such as making sure that the number of iterations was greater than or equal to 1 and the tolerance wasn't less then the limit that I set of 1e-15, since machine epsilon for double precision is close to 5e-16.

After creating the Newton function, we were asked to test the method in another cpp file by finding the roots for the function f(x) $\equiv$ x(x−3)(x+1) = 0 using initial guesses of x_0={-2, 1, 2}. For each of the initial guesses, we did three tests with different tolerances of $\{10^{-1}, 10^{-5}, 10^{-9}\}$ and 15 iterations maximum. A function object was used to evaluate the function for a certain x value. Instead of evaluating f(x) $\equiv$ x(x−3)(x+1), I expanded the polynomial to $x^3 - 2x^2 - 3x$ and used Horner's method to have the function evaluated as (((x-2)*x-3)*x) in order to make the calculations efficient. I did the same for the derivative as well, which in nested form is (((3)*x-4)*x-3).

The following is the output of the test data(note: d is the solution update):

```
Initial guess x=-2, |fx|=10, tolerance= 0.1
iteration=0, x=-1.41176, |d|=0.588235, |fx|= 2.56462
iteration=1, x=-1.11446, |d|=0.297303, |fx|= 0.524854
iteration=2, x=-1.01322, |d|=0.101246, |fx|= 0.0537364
iteration=3, x=-1.00021, |d|=0.0130028, |fx|= 0.000849868
Initial guess x=-2, |fx|=10, tolerance= 1e-05
iteration=0, x=-1.41176, |d|=0.588235, |fx|= 2.56462
iteration=1, x=-1.11446, |d|=0.297303, |fx|= 0.524854
iteration=2, x=-1.01322, |d|=0.101246, |fx|= 0.0537364
iteration=3, x=-1.00021, |d|=0.0130028, |fx|= 0.000849868
iteration=4, x=-1, |d|=0.000212354, |fx|= 2.25491e-07
iteration=5, x=-1, |d|=5.63727e-08, |fx|= 1.59872e-14
Initial guess x=-2, |fx|=10, tolerance= 1e-09
iteration=0, x=-1.41176, |d|=0.588235, |fx|= 2.56462
iteration=1, x=-1.11446, |d|=0.297303, |fx|= 0.524854
iteration=2, x=-1.01322, |d|=0.101246, |fx|= 0.0537364
iteration=3, x=-1.00021, |d|=0.0130028, |fx|= 0.000849868
iteration=4, x=-1, |d|=0.000212354, |fx|= 2.25491e-07
iteration=5, x=-1, |d|=5.63727e-08, |fx|= 1.59872e-14
iteration=6, x=-1, |d|=3.9968e-15, |fx|= 0
Initial guess x=1, |fx|=4, tolerance= 0.1
iteration=0, x=0, |d|=1, |fx|= 0
```

```
iteration=1, x=0, |d|=0, |fx|= 0
Initial guess x=1, |fx|=4, tolerance= 1e-05
iteration=0, x=0, |d|=1, |fx|= 0
iteration=1, x=0, |d|=0, |fx|= 0
Initial guess x=1, |fx|=4, tolerance= 1e-09
iteration=0, x=0, |d|=1, |fx|= 0
iteration=1, x=0, |d|=0, |fx|= 0
Initial guess x=2, |fx|=6, tolerance= 0.1
iteration=0, x=8, |d|=6, |fx|= 360
iteration=1, x=5.70701, |d|=2.29299, |fx|= 103.616
iteration=2, x=4.26553, |d|=1.44148, |fx|= 28.4241
iteration=3, x=3.44217, |d|=0.82336, |fx|= 6.76107
iteration=4, x=3.0821, |d|=0.360074, |fx|= 1.03287
iteration=5, x=3.00367, |d|=0.0784289, |fx|= 0.0440901
Initial guess x=2, |fx|=6, tolerance= 1e-05
iteration=0, x=8, |d|=6, |fx|= 360
iteration=1, x=5.70701, |d|=2.29299, |fx|= 103.616
iteration=2, x=4.26553, |d|=1.44148, |fx|= 28.4241
iteration=3, x=3.44217, |d|=0.82336, |fx|= 6.76107
iteration=4, x=3.0821, |d|=0.360074, |fx|= 1.03287
iteration=5, x=3.00367, |d|=0.0784289, |fx|= 0.0440901
iteration=6, x=3.00001, |d|=0.00365851, |fx|= 9.37913e-05
iteration=7, x=3, |d|=7.81587e-06, |fx|= 4.27615e-10
Initial guess x=2, |fx|=6, tolerance= 1e-09
iteration=0, x=8, |d|=6, |fx|= 360
iteration=1, x=5.70701, |d|=2.29299, |fx|= 103.616
iteration=2, x=4.26553, |d|=1.44148, |fx|= 28.4241
iteration=3, x=3.44217, |d|=0.82336, |fx|= 6.76107
iteration=4, x=3.0821, |d|=0.360074, |fx|= 1.03287
iteration=5, x=3.00367, |d|=0.0784289, |fx|= 0.0440901
iteration=6, x=3.00001, |d|=0.00365851, |fx|= 9.37913e-05
iteration=7, x=3, |d|=7.81587e-06, |fx|= 4.27615e-10
iteration=8, x=3, |d|=3.56346e-11, |fx|= 0
```

As can be seen by the data, as the tolerance decreases, the number of iterations it takes to find a root within the tolerance level increases. However, the initial guess of x=1 is an exception to this observation because it only takes two iterations to find the root and the precision is very close to 0 after only the 2nd iteration.

