Hybrid and Alternative Root Finding Techniques

Olivia Golden
Christopher Gonzalez-millan
Tim Kan

December 15, 2021

Abstract

This project aims to discuss and examine the different root-finding techniques covered in the APPM 4650 course, namely the bisection method, Iterative method, Newton's method, and the secant method. Each of these methods are dissected to explore their mathematical formulations, assumptions needed for successful convergence, advantages and disadvantages, and how combinations of methods can be constructed to achieve superior performance.

Three additional methods not discussed in class are also explored: the False Position method, Muller's method, and Brent's method. These three methods are unique but are constructed based on similar approaches to methods that were already explored. The False Position method is a mixture of both the bisection method and the secant method. However, it has a different order of convergence depending on the initial interval. This makes the False Position method a technique of interest as its convergence rate is dependent on the function. Muller's method is similar to the secant method but has faster convergence and can find complex roots without complex initial guesses. As for Brent's method, it is the official method for root finding in Matlab since it is very robust and can have up to superlinear convergence. Each of these methods are discussed regarding their derivation, convergence rate, and how combining them with other methods can improve the outcome. These new methods are then be carefully analyzed in comparison to the methods covered in the course.

Contents

1 Introduction **5**
1.1 Bisection Method 5
1.2 Fixed Point Iteration 6
1.3 Newton's Method 7
1.4 Secant Method 8

2 Mathematical Formulation **9**
2.1 Bisection Method 9
2.2 Fixed Point Iteration 10
2.3 Newton's Method 12
2.4 Secant Method 14
2.5 Method Summary Table 17

3 Computational Analysis **18**
3.1 Bisection Method 19
3.2 Fixed Point Method 20
3.3 Newton's Method 23
3.4 Secant Method 24
3.5 Computational Analysis Overview 26

4 Combination of Methods **27**
4.1 Bisection and Newton's Method 27
4.2 Bisection and Secant Method 30

5 Project Extension **32**
5.1 Introduction 32
5.2 Mathematical Formulation 32
5.2.1 False Position method 32
5.2.2 Muller's method 35
5.2.3 Brent's method 37
5.2.4 Project Extension Method Summary Table 39
5.3 Computational Analysis 39
5.4 False Position Method 40
5.4.1 Muller's Method 41
5.4.2 Brent's Method 43
5.5 Project Extension Overview 45

6 Conclusion **46**
6.1 Analysis of Results 47
6.2 Future Work 48

7 Appendix **50**
7.1 Intermediate Value Theorem (I.V.T) 50
7.2 Mean Value Theorem 50
7.3 Absolute Error 50
7.4 Code For Bisection 51
7.5 Code For Fixed Point Iteration 52
7.6 Code For Newton's method 52
7.7 Code for Secant method 53
7.8 Code for combined bisection and Newton's method 53
7.9 Code for naive combination of Bisection and Secant's method 55
7.10 Code for False Position 56
7.11 Code for Muller's Method 57
7.12 Code for Brent's Method 57

List of Figures

1 Bisection Method 6
2 Fixed Point Iteration 7
3 Newton's Method 8
4 the secant method 8
5 Error function for Bisection 19
6 Error function for Fixed Point Iteration 20
7 Fixed Point $f(x) = 1 + 2sin(x) - x$ 21
8 Fixed Point $f(x) = -16 + 6x - 12/x$ 22
9 Error function for Fixed Point Iteration 22
10 Error function for Newton's method 23
11 Error function Using Newton for Failed Function 24
12 Error function for Secant method 25
13 Error function for Secant method 26
14 Absolute Error Approx. over the iterations 27
15 Comparison of Absolute Error Approximation over the iterations 29
16 Error Bisection secant combination $x^3 - x^2 - x - 1$ 32
17 False Position method 34

18 Secant (a) v. Muller (b) . . . 35
19 Error function for Regula Falsi . . . 40
20 Error function for Muller's . . . 42
21 Error function for Muller's $\sqrt{x+1}$ . . . 43
22 Absolute Error Approximation for Brent's Method with multiple functions . 44
23 Absolute Error Approximation of All method on function $x^3 - 2$ . . . 46

List of Tables

1 Method Summary Table 1 . . . 17
2 Method Summary Table 2 . . . 18
3 Bisection Method Results . . . 19
4 Fixed Point Method Results . . . 20
5 Newton's Method Results . . . 23
6 Secant Method Results . . . 25
7 Comparison of the methods . . . 26
8 Bisection Newton combination compared to Bisection method . . . 28
9 Bisection Newton combination compared to Newtons method . . . 29
10 Bisection Secant combination compared to Bisection method . . . 31
11 Bisection Secant combination compared to Secant method . . . 31
12 Project Extension Method Summary Table . . . 39
13 False Position method . . . 40
14 Muller's Method Results . . . 41
15 Brent's Method Results . . . 44
16 Comparison of the methods on the function $x^3 - 2$ . . . 45

1 Introduction

This project specifically examines root-finding techniques for scalar functions with one variable. Many mathematical and scientific problems require the method of root-finding for a scalar function such as weather forecasting, circuit analysis, population dynamic, optimization, and oscillations. However, sometimes the analytical solution to the problem cannot be found. This is when the user can utilize various types of root-finding methods to arrive at a numerical solution to a particular problem. In addition, there are many different root finding techniques, so it is important to understand their advantages and disadvantages and when to use them.

The different root-finding methods explored in this project include the bisection method, Fixed Point Iteration method, Newton's Method, and the secant method, as well as the False Position method, Muller's method, and Brent's method. Descriptions of the first four methods can be found below, while descriptions of the latter three methods can be found in Section 5, the Project Extension section.

Although each of the aforementioned methods can be applied to a root-finding problem, it is important to recognize each method's limitations. This was the project's main goal: to identify these limitations and explore how they can be remedied through a combination of methods.

1.1 Bisection Method

The Bisection method is based on the Intermediate Value Theorem (IVT) [**7.1**]. Given the assumptions that the function f is continuous on the interval $[a, b]$ and that $f(a)$ and $f(b)$ have opposite signs, the IVT can be used to show that there exists a point $p \in [a, b]$ such that $f(p) = 0$. Typically, the user would bisect the interval with the value p being the midpoint of the interval $[a, b]$, then the user can apply the IVT on either interval, $[a, p]$ or $[p, b]$, to see which interval contains the root. The user would then continue to bisect on the interval until the the value of $f(p)$ is within the desired tolerance. A visualization of how the method works can be seen in Figure 1.

[2]

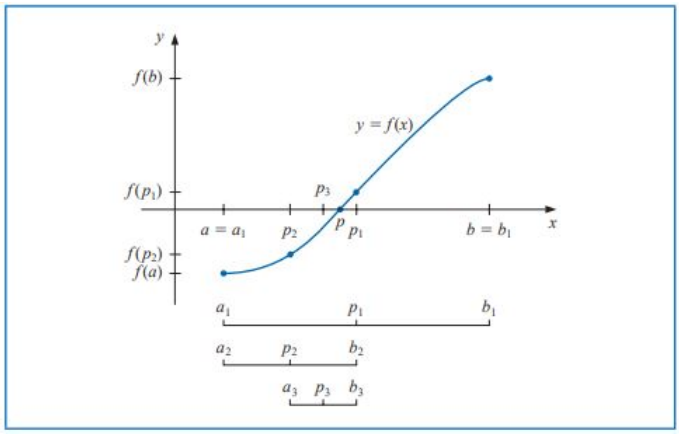
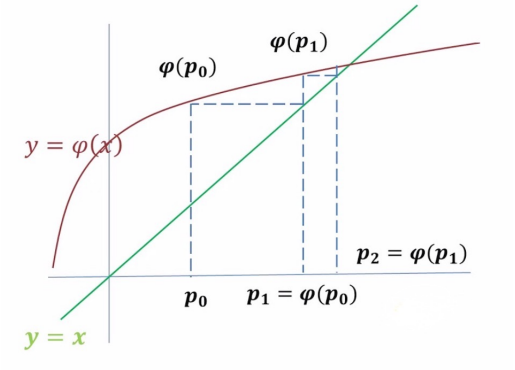


Figure 1: Bisection Method

1.2 Fixed Point Iteration

Fixed Point Iteration is based on the idea of a fixed point. A fixed point, p, of a given function $g(x)$ is defined by the property $g(p) = p$. In other words, the fixed point p on a given function is also on the line $y = x$.

It is possible to apply a fixed point to finding the root of a function f by defining a function $g(x)$ such that the root of f is the fixed point of g. Specifically, finding the fixed point of g is equivalent to finding the root of $f(x) = x - g(x)$. This can be done through Fixed Point Iteration. Given $g(x)$ and an initial starting point p_0, this method constructs a sequence $\{p_n\}_{n=0}^{\infty}$ such that $p_n = g(p_{n-1})$ for $n > 0$. If p_n converges to p and g is continuous, then $p = g(p)$ and p is the root of the equation f. A visualization of how the method works can be seen in Figure 2.



[2]

Figure 2: Fixed Point Iteration

1.3 Newton's Method

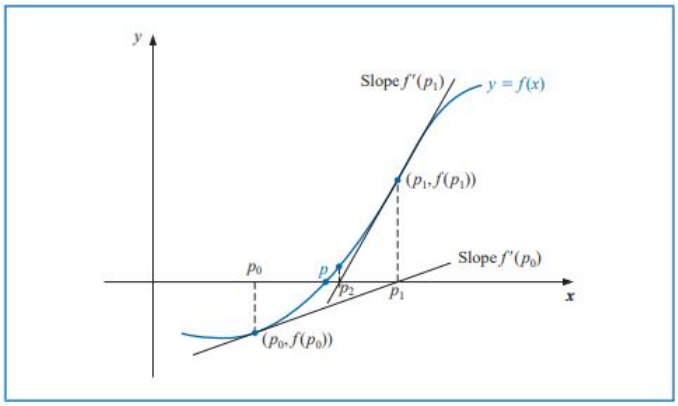
Newton's method is a powerful, iterative technique that can be used to approximate the root of a function based on a linear approximation of an initial point. For example, given a one dimensional case of a function $f(x)$ and an initial guess of the root x_1, a first-order Taylor expansion around x_1 can be constructed. This represents the tangent line to the function $f(x)$ at the point x_1. The updated estimate of the root then becomes the root of the created tangent line. Since x_1 is the initial guess of the root, the equation of the tangent line to x_1 is

$$y = f(x_0) + f'(x_0)(x - x_0).$$

Let x_2 be the x-intercept of the tangent line at the initial guess. Using the tangent line, obtain the equation of

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}.$$

There is no guarantee that the new value of x_1 has the value $f(x_1) \approx 0$, but the value x_1 is typically closer to the root than its previous value of x_0. From this, an approximation of the root can be found by continuing iterating through the process. A visualization of how the method works can be seen in Figure 3.



[2]

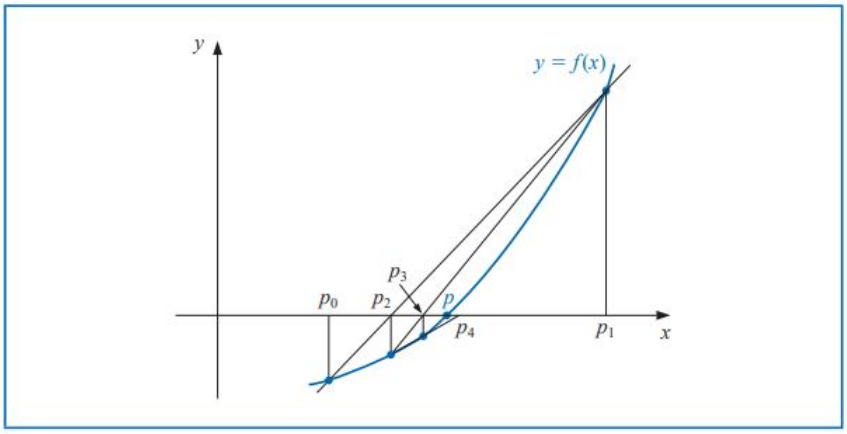
Figure 3: Newton's Method

1.4 Secant Method

The Secant Method, which is a variation of Newton's Method, uses the iteration step

$$x_{n+1} = x_n - f(x_n)\frac{x_{n-1}-x_n}{f(x_{n-1})-f(x_n)}.$$

Using the secant line in this way requires fewer calculations per step than Newton's method making it a cheaper alternative. This method is also useful when the derivative of a function is not known. However, the Secant Method converges superlinearly which is slower than Newton's quadratic convergence. A visualization of how the method works can be seen in Figure 4.



[2]

Figure 4: the secant method

2 Mathematical Formulation

This section analyzes the derivation of each of the aforementioned methods and when they should be applied. While exploring each method, the convergence rate is considered as well as any limitations that the method might have.

2.1 Bisection Method

The bisection method utilizes the Intermediate Value Theorem to find the root of the function f. Assuming that the function $f(x)$ is continuous on the defined, closed interval $[a, b]$ with $f(a)$ and $f(b)$ having opposite signs. The interval $[a, b]$ is divided into two different halves at $c = \frac{a+b}{2}$, where c is the midpoint of the interval $[a, b]$. If $f(c) = 0$ then the root for the function $f(x)$ at x^* has been successfully found.

If c turns out to not be the root to the function, then the method would continue to bisect the sub-interval that was created using the interval of $[a, c]$ or $[c, b]$. The correct interval can be found with IVT, by testing the product of the endpoints at each interval,

$$f(a)f(c) < 0 \text{ or } f(c)f(b) < 0.$$

From this, sufficient information can be obtained to decide which interval the root is in. The user can continue to bisect the sub-interval and apply the IVT until it is possible to find the exact solution c_n where $f(c_n) = 0$ or where c_n is relatively close enough to a predetermined tolerance. Another stopping criteria that can be set is by how many iterations the user wants the application to run. The bisection method runs until one of those criteria is met.

The algorithm shows that if f is continuous and defined on the closed interval $[a, b]$ and $f(a)f(b) < 0$, then there exists a root within that interval. As long as these assumptions are satisfied, the bisection method converges. The rate of convergence is slow, but the method guarantees convergence to a solution. This tells us that this method is robust and reliable but could take a lot of iterations to conclude.

