PHYS 410 - Homework 1

Felipe Garavelli
46427951

October 1, 2025

Contents

1 Introduction

For this homework assignment, we explore different methods for root-finding. The goal is to find solutions x such that $f(x) = 0$, where x and $f(x)$ can be either scalars or d-dimensional column vectors. In this report, we implement and test two algorithms: a hybrid bisection-Newton method for scalar functions, and a multidimensional Newton method using a finite-difference Jacobian. These algorithms were applied on two seperate testing scripts to find the roots of some given functions and test their convergence.

2 Theory

Root-finding methods aim to locate zeros of functions, i.e. values of x such that $f(x) = 0$. In class we learned two common techniques for this: the bisection method and Newton's method. Both require some preliminary information about where the root might be located.

The goal of any root-finding method is to drive the residual $f(x)$ closer and closer to zero.

2.1 Bisection Method

The bisection method is guaranteed to converge for continuous functions when there is a sign change over an interval $[x_{\min}, x_{\max}]$, i.e.

$$f(x_{\min})\, f(x_{\max}) < 0.$$

At each iteration, the method halves the interval and selects the subinterval that contains the root. The process continues until either the midpoint $\frac{1}{2}(x_{\min} + x_{\max})$ satisfies $f(x) \approx 0$, or the interval width falls below a user-defined tolerance. This makes bisection slow but very reliable.

2.2 Newton's Method

Newton's method converges much faster than bisection, provided we start with a good initial guess near the root. The basic update formula is

$$x^{(n+1)} = x^{(n)} - \frac{f(x^{(n)})}{f'(x^{(n)})}.$$

This technique achieves quadratic convergence when close enough to the true solution. For systems of nonlinear equations, Newton's method generalizes to

$$\mathbf{x}^{(n+1)} = \mathbf{x}^{(n)} - J(\mathbf{x}^{(n)})^{-1}\mathbf{f}(\mathbf{x}^{(n)}),$$

where $J(\mathbf{x})$ is the Jacobian matrix of partial derivatives. In practice, the Jacobian can be approximated using finite differences:

$$J_{ij}(\mathbf{x}) \approx \frac{f_i(x_1, \ldots, x_j + h, \ldots, x_d) - f_i(\mathbf{x})}{h}.$$

In summary, the bisection method is simple and guaranteed to work but converges slowly, while Newton's method is much faster but depends heavily on a good initial guess.

3 Problem 1

For question 1 we implemented a hybrid method to solve for roots of scalar functions. To test the implementation I solved for the roots of the function

$$f(x) = 128x^8 - 256x^6 + 160x^4 - 32x^2 + 1, \quad x \in [-1, 1]$$

that is shown in figure 1.

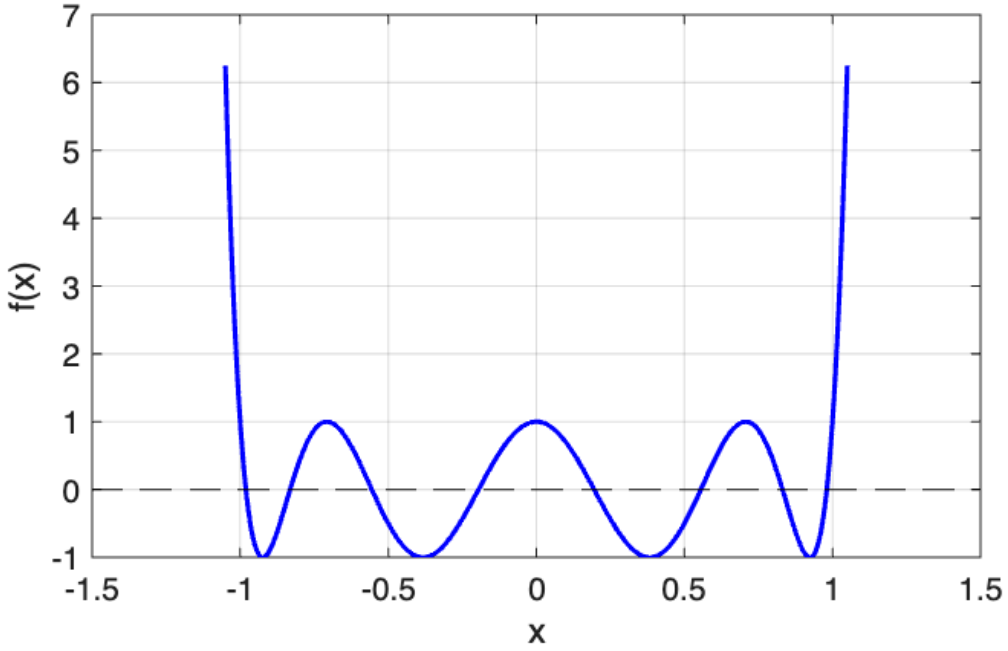

Figure 1: Function used to test implementation.

3.1 Numerical Approach and Implementation

The first problem asked for a root of a scalar function. To combine reliability with efficiency, I wrote a hybrid routine that first applies the bisection method and then switches to Newton's method.

The algorithm proceeds in two phases:

1. Bisection phase: repeatedly halve the interval $[x_{\min}, x_{\max}]$ until the relative interval width is below a tolerance `tol1` (or the residual of the midpoint is close to zero). The midpoint of the given interval is then taken as an initial guess:

$$x_{\text{mid}} = \tfrac{1}{2}(x_{\min} + x_{\max}).$$

2. Newton phase: starting from x_{mid}, update iterates using

$$x^{(n+1)} = x^{(n)} - \frac{f(x^{(n)})}{f'(x^{(n)})},$$

until the relative change in x is below another tolerance `tol2`. If the derivative becomes zero, the iteration stops with a warning.

This approach guarantees a valid initial guess (thanks to bisection) while still exploiting the fast quadratic convergence of Newton's method near the solution.

Implementing this hybrid technique was straightforward. I first wrote the algorithm for the bisection phase, and the value obtained from this step was then used as the initial guess for Newton's method. Testing the implementation required some tinkering. When I ran the function over the entire range $[-1, 1]$, it only located a single root. However, after plotting the function it was clear that there were actually eight roots in this interval. To address this, I divided the range $[-1, 1]$ into 1000 subintervals and applied the hybrid method on each piece. Using this approach, I was able to efficiently recover all eight roots.

In the case of a repeated root or when no root exists in a subinterval, the loop exits early so that no additional time is wasted.

3.2 Results

For Problem 1 my algorithim found a total of 8 roots which are shown in Table 1.

Table 1: Computed roots for Question 1.

Root	Value
1	-0.980785280403231
2	-0.831469612302545
3	-0.555570233019602
4	-0.195090322016128
5	0.195090322016128
6	0.555570233019602
7	0.831469612302545
8	0.980785280403231

To find these roots, I used a tolerance of $\texttt{tol1} = 10^{-2}$ for the bisection phase and $\texttt{tol2} = 10^{-12}$ for Newton's method. This choice allowed for both faster and more accurate convergence, reducing the total number of iterations and making the hybrid method more efficient. To illustrate this difference, I compared the iteration counts for a single root while varying `tol1`. In Table 2, the bisection phase required 11 iterations to produce an approximation accurate enough to hand off to Newton's method. In contrast, in Table 3, only one bisection iteration was needed before switching to Newton's method. Overall, using a lower tolerance for the bisection phase reduced the total number of iterations from 14 to 5, while maintaining the same final accuracy.

Table 2: Bisection–Newton iteration trace showing convergence. (tol1 = 1e-6, tol2 = 1e-12)

Method	Iter	x	$\|f(x)\|_2$
Bisection	1	-0.980980980980981	8.045×10^{-3}
	2	-0.980480480480481	1.245×10^{-2}
	3	-0.980730730730731	2.235×10^{-3}
	4	-0.980855855855856	2.897×10^{-3}
	5	-0.980793293293293	3.286×10^{-4}
	6	-0.980762012012012	9.539×10^{-4}
	7	-0.980777652652653	3.128×10^{-4}
	8	-0.980785472972973	7.897×10^{-6}
	9	-0.980781562812813	1.524×10^{-4}
	10	-0.980783517892893	7.227×10^{-5}
	11	-0.980784495432933	3.219×10^{-5}
Newton	1	-0.980785280411170	3.219×10^{-5}
	2	-0.980785280403231	3.256×10^{-10}
	3	-0.980785280403231	3.553×10^{-15}

Table 3: Bisection–Newton iteration trace showing convergence. (tol1 = 1e-2, tol2 = 1e-12)

Method	Iter	x	$\|f(x)\|_2$
Bisection	1	-0.980980980980981	8.045×10^{-3}
Newton	1	-0.980785772227360	8.045×10^{-3}
	2	-0.980785280406346	2.017×10^{-5}
	3	-0.980785280403230	1.278×10^{-10}
	4	-0.980785280403231	2.487×10^{-14}

4 Problem 2

For Question 2, we were tasked with finding a root of the following system of equations:

$$\begin{cases} x^2 + y^3 + z^4 = 1, \\ \sin(xyz) = x + y + z, \\ x = yz, \end{cases}$$

using an initial estimate of

$$\mathbf{x}^{(0)} = (3, -2, -1).$$

Since an initial guess is provided, only Newton's method is required. Because this is a three-dimensional system, it is also necessary to implement a routine to compute an approximate Jacobian, as discussed in Section 2.2.

4.1 Numerical Approach and Implementation

For the second problem, I implemented a Newton solver for nonlinear systems of equations. The method iteratively updates the solution using

$$\mathbf{x}^{(n+1)} = \mathbf{x}^{(n)} - J(\mathbf{x}^{(n)})^{-1}\mathbf{f}(\mathbf{x}^{(n)}),$$

where the Jacobian $J(\mathbf{x})$ is approximated numerically. Instead of deriving analytic derivatives, I wrote a routine that uses forward finite differences to build the Jacobian matrix directly from evaluations of $\mathbf{f}(\mathbf{x})$.

At each iteration, the code solves the linear system

$$J(\mathbf{x}^{(n)})\,\Delta\mathbf{x} = \mathbf{f}(\mathbf{x}^{(n)})$$

to update $\Delta\mathbf{x}$, in MATLAB this is computed by doing `dx = J \ res`. The new $\mathbf{x}^{(\mathbf{n+1})}$ is then updated as $\mathbf{x}^{(n+1)} = \mathbf{x}^{(n)} - \Delta\mathbf{x}$. Convergence is monitored using the root-mean-square (RMS) norm of the update, and the loop terminates once this quantity falls below a user-defined tolerance or a maximum iteration count is reached (as recomended in lecture). If the solver fails to converge within the limit, the program reports an error.

This implementation made it easy to apply Newton's method to the given system without needing closed-form derivatives. The only required inputs were the system of equations, a step size h for differencing, an initial guess, and a tolerance.

4.2 Results

For Problem 2, the d-dimensional Newton's method found the following solution:

$$\mathbf{x} = \begin{bmatrix} 2.315518418880531 \\ -1.881173775268253 \\ -1.230890228921217 \end{bmatrix}$$

Similar to Problem 1, we can analyze the convergence of each component and observe the quadratic convergence in Table 4. Since this algorithm requires a good initial guess, Newton's method converged in only six iterations.

Table 4: Newton iteration trace showing convergence of the solver.

Iter	x_1	x_2	x_3	$\|f(x)\|_2$
1	2.30051732616501	-1.81457899365974	-1.24296916625206	8.323×10^{-1}
2	2.32281359933536	-1.88671652574839	-1.23067051862848	4.146×10^{-1}
3	2.31567828186382	-1.88126959362919	-1.23091181369641	1.854×10^{-2}
4	2.31551850945438	-1.88117382713273	-1.23089024423081	2.424×10^{-4}
5	2.31551841888023	-1.88117377526793	-1.23089022892127	1.349×10^{-7}
6	2.31551841888053	-1.88117377526825	-1.23089022892122	1.483×10^{-12}

As shown in the table, the RMS norm of the residual decreases rapidly, illustrating the expected quadratic convergence of Newton's method when the initial guess is sufficiently close to the true root.

5 Conclusion

For this homework, I was tasked with implementing two root-finding algorithims. For the scalar problem (Question 1), a hybrid Bisection–Newton method was implemented. Bisection provided a reliable initial guess, which was then refined with Newton's method to achieve fast, accurate convergence. For the multidimensional problem (Question 2), Newton's method with a finite-difference Jacobian was used to solve a nonlinear system, demonstrating rapid quadratic convergence when starting from a good initial guess. Overall, bisection is simple, robust, and guarantees convergence but is relatively slow, while Newton's method converges much faster but relies on an accurate initial guess and, for systems, a proper Jacobian. Combining these techniques I was able to exploit the strengths of each method, achieving both reliability and efficiency in root finding.

I used ChatGPT as a guide to help me understand the methods and clarify concepts. All code was written by me. AI was primarily used for formatting tables in my LaTeX report, generating MATLAB plots, and checking spelling and grammar.

6 Files

The following MATLAB files were created for this assignment:

- **Question 1:** `hybrid.m`, `thybrid.m`
- **Question 2:** `newtond.m`, `tnewtond.m`, `jacfd.m`