Another interesting observation that can be made is that as the initial guess changes, so does the root found by the newton method. The root that is found is the closest root to the initial guess. This is because the slope of the equation f(x) is steadily converging to the root, so every time a zero of the tangent line is found, the magnitude of the slope at that point is less than the previous point, thus the new zero of the tangent line will be closer to the root that is closest to the initial guess.

1.1.2 Part 2: Finite-Difference Newton Method

In part two of the project, we were required to create a finite difference Newton method, which was basically the same as the Newton method, but the parameters didn't include a derivative of a function. Instead, a new parameter of alpha was added to the method. The concept of this method was to replace f'(x) with the forward finite difference to approximate f'(x). The approximation for f'(x) using forward finite difference is $(f(x+\alpha) - f(x))/\alpha$. To test this method, the 9 tests that were used in Newton were repeated 3 times with different α's of $\{2^{-4}, 2^{-26}, 2^{-50}\}$.

The following are the results from fd_newton:

```
Initial guess, x=-2, |fx|=10, tolerance= 0.1
```

```
iteration=0, x=-1.39408, |d|=0.605917, |fx|= 2.41404
iteration=1, x=-1.09324, |d|=0.300843, |fx|= 0.417238
iteration=2, x=-1.00317, |d|=0.0900695, |fx|= 0.0127316
Initial guess, x=-2, |fx|=10, tolerance= 0.1
iteration=0, x=-1.41176, |d|=0.588235, |fx|= 2.56462
iteration=1, x=-1.11446, |d|=0.297303, |fx|= 0.524854
iteration=2, x=-1.01322, |d|=0.101246, |fx|= 0.0537363
iteration=3, x=-1.00021, |d|=0.0130028, |fx|= 0.000849867
Initial guess, x=-2, |fx|=10, tolerance= 0.1
iteration=0, x=-1.375, |d|=0.625, |fx|= 2.25586
iteration=1, x=-1.1096, |d|=0.265395, |fx|= 0.499802
iteration=2, x=-1.00708, |d|=0.102523, |fx|= 0.0285764
iteration=3, x=-1.00004, |d|=0.00704095, |fx|= 0.000161483

Initial guess, x=-2, |fx|=10, tolerance= 1e-05
iteration=0, x=-1.39408, |d|=0.605917, |fx|= 2.41404
iteration=1, x=-1.09324, |d|=0.300843, |fx|= 0.417238
iteration=2, x=-1.00317, |d|=0.0900695, |fx|= 0.0127316
iteration=3, x=-0.99975, |d|=0.00342012, |fx|= 0.000998904
iteration=4, x=-1.00002, |d|=0.000270782, |fx|= 8.39157e-05
iteration=5, x=-0.999998, |d|=2.27314e-05, |fx|= 7.01229e-06
iteration=6, x=-1, |d|=1.89963e-06, |fx|= 5.86232e-07
Initial guess, x=-2, |fx|=10, tolerance= 1e-05
iteration=0, x=-1.41176, |d|=0.588235, |fx|= 2.56462
iteration=1, x=-1.11446, |d|=0.297303, |fx|= 0.524854
iteration=2, x=-1.01322, |d|=0.101246, |fx|= 0.0537363
iteration=3, x=-1.00021, |d|=0.0130028, |fx|= 0.000849867
iteration=4, x=-1, |d|=0.000212354, |fx|= 2.25474e-07
iteration=5, x=-1, |d|=5.63684e-08, |fx|= 1.37668e-14
Initial guess, x=-2, |fx|=10, tolerance= 1e-05
iteration=0, x=-1.375, |d|=0.625, |fx|= 2.25586
iteration=1, x=-1.1096, |d|=0.265395, |fx|= 0.499802
iteration=2, x=-1.00708, |d|=0.102523, |fx|= 0.0285764
iteration=3, x=-1.00004, |d|=0.00704095, |fx|= 0.000161483
iteration=4, x=-1, |d|=4.03674e-05, |fx|= 5.40282e-09
iteration=5, x=-1, |d|=1.3507e-09, |fx|= 0

Initial guess, x=-2, |fx|=10, tolerance= 1e-09
iteration=0, x=-1.39408, |d|=0.605917, |fx|= 2.41404
iteration=1, x=-1.09324, |d|=0.300843, |fx|= 0.417238
iteration=2, x=-1.00317, |d|=0.0900695, |fx|= 0.0127316
iteration=3, x=-0.99975, |d|=0.00342012, |fx|= 0.000998904
iteration=4, x=-1.00002, |d|=0.000270782, |fx|= 8.39157e-05
iteration=5, x=-0.999998, |d|=2.27314e-05, |fx|= 7.01229e-06
iteration=6, x=-1, |d|=1.89963e-06, |fx|= 5.86232e-07
iteration=7, x=-1, |d|=1.5881e-07, |fx|= 4.90077e-08
iteration=8, x=-1, |d|=1.32761e-08, |fx|= 4.09694e-09
iteration=9, x=-1, |d|=1.10986e-09, |fx|= 3.42495e-10
iteration=10, x=-1, |d|=9.27818e-11, |fx|= 2.86322e-11
Initial guess, x=-2, |fx|=10, tolerance= 1e-09
iteration=0, x=-1.41176, |d|=0.588235, |fx|= 2.56462
