Math 151A: Applied Numerical Methods

Winter 2016, Lecture 2

Project 1: Newton's Method and Bisection Method Applied to Kepler's equation

Theodore Nguyen

704-156-701

Documentation:

Note that a more detailed, step-by-step explanation to each function is found in the comments of each function file. Inside this submission are the following 5 files:

- BISECTION.m – This file is the implementation of the Bisection Method, taken nearly verbatim from the book algorithm. Takes a function **f**, two endpoints **a** and **b** for an interval [a,b], the tolerance **TOL**, and the maximum number of iterations **imax**. The function returns two endpoints **A**, **B**, which were the last two endpoints just before the satisfied p was found or after all the iterations had been exhausted, as well as **p**, the approximate solution of f(x) = 0

  function [A, B, p] = BISECTION(f, a, b, TOL, imax)

- NEWTONS.m – This file is the implementation of Newton's Method, taken more or less from the book's algorithm pseudocode. The function as arguments a function **f**, its derivative **df**, an initial guess **xo**, the tolerance **TOL**, and the maximum number of iterations **imax**. The function returns **p**, the approximate solution to f(x) = 0, and flag, where flag is 1 if Newton's method converged and 0 otherwise. Newton's method is undefined where the derivative at a point is equal to zero (since you cannot divide by zero), so there is an if statement satisfying that limitation as one of the only additions to the book algorithm.

  function[p, flag] = NEWTONS(f, df, xo, TOL, imax)

- HYBRID.m – This file is the implementation of Algorithm 1 in the specification. The code is nearly an exact replica in the correct syntax of the pseudocode given. The function takes a function **f**, its derivative **df**, two endpoints **a** and **b** for an interval [a,b] for Bisection method, the tolerance for the Bisection Method **tolb** (whose passed value is actually never used at all), the tolerance for Newton's Method **toln**, and the maximum number of iterations for each call to Newton's Method **nmax**. The function outputs/returns a value **p** which is an approximation to satisfy f(p) = 0. For more detailed information on the algorithm, please read the pseudocode in the specification or the comments in HYBRID.m

  function[p] = HYBRID(f, df, a, b, tolb, toln, nmax)

- KEPLER.m – This file is the "Kepler Equation Solver." Takes 4 arguments – **T, e**, and **t** which are all coefficients in Kepler's equation; **accuracy**, which is essentially the desired tolerance you wish to pass to **NEWTON**. The function returns one value – **soln** – which is just the solution E' to the Kepler Equation such that f(E') = 0. The function defines the specific Kepler function based off the coefficients and computes the maximum number of iterations and interval endpoints needed to pass onto **HYBRID** – which it calls at the end.

  function[soln] = KEPLER(T, e, t, accuracy)

- testscript.m – This file is a script. It contains the code for testing and evaluating, using all the previously created functions to compute a result. Essentially, this file initializes the fixed values for Kepler's equation in our case, namely **T**, **e**, and **t**, where they take on values 1, 0.25, and [0.01, 0.03, … 0.97, 0.99] respectively. It also initializes the desired accuracy to 6 digits as the form of 0.000001. Then, it calls **KEPLER** to compute the solutions E of the Kepler equation, and then stores them into an array **E** analogous to **t**. To execute this, just have this be in your working directory as well as the other 4 files. Type **testscript** at the prompt to obtain the result. Go into the code and change **T, e, t**, or accuracy as you wish to generate a different result.
- **ALL CODE** is run by using MatLab/Octave as normal. Call the above functions if you want to execute (or type in the script name) based off of those arguments.

- report.pdf – contains the documentation, analysis, and plots you are reading right now

As a sort of a streamlined summary, NEWTONS and BISECTION are standalone and work on any function, given all the arguments and conditions are fulfilled. Both are both called by HYBRID to do the same, but guarantee a mixture of convergence and speed. The latter is called by KEPLER, which specifically solves an equation f(E) as given in the spec given arguments, calling HYBRID. KEPLER is called on testscript.m, which initializes it based off the values desired in the spec. Run **testscript** if you just want to get E(t) for the 50 values of t = [0.01 0.03 … 0.97 0.99], which are stored in the array labeled **E**

Choice of Parameters

Endpoints:

We do a graphical analysis of

$$f(E) = \frac{2\pi t}{T} - E + e\sin(E).$$

To determine the ideal endpoints for the Newton-Bisection method.

For a fixed t, e, and T, when graphically analyzed, is an overall decreasing function following a path similar to linear functions of the form f(x) = – (x – a) + b, except that the graph is oscillating.

