PHYS 410
Melvin Mathews
57181142

Assignment 1 Report

Question 1: Hybrid

In this question we were asked to design a function that used both the bisection method as well as Newton iteration to determine the roots of a given function for a given interval.

Methodology

The bisection method uses a process of shortening intervals by determining if the product of the value of the function at a given input can contain a root. If the product is evaluated to be negative, then at most one interval produces a negative value for the function while the other produces a positive value. This process is repeated to a desired accuracy.

Newton's method iteratively determines the root of the tangent starting at some given starting point until the root of the tangent is within tolerance of the actual root of the given function.

Since the function signature for hybrid only provides an interval (and not a point) we use the bisection method first and then apply Newtons method once we have performed bisection up to the given tolerance. The newtons method as well as the hybrid function overall terminates once its associated tolerance is met.

File Descriptions

- hybrid.m: This file houses the core function which performs the bisection method followed by Newtons method given a function, the derivative of the function, an interval, as well as tolerances for both root finding methods.
- findIntervals.m: In the case that the given interval may contain multiple roots, the findIntervals function is used to return an array of intervals containing a single route. This is done through determining local extrema for a given interval and returning the x values of these extrema.
- hybridTest.m: This file contains several functions used to test the hybrid function. The hybridTest function itself calls the hybrid function with the appropriate parameters. Three functions (f, g, and p) are also included this file that represent three different functions used to test the robustness of hybrid. The function dfdx can evaluate the first order derivative of any provided function f at xval.

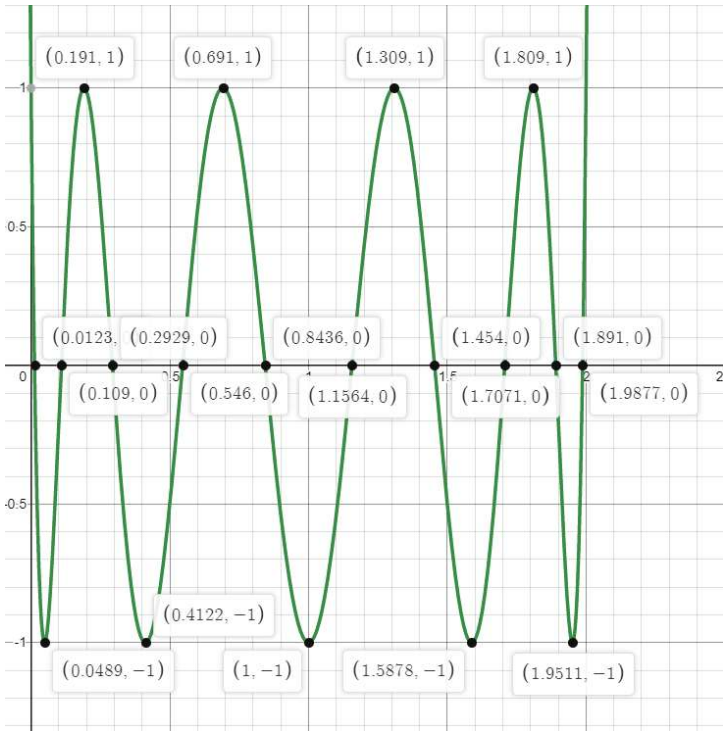
Results

The results for this question were evaluated by the implementations ability to determine the roots of the following function on the interval [0,2]:

$$f(x) = 512x^{10} - 5120x^9 + 21760x^8 - 51200x^7 + 72800x^6 - 64064x^5 + 34320x^4 - 10560x^3 + 1650x^2 - 100x + 1 \quad (1)$$

In the testing file hybridTest, this function is represented by the function g. This function is plotted in **Figure 1** below.

Figure 1. Graph of Equation 1 plotted in Desmos with local extrema and roots highlighted.



It is clear to see that in the interval 0 to 2 many roots exist. This is where the findIntervals function comes into play and returns a list of the local extrema that can then be used to recursively call hybrid. The hybrid function is only able to accurately determine a single root in an interval so determining valid intervals in this way is essential. Running hybridTest() with the g function selected and the appropriate interval prints out a series of roots into the console that match what is seen in the above plot to the provided tolerance.

Conclusion

The hybrid function successfully accomplishes the assigned task. Future work on this function could go towards improving performance time. A recursive execution is not strictly necessary so that would go a long way towards reducing time complexity. As well, caching the results of findInterval such that it does not need to go through the entire execution for the same or similar functions would also improve performance. However, for the requirements and objectives of this task, the implementation is more than suitable.

PHYS 410
Melvin Mathews
57181142

Question 2: NewtonD

In this question we were tasked with calculating the roots for a system of nonlinear equations using d-dimensional Newton iteration. As well, the Jacobian matrix for this system were to be calculated using a finite-difference technique, where each element of the matrix was calculated using the difference quotient determined using forward propagation.

Methodology

The general methodology is the same as for the Newtons method described in question 1. What differs here is that since there are multiple variables, a derivative must be computed of each nonlinear equation in terms of each of the variables. A method for organizing the various derivatives is known as a Jacobian. This Jacobian can then be evaluated at the appropriate point and the same iterative process can be conducted as before, but just with higher order visualizations (planes in place of tangents, etc.).

File Descriptions

- newtond.m: The main newtond function implementation can be found here. The function accepts a function, its corresponding Jacobian, a parameter for finite difference approximation, an initial starting value as well as a tolerance.
- newtondTest.m: This file contains several functions used to test the newtond function. The newtondTest function itself calls the newtond function with the appropriate parameters. Three functions (f, jacfd, and funcGradient) are also included this file. The f function contains within it the system of linear equations used to test newtond. The jacfd can generate a jacobian for any d-dimensional system of equations. The funcGradient is a helper function that calculates a single functions gradient which translates to a single row in the Jacobian. It is within funcGradient where the finite difference approximation is actually conducted.

Results

The results for this question were evaluated by the implementations ability to determine the roots of the following system of equations with an initial estimate of (-1.00, 0.75, 1.5):

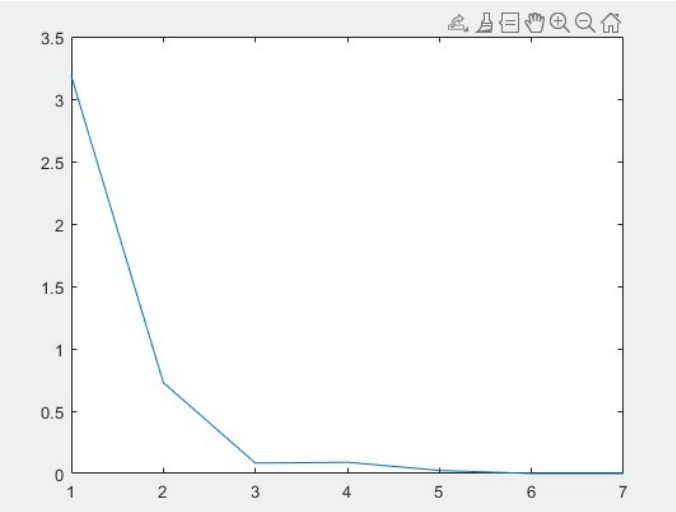
$$x^2 + y^4 + z^6 = 2 \qquad (2)$$

$$\cos(xyz^2) = x + y + z \quad (3)$$

$$y^2 + z^3 = (x + y - z)^2 \quad (4)$$

Visual methods of verifying this problem seem to be rather resource intensive so a different approach was utilized. The commented code that can be seen within newtondTest was used to verify the roots determined through newtond but through built in matlab functions instead. A graph of the norm of the x vector converging to 0 is generated within newtond and can be seen in **Figure 2** below.

Figure 2. Graph relating error (the norm of the system of equations evaluated at a given x vector) to the number of iterations.



We see that the implementation takes roughly 7 iterations to converge with an expected quadratic decrease in error. Running newtondTest() outputs the calculated roots (which match those generated through matlab functions) as well as the above graph.

Conclusion

The newtond function successfully completes the assigned task. Future work on the implementation would involve more thorough testing for more complicated systems of equations as well as an exploration of where performance could be improved and how it compares with available alternatives. As it stands, the implementation seems quite a suitable match for the problem at hand.