iteration=1, x=-1.11446, |d|=0.297303, |fx|= 0.524854
iteration=2, x=-1.01322, |d|=0.101246, |fx|= 0.0537363
iteration=3, x=-1.00021, |d|=0.0130028, |fx|= 0.000849867
```

```
iteration=4, x=-1, |d|=0.000212354, |fx|= 2.25474e-07
iteration=5, x=-1, |d|=5.63684e-08, |fx|= 1.37668e-14
iteration=6, x=-1, |d|=3.44169e-15, |fx|= 4.44089e-16
Initial guess, x=-2, |fx|=10, tolerance= 1e-09
iteration=0, x=-1.375, |d|=0.625, |fx|= 2.25586
iteration=1, x=-1.1096, |d|=0.265395, |fx|= 0.499802
iteration=2, x=-1.00708, |d|=0.102523, |fx|= 0.0285764
iteration=3, x=-1.00004, |d|=0.00704095, |fx|= 0.000161483
iteration=4, x=-1, |d|=4.03674e-05, |fx|= 5.40282e-09
iteration=5, x=-1, |d|=1.3507e-09, |fx|= 0
iteration=6, x=-1, |d|=0, |fx|= 0

Initial guess, x=1, |fx|=4, tolerance= 0.1
iteration=0, x=-0.0168818, |d|=1.01688, |fx|= 0.0500707
iteration=1, x=-0.000496782, |d|=0.016385, |fx|= 0.00148985
Initial guess, x=1, |fx|=4, tolerance= 0.1
iteration=0, x=0, |d|=1, |fx|= 0
iteration=1, x=0, |d|=0, |fx|= 0
Initial guess, x=1, |fx|=4, tolerance= 0.1
iteration=0, x=0, |d|=1, |fx|= 0
iteration=1, x=0, |d|=0, |fx|= 0

Initial guess, x=1, |fx|=4, tolerance= 1e-05
iteration=0, x=-0.0168818, |d|=1.01688, |fx|= 0.0500707
iteration=1, x=-0.000496782, |d|=0.016385, |fx|= 0.00148985
iteration=2, x=-1.91426e-05, |d|=0.00047764, |fx|= 5.74272e-05
iteration=3, x=-7.4251e-07, |d|=1.84001e-05, |fx|= 2.22753e-06
iteration=4, x=-2.8808e-08, |d|=7.13702e-07, |fx|= 8.64239e-08
Initial guess, x=1, |fx|=4, tolerance= 1e-05
iteration=0, x=0, |d|=1, |fx|= 0
iteration=1, x=0, |d|=0, |fx|= 0
Initial guess, x=1, |fx|=4, tolerance= 1e-05
iteration=0, x=0, |d|=1, |fx|= 0
iteration=1, x=0, |d|=0, |fx|= 0

Initial guess, x=1, |fx|=4, tolerance= 1e-09
iteration=0, x=-0.0168818, |d|=1.01688, |fx|= 0.0500707
iteration=1, x=-0.000496782, |d|=0.016385, |fx|= 0.00148985
iteration=2, x=-1.91426e-05, |d|=0.00047764, |fx|= 5.74272e-05
iteration=3, x=-7.4251e-07, |d|=1.84001e-05, |fx|= 2.22753e-06
iteration=4, x=-2.8808e-08, |d|=7.13702e-07, |fx|= 8.64239e-08
iteration=5, x=-1.11771e-09, |d|=2.76903e-08, |fx|= 3.35312e-09
iteration=6, x=-4.33653e-11, |d|=1.07434e-09, |fx|= 1.30096e-10
iteration=7, x=-1.68251e-12, |d|=4.16828e-11, |fx|= 5.04753e-12
Initial guess, x=1, |fx|=4, tolerance= 1e-09
iteration=0, x=0, |d|=1, |fx|= 0
iteration=1, x=0, |d|=0, |fx|= 0
Initial guess, x=1, |fx|=4, tolerance= 1e-09
iteration=0, x=0, |d|=1, |fx|= 0
iteration=1, x=0, |d|=0, |fx|= 0

Initial guess, x=2, |fx|=6, tolerance= 0.1
iteration=0, x=6.78505, |d|=4.78505, |fx|= 199.933
iteration=1, x=4.95284, |d|=1.83221, |fx|= 57.5764
```

```
iteration=2, x=3.83676, |d|=1.11608, |fx|= 15.5281
iteration=3, x=3.24887, |d|=0.587887, |fx|= 3.43541
iteration=4, x=3.03626, |d|=0.212613, |fx|= 0.444338
iteration=5, x=3.00197, |d|=0.034286, |fx|= 0.0236834
Initial guess, x=2, |fx|=6, tolerance= 0.1
iteration=0, x=8, |d|=6, |fx|= 360
iteration=1, x=5.70701, |d|=2.29299, |fx|= 103.616
iteration=2, x=4.26553, |d|=1.44148, |fx|= 28.4241
iteration=3, x=3.44217, |d|=0.82336, |fx|= 6.76107
iteration=4, x=3.0821, |d|=0.360074, |fx|= 1.03287
iteration=5, x=3.00367, |d|=0.0784289, |fx|= 0.0440901
Initial guess, x=2, |fx|=6, tolerance= 0.1
iteration=0, x=8, |d|=6, |fx|= 360
iteration=1, x=-inf, |d|=inf, |fx|= inf
iteration=2, x=nan, |d|=nan, |fx|= nan
iteration=3, x=nan, |d|=nan, |fx|= nan
iteration=4, x=nan, |d|=nan, |fx|= nan
iteration=5, x=nan, |d|=nan, |fx|= nan
iteration=6, x=nan, |d|=nan, |fx|= nan
iteration=7, x=nan, |d|=nan, |fx|= nan
iteration=8, x=nan, |d|=nan, |fx|= nan
iteration=9, x=nan, |d|=nan, |fx|= nan
iteration=10, x=nan, |d|=nan, |fx|= nan
iteration=11, x=nan, |d|=nan, |fx|= nan
iteration=12, x=nan, |d|=nan, |fx|= nan
iteration=13, x=nan, |d|=nan, |fx|= nan
iteration=14, x=nan, |d|=nan, |fx|= nan
iteration=15, x=nan, |d|=nan, |fx|= nan
iteration=16, x=nan, |d|=nan, |fx|= nan
iteration=17, x=nan, |d|=nan, |fx|= nan
iteration=18, x=nan, |d|=nan, |fx|= nan
iteration=19, x=nan, |d|=nan, |fx|= nan

Initial guess, x=2, |fx|=6, tolerance= 1e-05
iteration=0, x=6.78505, |d|=4.78505, |fx|= 199.933
iteration=1, x=4.95284, |d|=1.83221, |fx|= 57.5764
iteration=2, x=3.83676, |d|=1.11608, |fx|= 15.5281
iteration=3, x=3.24887, |d|=0.587887, |fx|= 3.43541
iteration=4, x=3.03626, |d|=0.212613, |fx|= 0.444338
iteration=5, x=3.00197, |d|=0.034286, |fx|= 0.0236834
iteration=6, x=3.00007, |d|=0.00189932, |fx|= 0.000864342
iteration=7, x=3, |d|=6.94673e-05, |fx|= 3.0698e-05
iteration=8, x=3, |d|=2.4674e-06, |fx|= 1.08917e-06
Initial guess, x=2, |fx|=6, tolerance= 1e-05
iteration=0, x=8, |d|=6, |fx|= 360
iteration=1, x=5.70701, |d|=2.29299, |fx|= 103.616
iteration=2, x=4.26553, |d|=1.44148, |fx|= 28.4241
iteration=3, x=3.44217, |d|=0.82336, |fx|= 6.76107
iteration=4, x=3.0821, |d|=0.360074, |fx|= 1.03287
iteration=5, x=3.00367, |d|=0.0784289, |fx|= 0.0440901
iteration=6, x=3.00001, |d|=0.00365851, |fx|= 9.37918e-05
iteration=7, x=3, |d|=7.81591e-06, |fx|= 4.2842e-10
Initial guess, x=2, |fx|=6, tolerance= 1e-05
iteration=0, x=8, |d|=6, |fx|= 360
```

```
iteration=1, x=-inf, |d|=inf, |fx|= inf
iteration=2, x=nan, |d|=nan, |fx|= nan
iteration=3, x=nan, |d|=nan, |fx|= nan
iteration=4, x=nan, |d|=nan, |fx|= nan
iteration=5, x=nan, |d|=nan, |fx|= nan
iteration=6, x=nan, |d|=nan, |fx|= nan
iteration=7, x=nan, |d|=nan, |fx|= nan
iteration=8, x=nan, |d|=nan, |fx|= nan
iteration=9, x=nan, |d|=nan, |fx|= nan
iteration=10, x=nan, |d|=nan, |fx|= nan
iteration=11, x=nan, |d|=nan, |fx|= nan
iteration=12, x=nan, |d|=nan, |fx|= nan
iteration=13, x=nan, |d|=nan, |fx|= nan
iteration=14, x=nan, |d|=nan, |fx|= nan
iteration=15, x=nan, |d|=nan, |fx|= nan
iteration=16, x=nan, |d|=nan, |fx|= nan
iteration=17, x=nan, |d|=nan, |fx|= nan
iteration=18, x=nan, |d|=nan, |fx|= nan
iteration=19, x=nan, |d|=nan, |fx|= nan

Initial guess, x=2, |fx|=6, tolerance= 1e-09
iteration=0, x=6.78505, |d|=4.78505, |fx|= 199.933
iteration=1, x=4.95284, |d|=1.83221, |fx|= 57.5764
iteration=2, x=3.83676, |d|=1.11608, |fx|= 15.5281
iteration=3, x=3.24887, |d|=0.587887, |fx|= 3.43541
iteration=4, x=3.03626, |d|=0.212613, |fx|= 0.444338
iteration=5, x=3.00197, |d|=0.034286, |fx|= 0.0236834
iteration=6, x=3.00007, |d|=0.00189932, |fx|= 0.000864342
iteration=7, x=3, |d|=6.94673e-05, |fx|= 3.0698e-05
iteration=8, x=3, |d|=2.4674e-06, |fx|= 1.08917e-06
iteration=9, x=3, |d|=8.7544e-08, |fx|= 3.86425e-08
iteration=10, x=3, |d|=3.10596e-09, |fx|= 1.37099e-09
iteration=11, x=3, |d|=1.10196e-10, |fx|= 4.86384e-11
Initial guess, x=2, |fx|=6, tolerance= 1e-09
iteration=0, x=8, |d|=6, |fx|= 360
iteration=1, x=5.70701, |d|=2.29299, |fx|= 103.616
iteration=2, x=4.26553, |d|=1.44148, |fx|= 28.4241
iteration=3, x=3.44217, |d|=0.82336, |fx|= 6.76107
iteration=4, x=3.0821, |d|=0.360074, |fx|= 1.03287
iteration=5, x=3.00367, |d|=0.0784289, |fx|= 0.0440901
iteration=6, x=3.00001, |d|=0.00365851, |fx|= 9.37918e-05
iteration=7, x=3, |d|=7.81591e-06, |fx|= 4.2842e-10
iteration=8, x=3, |d|=3.57017e-11, |fx|= 0
Initial guess, x=2, |fx|=6, tolerance= 1e-09
iteration=0, x=8, |d|=6, |fx|= 360
iteration=1, x=-inf, |d|=inf, |fx|= inf
iteration=2, x=nan, |d|=nan, |fx|= nan
iteration=3, x=nan, |d|=nan, |fx|= nan
iteration=4, x=nan, |d|=nan, |fx|= nan
iteration=5, x=nan, |d|=nan, |fx|= nan
iteration=6, x=nan, |d|=nan, |fx|= nan
iteration=7, x=nan, |d|=nan, |fx|= nan
iteration=8, x=nan, |d|=nan, |fx|= nan
iteration=9, x=nan, |d|=nan, |fx|= nan
```

```
iteration=10, x=nan, |d|=nan, |fx|= nan
iteration=11, x=nan, |d|=nan, |fx|= nan
iteration=12, x=nan, |d|=nan, |fx|= nan
iteration=13, x=nan, |d|=nan, |fx|= nan
iteration=14, x=nan, |d|=nan, |fx|= nan
iteration=15, x=nan, |d|=nan, |fx|= nan
iteration=16, x=nan, |d|=nan, |fx|= nan
iteration=17, x=nan, |d|=nan, |fx|= nan
iteration=18, x=nan, |d|=nan, |fx|= nan
iteration=19, x=nan, |d|=nan, |fx|= nan
```

