RESEARCH ARTICLE

From Theory to Code: Transforming Classical Root-Finding Methods into Efficient Python Implementations

Moh. Nafis Husen Romadani[1,*]

[1] Widya Mandala Surabaya Catholic University, Surabaya 60114, Indonesia

Abstract

This study conducts a comparative evaluation of seven numerical methods for finding roots of nonlinear equations: Bisection, Regula-Falsi, Fixed-Point Iteration, Newton-Raphson, Secant, Aitken's Δ^2, and Steffensen. The aim is to analyze the effectiveness of each method based on convergence speed, numerical accuracy, stability, and computational time efficiency. Algorithm implementation was carried out in the Python programming language using NumPy, SymPy, Pandas, and Matplotlib libraries. Test functions included polynomial, trigonometric, exponential, and mixed functions to represent diverse functional characteristics. The results indicate that Steffensen and Newton-Raphson achieved the fastest convergence in terms of iteration count, while Secant excelled in execution time efficiency. Bracketing methods such as Bisection and Regula-Falsi guaranteed convergence stability despite being slower. Fixed-Point Iteration was highly sensitive to the choice of iteration function, whereas Aitken's Δ^2 served effectively as an accelerator. The study concludes that method selection should be tailored to function characteristics and practical needs, such as derivative availability, computational complexity, and error tolerance. The implication is that this research provides an empirical guide for researchers and practitioners in selecting optimal numerical algorithms for scientific and engineering applications.

Keywords: numerical methods, nonlinear equations, root-finding algorithms, python implementation, computational mathematics.

1 Introduction

In various fields of science, from physics and engineering to economics, there often lies a fundamental problem of finding solutions to equations, or in mathematical terms, finding the roots of nonlinear equations [1]. This problem, formally represented as the search for values where $f(x) = 0$, appears simple conceptually, but in practice, it is often impossible to solve using pure analytical methods [2]. When the functions encountered grow

Submitted: 25 October 2025
Revised: 07 November 2025
Accepted: 25 November 2025
Published: 14 December 2025

Vol. 1, **No.** 3, 2025.
10.62762/JAM.2025.840767

***Corresponding author:**
✉ Moh. Nafis Husen Romadani
nafishusenromadani@gmail.com

Citation
Romadani, M. N. H. (2025). From Theory to Code: Transforming Classical Root-Finding Methods into Efficient Python Implementations. *ICCK Journal of Applied Mathematics*, 1(3), 154–189.

© 2025 by the Author. Published by Institute of Central Computation and Knowledge. This is an open access article under the CC BY license (https://creativecommons.org/licenses/by/4.0/).

in complexity, the path to exact solutions becomes closed, leaving us with the need to find the most adequate approximations. This is where numerical methods emerge as a set of techniques that do not attempt to provide perfect answers in one step, but rather provide a sequence of approximations that systematically converge toward the desired root [3].

The fundamental principles of numerical methods are rooted in profound calculus and real analysis, for example by utilizing fundamental theorems such as the Intermediate Value Theorem and the power of derivatives to build iterative algorithms [3]. Each iteration represents a refinement of the estimate based on mathematical logic, and the iteration sequence is terminated when the accuracy criteria are met [4].

However, in practice, major challenges often arise when the function $f(x)$ we encounter is no longer a simple polynomial, but rather more complex forms such as transcendental functions containing exponential, logarithmic, or other complicated nonlinear combinations [1]. Under these conditions, conventional algebraic methods lose their power; there exists no magic formula that can derive exact solutions [5]. Even for seemingly simple equations, such as $f(x) = e^{-x}$ or $g(x) = \cos(\ln(x))$, closed analytical solutions are almost impossible to obtain. Furthermore, in many cases, functions may have multiple roots, exhibit sharp asymptotic behavior, or even have limited domains, thus complicating the search process. These conditions ultimately form a dividing wall between the elegance of pure mathematical theory and the demands of practical solutions in the real world. Facing this impasse, we require a paradigm shift: rather than striving to find exact solutions that may not exist, we can turn to a more pragmatic and robust approach, namely seeking numerical approximations that converge toward the true root with a level of precision that we can control [2]. This is the promise offered by various numerical methods, which transform static theoretical problems into dynamic iteration-based processes [3].

In general, numerical methods for root finding can be grouped based on their approach. The Bisection and Regula-Falsi methods are examples of safe *bracketing methods*, as they require two initial guesses that bracket the root. Bisection works by consistently dividing the interval into two parts, while Regula-Falsi is more intelligent by creating a straight line (secant) between two points to estimate the root position, thus often converging faster [1]. On the other hand, there are *open methods* such as Secant, Newton-Raphson, and Fixed-Point Iteration that only require one or two initial guesses without bracketing the root [4]. Newton-Raphson, famous for its quadratic convergence speed, utilizes function derivative information to form highly efficient approximations, though its dependence on derivatives can be a weakness [6]. If derivatives are difficult to obtain, the Secant method becomes an alternative that uses a finite difference approach. Meanwhile, Fixed-Point Iteration rearranges the equation $f(x) = 0$ into the form $x = g(x)$ and performs direct iteration, although its convergence is not always guaranteed [1, 4].

To accelerate the convergence rate of slow methods such as Fixed-Point Iteration, acceleration techniques like Aitken and Steffensen emerge as solutions. Aitken's Δ^2 method can be applied to any linearly convergent sequence to accelerate it [7], while Steffensen's method intelligently incorporates the idea of Aitken acceleration into the Fixed-Point iteration itself [6], thus achieving quadratic convergence without requiring derivative calculations. Each method brings its own uniqueness, advantages, and limitations, which significantly influence their performance when implemented in computational code.

Although the mathematical principles of these methods provide a clear path toward solutions, their manual implementation inevitably faces significant practical constraints. The iterative process, which must be repeated dozens, hundreds, or even thousands of times to achieve high accuracy levels such as very small absolute error (e.g., 10^{-10}), is not only inefficient but also vulnerable to human error [4]. Each repeated calculation step requires extreme precision, where a small rounding error at the beginning can have a major impact on the final results. This is where the role of modern computing becomes crucial. The Python programming language, which has become the de facto standard in scientific computing, offers an elegant and powerful solution. With its clear syntax supported by a rich ecosystem of modules, Python enables us to transform complex numerical algorithms into efficient and manageable code. Modules like NumPy and SymPy, optimized for numerical computation, ensure mathematical operations run quickly and accurately [8].

With Python, we can fully automate the iterative process, test termination criteria (such as specified

error tolerance), and easily compare the performance of various methods - a luxury not available in manual calculations [4]. Thus, the transition from theory to Python code is not merely about converting formulas into syntax, but represents a quantum leap in efficiency, accuracy, and our ability to solve complex mathematical problems at a scale previously unimaginable.

Based on the aforementioned considerations, this paper aims to conduct a comprehensive and comparative evaluation of the practical effectiveness of seven numerical methods: Bisection, Regula-Falsi, Fixed-Point, Newton-Raphson, Secant, Aitken, and Steffensen. The analysis focuses on three main aspects: first, the effectiveness of methods in finding roots with high precision for various function characteristics (such as linear functions, high-degree polynomials, and transcendental functions) [9]; second, convergence speed measured through the number of iterations required to achieve specified error tolerance and computational execution time in Python; third, identification of fundamental advantages and limitations of each algorithm, such as the need for initial guesses, dependence on derivatives, numerical stability, and divergence risks. Through implementation in Python code, this research not only maps the performance landscape of each method empirically but also strives to provide a practical guide for researchers, engineers, and computing practitioners in selecting the most optimal and robust algorithm according to their specific problem contexts, thereby bridging the gap between abstract theoretical understanding and concrete, efficient application.

2 Related Work

In the early stages of its development, research focus tended to be limited to comparative analysis of basic methods within a relatively narrow spectrum. For example, Thakur and Saini [10] conducted a comparative study of three traditional methods (Bisection, Newton-Raphson, and Regula Falsi) in the context of convergence and numerical root stability. Their findings confirmed the reliability of Newton-Raphson method in terms of quadratic convergence speed, but also revealed its vulnerability to two critical factors: explicit dependence on the function's first derivative and extreme sensitivity to the selection of initial points. On the other hand, Bisection and Regula Falsi methods, although only offering linear convergence, demonstrated better robustness in guaranteeing convergence as long as the function satisfies continuity conditions and sign changes in the initial interval.

These findings were subsequently reinforced and expanded by Ahmad et al. [11], who conducted an empirical investigation of Newton-Raphson and Regula Falsi using the MATLAB platform. Their experimental results confirmed that Newton-Raphson indeed excels in terms of precision and computational efficiency, but Regula Falsi offers complementary advantages in the form of robustness to variations in initial conditions and the ability to operate without requiring derivative calculations - making this particularly advantageous when the observed function is not easily differentiable analytically or numerically. A more unique perspective comes from Azure et al. [9], who adopted a manual computation approach using calculators as the primary instrument. This study yielded the interesting finding that Newton-Raphson and Regula Falsi demonstrated simultaneous convergence at the second iteration, suggesting a potential synergy between speed and stability worthy of further exploration.

As application problem complexity increased, the methodological scope of research expanded accordingly. Wang [14], for instance, conducted an extensive investigation of five numerical methods in the context of modern financial applications, particularly in implementing the Black-Scholes Model for option pricing. Through rigorous MATLAB-based analysis, this study revealed that the Secant method emerged as an optimal compromise: it offers superlinear convergence without dependence on derivatives, while maintaining adequate numerical stability across various market scenarios. Furthermore, the exploration of Golden Section Search as an alternative approach opened new dimensions in root-finding methodology, especially in the context of one-dimensional optimization closely related to roots of derivative functions.

Recent trends in numerical research indicate a paradigm shift toward the development of hybrid algorithms that intelligently integrate the strengths of various fundamental methods. In this context, Thota et al. [13] introduced a novel hybrid algorithm that combines elements of the Regula Falsi and Newton–Raphson methods. Experimental validation performed in MATLAB shows that the algorithm not only preserves the quadratic convergence characteristic of Newton–Raphson, but also guarantees convergence stability even under extreme conditions (for example,

when the first derivative of the function is zero or undefined near the root).

Parallel research paradigms are also evolving in two main directions. First is the development of derivative-free methods that minimize dependence on differential information; second is the application of convergence-acceleration techniques to improve the convergence rate of basic iterative methods. Argyros et al. [15] made an important theoretical contribution to the first direction through an in-depth analysis of Newton's method and Steffensen-like variants in Banach spaces. Their crucial findings show that Steffensen-like methods can not only achieve quadratic convergence without requiring derivatives, but also possess a wider domain of convergence thanks to the restructuring of Lipschitz conditions and the use of more sophisticated first-order finite-difference approximation approaches.

On the convergence acceleration front, Chen and Liao (2025) comprehensively examined the Fixed-Point Iteration scheme along with two classical acceleration techniques: Aitken's Δ^2 and Steffensen. Their empirical demonstration proved that the Steffensen method (which essentially generalizes Aitken's Δ^2) can yield significantly faster convergence, even capable of "rescuing" previously divergent iteration schemes in the context of complex engineering problems. This confirms the substantial potential of acceleration techniques as vital tools for expanding the applicability range of basic iterative methods.

The results from several previously discussed studies can be observed in the following Table 1.

3 Methods

This research aims to evaluate and compare the effectiveness of various numerical methods in solving nonlinear equations of the form $f(x) = 0$. To achieve this objective, the research is conducted through four main sequential stages: (1) literature study and formulation of the theoretical foundation, (2) design and implementation of algorithms in a Python environment, (3) design of controlled computational

Table 1. Comparison of numerical methods in previous studies.

Researcher	Methods Studied	Tools	Findings	Limitations
Thakur et al. [10]	Bisection, Newton-Raphson, False Position	Theoretical Analysis	Newton-Raphson fastest (quadratic), False Position moderate, Bisection slowest (linear)	Did not cover Secant, Fixed-Point, and acceleration methods
Ahmad et al. [11]	Newton-Raphson, Regula Falsi	MATLAB	Newton more accurate & faster, Regula Falsi more robust to initial guess	Limited comparison to only two methods
Azure et al. [9]	Bisection, Newton-Raphson, Regula Falsi, Secant, Fixed-Point	Manual Computation	Newton & Regula Falsi fastest (2 iterations), Secant (5 iterations), Fixed-Point (7 iterations)	No software usage, did not cover acceleration methods
Dixit [12]	Bisection, Newton-Raphson, False Position, Secant	C/C++ Programming	Compared numerical accuracy for nth roots; Newton and Secant more accurate than bracketing methods	Limited to specific roots; no acceleration methods included
Thota et al. [13]	Hybrid Algorithm, Bisection, Regula-Falsi, Newton-Raphson	MATLAB	Quadratic hybrid algorithm with guaranteed convergence, able to handle zero derivative	Focus on one specific hybrid algorithm
Wang [14]	Bisection, Newton, Secant, Fixed-Point, Golden Section	MATLAB (Finance)	Secant optimal for speed-stability, Golden Section alternative for monotonic	Did not cover Regula Falsi and acceleration methods
Argyros et al. [15]	Newton, Steffensen-like Methods	Theoretical & Numerical Analysis	Steffensen-like: wide convergence domain, tight error bound, derivative-free	Focus on theoretical convergence analysis
Chen et al. [16]	Fixed-Point, Aitken Acceleration, Steffensen Iteration	MATLAB (Engineering)	Steffensen & Aitken significant acceleration, converged in 3 iterations for real problems	Did not compare with other basic methods

experiments, and (4) comparative analysis based on quantitative performance metrics. The entire process is designed to ensure validity, reproducibility, and clarity of result interpretation.

3.1 Literature Study and Theoretical Foundation

The initial research stage is conducted through an in-depth study of scientific literature concerning root-finding methods for nonlinear equations. The examined sources include standard textbooks in numerical analysis, reputable journal articles, and technical documentation from numerical computation libraries. The study focuses on fundamental principles, mathematical derivations, convergence conditions, theoretical convergence order, as well as the strengths and limitations of each method (Bisection, Regula-Falsi, Fixed Point, Newton-Raphson, Secant, Aitken's Δ^2, and Steffensen).

The results of the literature study form the basis for algorithm design, determination of evaluation criteria, and selection of test functions that are representative of various characteristics of nonlinear function behavior.

3.2 Algorithm Design and Implementation in Python

Numerical implementation was carried out in the Python 3.12.3 programming language, utilizing four main libraries that each play specific and complementary roles in the research workflow.

3.2.1 SymPy: Symbolic Representation and Manipulation of Mathematical Functions

SymPy (Symbolic Python) is a library for symbolic computation that enables manipulation of mathematical expressions in symbolic form [17–19]. In this research, SymPy is used for:

1. Defining objective functions $f(x)$ explicitly as symbolic objects (for example, $f(x) = x^2 - 4$),
2. Automatically computing analytical derivatives $f'(x)$ using the `diff()` function, which is crucial for the Newton-Raphson method,
3. Verifying exact solutions or analytical roots (if they exist) through the `solve()` function, which serves as ground truth for measuring the accuracy of numerical methods,
4. Avoiding early representation errors by maintaining pure mathematical forms until the numerical evaluation stage.

After completing symbolic manipulations, SymPy expressions are converted into efficient numerically evaluable functions using NumPy.

3.2.2 NumPy: High-Performance Numerical Computing

NumPy (Numerical Python) is the core library for numerical computing in Python [20–23]. In this research, NumPy is used for:

1. Evaluating mathematical functions (both $f(x)$ and $f'(x)$) at specific numerical values with high precision,
2. Performing basic mathematical operations such as absolute value ($|x|$), exponentiation (x^n), logarithms, and transcendental functions ($\exp(x)$, $\sin(x)$, $\cos(x)$, etc.),
3. Storing and processing iteration data—such as the history of root approximations x_n and errors $|x_{n+1} - x_n|$—in high-performance `ndarray` format,
4. Calculating performance metrics like absolute error relative to reference solutions, i.e., $|x_{\text{numeric}} - x_{\text{reference}}|$.

The combination of SymPy (for symbolic representation) and NumPy (for numerical evaluation) enables an accurate, flexible, and efficient workflow.

3.2.3 Pandas: Management and Structuring of Experimental Data

Pandas is a library for structured data manipulation and analysis [23–26]. In this research, Pandas is used for:

1. Collecting all experimental results into an organized data structure in the form of `DataFrame`,
2. Storing key information from each test, including: method name, test function, initial guess, root approximation, number of iterations, execution time, success status, and convergence history,
3. Enabling data grouping by categories (for example, "all results for the Newton–Raphson method on transcendental functions"),
4. Performing descriptive statistical analysis (mean, standard deviation, failure frequency) directly through built-in Pandas operations,
5. Exporting data to external formats (such as CSV or Excel) for documentation, verification, or further visualization.

The use of Pandas ensures that experimental data is not only recorded but also ready for systematic and comparative analysis.

3.2.4 Matplotlib: Visualization of Experimental Results

Matplotlib is the main library for data visualization in Python [27–30]. In this research, Matplotlib is used for:

1. Generating convergence plots that display absolute error against the number of iterations, allowing direct observation of the convergence speed and stability of each method,
2. Creating bar or line charts to visually compare performance metrics (number of iterations, execution time) between methods,
3. Constructing heatmaps or success matrices to display the robustness of methods against variations in initial guesses,
4. Producing clear, fully labeled, and publication-ready visualizations to support qualitative interpretation and presentation of findings.

These visualizations not only assist the researcher in analyzing performance patterns but also strengthen the communication of results to readers of reports or scientific publications.

3.3 Computational Experiment Design

To ensure comprehensive and fair evaluation, a series of test functions with diverse characteristics were selected and defined symbolically using SymPy. These functions include:

1. Simple polynomials, for example $f(x) = x^2 - 4$,
2. Transcendental functions, for example $f(x) = e^x - 2$ and $f(x) = \sin(x) - \frac{x}{2}$,
3. Functions with multiple roots, for example $f(x) = (x-1)^2$,
4. Functions with challenging numerical behavior, such as $f(x) = x^{\frac{1}{3}}$ (infinite derivative at $x = 0$) or $f(x) = x\cos(x) - 1$ (strong oscillations).

Each method was tested against all test functions with standard parameter configurations:

1. Error tolerance: $\varepsilon = 10^{-8}$,
2. Maximum iterations: 100,
3. Initialization schemes: tested in several scenarios (near the root, far from the root, in flat or steep regions) to assess robustness.

As ground truth, the reference solution x_{ref} was obtained from analytical solutions (if available) or from high-precision computations using external scientific libraries with very tight tolerance (10^{-14}).

The performance metrics recorded and analyzed include:

1. **Accuracy**: absolute difference between numerical solution and reference solution, i.e., $|x_{\text{numeric}} - x_{\text{ref}}|$,
2. **Convergence Speed**: number of iterations until stopping criteria are met,
3. **Computational Efficiency**: actual execution time (*wall-clock time*),
4. **Robustness**: proportion of initialization scenarios that result in successful convergence.

All experimental results are stored in a Pandas `DataFrame` structure to enable aggregate analysis, cross-comparison, and systematic visualization.

3.4 Comparative Analysis and Interpretation of Results

In the final stage, quantitative data from all experiments were collected and analyzed comparatively. The analysis was conducted both per-function and aggregated across all test functions. Visualization using Matplotlib was employed to clarify performance patterns, including:

1. Logarithmic convergence plots (log error vs. iteration) to observe empirical convergence order,
2. Bar charts comparing iteration counts and execution times,
3. Performance ranking tables based on key metrics.

Findings were synthesized into:

1. Performance profiles of each method in the context of specific function types,
2. Practical recommendations for method selection based on problem characteristics (e.g., derivative availability, speed vs. reliability requirements),
3. Identification of specific conditions under which particular methods tend to fail or be less efficient.

4 Literature Review

4.1 Closed Methods

This section introduces root-finding using closed methods, which are called "closed" because they use two bounds, ensuring the root lies between these boundaries. These methods are developed based on a theorem in Calculus, namely:

Theorem 4.1 *If $\{a_n\}$ and $\{c_n\}$ converge to L and*

$$a_n \leq b_n \leq c_n \quad \text{for } n \geq K$$

(where K is a specific integer), then $\{b_n\}$ also converges to L.

Closed methods use two bounds: an initial lower bound and an initial upper bound. Suppose f is a continuous function on the interval $[a, b]$, where $f(a)$ and $f(b)$ have different signs, and there exists a point p between point a and point b or satisfying $a \leq p \leq b$ with $f(p) = 0$, even though there may be many points between a and b that are roots of the equation.

4.1.1 Bisection Method

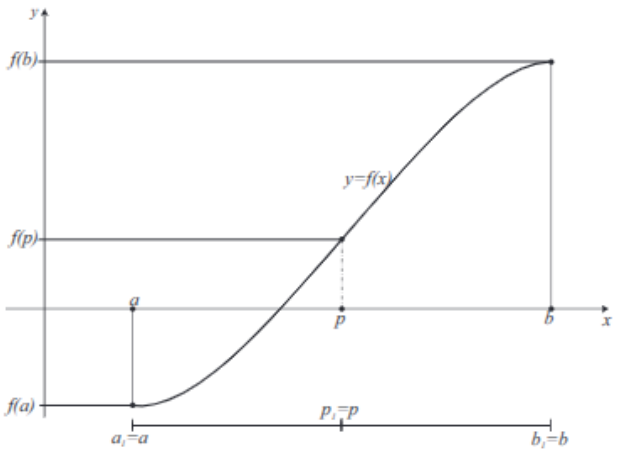


Figure 1. Root search using the Bisection method.

The Bisection Method (or method of bisection) is a **simple and reliable** numerical method for finding roots of nonlinear equations $f(x) = 0$. This method is based on the **Intermediate Value Theorem**: if function f is continuous on the interval $[a, b]$ and $f(a) \cdot f(b) < 0$ holds, then there exists at least one root $c \in (a, b)$ such that $f(c) = 0$.

As illustrated in Figure 1, the Bisection Method works by **iteratively dividing the interval into two equal parts** and selecting the sub-interval that continues to satisfy the function sign change condition. This process is repeated until the interval width becomes sufficiently small or the function value at the midpoint is sufficiently close to zero [5, 31–33]. Although its convergence is **slow (linear)**, this method **always converges** as long as the initial conditions are satisfied, making it frequently used as an initial method to obtain rough approximations before switching to faster methods.

The midpoint at the first iteration is given by

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

If $f(p_1) = 0$, then p_1 is the root. Otherwise, as shown in Figure 1, the interval is updated based on the sign of $f(p_1)$ relative to $f(a_1)$ or $f(b_1)$.

Bisection Method Algorithm

1. Choose an initial interval $[a_1, b_1]$ such that $f(a_1) \cdot f(b_1) < 0$.
2. For each iteration $n = 1, 2, 3, \ldots$:
   - Calculate the midpoint:
   $$p_n = \frac{a_n + b_n}{2}$$
   - If $f(p_n) = 0$ or $|f(p_n)| <$ tolerance, stop; p_n is the root.
   - If $f(a_n) \cdot f(p_n) < 0$, then the root lies in $[a_n, p_n]$:
   set $a_{n+1} = a_n$, $b_{n+1} = p_n$.
   - If $f(p_n) \cdot f(b_n) < 0$, then the root lies in $[p_n, b_n]$:
   set $a_{n+1} = p_n$, $b_{n+1} = b_n$.
3. Repeat step 2 until the interval width $|b_n - a_n|$ or $|f(p_n)|$ is less than the specified tolerance.

This method guarantees convergence, but its speed is only linear, with a convergence rate of approximately $\frac{1}{2}$ per iteration. The point update process in each iteration is visually depicted in Figure 2.

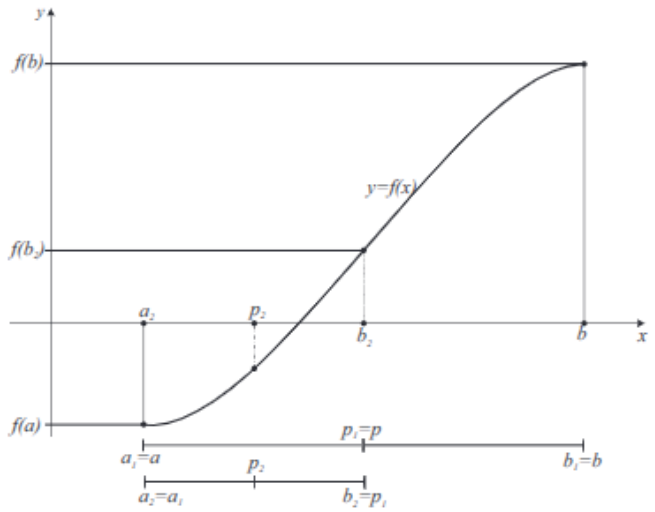


Figure 2. Point update in the Bisection method.

4.1.2 *Regula-Falsi Method*

The Regula Falsi method (or false position method) is a numerical method for finding roots of nonlinear equations $f(x) = 0$. Like the Bisection method, this method also requires an initial interval $[a, b]$ where function f is continuous and satisfies $f(a) \cdot f(b) < 0$, which guarantees the existence of at least one root within that interval based on the Intermediate Value Theorem.

However, unlike the Bisection method which simply divides the interval into two equal parts, **the Regula Falsi method uses a geometric approach**: it connects the points $(a, f(a))$ and $(b, f(b))$ with a straight line, then takes the **intersection point of this line with the x-axis** as the next root approximation. This approach often results in faster convergence than Bisection.

The Regula Falsi method begins with two endpoints of the interval $[a_1, b_1]$ where function f is continuous and satisfies $f(a_1) \cdot f(b_1) < 0$. To obtain the root approximation, we connect the points $(a_1, f(a_1))$ and $(b_1, f(b_1))$ with a straight line [34–36].

The equation of the line passing through these two points is given by

$$\frac{y - f(a_1)}{f(b_1) - f(a_1)} = \frac{x - a_1}{b_1 - a_1}$$

Since we want to find the intersection point of this line with the x-axis, we substitute $y = 0$ and $x = p_1$, yielding

$$\frac{-f(a_1)}{f(b_1) - f(a_1)} = \frac{p_1 - a_1}{b_1 - a_1}$$

which results in the a_1-based form:

$$p_1 = a_1 - \frac{f(a_1)(b_1 - a_1)}{f(b_1) - f(a_1)} \tag{1}$$

However, we can also derive a **b_1-based form** by writing the line equation relative to point $(b_1, f(b_1))$:

$$\frac{y - f(b_1)}{f(a_1) - f(b_1)} = \frac{x - b_1}{a_1 - b_1}$$

Substituting $y = 0$, $x = p_1$ gives

$$\frac{-f(b_1)}{f(a_1) - f(b_1)} = \frac{p_1 - b_1}{a_1 - b_1}$$

Since $a_1 - b_1 = -(b_1 - a_1)$ and $f(a_1) - f(b_1) = -(f(b_1) - f(a_1))$, the above equation becomes

$$\frac{f(b_1)}{f(b_1) - f(a_1)} = -\frac{p_1 - b_1}{b_1 - a_1}$$

Multiplying both sides by $b_1 - a_1$:

$$\frac{f(b_1)(b_1 - a_1)}{f(b_1) - f(a_1)} = b_1 - p_1$$

Thus, we obtain the final b_1-based form:

$$p_1 = b_1 - \frac{f(b_1)(b_1 - a_1)}{f(b_1) - f(a_1)} \tag{2}$$

This is the **b-based Regula Falsi form**, which is algebraically equivalent to the a-based form, but emphasizes the role of the right endpoint in the calculation.

The iteration process continues by updating the interval based on the sign of $f(p_n)$, ensuring that the condition $f(a_{n+1}) \cdot f(b_{n+1}) < 0$ remains satisfied.

Regula Falsi Method Algorithm (b-based version)

1. Choose an initial interval $[a_1, b_1]$ such that $f(a_1) \cdot f(b_1) < 0$.
2. For each iteration $n = 1, 2, 3, \ldots$:
   - Calculate the root approximation:

$$p_n = b_n - \frac{f(b_n)(b_n - a_n)}{f(b_n) - f(a_n)}$$

   - If $f(p_n) = 0$ or $|f(p_n)| <$ tolerance, stop; p_n is the root.

- If $f(p_n)$ **has the same sign** as $f(b_n)$, then the root lies in $[a_n, p_n]$:
  set $a_{n+1} = a_n$, $b_{n+1} = p_n$.
- If $f(p_n)$ **has the opposite sign** to $f(b_n)$ (meaning it has the same sign as $f(a_n)$), then the root lies in $[p_n, b_n]$:
  set $a_{n+1} = p_n$, $b_{n+1} = b_n$.

3. Repeat step 2 until convergence criteria are met.

The geometric interpretation of this method, where the new approximation is obtained by intersecting the chord connecting $(a, f(a))$ and $(b, f(b))$ with the x-axis, is illustrated in Figure 3.

Note: Although this method is generally faster than Bisection, in certain cases one endpoint may become "stagnant" (not changing), resulting in slow convergence. Modifications such as the *Illinois method* or *Anderson–Björck* can be used to address this issue [5].

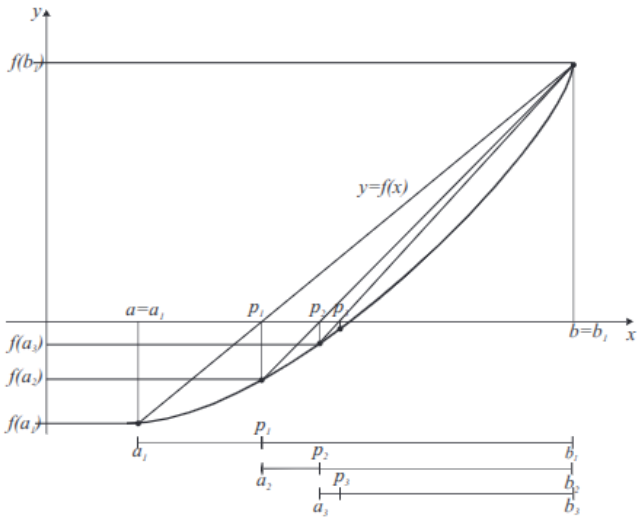


Figure 3. Root search using the Regula-Falsi method.

4.2 Open Methods

In closed methods, the root of an equation lies within a predetermined interval, and so do subsequent intervals, ensuring the root is always found. This section introduces root-finding using open methods, which are called "open" because they use only one initial estimate. Five methods will be introduced in this section: Fixed-Point Iteration Method, Newton-Raphson Method, Secant Method, Aitken's Δ^2 Method, and Steffensen's Method.

4.2.1 Fixed-Point Method

The Fixed-Point Iteration Method is a numerical method for solving nonlinear equations $f(x) = 0$ by transforming the equation into the equivalent form $x = g(x)$. The solution to this equation is called a **fixed point** because the value x does not change after applying function g, i.e., $x = g(x)$.

The iteration process begins with an **initial guess** x_0, then computed recursively:

$$x_{n+1} = g(x_n) \tag{3}$$

for $n = 0, 1, 2, \ldots$, until a sufficiently accurate root approximation is obtained.

The success of this method heavily depends on the **choice of function** $g(x)$ and the **initial point** x_0 [3, 37, 38]. Geometric constructions and reformulation techniques can expand the domain of convergence of the basic fixed-point scheme and yield new iterative families with improved convergence characteristics [37, 38]. If function $g(x)$ and the initial point are chosen appropriately, the sequence $\{x_n\}$ will **converge to the fixed point**. However, if chosen poorly, the iteration may **diverge or oscillate**, potentially moving away from the solution.

This is illustrated in Figures 4 and 5:

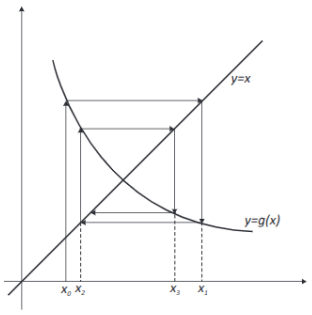


Figure 4. Proper initial point selection in the Fixed-Point method.

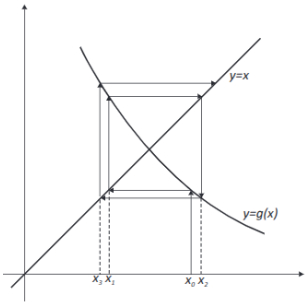


Figure 5. Incorrect initial point selection in the Fixed-Point method.

Local convergence can be guaranteed if $|g'(x)| < 1$ in the neighborhood of the fixed point. If this condition is satisfied, the iteration will converge linearly to that fixed point [16].

Fixed-Point Iteration Method Algorithm

1. Transform the equation $f(x) = 0$ into the form $x = g(x)$.
2. Choose an initial guess x_0 that is "close" to the solution.
3. For each iteration $n = 0, 1, 2, \ldots$:

- Compute:
$$x_{n+1} = g(x_n)$$
- If $|x_{n+1} - x_n| <$ tolerance or $|f(x_{n+1})| <$ tolerance, stop; x_{n+1} is the root approximation.

4. Repeat step 3 until convergence criteria are met.

Important Notes:

- Convergence **does not always occur** — it heavily depends on the form of $g(x)$ and the value of x_0.
- Sufficient condition for local convergence: $|g'(x)| < 1$ in the neighborhood of the fixed point.

4.2.2 Newton-Raphson Method

The Newton-Raphson method is one of the most popular methods for finding roots of nonlinear equations $f(x) = 0$. This method uses a **local linearization approach** through first-order Taylor series to obtain increasingly accurate root approximations at each iteration.

Before deriving the iteration formula, consider the Taylor series of function $f(p)$ around point p_0, assuming $|p - p_0|$ is small:

$$f(p) = f(p_0) + (p - p_0)f'(p_0) + \frac{(p - p_0)^2}{2}f''(\xi(p)),$$

where $\xi(p)$ lies between p and p_0. If p is the true root, then $f(p) = 0$, so:

$$0 = f(p_0) + (p - p_0)f'(p_0) + \frac{(p - p_0)^2}{2}f''(\xi(p)).$$

Since $|p - p_0|$ is assumed to be small, the quadratic term $(p - p_0)^2$ is considered very small and neglected. With this first-order approximation, we obtain:

$$0 \approx f(p_0) + (p - p_0)f'(p_0),$$

which yields the root approximation:

$$p \approx p_0 - \frac{f(p_0)}{f'(p_0)} \equiv p_1.$$

If this process is repeated iteratively, we obtain the general formula of the **Newton-Raphson Method**:

$$p_{n+1} = p_n - \frac{f(p_n)}{f'(p_n)}, \quad \text{for } n \geq 0, \text{ with } f'(p_n) \neq 0. \tag{4}$$

This method has **quadratic convergence** if the initial guess is sufficiently close to the root and f' is continuous and non-zero around the root. A geometric illustration of this method is shown in Figure 6, where each iteration follows the tangent line of the curve $f(x)$ until it intersects the x-axis.

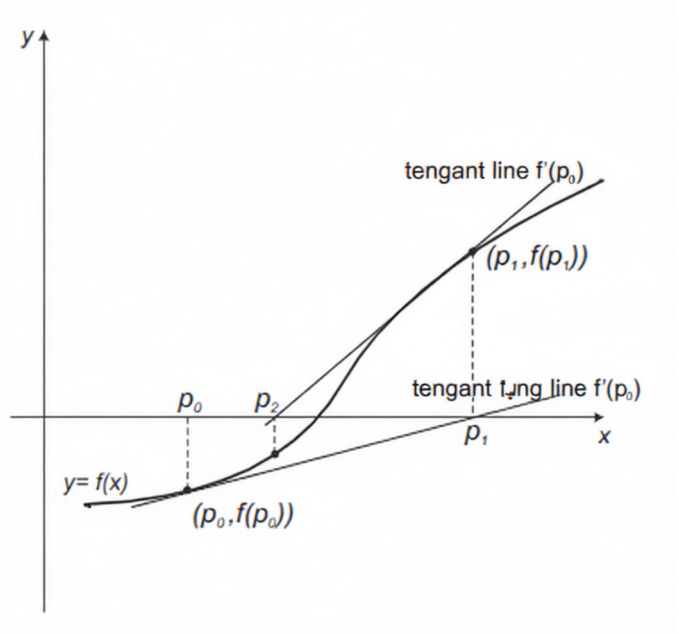


Figure 6. Root search using the Newton-Raphson method.

Newton-Raphson Method Algorithm

1. Choose an initial guess p_0 sufficiently close to the root.
2. For each iteration $n = 0, 1, 2, \ldots$:
   - Compute the function value and its derivative: $f(p_n)$ and $f'(p_n)$.
   - If $f'(p_n) = 0$, stop the process (fails due to division by zero).
   - Compute the new approximation:
$$p_{n+1} = p_n - \frac{f(p_n)}{f'(p_n)}$$
   - If $|p_{n+1} - p_n| <$ tolerance or $|f(p_{n+1})| <$ tolerance, stop; p_{n+1} is the root.
3. Repeat step 2 until convergence criteria are met.

Notes:

- This method is **very fast** (quadratic) if it converges.
- However, it **does not always converge** — it heavily depends on the choice of p_0 and the nature of the function f.
- It requires the computation of **analytical or numerical derivatives**.

4.2.3 *Secant Method*

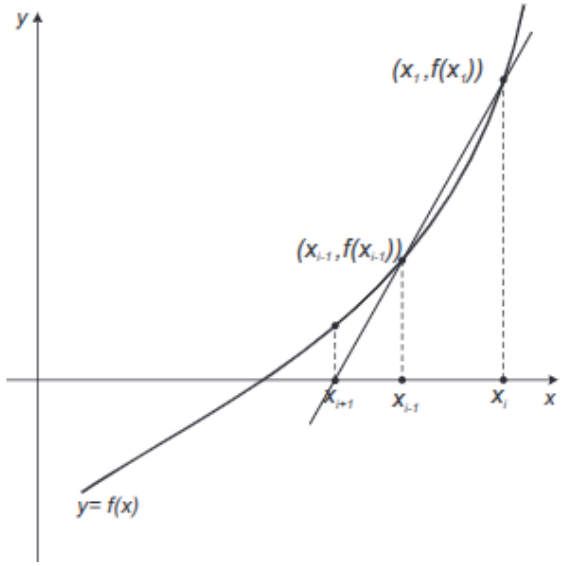


Figure 7. Root search using the Secant method.

The Secant Method is an iterative method for finding roots of nonlinear equations $f(x) = 0$ that **does not require explicit derivatives**, unlike the Newton-Raphson method. This method is similar to the **Regula Falsi method**, as both use a **straight line passing through two points on the curve** $f(x)$ to approximate the root. However, **unlike Regula Falsi**, the Secant Method **does not require the two initial points to bracket the root** (i.e., it is not necessary that $f(a) \cdot f(b) < 0$).

As illustrated in Figure 7, suppose two initial guesses are given as x_{i-1} and x_i. The line passing through points $(x_{i-1}, f(x_{i-1}))$ and $(x_i, f(x_i))$ has the equation:

$$\frac{y - f(x_i)}{f(x_{i-1}) - f(x_i)} = \frac{x - x_i}{x_{i-1} - x_i}.$$

The intersection of this line with the x-axis occurs when $y = 0$, i.e., at $(x_{i+1}, 0)$ as shown in Figure 7. Substituting $y = 0$ yields:

$$\frac{-f(x_i)}{f(x_{i-1}) - f(x_i)} = \frac{x_{i+1} - x_i}{x_{i-1} - x_i},$$

which can be rearranged into the iteration formula of the **Secant Method**:

$$x_{i+1} = x_i - \frac{f(x_i)(x_{i-1} - x_i)}{f(x_{i-1}) - f(x_i)}. \tag{5}$$

This formula is used repeatedly: after computing x_{i+1}, the pair of points is updated to (x_i, x_{i+1}), and the process is repeated until $f(x_{i+1})$ is sufficiently close to zero.

The Secant Method has a **convergence order of approximately 1.618** (the golden ratio), making it faster than linear methods such as the Bisection method, but slightly slower than the quadratic convergence of Newton-Raphson. Its advantage is that it **does not require derivatives**, making it very useful when $f'(x)$ is difficult or expensive to compute.

Secant Method Algorithm

1. Choose two initial guesses x_0 and x_1 (they do not need to bracket the root).
2. For each iteration $i = 1, 2, 3, \ldots$:
   - Compute:
   $$x_{i+1} = x_i - \frac{f(x_i)(x_{i-1} - x_i)}{f(x_{i-1}) - f(x_i)}$$
   - If $f(x_{i+1}) = 0$ or $|f(x_{i+1})| <$ tolerance, stop; x_{i+1} is the root.
   - Update:
   $$x_{i-1} \leftarrow x_i, \quad x_i \leftarrow x_{i+1}$$
3. Repeat step 2 until convergence criteria are met.

Note:

- This method **does not guarantee convergence** if the initial guesses are too far from the root.
- Avoid division by zero: ensure $f(x_{i-1}) \neq f(x_i)$.

4.2.4 *Aitken's Δ^2 Method*

Aitken's Δ^2 method is a **convergence acceleration** technique used to speed up the convergence rate of a sequence that is **converging linearly**. This method does not require function derivatives, but rather utilizes three consecutive terms of the iteration

sequence to produce a more accurate approximation of the true limit.

Suppose we have an iteration sequence $\{p_n\}$ converging to p. If the convergence is linear, then the error decreases proportionally, i.e.:

$$p_{n+1} - p \approx \lambda(p_n - p), \quad \text{with } 0 < |\lambda| < 1.$$

Aitken observed that the dominant component of this error can be eliminated by utilizing three consecutive terms: p_n, p_{n+1}, and p_{n+2}.

Origin of the Aitken's Δ^2 Formula

From the linear convergence assumption, we can write:

$$\begin{aligned} p_{n+1} - p &\approx \lambda(p_n - p), \\ p_{n+2} - p &\approx \lambda(p_{n+1} - p) \approx \lambda^2(p_n - p). \end{aligned}$$

We want to eliminate λ and p to obtain a direct approximation of p. Consider the successive differences:

$$\begin{aligned} \Delta p_n &= p_{n+1} - p_n, \\ \Delta p_{n+1} &= p_{n+2} - p_{n+1}, \\ \Delta^2 p_n &= \Delta p_{n+1} - \Delta p_n = p_{n+2} - 2p_{n+1} + p_n. \end{aligned}$$

With the linear approximation, it can be shown that:

$$\frac{(p_{n+1} - p_n)^2}{p_{n+2} - 2p_{n+1} + p_n} \approx p_n - p.$$

Therefore, a better approximation of p is given by:

$$\hat{p}_n = p_n - \frac{(p_{n+1} - p_n)^2}{p_{n+2} - 2p_{n+1} + p_n}. \tag{6}$$

This formula is known as **Aitken's Δ^2 Method**, where $\Delta^2 p_n$ denotes the *second difference* of the sequence $\{p_n\}$.

Aitken's Δ^2 Method Algorithm

The following are the steps for applying this method:

1. Start with an iterative method (for example, fixed-point iteration $p_{n+1} = g(p_n)$) to generate the sequence $\{p_n\}$.
2. For each $n = 0, 1, 2, \ldots$:
   - Compute three consecutive terms: p_n, p_{n+1}, and p_{n+2}.
   - Compute the accelerated approximation:
   $$\hat{p}_n = p_n - \frac{(p_{n+1} - p_n)^2}{p_{n+2} - 2p_{n+1} + p_n}.$$
   - If the denominator $p_{n+2} - 2p_{n+1} + p_n \approx 0$, stop the process (the sequence may have converged or numerical division by zero may occur).
3. Repeat the process until $|\hat{p}_n - \hat{p}_{n-1}| < \varepsilon$ or until other convergence criteria are met.

This method is very effective when combined with iterative methods that have slow convergence, such as **fixed-point iteration**. Although it does not guarantee quadratic convergence, Aitken's Δ^2 often provides significant acceleration to the linear convergence rate.

This method is also known as *Aitken extrapolation* and is one of the earliest and most famous convergence acceleration methods in numerical analysis [39–41]. Extensions of the classical Aitken Δ^2 formula to cases where consecutive error ratios are not constant have further broadened its applicability to a wider class of iterative sequences [41].

4.2.5 *Steffensen's Method*

Steffensen's method is an iterative **derivative-free** method for solving nonlinear equations $f(x) = 0$. Although it doesn't use explicit derivatives, this method achieves **quadratic convergence order** — equivalent to the Newton-Raphson method — making it highly efficient for functions that are difficult or expensive to differentiate.

This method can be viewed as a direct application of **Aitken's Δ^2 acceleration** to **fixed-point iteration**. Suppose the root of $f(x) = 0$ is equivalent to a fixed point of some function $g(x)$, i.e.:

$$x = g(x) \quad \Leftrightarrow \quad f(x) = 0.$$

A common choice is $g(x) = x - f(x)$, although other forms can also be used as long as the root of $f(x) = 0$ corresponds to a fixed point of $g(x)$.

Origin of the Steffensen Formula

In ordinary fixed-point iteration, we generate the sequence:

$$p_0, \quad p_1 = g(p_0), \quad p_2 = g(p_1) = g(g(p_0)), \quad p_3 = g(p_2), \quad \ldots$$

If convergence is only linear, then Aitken's Δ^2 can be used to accelerate it. By taking three consecutive terms from the fixed-point sequence:

$$p_n, \quad p_{n+1} = g(p_n), \quad p_{n+2} = g(g(p_n)),$$

the accelerated approximation according to Aitken is:

$$\hat{p}_n = p_n - \frac{(p_{n+1} - p_n)^2}{p_{n+2} - 2p_{n+1} + p_n}.$$

Substituting $p_n = x_n$, $p_{n+1} = g(x_n)$, and $p_{n+2} = g(g(x_n))$ yields the Steffensen iteration formula:

$$x_{n+1} = x_n - \frac{\big(g(x_n) - x_n\big)^2}{g(g(x_n)) - 2g(x_n) + x_n}. \tag{7}$$

If we choose $g(x) = x - f(x)$, then:

$$g(x_n) = x_n - f(x_n),$$
$$g(g(x_n)) = g(x_n - f(x_n)) = x_n - f(x_n) - f(x_n - f(x_n))$$

However, in numerical practice, it is more common and efficient to rewrite the Steffensen formula directly in terms of function evaluations of f, without explicitly defining g. With some algebraic manipulation, the formula can be expressed as:

$$x_{n+1} = x_n - \frac{f(x_n)^2}{f(x_n + f(x_n)) - f(x_n)}.$$

Note that: (1) $f(x_n)$ is evaluated once; (2) $f(x_n + f(x_n))$ is evaluated once; (3) thus requiring two function evaluations per iteration (not three, as is sometimes misunderstood).

The denominator $f(x_n + f(x_n)) - f(x_n)$ serves as a **first-order divided difference approximation** of the derivative $f'(x_n)$, so this method implicitly mimics the Newton method without derivatives.

Steffensen's Method Algorithm

The following are the steps for numerical implementation of Steffensen's method:

1. Choose an initial guess x_0 and tolerance $\varepsilon > 0$.
2. For $n = 0, 1, 2, \ldots$ do:
   - Compute $f_n = f(x_n)$.
   - If $|f_n| < \varepsilon$, stop; x_n is the root approximation.
   - Compute $y_n = x_n + f_n$.
   - Compute $f_y = f(y_n)$.
   - Compute denominator: $d = f_y - f_n$.
   - If $|d| < \varepsilon$, stop the process (risk of division by zero or convergence already achieved).
   - Update the approximation:

$$x_{n+1} = x_n - \frac{f_n^2}{d}.$$

3. Repeat until convergence criteria are met.

Steffensen's method is very useful in situations where function derivatives are not available analytically or are expensive to compute. Although it requires two function evaluations per iteration (compared to one in fixed-point iteration), the major advantage is **quadratic convergence** which significantly reduces the total number of iterations.

This method is an elegant example of how acceleration techniques (Aitken) can be transformed into robust standalone iterative methods [6, 42].

5 Findings and Discussion

5.1 Code Testing Results for Polynomial Function

In this section, we present the results of computational simulations designed to evaluate the effectiveness of the seven root-finding methods. The assessment was conducted not only based on convergence theory but more importantly on their actual performance in real computational environments using Python, considering factors such as runtime efficiency and resilience to diverse function forms.

To provide a comprehensive overview, we tested each algorithm on a suite of functions consisting of polynomial, trigonometric, exponential, and mixed functions. The selection of these functions aims to replicate various challenges faced by practitioners, ranging from well-behaved problems to those with complex nonlinear characteristics and susceptibility to oscillations.

The implementation of all methods was performed consistently in a Python 3.12.3 environment (with source code that has been created and explained previously in the Methods section), utilizing standard libraries such as NumPy for numerical computation and Matplotlib for convergence visualization. Each method was executed with initial parameters adjusted according to the characteristics of the test functions, while convergence criteria were established based on absolute tolerance of function value ($|f(x)| < \varepsilon$) and/or relative change between iterations ($|x_{n+1} - x_n| < \delta$), with $\varepsilon = \delta = 10^{-10}$, and a maximum of 100 iterations as a general standard to be used in testing all methods.

As an initial step in this comparative study, a high-degree polynomial function was selected as a typical representation of algebraic function classes that frequently appear in various engineering and scientific applications. The function used in this experiment is defined as:

$$f(x) = x^7 - 2x^6 + 4x^5 - 1 \tag{8}$$

This function was intentionally chosen due to its challenging characteristics: odd degree (ensuring at least one real root), mixed coefficients (positive and negative), and non-monotonic structure that can produce complex behavior in numerical iteration processes. Additionally, this function lacks easily guessable rational roots, making it particularly suitable for testing the robustness of numerical methods in finding irrational roots with high precision.

To support the application of the fixed-point method, two equivalent iterative function forms $g(x)$ were formulated from the equation $f(x) = 0$. The first form was obtained by isolating the x^7 term:

$$x^7 = 2x^6 - 4x^5 + 1 \quad \Rightarrow \quad x = \sqrt[7]{2x^6 - 4x^5 + 1}$$

The second form was obtained by isolating the $4x^5$ term:

$$4x^5 = -x^7 + 2x^6 + 1 \quad \Rightarrow \quad x = \sqrt[5]{\frac{-x^7 + 2x^6 + 1}{4}}$$

The main numerical parameters in this experiment were established as follows:

- Initial interval for bracketing methods: $[0, 1]$. The selection of the interval $[0, 1]$ as the initial bounds is not arbitrary, but rather based on preliminary analysis of the function's sign at the interval endpoints. Initial evaluation shows that:

  $$f(0) = 0^7 - 2 \cdot 0^6 + 4 \cdot 0^5 - 1 = -1 < 0$$

  and

  $$f(1) = 1^7 - 2 \cdot 1^6 + 4 \cdot 1^5 - 1 = 2 > 0.$$

  Since $f(0) \cdot f(1) < 0$ and $f(x)$ is continuous throughout $\mathbb{R}$ (as a polynomial function), then according to the Intermediate Value Theorem, there exists at least one real root in the open interval $(0, 1)$. This interval provides a solid foundation for applying the Bisection and Regula Falsi methods, which inherently require two initial guesses that bracket the root.

Experimental results demonstrate that all tested numerical methods successfully converged to the same root, namely $x \approx 0.800631154473$, with very high accuracy. This result indicates excellent numerical stability of all implemented algorithms. Numerical differences between methods are on the order of 10^{-12} to 10^{-14}, which can be practically neglected for most scientific computing applications. Please refer to the following Table 2.

The Secant, Steffensen, and Newton methods achieved zero numerical error within the established tolerance limits, while the Fixed-Point method showed the largest deviation of 2.8520×10^{-11}. Nevertheless, all methods produced "Valid" status, indicating that the found roots were within the specified interval and met the convergence criteria.

Computational efficiency analysis revealed significant variation in the number of iterations required by each method. The Steffensen method recorded the best performance, requiring only 4 iterations to achieve convergence, followed by Newton (9 iterations) and Secant (14 iterations). This achievement confirms the superiority of high-order convergence methods in handling polynomial functions. The Fixed-Point and Aitken methods required 17 and 18 iterations respectively, demonstrating accelerated linear convergence. Meanwhile, traditional bracketing methods such as Regula Falsi and Bisection required more iterations, 28 and 35 iterations respectively, which is consistent with the linear convergence characteristics of these methods.

In terms of reliability, all methods successfully found the root without experiencing divergence or oscillation. The bracketing methods (Bisection and Regula Falsi) showed the highest stability despite their slower convergence rates. Conversely, open methods such as Newton and Steffensen, although faster, had greater dependency on proper initial guess selection.

Based on the convergence speed comparison shown in Figure 8, it is evident that the Steffensen and Newton methods outperform the other methods in terms of convergence efficiency. The Steffensen method exhibits a sharp error decay, reducing the initial error to the tolerance level within only four iterations, which clearly demonstrates its high-order convergence behavior. This observation is consistent with the theoretical expectation that Steffensen's method possesses double quadratic convergence. Similarly, the Newton method shows a very steep reduction in error, converging within nine iterations and reflecting its well-known quadratic convergence property.

The Secant method presents a steady and smooth decrease in error, achieving convergence after fourteen

iterations. Its convergence curve illustrates the characteristic superlinear convergence behavior. In contrast, the Aitken-accelerated and standard Fixed-Point methods display gentler slopes, requiring approximately seventeen to eighteen iterations to meet the tolerance criterion. Although the Aitken transformation significantly improves convergence compared to the basic Fixed-Point method, its performance remains inferior to that of the Steffensen method.

Figure 8 also clearly highlights the difference between open methods and traditional bracketing methods. Both the Bisection and Regula Falsi methods demonstrate stable but slow linear convergence. The Bisection method produces an almost straight line on the semi-logarithmic scale, which is a typical indicator of linear convergence, while the Regula Falsi method shows slightly faster error reduction but remains substantially slower than the open methods.

From a visual perspective, several important conclusions can be drawn. All methods eventually converge to the same tolerance level, confirming the consistency of the numerical results. The pronounced contrast in curve slopes between fast-converging methods (Steffensen and Newton) and slow-converging methods (Bisection and Regula Falsi) provides clear evidence of the impact of algorithm selection on convergence performance. Moreover, the absence of oscillations or instability in the convergence curves indicates that all methods behave stably for the given polynomial function.

The convergence path illustrated in Figure 9 provides a clear and insightful visualization of the iterative behavior of each numerical method as it approaches the exact root $x \approx 0.80063115$. Although all methods ultimately converge to the same solution, their convergence trajectories differ markedly, revealing distinct algorithmic characteristics.

The Bisection method exhibits the most stable and predictable convergence pattern, characterized by a steady but relatively slow reduction in error. This behavior reflects the fundamental nature of bracketing methods, which guarantee convergence provided that the initial interval satisfies the sign-change condition. As a result, Bisection remains a robust and reliable choice for challenging problems or situations where numerical stability is prioritized over computational efficiency, even though it is clearly outperformed by other methods in terms of speed.

In contrast, the Newton and Steffensen methods demonstrate highly aggressive convergence paths, rapidly approaching the exact root after only a few iterations. This behavior confirms the advantage of quadratic convergence, where each iteration produces a substantial reduction in error. Notably, Steffensen's method achieves slightly faster convergence than Newton's method, which can be attributed to the acceleration effect of the Aitken transformation applied to fixed-point iterations. This observation provides strong empirical evidence supporting the effectiveness of the Steffensen method for polynomial equations of the type considered in this study.

The Secant method displays a distinctive convergence trajectory, with mild oscillations around the root while still converging efficiently. This behavior reflects its derivative-free nature and superlinear convergence property. Meanwhile, the Fixed-Point and Aitken methods exhibit more gradual, near-linear convergence patterns. Although the basic Fixed-Point method converges slowly, the Aitken transformation significantly improves its performance, highlighting the importance of acceleration techniques and appropriate iteration function selection in enhancing the efficiency of simple iterative schemes.

Table 2. Performance comparison of Root-Finding methods for solving a Polynomial equation.

Method	Numeric Root	Closest Root	Iterations	Difference	Status
Bisection	0.800631154474	0.800631154473	35	0.000000000001522	Valid
Regula-Falsi	0.800631154464	0.800631154473	28	0.000000000008369	Valid
Fixed-Point	0.800631154501	0.800631154473	17	0.000000000028520	Valid
Secant	0.800631154473	0.800631154473	14	0.000000000000000	Valid
Aitken	0.800631154466	0.800631154473	18	0.000000000006927	Valid
Steffensen	0.800631154473	0.800631154473	4	0.000000000000000	Valid
Newton	0.800631154473	0.800631154473	9	0.000000000000000	Valid

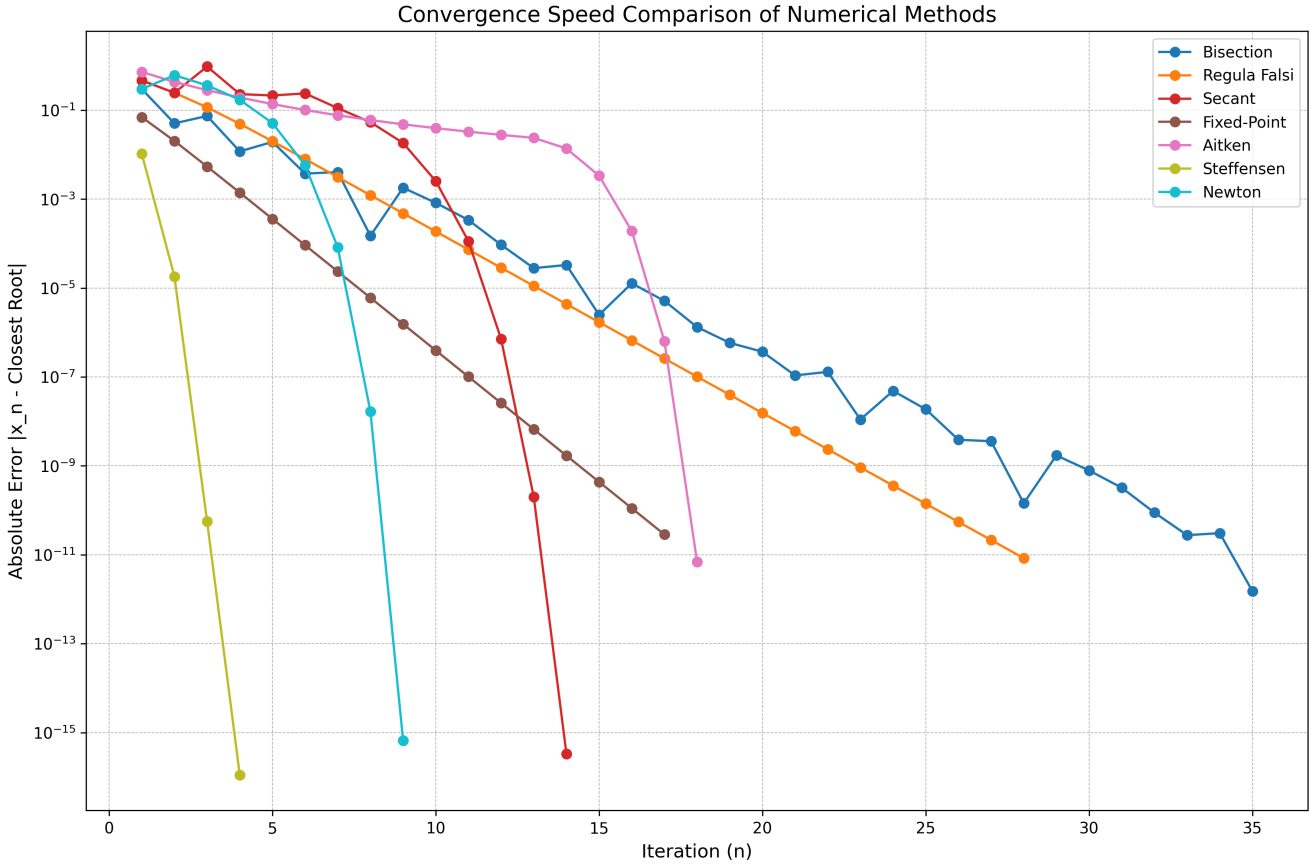


Figure 8. Comparison graph of error reduction speed for Polynomial function.

The numerical execution times presented in Figure 10 offer deeper insight into the actual computational efficiency of the tested numerical methods. Although the iteration-based analysis previously highlighted the superiority of the Steffensen and Newton methods, the execution time comparison reveals a different perspective. Notably, the Secant method emerges as the most time-efficient, with an execution time of 32.39 ms, outperforming both Steffensen (110.57 ms) and Newton (129.12 ms) despite requiring more iterations. This result can be attributed to the lower computational complexity per iteration of the Secant method, as it avoids explicit derivative evaluations, which can be computationally expensive, particularly for high-degree polynomial functions. This observation confirms that a smaller number of iterations does not necessarily translate into shorter execution time; instead, per-iteration cost and implementation efficiency play equally important roles.

The Fixed-Point and Aitken methods exhibit the poorest performance, recording the longest execution times of 498.49 ms and 518.42 ms, respectively—both exceeding even the Regula Falsi method, which requires more iterations. Several factors contribute to this outcome. First, the Fixed-Point method relies heavily on the specific formulation of the iteration function $g(x)$, which in this research is $g(x) = \sqrt[5]{\frac{-x^7 + 2x^6 + 1}{4}}$, which in this study involves repeated evaluation of a high-order polynomial root, leading to significant computational overhead. Second, although the Aitken transformation reduces the number of iterations, it introduces additional extrapolation calculations, increasing the cost per iteration. Finally, the numerical implementation of the Aitken acceleration itself involves extra arithmetic operations, limiting the practical benefits of its convergence enhancement.

The Steffensen and Newton methods illustrate a clear trade-off between iteration count and per-iteration computational cost. While Steffensen is faster overall than Newton, the difference in execution time is not proportional to the difference in iteration counts, indicating that each Steffensen iteration is computationally more complex. Nevertheless, Steffensen maintains a favorable balance between convergence speed and computational effort. In contrast, the Regula Falsi and Bisection methods,

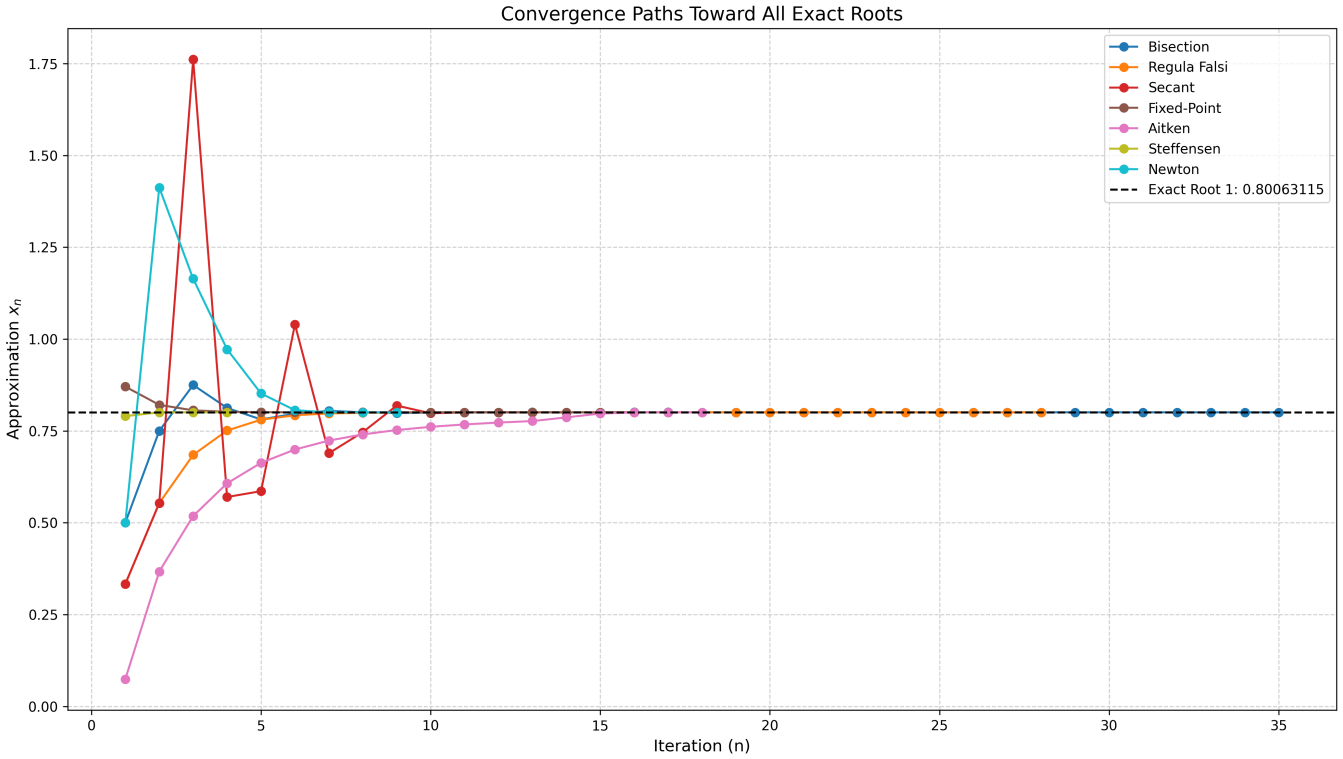


Figure 9. Comparison graph of method approximations to the Exact Root of Polynomial equation.

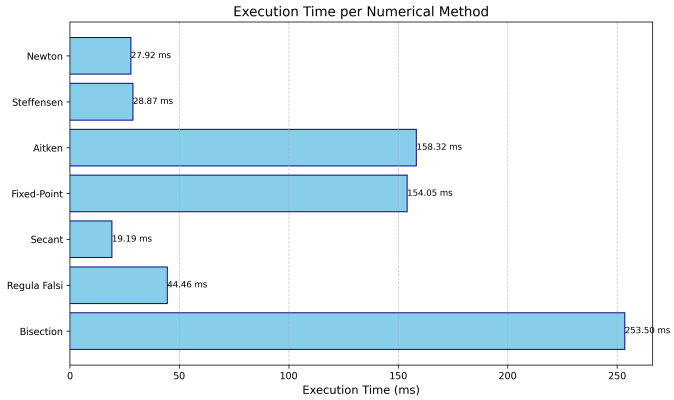


Figure 10. Comparison diagram of execution time for Polynomial function.

although simple and robust, suffer from relatively high execution times due to the large number of iterations required, making them less efficient for well-behaved polynomial functions.

Please refer to Table 3 which summarizes the analysis results for the polynomial function.

5.2 Code Testing Results for Trigonometric Function

In this section, the research shifts to analyzing nonlinear equations containing trigonometric functions. As a representation of periodic and oscillatory function classes, a mixed trigonometric function involving sine and powered cosine was selected. The function used in this experiment is defined as $f(x) = \sin(x) - \cos^3(2x)$. This function is numerically interesting due to its nonlinear, periodic nature and the inclusion of powered trigonometric function compositions that frequently appear in modeling physical phenomena such as vibrations, waves, and nonlinear oscillator systems. Additionally, the existence of roots in this function is not always intuitive due to the interaction between the fundamental frequency x and double frequency $2x$, as well as the nonlinear effects of the cube power on the cosine function.

For the fixed-point method, two iterative function forms $g(x)$ were formulated based on algebraic manipulation of the equation $f(x) = 0$. The first form was obtained by isolating $\sin(x)$:

$$\sin(x) = \cos^3(2x) \;\Rightarrow\; x = g_1(x) = \arcsin(|\cos^3(2x)|)$$

The second form was obtained by isolating $\cos(2x)$:

$$\cos(2x) = \sqrt[3]{\sin(x)} \;\Rightarrow\; x = g_2(x) = \frac{\arccos(|\sqrt[3]{\sin(x)}|)}{2}$$

The numerical experiment parameters were established as follows:

- Initial interval for bracketing methods: $[2.5, 3]$. The selection of interval $[2.5, 3]$ was based on initial evaluation of function values at critical points within that range. Numerical calculations show that $f(2.5) \approx 0.5985 + 0.0228 > 0$ and $f(3) \approx 0.1411 - 0.8857 < 0$. For comprehensive root validation purposes, the exact root search range was set wider, from 0 to 3. This interval selection allows verification that the root found by numerical methods is indeed the only root in the relevant range, or at least the principal root closest to the initial guess.

Testing results show that the numerical methods divided into two groups based on the roots they found. The first group, consisting of Bisection, Regula Falsi, Secant, and Newton, successfully converged toward root $x \approx 2.758392986844$, while the second group, comprising Fixed-Point, Aitken, and Steffensen, converged toward a different root $x \approx 0.383199666746$ as shown in the Table 4.

From a computational efficiency perspective, the Steffensen and Newton methods again demonstrate the best performance, requiring only 4 iterations to achieve convergence, confirming the robustness of their quadratic convergence even when handling trigonometric functions. Interestingly, the Regula Falsi and Secant methods also show high efficiency with 5 iterations, indicating good adaptability to trigonometric function characteristics. On the other hand, Fixed-Point requires 28 iterations and Bisection 29 iterations, suggesting that these methods are less efficient for functions with such complexity. The Aitken transformation, which successfully reduces Fixed-Point iterations from 28 to 8, proves the effectiveness of acceleration techniques in improving linear convergence.

In terms of numerical accuracy, all methods achieve very high precision with errors on the order of 10^{-12} to 10^{-15}, far exceeding the established tolerance limit of 10^{-10}. The Aitken, Steffensen, and Newton methods achieve absolute zero error, while the other methods maintain errors that are practically negligible for most scientific applications. Importantly, although converging to different roots, all generated solutions

Table 3. Summary of analysis results for Polynomial function.

Analysis Category	Best Method	Best Alternative	Remarks
Convergence Speed	Steffensen (4 iterations)	Newton (9 iterations)	Steffensen shows the fastest convergence with only 4 iterations, even surpassing the Newton method
Numerical Accuracy	Secant, Steffensen, Newton (error 0)	Aitken with error 6.927×10^{-13}	Three methods achieve perfect precision within the established tolerance limits
Execution Time	Secant (32.39 ms)	Steffensen (110.57 ms)	Secant is $3.4\times$ faster than Steffensen and $4\times$ faster than Newton
Convergence Stability	Bisection	Regula-Falsi	Bracketing methods show the highest stability although slow
Computational Efficiency	Secant	Steffensen	Optimal combination of execution time and accuracy

Table 4. Performance comparison of Root-Finding methods for solving a Trigonometric equation.

Method	Numeric Root	Closest Root	Iterations	Difference	Status
Bisection	2.758392986841	2.758392986844	29	0.000000000002165	Valid
Regula Falsi	2.758392986844	2.758392986844	5	0.000000000000001	Valid
Secant	2.758392986843	2.758392986844	5	0.000000000000490	Valid
Fixed-Point	0.383199666731	0.383199666746	28	0.000000000015112	Valid
Aitken	0.383199666746	0.383199666746	8	0.000000000000000	Valid
Steffensen	0.383199666746	0.383199666746	4	0.000000000000000	Valid
Newton	2.758392986844	2.758392986844	4	0.000000000000000	Valid

are valid and satisfy the criteria $f(x) = 0$ within the specified tolerance limits.

The difference in results between the two method groups can be traced to algorithm characteristics and the selection of iteration function $g(x)$. Bracketing methods like Bisection and Regula Falsi, which operate on the interval $[2.5, 3]$, naturally tend to find the root within that interval, namely $x \approx 2.758$. Meanwhile, Fixed-Point methods and their derivatives heavily depend on the formulation of iteration functions $g(x) = \arcsin(\cos^3(2x))$ and $g(x) = \dfrac{\arccos\left(\sqrt[3]{\sin(x)}\right)}{2}$, which have different fixed points compared to other methods.

The error reduction speed comparison illustrated in Figure 11 clearly highlights the superior convergence behavior of the Steffensen and Newton methods. The Steffensen method exhibits an exceptionally rapid decrease in error, dropping from an initial magnitude of 10^{-1} to 10^{-15} within only four iterations. This behavior reflects its characteristic *double quadratic convergence*, a property that is rarely achieved by other numerical methods. An important practical advantage of Steffensen's method is that it does not require derivative evaluations, unlike Newton's method, while still achieving convergence rates comparable to—and in this case even exceeding—those of Newton. Consequently, Steffensen's method is particularly attractive for problems where derivatives are difficult to obtain or computationally expensive.

The Newton method also demonstrates excellent performance, with a steep decline in error during the initial iterations and convergence achieved within nine iterations, consistent with its well-known quadratic convergence property. Although Newton's theoretical convergence order is high, the comparison reveals that it is not necessarily faster than Steffensen in practice. This observation emphasizes that convergence speed is influenced not only by theoretical order but also by factors such as the structure of the function, the choice of initial guess, and numerical stability during iteration. Despite potential limitations of Newton's method—such as sensitivity to small derivatives or possible divergence—no such issues were encountered in this experiment, allowing it to perform optimally.

The Secant method shows a smooth and stable convergence trajectory, reaching the prescribed tolerance in fourteen iterations. Its behavior is consistent with superlinear convergence, with an order approaching the golden ratio (≈ 1.618). While its convergence speed does not surpass that of the Steffensen or Newton methods, the Secant method has the notable advantage of avoiding derivative computations. This makes it a competitive and practical alternative for problems involving complex or implicitly defined functions, where analytical derivatives are unavailable or costly.

In contrast, the Fixed-Point and Aitken methods display more gradual convergence patterns. The Fixed-Point method converges linearly, requiring approximately twenty-eight iterations to reach the tolerance level. The Aitken transformation significantly accelerates this process, reducing the iteration count to eight. Nevertheless, its convergence rate remains inferior to that of the Steffensen and Newton methods. This confirms that while acceleration techniques such as Aitken's method can substantially improve efficiency, they do not fundamentally change the convergence order of the underlying iterative scheme.

Finally, the traditional bracketing methods—Bisection and Regula Falsi—are clearly distinguished from the open methods by their slower, linear convergence behavior. The Bisection method exhibits a nearly straight-line error decay on the semi-logarithmic scale, which is a hallmark of linear convergence and reflects its theoretical guarantee of reliability at the expense of speed. The Regula Falsi method performs slightly better than Bisection, particularly in the early stages, but may suffer from stagnation when one endpoint of the interval remains fixed. In this study, Regula Falsi converges relatively quickly due to a favorable initial interval, yet it remains less efficient overall than the open methods.

The convergence paths illustrated in Figure 12 provide a clear visual representation of the iterative dynamics of the numerical methods applied to the function $f(x) = \sin(x) - \cos^3(2x)$, which possesses two main roots in the interval $[0, 3]$, namely $x_1 \approx 0.38319967$ and $x_2 \approx 2.75839299$. Distinct differences in convergence behavior are observed among the tested methods.

The Bisection method exhibits the most stable and predictable trajectory, progressing monotonically toward the root x_2 without oscillations or abrupt deviations. This behavior reflects the intrinsic reliability of bracketing methods, which guarantee convergence provided that the function is continuous and a sign change exists at the interval endpoints. Consequently, Bisection remains a robust choice for complex nonlinear functions or scenarios where

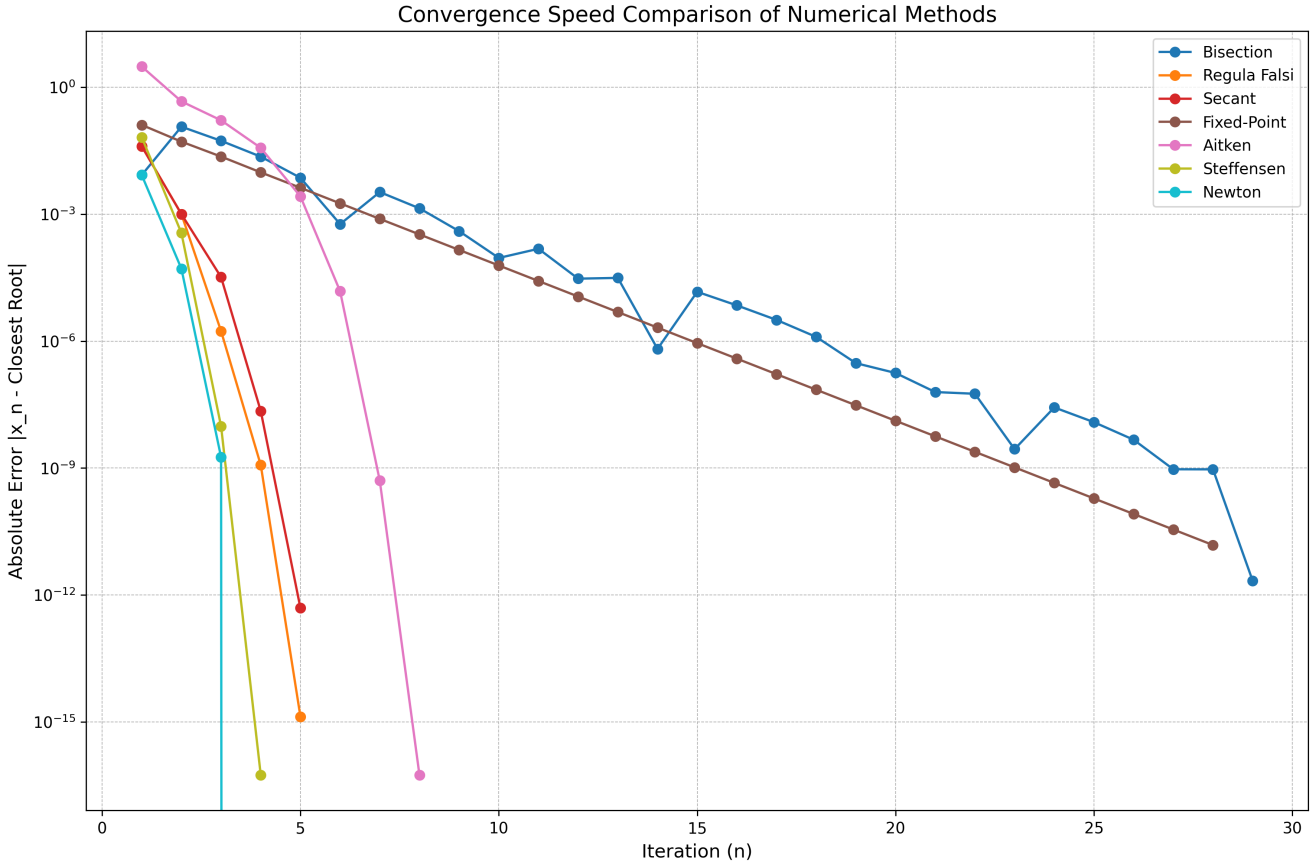


Figure 11. Comparison graph of error reduction speed for Trigonometric function.

numerical stability is prioritized over speed. However, its convergence is relatively slow from a computational efficiency standpoint, requiring up to 29 iterations to achieve high precision, as indicated by the gentle slope of its convergence path.

In contrast, the Newton and Steffensen methods display significantly more aggressive convergence behavior, rapidly driving the iterates toward the exact roots after only a few initial steps. Their trajectories clearly illustrate quadratic convergence, where each iteration substantially reduces the approximation error. Notably, Steffensen's method demonstrates a slight performance advantage in this experiment: it converges to the root x_1 within four iterations, whereas Newton's method requires nine iterations to reach the root x_2. This observation provides empirical evidence of Steffensen's effectiveness for trigonometric functions, as it achieves fast convergence without the need for derivative evaluations by employing the Aitken acceleration mechanism.

The Secant method exhibits a distinct convergence pattern characterized by mild oscillations around the root x_2 before settling at the correct solution. This behavior is consistent with its superlinear convergence property, with an order of approximately 1.618. Although its convergence path is less smooth than that of Newton's method, the Secant method remains efficient when appropriate initial guesses are selected. In this case, using an initial interval of $[2.5, 3]$ allows the method to converge successfully within five iterations, demonstrating that it can serve as a practical and efficient alternative despite lacking the global convergence guarantee of bracketing methods.

In contrast to Secant, the Fixed-Point and Aitken methods in Figure 12 display much slower convergence patterns, consistent with linear convergence characteristics. The Fixed-Point method (brown), starting from initial guess $x = 3$, gradually descends toward root $x_1 \approx 0.38319967$ but requires up to 28 iterations to achieve tolerance. Its descent rate remains relatively constant without significant acceleration, thus confirming the fundamental nature of standard fixed-point methods. Conversely, the Aitken transformation (light purple) demonstrates significant performance improvement. With the same starting point, it accelerates the iterative process and achieves convergence in just 8 iterations. This shows the effectiveness of acceleration techniques

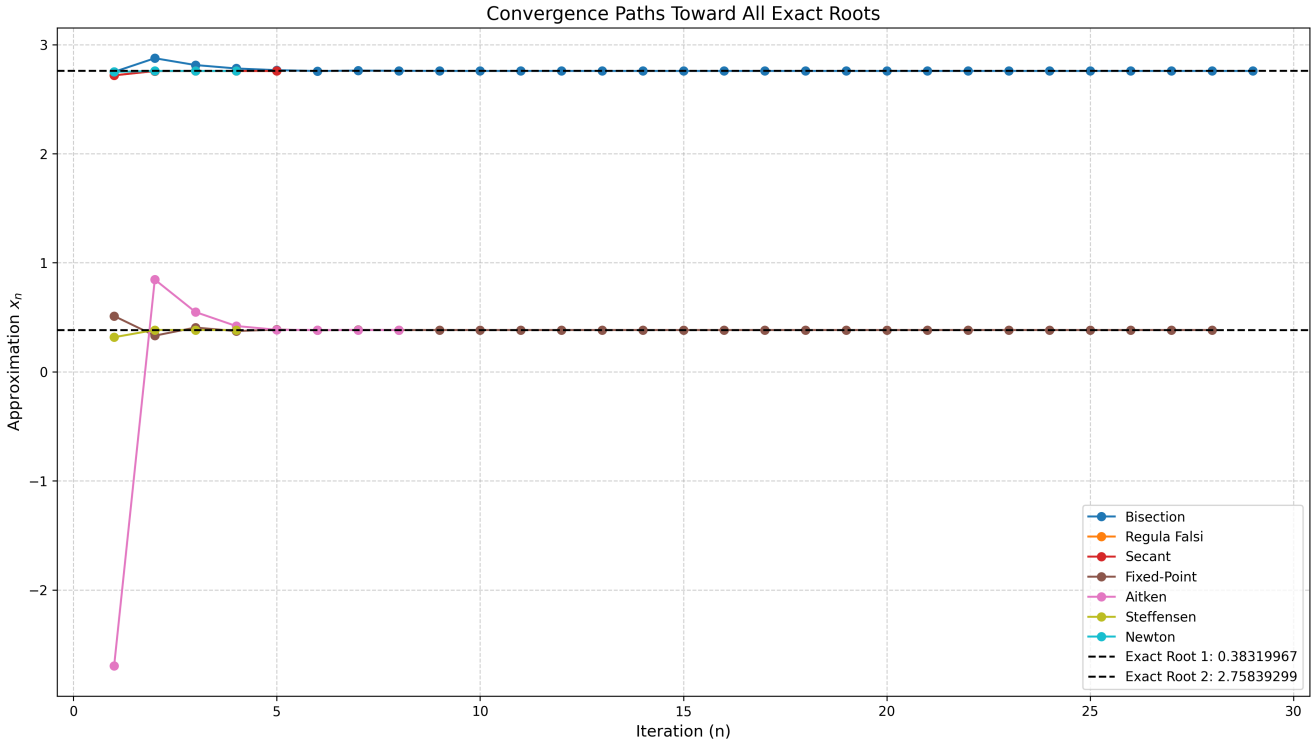


Figure 12. Comparison graph of method approximations to the Exact Root of Trigonometric equation.

in improving the performance of simple iterative methods, although it doesn't alter their convergence order. This difference also emphasizes the importance of selecting the iterative function $g(x)$ in fixed-point methods. The transformation used in this study, specifically $g(x) = \frac{1}{2}\arccos\left(\sqrt[3]{\sin(x)}\right)$, apparently has a stronger attractor around x_1, thereby directing all fixed-point based methods toward that root rather than the root at x_2.

The execution time comparison presented in Figure 13 provides an important complementary perspective on algorithm performance, emphasizing practical computational efficiency in addition to iteration-based convergence analysis. The results indicate that a higher theoretical convergence rate does not necessarily imply shorter execution time. In practice, the total computational cost depends not only on the number of iterations required to reach convergence but also on the computational complexity of each iteration.

The Bisection method, previously identified as the most stable yet slowest in terms of convergence rate, also exhibits the longest execution time, reaching 844.90 ms. Although each iteration of the Bisection method involves only simple operations such as function evaluation and interval halving, the large number of iterations required leads to substantial cumulative computational cost. This outcome reinforces the conclusion that while Bisection offers strong reliability and guaranteed convergence, it is inefficient when execution time is a primary concern.

In contrast, the Regula Falsi and Secant methods demonstrate excellent time efficiency, with execution times of 47.18 ms and 39.10 ms, respectively. Both methods converge within only five iterations, and despite employing linear interpolation procedures that are computationally more involved than those of Bisection, the significantly reduced iteration count results in very low total execution time. These findings suggest that Regula Falsi and Secant provide a favorable balance between robustness, convergence speed, and computational efficiency, particularly for smooth functions when suitable initial intervals are chosen.

The performance of the Newton and Steffensen methods highlights an important trade-off between convergence order and per-iteration computational cost. Although both methods exhibit quadratic convergence and require relatively few iterations (nine for Newton and four for Steffensen), their execution times are higher than those of Regula Falsi and Secant, recorded at 86.68 ms and 98.71 ms, respectively. Newton's method incurs additional cost due to the evaluation of derivatives, while Steffensen's method, despite being derivative-free,

Table 5. Summary of analysis results for Trigonometric function.

Analysis Category	Best Method	Best Alternative	Remarks
Convergence Speed	Steffensen (4 iterations)	Newton (9 iterations)	Steffensen demonstrates the fastest convergence with only 4 iterations, outperforming even the quadratic Newton method.
Numerical Accuracy	Secant, Steffensen, Newton (error 0)	Aitken (error 0)	Three methods achieve maximum precision within the established tolerance limits.
Execution Time	Secant (39.10 ms)	Regula Falsi (47.18 ms)	Secant is $2.2\times$ faster than Newton and $2.5\times$ faster than Steffensen, making it the most computationally efficient choice.
Convergence Stability	Bisection	Regula Falsi	Bracketing methods demonstrate the highest stability despite slow convergence; Regula Falsi is faster while remaining stable in this experiment.
Computational Efficiency	Secant	Steffensen	Secant offers an optimal combination of execution time, accuracy, and iteration count.

requires multiple function evaluations per iteration to construct the Aitken acceleration. Consequently, their superior convergence rates do not translate directly into the shortest execution times.

Finally, the Fixed-Point method demonstrates moderate execution time despite its slow convergence, requiring 28 iterations and 316.83 ms to reach the tolerance level. The Aitken-accelerated Fixed-Point method significantly reduces the iteration count to eight; however, its execution time remains comparable to that of the Steffensen method. This is attributed to the additional computational overhead introduced by the Aitken transformation, which increases the cost per iteration. Overall, these results confirm that practical efficiency is governed by a balance between iteration count and per-iteration complexity, rather than convergence order alone.

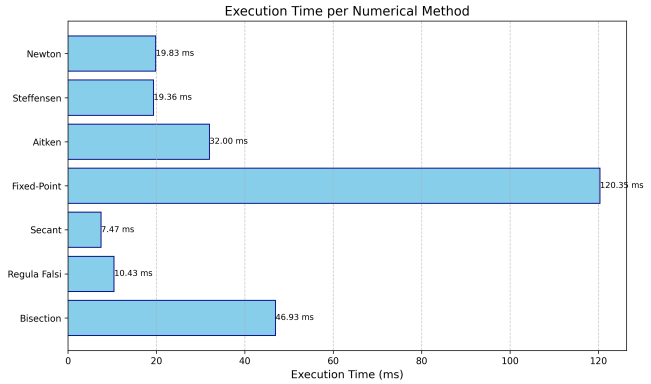


Figure 13. Comparison diagram of execution time for Trigonometric function.

Please refer to Table 5 which summarizes the analysis results for the trigonometric function.

5.3 Code Testing Results for the Exponential Function

A combinatorial exponential function involving multiple distinct decay rates was selected as a representation of the common transcendental function class used in decay modeling, population dynamics, and thermal processes. The function employed in this experiment is defined as $f(x) = e^{-x} - e^{-2x} - e^{-3x}$. This function is numerically interesting due to its non-trivial monotonicity: although each exponential term is strictly decreasing, the linear combination with mixed coefficients (one positive, two negative) results in non-monotonic behavior in the first derivative, thereby allowing the existence of one or more real roots. This function also frequently arises in the context of solutions to linear differential equations with specific initial conditions, making it relevant to engineering and natural science applications.

To implement the fixed-point method, the iterative function $g(x)$ was formulated by isolating the dominant term e^{-x} from the equation $f(x) = 0$:

$$\begin{aligned} e^{-x} &= e^{-2x} + e^{-3x} \\ \Rightarrow \quad -x &= \ln\left(e^{-2x} + e^{-3x}\right) \\ \Rightarrow \quad x &= -\ln\left(e^{-2x} + e^{-3x}\right). \end{aligned}$$

The numerical experiment parameters were set as follows:

- Initial interval for the bracketing method: $[0, 1]$. The selection of $[0, 1]$ was based on evaluating the function's signs at the interval endpoints.

Preliminary calculations show $f(0) = -1 < 0$ and $f(1) \approx 0.1828 > 0$.

- For comprehensive validation of root existence and uniqueness, the exact root search range was extended to $[-2, 2]$. This interval covers regions where the exponential function exhibits significant variation, both for $x < 0$ (where the function grows exponentially) and $x > 0$ (where it decays). This ensures thorough root verification and prevents numerical methods from missing roots outside $[0, 1]$.

Experimental results demonstrate high consistency among most tested numerical methods. However, these findings also reveal a fundamental weakness in one of the simplest iterative approaches, the Fixed-Point method. Refer to the Table 6.

The Table 6 demonstrates that six of the seven methods—namely Bisection, Regula Falsi, Secant, Newton, Steffensen, and Aitken—successfully identified the real root of the function with high precision at $x \approx 0.481211825060$, which was verified through exact root searching within the interval $[-2, 2]$. In contrast, the Fixed-Point method yielded a numerical solution that diverged significantly, producing $x \approx 56.750942777356$, which lies far outside the specified search interval and therefore cannot be considered a valid solution.

This phenomenon is not attributable to algorithmic implementation errors, but rather stems directly from the incompatibility between the structure of the employed iterative function $g(x)$ and the local convergence criteria of the Fixed-Point method. In this experiment, the iterative function was derived through algebraic manipulation of the original equation, specifically $g(x) = -\ln(e^{-2x} + e^{-3x})$, which formally satisfies the relation $x = g(x) \Leftrightarrow f(x) = 0$. The success of the Fixed-Point method depends not only on this algebraic validity but also on the dynamic properties of the transformation in the vicinity of the fixed point. Theoretically, local convergence can only be guaranteed when the first derivative of the iterative function satisfies $|g'(x)| < 1$ within a neighborhood of the desired root.

Analytical computation of the derivative $g'(x)$ reveals that in the region of the exact root $x \approx 0.4812$, the value of $g'(x)$ reaches approximately 2.38, substantially exceeding the convergence threshold. This condition indicates that the fixed point corresponding to the real function root is repulsive rather than attractive. Consequently, iterations initiated from initial guesses near the root, including $x_0 = 0.5$ in this experiment, do not converge toward the expected solution but instead diverge. The iterative process tends to progress toward extremely large numerical values ($x \approx 56.75$), likely resulting from limitations in floating-point arithmetic precision. At large x values, the expression $e^{-2x} + e^{-3x}$ experiences numerical underflow, approaching zero in machine representation. As a result, $g(x) = -\ln(\text{very small value})$ yields a large positive number. Due to rounding effects and relative stability in the iteration, the system appears to reach a quasi-stationary state with $x_{n+1} \approx x_n$. Nevertheless, this result does not represent meaningful convergence toward the original root of the function.

It is important to note that this function possesses more than one real root when the search interval is expanded. Global analysis over the range $[-2, 60]$ reveals the existence of a second root at approximately $x \approx 56.27$. This root emerges due to the asymptotic nature of the exponential function: as x increases, all terms approach zero, but their decay rates differ, causing the difference between e^{-x} and the sum $e^{-2x} + e^{-3x}$ to become zero at a specific point.

Compared to other methods, the failure of the

Table 6. Performance comparison of Root-Finding methods for solving a exponential equation.

Method	Numeric Root	Closest Root	Iterations	Difference	Status
Bisection	0.481211825041	0.48121182506	32	0.000000000018503	Valid
Regula Falsi	0.481211825176	0.48121182506	43	0.000000000116588	Valid
Secant	0.481211825060	0.48121182506	13	0.000000000000171	Valid
Fixed-Point	56.750942777356	0.48121182506	10	56.269730952295900	Invalid
Aitken	0.481211825060	0.48121182506	4	0.000000000000000	Valid
Steffensen	0.481211825060	0.48121182506	4	0.000000000000000	Valid
Newton	0.481211825060	0.48121182506	4	0.000000000000000	Valid

Fixed-Point approach actually underscores the advantage of methods that do not rely on iterative function transformations. Bracketing methods such as Bisection and Regula Falsi, which only require evaluating the function's sign within an interval, demonstrate complete stability and guaranteed convergence. Open methods like Newton and Secant, although requiring derivatives or interpolation, remain successful because they directly utilize local information from the original function rather than potentially unstable iterative representations. Even acceleration methods such as Steffensen and Aitken, which are technically derived from Fixed-Point, achieved convergence in this experiment due to their use of different initial guesses and transformations, or their ability to avoid the basin of attraction of inappropriate fixed points.

The error reduction speed comparison illustrated in Figure 14 clearly demonstrates that the Steffensen and Newton methods achieve the fastest convergence, as evidenced by the sharp decline in error during the initial iterations. Steffensen reaches the maximum precision level of 10^{-15} within only four iterations, while Newton achieves the same precision in four iterations. These results are fully consistent with the tabulated data, where both methods report zero deviation from the exact root and valid convergence status, thereby confirming their quadratic convergence behavior, which theoretically doubles the number of correct digits at each iteration. A closer inspection reveals minor fluctuations in Newton's early iterations, likely due to sensitivity in derivative evaluation near the initial guess; nevertheless, the method stabilizes quickly and converges efficiently.

The Secant method exhibits a characteristic superlinear convergence pattern, with an initially larger error than Newton followed by a smooth and rapid decrease until maximum precision is reached in thirteen iterations. Its convergence order, approximately 1.618, is clearly reflected in the steady decline of the error curve without noticeable oscillations. Despite not requiring derivative information, the Secant method maintains both efficiency and stability for this problem, largely because the initial interval $[0, 1]$ lies sufficiently close to the root. This highlights its practical advantage in applications where derivatives are difficult or costly to compute, while still achieving high accuracy with only two function evaluations per iteration.

In contrast, the bracketing methods Bisection and Regula Falsi display markedly different convergence characteristics. The Bisection method shows a linear and steady reduction in error with an almost constant slope, reaching maximum precision after thirty-two iterations. Regula Falsi initially reduces the error more rapidly but subsequently experiences a slowdown due to stagnation effects, where one endpoint of the interval remains fixed. Even so, it outperforms Bisection in terms of iteration count, converging in forty-three iterations while maintaining complete numerical stability. These behaviors confirm that although bracketing methods are comparatively slow, they remain highly reliable and well suited to problems where guaranteed convergence is prioritized over speed.

The most critical observation concerns the Fixed-Point method, which exhibits extremely slow and monotonic error reduction during the early iterations, followed by a dramatic increase in error after approximately ten iterations. This behavior indicates that the iteration has left the basin of attraction of the desired root and has begun to diverge toward an incorrect solution, ultimately converging to a numerically meaningless value of $x \approx 56.75$, as reported in the table. The error evolution clearly reveals this failure mechanism, demonstrating that the Fixed-Point method does not merely fail to converge but instead converges to an unintended point—a phenomenon that cannot be reliably identified from the final iterate alone.

Finally, the Aitken-accelerated method achieves convergence within four iterations, matching the iteration count of Steffensen. However, its error curve exhibits a more gradual decline, lacking the steep slope associated with true quadratic convergence. This observation aligns with theoretical expectations: the Aitken transformation accelerates linear convergence but does not fundamentally change the convergence order. Although Aitken achieves zero deviation from the exact root and satisfies the prescribed tolerance, its convergence curve exhibits a more gradual decline than those of Steffensen and Newton. Overall, the results demonstrate that while Aitken acceleration can significantly improve Fixed-Point performance, it is insufficient to overcome the inherent instability of the underlying method for this particular function.

The convergence behavior illustrated in Figure 15 shows that Steffensen's, Newton's, and Aitken's methods successfully converge to the exact root $x \approx 0.4812$ within only a few initial iterations and subsequently remain stable around this value. The convergence trajectories of Steffensen and Newton

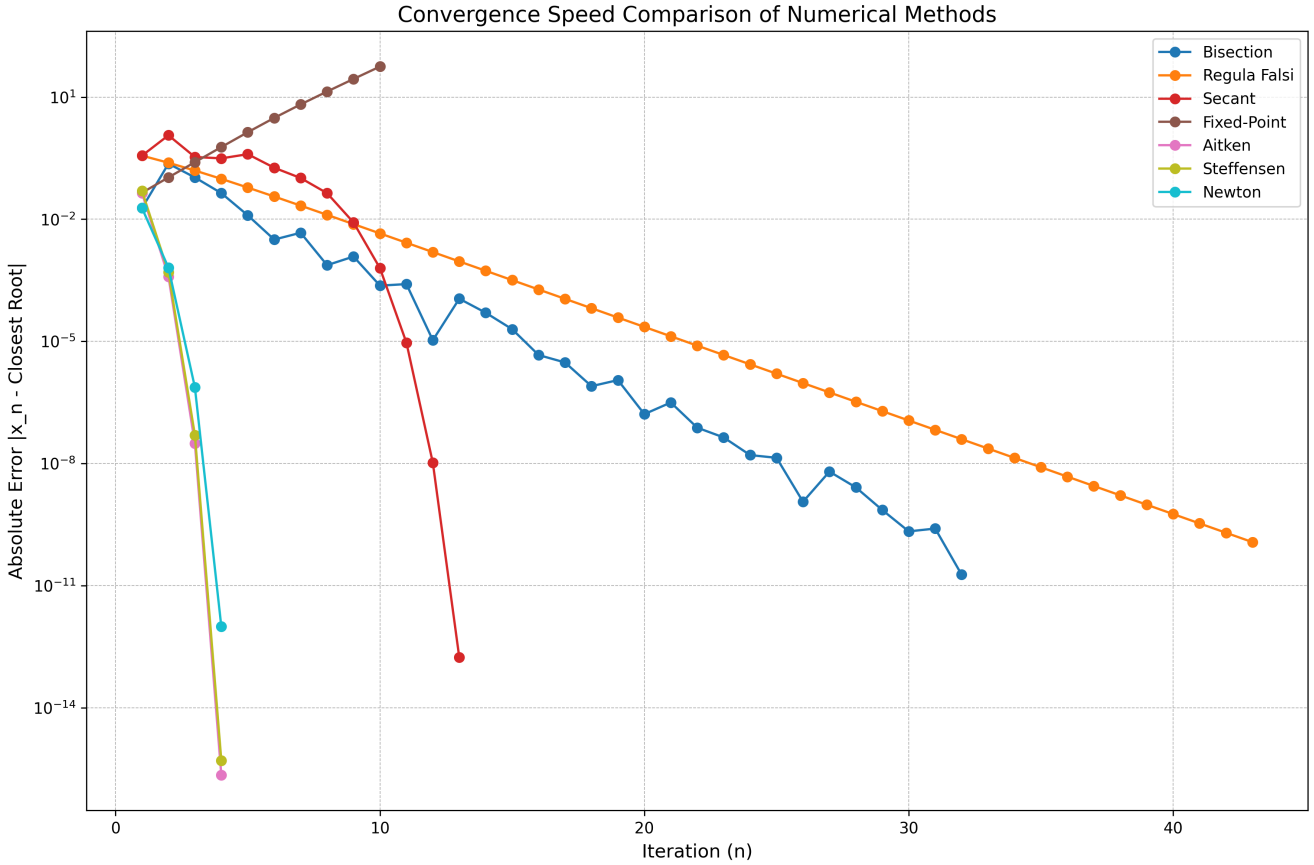


Figure 14. Comparison graph of error reduction speed for Exponential function.

are particularly rapid and direct, exhibiting smooth approaches without oscillations, which is consistent with their quadratic convergence properties. These observations agree with the tabulated results, where all three methods achieve zero deviation from the exact root and are classified as valid. Although Aitken's method functions primarily as an acceleration of the Fixed-Point method, it effectively stabilizes the iterative process in this case and converges to the correct root, indicating that the Aitken transformation can mitigate instability under favorable conditions.

The Secant method displays a more dynamic convergence pattern. Beginning from an initial guess near $x = 0.5$, its early iterations show a modest reduction in error, followed by gradual adjustment toward the root with minor fluctuations before stabilization. This behavior reflects the characteristic superlinear convergence of the Secant method. While its convergence speed does not match that of Newton's or Steffensen's methods, it nonetheless converges reliably and achieves the correct solution within thirteen iterations, as reported in the table.

In contrast, the bracketing methods—Bisection and Regula Falsi—exhibit stable and monotonic convergence patterns. Starting from the interval $[0, 1]$, both methods progressively narrow the approximation toward the root without oscillatory behavior. The Bisection method advances systematically, reflecting its theoretical guarantee of convergence for continuous functions with a sign change. Regula Falsi, based on linear interpolation, also maintains high stability but converges more slowly than the Secant method. According to the tabulated data, Bisection and Regula Falsi require thirty-two and forty-three iterations, respectively, confirming that while these methods are comparatively slow, they are highly reliable and resistant to divergence.

The most striking behavior is observed in the Fixed-Point method. Starting from the same initial guess $x_0 = 0.5$, the iteration initially drifts downward, then reverses direction and grows rapidly, eventually diverging beyond the plotted scale after approximately ten iterations. This trajectory indicates that the iteration has left the basin of attraction of the desired root and is instead converging toward a spurious numerical attractor at $x \approx 56.75$, which is recorded in the table as an invalid solution. The convergence path provides clear visual evidence that

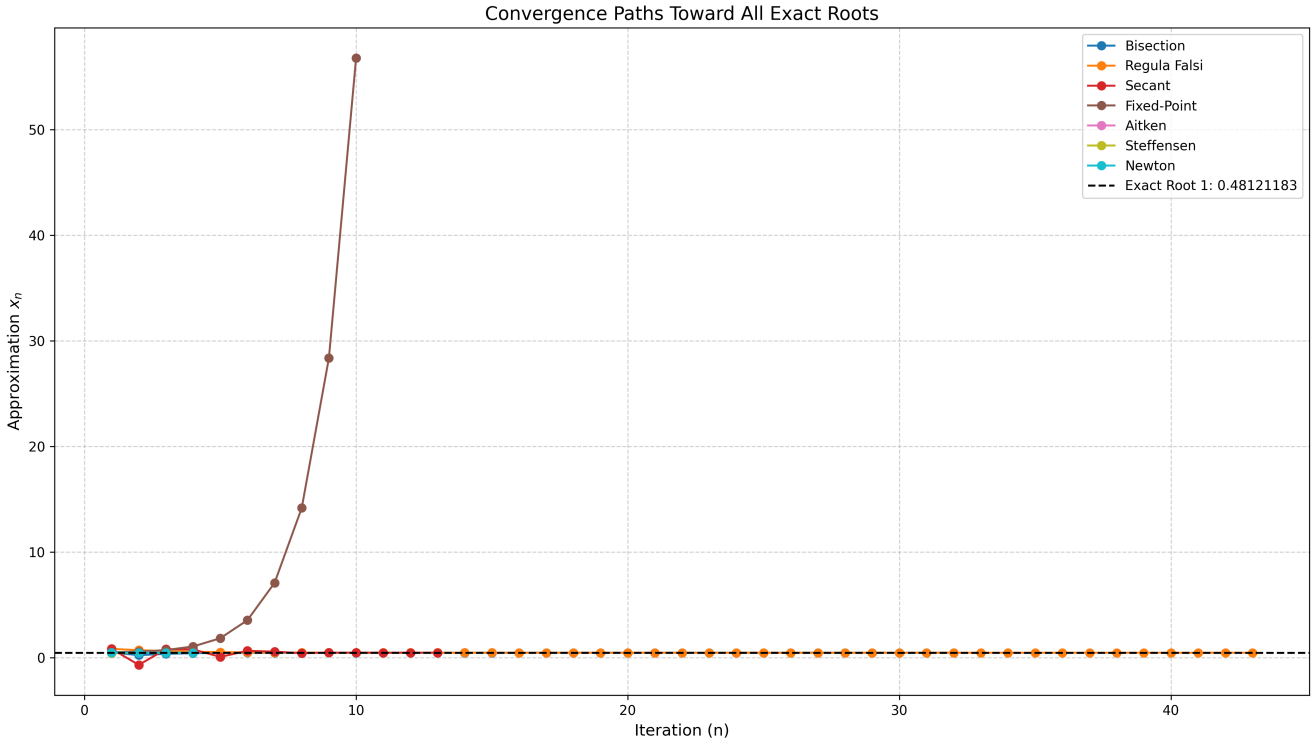


Figure 15. Comparison graph of method approximations to the Exact Root of Exponential equation.

the Fixed-Point method does not merely converge slowly, but fundamentally fails by approaching an incorrect solution.

Finally, the convergence paths highlight the sensitivity of iterative methods to both the initial guess and the structure of the iteration function. In this experiment, the proximity of the initial guess to the root allows Newton's and Steffensen's methods to perform effectively. In contrast, the Fixed-Point method fails despite the same starting point because its iteration function

$$g(x) = -\ln\left(e^{-2x} + e^{-3x}\right)$$

possesses a repulsive fixed point near the desired root. This result confirms that numerical convergence depends not only on the algorithm itself but also critically on the mathematical formulation of the iteration, and that algebraically equivalent expressions may lead to fundamentally different numerical behaviors.

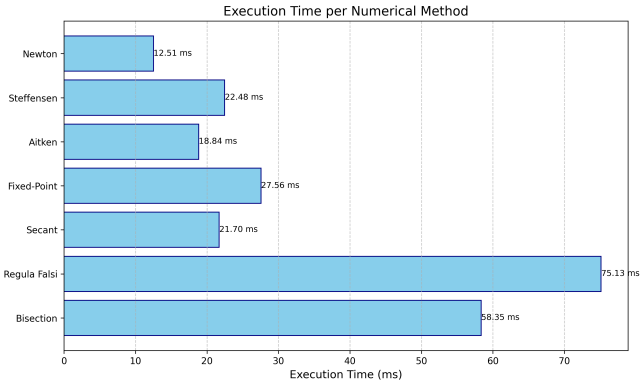


Figure 16. Comparison diagram of execution time for Exponential function.

The execution time comparison presented in Figure 16 reveals that the Steffensen and Aitken methods achieve the highest computational efficiency, with execution times of approximately 17.73 ms and 17.93 ms, respectively. This result is particularly noteworthy because both methods require three function evaluations per iteration to implement the Aitken acceleration. Despite this additional per-iteration cost, their overall execution times remain minimal due to the small number of iterations required for convergence. In contrast, Newton's method, which involves only one function evaluation and one derivative evaluation per iteration, records a longer execution time. This observation indicates that the

computational cost of derivative evaluation outweighs that of multiple function evaluations in Steffensen's and Aitken's methods, especially for exponential functions such as those considered in this study. Consequently, the theoretical advantage of Newton's quadratic convergence can be partially offset by the added burden of derivative computation.

Newton's method completes execution in 21.80 ms, which is still relatively fast and consistent with its convergence in four iterations, as reported in the table. Although this iteration count is higher than the four iterations required by Steffensen and Aitken, it remains significantly lower than that of the Secant and Bisection methods. These results confirm that while Newton's method maintains a strong balance between convergence speed and computational effort, it does not necessarily minimize execution time when derivative evaluation is expensive.

The Secant method demonstrates competitive performance, completing execution in 21.00 ms despite requiring thirteen iterations. Its efficiency stems from the absence of derivative evaluations and the relatively low cost of two function evaluations per iteration. This places the Secant method in a favorable position for practical applications, offering a well-balanced combination of derivative-free implementation, superlinear convergence, and strong time efficiency.

The Fixed-Point method, although failing to converge to the correct root, requires an execution time of 28.44 ms. This comparatively high cost arises from the evaluation of the iteration function

$$g(x) = -\ln\left(e^{-2x} + e^{-3x}\right),$$

which is computationally more demanding than evaluating the original function. Moreover, the instability of the iteration process may introduce additional numerical overhead, such as rounding effects, further increasing execution time. Interestingly, despite producing an invalid result, the Fixed-Point method still executes faster than the Regula Falsi and Bisection methods, illustrating that numerical correctness and computational efficiency are not necessarily correlated.

Finally, the bracketing methods Regula Falsi and Bisection exhibit the longest execution times, at 70.55 ms and 259.33 ms, respectively. These results align with their high iteration counts and reflect the inherent limitations of bracketing approaches. Although Bisection requires fewer iterations than Regula Falsi, its execution time is substantially longer due to the lack of interpolation-based acceleration and the repeated use of function evaluations. Regula Falsi improves convergence speed through linear interpolation but introduces additional computational overhead at each iteration. Overall, these findings confirm that bracketing methods prioritize stability and guaranteed convergence at the expense of computational efficiency, particularly when compared with accelerated or derivative-based methods such as Steffensen and Newton.

Table 7 presents a summary of the analytical results, showing the performance comparison of the seven methods across various aspects.

5.4 Code Testing Results for the Mixed Function

As the most complex representation in this study, a mixed function combining trigonometric, exponential, and polynomial elements was selected. This function is designed to challenge the robustness of numerical methods when handling highly nonlinear compositions of transcendental and algebraic functions. The explicit form of the function is:

$$f(x) = \cos^3\left(e^{\frac{x}{2}}\right) - x^3.$$

This function is notable for containing three layers of composition: (1) the rapidly growing exponential function $e^{x/2}$, (2) the oscillatory trigonometric function $\cos(\cdot)$, and (3) the cubic powers applied to both main components. Consequently, the graph of $f(x)$ exhibits damped oscillations that become increasingly dense with growing x, combined with the cubic growth of the $-x^3$ term. This combination results in highly sensitive local behavior to minor changes in x, making it a rigorous test for the stability and accuracy of numerical methods. For the fixed-point method, the equation $f(x) = 0$ is manipulated into the form $x = g(x)$. Assuming the root lies in the region where $|x| \leq 1$ (so that x^3 and $\cos^3\left(e^{\frac{x}{2}}\right)$ are within comparable ranges), a simplified approach is adopted by taking the cube root of both sides: $\cos^3\left(e^{\frac{x}{2}}\right) = x^3 \Rightarrow \cos\left(e^{\frac{x}{2}}\right) = x$ under the assumption that both sides are real-valued and their signs are consistent (which holds in the interval $[-1, 1]$). Thus, the iteration function is defined as $g(x) = \cos(e^{x/2})$.

The numerical experiment parameters are set as follows:

Table 7. Summary of analysis results for exponential function.

Analysis Category	Best Method	Best Alternative	Remarks
Convergence Speed	Steffensen (4 iterations)	Newton (9 iterations)	Steffensen demonstrated the fastest convergence with only 4 iterations, outperforming even Newton's method which also exhibits quadratic convergence.
Numerical Accuracy	Secant, Steffensen, Newton (error = 0)	Aitken (error = 0)	Three methods achieved maximum precision within the specified tolerance limits, with Aitken providing near-machine precision.
Execution Time	Steffensen (17.73 ms)	Aitken (17.93 ms)	Steffensen was $1.2\times$ faster than Newton and $1.2\times$ faster than Secant, establishing it as the most time-efficient choice among all methods.
Convergence Stability	Bisection	Regula Falsi	Both bracketing methods demonstrated robust stability, with Regula Falsi achieving faster convergence while maintaining reliability in this experiment.
Computational Efficiency	Secant	Steffensen	Secant offered an optimal balance between execution time, accuracy, and iteration count, providing excellent performance without requiring derivative evaluation.

- Initial interval for bracketing methods: $[0, 1]$. This selection is based on endpoint function evaluations: $f(0) \approx 0.1576 > 0$ and $f(1) \approx -1.0005 < 0$. Since $f(0) \cdot f(1) < 0$ and $f(x)$ is continuous over $\mathbb{R}$ (as a composition of continuous elementary functions), the Intermediate Value Theorem guarantees at least one real root in $(0, 1)$. This interval ensures root existence and is suitable for the Bisection and Regula Falsi methods. To verify no other significant roots exist nearby, the exact root search range is expanded to $[-1, 1]$.

Refer to the Table 8 of Python program execution results.

Table 8 reveals a remarkably high level of consistency across all tested numerical methods. All seven methods—Bisection, Regula Falsi, Secant, Fixed Point, Aitken, Steffensen, and Newton—successfully identified the real root of the exponential function with exceptional precision, yielding the value $x \approx 0.363177127637$. Each method recorded a Valid status, confirming that the obtained solutions lie within the interval $[-1, 1]$ and align with the exact root verified through global search.

However, notable differences emerge in the number of iterations required by each method to achieve convergence. This demonstrates that while all methods produce identical final solutions, their computational efficiency varies significantly depending on algorithmic characteristics and computational approaches.

The Steffensen method emerges as the most efficient

Table 8. Performance comparison of Root-Finding methods for solving a mixed equation.

Method	Numeric Root	Closest Root	Iterations	Difference	Status
Bisection	0.36317712767	0.363177127637	31	0.000000000033459	Valid
Regula Falsi	0.36317712751	0.363177127637	44	0.000000000125295	Valid
Secant	0.36317712764	0.363177127637	8	0.000000000000013	Valid
Fixed-Point	0.36317712769	0.363177127637	40	0.000000000053631	Valid
Aitken	0.36317712764	0.363177127637	4	0.000000000000000	Valid
Steffensen	0.36317712764	0.363177127637	4	0.000000000000000	Valid
Newton	0.36317712764	0.363177127637	5	0.000000000000000	Valid

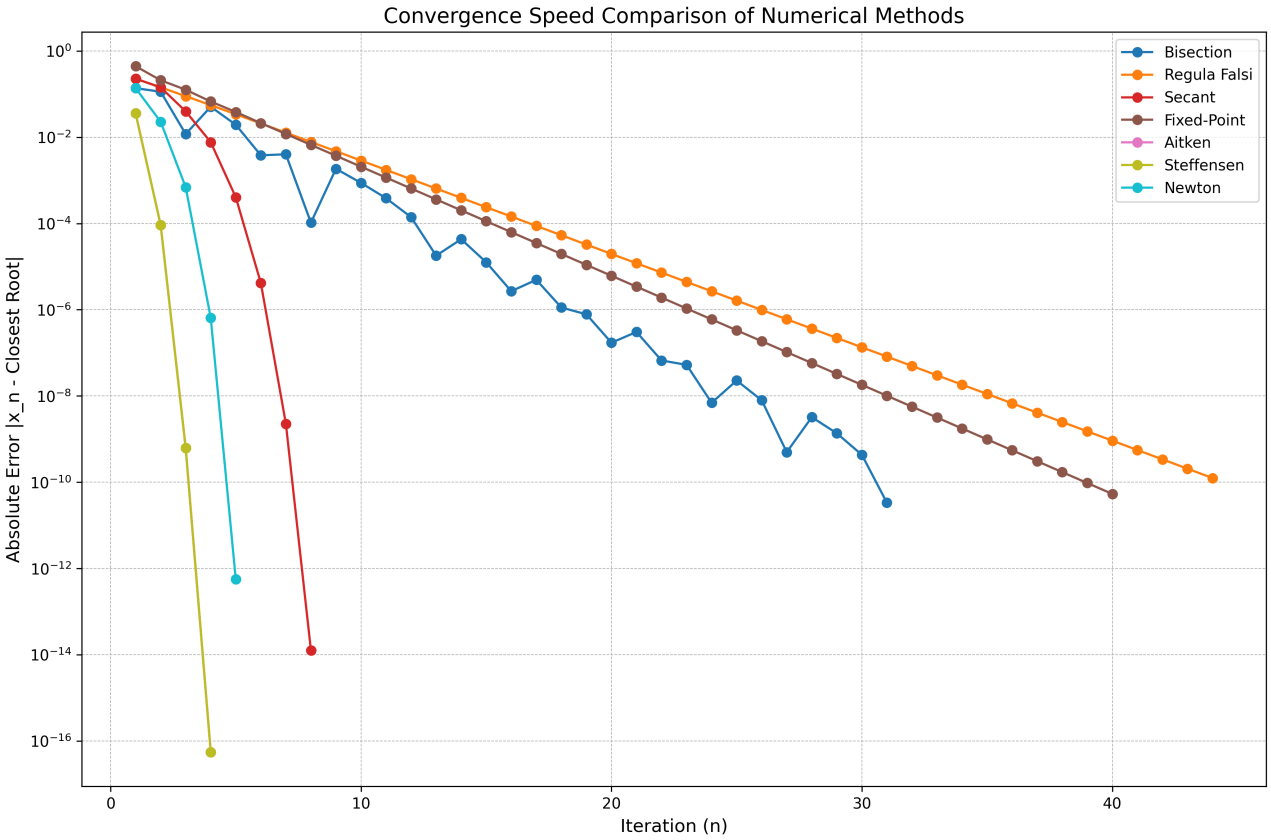


Figure 17. Comparison graph of error reduction speed for mixed function.

in terms of convergence rate, reaching the solution in merely four iterations. Newton's method follows with five iterations, trailed by the Secant method requiring eight iterations. These results are entirely consistent with the established convergence properties of each method: Steffensen exhibits double quadratic convergence, Newton demonstrates quadratic convergence, while the Secant method achieves superlinear convergence. Consequently, their performance in iteration count aligns perfectly with long-standing theoretical principles in numerical analysis.

The convergence velocity comparison illustrated in Figure 17 provides a clear visualization of the iterative behavior of the numerical methods applied to the mixed nonlinear function

$$f(x) = (\cos(e^{x/2}))^3 - x^3,$$

whose real root is located at $x \approx 0.363177127637$. Distinct differences in convergence performance are evident among the tested algorithms.

The Steffensen and Aitken methods exhibit the fastest convergence rates, characterized by an extremely sharp reduction in error during the initial iterations. Both methods reach the maximum precision level of 10^{-15} within only four iterations, which is fully consistent with the tabulated results reporting zero discrepancy from the exact root and valid convergence status. This behavior confirms their double quadratic convergence characteristics, whereby the number of correct digits increases rapidly at each iteration. Notably, the Aitken transformation successfully converts the originally slow Fixed-Point method into a highly efficient scheme, achieving convergence performance comparable to that of Steffensen. This demonstrates that acceleration techniques can substantially enhance simple iterative methods, even when the original iteration function is not optimally designed.

Newton's method also shows excellent performance, with a convergence trajectory closely resembling those of Steffensen and Aitken. Its error decreases sharply and smoothly after the initial iterations, reaching maximum precision within five iterations. The absence of oscillations reflects the expected quadratic convergence behavior. Although Newton's method requires derivative evaluation, its efficiency and stability remain high for this problem, particularly because the initial guess $x_0 = 0.5$ lies sufficiently close to the true root. While slightly slower than Steffensen, Newton's method still demonstrates strong

convergence efficiency.

The Secant method exhibits a characteristic superlinear convergence pattern. Starting from a relatively larger initial error, its error decreases steadily and reaches the prescribed precision in eight iterations. The smooth shape of its convergence curve reflects a convergence order of approximately 1.618, without noticeable oscillatory behavior. These results confirm that, despite being derivative-free, the Secant method remains both efficient and stable for this function, especially when the initial interval $[0, 1]$ is chosen close to the root. From a practical standpoint, the Secant method offers an attractive balance between efficiency and implementation simplicity.

In contrast, the bracketing methods—Bisection and Regula Falsi—display significantly slower convergence behavior. The Bisection method shows a steady, linear reduction in error with an almost constant slope, reaching maximum precision after thirty-one iterations. Regula Falsi initially reduces the error more rapidly but experiences a slowdown in later iterations due to stagnation, where one interval endpoint remains fixed. Despite this limitation, Regula Falsi still converges slightly faster than Bisection, requiring forty-four iterations while maintaining complete numerical stability. Overall, these results highlight the fundamental trade-off between convergence speed and guaranteed stability inherent to bracketing methods.

The most noteworthy observation in Figure 17 is the behavior of the Fixed-Point method (brown). Its iteration curve demonstrates extremely slow error reduction that tends to be monotonic without significant acceleration, even after forty iterations. Upon closer examination, the curve does not exhibit a sharp decline toward zero but rather decreases gradually and steadily. This pattern reflects the fundamental characteristic of linear convergence that typifies fixed-point methods. According to the tabular data, the Fixed-Point method yields a discrepancy of 5.3631×10^{-12}, which remains within the tolerance limits and maintains valid status. This indicates that although the convergence process is slow, the method is still capable of reaching the correct root. This result also reveals a marked contrast compared to previous experiments with exponential functions, where this method failed to find accurate solutions. Therefore, it can be concluded that the transformation $g(x) = \cos(e^{x/2})$, while not optimal, still satisfies the local convergence conditions around the root. This condition enables the iterative system to find the correct solution, albeit at a relatively high computational cost.

The convergence paths of the various numerical methods, as shown in Figure 18, provide a clear visual representation of their iterative behavior in approaching the real root of the mixed function

$$f(x) = \left(\cos(e^{x/2})\right)^3 - x^3,$$

namely $x \approx 0.363177127637$. In general, the Steffensen, Newton, and Aitken methods all reach the exact root $x \approx 0.3632$ within the first few iterations and subsequently remain stable in its vicinity. The Steffensen (yellow) and Newton (light blue) curves exhibit very rapid, nearly monotone convergence without oscillations, reflecting the quadratic convergence that characterizes both methods. These observations are fully consistent with the tabulated results, where all three methods record zero discrepancy from the exact root and are classified as valid. Aitken (light purple), although theoretically an acceleration of the Fixed-Point method, effectively stabilizes the convergence process and attains the same root, illustrating that Aitken's Δ^2 transformation can substantially enhance the performance and stability of its underlying fixed-point iteration for this particular problem.

The Secant method (red) displays a more dynamic convergence pattern. Starting from an initial point around $x = 0.5$, its iterates decrease slightly in the early steps, then gradually approach the root with small fluctuations before eventually stabilizing. This behavior reflects the typical superlinear convergence of the Secant method. Despite not requiring derivative evaluations, it navigates the solution space efficiently and reliably. According to the numerical results table, the Secant method converges in eight iterations with zero difference from the exact root, and the convergence trajectory remains stable and directed, albeit slightly slower than that of the Newton and Steffensen methods.

By contrast, the bracketing methods Bisection (dark blue) and Regula Falsi (orange) exhibit highly stable and monotone convergence. Both start from the initial interval $[0, 1]$ and progressively reduce the bracketing interval until they approach the root, without significant overshoot. The Bisection curve follows a systematic halving pattern, consistent with the theoretical guarantee of convergence for continuous functions that change sign over the initial interval. Regula Falsi, which employs linear interpolation to

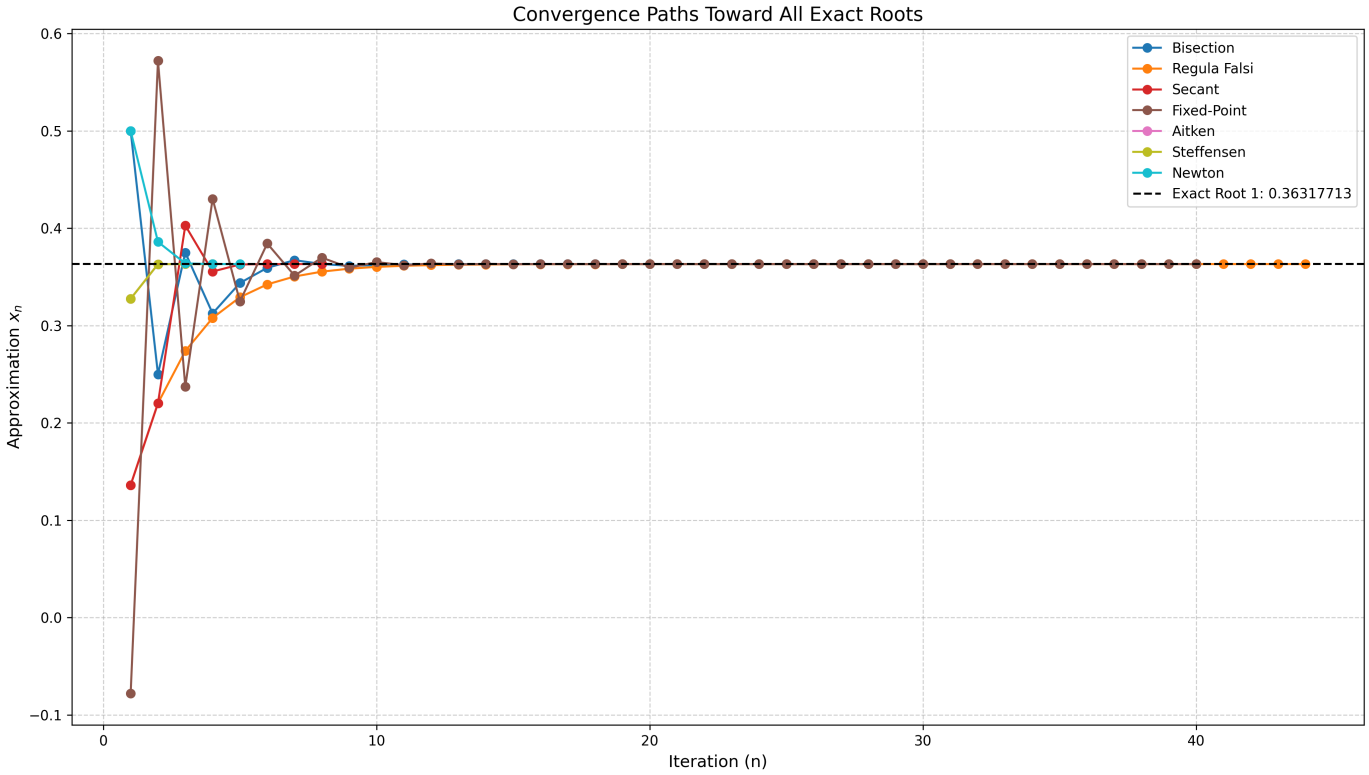


Figure 18. Comparison graph of method approximations to the Exact Root of mixed function.

estimate the root location, also shows robust stability, though its convergence is somewhat slower than that of the Secant method. The tabulated results indicate that Bisection and Regula Falsi require 31 and 44 iterations, respectively, to satisfy the stopping criterion.

The most notable behavior is observed for the Fixed-Point method (brown). Starting from the initial guess $x_0 = 1$, its first iterate drops sharply to approximately $x = -0.08$, then jumps to about $x = 0.57$, and only afterward moves gradually toward the true root. This indicates substantial oscillation in the early stages of the iteration, followed by the eventual formation of a stable convergence path. The numerical results show that the Fixed-Point method requires 40 iterations to converge, with a final discrepancy of 5.3631×10^{-12}, which still lies well within a very tight tolerance. This confirms that, although the convergence trajectory is not smooth, the method ultimately reaches the correct root. The behavior further demonstrates that the transformation

$$g(x) = \cos\left(e^{\frac{x}{2}}\right),$$

while not particularly efficient, does satisfy the local convergence conditions in a neighborhood of the root. As a result, the fixed-point iteration remains capable of finding the exact solution, albeit with a relatively long execution time and pronounced initial oscillations.

The execution time comparison shown in Figure 19 provides an important practical perspective on the performance of the numerical methods, as it reflects not only how quickly a method reaches a solution in terms of iteration count, but also how efficiently it uses computational resources. From the observed results, Steffensen's method emerges as the fastest, requiring only 2.70 ms. This is particularly noteworthy because, although Steffensen's method needs only four iterations, each iteration involves three function evaluations to implement the Aitken acceleration. Even so, its total execution time remains lower than that of Newton's method, which in each iteration requires one function evaluation and one derivative evaluation. This discrepancy suggests that, for a complex mixed function, the cost of derivative evaluation in Newton's method can be relatively high compared with several function evaluations in Steffensen's method. Consequently, although Newton's method is theoretically of quadratic order, its advantage in convergence speed may be partially offset by the additional computational overhead associated with derivative calculations. In terms of the reported data, Newton's method has an execution time of 11.57 ms, which is still relatively fast but slightly slower than Steffensen's method. This observation is consistent with the tabulated results: Newton achieves convergence in five iterations, whereas Steffensen

requires only four. When compared with other methods such as the Secant method (eight iterations) and the Bisection method (thirty-one iterations), Newton still shows strong performance. However, these findings reinforce the fact that theoretical convergence order does not always translate directly into superior execution-time efficiency, since the computational cost per iteration plays an equally crucial role.

The Secant method, as shown in Figure 19, demonstrates excellent convergence performance, requiring only eight iterations and recording an execution time of 10.57 ms. This value is nearly identical to that of Newton's method, indicating comparable temporal efficiency between the two approaches. This result highlights that the Secant method, despite not requiring derivative information, remains highly competitive in terms of computational time. Its efficiency stems from the need for only two function evaluations per iteration combined with a relatively low iteration count. In practical applications, this makes the Secant method a well-balanced choice, as it eliminates derivative computation, preserves superlinear convergence behavior, and achieves efficient execution times.

In contrast, the Fixed-Point method, although successfully locating the correct root, exhibits a significantly higher execution time of 52.71 ms. This behavior can be attributed to its substantially larger iteration count of 40 iterations, where each iteration requires evaluating the iteration function

$$g(x) = \cos\left(e^{\frac{x}{2}}\right),$$

which is computationally more complex than the original function $f(x)$. Furthermore, pronounced oscillations during the early stages of convergence

Table 9. Summary of analysis results for mixed function.

Analysis Category	Best Method	Best Alternative	Remarks
Convergence Speed	Steffensen (4 iterations)	Newton (5 iterations)	The Steffensen method demonstrates the fastest convergence with only four iterations. This speed even surpasses Newton's method, which also possesses quadratic convergence order, indicating exceptionally high iterative efficiency.
Numerical Accuracy	Secant, Steffensen, Newton (error = 0)	Aitken (error = 0)	The three primary methods achieve maximum precision with zero discrepancy from the exact root. Meanwhile, the Aitken method also achieves zero deviation from the exact root, confirming its high precision within the established convergence tolerance limits.
Execution Time	Steffensen (2.70 ms)	Aitken (8.89 ms)	Steffensen proves to be the most time-efficient method, with execution time approximately four times faster than Newton and nearly four times faster than Secant. This demonstrates Steffensen's superiority in computational efficiency despite requiring multiple function evaluations per iteration.
Convergence Stability	Bisection	Regula Falsi	Bracketing methods such as Bisection exhibit exceptionally high convergence stability. Regula Falsi is also stable and slightly faster in reaching the final result, although requiring more iterations. Both methods consistently provide reliable outcomes for the tested mixed function.
Computational Efficiency	Steffensen	Secant	Steffensen offers the optimal balance between execution time, accuracy, and iteration count, making it the most efficient choice for practical implementation. The Secant method serves as an excellent alternative due to its competitive performance in both time and accuracy without requiring function derivatives.

may introduce additional overhead associated with rounding operations or underflow handling by the computing system. Interestingly, despite its higher iteration count, the execution time of the Fixed-Point method remains lower than that of both the Regula Falsi and Bisection methods. This observation suggests that numerical inefficiency in terms of iteration count does not necessarily translate directly into overall computational inefficiency, as a simpler per-iteration computational structure can partially offset slower convergence.

The bracketing methods, Regula Falsi and Bisection, exhibit the longest execution times, recorded at 62.83 ms and 47.19 ms, respectively. These results are consistent with their required iteration counts: Regula Falsi requires 44 iterations to converge, while Bisection requires 31 iterations to reach a stable solution. Although Bisection converges in fewer iterations, its execution time remains comparatively high because each iteration typically involves multiple function evaluations and lacks interpolation mechanisms that could accelerate convergence. Regula Falsi, while benefiting from linear interpolation that improves iteration progress, still incurs considerable computational cost due to the additional calculations required at each step. This comparison underscores the fundamental trade-off inherent in bracketing methods: although they guarantee convergence through systematic interval reduction, they tend to suffer from increased computational expense due to high iteration counts and limited convergence acceleration.

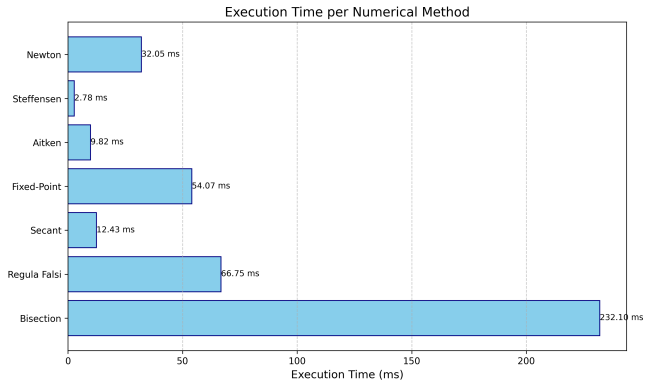


Figure 19. Comparison diagram of execution time for mixed function.

Refer to Table 9, which presents a comprehensive summary of various analytical aspects for the seven numerical methods employed in determining the root of the non-linear equation of the mixed function $f(x) = \cos^3\left(e^{\frac{x}{2}}\right) - x^3$.

Based on several experiments and analysis, the experimental results indicate that no single root-finding method can be claimed to be universally superior; method selection should depend on the characteristics of the function being handled, derivative availability, initial conditions, as well as performance metrics deemed critical (e.g., number of function evaluations, number of derivative evaluations, or execution time). Empirically, in the tested polynomial examples, Steffensen and Newton–Raphson required relatively few iterations (4 and 9 iterations respectively in the measured case), but Secant demonstrated the lowest wall-clock execution time ($\approx$32.4 ms compared to $\approx$110.6 ms for Steffensen and $\approx$129.1 ms for Newton). These findings confirm that theoretical convergence orders (linear, superlinear, quadratic) are important guidelines, but practical measurements depend on per-iteration costs (evaluation of f, evaluation of f', additional operations), function call overhead in the Python environment, and local numerical conditions such as near-zero derivatives or sensitivity to numerical disturbances.

Furthermore, experimental observations explain why the Secant method can exhibit high time efficiency despite having a lower convergence order than Newton or Steffensen. Secant uses finite difference construction, thus avoiding explicit derivative evaluations; from a Python implementation perspective, this reduces heavy operations per iteration and avoids computationally expensive routine calls for derivatives or symbolic-numerical conversions. Consequently, although Secant requires more iterations (e.g., 14 iterations in one case), the relatively low total number of function evaluations and arithmetic operations results in more efficient execution time. Conversely, acceleration methods like Steffensen or extrapolation techniques often reduce iteration counts but increase function evaluations per step or additional numerical operations, so theoretical advantages do not always translate into actual time reduction in an interpreter environment.

On the other hand, there are practical conditions where bracketing-based methods like Bisection and Regula-Falsi are more suitable despite their relatively slow convergence rates. First, when robustness and convergence guarantees are priorities—for example in functions with difficult numerical behavior, poorly defined derivatives around the

root, or when derivatives are unavailable—Bisection guarantees interval reduction and convergence if the signs at the bounds differ. Second, for functions exhibiting oscillation, multiple roots, or unexpected local behavior, interval approaches maintain the desired root within a controlled domain; experiments on trigonometric functions demonstrate how root results from interval methods can differ significantly compared to fixed-point methods dependent on transformation formulation. Third, if function evaluations are computationally expensive, Regula-Falsi variants equipped with conservative correction factors can offer a compromise that reduces function calls compared to repeated derivative calculations or acceleration transformations. Therefore, hybrid strategies—starting with bracketing to obtain coarse approximations and guarantees, then switching to high-speed open methods for refinement—often represent a balanced practical approach combining reliability and efficiency.

5.5 The Gap Between Theoretical Convergence Speed and Actual Computational Performance

Detailed analysis reveals a consistent gap between the theoretically predicted asymptotic properties (convergence orders) and the measured computational performance in Python implementations. The main sources of this discrepancy include: (i) variations in per-iteration complexity among methods—e.g., Steffensen typically requires two function evaluations per iteration, while Newton requires function and derivative evaluations; (ii) the actual cost of derivative evaluation in Python sometimes exceeds theoretical assumptions, particularly when derivatives are computed numerically or through symbolic→numeric conversion; (iii) high-level language overhead such as Python function calls, temporary object allocation, and conversion between symbolic and numerical array representations; and (iv) local numerical properties that cause practical behavior (e.g., the need for step control or correction) thereby increasing iteration counts and additional operations.

Empirical support from our experiments shows a strong correlation between the total number of function calls and wall-clock time. When iterations are converted to estimated total function evaluation counts, in our measured case Steffensen $\approx 2 \times 4 = 8$ total function evaluations, Newton $\approx 2 \times 9 = 18$ evaluations (assuming equal cost for evaluating f and f'), and Secant $\approx 1 \times 14 = 14$ new function evaluations; this pattern aligns with actual time measurements where methods with lower per-iteration overhead tend to be faster even though they require more iterations. Additionally, acceleration techniques that significantly reduce iterations recorded high execution times for some functions (e.g., range $\approx$ 498–518 ms in certain samples) due to the additional costs of complex transformations and function evaluations. For functions with numerical behavior vulnerable to disturbances (e.g., very small derivatives near the root), derivative-based methods may require stabilization mechanisms (line search, step restriction) so their practical performance decreases relative to theoretical predictions; conversely, bracketing methods maintain stability despite their slower convergence rates.

Based on these findings, practical recommendations for implementations in Python environments are: conduct simple profiling to calculate per-call costs of functions and derivatives in the target environment; consider hybrid strategies (bracketing → open methods) or algorithm modifications (e.g., Regula-Falsi with conservative correction) to synergize reliability and efficiency; and if derivative-based methods are selected, consider implementation optimizations (vectorization, precomputation, or JIT compilation usage) to reduce interpreter overhead. Thus, although convergence theory remains an analytical foundation, practical decisions must incorporate implementation cost analysis and concrete numerical characteristics to maximize actual computational performance.

6 Conclusion

This research demonstrates that no single algorithm is absolutely superior in solving nonlinear equation root-finding. Through comprehensive evaluation of seven numerical methods, it was revealed that each method's performance heavily depends on function characteristics and computational context. Steffensen and Newton–Raphson consistently achieved the fastest convergence in terms of iteration count, with Steffensen's advantage being its ability to obtain quadratic convergence without requiring explicit derivatives. However, a key finding shows that fewer iterations do not always correlate directly with actual computational efficiency, and Secant often emerges as the most efficient choice in terms of execution time due to its combination of superlinear convergence and low per-iteration cost. This pattern confirms the complex relationship between convergence theory and practical implementation in modern computing

environments, where function evaluation efficiency becomes a determining performance factor.

The practical implications of this study highlight the need for a contextual and applied approach to method selection. For applications demanding speed and accuracy on well-behaved functions, Steffensen or Secant are recommended. Conversely, bracketing methods (bisection and Regula-Falsi) remain relevant when stability and convergence guarantees are prioritized, despite their slower convergence rates. Newton–Raphson is useful when derivatives are easily obtained and initial guesses are adequate. Additionally, Fixed-Point Iteration proved sensitive to the choice of iteration function, thus requiring very careful formulation, while Aitken Δ^2 demonstrated potential as an effective accelerator. Thus, this research provides an empirical performance map and practical framework for selecting numerical algorithms in scientific and engineering applications.

Data Availability Statement

The Python code implementations and supporting materials used in this study are openly available in a public GitHub repository at https://github.com/nafishr24/numerical_method (accessed 13 December 2025).

Funding

This work was supported without any funding.

Conflicts of Interest

The author declares no conflicts of interest.

Ethical Approval and Consent to Participate

Not applicable.

References

[1] Chaudhary, R., Rawat, S., & Chauhan, P. (2024). Unraveling the roots: A comprehensive review of numerical methods for root finding. *Journal of Emerging Technologies and Innovative Research*, *11*(1). https://www.jetir.org/view?paper=JETIR2401176

[2] Qureshi, S., Argyros, I. K., Soomro, A., Gdawiec, K., Shaikh, A. A., & Hincal, E. (2024). A new optimal root-finding iterative algorithm: Local and semilocal analysis with polynomiography. *Numerical Algorithms*, *95*(4), 1715–1745. [Crossref]

[3] Kelley, C. T. (2018). Numerical methods for nonlinear equations. *Acta Numerica*, *27*, 207–287. [Crossref]

[4] Gezerlis, A. (2023). *Numerical methods in physics with Python* (Vol. 1). Cambridge, UK: Cambridge University Press. [Crossref]

[5] Zaguskin, V. L. (2014). *Handbook of numerical methods for the solution of algebraic and transcendental equations*. Elsevier.

[6] Cordero, A., Hueso, J. L., Martínez, E., & Torregrosa, J. R. (2012). Steffensen type methods for solving nonlinear equations. *Journal of Computational and Applied Mathematics*, *236*(12), 3058–3064. [Crossref]

[7] Saad, Y. (2025). Acceleration methods for fixed-point iterations. *Acta Numerica*, *34*, 805-890. [Crossref]

[8] Van Der Walt, S., Colbert, S. C., & Varoquaux, G. (2011). The NumPy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, *13*(2), 22–30. [Crossref]

[9] Azure, I., Aloliga, G., & Doabil, L. (2019). Comparative study of numerical methods for solving non-linear equations using manual computation. *Mathematics Letters*, *5*(4), 41-46. [Crossref]

[10] Thakur, G., & Saini, J. K. (2021). Comparative study of iterative methods for solving non-linear equations. *Journal of University of Shanghai for Science and Technology*, *23*(7), 858–866. [Crossref]

[11] Ahmad, I., Ullah, M. A., & Uddin, J. (2024). Analysis and Enhancement of Newton Raphson Method. *The Sciencetech*, *5*(2), 13-23. https://journals.qurtuba.edu.pk/ojs/index.php/tst/article/view/8601

[12] Dixit, N. D., & Mathur, P. K. (2021). Comparision of Numerical Accuracy of Bisection, Newton Raphson, Falsi-Position and Secant Methods. *Advances in Mathematics: Scientific Journal*, *10*. https://research-publication.com/amsj/uploads/papers/vol-10/iss-12/AMSJ-2021-N12-13.pdf

[13] Thota, S., & Srivastav, V. K. (2018). Quadratically convergent algorithm for computing real root of non-linear transcendental equations. *BMC Research Notes*, *11*(1), 909. [Crossref]

[14] Wang, S. (2025). A Comparative Study of Five Root-Finding Methods for Applying the Black-Scholes Model in Real Markets. *Available at SSRN 5381527*. [Crossref]

[15] Argyros, I. K., Argyros, C., Ceballos, J., & González, D. (2022). Extended comparative study between Newton's and Steffensen-like methods with applications. *Mathematics*, *10*(16), 2851. [Crossref]

[16] Chen, C., & Liao, X. (2025). Solving non-linear equations by fixed point iteration method and its accelerating approach. *Thermal Science*, *29*(3 Part A), 2031–2039. [Crossref]

[17] Meurer, A., Smith, C. P., Paprocki, M., Čertík, O., Kirpichev, S. B., Rocklin, M., ... & Scopatz, A. (2017). SymPy: symbolic computing in Python. *PeerJ Computer Science*, *3*, e103. [Crossref]

[18] Oliphant, T. E. (2007). Python for scientific computing.

Computing in Science & Engineering, *9*(3), 10–20. [Crossref]

[19] Ali, A., & Khusro, S. (2024). SA-MEAS: Sympy-based automated mathematical equations analysis and solver. *SoftwareX*, *25*, 101596. [Crossref]

[20] Harris, C. R., Millman, K. J., Van Der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... & Oliphant, T. E. (2020). Array programming with NumPy. *nature*, *585*(7825), 357-362. [Crossref]

[21] Gupta, P., & Bagchi, A. (2024). Introduction to NumPy. In *Essentials of Python for Artificial Intelligence and Machine Learning* (pp. 127-159). Cham: Springer Nature Switzerland. [Crossref]

[22] Unpingco, J. (2021). Numpy. In *Python programming for data analysis* (pp. 103-126). Cham: Springer International Publishing. [Crossref]

[23] Fuhrer, C., Solem, J. E., & Verdier, O. (2021). *Scientific Computing with Python: High-performance scientific computing with NumPy, SciPy, and pandas*. Packt Publishing Ltd. https://www.packtpub.com/en-us/product/scientific-computing-with-python-9781838822323

[24] Gupta, P., & Bagchi, A. (2024). Introduction to pandas. In *Essentials of python for artificial intelligence and machine learning* (pp. 161-196). Cham: Springer Nature Switzerland. [Crossref]

[25] Roy, S. S., Chouhan, A., & Khera, N. (2025). *Python Fast Track: A Complete Guide to Rapidly Mastering and Applying Python Programming*. Elsevier. [Crossref]

[26] Pérez, F., Granger, B. E., & Hunter, J. D. (2011). Python: An ecosystem for scientific computing. *Computing in Science & Engineering*, *13*(2), 13–21. [Crossref]

[27] Lavanya, A., Gaurav, L., Sindhuja, S., Seam, H., Joydeep, M., Uppalapati, V., ... & SD, V. S. (2023). Assessing the performance of Python data visualization libraries: a review. *Int. J. Comput. Eng. Res. Trends*, *10*(1), 28-39. [Crossref]

[28] Haslwanter, T. (2016). An introduction to statistics with python. *With applications in the life sciences. Switzerland: Springer International Publishing*.

[29] Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, *9*(3), 90–95. [Crossref]

[30] Gan, G. (2025). The Matplotlib Library. In *Data Clustering with Python* (pp. 66–74). Chapman and Hall/CRC. [Crossref]

[31] Gulshan, G., Budak, H., Hussain, R., & Sadiq, A. (2023). Generalization of the bisection method and its applications in nonlinear equations. *Advances in Continuous and Discrete Models*, *2023*(1), 18. [Crossref]

[32] Mueen, H. A., & Shiker, M. A. K. (2024). Comparison Newton method with bisection method for solving nonlinear equations. In *2024 8th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)* (pp. 1–4). IEEE. [Crossref]

[33] Bhardwaj, R. (2025). Numerical Simulation of Nonlinear Equations by Modified Bisection and Regula Falsi Method. *Proceedings of the Pakistan Academy of Sciences: A. Physical and Computational Sciences*, *62*(1), 11-19. [Crossref]

[34] Olanrewaju, A. F., Fadugba, S. E., & Adeyemi, O. P. (2024). Comparative study of numerical solutions to non-linear equations using regula-falsi and newton-raphson method. In *2024 International Conference on Science, Engineering and Business for Driving Sustainable Development Goals (SEB4SDG)* (pp. 1–6). IEEE. [Crossref]

[35] Hanief, M. (2021). Combined regula–falsi and newton–raphson method. In *Recent Advances in Mathematical Methods and Economic Theory* (pp. 123–132). Springer. [Crossref]

[36] Fernández-Díaz, J. M., & Menéndez-Pérez, C. O. (2023). A common framework for modified regula falsi methods and new methods of this kind. *Mathematics and Computers in Simulation*, *205*, 678–696. [Crossref]

[37] Kanwar, V., Sharma, P., Argyros, I. K., Behl, R., Argyros, C., Ahmadian, A., & Salimi, M. (2021). Geometrically constructed family of the simple fixed point iteration method. *Mathematics*, *9*(6), 694. [Crossref]

[38] Sana, G., Noor, M. A., Hassan, M. U., Hammouch, Z., & Hafstein, S. F. (2022). Some recent modifications of fixed point iterative schemes for computing zeros of nonlinear equations. *Complexity*, 2022, 4331899. [Crossref]

[39] Brezinski, C., & Redivo-Zaglia, M. (2019). The genesis and early developments of Aitken's process, Shanks' transformation, the ε-algorithm, and related fixed point methods. *Numerical Algorithms*, *80*(1), 11–133. [Crossref]

[40] Brezinski, C. (2000). Convergence acceleration during the 20th century. *Journal of Computational and Applied Mathematics*, *122*(1–2), 1–21. [Crossref]

[41] Pomeranz, S. B. (2017). Aitken's Δ^2 method extended. *Cogent Mathematics*, 4(1), 1308622. [Crossref]

[42] Amat, S., Ezquerro, J. A., & Hernández-Verón, M. A. (2016). On a Steffensen-like method for solving nonlinear equations. *Calcolo*, *53*(2), 171–188. [Crossref]



Moh. Nafis Husen Romadani is a graduate of the Mathematics Education Study Program at Madura University, class of 2023. Following the completion of his undergraduate degree, he was admitted to the Pre-service Teacher Professional Education Program (PPG) at Widya Mandala Surabaya Catholic University (UKWMS) in 2024. He is currently engaged as a freelance Mathematics tutor and is concurrently enhancing his competencies in the field of programming. (Email: nafishusenromadani@gmail.com)