Some naive and computationally polished techniques to approximate roots of equations.

Shubhajit Dey

Indian Institute of Science Education and Research - Bhopal
Madhya Pradesh, India

Abstract

Abstract. We primarily study the Bisection Method, the Fixed Point Iteration Method, the Newton-Raphson Method, the Secant Method, the Method of False Position and some other computational methods and apply these to conduct experiments aiming to approximate roots of equations, doing which algebraically is near to 'impossible'. The target is to minimize the relative error associated with approximation well within the order of 10^{-10} or lesser and to present a hierarchy amongst the considered methods.

1 Introduction

Numerical Analysis is a branch of applied mathematics that studies methods for solving complicated equations using arithmetic operations. It is often so complex that they require a computer to approximate the processes of analysis (i.e., calculus). The arithmetic model for such an approximation is called an **algorithm**, the set of procedures the computer executes is called a **program**, and the commands that carry out the procedures are called **code**. An example is an algorithm for deriving π[3] by calculating the perimeter of a regular polygon as its number of sides becomes very large and similar, theoretically advanced methods.

Many problems in applied mathematics involve solving systems of linear equations, with the linear system occurring naturally in some cases and as a part of the solution process in others. Solving linear systems with up to 1,000 variables is now considered relatively straightforward. For small to moderately-sized linear systems (say, n $\leq$ 1,000), the favoured numerical method is Gaussian elimination and its variants. For larger linear systems, there is a variety of computational approaches depending on the structure of the problem at hand.

Direct methods lead to a theoretically exact solution in a finite number of steps, with Gaussian elimination as the best-known example. In practice, there are errors in the computed value of the solution due to rounding errors in the computation, arising from the finite length of numbers in standard computer arithmetic. Thus, **Iterative methods**[1], which are approximate methods that create a sequence of approximating solutions of increasing accuracy, are used. Numerical analysis is concerned not just with the numerical result of such a process but with determining whether the error at any stage is within acceptable bounds.

This article will explore and explain the working principles of finding roots of equations made of 'nice-enough' functions by the Bisection Method, the Fixed Point Iteration Method, the Newton-Raphson Method, the Secant Method, the Method of False Position and many more. The platform/language used to implement the codes is MATLAB. The style used to explain and introduce experiments in this article is beginner-friendly and can be easily understood by entry-level programmers and mathematics students. Please note that some preliminary mathematical concepts from undergraduate studies are prerequisites.

2 Method 1: The Bisection Method

The first method used to find roots of equations is based on the classic Intermediate Value Theorem and is known as the Bisection Method. In much computer-science standard literature, this method is known as the Binary Search Method.

Methodology : The theory for the method goes as follows:
Suppose f is a continuous function defined on the interval $[a, b]$, with $f(a)$ and $f(b)$ of opposite sign. The Intermediate Value Theorem implies that $\exists p \in (a, b)$ such that $f(p) = 0$.

Although the procedure will work when there is more than one root in the interval (a, b), we assume for simplicity that the root in this interval is unique. The method calls for a repeated halving (or bisecting) of sub-intervals of $[a, b]$ and, at each step, locating the half containing p.

To begin, set $a1 = a$ and $b1 = b$, and let $p1$ be the midpoint of $[a, b]$, that is,

$$p_1 = a_1 + \frac{b_1 - a_1}{2} = \frac{a_1 + b_1}{2}.$$

- If $f(p1) = 0$,then $p = p1$, and we are done.
- If $f(p1) \neq 0$, then $f(p1)$ must have the same sign as either $f(a1)$ or $f(b1)$.
  - If $f(p1)$ and $f(a1)$ have the same sign, $p \in (p1, b1)$. Set $a2 = p1$ and $b2 = b1$.
  - If $f(p1)$ and $f(a1)$ have opposite signs, $p \in (a1, p1)$. Set $a2 = a1$ and $b2 = p1$.

Then repeat.