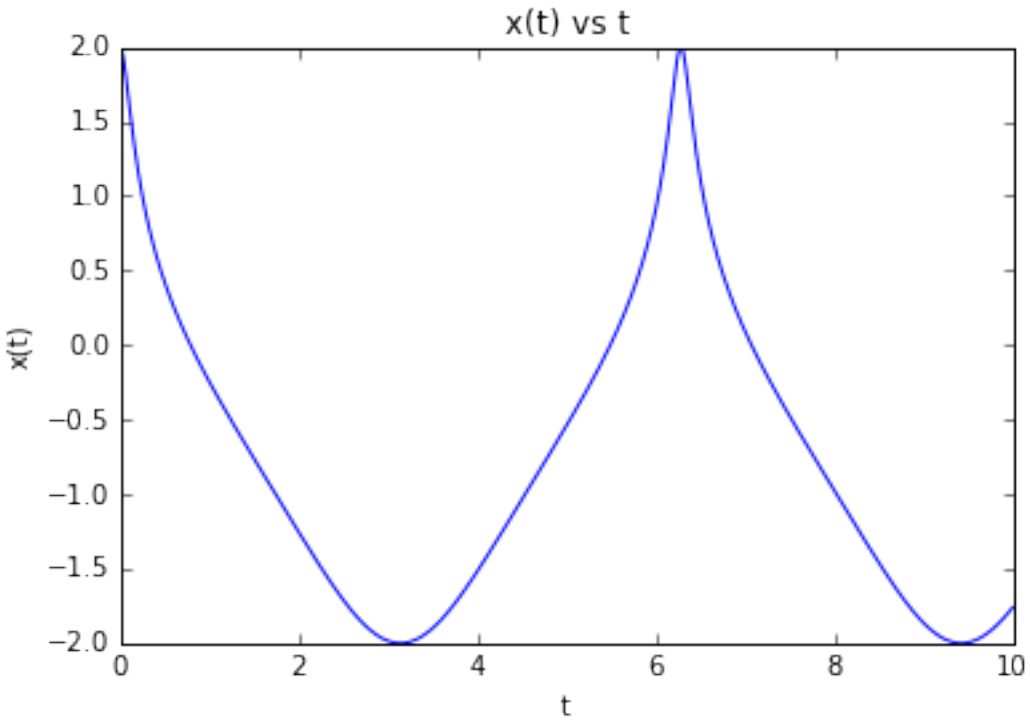
Based on the data, we can observe that as alpha gets smaller, the number of iterations it takes to get to a root with a specified tolerance decreases. The finite difference Newton method seems to behave similarly to the Newton method. However, at initial guess of x=2, and $\alpha = 2^{-50}$, the result becomes an inf. After the first iteration, the numerator of the finite difference approximation, f(x+α) − $f(x)$ becomes so small that it is below the machine epsilon and set to zero and the correction value becomes infinite, since it is calculated by f(x)/f'(x). This can also be explained by the upper bound of relative error for finite difference, $c1 * \alpha + c2/\alpha$ which goes to infite as alpha becomes extremly small and extremely large, as seen in project 1. Thus, the best α to select would be the one in the middle, which in this case is 2^{-26}. This conclusion can also be drawn from the fact that when α is 2^{-26}, finite difference Newton and Newton produce results that are approximately equal to each other.

