Project 2

Tran, Van

10/8/15

Part 1:

In the first part of the project we create the code for Newton's method to estimate the root of a function. The algorithm came from the book, and pseudocode covered in class. As we change the initial guess, the root at which Newton's method converges changes. This is because as the initial guess changes, the f(x) and f'(x) change, and if this change is significant enough, the Newton's method will find a different root. As we change the tolerance, the more iterations of Newton's method is allowed, making the estimation have less error.

I started out by translating the book's pseudocode and the instructor's format of bisection to create the code for newton.cpp which relies on two functors (objects who's constructors act like functions) which represent f(x) and f'(x). I inputted the values as instructed by the lab manual in a main method defined in test_newton.cpp and analyzed them to make the observations above.

Part 2:

In the second part of the project we created the code for the Secant method, which is basically Newton's method, but instead of requiring a derivative, the secant method uses the definition of a derivative to estimate the root. F'(x) = lim as h approaches 0 (f(x + h) – f(x))/ h. The secant method uses this definition, but does not require h to approach 0, therefore giving an estimation of the derivative. When tested, the secant method required many more iterations than the newton method to converge to the tolerance given. Also there were cases where the secant method failed altogether and returned nans. Although the Newton's method seems superior to the secant method (secant being superlinear while newton being quadratic).

This comparison is misleading. The computation of the derivative can be expensive (in this case the derivative was done by hand and then put on the computer) and difficult to find. Newton also requires the evaluation of both f(x) and f'(x) at each step whereas secant only requires the evaluation of f(x). So in many practical applications secant can be faster and more practical than newton's method.

This part was very much like part 1. fd_newton.cpp was nearly identical (6 line difference on github), and the only change was to get rid of the f'(x) functor and instead estimate it. The test file was also nearly identical and the comparisons between the output of fd_newton (secant) and newton were used to make the observations above.

Part 3:

In the third part of the project we created code to estimate the roots of Kepler's equation at different time intervals. This part of the project shows how powerful the newton method really is, and how computationally cost-effective the algorithm is. The hardest part was realizing that w was an angular quantity and to be represented on a Cartesian axis required it to be converted into its linear quantities (x, y). I got the expected ellipse to predict the trajectory of the orbit.

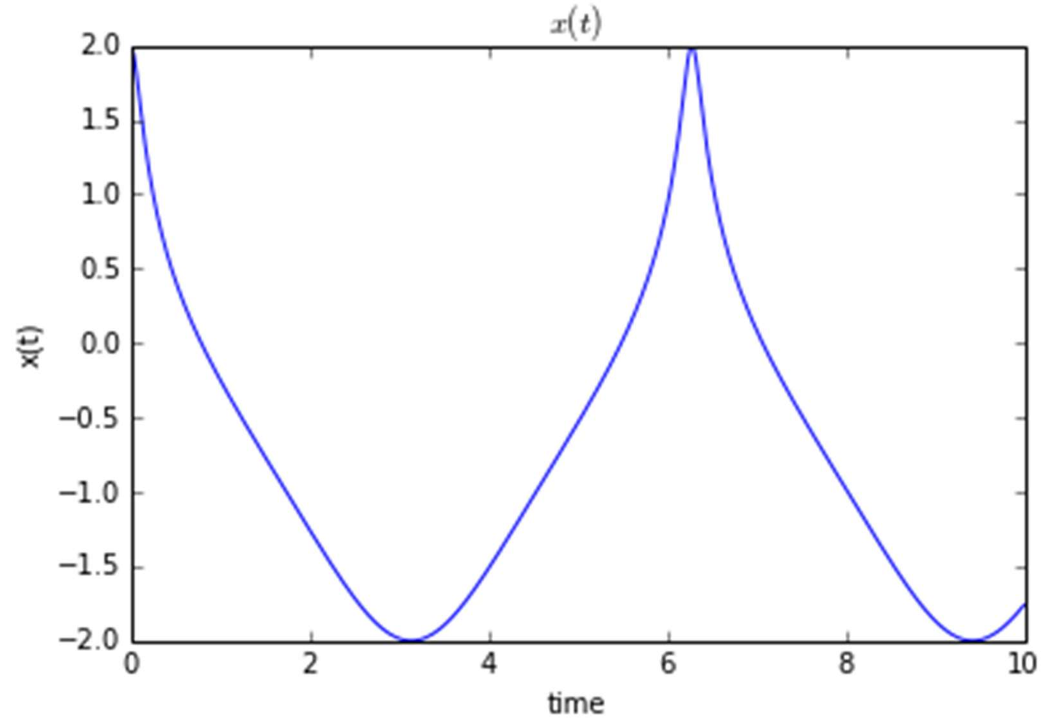
NEWTON OUTPUT (iterates set to 0 for readability)

```
INITIAL guess: -2

TOLERANCE: 1

The approximate root is: -1.411764705882353

TOLERANCE: 0.0001

The approximate root is: -1.000000000000004

TOLERANCE: 1e-08

The approximate root is: -1

INITIAL guess: 1

TOLERANCE: 1

The approximate root is: 0

TOLERANCE: 0.0001

The approximate root is: 0

TOLERANCE: 1e-08

The approximate root is: 0

INITIAL guess: 2

TOLERANCE: 1

The approximate root is: 3.442168945858322

TOLERANCE: 0.0001

The approximate root is: 3.000000000035635

TOLERANCE: 1e-08

The approximate root is: 3
```