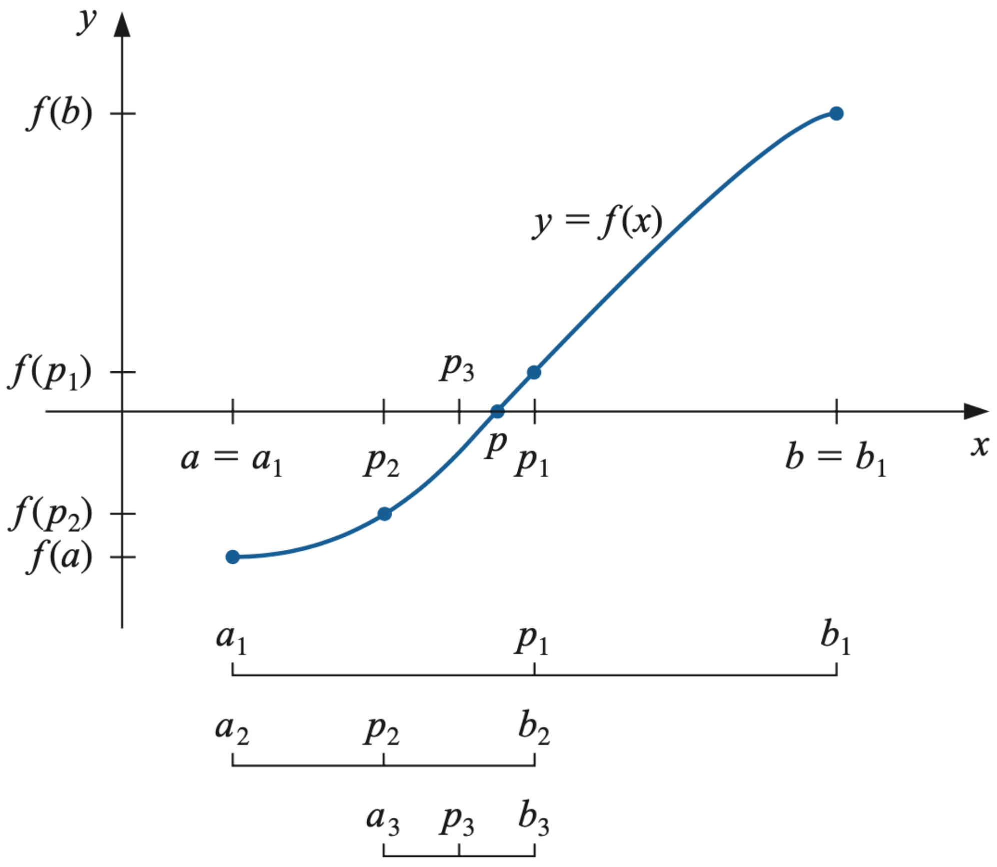

Figure 1: Bisection Method for a function f.

Code : GitHub.

Results : The calculations are for 1.000000e-05 accurate computed solution.

$f(x)$	Interval	Approx. Root	Rel. Error	Time Elapsed (in sec.)	Iterations
$x^4 - 2x^3 - 4x^2 + 4x + 4$	$[1, 2]$	1.4142	7.62939e-06	0.045980	17
$x^2 - 4 * x + 4 + \log(x)$	$[0, 1]$	0.0198	7.62939e-06	0.049912	17
$2x + 3\cos(x) - e^x$	$[1, 2]$	1.2397	7.62939e-06	0.061226	17
$2x\cos(2x) - (x+1)^2$	$[-1, 0]$	-0.7982	7.62939e-06	0.005145	17
$x + 1 - 2 * \sin(\pi x)$	$[-2, 1]$	-1.0000	5.72204e-06	0.004549	19
$x\cos(x) - 2x^2 + 3x - 1$	$[0, 1]$	0.2975	7.62939e-06	0.012767	17
$x - 2^{-x}$	$[0, 1]$	0.6412	7.62939e-06	0.010940	17
$3x - e^x$	$[-1, 0]$	-0.2576	7.62939e-06	0.007315	17
$x^2 - 3$	$[0, 2]$	1.7320	7.62939e-06	0.003862	18
$x^3 + 4x^2 - 10$	$[1, 1.5]$	1.3652	7.62939e-06	0.012280	16

Table 1: Experiment data of Bisection Method applied on $f(x)$.

Notes :

- The function $f(x) = x + 1 - 2\sin(\pi x)$ and the details highlighted by **Gray** in the table above actually has three roots in the mentioned interval (case of multiple roots).
- The function $g(x) = x^2 - 3$ and the details highlighted by **Blue** in the above table actually can be an alternate method to approximate irrationals; in this case, it is $\sqrt{3}$.
- While finding a root in the multiple roots case (**Gray**), the number of iterations needed by the algorithm is comparatively higher in the bundle of examples due to the longer length of the input interval. This factor stands true despite the fact that time elapsed for it accounts for one of the lowest along with the lowest relative error of approximation.
- The case marked in **Red** is used to compare this method with the next method.

3 Method 2: The Fixed-Point Iteration Method

A fixed point for a function is a number at which the value of the function does not change when the function is applied. Mathematically, The number p is a fixed point for a given function g if $g(p) = p$. The method under discussion is called the **fixed-point iteration method** or, in many standard literatures as **functional iteration method**.

This method's concerned theorems and mathematical corners can be found in [1], along with detailed proofs. These include the theorem for the existence and uniqueness of fixed point for a function.

Concept : The primary concept for approximating a root for a given function f is to construct a function g involving the function f in such a way that when $f(p) = 0$, we get $g(p) = p$. Basically, the root of the function f becomes a fixed point for the function g.

A trivial example of such construction can be presented as :

$$g(x) = x - f(x).$$

Later on, we will also see that even the way of constructing the function g will give rise to an efficient method in the context.

Methodology : The first task for implementation is to successfully construct a relevant function g as discussed above for a given function f.

- Next, we select a point $p_0 \in [a, b]$ either randomly or using a smart way!
- We then make a sequence using p_0 obtained in the above state as the initial point. the syntax of convergence goes as $p_n = g(p_{n-1})$, $\forall n \geq 1$.

By the concept of sequential continuity, if the sequence $\{p_n\}$ converges to p then we have $g(p) = p$. Figure (2) provides a visualisation of the algorithm presented.

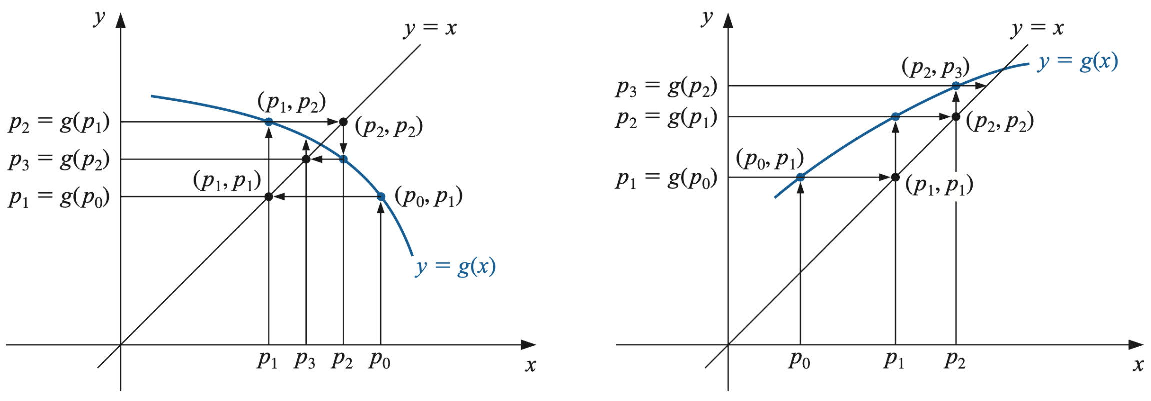

Figure 2: Fixed-Point Method for two functions.

Code : GitHub.

Results : The calculations are for 1.000000e-05 tolerance and a maximum iteration (*maxit*) of 10.

$g(x)$	p_0	p	Rel. Error	Time Elapsed (in sec.)	Iterations
$x - f(x)$	1.5	Divergent p_n	`NaN`	---	---
$\sqrt{\frac{f(x)-10}{x} - 4x}$	1.5	Complex p_n values	`NaN`	---	---
$\frac{1}{2}\sqrt{f(x) - x^3 + 10}$	1.5	1.3649	7.60886e-04	0.012710	10
$\sqrt{\frac{f(x)-10}{4+x}}$	1.5	1.3652	3.64884e-06	0.038816	7
$x - \frac{f(x)}{f'(x)}$	1.5	1.3652	3.67740e-10	0.009563	5

Table 2: Data for $f(x) = x^3 + 4x^2 - 10$.

Note :

- The form of construction for $g(x)$ highlighted by **Green** Table (2) is significantly faster than the other forms of construction. We will provide a deeper explanation for this in the upcoming section.
- Comparing the relative error in approximation for the example marked in **Red** in the table (1) with the **Yellow** marked example in the table (2), we get that the Fixed-Point Method performed slightly better than the Bisection Method.
- But before reaching to a solid conclusion on the hierarchy of method efficiencies, one must note that the efficiency of the Fixed point method depends on the proper guess of the initial point p_0, which in this case is being borrowed from the Bisection method.

4 Method 3: The Newton-Raphson Method

Newton's Method or **Newton-Raphson Method** is one of the most powerful and well-known numerical methods for solving a root-finding problem. In principle, the Newton-Raphson Method is powered up by the concept of Taylor's Series Expansion[2] of a function f.

We assume all the necessary hypotheses and conditions on a function f. We further consider p as the actual root of f and p_0 as the estimated root. The Taylor Series Expansion of f can be put as:

$$\begin{aligned} & f(p) = f(p_0) + (p - p_0) \cdot f'(p_0) + (p - p_0)^2 \cdot \frac{f''(\xi_n)}{2!} + \cdots \\ \implies & f(p) \approx f(p_0) + (p - p_0) \cdot f'(p_0) \\ \implies & f(p_0) + (p - p_0) \cdot f'(p_0) \approx 0 \\ \implies & p \approx p_0 - \frac{f(p_0)}{f'(p_0)}. \end{aligned}$$

Therefore, we have 'roughly' :

$$\boxed{p = p_0 - \frac{f(p_0)}{f'(p_0)}.} \tag{1}$$

Methodology : After absorbing the theory of Taylor's Series Expansion[2] of a 'nice-enough' function and the derivation of equation (1), we can state the algorithm for this method.

Basically, we guess or 'smartly' pick an initial point p_0 and start generating a sequence $\{p_n\}$ as :

$$p_1 = p_0 - \frac{f(p_0)}{f'(p_0)};\ p_2 = p_1 - \frac{f(p_1)}{f'(p_1)},\ \cdots,\ p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})}.$$

One can observe that the sequence-generating function in the Newton-Raphson Method can be represented as a Fixed-Point Function g as :

$$g(p) = p - \frac{f(p)}{f'(p)}.$$

In reality, the series generated in such a way, actually converges very rapidly. A piece of vague evidence can be found at the **Green** marked example in the table (2). Figure (3) visualizes the algorithm for Newton-Raphson Method.

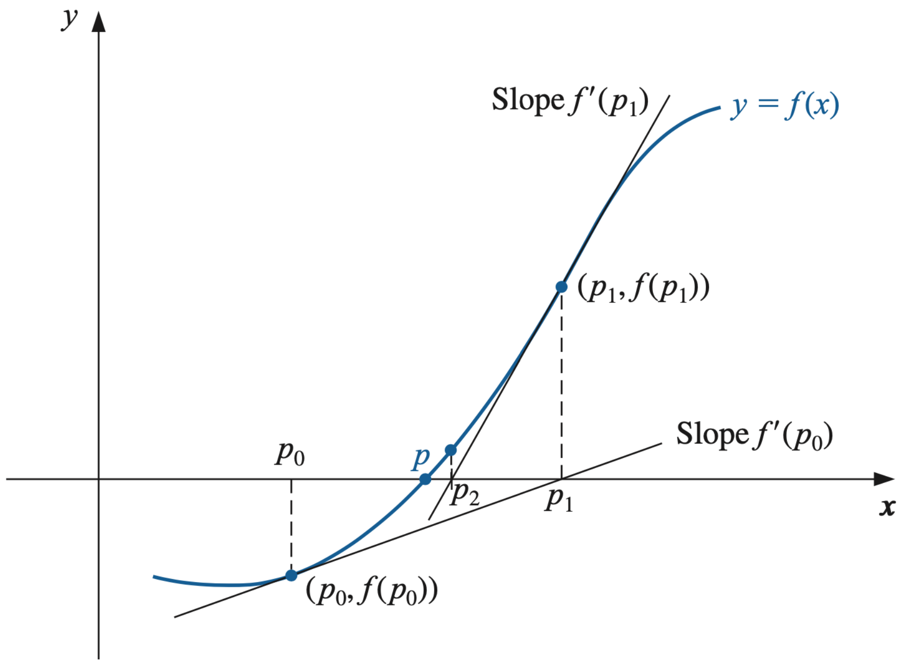

Figure 3: A visualization for the Newton-Raphson Method.

Code : GitHub.

Results :

$g(x)$	p_0	p	Rel. Error	Time Elapsed (in sec.)	Iterations
$\cos(x) - x$	$\pi/4$ [3]	7.3908e-01	6.07381e-08	0.698205	4
$x^3 + 4x^2 - 10$	1.5	1.3652	3.67741e-10	0.416063	5
🟧	🟧	🟧	🟧	🟧	🟧
$x + 1 - 2\sin(\pi x)$	-0.7 -0.5 0 0.5	-1.0000 -2.6820 2.0603e-01 6.8197e-01	1.54562e-07 5.75925e-06 1.03004e-06 6.60004e-10	0.255974 0.425739 0.093583 0.470853	5 6 5 7
🟦	🟦	🟦	🟦	🟦	🟦
$x^2 - 3$	1.5	1.7320	1.41211e-09	0.277350	5
🟨	🟨	🟨	🟨	🟨	🟨
$e^x + 2^{-x} + 2\cos(x) - 6$	1.5	1.8294	6.86587e-09	0.209345	6
$\log(x-1) + \cos(x-1)$	1.5	1.3977	4.57140e-07	0.183929	5
$e^x - 3x^2$	-1.5 1.5 3.5	-4.5896e-01 9.1001e-01 3.7330	1.07786e-05 1.01915e-05 3.65857e-06	0.214719 0.177978 0.173159	6 5 5
$\sin(x) - e^{-x}$	0 3 5 6	5.8853e-01 3.0964 9.4247 6.2850	5.66043e-06 1.01915e-11 3.39398e-08 4.28381e-08	0.337572 0.106676 0.189421 0.117493	5 5 7 4

Table 3: Data of performance by the Newton-Raphson Method for $g(x) = 0$.

Note :

- The function example marked in **Orange** in the table (3), gets almost the same performance data as the case marked in **Green** in the table (2). The reason is obvious. Further, we get that Newton-Raphson is more efficient than Bisection Method.
- In table (3), the case marked in **Teal** can be compared with that marked in **Gray**. While the former produced the multiple roots of the function and also provided much better approximations. Computationally, the number of iterations required by the Newton-Raphson Method is even less than half of those required by the Bisection Method. This clearly justifies the dominance of the Newton-Raphson Method over the Bisection Method.
- Comparing the **Blue** marked results in the table (1) and the **Olive** marked ones in the table (3), we can observe that the Newton-Raphson Method performed well than the Bisection Method to approximate the value of $\sqrt{3}$ in terms of relative error, time complexity and also the number of iterations.
- The main disadvantage of this method, however, is that it requires an appropriate initial guess or a starting point. Also, calculating derivatives might become troublesome in many cases.

4.1 Method 4: The Secant Method

By now, the practical examples presented in section 4 and the theories present in [1]-page[67] have established the fact that the Newton-Raphson Method is a powerful method to approximate roots. But it has its disadvantages, and to mention one of them, is that derivatives can sometimes be complicated to calculate. Thus, to circumvent the problem of the derivative evaluation in Newton's method, we introduce a slight variation in the iteration formula (1). By definition, we have :

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}.$$

Now, if p_{n-2} becomes very close to p_{n-1} by the properties of a Cauchy Sequence, then we would have:

$$f'(p_{n-1}) \approx \frac{f(p_{n-2}) - f(p_{n-1})}{p_{n-2} - p_{n-1}} = \frac{f(p_{n-1}) - f(p_{n-2})}{p_{n-1} - p_{n-2}}. \tag{2}$$

Methodology : Thus replacing the $f'(p_{n-1})$ in equation (1) with the form (2), we get a new iteration function:

$$\boxed{p_n = p_{n-1} - \frac{f(p_{n-1}) \cdot (p_{n-1} - p_{n-2})}{f(p_{n-1}) - f(p_{n-2})}}$$

This forms the key significance of the **Secant Method**, by all means, can be marked as an extension of the Newton-Raphson Method. Figure(4) provides an illustrative visualization of how the method works over iterations. A major difference one can note between the Secant method and the Newton-Raphson Method is that this requires two initial points or guesses rather than only one in the case of Newton's method.

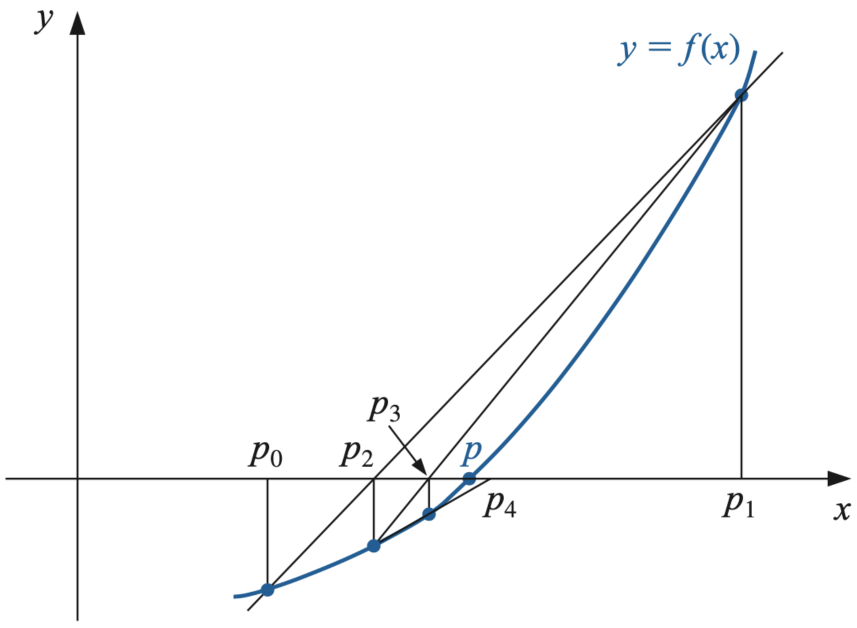

Figure 4: A visualization for the Secant Method.

Code : GitHub.

Results :

$g(x)$	p_0	p_1	p	Rel. Error	Time El. (sec)	Iterations
$\cos(x) - x$	$\pi/4$ [3]	$\pi/3.5$ [3]	7.3908e-01	6.28811e-09	0.006283	5
$x^3 + 4x^2 - 10$	1.5	1.4	1.3652	2.82189e-08	0.013106	5
$x + 1 - 2\sin(\pi x)$	-0.7	-0.8	-9.9999e-01	8.92252e-06	0.012891	5
	-0.5	-0.55	-2.2060	9.82524e-07	0.001029	7
	0	2	-1.0000	2.22045e-16	0.000396	3
	0.6	0.5	6.8197e-01	3.21080e-06	0.000254	7
$x^2 - 3$	1.5	1.6	1.7320	1.16276e-06	0.001729	5
	1.6	1.7	1.7320	6.41766e-07	0.000263	5
$e^x + 2^{-x} + 2\cos(x) - 6$	1.5	1.6	1.8294	9.15833e-06	0.013819	6
	1.5	1.8	1.8294	1.03743e-06	0.000884	5
$\log(x-1) + \cos(x-1)$	1.5	1.4	1.3977	1.10660e-06	0.006258e	4
$e^x - 3x^2$	-1.5	-0.5	-4.5896e-01	1.82577e-05	0.009866	5
	1.5	1	9.1001e-01	6.21544e-09	0.000338	7
	3.5	3.7	3.7330	5.20509e-07	0.000275	5
$\sin(x) - e^{-x}$	0	0.5	5.8853e-01	5.62868e-06	0.002473	5
	3	3.045	3.0964	3.50701e-07	0.000855	4
	9	9.3	9.4247	1.33097e-06	0.000253	4
	6	6.15	6.2850	1.15163e-06	0.000276	4

Table 4: Data of performance by the Secant Method for $g(x) = 0$.

Note :

- Trivial way of comparisons can be done as done in the previous sections, and inferences can be deduced. The significant remark to be observed is that the time elapsed for this iterator is far less than the other methods for producing similar results with comparable accuracy.
- This, in a way, can justify the usage of the theoretical first-order primitive definition of the derivative of a function at a point.
- Nonetheless, the dependency on slower methods like the Bisection Method still persists because of the involvement of not one but two initial guesses. This still remains one of the disadvantages of Newton's Method and its extensions.

4.2 Method 5: The Method of False Position

We have seen that each successive pair of approximations in the Bisection method brackets a root p of the given equation; that is, for each positive integer n, a root lies between a_n and b_n. Root bracketing is not guaranteed for either Newton's method or the Secant method. In Example 1 of the table(3), Newton's method was applied to $f(x) = cosx - x$, and an approximate root was found to be 0.7390851332. Table(3) shows that this root is not bracketed by either p_0 and p_1 or p_1 and p_2. The Secant method approximations for this problem are also given in Table (4). In this case, the initial approximations p_0 and p_1 bracket the root, but the pair of approximations p_3 and p_4 fail to do so.

The method of False Position (also called **Regula Falsi**) generates approximations in the same manner as the Secant method, but it includes a test to ensure that the root is always bracketed between successive iterations. Although it is not a method generally recommended, it illustrates how bracketing can be incorporated.

Methodology : We first choose initial approximations p_0 and p_1 with $f(p_0) \cdot f(p_1) < 0$. The approximation p_2 is chosen in the same manner as in the Secant method, as the x-intercept of the line joining $(p_0, f(p_0))$ and $(p_1, f(p_1))$. To decide which secant line to use to compute p_3, consider $f(p_2) \cdot f(p_1)$, or more correctly $sgn(f(p_2) \cdot sgn(f(p_1))$.

- If $sgn(f(p_2) \cdot sgn(f(p_1)) < 0$, then p_1 and p_2 bracket a root. Choose p_3 as the x-intercept of the line joining $(p_1, f(p_1))$ and $(p_2, f(p_2))$.
- If not, choose p_3 as the x-intercept of the line joining $(p_0, f(p_0))$ and $(p_2, f(p_2))$, and then interchange the indices on p_0 and p_1.
- In a similar manner, once p_3 is found, the sign of $f(p_3) \cdot f(p_2)$ determines whether we use p_2 and p_3 or p_3 and p_1 to compute p_4. In the latter case, a relabeling of p_2 and p_1 is performed. The relabeling ensures that the root is bracketed between successive iterations.

Figure(5) provides an illustrative visualization of how the method works over iterations.

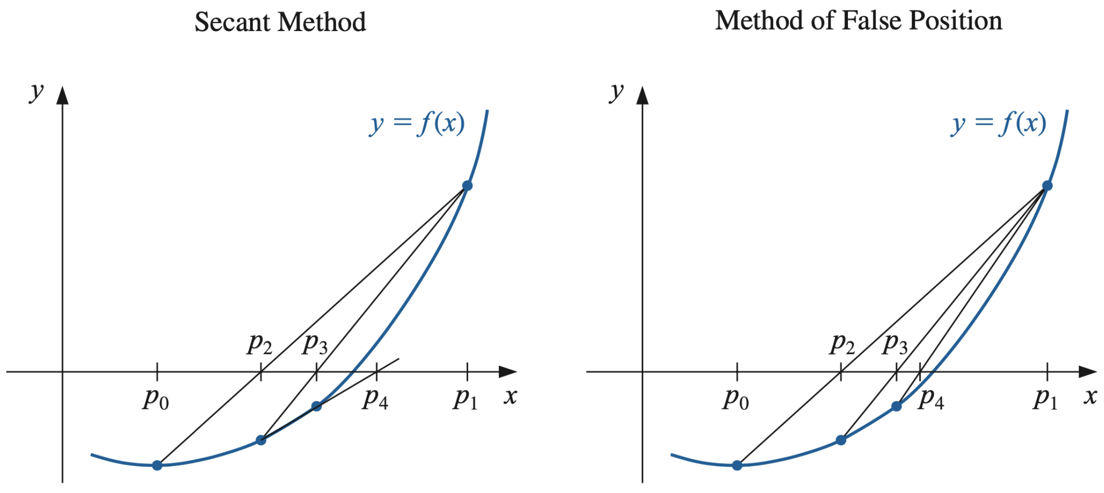

Figure 5: A visualization for the Secant Method as compared to that of the Regula Falsi.

Code : GitHub.

Results :

$g(x)$	p_0	p_1	p	Rel. Error	Time El. (sec)	Iterations
$\cos(x) - x$	0.5	$\pi/4$ [3]	7.3908e-01	3.608676e-07	0.007893	6
	$\pi/5$ [3]	$\pi/4$ [3]	7.3908e-01	1.56376e-07	0.019639	6
🟩	🟩	🟩	🟩	🟩	🟩	🟩
$x^3 + 4x^2 - 10$	1.4	1.3	1.3652	2.29459e-07	0.013970	6
🟦	🟦	🟦	🟦	🟦	🟦	🟦
$x + 1 - 2\sin(\pi x)$	-0.8	-1.2	-1.0000	0	0.004924	4
	-2.75	-2.65	-2.6819	1.61956e-06	0.014446	7
	0.1	0.5	2.06035	8.79322e-07	0.000697	9
	0.5	1	6.8196e-01	2.44309e-05	0.003491	11
$x^2 - 3$	1.5	2	1.7320	3.54157e-06	0.004142	7
$e^x + 2^{-x} + 2\cos(x) - 6$	1.5	1.6	1.8294	3.15050e-06	0.002968	9
	-1	-3	-2.9865	7.80599e-07	0.000254	6
$e^x - 3x^2$	-1	0	-4.5895e-01	7.77070e-05	0.013741	11
	0	1	9.1001e-01	9.06661e-07	0.001847	8
	3.5	4	3.7330	5.50553e-06	0.000275	9

Table 5: Data of performance by the Regula Falsi for $g(x) = 0$.

Note :

- Well, the Regula Falsi method could not perform better than the Secant method or the Newton-Raphson method. This is broadly evident from table (5). To add further, one can even have a dirct head-to-head comparison for the example marked in **Green** and **Cyan** in table (5) with the other methods.

References

[1] A. Burden, R. Burden, and J. Faires. *Numerical Analysis, 10th ed.* Cengage Pvt. Ltd., 01 2016.

[2] S. Dey. On taylor's approximation and some associated theorems on convergence of power series. *Researchgate*, 12 2021.

[3] S. Dey. Some naive and computationally polished techniques to approximate π. *Researchgate*, 01 2023.

* * *

- TOOLS USED -

1.	`overleaf.com` ················	LaTeX platform.
2.	`MATLAB R2022b 9.13` ·········	to implement codes.