Theorem 1. *[**2**] Suppose that the function f is continuous on the defined interval $[a, b]$ and $f(a)f(b) < 0$. Then it is feasible to create a sequence using Bisection method:*

$$|p_n - p| \leq \frac{b-a}{2^n}, n \geq 1$$

Proof. [**2**] For each $n \geq 1$:

$$b_n - a_n = \frac{b-a}{2^{n-1}}$$

Since it is know that $p_n = \frac{1}{2}(a_n + b_n)$ for all $n \geq 1$, then is achievable to show that

$$|p_n - p| \leq \frac{b-a}{2^n}$$

This shows that the sequence $\{p_n\}_{n=1}^{\infty}$ converges to the p with convergence rate of $1/2$ with linear order of convergence □

The bisection method is very easy to implement when a continuous function is given. The user would only need to find an interval that satisfy IVT for the function to successfully converge. If the two assumptions are met, the bisection method is guarantee to converge. The main drawback for bisection is that it is slow to converge.

2.2 Fixed Point Iteration

Fixed Point Iteration is based on the idea of a fixed point, p, that is, such that given a function g, $g(p) = p$.

$g(x)$ can be found by algebraically converting the equation $f(x) = 0$ into a fixed point function that satisfies $x = g(x)$. Applying the iterative scheme results in:

$$x_{i+1} = g(x_i)$$

However, it is important to note that not every interval $[a, b]$ of a function f contains a fixed point.

Theorem 2. *[**2**]*
i) *If $g \in C[a, b]$ and $g(x) \in [a, b]$ for all $x \in [a, b]$, then g has at least one fixed point in $[a, b]$.*
ii) *If, in addition, $g'(x)$ exists on (a, b) and a positive constant $k < 1$ exists with $|g'(x) \leq k|$ for all $x \in (a, b)$ then there is exactly one fixed point in $[a, b]$*

Proof. [**2**] **i)** Let $a, b \in \mathbb{R}$ such that a or b is a fixed point if $g(a) = a$ or $g(b) = b$, respectively. Otherwise, $g(a) > a$ and $g(b) < b$. Let us also define $f(x) = x - g(x)$.
This implies that $f(a) > 0$ and $f(b) < 0$. By the Intermediate Value Theorem, there must exist a $p \in [a, b]$ such that $f(p) = 0$ and $g(p) = p$.

ii) Suppose, in addition, that $|g'(x)| \leq k < 1$ and that p and q are both fixed points in $[a, b]$. If $p \neq$q, then the Mean Value Theorem [**7.2**] implies that a number ξ exists between p and q, and hence in $[a, b]$, with $\frac{g(p)-g(q)}{p-q} = g'(\xi)$.
Therefore,

$$|p-q| = |g(p) - g(q)| = |g'(\xi)||p-q| \leq k|p-q| < |p-q|.$$

This cannot be true and $p \neq q$. Thus, $p = q$ and the fixed point on a, b is unique. □

A limitation of Fixed Point Iteration is that is does not always converge.

Theorem 3. ***Fixed Point Theorem*** *[**2**] Let $g \in C[a,b]$ such that $g(x) \in [a,b]$ for all $x \in [a,b]$.*
Suppose, in addition, that $g(x)$ exists on (a,b) and there exists a constant k such that $0 < k < 1$ with $|g'(x)| < k$ for all $x \in [a,b]$. Note that $g'(x)$ does not need to exist at p so long as the limit approaches 0 and x approaches p.
Then for any initial guess $p_0 \in [a,b]$, the sequence defined by $p_n = g(p_{n-1})$, $n \geq 1$ converges to a unique fixed point in $[a,b]$.

Proof. [**2**] Theorem 2 implies that a unique point p exists in $[a,b]$ with $g(p) = p$. Since g maps $[a,b]$ into itself creating a contractive map, the sequence $p_n{}_{n=0}^{\infty}$ is defined for all $n \geq 0$, and $p_n \in [a,b]$ for all n. Using the fact that $|g'(x)| \leq k$ and the Mean Value Theorem, then for each n,

$$|p_n - p| = |g(p_{n-1}) - g(p)| = |g'(\xi_n)||p_{n-1} - p| \leq k|p_{n-1} - p|,$$

where $\xi_n \in (a,b)$. Applying this inequality inductively gives

$$|p_n - p| \leq k|p_{n-1} - p| \leq k^2|p_{n-2} - p| \leq \cdots \leq k^n|p_0 - p|.$$

Since $0 < k < 1$, then $\lim_{n\to\infty} k^n = 0$ and

$$\lim_{n\to\infty} |p_n - p| \leq \lim_{n\to\infty} k^n|p_0 - p| = 0.$$

Therefore, $\{p_n\}_{n=0}^{\infty}$ converges to p. □

Now, the next theorem examines the rate of convergence for Fixed Point Iteration.

Theorem 4. *Let $g \in C[a,b]$ satisfy $g(x) \in [a,b]$ for all $x \in [a,b]$. Assume $g'(x)$ continuous with $|g'(x)| \leq k$ for all $x \in (a,b)$ and constant $k < 1$.*
i) *If $g'(p) \neq 0$, the sequence converges linearly to the fixed point.*
ii) *If $g'(p) = 0$, the sequence converges at least quadratically to the fixed point.*

Proof. **i)** By Fixed Point Theorem, Fixed Point $p \in [a,b]$ uniquely exists, and Fixed Point Iteration converges to p. By Mean Value Theorem, there exists a ξ_k in between p_k and p such that

$$g(p_k) - g(p) = g'(\xi_k)(p_k - p), \text{ for all k.}$$

Therefore,

$$\lim_{k\to\infty} g'(\xi_k) = g'(p)$$

$$\lim_{k\to\infty} \frac{|p_{k+1}-p|}{|p_k-p|} = \lim_{k\to\infty} \frac{|g(p_k)-g(p)|}{|p_k-p|} = \lim_{k\to\infty} \frac{|g'(\xi_k)(p_k-p)|}{|p_k-p|} = |g'(p)|$$

ii) Assume Fixed Point Iteration $\{p_k\}_{k=1}^{\infty}$ converges to p, with $g'(p) = 0$ and $g''(p) \neq 0$.

Using Taylor expansion at p with ξ_k between p and p_k,

$$g(p_k) = g(p) + g'(p)(p_k - p) + \tfrac{1}{2}g''(\xi_k)(p_k - p)^2 = g(p) + \tfrac{1}{2}g''(\xi_k)(p_k - p)^2$$

since $g'(p) = 0$. Therefore,

$$\lim_{k\to\infty} = \frac{|p_{k+1}-p|}{|p_k-p|^2} = \lim_{k\to\infty} = \frac{|g(p_k)-p|}{|p_k-p|^2} = \frac{1}{2}\lim_{k\to\infty} = \frac{|g''(\xi_k)(p_k-p)^2|}{|p_k-p|^2} = \frac{1}{2}|g''(p)|$$

□

The three theorems above show that Fixed Point Iteration converges when $g(x)$ exists on (a, b) and there exists a constant k such that $0 < k < 1$ with $|g'(x)| < k$ for all $x \in [a, b]$. It converges linearly if $g'(p) \neq 0$, and at least quadratically if $g'(p) = 0$.

Overall, Fixed Point Iteration is easy to implement and is a good general algorithm. However, it does not always have quadratic convergence and should be primarily used when it does, or when $g'(p) = 0$.

2.3 Newton's Method

Newton's Method creates a tangent line at a chosen initial point, $(x_0, f(x_0))$ on the function f. Using IVT, the method can obtain a good approximation of the interval the root is in. The user must chose an initial point x_0 that is close enough to the root for the algorithm to successfully converge. Since the coordinates of the initial point and the equation of the tangent line are know, it is possible to construct:

$$y = f(x_0) + f'(x_0)(x - x_0).$$

From this, we can find where the tangent line intersects the x-axis and choose this to be the next guessing point. Rearranging the equation would yield

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Theorem 5. *[2] Let $f(x) \in C^2[a,b]$ and that there exist $x^* \in (a,b)$ with $f'(x^*) \neq 0, f(x^*) = 0$ and $|f''(x)| < \epsilon$ near x^* Let x^* be a point near the x. This is the information required to create a first degree Taylor polynomial for $f(x)$ about x^**

$$f(x) = f(x^*) + (x - x^*)f'(x) + \frac{(x - x^*)^2}{2}f''(\xi(x))$$

where $\xi(x)$ lies between x and x^*. Since $f(x^*) = 0$ we have:

$$0 = f(x^*) + (x - x^*)f'(x^*)$$

solving for x:

$$x = x^* - \frac{f(x^*)}{f'(x^*)}$$

and we can obtain the a better successive approximations for the method with iterating:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Let's assume that x_n converges to x^* as $k \to \infty$. Then for sufficiently large enough n:

$$|x_{n+1} - x^*| \leq M|x_n - x^*|^2, M > \frac{|f''(x^*)|}{2|f'(x^*)|}$$

Thus, x_k converges to x^* quadratically.

Proof. [**3**] Assume that Newton's iteration x_{x+1} converges toward the point x^* with $f'(x^*) \neq 0$ and the error can be defined after the n^{th} step by

$$\epsilon_n = x_n - x^*$$

we can expand use Taylor polynomial for $f(x_n)$ about x^*

$$f(x_n) = f(x^*) + f'(x^*)\epsilon_n + \frac{1}{2}f''(x^*)\epsilon_n^2 + ...$$

$$f(x_n) = f'(x^*)\epsilon_n = \frac{1}{2}f''(x^*)\epsilon_n^2$$

$$f'(x_n) = f'(x^*) + f''(x^*)\epsilon_n$$

but since,

$$\epsilon_{n+1} = \epsilon_n + (x_n + 1 - x_n)$$

$$\epsilon_{n+1} = \epsilon_n - \frac{f(x_n}{f'(x_n)}$$
$$\epsilon_{n+1} \approx \epsilon_n - \frac{f'(x^*)\epsilon_n + \frac{1}{2}f''(x^*)\epsilon_n^2}{f'(x^*) + f''(x^*)\epsilon_n}$$

giving us:

$$\epsilon_{n+1} \approx \frac{f''(x^*)}{2f'(x^*)}\epsilon_k^2$$

Showing us that Newton's method converges quadratically □

Newton's Method is powerful as it has fast convergence. However, if the function has a hard derivative to obtain, has multiple roots, or if the initial guess given by the user is not within the basin of convergence, then Newton's method is not guaranteed to converge and could possibly diverge.

2.4 Secant Method

The main idea behind the Secant Method is to adapt Newton's Method to find the root without the use of the derivative. This can be done by using a secant line instead of a tangent.

Derivation We can derive the Secant Method from the definition of a secant line in point-slope form given the points (x_0, x_1), (y_0, y_1).

$$y = y_0 + \tfrac{y_1 - y_0}{x_1 - x_0}(x - x_0)$$

Let $(x_0, f(x_0))$ and $(x_1, f(x_1))$ be two nearby points on the graph of $f(x)$. Using the definition above, we get the line

$$y = F(x) \equiv f(x_1) + \tfrac{f(x_0) - f(x_1)}{x_0 - x_1}(x - x_1)$$

The function $F(x)$ has a zero at x_2 if

$$0 = f(x_1) + \tfrac{f(x_0) - f(x_1)}{x_0 - x_1}(x_2 - x_1)$$
$$x_2 = x_1 - f(x_1)\tfrac{x_0 - x_1}{f(x_0) - f(x_1)}$$

If we consider x_0, x_1, x_2 to be successive approximations of the zero of $f(x)$, and set $x_2 = x_{n+1}$, $x_1 = x_n$, $x_0 = x_{n-1}$, we can generalize the equation to be

$$x_{n+1} = x_n - f(x_n)\tfrac{x_{n-1} - x_n}{f(x_{n-1}) - f(x_n)}$$

which is the Secant Method iteration equation.

Now, we consider the Secant Method's rate of convergence. The below proof shows that the convergence rate of the Secant Method is 1.618.

Proof. [**4**] As shown above, the iteration scheme of the secant method is

$$x_{n+1} = x_n - f(x_n)\frac{x_{n-1}-x_n}{f(x_{n-1})-f(x_n)}$$

Let $f(\xi) = 0$ and $e_n = (\xi - x_n)$ where e_n is the error of the n^{th} iteration estimating ξ.

From this we get

$$e_{n+1} = \frac{e_{n-1}f(x_n) - f(x_{n-1})e_n}{f(x_n) - f(x_{n-1})} \tag{1}$$

By the Mean Value Theorem,

$$f'(\eta_n) = \frac{f(x_n)-f(\xi)}{x_n-\xi}$$
$$\therefore f(\xi) = 0, x_n - \xi = e_n$$

We then get

$$f'(\eta_n) = \frac{f(x_n)}{e_n} \text{ and}$$
$$f'(\eta_n) = f(x_n)e_n \tag{2}$$

$$f(x_{n-1}) = f'(\eta_{n-1})e_{n-1} \tag{3}$$

Using (1), (2), and (3),

$$e_{n+1} = e_n e_{n-1}\frac{f'(\eta_n) - f'(\eta_{n-1})}{f(x_n) - f(x_{n-1}} \tag{4}$$
$$e_{n+1} \propto e_n e_{n-1}$$

By the definition of rate of convergence,

$$e_n \propto e_{n-1}^p, e_{n+1} \propto e_n^p \tag{5}$$

From (4) and (5),

$$e_n^p \propto e_{n-1}e_{n-1}$$
$$e_n \propto e_{n-1}^{\frac{p+1}{p}} \tag{6}$$

Using (5) and (6),

$$p = \frac{(p+1)}{p}$$
$$p^2 - p - 1 = 0$$

Solving the quadratic gives

$$p = \frac{1 \pm \sqrt{5}}{2}$$
$$\therefore p > 0, p = 1.618$$
$$\therefore e_{n+1} \propto e_n^{1.618}$$

Since, $1.618 < 2$, Secant Method's convergence is superlinear. □

There are some advantages and disadvantages to using the Secant Method. As shown above, it converges faster than linear convergence, but it is slower than Newton's quadratic convergence. Unlike Newton's Method, it does not require a derivative and requires only one function evaluation per iteration. However, the Secant Method may not converge if the initial guess is not close enough and there is no guaranteed error bound for the computed iterates.

Overall, this method is useful in situations where the derivative is too expensive or cannot be computed.

