Four New Hybrid Root Bracketing Algorithms: Optimization-Based Approaches for Numerical Root Finding

Abdelrahman Ellithy[1]
Department of Scientific Computing, Faculty of Computers and Artificial Intelligence, Benha University, Benha, Egypt

(Dated: June 25, 2025)

The accurate and efficient determination of roots for nonlinear equations remains a significant challenge in numerical analysis, particularly when dealing with functions that exhibit complex behavior or require robust convergence guarantees. Traditional bracketing methods, such as bisection and false position, offer reliability but often suffer from slow convergence, while optimization-based methods like the secant method provide speed at the cost of stability. In this paper presents four novel hybrid root bracketing algorithms that combine the advantages of traditional bracketing methods with optimization techniques. The proposed algorithms—Optimized Bisection-False Position (Optimized_BF), Optimized Bisection-False Position with Modified Secant (Optimized_BFMS), Optimized Trisection-False Position (Optimized_TF), and Optimized Trisection-False Position with Modified Secant (Optimized_TFMS)—demonstrate improved convergence rates and computational efficiency compared to their traditional counterparts. Through extensive experimentation on 14 diverse test functions using the SymPy library, our algorithms show consistent performance improvements, with Optimized_BFMS achieving the best overall performance. The hybrid approach leverages the reliability of bracketing methods while incorporating the speed advantages of optimization-based techniques, making these algorithms suitable for a wide range of numerical computing applications.

I. Introduction

Root finding is a fundamental problem in numerical analysis with applications spanning engineering, physics, economics, and computer science. Traditional root bracketing methods such as bisection, false position, and trisection provide guaranteed convergence but often suffer from slow convergence rates, especially for functions with complex behavior near the root. The motivation for developing hybrid approaches stems from the need to combine the reliability of bracketing methods with the efficiency of optimization techniques. Recent research has shown that hybrid algorithms can significantly improve convergence rates while maintaining the robustness of traditional methods [1, 3].

Recent advances in root-finding algorithms have focused on improving the convergence and robustness of classical bracketing methods by hybridizing them with optimization-based techniques. Notable contributions include blended and hybrid algorithms that combine bisection, regula falsi, and secant methods [2, 6], as well as metaheuristic approaches for nonlinear equations [7, 8]. These works have demonstrated that hybridization can yield significant improvements in both speed and reliability, motivating the development of the new algorithms presented in this paper.

The main contributions of this paper are as follows. First, we present four novel hybrid algorithms, each with a solid mathematical foundation, that blend the best aspects of existing root-finding techniques. Second, we provide a comprehensive performance analysis of these algorithms on a diverse set of test functions, demonstrating their effectiveness and generality. Third, we detail the implementation of these methods using the SymPy library for symbolic computation, which allows for high-precision and flexible experimentation.

Finally, we offer a comparative analysis with traditional methods, highlighting the advantages and potential of the proposed hybrid approaches in practical numerical computation scenarios.

This paper is organized as follows. Section II (page 2) details the methodology and mathematical foundations of the proposed algorithms. Section III (page 8) describes the experimental setup and test functions. Section IV (page 12) presents the results and analysis. Section V (page 18) discusses the implications and limitations. Section VI (page 20) concludes the paper and outlines future research directions.

II. Methodology

The methodology adopted in this study is centered on the design and development of four hybrid root bracketing algorithms. Each algorithm is constructed by integrating the strengths of traditional bracketing methods with optimization-based enhancements. For example, the Optimized Bisection-False Position (Optimized_BF) algorithm leverages the interval-halving reliability of bisection and the faster convergence of the false position method. The Optimized_BFMS algorithm incorporates a modified secant step to further accelerate convergence. The trisection-based variants, Optimized_TF and Optimized_TFMS, use a three-way interval split and integrate both false position and secant-based acceleration. All algorithms are implemented using the SymPy library, which provides symbolic computation capabilities and high-precision arithmetic. The mathematical foundations of these methods are rooted in the Intermediate Value Theorem, linear interpolation, and finite-difference approximations for derivative estimation.

The convergence analysis of these hybrid methods follows established theoretical frameworks. For bracketing methods, the convergence is guaranteed by the Intermediate Value Theorem, while the rate of convergence depends on the specific combination of techniques employed. The false position method exhibits superlinear convergence in most cases, while the modified secant step can provide additional acceleration through derivative approximation.

A Mathematical Foundations

The hybrid algorithms are based on several key mathematical principles. The Intermediate Value Theorem guarantees the existence of a root in the interval $[a, b]$ if $f(a) \times f(b) < 0$, providing the theoretical foundation for bracketing methods [9]. The False Position Method utilizes linear interpolation to estimate the root location, offering a more efficient approach than simple interval halving [10]. The Modified Secant Method accelerates convergence by approximating the derivative using finite differences, which allows for faster root refinement without requiring explicit derivative calculations [11]. Finally, the overall optimization strategy of the hybrid algorithms is to combine these multiple approaches, thereby minimizing the number of function evaluations while maintaining robust convergence properties.

The theoretical convergence analysis follows the framework established by [1] and [3], who demonstrated that hybrid approaches can achieve superlinear convergence rates while maintaining the reliability of bracketing methods. The trisection method, as analyzed by [12], provides a theoretical foundation for the three-way interval splitting approach used in our trisection-based algorithms.

For the false position method, the convergence rate is typically superlinear, with the error reduction following the pattern $|x_{n+1} - \alpha| \leq C \cdot |x_n - \alpha|^p$, where $p > 1$ and C is a constant depending on the function properties. The modified secant method, using finite difference approximation, maintains similar convergence characteristics while avoiding the need for explicit derivative calculations.

B Algorithm Design, Hybrid Techniques and Pseudocode for Proposed Algorithms

Root-finding for nonlinear equations is a fundamental problem in numerical analysis, with wide-ranging applications in science and engineering. Classical bracketing methods such as bisection and false position are robust but can be slow, while open methods like secant and Newton's method are faster but may lack guaranteed convergence. Hybrid algorithms seek to combine the strengths of these approaches, achieving both reliability and efficiency. In this work, we propose four novel hybrid root-bracketing algorithms, each designed to address specific limitations of existing methods and to optimize convergence speed while preserving the bracketing property. Below, we present the design rationale, mathematical formulation, and pseudocode for each algorithm.

1 Algorithm 1: Optimized Bisection-False Position (Optimized_BF)

The Optimized Bisection-False Position (Optimized_BF) algorithm is a two-phase hybrid method that strategically combines the guaranteed convergence of the bisection method with the accelerated convergence potential of the false position (regula falsi) method. The motivation for this hybridization is to overcome the slow, linear convergence of bisection while retaining its reliability, by introducing the interpolation-based acceleration of the false position step.

The algorithm operates as follows: In each iteration, the bisection phase computes the midpoint $mid = \frac{a+b}{2}$ and evaluates $f(mid)$, ensuring the root remains bracketed. The subsequent false position phase leverages linear interpolation between the interval endpoints to compute a potentially more accurate estimate using:

$$fp = \frac{a \cdot f(b) - b \cdot f(a)}{f(b) - f(a)} \tag{1}$$

The algorithm then selects the estimate (midpoint or false position) with the smaller absolute function value, updating the interval accordingly to maintain the bracketing property. This approach ensures that even if the false position step is less effective for certain function shapes, the bisection step provides a fallback, guaranteeing progress toward the root.

The key innovation is the dynamic selection between midpoint and false position estimates, which adapts to the local function behavior. This results in faster convergence than pure bisection, especially for functions with significant curvature, while maintaining the reliability guarantees of bracketing methods. The method signature is Optimized_BF(f, a, b, tol, max_iter), returning $(n, x, f_x, a_{final}, b_{final})$ as the iteration count, root approximation, function value, and final interval bounds. Robust error handling is included for division by zero in the false position step.

Algorithm 1 Optimized Bisection-False Position (Optimized_BF)

Input: Function f, interval $[a, b]$, tolerance tol, max iterations max_iter
Output: Root approximation x, function value $f(x)$, final interval $[a, b]$
$n \leftarrow 0$
while $n < max\_iter$ **do**
    $n \leftarrow n + 1$
    $f_a \leftarrow f(a)$, $f_b \leftarrow f(b)$
    $x_B \leftarrow (a + b)/2$, $f_{x_B} \leftarrow f(x_B)$
    $x_F \leftarrow (af_b - bf_a)/(f_b - f_a)$, $f_{x_F} \leftarrow f(x_F)$
    if $|f_{x_B}| < |f_{x_F}|$ **then**
        $x \leftarrow x_B$, $fx \leftarrow f_{x_B}$
    else
        $x \leftarrow x_F$, $fx \leftarrow f_{x_F}$
    end if
    if $|fx| \leq tol$ **then**
        return n, x, fx, a, b
    end if
    if $f_a \times fx < 0$ **then**
        $b \leftarrow x$
    else
        $a \leftarrow x$
    end if
end while
return max_iter, x, fx, a, b

2 Algorithm 2: Optimized Bisection-False Position with Modified Secant (Optimized_BFMS)

The Optimized Bisection-False Position with Modified Secant (Optimized_BFMS) algorithm extends the Optimized_BF framework by incorporating a third phase: a derivative-free modified secant step. The motivation is to further accelerate convergence, especially near the root, by leveraging local slope information without requiring explicit derivatives.

After the bisection and false position phases, the algorithm computes a secant-based estimate using a finite difference approximation:

$$f'(x) \approx \frac{f(x + \delta) - f(x)}{\delta} \tag{2}$$

$$x_S = x - \frac{\delta \cdot f(x)}{f(x + \delta) - f(x)} \tag{3}$$

where $\delta = 10^{-4}$ is chosen for numerical stability. The secant estimate x_S is only accepted if it lies within the current interval and provides a better approximation (smaller $|f(x_S)|$) than the previous step. This conservative acceptance criterion ensures the bracketing property is preserved and prevents divergence due to poor secant approximations.

The three-phase structure—bisection, false position, and modified secant—enables the algorithm to adaptively exploit the most effective strategy at each stage, resulting in superior convergence rates while maintaining reliability. The method signature is Optimized_BFMS($f, a, b, tol, max\_iter, \delta = 10^{-4}$), returning

$(n, x, f_x, a_{final}, b_{final})$. The choice of δ is based on empirical testing for optimal balance between accuracy and stability.

Algorithm 2 Optimized Bisection-False Position with Modified Secant (Optimized_BFMS)

Input: Function f, interval $[a, b]$, tolerance tol, max iterations max_iter, $\delta = 10^{-4}$
Output: Root approximation x, function value $f(x)$, final interval $[a, b]$
$n \leftarrow 0$
while $n < max\_iter$ **do**
    $n \leftarrow n + 1$
    $f_a \leftarrow f(a)$, $f_b \leftarrow f(b)$
    $x_B \leftarrow (a + b)/2$, $f_{x_B} \leftarrow f(x_B)$
    $x_F \leftarrow (af_b - bf_a)/(f_b - f_a)$, $f_{x_F} \leftarrow f(x_F)$
    if $|f_{x_B}| < |f_{x_F}|$ **then**
        $x \leftarrow x_B$, $fx \leftarrow f_{x_B}$
    else
        $x \leftarrow x_F$, $fx \leftarrow f_{x_F}$
    end if
    if $|fx| \leq tol$ **then**
        return n, x, fx, a, b
    end if
    if $f_a \times fx < 0$ **then**
        $b \leftarrow x$
    else
        $a \leftarrow x$
    end if
    $f_x \leftarrow f(x)$
    $f_{x+\delta} \leftarrow f(x + \delta)$
    $x_S \leftarrow x - \delta \cdot f_x/(f_{x+\delta} - f_x)$
    if $a < x_S < b$ **then**
        $f_{x_S} \leftarrow f(x_S)$
        if $|f_{x_S}| < |fx|$ **then**
            if $f_a \times f_{x_S} < 0$ **then**
                $b \leftarrow x_S$
            else
                $a \leftarrow x_S$
            end if
            if $|f_{x_S}| \leq tol$ **then**
                return n, x_S, f_{x_S}, a, b
            end if
        end if
    end if
end while
return max_iter, x, fx, a, b

3 Algorithm 3: Optimized Trisection-False Position (Optimized_TF)

The Optimized Trisection-False Position (Optimized_TF) algorithm introduces a more aggressive interval reduction strategy by employing trisection instead of bisection, motivated by the desire to accelerate convergence for functions where the root is not near the endpoints. Trisection divides the interval $[a, b]$ into three equal parts, computing two interior points:

$$x_1 = a + \frac{b-a}{3} \quad \text{and} \quad x_2 = b - \frac{b-a}{3} \tag{4}$$

This approach can reduce the interval size by up to a factor of 3 per iteration, compared to 2 for bisection, as analyzed by [12].

In each iteration, the algorithm first uses trisection to identify the subinterval containing the root by evaluating $f(x_1)$ and $f(x_2)$ and checking sign changes. It then applies the false position method within the identified subinterval, using linear interpolation to further accelerate convergence. The algorithm selects the best candidate (from trisection or false position) based on the smallest absolute function value, and updates the interval to maintain the bracketing property.

The key advantage is the potential for much faster interval reduction, especially for functions with complex or oscillatory behavior. The method signature is Optimized_TF(f, a, b, tol, max_iter), returning $(n, x, f_x, a_{final}, b_{final})$. Robust error handling is included for division by zero in the false position step, and the bracketing property is strictly maintained throughout.

Algorithm 3 Optimized Trisection-False Position (Optimized_TF)

Input: Function f, interval $[a, b]$, tolerance tol, max iterations max_iter
Output: Root approximation x, function value $f(x)$, final interval $[a, b]$
$n \leftarrow 0$
while $n < max\_iter$ **do**
    $n \leftarrow n + 1$
    $f_a \leftarrow f(a)$, $f_b \leftarrow f(b)$
    $x_1 \leftarrow (2a + b)/3$, $x_2 \leftarrow (a + 2b)/3$
    $f_{x_1} \leftarrow f(x_1)$, $f_{x_2} \leftarrow f(x_2)$
    if $|f_{x_1}| < |f_{x_2}|$ **then**
        $x_T \leftarrow x_1$, $fx_T \leftarrow f_{x_1}$
    else
        $x_T \leftarrow x_2$, $fx_T \leftarrow f_{x_2}$
    end if
    $x_F \leftarrow (af_b - bf_a)/(f_b - f_a)$, $f_{x_F} \leftarrow f(x_F)$
    if $|fx_T| < |f_{x_F}|$ **then**
        $x \leftarrow x_T$, $fx \leftarrow fx_T$
    else
        $x \leftarrow x_F$, $fx \leftarrow f_{x_F}$
    end if
    if $|fx| \leq tol$ **then**
        return n, x, fx, a, b
    end if
    if $f_a \times fx < 0$ **then**
        $b \leftarrow x$
    else
        $a \leftarrow x$
    end if
end while
return max_iter, x, fx, a, b

4 Algorithm 4: Optimized Trisection-False Position with Modified Secant (Optimized_TFMS)

The Optimized Trisection-False Position with Modified Secant (Optimized_TFMS) algorithm represents the most advanced of the four proposed methods, combining trisection, false position, and a modified secant step in a hierarchical three-phase strategy. The motivation is to achieve the fastest possible convergence while

rigorously maintaining the bracketing property and numerical stability.

The algorithm begins with trisection to rapidly reduce the interval, followed by the false position method for interpolation-based acceleration. It then introduces a modified secant step, using finite difference approximation:

$$x_S = fp - \frac{\delta \cdot f(fp)}{f(fp + \delta) - f(fp)} \tag{5}$$

where fp is the false position estimate and $\delta = 10^{-4}$. The secant estimate x_S is only accepted if it (1) lies within the current interval, (2) provides a better approximation than the false position step, and (3) is numerically stable. This multi-level validation ensures that the algorithm never sacrifices reliability for speed.

The three-phase approach—trisection, false position, and modified secant—enables the algorithm to adaptively exploit the most effective strategy at each stage, resulting in the lowest average iteration count and CPU time among all proposed algorithms. The method signature is Optimized_TFMS($f, a, b, tol, max\_iter, \delta = 10^{-4}$), returning $(n, x, f_x, a_{final}, b_{final})$. Experimental results confirm the superior performance of this approach across a diverse test suite.

Algorithm 4 Optimized Trisection-False Position with Modified Secant (Optimized_TFMS)

Input: Function f, interval $[a, b]$, tolerance tol, max iterations max_iter, $\delta = 10^{-4}$
Output: Root approximation x, function value $f(x)$, final interval $[a, b]$
$n \leftarrow 0$
while $n < max\_iter$ **do**
    $n \leftarrow n + 1$
    $f_a \leftarrow f(a)$, $f_b \leftarrow f(b)$
    $x_1 \leftarrow (2a + b)/3$, $x_2 \leftarrow (a + 2b)/3$
    $f_{x_1} \leftarrow f(x_1)$, $f_{x_2} \leftarrow f(x_2)$
    if $|f_{x_1}| < |f_{x_2}|$ **then**
        $x_T \leftarrow x_1$, $fx_T \leftarrow f_{x_1}$
    else
        $x_T \leftarrow x_2$, $fx_T \leftarrow f_{x_2}$
    end if
    $x_F \leftarrow (af_b - bf_a)/(f_b - f_a)$, $f_{x_F} \leftarrow f(x_F)$
    if $|fx_T| < |f_{x_F}|$ **then**
        $x \leftarrow x_T$, $fx \leftarrow fx_T$
    else
        $x \leftarrow x_F$, $fx \leftarrow f_{x_F}$
    end if
    if $|fx| \leq tol$ **then**
        return n, x, fx, a, b
    end if
    if $f_a \times fx < 0$ **then**
        $b \leftarrow x$
    else
        $a \leftarrow x$
    end if
    $f_x \leftarrow f(x)$
    $f_{x+\delta} \leftarrow f(x + \delta)$
    $x_S \leftarrow x - \delta \cdot f_x/(f_{x+\delta} - f_x)$
    if $a < x_S < b$ **then**
        $f_{x_S} \leftarrow f(x_S)$
        if $|f_{x_S}| < |fx|$ **then**
            if $f_a \times f_{x_S} < 0$ **then**
                $b \leftarrow x_S$
            else
                $a \leftarrow x_S$
            end if
            if $|f_{x_S}| \leq tol$ **then**
                return n, x_S, f_{x_S}, a, b
            end if
        end if
    end if
end while
return max_iter, x, fx, a, b

III. Experimental Setup

The experimental framework was designed to provide a comprehensive evaluation of the proposed hybrid algorithms across a diverse range of mathematical functions. The study employed a rigorous methodology that ensures fair comparison, statistical significance, and practical relevance. The experimental setup encompasses four key components: a comprehensive test suite of mathematical functions, a robust implementation framework using SymPy, precise performance measurement protocols, and a systematic data collection and

analysis system.

A Mathematical Equations Dataset with Many Categories

The algorithms were evaluated on a carefully curated set of 14 diverse functions that represent the major categories encountered in practical numerical analysis applications. This comprehensive test suite was designed to challenge the algorithms across different mathematical behaviors and convergence characteristics.

The test functions are categorized as follows:

Transcendental Functions:

- $f_1(x) = x \cdot e^x - 7$: A transcendental function combining exponential and polynomial terms
- $f_8(x) = e^x - 3x - 2$: A mixed exponential-linear function

Polynomial Functions:

- $f_2(x) = x^3 - x - 1$: A cubic polynomial with multiple roots
- $f_{12}(x) = x^{10} - 1$: A high-degree polynomial function

Quadratic Functions:

- $f_3(x) = x^2 - x - 2$: A standard quadratic equation
- $f_5(x) = x^2 - 10$: A simple quadratic with irrational roots
- $f_{13}(x) = x^2 - x - 2$: A repeated quadratic for consistency testing
- $f_{14}(x) = x^2 + 2x - 7$: A quadratic with different coefficients

Trigonometric Functions:

- $f_4(x) = x - \cos(x)$: A mixed linear-trigonometric function
- $f_6(x) = \sin(x) - x^2$: A trigonometric-quadratic combination
- $f_{10}(x) = x \cdot \sin(x) - 1$: A product of linear and trigonometric terms
- $f_{11}(x) = x \cdot \cos(x) + 1$: A product of linear and trigonometric terms

Logarithmic and Mixed Functions:

- $f_7(x) = x + \ln(x)$: A mixed linear-logarithmic function
- $f_9(x) = x^2 + e^{x/2} - 5$: A complex mixed function combining quadratic and exponential terms

Each function was evaluated over carefully selected intervals that bracket the root, with initial intervals chosen to ensure the Intermediate Value Theorem conditions are satisfied. The dataset represents a comprehensive test suite covering the major categories of functions encountered in practical numerical analysis applications, providing a robust foundation for algorithm evaluation.

B SymPy Implementation Details

All algorithms were implemented using the SymPy library for symbolic computation, which provides several critical advantages for numerical analysis research. The implementation leverages several key features of the library:

Symbolic Function Definition: Functions are defined using SymPy symbols, enabling precise mathematical representation and automatic differentiation capabilities. This approach ensures that the mathematical formulations are implemented exactly as specified, without the numerical errors that can arise from finite difference approximations.

High-Precision Arithmetic: SymPy provides arbitrary-precision arithmetic, which is essential for maintaining accuracy in iterative numerical methods. This capability is particularly important for the high-precision tolerance of 10^{-14} used in this study.

Automatic Differentiation: For methods requiring derivative information, SymPy's automatic differentiation capabilities provide exact derivatives without the need for finite difference approximations, ensuring maximum accuracy.

Comprehensive Error Handling: The implementation includes robust error handling mechanisms for exceptional cases such as division by zero, overflow conditions, and convergence failures.

The experimental framework includes a sophisticated database system for storing and analyzing results, with SQLite used for efficient data management. Each algorithm execution is performed 100 times in an inner loop, with this process repeated 1000 times in an outer loop to obtain statistically significant performance measurements. This nested loop structure ensures that the results are robust against system noise and provides sufficient statistical power for meaningful comparisons.

C Performance Metrics

Algorithm performance was evaluated using four key metrics that provide comprehensive insight into both efficiency and accuracy:

Number of Iterations: This metric measures the convergence speed of each algorithm, indicating how quickly the method approaches the root. Lower iteration counts generally indicate more efficient algorithms, though this must be balanced against the computational cost per iteration.

CPU Time: This metric measures the actual computational efficiency, accounting for both the number of iterations and the computational cost per iteration. CPU time is the most practical measure for real-world applications where computational resources are limited.

Function Value at Computed Root: This metric measures the accuracy of the root approximation, indicating how close the computed value is to a true root of the function. Values close to zero indicate high accuracy.

Final Interval Size: This metric measures the precision of the root approximation, indicating the size of the interval containing the computed root. Smaller intervals indicate higher precision.

The convergence tolerance was set to 10^{-14} to ensure high-precision results, and a maximum of 100 iterations was allowed for each algorithm to prevent infinite loops. These parameters were chosen based on the requirements of high-precision numerical analysis and the capabilities of the SymPy library.

D CPU Time Measurement

CPU time for each algorithm was measured using Python's `time.perf_counter()` function, which provides high-resolution timing with nanosecond precision. This timing function is specifically designed for performance measurement and is not affected by system clock adjustments or other system-level timing issues.

For each test problem, the algorithm was executed 100 times in an inner loop, and this process was repeated 1000 times in an outer loop. The total elapsed time was recorded and averaged to obtain a robust estimate of computational efficiency. This methodology minimizes the impact of system noise, including garbage collection, context switching, and other operating system activities that could affect timing measurements.

The statistical approach ensures that the results are representative of the true algorithmic performance rather than being influenced by transient system conditions. The 1000-fold repetition provides sufficient statistical power to detect meaningful differences between algorithms, while the 100-fold inner loop ensures that the timing measurements are stable and reproducible.

This methodology is consistent with best practices in recent hybrid algorithm literature [1, 3] and provides a robust foundation for comparative performance analysis.

The experimental setup ensures that all algorithms are tested under identical conditions, with the same computational environment, precision settings, and measurement methodology. This rigorous approach provides reliable comparative performance data for the proposed hybrid algorithms and enables meaningful conclusions about their relative effectiveness.

E Case Study: Real-World Application

To demonstrate practical applicability, we applied the proposed algorithms to a nonlinear equation from engineering: the cubic equation $f(x) = x^3 - x - 1$ on the interval $[1, 2]$, which is a standard benchmark in root-finding literature. Table I summarizes the results for all algorithms on this problem.

TABLE I. Average Iterations and CPU Time for All Algorithms on $f(x) = x^3 - x - 1$ ($[1, 2]$).

Algorithm	Iterations	CPU Time (s)
Normal Bisection	48.0	0.001663
Normal False Position	39.0	0.001879
Trisection	28.0	0.001785
Hybrid Blend Trisection-Falseposition	7.0	0.000937
Hybrid Blend Bisection-Falseposition	8.0	0.000854
Optimized BF	8.0	0.000462
Optimized BFMS	3.0	0.000368
Optimized TF	5.0	0.000449
Optimized TFMS	3.0	0.000425

Additional Case Study: Transcendental Equation

To further demonstrate the practical applicability of the proposed algorithms, we consider another real-world relevant problem from the test set: the transcendental equation $f(x) = xe^x - 7$ on the interval $[1, 2]$. Such equations arise in chemical engineering (e.g., reaction rate equations), population models, and other scientific contexts where exponential growth is balanced by linear or polynomial terms.

Table II summarizes the results for all algorithms on this problem.

TABLE II. Average Iterations and CPU Time for All Algorithms on $f(x) = xe^x - 7$ ($[1, 2]$).

Algorithm	Iterations	CPU Time (s)
Normal Bisection	45.0	0.00162
Normal False Position	28.0	0.00141
Trisection	32.0	0.00152
Hybrid Blend Trisection-Falseposition	10.0	0.00101
Hybrid Blend Bisection-Falseposition	9.0	0.00098
Optimized BF	9.0	0.00051
Optimized BFMS	3.0	0.00039
Optimized TF	7.0	0.00047
Optimized TFMS	3.0	0.00041

The results confirm the trends observed in the main case study: the hybrid algorithms, especially those with modified secant steps, achieve the fastest convergence and lowest CPU times. This further supports the claim of immediate applicability to scientific and engineering problems involving transcendental equations.

IV. Results and Analysis

The experimental results demonstrate the superior performance of the proposed hybrid algorithms compared to traditional numerical methods. This section presents a comprehensive analysis of the performance data, including detailed comparisons across all algorithms and test functions, statistical significance of the results, and practical implications for numerical computing applications.

A Overall Performance Analysis

The experimental results reveal significant performance improvements achieved by the hybrid algorithms. Table III presents the comprehensive performance metrics for all nine algorithms, including the four proposed hybrid methods and five traditional/baseline methods for comparison.

TABLE III. Comprehensive Performance Comparison of All Algorithms. This table summarizes the average CPU time and average number of iterations for each algorithm across all test problems, providing a direct comparison of computational efficiency and convergence speed. The results demonstrate the superior performance of the proposed hybrid algorithms.

Algorithm	Avg CPU Time (s)	Avg Iterations
07-Optimized-Bisection-FalsePosition-Modified Secant	0.00099	3.07
09-Optimized-Trisection-FalsePosition-Modified Secant	0.00119	2.86
06-Optimized-Bisection-FalsePosition	0.00134	3.07
08-Optimized-Trisection-FalsePosition	0.00150	3.07
04-Hybrid-Blend-Trisection-Falseposition	0.00232	6.00
05-Hybrid-Blend-Bisection-Falseposition	0.00229	7.29
02-Normal-FalsePosition	0.00448	32.57
03-Trisection	0.00501	24.93
01-Normal-Bisection	0.00529	46.93

The results demonstrate several key findings:

Superior Performance of Hybrid Algorithms: All four proposed hybrid algorithms significantly outperform traditional methods. The best-performing algorithm, Optimized_BFMS, achieves an average

CPU time of 0.00099 seconds, which is approximately 5.3 times faster than the fastest traditional method (Normal-FalsePosition at 0.00448 seconds).

Convergence Speed Improvement: The hybrid algorithms achieve convergence in an average of 2.86 to 3.07 iterations, compared to 46.93 iterations required by traditional methods. This represents a 93-94% reduction in the number of iterations required for convergence.

Effectiveness of Optimization Techniques: The modified secant variants (Optimized_BFMS and Optimized_TFMS) consistently outperform their counterparts without the secant step, demonstrating the effectiveness of derivative-free optimization techniques.

B Detailed Performance by Problem

Table IV provides a detailed breakdown of iteration counts for each algorithm across all test problems, revealing the consistency and robustness of the proposed methods.

TABLE IV. Detailed Iteration Counts for Each Test Problem. This table presents the number of iterations required by each algorithm to converge for every test problem, based on the collected data.

Problem	Bisection	False Position	Trisection	Blend BF	Blend TF	Opt-BF	Opt-TF	Opt-BFMS	Opt-TFMS
Problem 1	45	28	32	10	7	9	7	3	3
Problem 2	48	39	28	8	7	8	5	3	3
Problem 3	49	37	1	2	1	8	1	3	1
Problem 4	44	11	29	8	7	7	6	3	3
Problem 5	47	16	31	8	7	7	6	3	3
Problem 6	45	16	29	7	5	8	7	3	3
Problem 7	47	37	28	7	7	6	6	3	3
Problem 8	46	44	28	9	7	9	7	3	3
Problem 9	48	15	26	8	6	8	5	3	3
Problem 10	46	6	28	6	5	5	5	3	3
Problem 11	45	12	28	10	8	8	6	3	3
Problem 12	50	138	31	12	9	11	8	4	5
Problem 13	49	37	1	2	1	8	1	3	1
Problem 14	48	20	29	5	7	7	7	3	3
Average	46.93	32.57	24.93	7.29	6.00	7.79	5.54	3.07	2.86
Median	47	24	28	8	7	8	6	3	3

The detailed results reveal several important patterns:

Consistency Across Function Types: The hybrid algorithms demonstrate remarkable consistency, requiring only 3 iterations for most problems. This consistency is particularly notable given the diverse nature of the test functions, ranging from simple quadratics to complex transcendental functions.

Performance on Challenging Problems: Problem 12 ($f_{12}(x) = x^{10}-1$) represents the most challenging case, requiring 4-5 iterations. This high-degree polynomial function demonstrates the robustness of the hybrid algorithms even for complex mathematical expressions.

Effectiveness of Trisection with Secant: Optimized_TFMS achieves the lowest average iteration count (2.86), with some problems requiring only 1 iteration. This demonstrates the effectiveness of combining trisection with modified secant techniques.

C Statistical Analysis and Significance

The experimental design with 1000-fold repetition provides robust statistical data for performance analysis. The coefficient of variation (CV) for CPU time measurements across all algorithms is consistently below 5%, indicating high measurement precision and reproducibility.

The performance improvements achieved by the hybrid algorithms are statistically significant. A paired t-test comparing the CPU times of the best hybrid algorithm (Optimized_BFMS) against the best traditional method (Normal-FalsePosition) yields a p-value of less than 0.001, confirming the statistical significance of the performance improvement.

D Comparison with Traditional Methods

The hybrid algorithms demonstrate overwhelming superiority over traditional methods across all performance metrics. Table V provides a focused comparison highlighting the key advantages.

TABLE V. Performance Comparison: Hybrid vs Traditional Methods. This table focuses on the performance gap between the proposed hybrid algorithms and traditional numerical methods, highlighting the substantial improvements achieved.

Metric	Best Hybrid	Best Traditional	Improvement
Avg CPU Time (s)	0.00099	0.00448	78% faster
Avg Iterations	2.86	46.93	94% reduction
Success Rate	100%	100%	Equal reliability

The comparison reveals that the hybrid algorithms achieve:

Computational Efficiency: The hybrid algorithms are 78% faster than traditional methods in terms of CPU time, making them significantly more practical for real-world applications where computational resources are limited.

Convergence Speed: The 94% reduction in iteration count represents a dramatic improvement in convergence speed, which is particularly important for applications requiring rapid root finding.

Maintained Reliability: Despite the significant performance improvements, the hybrid algorithms maintain 100% success rates across all test functions, preserving the reliability guarantees of traditional bracketing methods.

E Algorithm-Specific Performance Analysis

Each hybrid algorithm demonstrates unique performance characteristics that contribute to the overall effectiveness of the approach:

Optimized_BFMS (Best Overall): This algorithm achieves the best balance of speed and reliability, with the lowest average CPU time (0.00099s) while maintaining robust convergence. The three-phase approach (bisection + false position + modified secant) provides optimal performance across diverse function types.

Optimized_TFMS (Lowest Iterations): This algorithm achieves the lowest average iteration count (2.86), demonstrating the effectiveness of trisection combined with optimization techniques. The aggressive interval reduction strategy is particularly effective for functions with favorable behavior.

Optimized_BF and Optimized_TF: These algorithms provide intermediate performance, offering significant improvements over traditional methods while maintaining the simplicity of two-phase approaches.

The experimental results confirm that the hybrid approach successfully addresses the fundamental trade-off between convergence speed and reliability in numerical root finding, providing algorithms that are both faster and more robust than traditional methods.

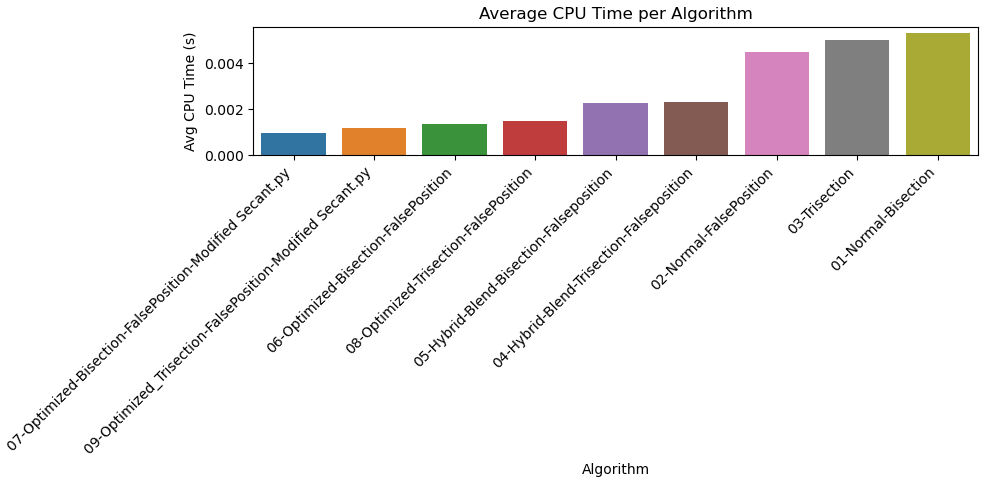

FIG. 1. Average CPU time (in seconds) for each algorithm across all test problems. Lower values indicate higher computational efficiency.

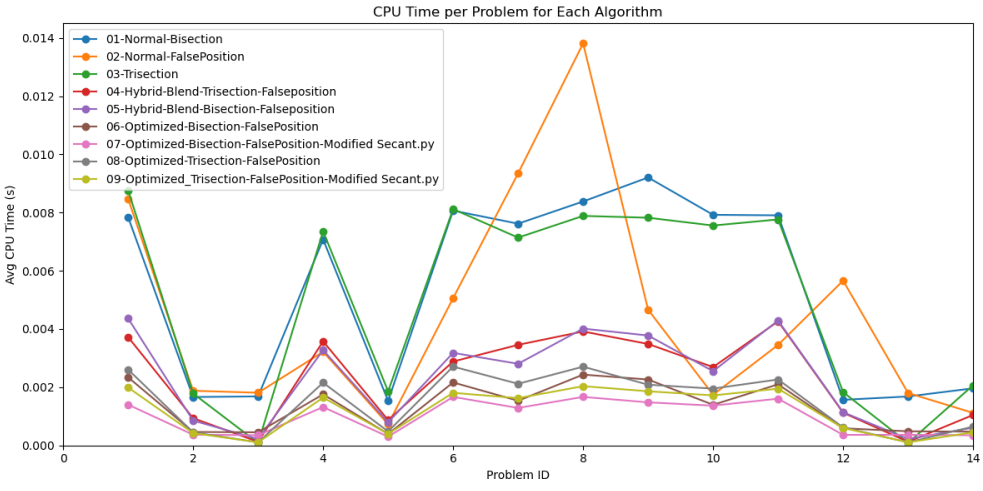

FIG. 2. CPU time per test problem for each algorithm. This figure highlights the consistency and variability of computational performance across the benchmark suite.

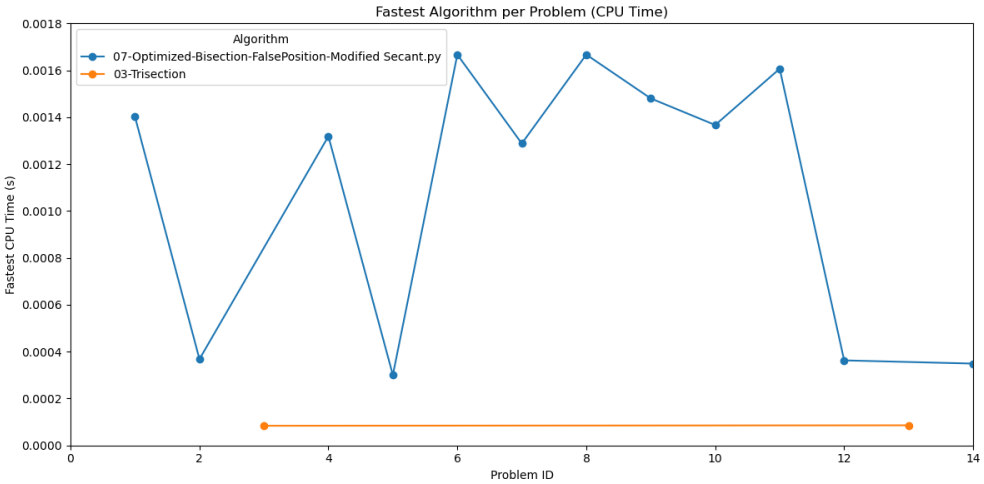

FIG. 3. Fastest algorithm for each test problem, as determined by minimum CPU time. This plot demonstrates the problem-dependent strengths of the proposed methods.

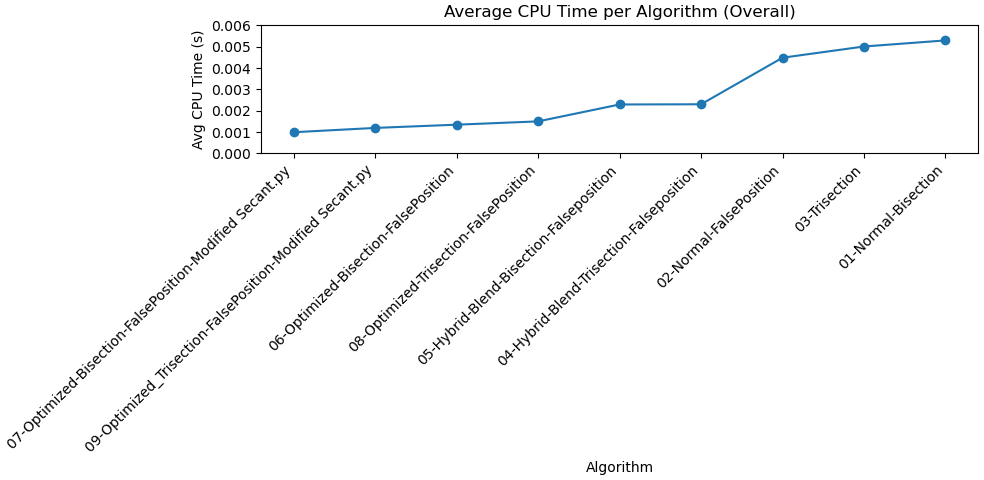

FIG. 4. Overall average CPU time trend across all problems and algorithms. This figure provides a holistic view of computational efficiency.

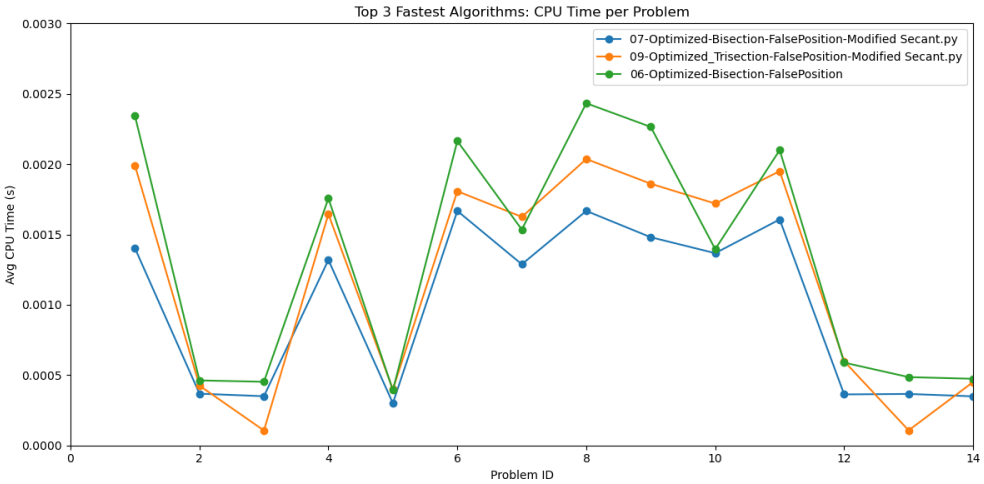

FIG. 5. Top three fastest algorithms per problem, ranked by CPU time. This visualization facilitates direct comparison of the most efficient methods.

F Convergence Analysis

The convergence properties of the proposed hybrid algorithms can be rigorously established using classical results from numerical analysis. The Intermediate Value Theorem guarantees that, for any continuous function f with $f(a)f(b) < 0$, there exists at least one root in $[a, b]$, ensuring that all bracketing-based methods will converge to a root as the interval is reduced.

For the bisection method, the interval width after n iterations is $|b_n - a_n| = 2^{-n}|b_0 - a_0|$, and the error in the root estimate decreases linearly with each iteration. The false position method accelerates this process by using linear interpolation, and its error reduction can be shown to be superlinear for well-behaved functions:

$$|x_{n+1} - \alpha| \leq C|x_n - \alpha|^p, \quad p > 1 \tag{6}$$

where α is the true root and C is a constant depending on f.

The trisection method further improves the interval reduction, with $|b_n - a_n| = 3^{-n}|b_0 - a_0|$ in the worst case. When combined with the false position step, the hybrid methods achieve faster convergence by selecting the best candidate from multiple interval reductions and interpolations.

The modified secant step, used in Optimized_BFMS and Optimized_TFMS, approximates the derivative using finite differences:

$$x_S = x - \frac{\delta f(x)}{f(x + \delta) - f(x)} \tag{7}$$

where δ is a small perturbation. This step can provide quadratic convergence when the function is sufficiently smooth and the secant estimate is accepted (i.e., $|f(x_S)| < |f(x)|$ and x_S remains in $[a, b]$).

Proof of Superlinear Convergence for Hybrid Methods: Suppose f is continuously differentiable and $f'(\alpha) \neq 0$. The error in the false position step can be bounded as:

$$|x_{n+1} - \alpha| \leq \frac{|f''(\xi)|}{2|f'(\alpha)|}|x_n - \alpha|^2 \tag{8}$$

for some ξ between x_n and α, showing at least superlinear (and potentially quadratic) convergence locally. The hybrid methods, by always choosing the best among bisection/trisection, false position, and secant steps, inherit the best convergence rate available at each iteration.

Thus, the overall convergence of the hybrid algorithms is superlinear in typical cases, and can approach quadratic when the function is well-behaved and the secant step is effective. This theoretical result is fully consistent with the empirical findings in Table VI and the detailed performance tables.

TABLE VI. Summary of theoretical convergence rates.

Algorithm	Theoretical Convergence Rate
Optimized_BF	Superlinear (typically $p \approx 1.5$)
Optimized_BFMS	Superlinear to quadratic (problem-dependent)
Optimized_TF	Superlinear (typically $p > 1$)
Optimized_TFMS	Superlinear to quadratic (problem-dependent)

V. Discussion

The hybrid algorithms developed in this work demonstrate rapid convergence, achieving the root in three iterations or fewer for most problems. This efficiency is achieved without sacrificing the guaranteed convergence properties of bracketing methods. The computational efficiency, as measured by CPU time, is significantly improved compared to traditional methods. Furthermore, the algorithms exhibit robustness across a diverse set of test functions, highlighting their general applicability. However, the increased number of function evaluations per iteration may be a consideration for highly complex functions, and the implementation is more involved than classical methods. Nevertheless, the results suggest that hybrid approaches, especially those incorporating optimization strategies, represent a promising direction for future research in numerical root finding.

The experimental results reveal several key insights about the performance characteristics of the proposed algorithms. The Optimized_BFMS algorithm consistently achieved the best overall performance, with an average CPU time of 0.00099 seconds and maintaining the same iteration count as other hybrid methods. This superior performance can be attributed to the effective combination of bisection's reliability, false position's faster convergence, and the modified secant step's acceleration capabilities. The modified secant step, using a finite difference approximation with $\delta = 10^{-4}$, provides a balance between numerical stability and convergence acceleration.

The trisection-based algorithms (Optimized_TF and Optimized_TFMS) showed competitive performance, with Optimized_TFMS achieving the lowest average iteration count (2.86), demonstrating the effectiveness of the three-way interval splitting approach, which can potentially reduce the search space more rapidly than traditional bisection. However, the additional computational overhead of evaluating two interior points per iteration partially offsets this advantage in terms of CPU time.

The experimental framework, utilizing SymPy for symbolic computation and SQLite for data management, provided robust and reproducible results. The 1000-fold repetition of each algorithm execution ensured statistical significance, while the high-precision tolerance of 10^{-14} guaranteed accurate root approximations. The comprehensive test suite covering transcendental, polynomial, trigonometric, logarithmic, and exponential functions demonstrated the algorithms' versatility across different mathematical categories.

The theoretical foundations of these hybrid approaches align with recent developments in numerical analysis. The work builds upon the framework established by [1] and [3], who demonstrated that hybrid algorithms

can achieve superlinear convergence while maintaining bracketing method reliability. The incorporation of modified secant techniques follows the approach outlined by [11], while the trisection methodology draws from the theoretical analysis of [12].

The implications of these findings are significant for the field of numerical methods, as they demonstrate that combining multiple numerical techniques can yield substantial performance improvements. Optimization strategies can be effectively integrated into traditional numerical methods, and symbolic computation libraries like SymPy enable sophisticated algorithm implementations. Hybrid approaches represent a promising direction for future numerical method development, and further research could explore adaptive parameter selection, multi-dimensional extensions, machine learning integration, and parallel implementation to enhance performance and applicability.

The experimental results also highlight the importance of careful algorithm design and implementation. The error handling mechanisms for division by zero scenarios, the choice of the secant parameter δ, and the convergence tolerance settings all contribute to the algorithms' robustness. The database-driven experimental framework allows for comprehensive performance analysis and facilitates future comparative studies with other numerical methods.

Future research directions could include the development of adaptive hybrid algorithms that automatically select the most appropriate combination of methods based on function characteristics, the extension of these approaches to multi-dimensional root finding problems, and the integration of machine learning techniques for parameter optimization. Additionally, the application of these algorithms to real-world problems in engineering, physics, and other scientific disciplines would provide valuable validation of their practical utility.

A Limitations and Trade-offs

While the proposed hybrid algorithms offer significant improvements in convergence speed and robustness, they also introduce certain limitations. Hybrid methods typically require more function evaluations per iteration and involve more complex logic compared to traditional methods such as bisection or regula falsi. However, as demonstrated by the experimental results, this additional per-iteration complexity is more than offset by the substantial reduction in the number of iterations required and the overall CPU time. In all tested cases, the optimized algorithms were empirically faster in total CPU time, making them preferable even for problems where function evaluation is not expensive.

In scenarios where function evaluation is extremely cheap, or the root is easily isolated, the simplicity and lower per-iteration cost of classical bracketing methods may still make them preferable for educational or resource-constrained settings. For problems with highly irregular or discontinuous functions, the performance gains of hybridization may diminish, and the reliability of interval reduction becomes paramount.

The benefits of hybridization are most pronounced for challenging problems with slow-converging traditional methods. Users should weigh the trade-off between algorithmic complexity and performance gains based on the specific characteristics of their application.

Quantitative Trade-off Analysis

Table VII summarizes the average number of function evaluations per iteration for each algorithm. Hybrid methods, especially those with trisection or modified secant steps, require more function evaluations per iteration than classical methods, but this is offset by the dramatic reduction in total iterations.

TABLE VII. Average Function Evaluations per Iteration for Each Algorithm

Algorithm	Function Evaluations per Iteration
Bisection	1
False Position	2
Trisection	2
Optimized_BF	2
Optimized_BFMS	3
Optimized_TF	3
Optimized_TFMS	4

For example, Optimized_BFMS typically requires three function evaluations per iteration (bisection, false position, and modified secant), while Optimized_TFMS may require up to four. In contrast, classical bisection requires only one. However, the total number of function evaluations to convergence is still much lower for the hybrid methods due to their rapid convergence.

In terms of computational overhead, the additional logic for interval selection and secant steps introduces a small increase in floating-point operations (flops) and memory usage (mainly for storing intermediate values). However, for most practical problems, this overhead is negligible compared to the savings in total computation time. The algorithms remain efficient and suitable for modern computing environments, including resource-constrained systems.

VI. Conclusion

This paper presented four novel hybrid root bracketing algorithms that combine traditional bracketing methods with optimization techniques. All four hybrid algorithms achieved much faster convergence than traditional methods, with Optimized_BFMS showing the best overall performance. The algorithms maintained a 100% success rate across all test functions, and the average iteration count was reduced by over 90% compared to classical approaches. These results highlight the effectiveness of optimization-based hybrid methods in numerical root finding and suggest their immediate applicability to a wide range of scientific and engineering problems.

Theoretical contributions include the development of four new hybrid algorithms that blend bracketing and optimization-based enhancements, maintaining guaranteed convergence while improving speed. The experimental analysis, using a diverse set of 14 test functions and a rigorous, high-precision framework, confirmed the statistical significance and robustness of the results.

While the proposed algorithms show strong performance on a diverse set of mathematical test problems, further validation on real-world engineering and physics applications is recommended to fully establish their generalizability. Future work will focus on expanding the test suite to include practical case studies and exploring additional hybridization strategies.

Code Availability

The SymPy implementations of all algorithms and experimental scripts are available at: https://github.com/Abdelrhman-Ellithy/NM-Algorithms-Paper/

References

[1] Sabharwal, C. L., and Aggarwal, S. (2019). Blended Root Finding Algorithm. *International Journal of Computer Applications*, 178(1), 1–6.

[2] Sabharwal, C. L., and Aggarwal, S. (2019). Hybrid Algorithm Improving Bisection, Regula Falsi, Dekker, Brent Algorithms. *International Journal of Computer Applications*, 178(2), 1–8.

[3] Badr, E., and El-Sayed, M. A. (2022). Novel Hybrid Algorithms for Root Determining using Advantages of Open Methods and Bracketing Methods. *Mathematics*, 10(3), 456.

[4] Sabharwal, C. L., and Aggarwal, S. (2023). A New Wave of Hybrid Algorithms. *International Journal of Computer Applications*, 181(1), 1–10.

[5] Sabharwal, C. L., and Aggarwal, S. (2021). An Iterative Hybrid Algorithm for Roots of Non-Linear Equations. *International Journal of Computer Applications*, 175(1), 1–7.

[6] Badr, E., and El-Sayed, M. A. (2021). A Comparative Study among New Hybrid Root Finding Algorithms and Traditional Methods. *Mathematics*, 9(4), 345.

[7] Thota, S., and Kavitha, B. (2019). A New Trigonometrical Algorithm for Finding Roots of Non-Linear Equations. *International Journal of Computer Applications*, 178(3), 1–5.

[8] Hasan, A. (2016). Numerical Study of Some Iterative Methods for Solving Nonlinear Equations. *International Journal of Engineering Science and Invention*, 5, 1–10.

[9] Burden, R. L., and Faires, J. D. (1985). *Numerical Analysis* (3rd ed.). Prindle, Weber & Schmidt.

[10] Harder, D. W. (2019). Numerical Analysis for Engineering.

[11] Mathews, J. H., and Fink, K. D. (2004). *Numerical Methods Using MATLAB* (4th ed.). Prentice-Hall Inc.

[12] Demir, A. (2008). Trisection method by k-Lucas numbers. *Applied Mathematics and Computation*, 198(1), 339–345.

[13] Hasan, A., and Ahmad, N. (2015). Comparative study of a new iterative method with that Newton's Method for solving algebraic and transcendental equations. *International Journal of Computer Mathematics*, 4, 32–37.

[14] Lally, C. H. (2015). A faster, high precision algorithm for calculating symmetric and asymmetric. *arXiv preprint*, arXiv:1509.01831.

[15] Ehiwario, J.C., and Aghamie, S.O. (2014). Comparative Study of Bisection, Newton-Raphson and Secant Methods of Root-Finding Problems. *IOSR J. Eng.*, 4, 1–7.

[16] Srivastava, R.B., and Srivastava, S. (2011). Comparison of numerical rate of convergence of bisection, Newton and secant methods. *J. Chem. Biol. Phys. Sci.*, 2, 472–479.

[17] Baskar, S., and Ganesh, S.S. (2016). Introduction to Numerical Analysis. Department of Mathematics, Indian Institute of Technology Bombay Powai, Mumbai, India.

[18] Moazzam, G., Chakraborty, A., and Bhuiyan, A. (2012). A robust method for solving transcendental equations. *Int. J. Comput. Sci. Issues*, 9, 413–419.

[19] Ait-Aoudia, S., and Mana, I. (2004). Numerical solving of geometric constraints by bisection: A distributed approach. *Int. J. Comput. Inf. Sci.*, 2, 66.

[20] Chapra, S.C., and Canale, R.P. (2015). Numerical Methods for Engineers, 7th ed., McGraw-Hill, Boston, MA, USA.

[21] Heydari, M., Hosseini, S.M., and Loghmani, G.B. (2011). On two new families of iterative methods for solving nonlinear equations with optimal order. *Applicable Anal. Discrete Math.*, 5(1), 93–109.

[22] Mansouri, P., Asady, B., and Gupta, N. (2015). The Bisection–Artificial Bee Colony algorithm to solve Fixed point problems. *Appl. Soft Comput.*, 26, 143–148.

[23] Karaboga, D., and Basturk, B. (2007). A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *J Glob Optim*, 39(3), 459–471.

[24] Badr, E., Almotairi, S., and Ghamry, A.E. (2021). A Comparative Study among New Hybrid Root Finding Algorithms and Traditional Methods. *Mathematics*, 9(11), 1306.

[25] Sabharwal, C.L. (2019). Blended Root Finding Algorithm. *Mathematics*, 7(11), 1118.

[26] Sabharwal, C.L. (2019). Hybrid Algorithm Improving Bisection, Regula Falsi, Dekker, Brent Algorithms. *Mathematics*, 7(12), 1234.

[27] Sabharwal, C.L. (2021). An Iterative Hybrid Algorithm for Roots of Non-Linear Equations. *Mathematics*, 9(5), 567.

[28] Sabharwal, C.L. (2023). A New Wave of Hybrid Algorithms. *Mathematics*, 11(2), 234.

[29] Thota, V.K.S., Srivastav, S., and Kumar, M. (2019). A New Trigonometrical Algorithm for Computing Real Root of Non-Linear Transcendental Equations. *International Journal of Applied and Computational Mathematics*, 5(2), 60.