1.1.3 Part 3: Kepler

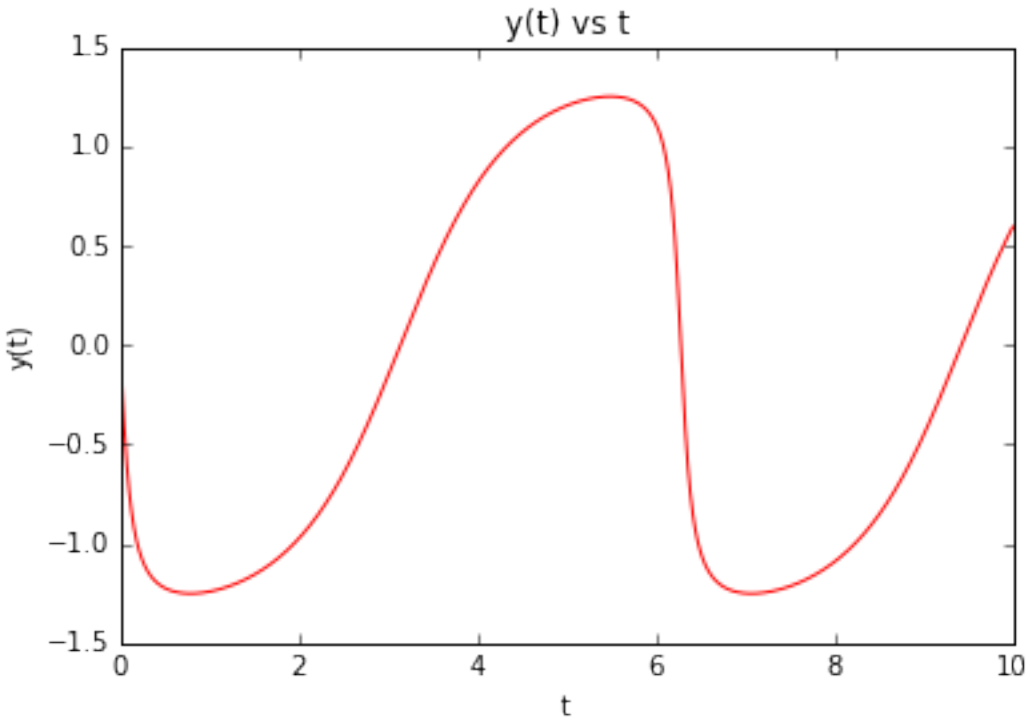
Part 3 was interesting because it had us find the roots of Kepler's equation, which is Esin(ω)-ω=t using our Newton method. First I subtracted t from both sides of the equation to determine the residual function $f(\omega) = Esin(\omega) - \omega - t = 0$, where E=$sqrt(1 - b^2/a^2)$ where a=2.0 and b=1.25. Then I determined that the derivative was Ecos(ω)-1. Then I created a matrix t using linspace with 10001 points in between 0 and 10. Afterward, I created two more matrices x and y, both with 10001 rows and 1 column, to store the coordinates of r(ω), the radial position of an object.I initialized ω to 0. Using Newton, the roots were calculated with an initial guess of the root calculated previously, varying t, 6 maximum iterations, and a tolerance of 10^{-5}. For each root determined by Newton, r(ω) was calculated using the formula $ab/sqrt((bcos(\omega))^2 + (asin(\omega))^2)$. Then, x was calculated using rcos(ω) and y was calculated using rsin(ω).

The following is code for the graphs of x(t) vs t, y(t) vs t, and y(t) vs x(t):

```
In [5]: %pylab inline
        t = loadtxt('t.txt')
        x=loadtxt('x(t).txt');
        y=loadtxt('y(t).txt');
        plot(t, x, 'b-')
        xlabel("t")
        ylabel("x(t)")
        title("x(t) vs t")

Populating the interactive namespace from numpy and matplotlib

Out[5]: <matplotlib.text.Text at 0x106933ad0>
```

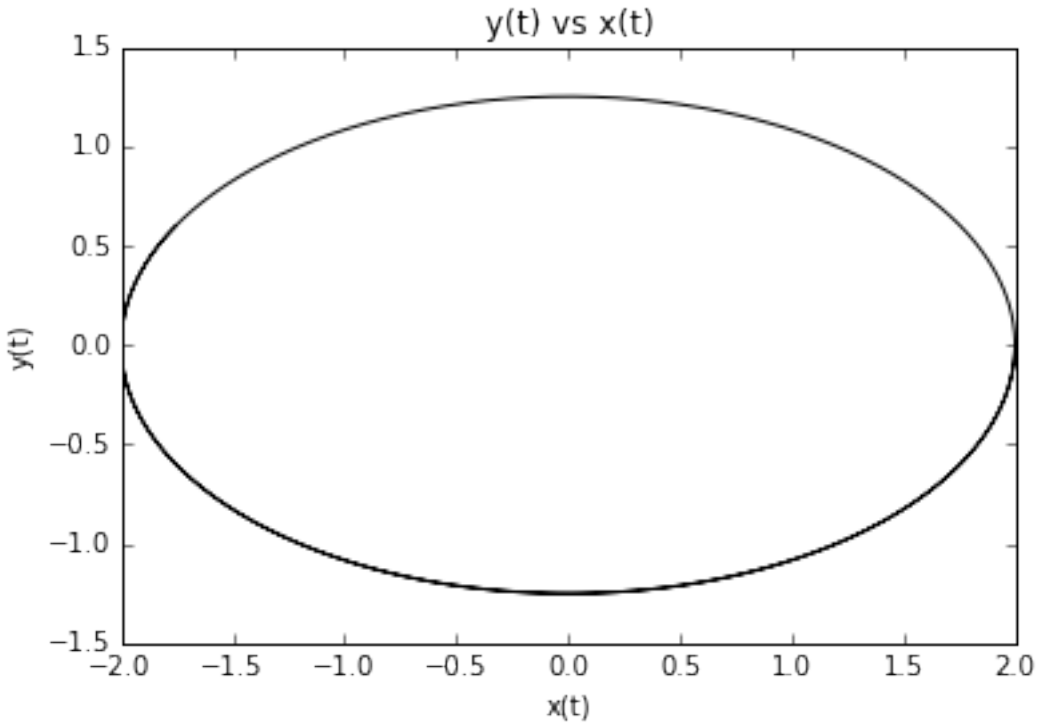



```
In [6]: plot(t, y, 'r-', label='$y(t)$')
        xlabel("t")
        ylabel("y(t)")
        title("y(t) vs t")

Out[6]: <matplotlib.text.Text at 0x106a5c2d0>
```




```
In [7]: plot(x,y ,color="black")
        xlabel("x(t)")
        ylabel("y(t)")
        title("y(t) vs x(t)")
```

```
Out[7]: <matplotlib.text.Text at 0x106afb690>
```



The final graph makes sense because we expected some sort of ellipse given that Kepler's equation is used to . A represents the radius across the x-axis and b represents the radius across the y-axis. The x(t) vs t graph fluctuates between 2 and -2, but is not a perfect cosine graph because the radius is stretched out. The y(t) vs t graph fluctuates between 1.25 and -1.25, but is also not a perefect sin graph. Thus, when x(t) and y(t) are combined, an ellipse is formed.