2.5 Method Summary Table

Method	Assumptions for Convergence	Convergence Rate	Order of Convergence
Bisection Method	• Function is continuous • Given interval [a,b], $f(a)f(b) < 0$	$\frac{1}{2}$	Linear
Fixed-Point Method	• Function is continuous • if $g'(x)$ exists, there is a k such that $0 < k < 1$ with $\|g'(x)\| < k$ for all $x \in [a, b]$ • $g'(x)$ forms a contractive map	• Function is continuous • $g'(p) \neq 0$ Linear • $g'(p) = 0$ At least quadratic	depends on $g'(p)$

Table 1: Method Summary Table 1

Method	Assumptions for Convergence	Convergence Rate	Order of Convergence
Newton's Method	• $f(x) \in C^2[a,b]$ and $f'(x^*) \neq 0$ where x^* is approximation of the root. • initial guess on basin of convergence	2	Quadratic
Secant Method	• initial guess on basin of convergence • if f(a) are f(b) too similar, divide by close to 0 or 0	$\frac{1+\sqrt{5}}{2}$	Superlinear

Table 2: Method Summary Table 2

3 Computational Analysis

From the Mathematical Formulation section, it is known that some methods converge faster than others. For this section, we implement the aforementioned root-finding algorithms using Python and apply them across various functions. We then compare the methods against each other, while considering the number of iterations it takes to converge and its run time. By doing so, we gain a better understanding of how some methods are better than others and use this knowledge when considering combinations of methods.

To keep everything consistent, we test all the methods with the same error tolerance of **1e-10**. For each of the tests, we record the number of iterations it took and its run time in seconds. We then analyze why a specific function failed or to what extent it was successful.

3.1 Bisection Method

For this section, we chose functions that show the robustness of the bisection method. We chose three high-degree polynomials, a trig function, and an exponential function.

Function	Interval	Iterations	Approximated Root(x^*)	Run time (seconds)
$x^2 - 8$	[1,3]	35	2.8284271247393917	0.10
$x^3 - 2$	[0,2]	35	1.259921049902914	0.09
$x\sin(x) - 1$	[0,2]	33	1.114157140837051	0.09
$x^{10} - 1$	[0,1.5]	35	0.999999999992724	0.10
$e^x - 3x - 2$	[2,3]	34	2.125391198816942	0.24

Table 3: Bisection Method Results

As expected, the bisection method converges to the root x^* of all the functions, but on average, takes many more iterations than the other methods. As discussed, the bisection method is robust but slow. We tested a variety of functions for this method, but each took roughly the same amount of iterations so the *log* error plots look similar.

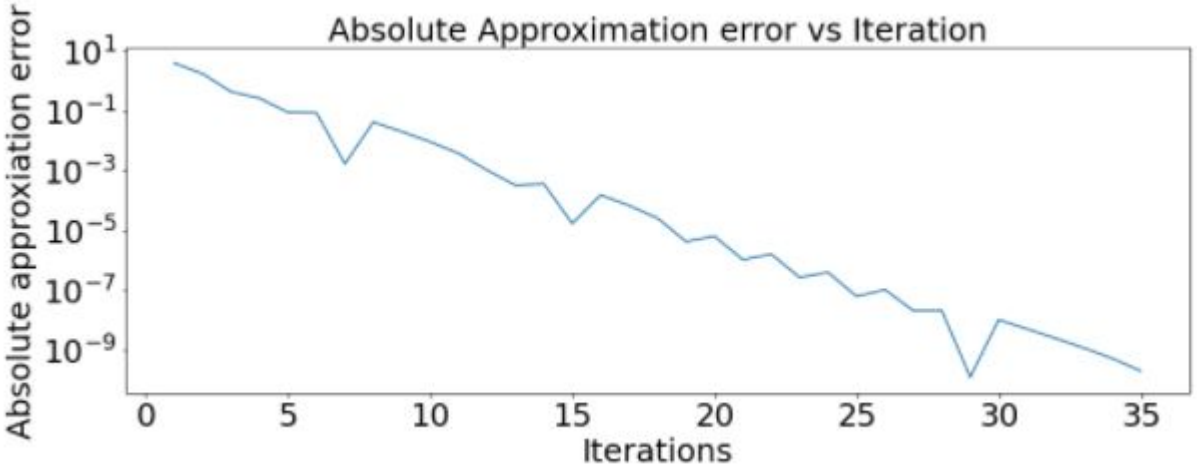


Figure 5: Error function for Bisection

Figure 5 is the *log* error plot for the first function $x^2 - 8$. As we can see, the plot forms a line that decreases at a constant rate, showing that the bisection method converges linearly.

3.2 Fixed Point Method

For this method, we choose to analyze a high-degree polynomial and a trig function to see how Fixed Point Iteration behaves for those types.

We also analyzed a function with a $g'(x) = 0$, a function where Fixed Point oscillates, and a function where Fixed Point diverges.

Function	g(x)	Interval	Init Pt	Iterations	Approx Root(x^*)	Run time (s)
$x^3 - 2$	$\sqrt{2/x}$	[0.1, 2]	.5	35	1.25992105	0.1288
1 + sin(x) - x	1+sin(x)	[0,3]	0	23	1.93456321	0.2533
$(2/3)x+1/x^2 - x$	$(2/3)x+1/x^2$	[0.1,1.5]	1.2	4	1.44224957	0.0578
1+2 sin(x)-x	1+2 sin(x)	[1,3]	2	FAIL	FAIL	FAIL
-16+5x+12/x	-16+6x+12/x	[1,3]	2	FAIL	FAIL	FAIL

Table 4: Fixed Point Method Results

We see that Fixed Point Iteration is less robust than Bisection method, but in general, is able to find a root in fewer iterations.

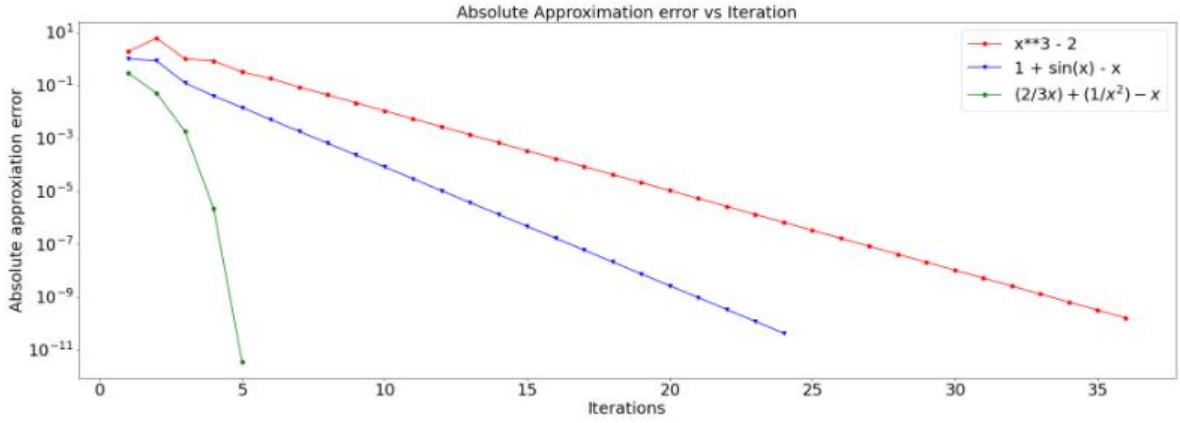


Figure 6: Error function for Fixed Point Iteration

The first three functions we analyzed were successful, however, they have different convergence rates as seen in Figure 6. $x^3 - 2$ and $1 + sin(x) - x$ both have a $g'(p) \neq 0$ and so have linear convergence. $\frac{2}{3}x + \frac{1}{x^2} - x$, though, has a $g'(p) = 0$ and converges much faster with

quadratic convergence.

However, for the last 2 functions, Fixed Point failed. $1 + 2sin(x) - x$ does not have the quality $|g'(x)| < 1$ on the interval $[1, 3]$. However, $|g'(x)|$ is not always greater than 1 on this interval and where it is greater, it is close to one.

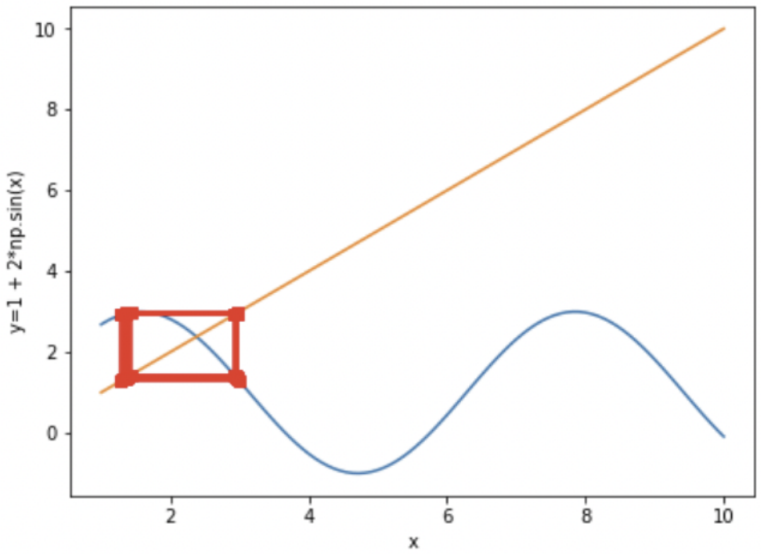


Figure 7: Fixed Point $f(x) = 1 + 2sin(x) - x$

We can see from the figure 7 that it does not form a contractive map, but does not diverge. Instead the algorithm oscillates.

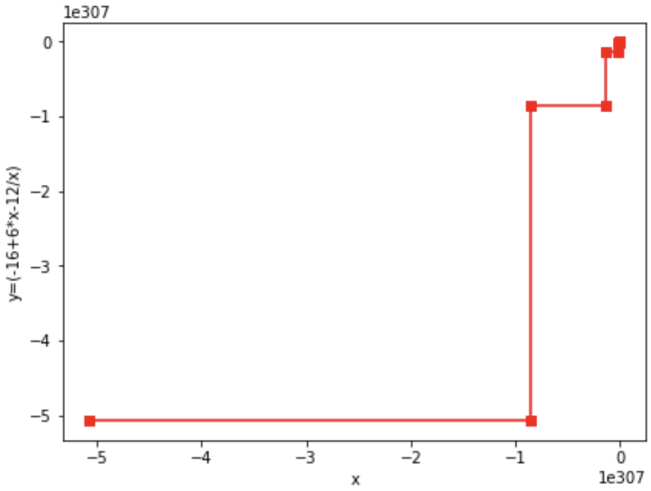


Figure 8: Fixed Point $f(x) = -16 + 6x - 12/x$

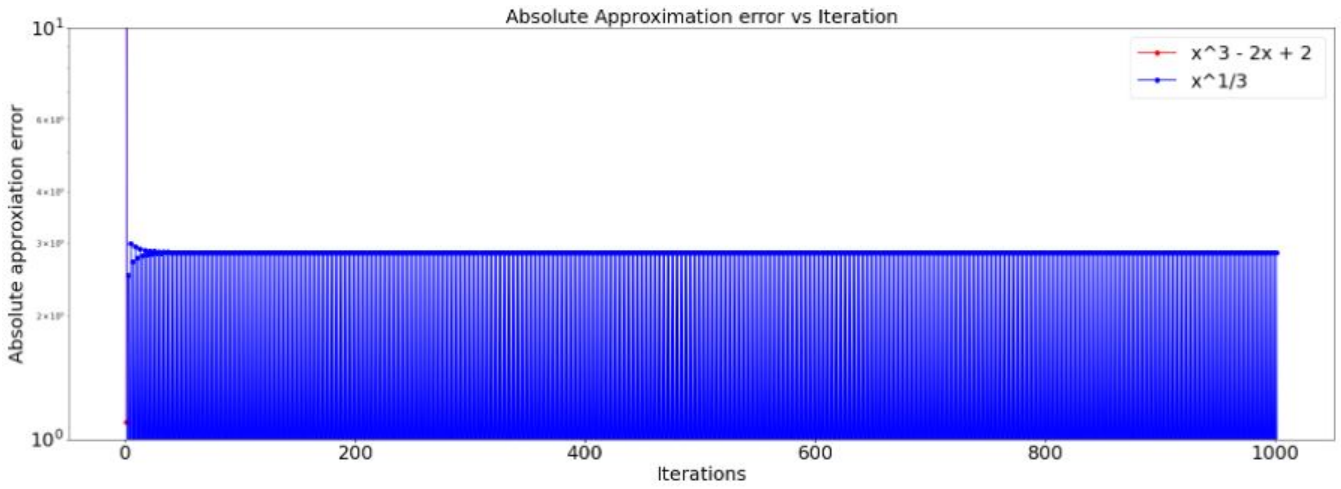


Figure 9: Error function for Fixed Point Iteration

The last function we consider diverges. $-16 + 5x + 12/x$ has a $|g'(x)| > 1$ on the interval $[1, 3]$. It therefore does not form a contractive map as seen in Figure 8 and diverges as seen in Figure 9.

3.3 Newton's Method

For this section, we chose to analyze a high-degree polynomial, a function with a multiple root, two functions that with an initial points far away from the root, a function with a zero derivative initial point, a function that makes Newton's method oscillate, and a function that makes Newton's method diverge.

Function	Initial Point	Iterations	Approx. Root(x^*)	Run time (seconds)
$x^3 - 2$	2	5	1.2599218605	0.0005
$(x-3)^3$	2	19	2.9995489070	0.0017
$x^{10} - 1$	100	48	1.000000000	0.0372
x+cos(2x)-3	1	136	2.5754351242	0.0012
$x^2 - 8$	0	FAIL	FAIL	FAIL
$x^3 - 2x + 2$	0.1	FAIL	FAIL	FAIL
$x^{1/3}$	0.5	FAIL	FAIL	FAIL

Table 5: Newton's Method Results

As we can see Newton's method converges quickly but is not very robust.

