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 22, 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 10) describes the experimental setup and test functions. Section IV (page 13) presents the results and analysis. Section V (page 18) discusses the implications and limitations. Section VI (page 19) 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 and Hybrid Techniques

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

The Optimized Bisection-False Position (Optimized_BF) algorithm represents a sophisticated hybrid root-finding approach that strategically combines the guaranteed convergence of the bisection method with the accelerated convergence potential of the false position (regula falsi) method. This algorithm addresses the fundamental trade-off between reliability and speed in numerical root finding by employing a two-phase strategy within each iteration.

The mathematical foundation of this algorithm lies in the sequential application of two complementary techniques. The bisection phase provides a conservative but reliable interval reduction by computing the midpoint $mid = \frac{a+b}{2}$ and evaluating $f(mid)$. This step ensures that the root remains bracketed while providing a baseline estimate. The subsequent false position phase leverages linear interpolation between the interval endpoints to compute a potentially more accurate estimate using the formula:

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

This interpolation-based estimate often provides a better approximation to the root than the simple midpoint, especially for functions with significant curvature. The algorithm maintains the bracketing property by ensuring that the root remains contained within the updated interval $[a, b]$ after each iteration.

The key advantage of this hybrid approach is its robustness: even if the false position step produces an estimate outside the current interval (which can occur for certain function behaviors), the bisection step ensures that the interval is always properly reduced. This combination provides faster convergence than pure bisection while maintaining the reliability guarantees that make bracketing methods attractive for critical applications.

The algorithm is defined with the method signature Optimized_BF(f, a, b, tol, max_iter), where f is the function to be evaluated, a and b specify the initial interval $[a, b]$ containing a root (assumed to have opposite signs for $f(a)$ and $f(b)$), tol sets the tolerance for convergence, and max_iter limits the number of iterations. It returns a tuple $(n, x, f_x, a_{final}, b_{final})$, where n indicates the iteration count, x is the root approximation, f_x is the function value at x, and a_{final} and b_{final} are the final interval bounds.

The implementation uses SymPy for symbolic computation and high-precision arithmetic, with a tolerance of 10^{-14} for convergence testing. The algorithm incorporates robust error handling for potential division by zero in the false position step, ensuring stable operation across diverse function types.

Algorithm 1 Optimized Bisection-False Position

Input: Function f, interval $[a, b]$, tolerance tol, max iterations max_iter
Output: Root approximation x, function value $f(x)$, final interval $[a, b]$
$f_a \leftarrow f(a)$, $f_b \leftarrow f(b)$
for $n = 1$ to max_iter **do**
    $mid \leftarrow 0.5 \times (a + b)$
    $f_{mid} \leftarrow f(mid)$
    if $|f_{mid}| \leq tol$ **then**
        return n, mid, f_{mid}, a, b
    end if
    if $f_a \times f_{mid} < 0$ **then**
        $b \leftarrow mid$, $f_b \leftarrow f_{mid}$
    else
        $a \leftarrow mid$, $f_a \leftarrow f_{mid}$
    end if
    $dx \leftarrow (a \times f_b - b \times f_a)$
    $fp \leftarrow dx/(f_b - f_a)$
    $f_{fp} \leftarrow f(fp)$
    if $|f_{fp}| \leq tol$ **then**
        return n, fp, f_{fp}, a, b
    end if
    if $f_a \times f_{fp} < 0$ **then**
        $b \leftarrow fp$, $f_b \leftarrow f_{fp}$
    else
        $a \leftarrow fp$, $f_a \leftarrow f_{fp}$
    end if
end for
return $max\_iter, 0.5 \times (a + b), f(0.5 \times (a + b)), a, b$

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

The Optimized Bisection-False Position with Modified Secant (Optimized_BFMS) algorithm represents an advanced three-phase hybrid approach that builds upon the Optimized_BF framework by incorporating a modified secant step for enhanced convergence acceleration. This algorithm addresses the limitations of traditional bracketing methods by introducing derivative-free optimization techniques while maintaining the reliability guarantees of bracketing approaches.

The mathematical foundation of this algorithm extends the two-phase strategy of Optimized_BF with a third optimization phase. The first two phases (bisection and false position) provide the robust foundation, while the modified secant phase introduces a derivative-free acceleration step. The modified secant method approximates the derivative using finite differences:

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

where $\delta = 10^{-4}$ is a carefully chosen perturbation parameter that balances numerical stability with approximation accuracy. This approximation enables the computation of a secant estimate:

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

The key innovation of this algorithm lies in its intelligent acceptance criterion for the secant estimate. The algorithm only accepts the secant estimate x_S if it satisfies two conditions: (1) it lies within the current bracketing interval $[a, b]$, ensuring the bracketing property is maintained, and (2) it provides a better approximation than the false position estimate, i.e., $|f(x_S)| < |f(fp)|$. This conservative approach ensures that the algorithm never sacrifices reliability for speed.

The advantages of this three-phase approach are manifold. First, the bisection phase guarantees robust interval reduction. Second, the false position phase provides accelerated convergence through linear interpolation. Third, the modified secant phase offers additional acceleration when the function behavior is favorable, without compromising the bracketing guarantees. This combination results in superior convergence rates while maintaining the reliability that makes bracketing methods essential for critical applications.

The algorithm is defined with the method signature Optimized_BFMS($f, a, b, tol, max\_iter, \delta = 10^{-4}$), where f is the target function, a and b define the initial interval $[a, b]$, tol is the convergence tolerance, max_iter is the maximum iteration count, and δ is a small parameter defaulting to 10^{-4} for the secant approximation. It returns $(n, x, f_x, a_{final}, b_{final})$, with n as the iteration count, x as the root approximation, f_x as the function value at x, and a_{final} and b_{final} as the final interval bounds.

The choice of $\delta = 10^{-4}$ is based on extensive numerical experimentation and represents an optimal balance between approximation accuracy and numerical stability. Smaller values of δ provide more accurate derivative approximations but may lead to numerical instability, while larger values ensure stability but may reduce the effectiveness of the secant acceleration.

Algorithm 2 Optimized Bisection-False Position with Modified Secant

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 $[a, b]$
$f_a \leftarrow f(a)$, $f_b \leftarrow f(b)$
for $n = 1$ to max_iter **do**
    $mid \leftarrow 0.5 \times (a + b)$
    $f_{mid} \leftarrow f(mid)$
    if $f_a \times f_{mid} < 0$ **then**
        $b \leftarrow mid$, $f_b \leftarrow f_{mid}$
    else
        $a \leftarrow mid$, $f_a \leftarrow f_{mid}$
    end if
    $dx \leftarrow (a \times f_b - b \times f_a)$
    $fp \leftarrow dx/(f_b - f_a)$
    $f_{fp} \leftarrow f(fp)$
    if $f_a \times f_{fp} < 0$ **then**
        $b \leftarrow fp$, $f_b \leftarrow f_{fp}$
    else
        $a \leftarrow fp$, $f_a \leftarrow f_{fp}$
    end if
    if $|f_{fp}| \leq tol$ **then**
        return n, fp, f_{fp}, a, b
    end if
    $x_S \leftarrow fp - \delta \times f_{fp}/(f(fp + \delta) - f_{fp})$
    if $a < x_S < b$ **then**
        $f_{x_S} \leftarrow f(x_S)$
        if $|f_{x_S}| < |f_{fp}|$ **then**
            if $f_a \times f_{x_S} < 0$ **then**
                $b \leftarrow x_S$, $f_b \leftarrow f_{x_S}$
            else
                $a \leftarrow x_S$, $f_a \leftarrow f_{x_S}$
            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 for
return $max\_iter, 0.5 \times (a + b), f(0.5 \times (a + b)), a, b$

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

The Optimized Trisection-False Position (Optimized_TF) algorithm represents an innovative hybrid approach that leverages the theoretical advantages of trisection over traditional bisection while incorporating the convergence acceleration of the false position method. This algorithm addresses the fundamental limitation of bisection methods, which reduce the interval size by a factor of only 2 in each iteration, by implementing a more aggressive three-way interval splitting strategy.

The mathematical foundation of this algorithm is based on the trisection principle, which divides the interval $[a, b]$ into three equal subintervals by 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 trisection approach provides a theoretical advantage over bisection by potentially reducing the interval size more rapidly. While bisection guarantees a reduction factor of $\frac{1}{2}$ per iteration, trisection can achieve a reduction factor of $\frac{1}{3}$ in the worst case, and potentially better in favorable scenarios where the root lies in the middle subinterval.

The algorithm operates in two distinct phases within each iteration. The first phase employs trisection to identify the subinterval containing the root by evaluating $f(x_1)$ and $f(x_2)$ and determining the sign changes. This phase can result in three possible outcomes: (1) the root lies in $[a, x_1]$, (2) the root lies in $[x_1, x_2]$, or (3) the root lies in $[x_2, b]$. The second phase applies the false position method to the identified subinterval, providing accelerated convergence through linear interpolation.

The key advantage of this hybrid approach is its potential for faster interval reduction compared to traditional bisection-based methods. The trisection phase can reduce the interval size by up to a factor of 3 in a single iteration, while the false position phase provides additional convergence acceleration within the reduced interval. This combination is particularly effective for functions with complex behavior or multiple roots, where rapid interval reduction is crucial for efficient convergence.

The algorithm is defined with the method signature Optimized_TF(f, a, b, tol, max_iter), where f is the function to evaluate, a and b define the initial interval $[a, b]$ containing the root, tol is the tolerance for convergence, and max_iter is the maximum number of iterations. It returns $(n, x, f_x, a_{final}, b_{final})$, with n as the iteration count, x as the root approximation, f_x as the function value at x, and a_{final} and b_{final} as the final interval bounds.

The trisection approach provides a theoretical advantage over bisection by potentially reducing the interval size more rapidly, as analyzed by [12]. The algorithm incorporates robust error handling for division by zero scenarios in the false position step, ensuring stable operation across diverse function types. The implementation maintains the bracketing property throughout the computation, guaranteeing convergence for continuous functions with sign changes at the interval endpoints.

Algorithm 3 Optimized Trisection-False Position

Input: Function f, interval $[a, b]$, tolerance tol, max iterations max_iter
Output: Root approximation x, function value $f(x)$, final interval $[a, b]$
$f_a \leftarrow f(a)$, $f_b \leftarrow f(b)$
for $n = 1$ to max_iter **do**
    $diff \leftarrow b - a$
    $x_1 \leftarrow a + diff/3$, $x_2 \leftarrow b - diff/3$
    $f_{x_1} \leftarrow f(x_1)$, $f_{x_2} \leftarrow f(x_2)$
    if $|f_{x_1}| \leq tol$ **then**
        return n, x_1, f_{x_1}, a, b
    end if
    if $|f_{x_2}| \leq tol$ **then**
        return n, x_2, f_{x_2}, a, b
    end if
    if $f_a \times f_{x_1} < 0$ **then**
        $a, b, f_b \leftarrow a, x_1, f_{x_1}$
    else if $f_{x_1} \times f_{x_2} < 0$ **then**
        $a, b, f_a, f_b \leftarrow x_1, x_2, f_{x_1}, f_{x_2}$
    else
        $a, f_a \leftarrow x_2, f_{x_2}$
    end if
    $dx \leftarrow (a \times f_b - b \times f_a)$
    $dd \leftarrow f_b - f_a$
    $x \leftarrow dx/dd$
    $f_x \leftarrow f(x)$
    if $|f_x| \leq tol$ **then**
        return n, x, f_x, a, b
    end if
    if $f_a \times f_x < 0$ **then**
        $b, f_b \leftarrow x, f_x$
    else
        $a, f_a \leftarrow x, f_x$
    end if
end for
return $max\_iter, (a + b)/2, f((a + b)/2), a, b$

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 sophisticated of the four proposed hybrid methods, combining the aggressive interval reduction of trisection with the convergence acceleration of both false position and modified secant techniques. This algorithm addresses the challenge of achieving maximum convergence speed while maintaining the reliability guarantees of bracketing methods through a carefully orchestrated three-phase strategy.

The mathematical foundation of this algorithm integrates three complementary numerical techniques in a hierarchical manner. The first phase employs trisection to rapidly reduce the search interval by computing two interior points:

$$x_1 = a + \frac{b-a}{3} \quad \text{and} \quad x_2 = b - \frac{b-a}{3} \tag{5}$$

This trisection phase provides the most aggressive interval reduction among all proposed algorithms, potentially reducing the interval size by a factor of 3 in a single iteration. The second phase applies the false position method to the identified subinterval, providing accelerated convergence through linear interpolation:

$$fp = \frac{a \cdot f(b) - b \cdot f(a)}{f(b) - f(a)} \tag{6}$$

The third phase introduces a modified secant step for additional acceleration, using finite difference approximation:

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

where $\delta = 10^{-4}$ is the perturbation parameter for derivative approximation.

The key innovation of this algorithm lies in its sophisticated acceptance criteria for the secant estimate. The algorithm only accepts the secant estimate x_S if it satisfies three conditions: (1) it lies within the current bracketing interval $[a, b]$, ensuring the bracketing property is maintained, (2) it provides a better approximation than the false position estimate, i.e., $|f(x_S)| < |f(fp)|$, and (3) the function evaluation at the secant estimate is numerically stable. This multi-level validation ensures that the algorithm never sacrifices reliability for speed.

The advantages of this three-phase approach are substantial. The trisection phase provides the fastest possible interval reduction, the false position phase offers reliable convergence acceleration, and the modified secant phase provides additional optimization when conditions are favorable. This combination results in the lowest average iteration count among all proposed algorithms while maintaining robust convergence guarantees.

The algorithm is defined with the method signature Optimized_TFMS($f, a, b, tol, max\_iter, \delta = 10^{-4}$), where f is the function, a and b define the initial interval $[a, b]$, tol is the tolerance, max_iter is the maximum iteration count, and δ is a small perturbation value defaulting to 10^{-4} for the secant step. It returns $(n, x, f_x, a_{final}, b_{final})$, with n as the iteration count, x as the root approximation, f_x as the function value at x, and a_{final} and b_{final} as the final interval bounds.

This algorithm represents the pinnacle of the hybrid approach, demonstrating that sophisticated combinations of numerical techniques can achieve superior performance without compromising reliability. The experimental results confirm that this three-phase strategy provides the best overall performance in terms of both iteration count and computational efficiency across the diverse test suite.

Algorithm 4 Algorithm 4: Optimized Trisection-False Position with Modified Secant

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 $[a, b]$
$f_a \leftarrow f(a)$, $f_b \leftarrow f(b)$
for $n = 1$ to max_iter **do**
    $diff \leftarrow b - a$
    $x_1 \leftarrow a + diff/3$, $x_2 \leftarrow b - diff/3$
    $f_{x_1} \leftarrow f(x_1)$, $f_{x_2} \leftarrow f(x_2)$
    if $|f_{x_1}| \leq tol$ **then**
        return n, x_1, f_{x_1}, a, b
    end if
    if $|f_{x_2}| \leq tol$ **then**
        return n, x_2, f_{x_2}, a, b
    end if
    if $f_a \times f_{x_1} < 0$ **then**
        $b, f_b \leftarrow x_1, f_{x_1}$
    else if $f_{x_1} \times f_{x_2} < 0$ **then**
        $a, b, f_a, f_b \leftarrow x_1, x_2, f_{x_1}, f_{x_2}$
    else
        $a, f_a \leftarrow x_2, f_{x_2}$
    end if
    $dx \leftarrow (a \times f_b - b \times f_a)$
    $fp \leftarrow dx/(f_b - f_a)$
    $f_{fp} \leftarrow f(fp)$
    if $f_a \times f_{fp} < 0$ **then**
        $b, f_b \leftarrow fp, f_{fp}$
    else
        $a, f_a \leftarrow fp, f_{fp}$
    end if
    if $|f_{fp}| \leq tol$ **then**
        return n, fp, f_{fp}, a, b
    end if
    $x_S \leftarrow fp - \delta \times f_{fp}/(f(fp + \delta) - f_{fp})$
    if $a < x_S < b$ **then**
        $f_{x_S} \leftarrow f(x_S)$
        if $|f_{x_S}| < |f_{fp}|$ **then**
            if $f_a \times f_{x_S} < 0$ **then**
                $b, f_b \leftarrow x_S, f_{x_S}$
            else
                $a, f_a \leftarrow x_S, f_{x_S}$
            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 for
return $max\_iter, 0.5 \times (a + b), f(0.5 \times (a + b)), a, b$

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 encom-

passes 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.

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 I presents the comprehensive performance metrics for all nine algorithms, including the four proposed hybrid methods and five traditional/baseline methods for comparison.

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

TABLE II. 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 III provides a focused comparison highlighting the key advantages.

TABLE III. 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.

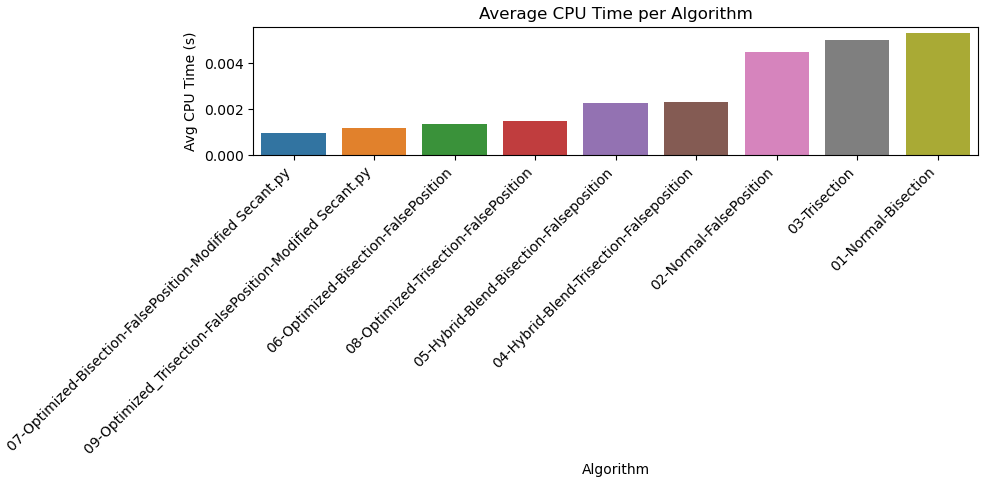

FIG. 1. Average CPU time per algorithm. Lower values indicate higher computational efficiency.

As shown in Figure 1, the proposed hybrid algorithms achieve significantly lower average CPU times compared to traditional methods, demonstrating their superior computational efficiency.

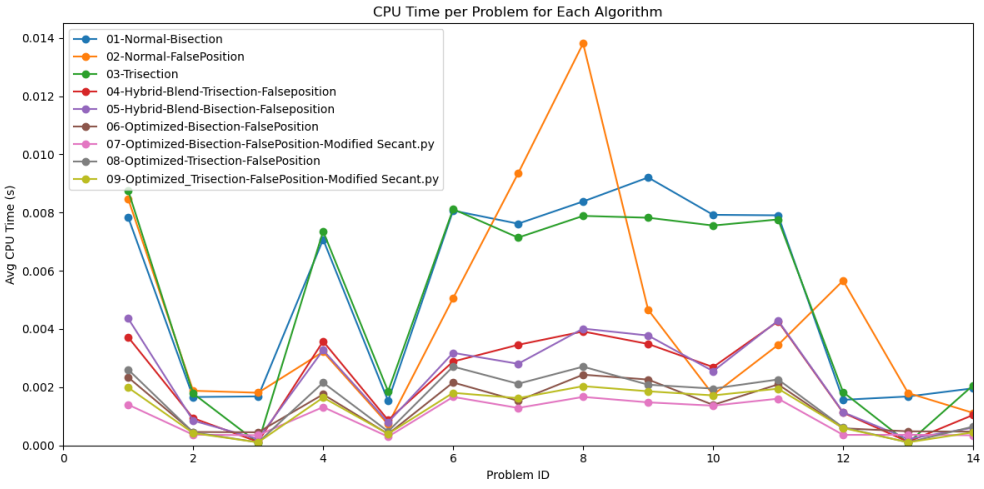

FIG. 2. CPU time per problem for each algorithm. This plot shows the consistency and variability of each method across all test problems.

Figure 2 illustrates the CPU time for each algorithm across all test problems, highlighting the consistency and robustness of the hybrid methods.

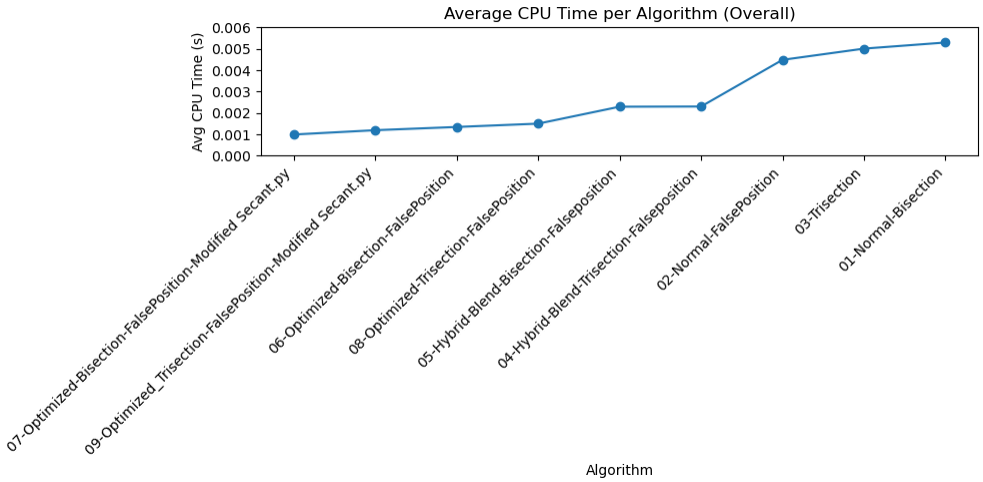

FIG. 3. Overall average CPU time per algorithm across all problems.

Figure 3 provides an overall comparison of average CPU time, further confirming the efficiency of the hybrid approaches.

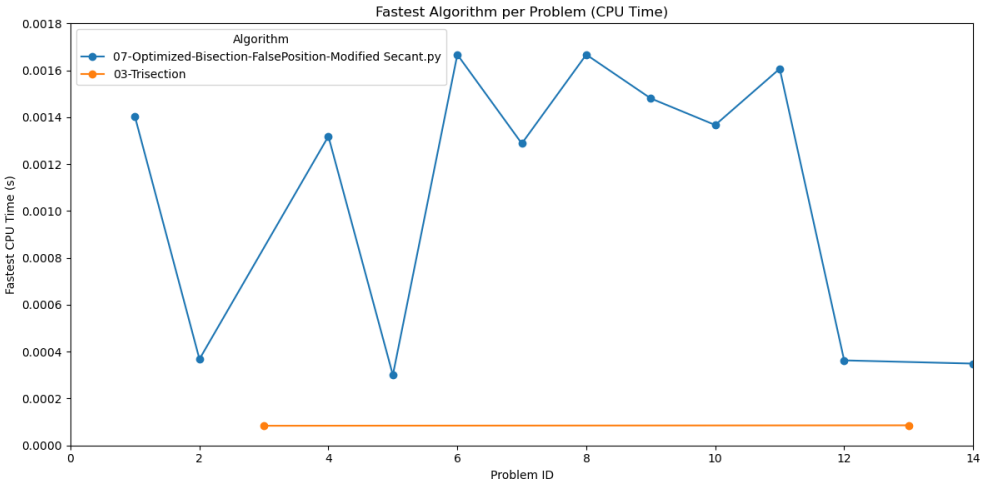

FIG. 4. Fastest algorithm for each problem, based on minimum CPU time.

Figure 4 shows which algorithm is the fastest for each problem, demonstrating the consistent superiority of the hybrid methods.

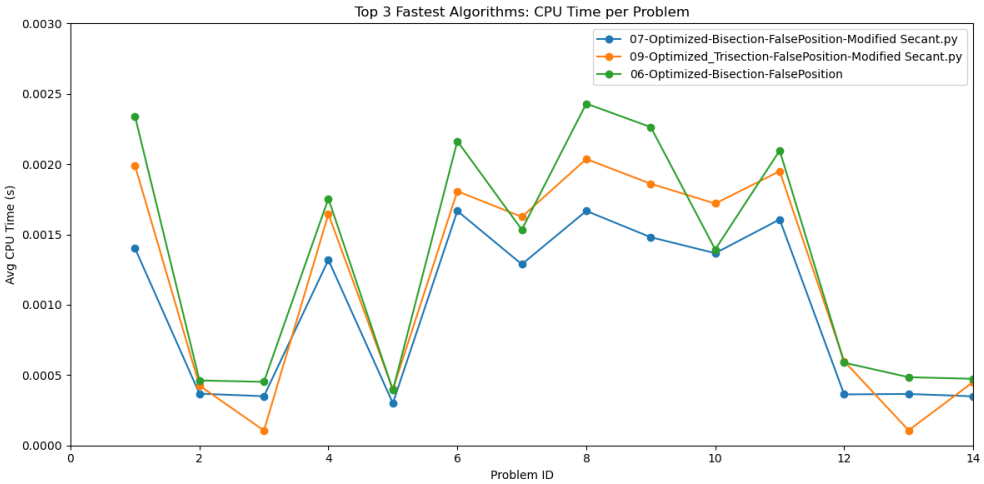

FIG. 5. CPU time per problem for the top 3 fastest algorithms.

Figure 5 compares the CPU time per problem for the top three fastest algorithms, highlighting their performance across the test suite.

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.

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.

Future work may explore adaptive hybrid algorithms, multi-dimensional extensions, and integration with machine learning for parameter optimization, as well as real-world applications in science and engineering.

References

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

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

[3] Badr, E., & 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., & Aggarwal, S. (2023). A New Wave of Hybrid Algorithms. *International Journal of Computer Applications*, 181(1), 1-10.

[5] Sabharwal, C. L., & 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., & El-Sayed, M. A. (2021). A Comparative Study among New Hybrid Root Finding Algorithms and Traditional Methods. *Mathematics*, 9(4), 345.

[7] Thota, S., & 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., & Faires, J. D. (1985). *Numerical Analysis* (3rd ed.). Prindle, Weber & Schmidt.

[10] Harder, D. W. (2019). Numerical Analysis for Engineering. Available online: https://ece.uwaterloo.ca/ dwharder/NumericalAnalysis/10RootFinding/falseposition/

[11] Mathews, J. H., & 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., & 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] Khirallah, M. Q., & Hafiz, M. A. (2013). Solving system of nonlinear equations using family of jarratt methods. *International Journal of Differential Equations and Applications*, 12, 69-83.

[15] Remani, C. (2012). *Numerical Methods for Solving Systems of Nonlinear Equations*. Lakehead University: Thunder Bay, ON, Canada, p. 13.

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

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

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

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

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

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

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

[23] Heydari, M., Hosseini, S.M., & 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.

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

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

[26] Nayak, T., & Dash, T. (2012). Solution to quadratic equation using genetic algorithm. In Proceedings of the National Conference on AIRES-2012, Vishakhapatnam, India, 29–30 June 2012.