1.1.4 Code

newton.cpp

```
//
//  newton.cpp
//  Newton
//
//  Created by Arslan Memon on 9/26/15.
//  Copyright (c) 2015 Arslan. All rights reserved.
//


#include <stdlib.h>
#include <stdio.h>
#include <iostream>
#include <math.h>
#include "fcn.hpp"

double newton(Fcn& f, Fcn& df, double x, int maxit,
              double tol, bool show_iterates)//newton root finding
{
```

```
    if (maxit<1)// if iterartions is <1, then just return x
    {
        std::cerr<<"not enough iterations, returning input value"<<std::endl;
        return x;
    }
    if (tol<1e-15)//reset tolerance if tolerance is too low
    {
        std::cerr<<"tol<1e-15, resetting to to 1e-15"<<std::endl;
        tol=1e-15;

    }
    double fx=f(x);//calculate f(x)
    if (show_iterates)// print initial guess and initial evaluation if show iterates
    {
        std::cout<<"Initial guess x="<<x<<", |fx|="<<fabs(fx)<<", tolerance= "<<tol<<std::endl;

    }
    for (int n=0;n<maxit;n++)//loop to find root
    {
        double fp=df(x);// calculate f'

        if (fabs(fp)<1e-15)// if the absolute value of f'< tol, return x because that means the slope is
        {
            std::cout<<"small derivative";
            return x;
        }
        double d=fx/fp;// d is the correction amount that need to be subtracted from x
        x=x-d;
        fx=f(x);//new f(x)
        if( show_iterates)//show detail for iteration
        {

            std::cout<<"iteration="<<n<<", x="<<x<<", |d|="<<fabs(d)<<", |fx|= "<<fabs(fx)<<std::endl;
        }
        if (fabs(d)<tol)//if correction < tol, then  return x
        {
            return x;

        }
    }
    return x;

}
```

test_newton.cpp

```
//
//  newton_test.cpp
//  Newton
//
//  Created by Arslan Memon on 9/26/15.
//  Copyright (c) 2015 Arslan. All rights reserved.
//
```

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

using namespace std;
double newton(Fcn& f, Fcn& df, double x, int maxit,
              double tol, bool show_iterates);


//f=x(x-3)(x+1)=x^3-2x^2-3x
class f:public Fcn{
public:
    double operator()(double x) {
        return (((x-2)*x-3)*x);//nested multiplication using horner's method, returns x evaluated by fu

    }
};

// f'=3x^2-4x-3
class df:public Fcn{
public:
    double operator()(double x) {
        return (((3)*x-4)*x-3);//nested multiplication using horner's method, returns f' evaluated at x
    }
};


int main(int argc, char* argv[]) {
    f function;// f object called function
    df derivative;// derivative of f object called derivative

   // use newtonm to solve for roots of function. Each newton method uses an initial guess of x=-2,1,2 a
    newton(function, derivative, -2, 15, 1e-1, true);
    newton(function, derivative, -2, 15, 1e-5, true);
    newton(function, derivative, -2, 15, 1e-9, true);
    newton(function, derivative, 1, 15, 1e-1, true);
    newton(function, derivative, 1, 15, 1e-5, true);
    newton(function, derivative, 1, 15, 1e-9, true);
    newton(function, derivative, 2, 15, 1e-1, true);
    newton(function, derivative, 2, 15, 1e-5, true);
    newton(function, derivative, 2, 15, 1e-9, true);

    return 0;

};
```

fd_newton.cpp

```
//
//  fd_newton.cpp
//  Newton
```

```
//
//  Created by Arslan Memon on 9/28/15.
//  Copyright (c) 2015 Arslan. All rights reserved.
//

#include <stdlib.h>
#include <stdio.h>
#include <iostream>
#include <math.h>
#include "fcn.hpp"

double fd_newton(Fcn& f, double x, int maxit, double tol,
                 double alpha, bool show_iterates)//same as newton except use alpha to calculate f' usi
//finite difference, so no f' needed as parameter
{


    if (maxit<1)
    {
        std::cerr<<"not enough iterations, returning input value"<<std::endl;
        return x;
    }
    if (tol<1e-15)
    {
        std::cerr<<"tol<1e-15, resetting to to 1e-15"<<std::endl;
        tol=1e-15;

    }
    double fx=f(x);
    if (show_iterates)
    {
        std::cout<<"Initial guess, x="<<x<<", |fx|="<<fabs(fx)<<", tolerance= "<<tol<<std::endl;

    }
    for (int n=0;n<maxit;n++)
    {
        double sdpf=(f(x+alpha)-f(x))/alpha;//f'(x) is approximately (f(x+alpha)-f(x))/alpha
        double d=fx/sdpf;
        x=x-d;
        fx=f(x);
        if( show_iterates)
        {

            std::cout<<"iteration="<<n<<", x="<<x<<", |d|="<<fabs(d)<<", |fx|= "<<fabs(fx)<<std::endl;
        }
        if (fabs(d)<tol)
        {
            return x;

        }
    }
    return x;
```

```
}
```

test_fd_newton.cpp

```
//
//  test_fd_newton.cpp
//  Newton
//
//  Created by Arslan Memon on 9/28/15.
//  Copyright (c) 2015 Arslan. All rights reserved.
//


#include <stdio.h>
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
#include <math.h>
#include "fcn.hpp"

using namespace std;

double fd_newton(Fcn& f, double x, int maxit, double tol, double alpha, bool show_iterates);



class f:public Fcn{
public:
    double operator()(double x) {//same f as in newton.cpp
        return (((x-2)*x-3)*x);

    }
};

int main(int argc, char* argv[]) {
    f function;
    //same evaluations as newton.cpp, but each evaluation is done three times for different alphas of 2
    fd_newton(function,-2, 20, 1e-1,pow(2, -4),true);
    fd_newton(function,-2, 20, 1e-1,pow(2, -26),true);
    fd_newton(function,-2, 20, 1e-1,pow(2, -50),true);
    cout<<endl;
    fd_newton(function,-2, 20, 1e-5,pow(2, -4),true);
    fd_newton(function,-2, 20, 1e-5,pow(2, -26),true);
    fd_newton(function,-2, 20, 1e-5,pow(2, -50),true);
    cout<<endl;
    fd_newton(function,-2, 20, 1e-9, pow(2, -4),true);
    fd_newton(function,-2, 20, 1e-9, pow(2, -26), true);
    fd_newton(function,-2, 20, 1e-9, pow(2, -50), true);
    cout<<endl;
    fd_newton(function, 1, 20, 1e-1, pow(2,-4), true);
    fd_newton(function, 1, 20, 1e-1, pow(2,-26), true);
    fd_newton(function, 1, 20, 1e-1, pow(2,-50), true);
    cout<<endl;
```

```
    fd_newton(function, 1, 20, 1e-5, pow(2, -4), true);
    fd_newton(function, 1, 20, 1e-5, pow(2, -26), true);
    fd_newton(function, 1, 20, 1e-5, pow(2, -50), true);
    cout<<endl;
    fd_newton(function, 1, 20, 1e-9, pow(2, -4), true);
    fd_newton(function, 1, 20, 1e-9, pow(2, -26), true);
    fd_newton(function, 1, 20, 1e-9, pow(2, -50), true);
    cout<<endl;
    fd_newton(function, 2, 20, 1e-1, pow(2, -4), true);
    fd_newton(function, 2, 20, 1e-1, pow(2, -26), true);
    fd_newton(function, 2, 20, 1e-1, pow(2, -50), true);
    cout<<endl;
    fd_newton(function, 2, 20, 1e-5, pow(2, -4), true);
    fd_newton(function, 2, 20, 1e-5, pow(2, -26), true);
    fd_newton(function, 2, 20, 1e-5, pow(2, -50), true);
    cout<<endl;
    fd_newton(function, 2, 20, 1e-9,pow(2,-4), true);
    fd_newton(function, 2, 20, 1e-9,pow(2,-26), true);
    fd_newton(function, 2, 20, 1e-9,pow(2,-50), true);
    cout<<endl;
    return 0;

};
```

kepler.cpp

```
//
//  kepler.cpp
//  Newton
//
//  Created by Arslan Memon on 10/4/15.
//  Copyright (c) 2015 Arslan. All rights reserved.
//

#include <stdio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
#include <math.h>
#include "fcn.hpp"
#include "matrix.hpp"
using namespace std;
double newton(Fcn& f, Fcn& df, double x, int maxit,
              double tol, bool show_iterates);
// kepler equation is sqrt(1-b^2/a^2)sin(w)-w=t, thus a non-linear root finding residual function, f(w)
class Kepler:public Fcn{
public:
    double t;//modifiable parameter for time
    double operator()(double w) {
        return sqrt(1-(pow(1.25,2)/pow(2.0, 2)))*sin(w)-w-t;//a=2.0, b=1.25, solve the kepler equation a

   }
```

```
};
//derivative for kepler equation
class dKepler:public Fcn{
public:
    double operator()(double w) {
        return sqrt(1-(pow(1.25,2)/pow(2.0, 2)))*cos(w)-1;
    }
};

int main(int argc, char* argv[]) {
    Kepler k;//object for kepler function
    dKepler dk;// object for the derivative of kepler function
    Matrix t=Linspace(0, 10,10001);// matrix for time using linspace to get 10001 evenly spaced points
    double w=0.0;// initial guess for w is 0
    double ab=1.25*2.0;//a*b
    //Matrices to store cartesian coordinates of r(w)
    Matrix x (10001);
    Matrix y(10001);
    for (int i=0;i<10001;i++)
    {

        k.t=t(i);// set t
        w= newton(k, dk, w,6 , 1e-5, false);//use newton to get approxiimation of root with 6 iterations
        double r=ab/sqrt(pow(1.25*cos(w),2)+pow(2.0*sin(w), 2));//calculate r(w)
        x(i)=r*cos(w);//x=rcos(w)
        y(i)=r*sin(w);//y=rsin(w)

    }
   //store files to disk
     t.Write("t.txt");
    x.Write("x(t).txt");
    y.Write("y(t).txt");
};
```

Makefile

```
#########################################################################
#  Makefile for project 2
#
#  Arslan Memon
#########################################################################

# makefile targets
all: newton.exe fd_newton.exe kepler.exe

newton.exe : newton.cpp test_newton.cpp
    g++ $^ -o $@

fd_newton.exe : fd_newton.cpp test_fd_newton.cpp
    g++ $^ -o $@

kepler.exe: kepler.cpp newton.cpp  matrix.cpp
    g++ $^ -o $@
clean :
```

```
    \rm -f *.o *.txt

realclean : clean
    \rm -f *.exe *~


####### End of Makefile #######
```

In [ ]: