

CS/MATH 375 - Homework 7

Adam Fasulo

November 28, 2025

Problem 1: Plotting the Function

First, we need to plot the function $f(x) = x^3 - 8$ for $a = 8$. The goal is to visualize the zero crossings.

Generated Plot

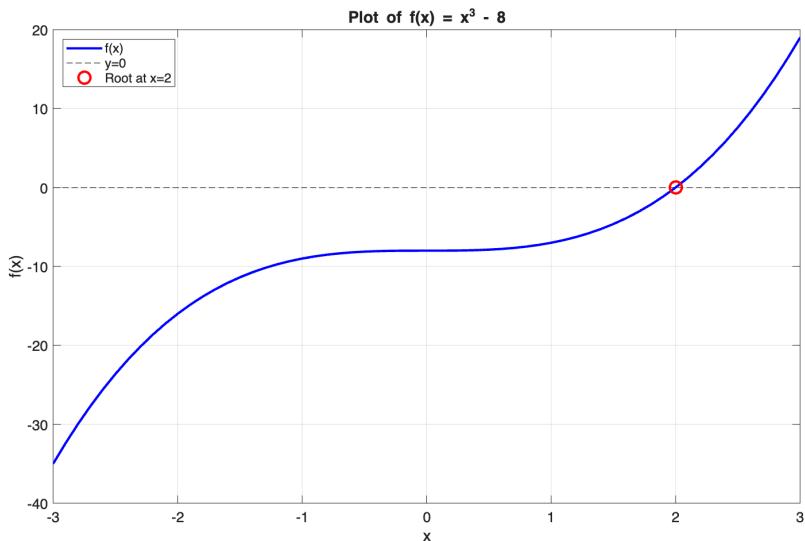


Figure 1: Plot of $f(x) = x^3 - 8$ showing its single real root.

How many zeros are there?

For the function $f(x) = x^3 - 8$, there is only one real zero (or root). As seen from the plot, the graph crosses the x-axis only once, at $x = 2$. While a cubic polynomial has three roots in the complex plane, only one is real.

Problem 2: Rootfinding Algorithms

The Bisection and Newton's method algorithms were implemented as Matlab functions. The complete code for these functions is included in Appendix A at the end of this report.

Problem 3: Testing the Methods

The implemented methods were tested using a tolerance of ' tol ' = 10^{-10} and $a = 8$. The initial bracket for the bisection method was [1, 4], and the initial guess for Newton's method was 4.

Results

The computed root and number of iterations for each method are reported below.

- **Bisection Method:**

- Computed Root: 1.999999999985448
- Iterations: 36

- **Newton's Method:**

- Computed Root: 2.000000000000000
- Iterations: 7

As expected, Newton's method converged to a highly accurate root in significantly fewer iterations than the Bisection method.

Problem 4: Convergence Rate Analysis

The convergence rates were analyzed using the true root $x^* = \sqrt[3]{8} = 2$ to calculate the error $e_k = |x_k - x^*|$ at each step. For Bisection, we use $r = 1$, and for Newton's method, we use $r = 2$.

Bisection Method Convergence

Table 1: Convergence of Bisection Method ($r = 1$)

Iteration (k)	Error (e_k)	Ratio (e_{k+1}/e_k)
1	5.000000×10^{-1}	0.500000
2	2.500000×10^{-1}	0.500000
3	1.250000×10^{-1}	0.500000
...
34	5.820766×10^{-11}	0.500000
35	2.910383×10^{-11}	0.500000
36	1.455192×10^{-11}	—

Observed Convergence: The Bisection method demonstrates linear convergence. The ratio e_{k+1}/e_k is consistently 0.5, which is the classic behavior for this method, as the error is halved at each iteration.

Newton's Method Convergence

Table 2: Convergence of Newton's Method ($r = 2$)

Iteration (k)	Error (e_k)	Ratio (e_{k+1}/e_k^2)
1	2.000000×10^0	0.208333
2	8.333333×10^{-1}	0.318339
3	2.210688×10^{-1}	0.435296
4	2.127354×10^{-2}	0.493002
5	2.231146×10^{-4}	0.499926
6	2.488636×10^{-8}	0.717047
7	4.440892×10^{-16}	0.000000
8	0.000000×10^0	—

Observed Convergence: Newton's method exhibits quadratic convergence. The ratio e_{k+1}/e_k^2 approaches a constant, meaning the number of correct decimal places roughly doubles with each iteration until machine precision is reached.

Problem 5: By-Hand Convergence Proofs

The convergence behavior is proven theoretically.

(a) General Convergence for $f(x) = x^3 - a$, where $a \neq 0$

Theorem: For a simple root x^* where $f'(x^*) \neq 0$, Newton's method converges quadratically if the initial guess is sufficiently close.

Proof: For $f(x) = x^3 - a$, the root is $x^* = a^{1/3}$. The derivative is $f'(x) = 3x^2$. At the root, $f'(x^*) = 3(a^{1/3})^2 = 3a^{2/3}$. Since $a \neq 0$, $f'(x^*) \neq 0$. Thus, the root is simple and the method converges quadratically.

(b) Convergence Rate for $a = 0$

Analysis: If $a = 0$, $f(x) = x^3$ and the root is $x^* = 0$. The derivative at the root is $f'(0) = 3(0)^2 = 0$. This is a multiple root, so convergence is not quadratic. The iteration formula becomes:

$$x_{k+1} = x_k - \frac{x_k^3}{3x_k^2} = x_k - \frac{1}{3}x_k = \frac{2}{3}x_k$$

The error is $e_{k+1} = |x_{k+1}| = \frac{2}{3}|x_k| = \frac{2}{3}e_k$. The error is reduced by a constant factor, which is the definition of linear convergence.

A Matlab Function Code

Main Function and Derivative

Listing 1: my_fcn.m

```
1 function y = my_fcn(x)
2     a = 8;
3     y = x.^3 - a;
4 end
```

Listing 2: my_dfcn.m

```
1 function y = my_dfcn(x)
2     y = 3*x.^2;
3 end
```

Rootfinding Algorithms

Listing 3: my_bisection.m

```
1 function [x_arr] = my_bisection(f, c, d, tol)
2     % f: function handle
3     % [c, d]: initial bracket
4     % tol: tolerance
5
6     x_arr = []; % Initialize array of iterates
7
8     if f(c) * f(d) >= 0
9         error('f(c) and f(d) must have opposite signs.');
10    end
11
12    while true
13        mid = (c + d) / 2;
14        x_arr = [x_arr, mid];
15
16        delta_x = d - c; % Length of the current interval
```

```

17     if (abs(delta_x) < tol) || (abs(f(mid)) < eps)
18         break;
19     end
20
21     if f(c) * f(mid) < 0
22         d = mid;
23     else
24         c = mid;
25     end
26 end
27 end

```

Listing 4: my_newton.m

```

1 function [x_arr] = my_newton(f, df, x0, tol)
2 % f: function handle
3 % df: derivative function handle
4 % x0: initial guess
5 % tol: tolerance
6
7 x_arr = [x0]; % Initialize with the initial guess
8 xk = x0;
9
10 while true
11     fxk = f(xk);
12     dfxk = df(xk);
13
14     if dfxk == 0
15         error('Derivative is zero. Newton''s method failed.');
16     end
17
18     xk_plus_1 = xk - fxk / dfxk;
19     delta_x = xk_plus_1 - xk;
20     x_arr = [x_arr, xk_plus_1];
21
22     if (abs(delta_x) < tol) || (abs(f(xk_plus_1)) < eps)
23         break;
24     end
25
26     xk = xk_plus_1;
27 end
28 end

```