Upon further graphical analysis, we can see that when removing certain terms, the graph follows a certain trend. Keeping only the sine term gives the function just the properties of a sine graph oscillating along the E-axis, where e is actually the amplitude of the oscillations. The term $2\pi t/T$ shifts the latter graph vertically, proportional to that term's value. The addition of –E "projects" the latter graph onto that of f(x) = –x.

Putting it all together, e determines the amplitude of the oscillations – as e increases, the amplitude increases – the negative E term in the middle projects the trigonometric function onto its polynomial (in this case, linear) equivalent, and as t increases, the x-intercept(s) of the graph increases, shifting the graph to the right.

Since we are talking about a fixed case here: T = 1, e = 0.25, and t = 0.01, 0.03, …, 0.97, 0.99, it may be useful to note that for these fixed constant cases, the solution to f(E) = 0 ranges from E = 0.0837432 for t

= 0.01 to E = 6.19944 for t = 0.99, while the rest of the solutions lie within that range (0.0837432, 6.19944) approximately.

Upon doing trial and error, we discover that the value of the first term, the expression $2\pi t/T$ is "generally in the neighborhood" of the actual zero of the function. For example, for t = 0.01, the expression is equal to 0.062831853, while the actual root is E = 0.0837432; when t = 0.99, the expression is equal to 6.220353454, while the actual root is E = 6.19944 (these are performed with T = 1, e = 0.25). This finding fits with the previously mentioned idea that as t increases, the E-intercept(s) of the graph increases, shifting the graph to the right. The graph is centered with its root near zero for when t = 0.01, and is centered on zero when the expression equals zero.

"Generally in the neighborhood" is a completely heuristic term; for our purposes, we just want to find an interval such that the endpoints evaluated at the function are opposite signs. Therefore, we employ another final heuristic measure for the endpoints needed for bisection method: the endpoints will be plus and minus 1 of the expression's value $2\pi t/T$. For example, for t = 0.99, the value for a will be a = 6.2203533454 – 1, and b = 6.2203533454 + 1. This heuristic should provide two points with opposite signs, at least for fixed T = 1 and e = 0.25.

Tolerance:

Taken as an argument; not computed. Specification desires 6 digits of accuracy, so we let the variable **accuracy** = 0.000001**,** which doubles as ntol (not btol technically because it gets overwritten immediately).

Maximum Number of Iterations:

NEWTON is actually the only function that takes in this value – imax or nmax or N, whatever you want to call it. This is because we use the hybrid Algorithm 1 for our root searching: **BISECTION** is only called with one maximum iteration each time; this is because bisection is only being used as a back-up in case Newton's method diverges. This obviously makes sense, as Newton's method converges quadratically while the Bisection Method takes much longer to converge, even though it is a backup. From this, we can infer that if the Bisection Method converges, then Newton's method converges in our case, especially when Bisection is only called for one max iteration while Newton is called for nmax.

However, we still want the Bisection Method to converge; even though there is guaranteed convergence, it is only guaranteed if there are enough iterations to allow it to converge. Thus, nmax is determined solely by the properties of the Bisection Method.

We want to have an accuracy ϵ; the length of the Bisection Method interval after N iterations is $\frac{b_0-a_0}{2^N}$

Therefore, if we want to have that same accuracy, we must have $\frac{b_0-a_0}{2^N} \subseteq \epsilon$

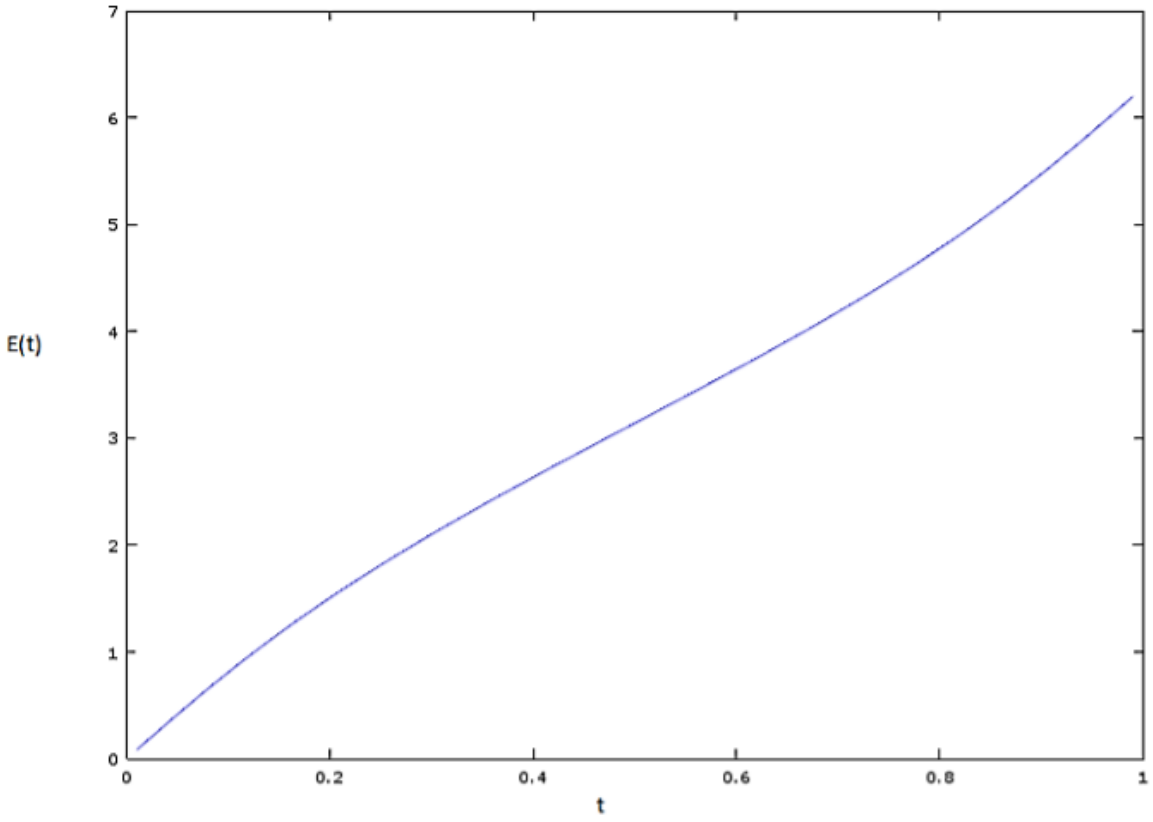
$$2^{-N}(b_0 - a_0) \leq \epsilon,$$

$$2^N \geq \frac{b_0 - a_0}{\epsilon},$$
$$N \ln 2 \geq \ln(b_0 - a_0) - \ln \epsilon,$$
$$N \geq \frac{\ln(b_0 - a_0) - \ln \epsilon}{2}.$$

Then, because that expression is the lower bound for which the Bisection Method achieves desirable accuracy, we take the ceiling of the expression to be the max number of iterations.

Plots of Data

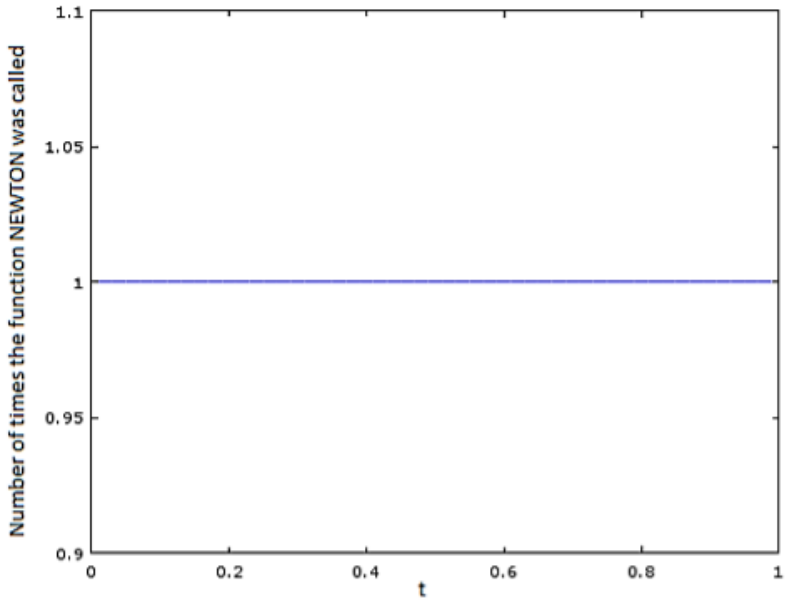
Plot of the curve E(t) against t, for t-values = 0.01, 0.03, … 0.97, 0.99:



We can see that E(t) increases as t increases and there is a distinct positive correlation. Doesn't exactly seem to be linear, but it might be depending on how the rest of the data looks

The next two graphs show that (1) for these coefficient parameters in the equation, NEWTON does not diverge at all, as KEPLER only calls NEWTON once for every t. However, in each of these NEWTON calls, (2) we need more iterations in these single calls to NEWTON to converge the closer we are to t = 0.2 or 0.8

How many times KEPLER calls NEWTON, per t = 0.01, 0.03….0.99:



How many iterations of NEWTON are gone through per function at t: