Project 1 Report

Bisection Method: `function [l,r,p] = BISECTION(f,l,r,tlob,bmax)`

Description:

The Bisection method approximates a root of a function f within the interval [l,r]. In order for the method to work it is necessary to have f(l)*f(r) < 0 and if this condition is satisfied the function returns a new interval [l,r] as well as a point p which is close to the actual root. In our project, we don't allow the Bisection method to perform more than one iterations at a given time (bmax is always 1). Instead we supply the result of the Bisection Method to another routine that performs Newton's Method for approximating a root.

This function accepts as input:

- f: The function on which we wish to perform the Bisection Method
- l: The left endpoint of the interval in which we are searching for a root
- r: The right endpoint of the interval in which we are searching for a root
- tlob: The tolerance that determines whether we are close enough to the solution

  bmax: The maximum number of iterations the Bisection method must perform. In our case it is always 1.

Calling Syntax:

To call the function we must supply a value for its parameters. So for example:

```
p = BISECTION(@(x) x^3-25,2,3,0.000001,bmax)
```

will return p=2.5 since only one iteration is performed.

Newton's Method: `function [p,flag] = NEWTON(F,F_prime,p0,TOL,nmaximum)`

Description:

Newton's method attempts to find a root of F close to the point p0. This method converges quadratically therefore for most cases no more than 15 iterations are required in order to approximate the root in six decimal places. The function returns a point p and a flag that signifies whether the method converged.

This function accepts as input:

- F: The function on which we wish to perform Newton's Method
- F_prime: The derivative of F
- p0: An initial approximation close to the root of F
- TOL: The tolerance that determines whether we are close enough to the solution
- nmaximum: The maximum number of iterations the method must perform

Calling Syntax:

To call the function we must supply a value for its parameters. So for example:

```
p = NEWTON(@(x) x^3-25,3x^2,2.5,0.000001,bmax)
```

will return p=2.92401 which is an approximation to the cubic root of 25.

Hybrid Method Combining Bisection and Newton's Method: `function v= HYBRIDBISECTIONNEWTON(f,f_prime,a,b,~,toln,nmax)`

Description:

This is the implementation of Algorithm 1 as provided in the spec.

This function accepts as input:

- f: The function on which we wish to perform the Method
- f_prime: The derivative of f

- a: The right endpoint of the interval. This is supplied to the Bisection Method.
- b: The left endpoint of the interval. This is supplied to the Bisection Method.
- ~: Stands for tolb in the specification. It is not named as a parameter since it is always set to the absolute value of b-a.
- toln: The tolerance supplied to Newton's Method and determines whether we are close enough to the solution
- nmax: The maximum number of iterations Newton's Method can perform when searching for a solution

Calling Syntax:

To call the function we must supply a value for its parameters. So for example:

```
v = HYBRIDBISECTIONNEWTON(@(x) x^3-25,3x^2, 0, 10, ~, 0. 000001,15);
```

will return p=2.92401 which is very close approximation to the cubic root of 25.

Kepler Solver: `function y= KEPLERSOLVER(T,e,t)`

Description:

This is a very simple function that solves Kepler's equation (Equation 4 in the specification) for fixed period, eccentricity and time. It accomplishes this task by calling the Hybrid Method and supplying all the necessary parameters. Discussion on how the endpoints as well as the tolerances are chosen is made later in the report.

This function accepts as input:

- T: The period
- e: The eccentricity
- t: A time t between 0.01 and 0.99

Calling Syntax:

To call the function we must supply a value for its parameters. So for example:

`z=KEPLERSOLVER(1,0.25,0.93)` will return in z= 5.7072.

Testing Kepler Solver: `(script) KEPLER_INIT`

Description:

This is a script that initializes the Kepler solver and tests it against different values of the parameters T,e, and t. This is accomplished by setting T=1, e=0.25 and letting t vary between 0.01 and 0.99 incremented by 0.02 after each iteration. For each t we call KEPLERSOVLER supplying the values of the parameters and storing the result in an array F. At the last step, after t has taken all of its values we plot E(t) against t.

This script accepts no input therefore to call it we type in the command line:

`KEPLER_INIT` and we produce the plot.

Choosing Endpoints:

The Interval that I chose for the function $f(E)=\frac{2\pi t}{T}-E+e\sin\left(\frac{2\pi t}{T}\right)$ is:

$I=\{t:E_a<t<E_b\}$ for $t<\frac{T}{2}$

and

$I=\{t:E_b<t<E_a\}$ for $t>\frac{T}{2}$

Where:

$E_a=\frac{2\pi t}{T}, E_b=\pi$

The reason why I chose these endpoints is because they satisfy four basic properties:

1) $f(E_a)*f(E_b)<0 \forall t: 0<t<1$

2) There exists only one root of f between the two endpoints

3) Between these endpoints the function is not very flat (first derivative does not approache 0), thus the root has multiplicity 1 and this also allows a fast convergence when applying Newton's Method.

4) Between these points Newton's method always converges to a root

Proof of Property 1: $f(E_a)*f(E_b)<0 \forall t: 0<t<1$

$$f(E_a)=\frac{2\pi t}{T}-\frac{2\pi t}{T}+e\sin\left(\frac{2\pi t}{T}\right)=e\sin\left(\frac{2\pi t}{T}\right)$$

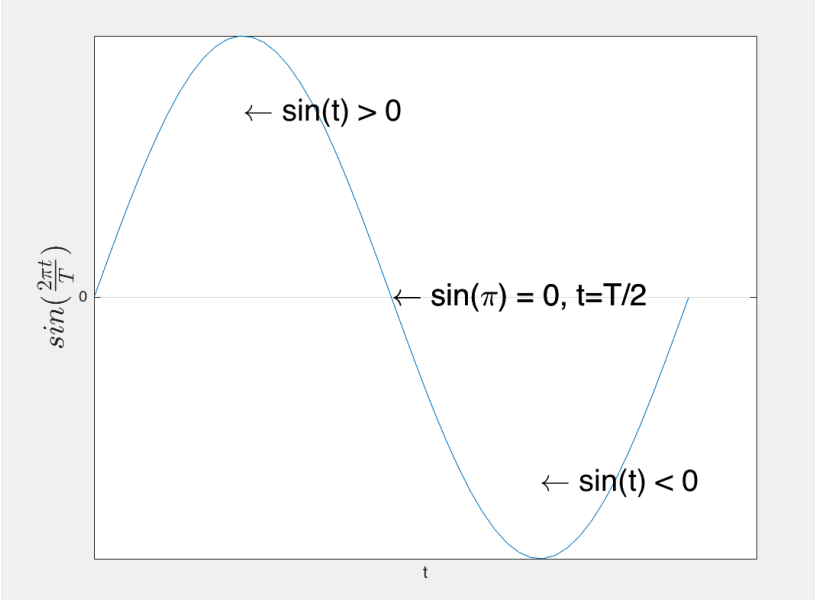
$$f(E_b)=\frac{2\pi t}{T}-\pi+0=\frac{2\pi t}{T}-\pi$$

$$f(E_a)*f(E_b)=e\sin\left(\frac{2\pi t}{T}\right)*\left[\frac{2\pi t}{T}-\pi\right]$$

Now notice that if : $\frac{2\pi t}{T}-\pi<0 \Leftrightarrow \frac{2t}{T}<0 \Leftrightarrow t<\frac{T}{2}$

But for $t<\frac{T}{2}$ we have then $\sin\left(\frac{2\pi t}{T}\right)>0$

This is because $\sin\left(\frac{2\pi t}{T}\right)$ has period $P=\frac{2\pi}{\frac{2\pi}{\mathrm{T}}}=\mathrm{T}$



Similarly if we have $t > \frac{T}{2}$ then $\sin\left(\frac{2\pi t}{T}\right) < 0$.

Proof of Property 2:

Since Property 1 holds then we know by the Intermediate Value Theorem that there exists at least one root in the interval *I*. Now we have to prove that this root is unique. If we consider the first derivative we see that $f'(E) = -1 - e\cos(E)$. But:

$$-1 \le \cos(E) \le 1 \Leftrightarrow e \ge -e\cos(E) \ge -e \Leftrightarrow -e \le -e\cos(E) \le e$$
$$\Leftrightarrow -1 - e \le -1 - e\cos(E) \le -1 + e \Leftrightarrow -1.25 \le -1 - e\cos(E) \le -0.75$$
$$\Leftrightarrow -1.25 \le f'(E) < -0.75$$

Therefore we conclude that the function is monotonically decreasing. Therefore, since there exists at least one root $r \in I$ this root has to be unique.

Proof of Property 3:

Since the derivative of f is bounded $-1.25 \leq f'(E) < -0.75$ it never approaches 0 so the function is never too flat within the interval *I*. Also since the derivative is bounded, we know that for the root $r \in I$ $f'(r)$ does not approach zero. Therefore the multiplicity of the root is 1.

Proof of Property 4:

Use Theorem 2.6 of Chapter 2.3 in the textbook.

$f'(E) = -1 - e\cos(E)$
$f''(E) = -e\sin(E)$

Therefore since both the first and second derivatives are continuous we have that $f \in C^2$ in the interval *I*. Also from the previous properties there exists a root $p \in I$ and the derivative of the root is not zero therefore we conclude that there exists δ>0 such that Newton's Method generates a sequence $\left\{ p_n \right\}_0^\infty$ converging to the root p for any initial approximation $p_0 \in [p-\delta, p+\delta]$.

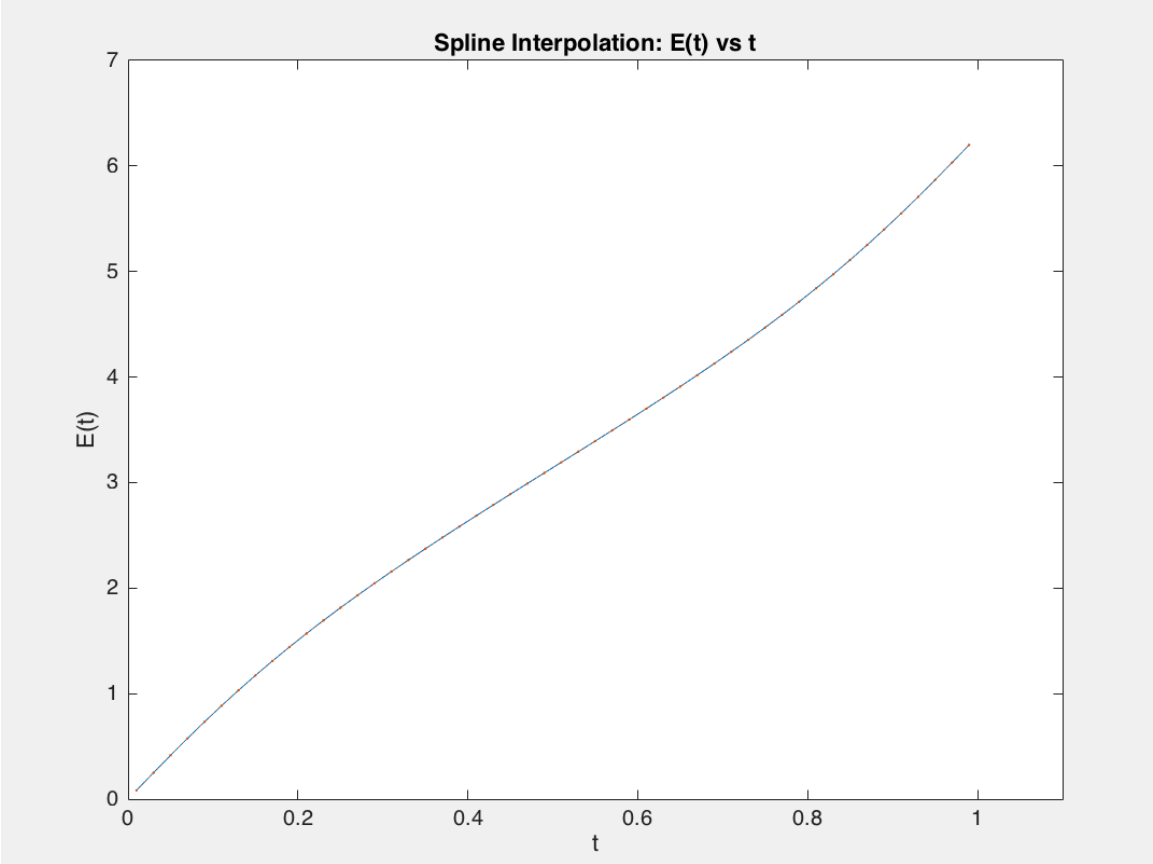
Choice for Tolerance:

Since we need to find E within six digits of accuracy I set the tolerance for Newton's Method to be $t = 10^{-6}$. The tolerance of the Bisection Method is automatically set after each iteration so we don't have to supply a value.

Choice for Number of Iterations:

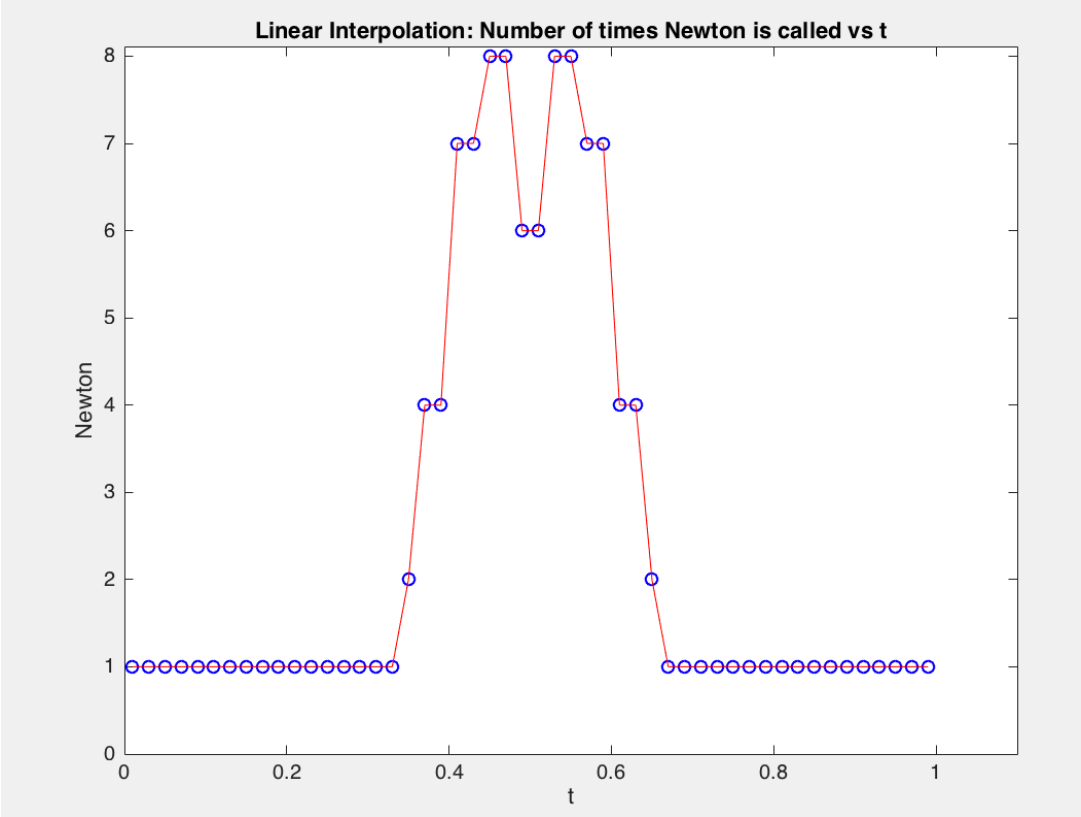
Since Newton's Method converges qudratically , a good tolerance would be between 10 and 20 iterations. I choose the number of maximum iterations to be 15.

Graph of E(t) vs t



The KEPLER_INIT.m script produces this graph.

Graph of Number of times Newton's Method was called vs t



The PLOTNEWTON.m file produces this Graph. To run it write PLOTNEWTON in the command line and this graph will be produced.