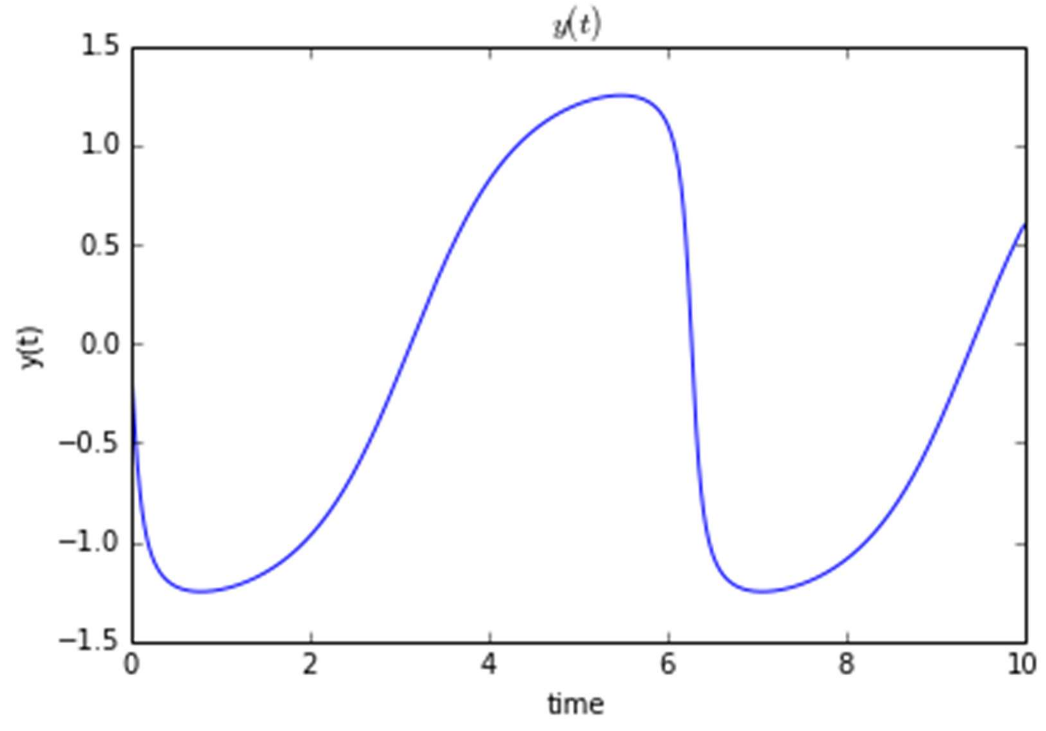
FD NEWTON OUTPUT (show iterates set to 0 for readability)

```
ERROR answer holds no significance. exceeds tolerance .

The approximate root is: 4.263759707841365

The approximate root is: -nan

The approximate root is: -nan

TOLERANCE: 0.0001

ERROR answer holds no significance. exceeds tolerance .

The approximate root is: 3.0000081878921

The approximate root is: -nan

The approximate root is: -nan

TOLERANCE: 1e-08

ERROR answer holds no significance. exceeds tolerance .

The approximate root is: 3.000000000994267

The approximate root is: -nan

The approximate root is: -nan
```

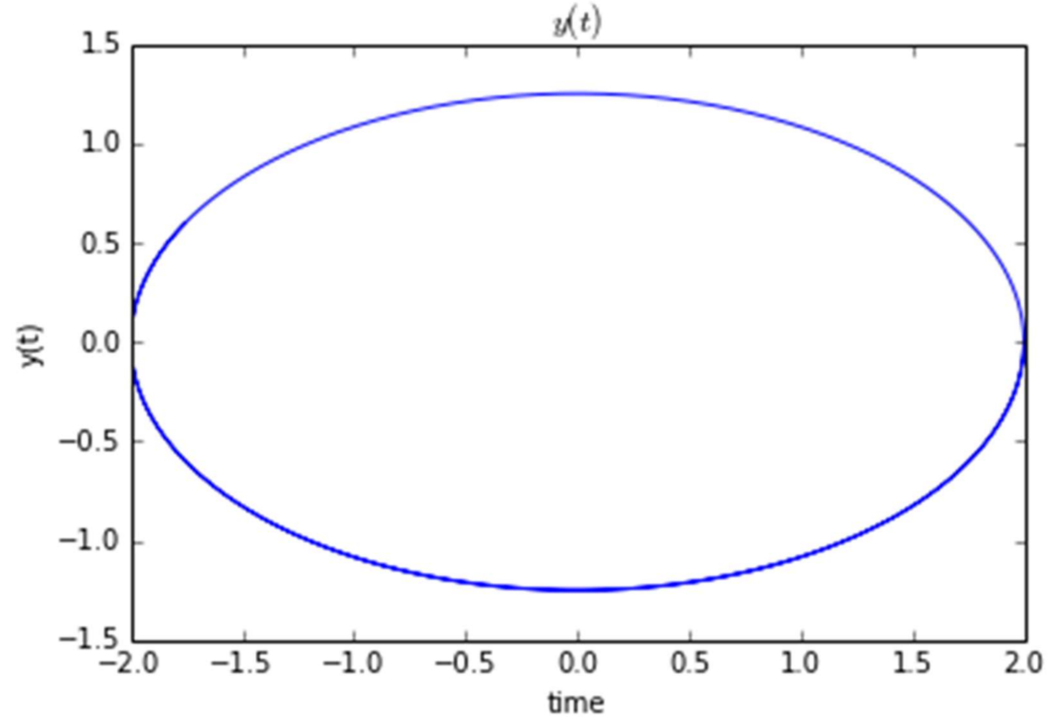
GRAPHS:



$x(t)$
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
x(t)
0
2
4
6
8
10
time



$y(t)$
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
y(t)
0
2
4
6
8
10
time


$y(t)$
y(t)
time
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0