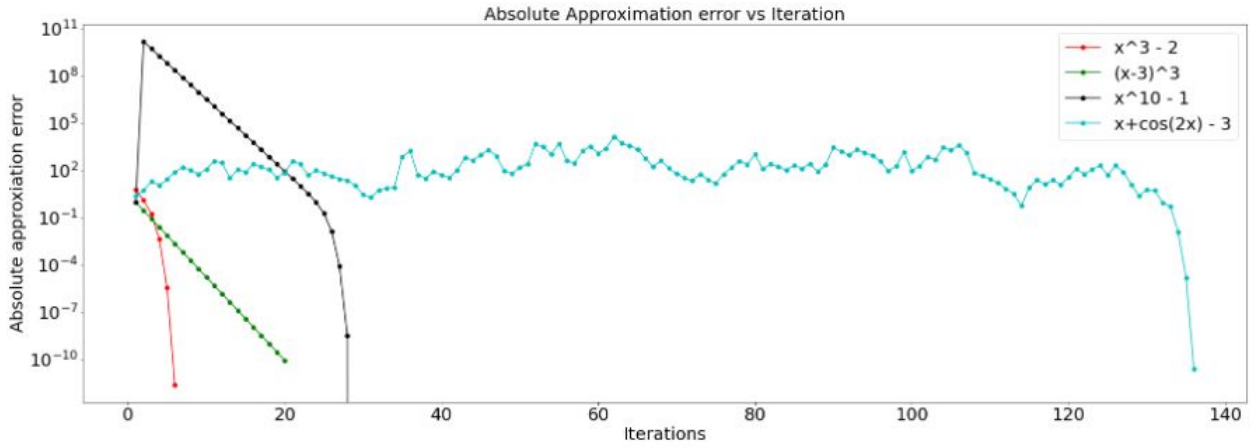


Figure 10: Error function for Newton's method

The first four functions converge, but the $x^3 - 2$ is the only function that posses quadratic convergence as seen in Figure 10. $(x-3)^3$ has multiple roots which decreases Newton's

method to linear convergence. $x^{10}-1$ and $x+cos(2x)-3$ both have initial points that start outside the basin of attraction. They converge, but only after many iterations.

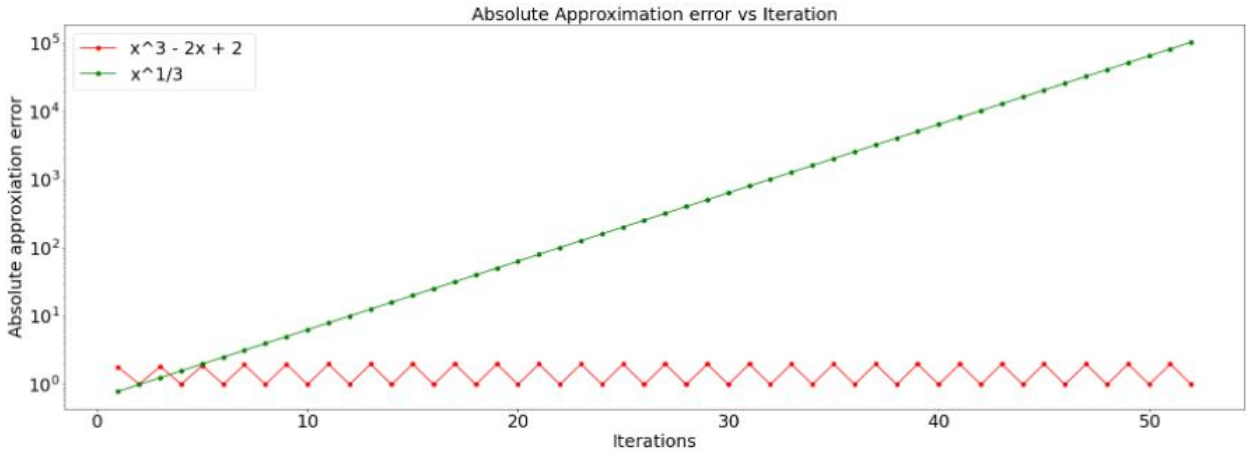


Figure 11: Error function Using Newton for Failed Function

The last three functions failed. x^2-8 at initial point 0 has a derivative of 0. This results in a divide by 0 error and the Newton's method fails. x^3-2x+2 oscillates because of the starting points, making it enter an infinite cycle, preventing convergence. Since the first iteration produces 1 and the second iteration returns back to 0, the sequence oscillates between the two without converging to a root. $x^{1/3}$ diverges since the derivative at the root is being divided by zero resulting in INF.

3.4 Secant Method

For this method we chose to analyze a high-degree polynomial, two functions where Newton failed, a function that causes Secant Method to oscillate, and a function that starts with two initial f(x)'s close together.

Function	Initial Point	Iterations	Approx. Root(x^*)	Run time (seconds)
$x^3 - 2$	0,2	10	1.2599218605	0.002
$x^{1/3}$	0,2	1	0	0.0008
x+cos(2x)-3	0,2	9	2.5754351159	0.0022
$x^{10} - 1$	0.5,1.5	FAIL	FAIL	FAIL
$x^3 - x^2 - x - 1$	0,2	FAIL	FAIL	FAIL

Table 6: Secant Method Results

We see that the secant method is more robust than Newton's method but requires more iterations to arrive at the root.

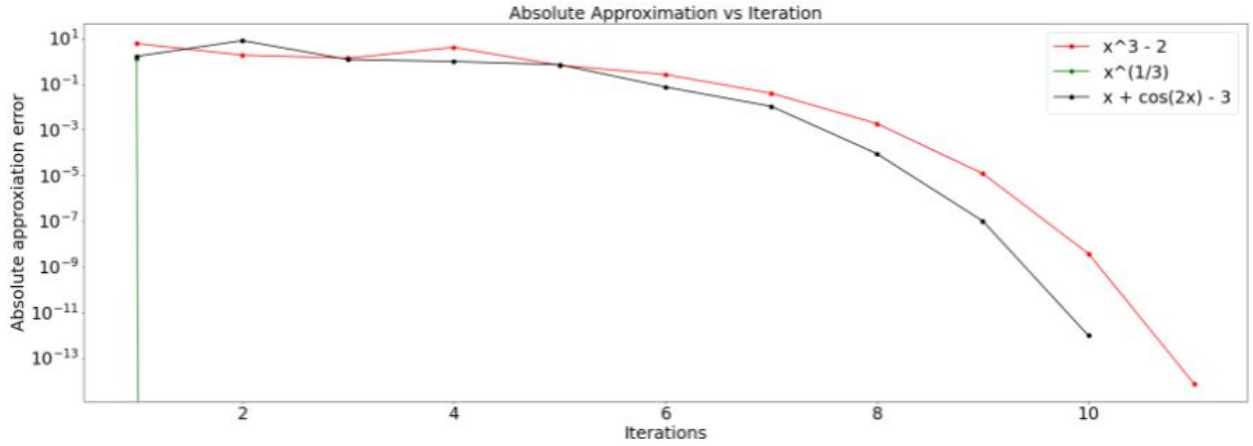


Figure 12: Error function for Secant method

The first three functions were successful. As seen in Figure 12, x^3-2 and $x+cos(2x)-3$ both have superlinear convergence. In Newton's method, $x + cos(2x) - 3$ took many iterations due to it being outside of the basin of attraction. Secant's method was able to find the root after 9 iterations v. Newton's 136 iterations. Similarly, $x^{1/3}$ failed with Newton since the derivative at the root was INF. However, Secant's method does not rely on the derivative and was able to arrive at the root after one iteration.

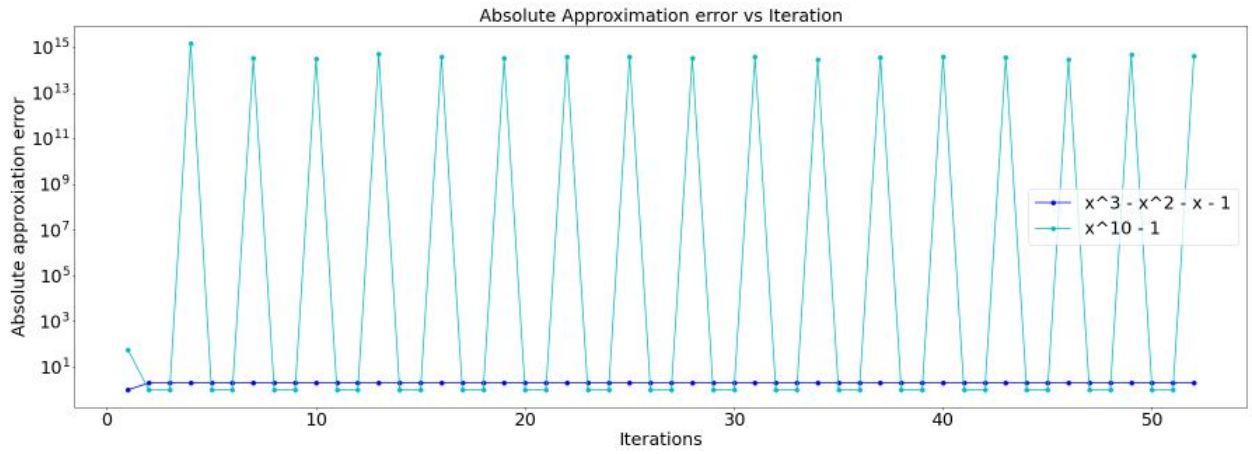


Figure 13: Error function for Secant method

The last two functions failed. As seen in Figure 13, $x^{10} - 1$ oscillates since the slope of the secant line near the root is 0. For $x^3 - x^2 - x - 1$, we choose specific initial points that cause the initial $f(x)$s to be close together. This way, the Secant method is unable to move away from the initial guess and fails.

3.5 Computational Analysis Overview

Method	Interval	Initial Point(s)	Iterations	Approx. Root(x^*)
Bisection method	[0,2]	0,2	35	1.2599210498
Fixed Point method	[0,1]	0.5	35	1.25992105
Newton's method	[-2,2]	2	5	1.2599218605
Secant method	[0,5]	3,4	10	1.2599210498

Table 7: Comparison of the methods

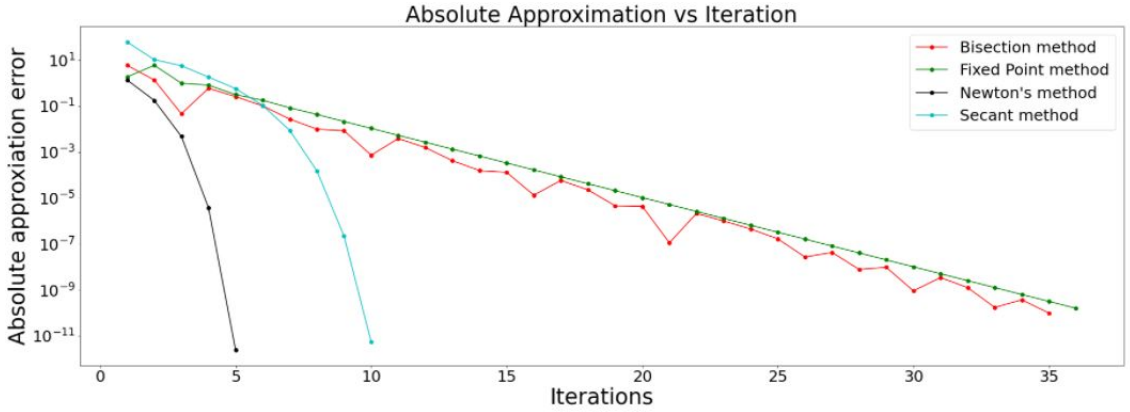


Figure 14: Absolute Error Approx. over the iterations

We tested each methods with the function $x^3 - 2$. As expected, in Figure 14, we see that both bisection method and Fixed Point converge linearly, at 35 iterations, since $g'(p) \neq 0$. Secant method converges superlinearly with 10 iterations, which is slightly slower than Newton's quadratic convergence at 5 iterations.

4 Combination of Methods

Another heavily investigated aspect of root finding techniques is the ability to combine multiple methods to achieve either faster convergence or guarantee convergence. The interesting aspect of this topic is that there is a diverse way to combine the different root finding methods that were mentioned in the previous section. There is an extensive set of literature that discuss combining methods such as letting one algorithm that is know to always converge find an initial guess that is in the basin of convergence for a faster algorithm, adapting algorithms to the unique problem or function we are evaluating, or combining root finding algorithms to decision/optimization algorithms found in computer science. The following section explores the most popular of these Frankenstein approaches to root finding algorithms.

4.1 Bisection and Newton's Method

The combination of the bisection and Newton method is of the most researched topics in this field. The hope behind this combination is to minimize the cons of each algorithm by using what makes each algorithm unique and beneficial. The idea behind this combination is that the bisection method always converges but does so linearly, while Newton's Method

is an algorithm that can achieve a faster quadratic convergence when its initial guess is in the basin of convergence but has no guarantee of converging.

The naïve approach to combining these two algorithms is by letting bisection Method drive the initial guess to a point that is with in the basin of convergence to allow Newton's method to quickly find a solution. There is a couple of issues with this approach, the most notable being that there is no way to figure out when an initial guess is within the basin of convergence for Newton's method.

This section explores the previously mentioned approach of combining Newton and Bisection method and illustrate its improvement over each individual algorithm. To accomplish this, we investigate 4 unique conditions that are known to prevent Newton method from converging quadratically. To illustrate its improvement of speed of the combination method over bisection method we also compare the number of iterations. The goal of this experiment is to reach a root (e.i $f(x) = 0$) or be able to reach an approximation that has an absolute error less than $1e - 10$.

In the algorithm we inputted a function, a range to search for roots, and the derivative of the function. It then takes these inputs and runs the bisection algorithm with exit conditions if it reaches a root (e.i $f(x) = 0$) or was able to reach an approximation that has an arbitrary absolute error less than 1e-2. Finally, it runs Newton method until it reaches our previously mentioned overall goal.

To begin, lets compare the results found in section 3.1 where the bisection method is explored by running the same functions and intervals through this new algorithm. In the table below, it is obvious that this algorithm is able to approximate to $1e-10$ with half to a fourth of the iterations required for bisection methods. However, this does not prove much, since under certain conditions Newton's Method is also able to quadratically converge to solutions.

		Bisection Method		Bisections-Newton Method	
Function	Interval	Iterations	Approximated Root(x*)	Iterations	Approximated Root(x*)
$x^3 - 2$	[0,2]	35	1.259921049902914	13	1.2599210498951001
$e^x - 3x - 2$	[2,3]	34	2.125391198816942	6	2.1253911988111414

Table 8: Bisection Newton combination compared to Bisection method

To really see how this algorithm is an improvement over each individual algorithm we need to see it under conditions that are not optimum for Newtons method and may even cause the algorithm to fail. To do this, the same functions and intervals used in section 3.3 are

used to compare against this combination implementation Newton-Bisection Method. The results of this experiments are displayed in the table 9.

			Newtons Method		Bisections-Newton Method	
Function	Interval	x_0	Iterations	Approx. Root(x*)	Iterations	Approx. Root(x*)
$x^3 - 2$	[0,2]	2	5	1.2599218605659261	12	1.2599210499000755
$x^{10} - 1$	[0,1.5]	1	193	1.000000000008937	13	1.0
$x + cos(2x) - 3$	[0,2]	1	136	2.57543512	1113	2.575435115917459
$x^2 - 8$	[0,3]	0	FAIL	FAIL	12	2.828427124746231
$x^3 - 2x + 2$	[0,1]	0	FAIL	FAIL	2000	1.0
$x^{1/3}$	[0,1]	0	FAIL	FAIL	FAIL	FAIL

Table 9: Bisection Newton combination compared to Newtons method

From the table it is possible to gather that this method is an improvement over using only Newton's method. This algorithm is able to find a solution for the function $f(x) = x^2 - 8$, where it failed before because the initial guess was at an optimum (e.i first derivative equals zero). Similarly, the algorithm is able to give an approximation to the function $f(x) = x^3 - 2x + 2$ where due to the initial guess, Newton's Method oscillates between $x = 0$ and $x = 1$. An approximation for $f(x) = x^{1/3} + 2$ is found because this algorithm is able to avoid a saddle at $(0, 2)$ that also causes Newton's method to oscillate. The bisection method in this combination is able to avoid initial conditions where Newton's Method fails.

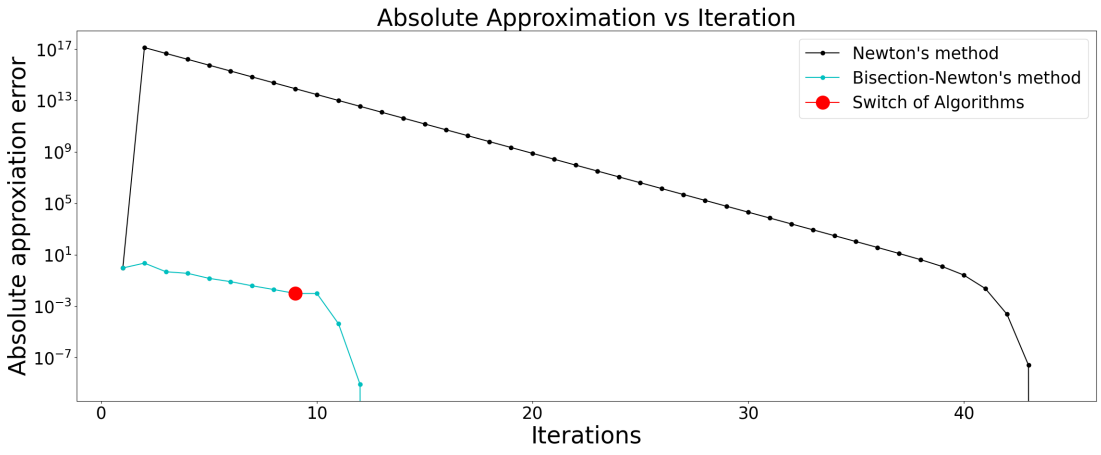


Figure 15: Comparison of Absolute Error Approximation over the iterations

For functions such as $f(x) = x^{10} - 1$ it is able to to drive the initial guess deeper into

the basin of convergence before using Newton's method, allowing for less iterations. Figure 15 shows that if the initial guess is not in the basin of convergence it shots approximation far away from root causing the algorithm to take many more iterations to find the root. This plot also illustrates that for the first 9 iterations the algorithm uses bisection method. Therefore, the convergence is linear. The moment it switches over to Newton's method it begins to converge quadratically.

The first two functions, $f(x) = x^3 - 2$ and $f(x) = x + cos(2x) - 3$, begin to illustrate the downsides to this combination approach. The issue arises when the difference between each iteration of bisection method is arbitrarily small causing slow convergence. This can be improved by using an adaptive algorithm that notices when the bisection method is taking a long time and switches to Newton's method to try to speed up the run time.

Another issue arises when the function that we are analyzing has a root where the derivative is equal to 0 or does not exist such as $f(x) = x^{1/3}$. A function like this makes Newton's method fail because as it get close to the answer it begins getting near the territory of dividing by zero. This becomes a serious issue when there is a function that has multiple roots. However, the given initial point makes the algorithm converge to the root whose derivative is equal to zero. Again, this is an issue that can be remedied by using an adaptive approach that uses Newtons method to get as close as possible to the root and then switches over to bisection method to finish the rest of the approximation.

4.2 Bisection and Secant Method

Another combination that is heavily explored in numerical analysis is that of bisection and secant Method. Similar to the previous section, we are hoping to achieve a faster convergence than bisection method with more stability than the secant method provides. The advantage to this approach is that unlike Newton's Method, Secant method does not require the user to know the derivative of the function whose root they are trying to approximate. The benefits become the most apparent when dealing with functions where finding the derivative is computationally expensive.

Exactly like before, an implementation of a naive combination of Bisection and Secant's method is explored. In the algorithm we inputted a function, a range to search for roots and the derivative of the function. It takes these inputs and runs the bisection algorithm with exit conditions if it reaches a root (e.i f(x) = 0) or was able to reach an approximation that has an arbitrary absolute error less than 1e-2. Finally, it runs Secant's Method until it

reaches a root (e.i f(x) = 0) or is able to reach an approximation that has an absolute error less than 1e-10.

Lets begin by comparing this naive implementation of Bisection-Secant with the same function used in section 3.1.

		Bisection Method		Bisections-Secant Method	
Function	Interval	Iterations	Approximated Root(x*)	Iterations	Approximated Root(x*)
$x^3 - 2$	[0,2]	35	1.259921049902914	15	1.2599210498932347
$e^x - 3x - 2$	[2,3]	34	2.125391198816942	13	2.125391198805407

Table 10: Bisection Secant combination compared to Bisection method

The speed of the combination method over just using bisection method is apparent. The number of iterations required by the combination method is half of that required by Bisection method. Intuitively, this algorithm should be slower than those explored in the Bisection Newton Combination section as the fastest algorithm here is secant's method that converges super-linearly. For the same functions and ranges, this implementation tends to be a couple iterations behind Bisection-Newton's naive implementation.

			Secant's Method		Bisections-Secant's Method	
Function	Interval	x_0	Iterations	Approx. Root(x*)	Iterations	Approx. Root(x*)
$x^3 - 2$	[0,2]	0,2	10	1.2599218605	11	1.2599210498951985
$x^{1/3}$	[0,2]	0,2	1	0	FAIL	FAIL
$x + cos(2x) - 3$	[0,2]	0,2	9	2.57543512	FAIL	FAIL
$x^{10} - 1$	[0.5,1.5]	0.5,1.5	FAIL	FAIL	1	1
$x^3 - x^2 - x - 1$	[0,2]	0,2	FAIL	FAIL	13	1.8392867552067251

Table 11: Bisection Secant combination compared to Secant method

The combination of secant and bisection method improves secant method, but only for certain cases. For functions such as $f(x) = x^{1/3}$ and $f(x) = x + cos(2x) - 3$ secant fails due to other reasons besides the initial guess not being on the basin of convergence and cannot be remedied by a combination of methods. On the other hand with functions such as $f(x) = x^{10} - 1$ and $f(x) = x^3 - x^2 - x - 1$ the combination method does what it is is supposed to do and drives initial guess into the basin of convergence. To improve on this, we would want an adaptive algorithm that switches between methods depending on the current conditions of the function.

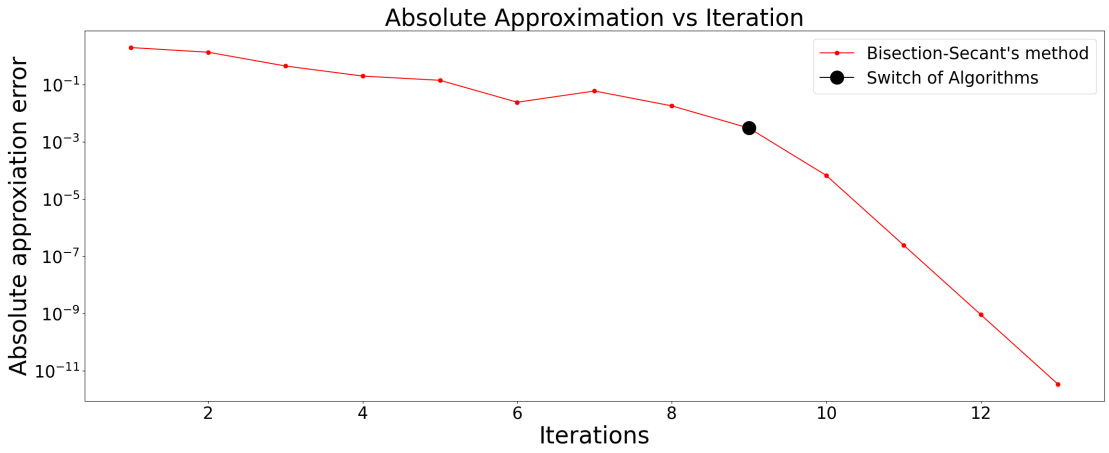


Figure 16: Error Bisection secant combination $x^3 - x^2 - x - 1$

Bisection secant combination is successful for $x^3 - x^3 - x - 1$, where secant alone fails. Figure 16 shows a switch from bisection to secant at the 9th iteration. Here, the convergence rate changes from linear to superlinear.

5 Project Extension

5.1 Introduction

For the project extension, we are exploring three new root finding techniques that were not covered in class. As mentioned earlier, these techniques are the False Position method, Muller's method, and Brent's method. For each of these methods, we look at how the method was constructed, its convergence rate, how the methods are different from ones we previously covered.

5.2 Mathematical Formulation

In this section we cover why the method was chosen, a derivation of the method, the assumptions the method functions under, and the convergence rate of the method.

5.2.1 False Position method

The False Position method is another root finding technique that was not covered in class. However, it is very similar to the bisection and secant method that were mentioned earlier.

False Position method utilizes trial and error to find the root while adjusting the test value based on the outcome.

Derivation [**4**] Similar to the bisection method, a sufficient interval of $[x_1, x_2]$ on a continuous function $f(x)$ is needed such that $f(x_1)$ and $f(x_2)$ have opposite signs. Using the IVT, there's a value $x_3 \in [x_1, x_2]$ such that $f(x_3) = 0$. After this, the bisection method would use the midpoint of the interval $[x_1, x_2]$ and iterate through the algorithm to converge to a root.

Unlike the bisection method, the False Position method creates a tangent line joining the points $(x_1, f(x_1))$ and $(2, f(x_2))$. Since $f(x_1)f(x_2) < 0$, there exists a secant line would cross the x-axis at x_3.

The user can use the equation for slope m of the tangent line to find the value of c.

Using the points $(x_1, f(x_1))$ and $(x_2, f(x_2))$:

$$m = \frac{f(x_2) - f(x_1)}{(x_2 - x_1)}$$

Using the midpoint where $(x_3, 0)$ and $(x_2, f(x_2))$:

$$m = \frac{0 - f(x_2)}{x_3 - x_2}$$

The user can then set the two equations equal to each other and obtain the value of c:

$$\frac{f(b) - f(a)}{b - a} = \frac{0 - f(x_2)}{x_3 - x_2}$$

This gives

$$x_3 = x_1 - \frac{f(x_2)(x_2 - x_1)}{f(x_2) - f(x_1)} = \frac{x_1 f(x_2) - x_2 f(x_1)}{f(x_2) - f(x_1)}$$

This allows for three different locations of where the root could fall. If $f(a)f(c) < 0$, then by IVT, there's a value $c \in [a, c]$ where $f(c) = 0$. However, if $f(c)f(b) < 0$, then there is a value $c \in [c, b]$ where root $f(c) = 0$. Finally, if $f(c) = 0$ then we have arrived at the root. From this, a generalized equation can be created for an iterative scheme:

$$x_{n+1} = \frac{x_{n-1} f(x_n) - x_n f(x_{n-1})}{f(x_n) - f(x_{n-1})} \tag{7}$$

Under these assumptions, the False Position method encapsulates the root within the interval, like bisection method, allowing it to be a robust technique.

Convergence If the function $f(x)$ is convex in the interval of $[x_1, x_2]$ containing the root, then one of the points x_1 or x_2 is fixed as the other point varies. Since x_1 is fixed as the end point, the function continues to make a secant approximation through $(x_1, f(x_1))$ and $(x_n, f(x_n))$
Using (8) from [**7.3**] in the Appendix:

$$e_{n+1} = Me_n^{\alpha}$$

where ξ is assumed to be the simple root of $f(x) = 0$

$$M = \frac{1}{2}\frac{f''(\xi)}{f'(\xi)}$$

and $e_1 = x_1 - \xi$ is dependent on n creating:

$$e_{n+1} = M'e_n^{\alpha}$$

where $M' = Me_1$, showing that the asymptotic error is constant and that False Position Method has linear order of convergence. While bisection method has a fixed convergence rate of $\frac{1}{2}$, the False Position method has a multiplier of M that is dependent on the function.

[**2**]

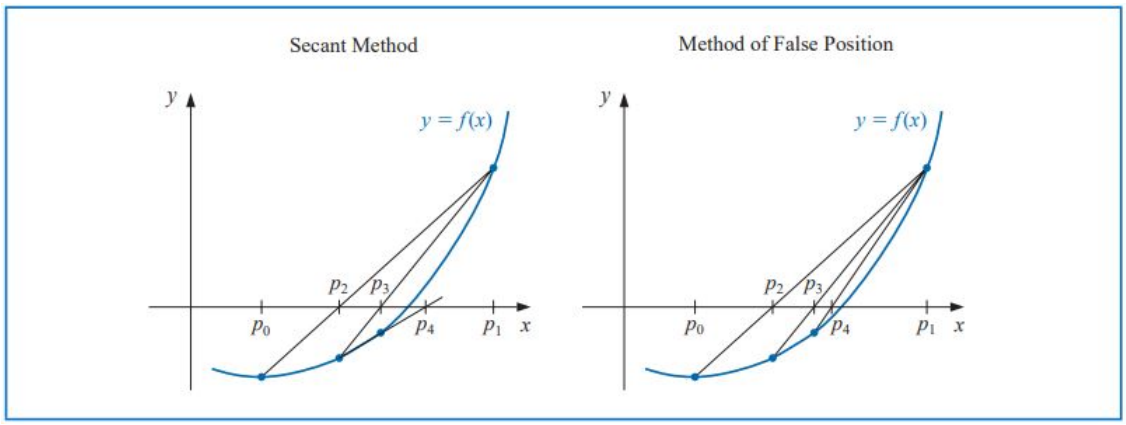
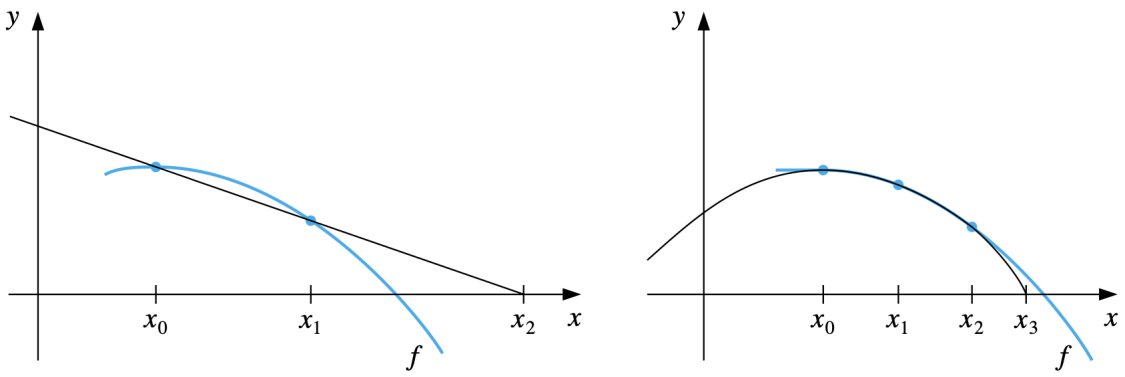


Figure 17: False Position method

5.2.2 Muller's method

Muller's method utilizes the benefits of Newton's and Secant method. While Newton's method has quadratic convergence, it is expensive to compute. Secant method is cheaper, but converges superlinearly. Muller's method utilizes the advantages of both methods, converging faster than the Secant method and costing less than Newton's method. However, maybe the most important advantage of Muller's and the main reason we are exploring it in this paper, is it's ability to find complex roots of polynomials. Specifically, Muller's does not require a complex initial guess to return a complex root. Further, Muller's Method can take in initial values that leads to complex a, b, c values. This is due to the presence of $\sqrt{b^2 - 4ac}$.

However, Muller's method is most similar to Secant method in that it follows the same algorithm but uses quadratic interpolation instead of linear. For each iteration, instead of a secant line between two points, Muller's method uses a parabola between three points as shown in Figure 17. After the equation of the parabola is found, Muller's method uses the quadratic equation to find its roots. This root becomes the third interpolation point, and the most previous interpolation point is dropped. In other words, if the first three points are p_0, p_1, p_2, the root of interpolating parabola is p_3. Then, the next iteration uses p_1, p_2, p_3 as the interpolation points. This process repeats until a tolerance is met and the root of the original function is found.



[2]

Figure 18: Secant (a) v. Muller (b)

Lemma 6. ***Muller's Method*** *[1] Let $f : \mathbb{C} \to \mathbb{C}$ be an analytic complex valued function.*

Let $p_0, p_1, p_2 \in \mathbb{C}$ be three initial guesses for $c \in \mathbb{C}$ such that $f(c) = 0$. For $n \geq 3$,

$$p_n = p_{n-1} - \frac{2c}{b+sgn(b)\sqrt{b^2 4ac}}$$

where

$$a = \frac{(p_{n-2}-p_{n-1})[f(p_{n-3})-f(p_{n-1})]-(p_{n-3}-p_{n-1})[f(p_{n-2})-f(p_{n-1})]}{(p_{n-3}-p_{n-1})(p_{n-2}-p_{n-1})(p_{n-3}-p_{n-2})}$$

$$b = \frac{(p_{n-3}-p_{n-1})^2[f(p_{n-2})-f(p_{n-1})]-(p_{n-2}-p_{n-1})^2[f(p_{n-3})-f(p_{n-1})]}{(p_{n-3}-p_{n-1})(p_{n-2}-p_{n-1})(p_{n-3}-p_{n-2})}$$

$$c = f(p_{n-1})$$

and $\lim_{n\to\infty} p_n = c$

Derivation Let $(p_0, f(p_0)), (p_1, f(p_1)), (p_2, f(p_2))$ be the three initial guesses. To get p_3 we start with the quadratic polynomial

$$P(x) = a(x - p_2)^2 + b(x - p_2) + c$$

The polynomial must go through the initial guesses, so we get the three equations

$$\begin{aligned} f(p_0) &= a(p_0 - p_2)^2 + b(p_0 - p_2) + c \\ f(p_1) &= a(p_1 - p_2)^2 + b(p_1 - p_2) + c \\ f(p_2) &= a(p_2 - p_2)^2 + b(p_2 - p_2) + c = c \end{aligned}$$

Using $f(p_2) = c$, the above equations can be rewritten as

$$\begin{bmatrix} (p_0 - p_2)^2 & (p_0 - p_2) \\ (p_1 - p_2)^2 & (p_1 - p_2) \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} f(p_0) - f(p_2) \\ f(p_1) - f(p_2) \end{bmatrix}$$

Solving for a, b yields

$$P = p_2 - f(p_2)[\frac{p_0 - p_2}{f(p_0) - f(p_2)}]$$

Finding the zero of the polynomial, p_3, gives

$$p_3 = p_2 - \frac{2c}{b+sgn(b)\sqrt{b^2-4ac}}$$

Generalizing this equation leads to the formula of Muller's method.

The convergence rate of Muller's method can be found through a similar proof to that in Section 2, Secant method. However Muller's method incorporates the past three errors giving $e_{n+1} \propto e_n e_{n-1} e_{n-2}$ since this method utilizes three interpolation points. In a similar way to the proof mentioned above, we get the cubic $F(p) = p^3 - p^2 - p - 1 = 0$. Solving for the root gives 1.84, Muller's rate of convergence. It is faster than the secant method and slower than Newton's method. However Muller's method does not always converge. Similar to Newton's and Secant method, Muller's method only converges if the initial guess is close enough and if the initial values result in dissimilar $f(x)$s.

5.2.3 Brent's method

Brent's Method is an another combination root-finding algorithm that has the reliability of bisection but under certain conditions it can be as quick as some of the less-reliable methods. The idea behind this method is similar to that of Muller's Method where a parabola is interpolated from three points and it is used to approximate the root for for the given function. However, unlike previous methods discussed, this algorithm attempts to eliminate all edge cases that would cause it to fail by slowly failing back to the secant method, bisection method to finally forcing a bisection when needed.

Dekker's Method
To begin, we explore an algorithm that is usually referred to as Zeroin or Dekker's Method that was first introduced by T.J Dekker in 1969. The idea behind this Method is that it tries to use the secant since it has a super linear rate of convergence, but falls back to bisection method when needed. [**7**]

Lets assume that we want to solve for a root e.i f(x) = 0. We have two points like bisection method, such that $f(a_0)$ and $f(b_0)$ have opposite signs. Just like bisection method, if f is continuous on $[a_0, b_0]$, then, by the IVT, a solution exists in between those two points. We want to keep b_i as the better choice which we define as:

$$|f(a)| < |f(b)|$$

Let us also declare a variable c which keeps track of the previous better value (b_{i-1}). We keep a_i as a counterpoint (e.i a point such that $f(a_i)$ and $f(b_i)$ have opposite signs) such that the intermediate value theorem holds true).

At the beginning of each iteration we calculate the midpoint of the range and call that variable M and calculate the secant secant step between those two points with the variable name s. This leads to two possible outcomes. The first case is that s, the secant line, is between M and b. We would then want to make our new b (e.i b_{i+1} equal to S). The second case is that s is not between the midpoint and b and we therefore would make M the new b.

From here, the points would be updated such that the previous conditions are met. We would keep track of the previous best choice in the variable c. We would also choose a new contraption such the a_i and b_i have opposing signs. The better choice condition also needs to be met, and, therefore, a_i and b_i would be swapped.

The benefit of this algorithm is that it does not require the function f to be smooth like Secant's method for it to function. It does however benefit from the convergence rate of Secant's method when a smooth function is encountered. There is no identifiable proof for the rate of convergence but it is safe to assume that this algorithm, in the worst case, has a convergence rate $\frac{1}{2}$ and in best case have Secant's $\frac{1+\sqrt{5}}{2}$ convergence rate.

Derrek's Method does well when the function it is finding is well behaved. However, it does fall into scenarios where our best choice b_i converges very slowly (e.i the difference between b_i and b_{k-1} is arbitrarily small).

5.2.4 Project Extension Method Summary Table

Method	Assumptions for Convergence	Convergence Rate	Order of Convergence
False Position method	• Function is continuous on interval [a,b] • $f(a)f(b) < 0$, satisfying the Intermediate Value Thm • if f(a) are f(b) too similar, divide by close to 0 or 0	Function Dependent	Linear
Muller's method	• initial guess falls in basin of convergence • initial parabola not flat	1.84	Superlinear
Brent's method	• Function is continuous • Given interval [a,b], f(a) and f(b) have different signs	$1.0 \leq \mu \leq 1.86$	Variable

Table 12: Project Extension Method Summary Table

5.3 Computational Analysis

In this section, we look at how the False Position method, Muller's method, and Brent's methods are implemented. We analyze their performance using functions that showcase each methods strengths and weaknesses and consider how many iterations it takes to converge successfully. This demonstrates what makes these methods unique as well as some advantages or disadvantages these methods have over the techniques we have already covered in class.

5.4 False Position Method

For this section we analyze the same functions used to evaluate bisection method, since the methods are similar to each other.

Functions	Interval	Iterations	Approximated Root(x^*)	Run Time (seconds)
$x^3 - 2$	[0,2]	16	1.2599210498	0.0068
x sin(x) - 1	[0,2]	6	1.1141571408	0.0012
$x^{10} - 1$	[0,1.5]	309	0.9999999999	0.0731
$x^3 - x^2 - x - 1$	[0,2]	14	1.8392867551	0.0021
$e^x - 3x - 2$	[2,3]	33	2.1253911988	0.0045

Table 13: False Position method

As mentioned, the False Position method has linear order of convergence, while the rate of convergence is function dependent. As expected, some of its approximations out perform the previous methods that were mentioned.

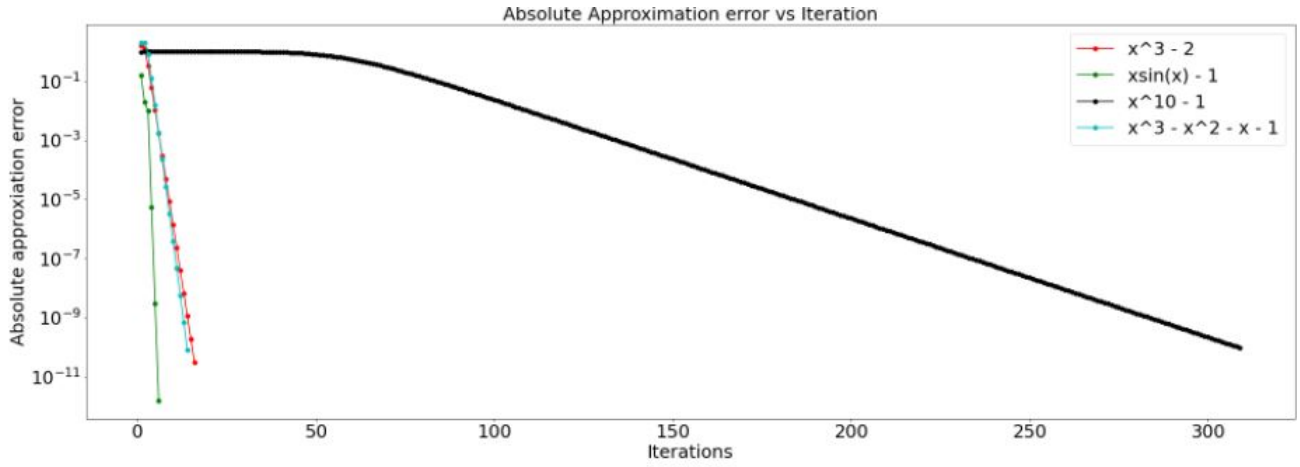


Figure 19: Error function for Regula Falsi

Looking at figure 18, most of the functions out perform its predecessor, the bisection method, converging at almost half the bisection's iterations. However, functions like $x^{10} - 1$ require

more iterations since the chosen points on the functions create a secant line that is further away from the basin of attraction.

5.4.1 Muller's Method

For this section, we consider a high-degree polynomial, a trig function, and an exponential function to showcase Muller's method's robustness. We also consider a polynomial with a complex root, a polynomial with a complex initial value, and a case where Muller's method does not converge.

Function	Interval	Initial Point	Iterations	Approximated Root(x^*)	Run Time (s)
$x^3 - 2$	[0,2]	0,1,2	5	1.2599091297928626	.0034
x sin(x) - 1	[0,2]	0,1,2	5	1.1142	.0021
$e^x - 3x - 2$	[2,3]	2, 2.5, 3	5	2.125400043021506	.0024
$x^2 + 2$	[0,2]	0,1,2	1	0+1.4142i	.0007
$\sqrt{x+1}$	[-2,2]	-2,0,2	882	1-3.4007e-21i	.3787
sin(x)-1	[-5, 15]	$\frac{-\pi}{2}$, $\frac{3\pi}{2}$, $\frac{7\pi}{2}$	1	Inf	.0006

Table 14: Muller's Method Results

As shown by the results above, Muller's Method is very powerful. Not only was it able to successfully find a root at almost every polynomial, it did so in a relatively few number of iterations. In addition, x^2+2 has a complex root but Muller was able to find a solution after 1 iteration.

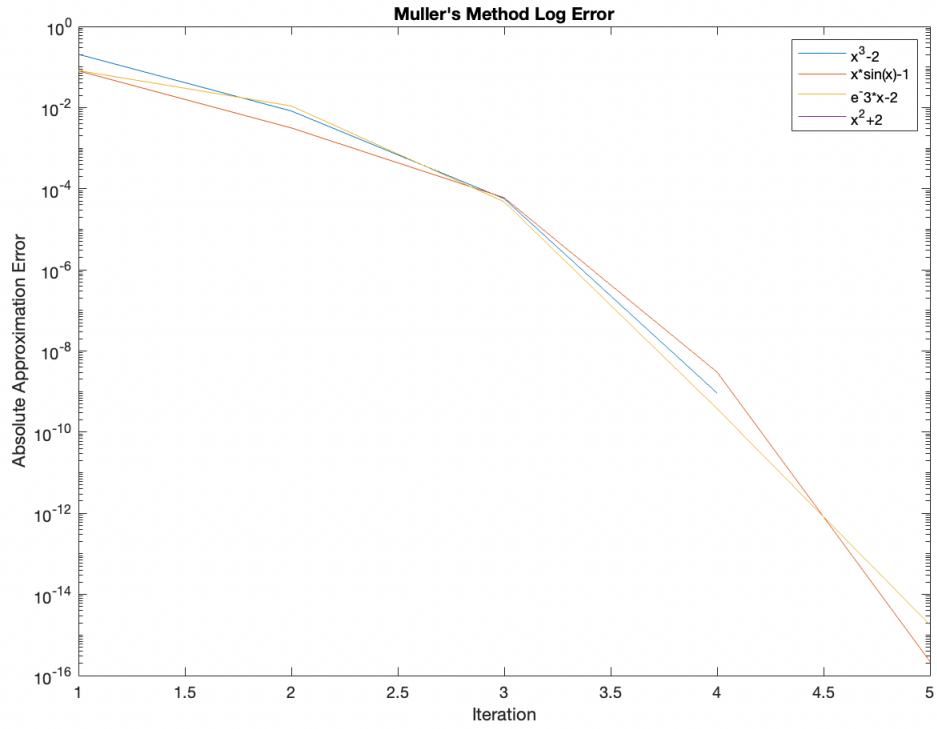


Figure 20: Error function for Muller's

The above plot shows the error of Muller's method when it was successful. Each function has a similar convergence rate, 1.84.

However, for last two cases Muller's method was not successful. $\sqrt{x+1}$ is complex at the initial point -2, which made interpolating the quadratic slow. Even though Muller's method is able to converge for a polynomial with a complex initial value, it does so after many iterations.

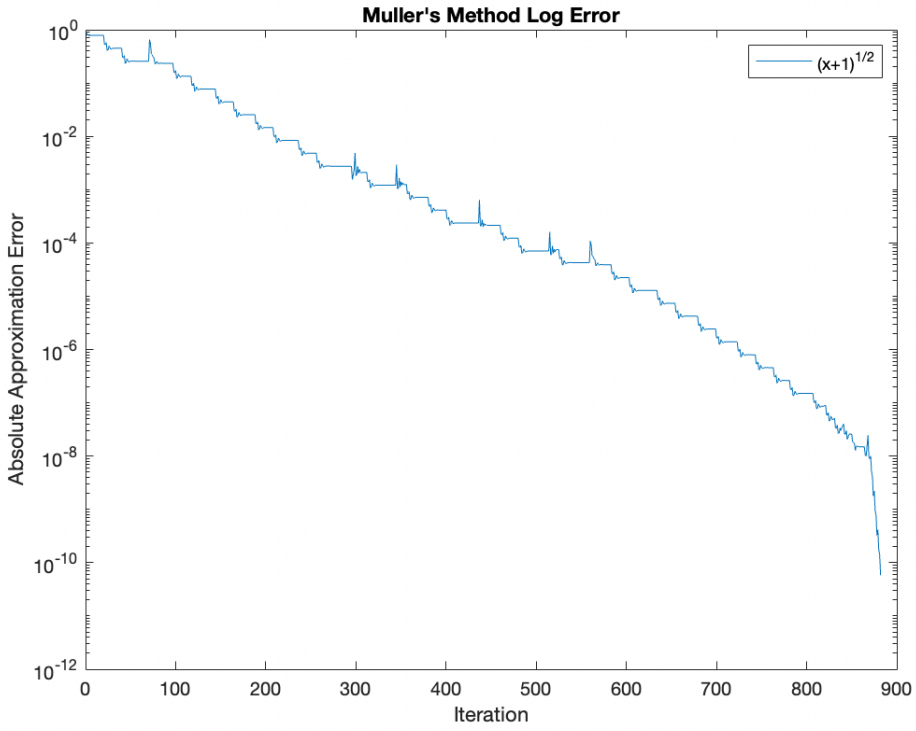


Figure 21: Error function for Muller's $\sqrt{x+1}$

Figure 20 shows that the convergence rate for $\sqrt{x+1}$ at initial values $-2, 0, 2$ has a much slower convergence rate than the successful functions in Figure 19.

For $sin(x) - 1$, initial points were specifically chosen to create a very flat parabola. Muller's method then fails, since it is not able to move away from the initial three points as it is interpolating a line-like parabola.

5.4.2 Brent's Method

Brent's Method was designed to be a robust method that, in the worst case scenario, would run at least as good as bisection method, and, in general, perform super linearly. Due to the use of Quasi-Newtonian methods, the computation of this method is inexpensive but theoretically can be used on functions whose derivatives are unknown. With multiple different mathematicians working on this problem, Brent's Method is known as one of the most robust and versatile methods. It is so popular that the function $fzero$ in MATLAB, which is the default root finding algorithm, is an implementation of Brent's method. In every

programming language that is used for numerically analysis there is at least one library that implements Brent's method.[**8**]

Multiple functions that where shown to fail with previous methods are used to show the robustness of Brent's Method. The functions were run until they either found the exact solution or achieved and absolute error of at least $1e-10$.

Functions	Interval	Iterations	Approximated Root(x^*)
$(x+3)(x-1)^2$	[-4,4]	16	-2.999999999976921
x^2-8	[1,3]	11	2.828427124800383
$1+2sin(x)-x$	[1,4]	13	2.3800612734140856
x^3-x^2-x-1	[0,2]	8	1.8392867552140784
$x^1/3$	[-.5,1]	36	5.8207647035679574e-11

Table 15: Brent's Method Results

It is obvious with functions such as $f(x)=x^2-8$ that this algorithm is able approximate a root with the same accuracy as bisection method with almost half the number of iterations. Next, the difference between each iteration is plotted to illustrate the convergence of Brent's for each function.

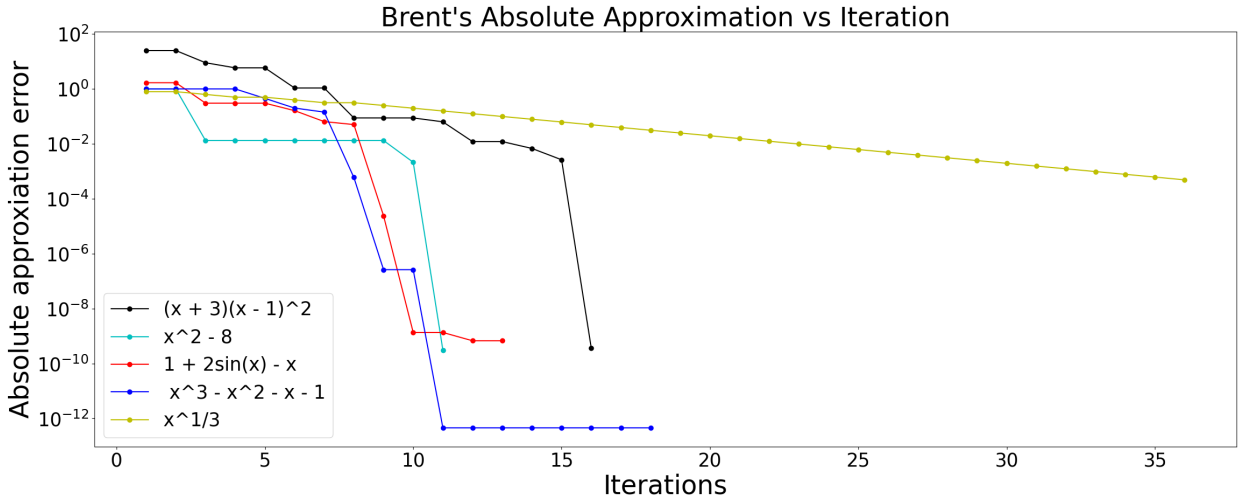


Figure 22: Absolute Error Approximation for Brent's Method with multiple functions

Due to the fact that Brent's method uses multiple methods, each with different rate of convergence, Figure 21 is able to illustrate when Brent's switches between methods. Function $f(x) = (x+3)(x-1)^2$ shows how Brent's method begins by using secant method to approximate until it meets the condition to make a large leap with inverse quadratic interpolation at the 8th iteration. Next, it relies on bisection method for a couple of iterations until again it meets the conditions for inverse quadratic interpolation and quickly reaches a solution with a tolerance of $1e-10$.

On the other hand, the plot of $f(x) = x^{1/3}$ shows how Brent's is stuck using bisection all the way through. Brent derives it robustness from the fact that in the worst-case, it can fall back to bisection method. This also points out one of Brent's largest downsides, if the function and given interval don't at the very least satisfy the condition of bisection method it fails.

Overall Brent's method is shown to be robust and generally faster than using bisection method alone. However, it still far away from being a bullet proof of finding roots.

5.5 Project Extension Overview

In this section, take a closer look at the newly introduced methods and compare them to the methods that were already introduced in class. All the of the methods are analyzed with the function $x^3 - 2$ and tolerance of $1e-10$.

Method	Interval	Initial Point(s)	Iterations	Approx. Root(x^*)
Bisection method	[0,2]	0,2	35	1.2599210498
Fixed Point method	[0,1]	0.5	35	1.2599210536
Newton's method	[1,3]	2	5	1.2599218605
Secant method	[0,5]	3,4	10	1.2599210498
False Position method	[0,1.5]	0,1.5	16	1.2599210498
Muller's method	[0,2]	0,1,2	5	1.2599091297
Brent's method	[0,2]	0,2	12	1.2599210498

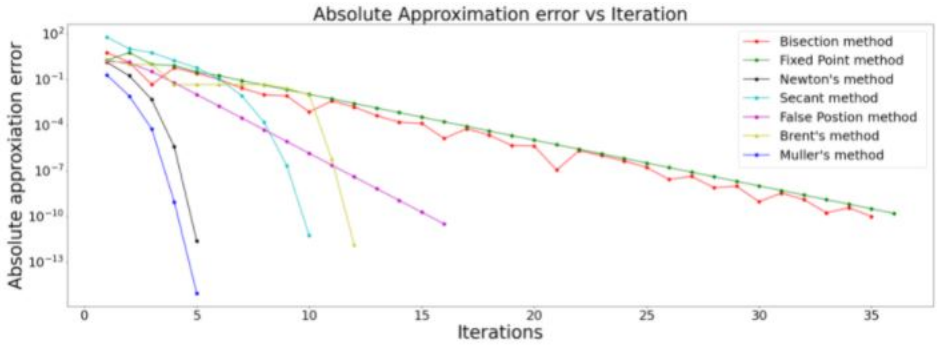


Figure 23: Absolute Error Approximation of All method on function $x^3 - 2$

Figure 23. is a comparison of all the methods examined in this paper using $x^3 - 2$. Muller's and Newton's method are the fastest. This is most likely due to the initial point. However, it was expected that these methods would be the fastest. Brent's and secant methods are the second fastest. Brent's method is a bit slower, which was expected since Brent's method has the same convergence rate as secant method in the best case. False Position is the next fastest. As mentioned it should be faster than bisection method, but still relatively slow. The two slowest methods are bisection and Fixed Point method which converge linearly.

6 Conclusion

Scalar Root finding is a widely applicable and powerful topic. There exists many different root finding techniques that have a variety of application in fields such as engineering, statistics, and physics. Since it is so broadly used, it is important to understand the strengths and weaknesses of these different techniques and when they should be applied to achieve optimal performance.

In our project, we analyzed bisection method, Fixed Point method, Newton's method, secant method. We then analyzed a combination of Bisection and Newton and a combination of Bisection and Secant. Finally, in our project extension we considered False Position method, Muller's method and Brent's method.

6.1 Analysis of Results

We found that under the right assumption, the bisection method was robust but had the slowest convergence. Fixed Point could converge faster than Bisection but was less robust, depending on the derivative of the function. Newton's method had the fastest convergence when the function and its derivative are continuous and the derivative is obtainable. While the secant method was more robust than Newton's method, however, it has a slower convergence order.

The analysis also concludes that, if the initial point is close to the root, the function is continuous, and the derivative is cheap to calculate, Newton's method should be used. However if the derivative is not cheap to calculate quasi-Newton methods such as Secant's method should be used.

These results were expected, since we had already analyzed these different methods in the APPM 4650 course.

However, Newton's method and Secant's method can be improved by combining with Bisection. Using bisection to get into the basin of convergence and then switching to Newton's method was successful where using Newton's method alone was not. However, the case where the derivative at the root was INF and the case of x+cos(2x)-3 still failed with the combination.

Similarly, using bisection and secant is not always an improvement over using secant alone. For $x^{1/3}$ and $x + cos(2x) - 3$, the combination drove the initial range out of the basin of convergence. However, for $x^{10} - 1$ and $x^3 - x^2 - x - 1$, it was able to converge where secant alone was not.

For our project extension, we examined three different methods not introduced in the APPM 4650 course: False Position method, Muller's method, Brent's method. False Position is similar to bisection, and like Bisection it is robust but slow. Muller's method is similar to secant method but utilizes three points for quadratic interpolation instead of two. It has faster convergence than secant method but is not as expensive as Newton. In addition, it can be used to find complex roots. It is fast and is more robust than Newton's. Brent's method is a combination of bisection, secant, and inverse quadratic interpolation. It is robust and has the ability converge faster than secant method alone. Both Brent's and Muller's method can be used for a wide variety of functions so long as their assumptions are met.

6.2 Future Work

To further this project, we would like to see how the methods analyzed in the project extension could be improved through a combination of methods. For example, Muller's method does not converge when the initial parabola is too flat. We could test to see if combining Bisection with Muller's method would converge for a flat initial parabola. In addition, since the False Position method is similar to bisection method, we could test to see if combining False Position method with Newton led to a larger improvement than combining Bisection and Newton.

In addition, it would be beneficial to further explore a combination of methods between bisection and Newton and quasi-newton method. It is evident that it would always be an improvement over using Newton or secant alone. This would need to involve an adaptive algorithm that switched between methods depending on the conditions. It could even be improved by using decision algorithm that are found in computer science.

References

[1] Ameen, Taha. "An Analysis of Muller's Method." *American University of Sharjah*, 29 Nov. 2019.

[2] Burden, Richard L., and J. Douglas Faires. Numerical Analysis. 9th ed., Cengage Learning, 2010

[3] Chabert, Jean-Luc. A History of Algorithms. 1999th ed., Springer Science Business Media, 2012, pp. 83–88.

[4] Díez, P. "A Note on the Convergence of the Secant Method for Simple and Multiple Roots." Applied Mathematics Letters, vol. 16, no. 8, 2003, pp. 1211–1215., https://doi.org/10.1016/s0893-9659(03)90119-4.

[5] Press, William H., et al. Numerical Recipes in FORTRAN 77: Volume 1, Volume 1 of Fortran Numerical Recipes. 2nd ed., Cambridge University Press, 1992, pp. 347–55.

[6] Wu, Xinyuan. "Improved Muller Method and Bisection Method with Global and Asymptotic Superlinear Convergence of Both Point and Interval for Solving Nonlinear Equations." Applied Mathematics and Computation, vol. 166, no. 2, 2005, pp. 299–311., doi:10.1016/j.amc.2004.04.120.

[7] Brent, Richard P. "Chapter 4: An Algorithm with Guaranteed Convergence for Finding a Zero of a Function." Algorithms for Minimization without Derivates, Prentice-Hall, Englewood Cliffs, 1973.

[8] Dejon, Bruno, and Peter Henrici. "Finding a Zero by Means of Successive Linear Interpolation." Constructive Aspects of the Fundamental Theorem of Algebra; Proceedings of a Symposium Conducted at the IBM Research Laboratory, Zuerich-Rueschlikon, Switzerland, June 5-7, 1967, Wiley-Interscience, London, 1969.

7 Appendix

7.1 Intermediate Value Theorem (I.V.T)

The Intermediate Value Theorem states that if f is a continuous function on the interval $[a, b]$ and $f(a) < 0, f(b) > 0$, then $f(x) = 0$ at some point $x \in (a, b)$

Proof. The main idea behind the proof is to find the point at which the function f crosses the axis. Let $X = \{x \in [a, b] | f(y) \leq 0 \forall y \in [a, x]\}$. X is therefore not empty since $a \in X$ and $X \subseteq [a, b]$ so X is bounded. Using Completeness Axiom, X has an upper bound of at least α claiming that $f(\alpha) = 0$ □

7.2 Mean Value Theorem

The Mean Value Theorem states that if a function f is continuous on the interval $[a, b]$ and differentiable on the interval (a, b), then there is a number c such that $a < c < b$ and

$$f'(c) = \frac{f(b)-f(a)}{b-a}.$$

7.3 Absolute Error

[**3**] Looking at the absolute error: $error = |x_n - r|$, where x_n is the current initial guess and r is the actual root, calculation of error can be derived as

$$e_{n+1} = Me_n^{\alpha} \tag{8}$$

where α is the order. Higher *alpha* resulting in higher order of convergence, hence less iterations. Move one iteration back:

$$e_n = Me_{n-1}^{\alpha} \tag{9}$$

Divide (9) from (8):

$$\frac{e_{n+1}}{e_n} = \frac{Me_n^{\alpha}}{Me_{n-1}^{\alpha}}$$

$$\frac{e_{n+1}}{e_n} = \frac{e_n^{\alpha}}{e_{n-1}^{\alpha}}$$

$$ln(\frac{e_{n+1}}{e_n}) = ln(\frac{e_n^{\alpha}}{e_{n-1}^{\alpha}})$$

$$ln(\frac{e_{n+1}}{e_n}) = \alpha ln(\frac{e_n}{e_{n-1}})$$

$$\alpha = \frac{ln(\frac{e_{n+1}}{e_n})}{ln(\frac{e_n}{e_{n-1}})}$$

and the convergence rate M can be found:

$$M = \frac{e_{n+1}}{e_n^{\alpha}} \tag{10}$$

7.4 Code For Bisection

```
import numpy as np
import math

def bisection(f,a,b,tol):
    left = float(a)           # left side
    right = float(b)          # right side
    mdpt = (right + left)/2   #the midpoint
    flag = 0                  #flag 0 is okay & 1 is bad
    counter = 0 #counter for run
    if f(left) * f(right) > 0: #check if the given interval is good
        flag = 1               #change to 1 cause bad
        root = None            #bad so no root
        print('Given interval is inadequate. Try a different range.')
        return root,flag
    else:                      #computation didn't fail
        while counter <= 100:  #run the algorithm about 25 times
            if np.abs(f(mdpt)) < tol: #if midpoint is < than tolerance
                return mdpt,counter
            if f(mdpt)*f(right) < 0:  #check interval
                left = mdpt           #pick new 'left' interval
            else:                     #if the root is not there
                right = mdpt          #make new right interval mid.
            mdpt = (right+left)/2    #find new mid & keep on going
            counter = counter + 1    #how many times it is run
    root = mdpt  # that new midpoint is where the root is
    return root,counter
```

7.5 Code For Fixed Point Iteration

```
import numpy as np
import math

def fixedpnt1(f,x0,tol,nMax):
    i = 0    #initialize the counter
    xn = x0  #initialize the starting point
    r = xn   #initialized the root
    flag = 0 #flag just incase it fail
    while i <= nMax: #iterate till max
        if np.abs(f(xn) - xn) < tol: #check for within tolerance
            break;
        i = i + 1   #increase counter
        xn = f(xn)  #make new iteration xn

    r = xn          #reinitialized the root
    return r, i+1   #return
```

7.6 Code For Newton's method

```
def newt_meth(f,df,x0,tol,nmax):
    xn = x0          #designate the first initial point
    for n in range(0,nmax):     #loop through till max iteration
        fxn = f(xn)          #function evaluated at the initial point
        if abs(fxn) < tol:   #check for within tolerance
            print('Found solution after',n,'iterations.')
            return xn        #counter for the number of tolerance
        Dfxn = df(xn)        #check derivative at the point
        if Dfxn == 0:        #make sure the derivative not equal zero
            print('Zero derivative. No solution found.')
            return None
        xn = xn - fxn/Dfxn   #doing the newton's step
    print('Exceeded maximum iterations. No solution found.')
    return xn                #print out the root approx
```

7.7 Code for Secant method

```
def secant_method(f,x0,x1,tol,nmax):
    #Initialize iterates and iterate list
    xnm=x0; xn=x1;
    rn=np.array([x1]);
    fn=f(xn); fnm=f(xnm);
    nfun=2;
    n=0;
    #keep running until it arrives at small enough value or max iterations
    while abs(fn)>tol and n<=nmax:
        dsec = (fn - fnm)/(xn-xnm); # secant slope
        if abs(dsec)>tol: #doing the secant computation
            xnm = xn;
            xn  = xn - fn/dsec;
        else: #if its stil lbigger than tolerance
            #do secant method
            xn = xn - 0.5*(xn-xnm);
            dsec = (f(xn) - fnm)/(xn-xnm);
            xnm=xn;
            xn  = xn - f(xn)/dsec;

        n+=1; #increase counter
        rn=np.append(rn,xn); #append root to list of prev found
        fn=f(xn); fnm=f(xnm); #evaluated function at root and
        nfun+=2;

    r=xn;

    if abs(fn)>tol: #flag to catch covergence
        print('Secant method failed to converge, n=%d, f(r)=%1.1e\n'' %(n,np.
    else: #flag to catch failed convergence
        print('Secant method converged succesfully, n=%d, f(r)=%1.1e' %(n,np.

    return (r,rn,nfun) #returning what was found, root
```

7.8 Code for combined bisection and Newton's method

```
def bisectionNewton(f, df, a, b, nmax=1000):
    xi = None
    i = None
    j = None

    # iteration of bisection
    for i in range(1, nmax + 1):
        xi = (a + b) /2

        if f(xi) == 0:
            print("Bisection Iterations: {}".format(i))
            print("Newton's Iterations: {}".format(0))
            return xi

        if np.abs(f(xi)) < 1e-2:
            break

        if f(a) * f(xi) < 0:
            b = xi
        else:
            a = xi

    #iterations of newton
    for j in range(1, nmax + 1):
        if df(xi) == 0:
            print('Zero derivative. No solution Found')
            return None

        x = xi - (f(xi) / df(xi))

        if f(xi) == 0 or np.abs(f(x) - f(xi)) < 1e-10:
            print("Bisection Iterations: {}".format(i))
            print("Newton's Iterations: {}".format(j))
            return xi

        xi = x

    print("Bisection Iterations: {}".format(i))
```

```
    print("Newton's Iterations: {}".format(j))


    return xi
```

7.9 Code for naive combination of Bisection and Secant's method

```
def bisectionSecant(f, a, b, nmax=1000):
    xi = None
    i = None
    j = None
    # iteration of bisection
    for i in range(1, nmax + 1):
        xi = (a + b) /2

        if f(xi) == 0:
            print("Bisection Iterations: {}".format(i))
            print("Secant's Iterations: {}".format(0))
            return xi

        if np.abs(f(xi)) < 1e-2:
            break

        if f(a) * f(xi) < 0:
            b = xi
        else:
            a = xi

    #iterations of newton
    for j in range(1, nmax + 1):
        if (f(b) - f(a)) == 0:
            print("Secant method Fails.")
            return None
        xi = a - f(a)*(b - a)/(f(b) - f(a))

        if f(xi) == 0 or np.abs(f(xi)) < 1e-10:
            print("Bisection Iterations: {}".format(i))
```

```
            print("Secant's Iterations: {}".format(j))
            return xi

        elif f(a)*f(xi) < 0:
            b = xi

        elif f(b)*f(xi) < 0:
            a = xi

        else:
            print("Secant method Fails.")
            return None

    print("Bisection Iterations: {}".format(i))
    print("Secant's Iterations: {}".format(j))
    return xi
```

7.10 Code for False Position

```
def falsePosition(f,x0,x1,e):
    x2 = (x0+x1)/2 #initial midpoint
    rn=np.array([x2]) #array to hold the estimate root
    step = 1 #iteration counter
    print('\n\n*** Doing False Position Method ***')
    while abs(f(x2)) > e :
        x2 = x0 - (x1-x0) * f(x0)/( f(x1) - f(x0)) #running the False Positio
        print('Iteration-%d, x2 = %0.6f and f(x2) = %0.6f' % (step, x2, f(x2)
        if f(x0) * f(x2) < 0: #check the interval
            x1 = x2 #change new interval when sufficent
        else:
            x0 = x2
        step = step + 1 #increment counter
        rn = np.append(rn,x2); #append the found root the list
    return x2, rn, step #return approx root, the array root,and # of iteratio
```

7.11 Code for Muller's Method

[1]

```
function [] = Mullers_Method(f, p0, p1, p2, Tol, NO)
f=inline(f); %convert string to function
j=1;
while j<NO && abs(f(p2))>Tol %while above tolerance
%and less the number of iterations
    %calcuate a,b,c
    a=((p1-p2)^1*(f(p0)-f(p2))-(p0-p2)^1*(f(p1)-f(p2)))
    /((p0-p2)*(p1-p2)*(p0-p1));
    b=((p0-p2)^2*(f(p1)-f(p2))-(p1-p2)^2*(f(p0)-f(p2)))
    /((p0-p2)*(p1-p2)*(p0-p1));
    c=(f(p2));

    p3=p2-(2*c)/(b+sign(real(b))*sqrt(b^2-4*a*c));
    %calculate p3 with quadratic equation

    p0=p1; %reset values
    p1=p2;
    p2=p3;

    j=j+1;
end

if (abs(f(p2))) > Tol %if tolerance is not met
    disp('Maximum Iterations Reached before Tolerance');
else
    disp(['Tolerance reached in ' num2str(j-1) ' iterations']);
    disp(['p2=' num2str(p2)]);
    disp(['f(p2)=' num2str(f(p2))])
end
end
```

7.12 Code for Brent's Method

```
import numpy as np
```

```
import matplotlib.pyplot as plt

def brents(f, a, b, nmax=1000, tol=1e-10):

    if f(a)*f(b) > 0:
        print("root not bracketed")
        return

    if np.abs(f(a)) < np.abs(f(b)):
        a,b = b,a

    c = a
    d = None
    mflag = True
    iterations = 0
    sol = []

    while iterations < nmax and np.abs(b-a) > tol:
        sol.append(b)
        fa = f(a)
        fb = f(b)
        fc = f(c)

        #inverse quadratic interpolation
        if fa != fc and fb != fc:
            L0 = (a * fb * fc) / ((fa - fb) * (fa - fc))
            L1 = (b * fa * fc) / ((fb - fa) * (fb - fc))
            L2 = (c * fa * fb) / ((fc - fa) * (fc - fb))
            s = L0 + L1 + L2
        #secant method
        else:
            s = b - ( (fb * (b - a)) / (fb - fa) )

        #bisection method
        if ((s < ((3 * a + b) / 4) or s > b) or
            (mflag == True and (np.abs(s - b)) >= (np.abs(b - c) / 2)) or
            (mflag == False and (np.abs(s - b)) >= (np.abs(c - d) / 2)) or
            (mflag == True and (np.abs(b - c)) < tol) or
```

```
            (mflag == False and (np.abs(c - d)) < tol)):

            s = (a + b) / 2
            mflag = True

        else:
            mflag = False

        fs  = f(s)
        d, c = c, b

        if fa * fs < 0:
            b = s
        else:
            a = s

        if np.abs(fa) < np.abs(fb):
            a, b = b, a

        iterations += 1
        #print(a, b)

    return b, np.array(sol), iterations
```