



Article

A Comparative Study among New Hybrid Root Finding Algorithms and Traditional Methods

Elsayed Badr ^{1,2,*} , Sultan Almotairi ^{3,*}  and Abdallah El Ghamry ⁴

- ¹ Scientific Computing Department, Faculty of Computers & Artificial Intelligence, Benha University, Benha 13518, Egypt
- ² Higher Technological Institute, 10th of Ramadan City, Embassies District, Nasr City, Cairo 11765, Egypt
- ³ Department of Natural and Applied Sciences, Community College Majmaah University, Al-Majmaah 11952, Saudi Arabia
- ⁴ Computer Science Department, Faculty of Computers & Artificial Intelligence, Benha University, Benha 13518, Egypt; abdallah17163@fci.bu.edu.eg
- * Correspondence: badrgraph@gmail.com (E.B.); almotairi@mu.edu.sa (S.A.)

Abstract: In this paper, we propose a novel blended algorithm that has the advantages of the trisection method and the false position method. Numerical results indicate that the proposed algorithm outperforms the secant, the trisection, the Newton–Raphson, the bisection and the regula falsi methods, as well as the hybrid of the last two methods proposed by Sabharwal, with regard to the number of iterations and the average running time.

Keywords: hybrid method; trisection; bisection; false position; Newton–Raphson; secant; dynamical systems



Citation: Badr, E.; Almotairi, S.; Ghamry, A.E. A Comparative Study among New Hybrid Root Finding Algorithms and Traditional Methods. *Mathematics* **2021**, *9*, 1306. <https://doi.org/10.3390/math9111306>

Academic Editor: Ioannis Dassios

Received: 29 April 2021

Accepted: 3 June 2021

Published: 7 June 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

There are many sciences (mathematics, computer science, dynamical systems in engineering, agriculture, biomedical, etc.) that require finding the roots of non-linear equations. When there is not an analytic solution, we try to determine a numerical solution. There is not a specific algorithm for solving every non-linear equation efficiently.

There are several pure methods for solving such problems, including the pure, metaheuristic and blended methods. Pure methods include classical techniques such as the bisection method, the false position method, the secant method and the Newton–Raphson method, etc. Metaheuristic methods use metaheuristic algorithms such as particle swarm optimization, firefly, and ant colony for root finding, whereas blended methods are hybrid combinations of two classical methods.

There is not a specific method for solving every non-linear equation efficiently. In general, we can see more details about classical methods in [1–4] and especially for the bisection and Newton–Raphson methods in [5–8]. Other problems such as minimization, target shooting, etc. are discussed in [9–14].

Sabharwal [15] proposed a novel blended method that is a dynamic hybrid of the bisection and false position methods. He deduced that his algorithm outperformed pure methods (bisection and false position). On the other hand, he observed that his algorithm outperformed the secant method and the Newton–Raphson method according to the number of iterations. Sabharwal did not analyze his algorithm according to the running time, but he was satisfied with the iterations number only. Perhaps there is a method that has a small number of iterations, but the execution time is large and vice versa. For this reason, the iteration number and the running time are important metrics to evaluate the algorithms. Unfortunately, most researchers have not paid attention to the details of finding the running time. Furthermore, they did not discuss and did not answer the following question: why does the running time change from one run to another on the used software package?

The genetic algorithm was used to compare among the classical methods [9–11] based on the fitness ratio of the equations. The authors deduced that the genetic algorithm is more efficient than the classical algorithms for solving the functions $x^2 - x - 2$ [12] and $x^2 + 2x - 7$ [11]. Mansouri et al. [12] presented a new iterative method to determine the fixed point of a nonlinear function. Therefore, they combined ideas proposed in the artificial bee colony algorithm [13] and the bisection method [14]. They illustrate this method with four benchmark functions and compare results with others methods, such as artificial bee colony (ABC), particle swarm optimization (PSO), genetic algorithm (GA) and firefly algorithms.

For more details about the classical methods, hybrid methods and the metaheuristic approaches, the reader can refer to [16,17].

In this work, we propose a novel blended algorithm that has the advantages of the trisection method and the false position algorithm. The computational results show that the proposed algorithm outperforms the trisection and regula falsi methods. On the other hand, the introduced algorithm outperforms the bisection, Newton–Raphson and secant methods according to the iteration number and the average of running time. Finally, the implementation results show the superiority of the proposed algorithm on the blended bisection and false position algorithm, which was proposed by Sabharwal [15]. The results presented in this paper open the way for presenting new methods that compete with traditional methods and may replace them in software packages.

The rest of this work is organized as follows: The pure methods for determining the roots of non-linear equations are introduced in Section 2. The blended algorithms for finding the roots of non-linear equations are presented in Section 3. In Section 4, the numerical results analysis and statistical test among the pure methods and the blended algorithms are provided. Finally, conclusions are drawn in Section 5.

2. Pure Methods

In this section, we introduce five pure methods for finding the roots of non-linear equations. These methods are the bisection method, the trisection method, the false position method, the secant method and the Newton–Raphson method. We contribute to implementing the trisection algorithm with equal subintervals that overcomes the bisection algorithm on fifteen benchmark equations as shown in Section 3. On the other hand, the trisection algorithm also outperforms the false position method, secant method and Newton–Raphson method partially, as shown in Section 3.

2.1. Bisection Method

We assume that the function $f(x)$ is defined and continuous on the closed interval $[a, b]$, where the signals of $f(x)$ at the ends (a and b) are different. We divide the interval $[a, b]$ into two halves, where $x = \frac{a+b}{2}$, if $f(x) = 0$; then, x becomes a solution for the equation $f(x) = 0$. Otherwise, ($f(x) \neq 0$) and we can choose one subinterval $[a, x]$ or $[x, b]$ that has different signals of $f(x)$ at its ends. We repeat dividing the new subinterval into two halves until we reach the exact solution x where $f(x) = 0$ or the approximate solution $f(x) \approx 0$ with tolerance, eps . The value of eps closes to zero as shown in Algorithm 1 and other algorithms.

The size of the interval was reduced by half at each iteration. Therefore the value eps is determined from the following formula:

$$eps = \frac{b - a}{2^n} \quad (1)$$

where n is the number of iterations. From (1), the number of iterations is found by

$$n = \left\lceil \log_2 \left(\frac{b - a}{eps} \right) \right\rceil \quad (2)$$

Algorithm 1. Bisection(f, a, b, eps).

Input: The function $f(x)$,
The interval $[a, b]$ where the root lies in,
The absolute error (eps).

Output: The root (x),
The value of $f(x)$
Numbers of iterations (n),
The interval $[a, b]$ where the root lies in

```

n := 0
while true do
  n := n + 1
  x := (a + b)/2
  if |f(x)| <= eps.
    return x, f(x), n, a, b
  else if f(a) * f(x) < 0
    b := x
  else
    a := x
end (while)

```

The bisection method is a bracketing method, so it brackets the root in the interval $[a, b]$, and at each iteration, the size of the interval $[a, b]$ is halved. Accordingly, it reduces the error between the approximation root and the exact root for any iteration. On the other hand, the bisection method works quickly if the approximate root is far from the endpoint of the interval; otherwise, it needs more iterations to reach the root [17].

Advantages and Disadvantages of the Bisection Method

The bisection method is simple to implement, and its convergence is guaranteed. On the other hand, it has a relatively slow convergence, it needs different signs for the function values of the endpoints, and the test for checking this affects the complexity in the number of operations.

2.2. Trisection Method

The trisection method is like the bisection method, except that it divides the interval $[a, b]$ into three subintervals, while the bisection method divides the interval $[a, b]$ into two partial periods. Algorithm 2 divides the interval $[a, b]$ into three equal subintervals and searches for the root in the subinterval that contains different signs of the function values at the endpoints of this subinterval.

If the condition of termination is true, then the iteration has finished its task; otherwise, the algorithm repeats the calculations.

In order to divide the interval $[a, b]$ into equal three parts by x_1 and x_2 , we need to know the locations of x_1 and x_2 as the following:

As shown in Figure 1, since

$$x_1 - a = b - x_2 \quad (3)$$

$$x_2 - x_1 = x_1 - a \quad (4)$$

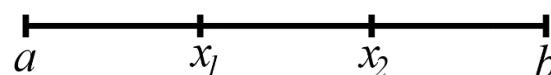


Figure 1. How to divide the interval $[a, b]$ into three subintervals.

By solving Equations (3) and (4),

We get

$$x_1 = \frac{2a + b}{3}$$

And

$$x_2 = \frac{2b + a}{3}$$

The size of the interval $[a, b]$ decreases to a third with each repetition. Therefore, the value eps is determined from the following formula:

$$eps = \frac{b - a}{3^n} \quad (5)$$

where n is the number of iterations. From (5) the number of iterations is found by

$$n = \left\lceil \log_3 \left(\frac{b - a}{eps} \right) \right\rceil \quad (6)$$

When we compare Equations (2) and (6), we conclude that the iterations number of the trisection algorithm is less than the iterations number of the bisection algorithm. We might think that the trisection algorithm is better than the bisection algorithm since it requires a few iterations. However, it might be the case that one iteration of the trisection algorithm has an execution time greater than the execution time of one iteration of the bisection algorithm. Therefore, we will consider both execution time and the number of iterations to evaluate the different algorithms.

Algorithm 2. Trisection(f, a, b, eps).

Input: The function $f(x)$,
 The interval $[a, b]$ where the root lies in,
 The absolute error (eps).

Output: The root (x),
 The value of $f(x)$
 Numbers of iterations (n),
 The interval $[a, b]$ where the root lies in

```

n := 0
while true do
  n := n + 1
  x1 := (b + 2*a)/3
  x2 := (2*b + a)/3
  if |f(x1)| < |f(x2)|
    x := x1
  else
    x := x2
  if |f(x)| <= eps
    return x, f(x), n, a, b
  else if f(a) * f(x1) < 0
    b := x1
  else if f(x1) * f(x2) < 0
    a := x1
    b := x2
  else
    a := x2
end (while)

```

Advantages and Disadvantages of the Trisection Method

The trisection method has the same advantages and disadvantages of the bisection method, in addition to being faster than it, as shown in Tables 1–9.

2.3. False Position (Regula Falsi) Method

There is no unique method suitable for finding the roots of all nonlinear functions. Each method has advantages and disadvantages. Hence, the false position method is a

dynamic and fast method when the nature of the function is linear. The function $f(x)$, whose roots are in the interval $[a, b]$ must be continuous, and the values of $f(x)$ at the endpoints of the interval $[a, b]$ have different signs. The false position method uses two endpoints of the interval $[a, b]$ with initial values ($r_0 = a, r_1 = b$). The connecting line between the two points $(r_0, f(r_0))$ and $(r_1, f(r_1))$ intersects the x -axis at the next estimate, r_2 . Now, we can determine the successive estimates, r_n from the following relationship

$$r_n = r_{n-1} - \frac{f(r_{n-1})(r_{n-1} - r_{n-2})}{f(r_{n-1}) - f(r_{n-2})} \quad (7)$$

for $n \geq 2$.

Remark: The regula falsi method is very similar to the bisection method. However, the next iteration point is not the midpoint of the interval but the intersection of the x -axis with a secant through $(a, f(a))$ and $(b, f(b))$.

Algorithm 3 uses the relation (7) to get the successive approximations by the false position method.

Algorithm 3. False Position(f, a, b, eps).

Input: The function (f),
The interval $[a, b]$ where the root lies in,
The absolute error (eps).

Output: The root (x),
The value of $f(x)$
Numbers of iterations (n),
The interval $[a, b]$ where the root lies in

```

n := 0
while true do
  n := n + 1;
  x = a - (f(a)*(b - a))/(f(b) - f(a))
  if |f(x)| <= eps
    return x, f(x), n, a, b
  else if f(a) * f(x) < 0
    b := x
  else
    a := x
end (while)

```

Advantages and Disadvantages of the Regula Falsi Method

It is guaranteed to converge, and it is fast when the function is linear. On the other hand, we cannot determine the iterations number needed for convergence. It is very slow when the function is not linear.

2.4. Newton–Raphson Method

This method depends on a chosen initial point x_0 . This point plays an important role for Newton–Raphson method. The success of the method depends mainly on the point x_0 , and then the method may converge to its root or diverge based on the choice of the point x_0 . Therefore, the first estimate can be determined from the following relation.

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \quad (8)$$

The successive approximations for the Newton–Raphson method can be found from the following relation:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \quad (9)$$

such that the $f'(x_i)$ is the first derivative of the function $f(x)$ at the point x_i .

Algorithm 4 uses the relation (9) to get the successive approximations by the Newton–Raphson method.

Algorithm 4. Newton(f, x_i, eps).

This function implements Newton’s method.

Input: The function (f),
An initial root x_i ,
The absolute error (eps).

Output: The root (x),
The value of $f(x)$
Numbers of iterations (n),

$g(x) := f'(x)$

$n = 0$

while true do

$n := n + 1$

$x_i = x_i - f(x_i)/g(x_i)$

if $|f(x)| \leq eps$

return $x_i, f(x_i), n$

end (while)

Advantages and Disadvantages of the Newton–Raphson Method

It is very fast compared to other methods, but it sometimes fails, meaning that there is no guarantee of its convergence.

2.5. Secant Method

Just as there is the possibility of the Newton method failing, there is also the possibility that the secant method will fail. The Newton method uses the relation (9) to find the successive approximations, but the secant method uses the following relation:

$$x_{i+1} = x_i - \frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})} f(x_i) \quad (10)$$

Algorithm 5 uses the relation (10) to get the successive approximations by the secant method.

Algorithm 5. Secant(f, a, b, eps).

This function implements the Secant method.

Input: The function (f),
Two initial roots: a and b ,
The absolute error (eps).

Output: The root (x),
The value of $f(x)$
Numbers of iterations (n),

$n := 0$

while true do

$n := n + 1$

$x := b - f(b)*(b - a)/(f(b) - f(a))$

if $|f(x)| \leq eps$

return $x, f(x), n$

$a := b$

$b := x$

end (while)

Advantages and Disadvantages of the Secant Method

It is very fast compared to other methods, but it sometimes fails, meaning that there is no guarantee of its convergence.

3. Hybrid Algorithms

In this section, instead of pure methods such as the bisection method, the trisection method, the false position method, the secant method and the Newton–Raphson method, we propose a new hybrid root-finding algorithm (trisection–false position), which outperforms the algorithm (bisection–false position) that was proposed by Sabharwal [15].

3.1. Blended Bisection and False Position

Sabharwal [15] proposed a new algorithm that has the advantages of both the bisection and the false position methods. He built a novel hybrid method, Algorithm 6, which overcame the pure methods (bisection and false position).

Algorithm 6. `blendBF(f, a, b, eps)`.

This function implements the blended method of bisection and false position methods.

```

Input: The function  $f$ ,
          The interval  $[a, b]$  where the root lies in,
          The absolute error ( $eps$ ).
Output: The root  $x$ , The value of  $f(x)$ ; Numbers of iterations ( $n$ ),
          The interval  $[a, b]$  where the root lies in
 $n := 0$ 
 $a1 := a$ 
 $a2 := a$ 
 $b1 := b$ 
 $b2 := b$ 
while true do
   $n := n + 1$ 
   $xB := (a + b)/2$ 
   $xF := a - (f(a)*(b - a))/(f(b) - f(a))$ 
  if  $|f(xB)| < |f(xF)|$ 
     $x := xB$ 
  else
     $x := xF$ 
  if  $|f(x)| \leq eps$ 
    return  $x, f(x), n, a, b$ 
  if  $f(a)*f(xB) < 0$ 
     $b1 := xB$ 
  else
     $a1 := xB$ 
  if  $f(a) * f(xF) < 0$ 
     $b2 := xF$ 
  else
     $a2 := xF$ 
   $a := \max(a1, a2);$ 
   $b := \min(b1, b2)$ 
end (while)

```

Advantages and Disadvantages of the Blended Algorithm

It is guaranteed to converge, and it is efficient more than the classical methods but it sometimes takes a long time to get root.

3.2. Blended Trisection and False Position

We exploit the superiority of the trisection over the bisection method (as shown in Section 4) in order to present a new hybrid method (Algorithm 7) that overcomes the hybrid method presented by Sabharwal [15]. The blended method (trisection–false position) is based on calculating the segment line point in the false position method and also calculating two points that divide the interval $[a, b]$ in the trisection method and then choosing the best of them, which converges to the approximating root. The number of iterations $n(eps)$ of

the proposed hybrid method is less than or equal to $\min\{n_f(eps), n_t(eps)\}$, where $n_f(eps)$ and $n_t(eps)$ are the number of iterations of the false position method and the trisection method, respectively. Algorithm 7 outperforms all the classical methods (Tables 1–9).

Algorithm 7. blendTF(f, a, b, eps).

This function implements the blended method of trisection and false position methods.

Input: The function (f); The interval $[a, b]$ where the root lies in,
The absolute error (eps).

Output: The root (x), The value of $f(x)$, Numbers of iterations (n),
The interval $[a, b]$ where the root lies in

$n := 0$; $a1 := a$; $a2 := a$; $b1 := b$, $b2 := b$

while true do

$n := n + 1$

$xT1 := (b + 2*a)/3$

$xT2 := (2*b + a)/3$

$xF := a - (f(a)*(b - a))/(f(b) - f(a))$

$x := xT1$

$fx := fxT1$

if $|f(xT2)| < |f(x)|$

$x := xT2$

if $|f(xF)| < |f(x)|$

$x := xF$

if $|f(x)| \leq eps$

return $x, f(x), n, a, b$

if $fa * f(xT1) < 0$

$b1 := xT1$

else if $f(xT1) * f(xT2) < 0$

$a1 := xT1$

$b1 := xT2$

else

$a1 := xT2$

if $fa * f(xF) < 0$

$b2 := xF$;

else

$a2 := xF$;

$a := \max(a1, a2)$; $b := \min(b1, b2)$

end (while)

Advantages and Disadvantages of the Blended Algorithm (The Proposed Algorithm)

It is guaranteed to converge, and it is more efficient than the classical methods and the blended algorithm that was proposed in [15], as shown in Tables 1–9.

4. Computational Study

The numerical results of the pure methods bisection method, trisection method, false position method, secant method and Newton–Raphson method are proposed. In addition to the computational results for the hybrid methods, the bisection–false position and trisection–false position are proposed. We compare the pure method and the hybrid method with the proposed hybrid method according to the number of iterations and CPU time. We used fifteen benchmark problems for this comparison, as shown in Table 1. We ran each problem ten times, and then we computed the average of CPU time and the number of iterations.

Table 1. Fifteen benchmark problems.

No.	Problem	Intervals	References
P1	$x^2 - 3$	[1, 2]	Harder [18]
P2	$x^2 - 5$	[2, 7]	Srivastava [9]
P3	$x^2 - 10$	[3, 4]	Harder [18]
P4	$x^2 - x - 2$	[1, 4]	Moazzam [10]
P5	$x^2 + 2x - 7$	[1, 3]	Nayak [11]
P6	$x^3 - 2$	[0, 2]	Harder [18]
P7	$xe^x - 7$	[0, 2]	Callhoun [19]
P8	$x - \cos(x)$	[0, 1]	Ehiwario [6]
P9	$x \sin(x) - 1$	[0, 2]	Mathews [20]
P10	$x \cos(x) + 1$	[-2, 4]	Esfandiari [21]
P11	$x^{10} - 1$	[0, 1.3]	Chapra [17]
P12	$x^2 + e^{x/2} - 5$	[1, 2]	Esfandiari [21]
P13	$\sin(x)\sinh(x) + 1$	[3, 4]	Esfandiari [21]
P14	$e^x - 3x - 2$	[2, 3]	Hoffman [22]
P15	$\sin(x) - x^2$	[0.5, 1]	Chapra [17]

Table 2. Comparison among pure methods and blended algorithms according to iterations, AppRoot, error and interval bounds.

Method	Iter.	AppRoot	Error	LowerB	UpperB
Bisection	19	2.0000019073486328	0.0000057220495364	1.9999961853027344	2.0000076293945313
Trisection	1	2.0000000000000000	0.0000000000000000	1.0000000000000000	4.0000000000000000
FalsePosition	15	1.9999983893881288	0.0000048318330195	1.9999959734735644	4.0000000000000000
Secant	6	2.0000000786432022	0.0000002359296127	na	na
NewtonRaphson	5	2.0000000006984919	0.0001373332926100	na	na
Hybrid [15]	2	2.0000000000000000	0.0000000000000000	1.5000000000000000	2.5000000000000000
OurHybrid	1	2.0000000000000000	0.0000000000000000	1.0000000000000000	4.0000000000000000

Table 3. Solutions of fifteen problems by the bisection method.

Problem	Bisection Method					
	Iter	Average CPU Time	Approximate Root	Function Value	Lower Bound	Upper Bound
P1	44	0.514839	1.7320508075688963	0.0000000000000000	1.7320508075688394	1.7320508075689531
P2	44	0.339006	2.2360679774997720	0.0000000000000000	2.2360679774994878	2.2360679775000563
P3	44	0.330300	3.1622776601683995	0.0000000000000000	3.1622776601683427	3.1622776601684564
P4	45	0.339274	2.0000000000000284	0.0000000000000000	1.9999999999999432	2.0000000000001137
P5	48	0.413062	1.8284271247461916	0.0000000000000086	1.8284271247461845	1.8284271247461987
P6	49	0.373710	1.2599210498948743	0.0000000000000054	1.2599210498948707	1.2599210498948779
P7	46	0.381111	1.5243452049841437	-0.0000000000000075	1.5243452049841153	1.5243452049841721
P8	44	0.345850	0.7390851332151556	-0.0000000000000085	0.7390851332150987	0.7390851332152124
P9	46	0.556300	1.1141571408719244	-0.0000000000000079	1.1141571408718960	1.1141571408719528
P10	45	0.454494	2.0739328090912181	-0.0000000000000074	2.0739328090910476	2.0739328090913887
P11	44	0.338134	1.0000000000000058	0.0000000000000000	0.9999999999999318	1.0000000000000795
P12	48	0.379392	1.6490132683031895	-0.0000000000000028	1.6490132683031860	1.6490132683031931
P13	48	0.390438	3.2215883990939425	-0.0000000000000056	3.2215883990939389	3.2215883990939460
P14	46	0.354950	2.1253911988111298	-0.0000000000000007	2.1253911988111156	2.1253911988111440
P15	45	0.359546	0.8767262153950668	-0.0000000000000048	0.8767262153950526	0.8767262153950810

Table 4. Solutions of fifteen problems by the trisection method.

Problem	Trisection Method					
	Iter	Average CPU Time	Approximate Root	Function Value	Lower Bound	Upper Bound
P1	26	0.292349	1.7320508075688856	0.0000000000000000	1.7320508075680989	1.7320508075692791
P2	28	0.311319	2.2360679774997863	0.0000000000000000	2.2360679774993493	2.2360679775000047
P3	28	0.312939	3.1622776601683911	0.0000000000000000	3.1622776601683040	3.1622776601684350
P4	1	0.011161	2.0000000000000000	0.0000000000000000	1.0000000000000000	4.0000000000000000
P5	29	0.330426	1.8284271247461907	0.0000000000000036	1.8284271247461616	1.8284271247462491
P6	30	0.341553	1.2599210498948719	−0.0000000000000062	1.2599210498948623	1.2599210498948914
P7	31	0.349806	1.5243452049841439	−0.0000000000000049	1.5243452049841375	1.5243452049841473
P8	29	0.326833	0.7390851332151560	−0.0000000000000078	0.7390851332151415	0.7390851332151852
P9	28	0.773690	1.1141571408719348	0.0000000000000066	1.1141571408717601	1.1141571408720223
P10	28	0.316154	2.0739328090912146	0.0000000000000007	2.0739328090906901	2.0739328090914770
P11	26	0.297432	1.00000000000000393	0.0000000000000000	0.999999999995278	1.000000000010620
P12	26	0.299995	1.6490132683031904	0.0000000000000012	1.6490132683024035	1.6490132683035839
P13	31	0.360716	3.2215883990939425	−0.0000000000000056	3.2215883990939407	3.2215883990939456
P14	28	0.323873	2.1253911988111311	0.0000000000000065	2.1253911988110437	2.1253911988111747
P15	29	0.334640	0.8767262153950647	−0.0000000000000025	0.8767262153950502	0.8767262153950720

Table 5. Solutions of fifteen problems by the false position method.

Problem	False Position Method					
	Iter	Average CPU Time	Approximate Root	Function Value	Lower Bound	Upper Bound
P1	12	0.134719	1.7320508075688599	0.0000000000000000	1.7320508075686347	2.0000000000000000
P2	46	0.510051	2.2360679774997747	0.0000000000000000	2.2360679774997609	7.0000000000000000
P3	14	0.155169	3.1622776601683644	0.0000000000000000	3.1622776601682516	4.0000000000000000
P4	34	0.382718	1.999999999999558	0.0000000000000000	1.999999999998894	4.0000000000000000
P5	20	0.556411	1.8284271247461896	−0.0000000000000027	1.8284271247461874	3.0000000000000000
P6	40	0.455044	1.2599210498948719	−0.0000000000000062	1.2599210498948701	2.0000000000000000
P7	29	0.330403	1.5243452049841437	−0.0000000000000075	1.5243452049841419	2.0000000000000000
P8	11	0.124456	0.7390851332151551	−0.0000000000000092	0.7390851332150500	1.0000000000000000
P9	6	0.078088	1.1141571408719306	0.0000000000000008	1.0997501702946164	1.1141571408730828
P10	12	0.138026	2.0739328090912146	0.0000000000000007	2.0739328090912039	2.5157197710146586
P11	127	1.447224	0.999999999999812	0.0000000000000000	0.999999999999755	1.3000000000000000
P12	15	0.176429	1.6490132683031899	−0.0000000000000008	1.6490132683031871	2.0000000000000000
P13	44	0.507856	3.2215883990939416	0.0000000000000063	3.2215883990939407	4.0000000000000000
P14	44	0.504918	2.1253911988111285	−0.0000000000000079	2.1253911988111267	3.0000000000000000
P15	16	0.185620	0.8767262153950552	0.0000000000000080	0.8767262153950091	1.0000000000000000

Table 6. Solutions of fifteen problems by Newton’s method.

Problem	Newton’s Method			
	Iter	Average CPU Time	Approximate Root	Function Value
P1	6	0.240500	1.7320508075688774	0.0000000000000000
P2	5	0.186819	2.2360679774997898	0.0000000000000000
P3	5	0.185136	3.1622776601683795	0.0000000000000000
P4	7	0.244393	2.0000000000000000	0.0000000000000000
P5	6	0.214349	1.8284271247461901	−0.0000000000000002
P6			Fail	
P7	14	0.436972	1.5243452049841444	0.0000000000000002
P8	6	0.214512	0.7390851332151607	0.0000000000000001
P9			Fail	
P10	14	0.439575	−4.9171859252871322	0.0000000000000011
P11			Fail	
P12	6	0.221387	1.6490132683031902	0.0000000000000002
P13	6	0.221157	3.2215883990939420	0.0000000000000004
P14	5	0.191465	2.1253911988111298	0.0000000000000017
P15	9	0.306000	0.8767262153950625	0.0000000000000000

Table 7. Solutions of fifteen problems by the secant method.

Problem	Secant Method			
	Iter	Average CPU Time	Approximate Root	Function Value
P1	6	0.068071	1.7320508075688772	0.0000000000000000
P2	7	0.077763	2.2360679774997898	0.0000000000000000
P3	5	0.055822	3.1622776601683764	0.0000000000000000
P4	8	0.089142	2.0000000000000000	0.0000000000000000
P5	6	0.066767	1.8284271247461907	0.0000000000000036
P6	10	0.114367	1.2599210498948716	−0.0000000000000073
P7	9	0.101345	1.5243452049841444	0.0000000000000002
P8	6	0.067514	0.7390851332151607	0.0000000000000001
P9	5	0.073486	1.1141571408719304	0.0000000000000004
P10	8	0.091206	2.0739328090912150	−0.0000000000000003
P11			Fail	
P12	6	0.069605	1.6490132683031902	0.0000000000000002
P13	8	0.093442	3.2215883990939420	0.0000000000000004
P14	7	0.081160	2.1253911988111298	−0.0000000000000007
P15	7	0.080784	0.8767262153950625	−0.0000000000000000

Table 8. Solutions of fifteen problems by the hybrid method bisection-false position.

Problem	Bisection-False Position Method					
	Iter	Average CPU Time	Approximate Root	Function Value	Lower Bound	Upper Bound
P1	8	0.121232	1.7320508075688772	0.0000000000000000	1.7320508075688001	1.7350578402209837
P2	10	0.148720	2.2360679774997898	0.0000000000000000	2.2360679774993639	2.2439291539836148
P3	7	0.103442	3.1622776601683702	0.0000000000000000	3.1622776601625873	3.1721597778622157
P4	2	0.030053	2.0000000000000000	0.0000000000000000	1.5000000000000000	2.5000000000000000
P5	5	0.074941	1.8284271247461901	−0.0000000000000002	1.8284271247430004	1.8284271247493797
P6	9	0.137275	1.2599210498948723	−0.0000000000000041	1.2599210498939839	1.2611286403176987
P7	11	0.165907	1.5243452049841444	0.0000000000000002	1.5243452049841386	1.5260333371087631
P8	8	0.120322	0.7390851332151607	0.0000000000000001	0.7390851332151470	0.7422270732175922
P9	6	0.101488	1.1141571408719302	0.0000000000000001	1.1132427327642707	1.1141571408719768
P10	10	0.150611	2.0739328090912150	−0.0000000000000003	2.0739328090911866	2.0789350033373930
P11	12	0.184145	0.9999999999999999	0.0000000000000000	0.9999999999999905	1.0003433632829859
P12	8	0.123939	1.6490132683031895	−0.0000000000000028	1.6490132683026435	1.6531557562694839
P13	9	0.139654	3.2215883990939420	0.0000000000000004	3.2215883990939242	3.2224168881395068
P14	9	0.139775	2.1253911988111289	−0.0000000000000055	2.1253911988104042	2.1275191334463157
P15	7	0.107493	0.8767262153950581	0.0000000000000048	0.8767262153886712	0.8772684454348731

Table 9. Solutions of fifteen problems by the hybrid method trisection-false position.

Problem	Trisection-False Position Method					
	Iter	Average CPU Time	Approximate Root	Function Value	Lower Bound	Upper Bound
P1	7	0.131418	1.7320508075688772	0.0000000000000000	1.7320508075687824	1.7324926951584967
P2	8	0.149270	2.2360679774997894	0.0000000000000000	2.2360679774987138	2.2373661277171197
P3	6	0.111231	3.1622776601683777	0.0000000000000000	3.1622776601623102	3.1638711488008444
P4	1	0.018535	2.0000000000000000	0.0000000000000000	1.0000000000000000	4.0000000000000000
P5	7	0.131906	1.8284271247461901	−0.0000000000000002	1.8284271247461521	1.8288267084339278
P6	8	0.152130	1.2599210498948730	−0.0000000000000009	1.2599210498938187	1.2602675857311345
P7	7	0.131670	1.5243452049841444	0.0000000000000002	1.5243452049840662	1.5244112793655715
P8	7	0.131345	0.7390851332151607	0.0000000000000001	0.7390851332151193	0.7396432352779715
P9	5	0.213409	1.1141571408719326	0.0000000000000035	1.1126440519145675	1.1141571409109841
P10	8	0.150378	2.0739328090912150	−0.0000000000000003	2.0739328090912079	2.0745363211703700
P11	9	0.170638	1.0000000000000000	0.0000000000000000	0.9999999999999090	1.0000567349034972
P12	6	0.115872	1.6490132683031897	−0.0000000000000018	1.6490132683015255	1.6496393349802922
P13	7	0.135481	3.2215883990939420	0.0000000000000004	3.2215883990931498	3.2217303732361522
P14	7	0.134990	2.1253911988111298	−0.0000000000000007	2.1253911988110636	2.1254846670968397
P15	5	0.096275	0.8767262153950616	0.0000000000000010	0.8767262151142412	0.8767286917958327

We used MATLAB v7.01 Software Package to implement all the codes. All codes were run under 64-bit Window 8.1 Operating System with Core(TM)i5 CPU M 460 @2.53GHz, 4.00 GB of memory.

Dataset and Evaluation Metrics

There are different ways to terminate the numerical algorithms such as the absolute error (eps) and the number of iterations. In this paper, we used the absolute error ($\text{eps} = 10^{-14}$) to terminate all the algorithms. Perhaps there is a method that has a small number of iterations, but the execution time is large and vice versa. For this reason, the iteration number and the running time are important metrics to evaluate the algorithms. Unfortunately, most researchers did not pay attention to the details of finding the running time. Furthermore, they did not discuss and did not answer the following question: why does the running time change from one run to another with the used software package? Therefore, we ran every algorithm ten times and calculated the average of the running time to obtain an accurate running time and avoid the problems of the operating systems.

In Table 2, the abbreviations AppRoot, Error, LowerB and UpperB are used to denote the approximation root, the difference between two successive roots, lower bound and upper bound, respectively. Table 2 shows the performance of all classical methods and blended algorithms for solving the Problem 4. It is clear that both the trisection and the proposed blended algorithm (trisection-false position) outperformed the other algorithms. Because it is not accurate enough to make a conclusion from one function, we used fifteen benchmark functions (Table 1) to evaluate the proposed algorithm.

Ali Demir [23] proved that the trisection method with k -Lucas number works faster than the bisection method. From Tables 3 and 4 and Figure 2, it is clear that the trisection method is better than the bisection method with respect to the running time for all problems except for problem 9. On the other hand, the trisection method determined the exact root (2.0000000000000000) of problem 4 after one iteration, but the bisection method found the approximate root (2.0000000000000284) after 45 iterations. Figure 3 shows that the trisection method always has fewer iterations than the bisection method. We can determine the number of iterations for the trisection method by $n = \left\lceil \log_3 \left(\frac{b-a}{\text{eps}} \right) \right\rceil$ and the number of iterations for the bisection method by $n = \left\lceil \log_2 \left(\frac{b-a}{\text{eps}} \right) \right\rceil$. The authors [6,11] explained that the secant method is better than the bisection and Newton–Raphson methods for problem 8. It is not accurate to draw a conclusion from one function [15], so we experimented on fifteen benchmark functions. From Table 7, it is clear that the secant method failed to solve problem 11.

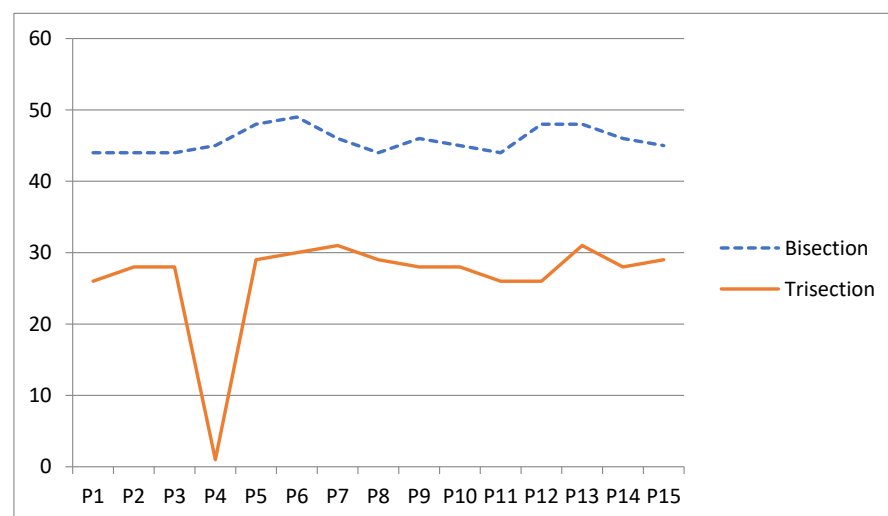


Figure 2. A comparison among 7 methods on 15 problems according to the number of iteration.

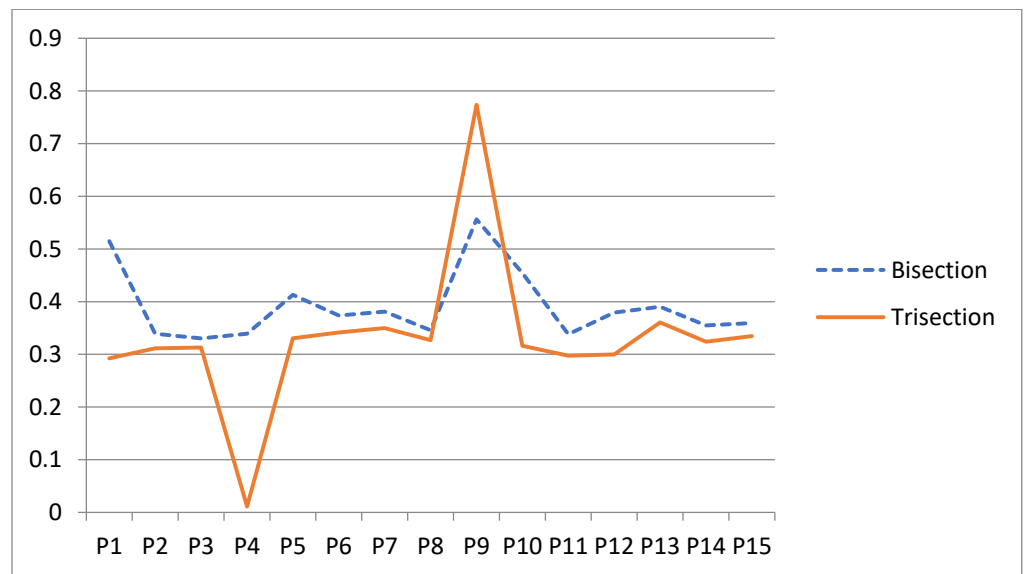


Figure 3. A comparison among 7 methods on 15 problems according to the CPU time.

From Tables 5–7, we deduce that the proposed hybrid algorithm (trisection-false position) is better than the Newton–Raphson, false-position and secant. The Newton–Raphson method failed to solve problems P6, P9 and P11, and the secant method failed to solve P11.

From Figure 4 and Tables 8 and 9, it is clear that the proposed blended algorithm (trisection–false position) has fewer iterations than the blended algorithm (bisection–false position) [15] on all the problems except problem 5 (i.e., according to the number of iterations, the proposed algorithm achieved 93.3% of fifteen problems but Sabharwal’s algorithm achieved 6.6%).

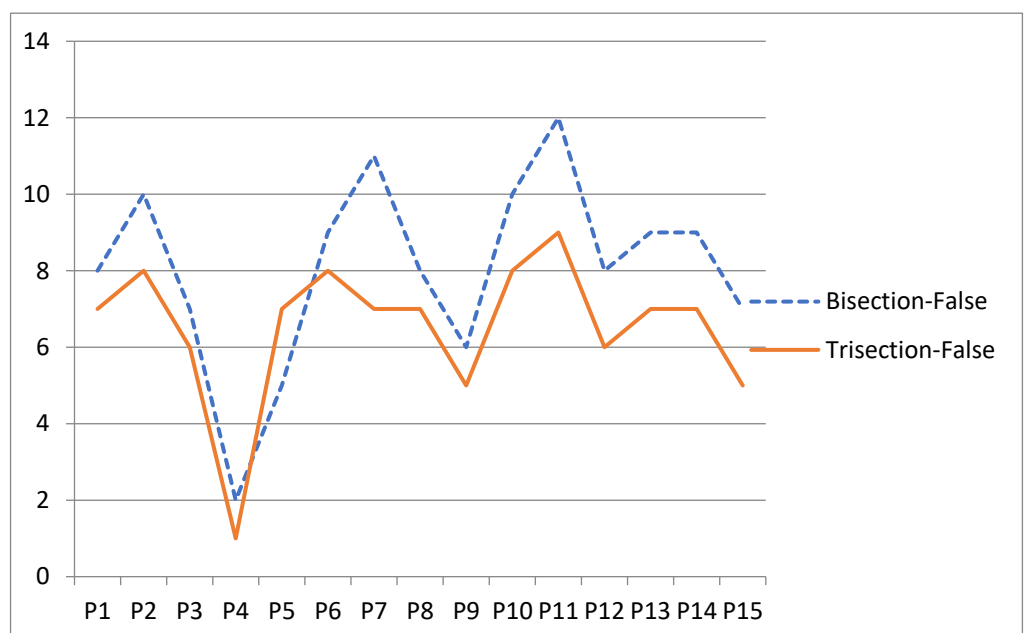


Figure 4. A comparison among 7 methods on 15 problems according to the number of iterations.

From Figure 5 and Tables 8 and 9, it is clear that the proposed blended algorithm (trisection–false position) outperforms the blended algorithm (bisection–false position) [15] for eight problems versus seven problems (i.e., the proposed algorithm achieved 53.3%

of fifteen problems but Sabharwal's algorithm achieved 46.6%). On the other hand, the trisection method determined the exact root (1.0000000000000000) of the problem 4 after nine iterations, but the bisection method found the approximate root (0.9999999999999999) after 12 iterations.

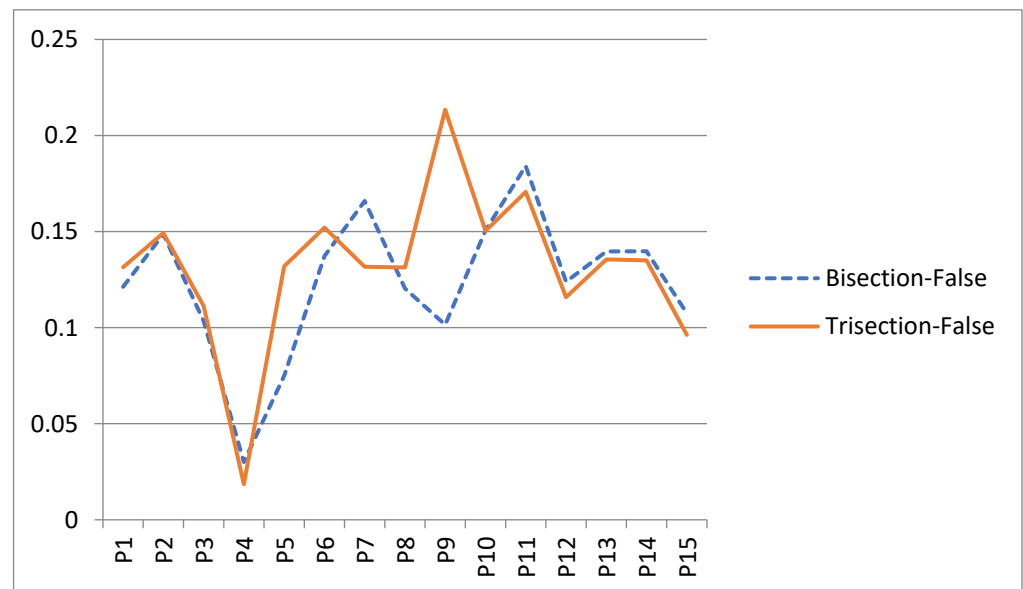


Figure 5. A comparison among 7 methods on 15 problems according to the CPU time.

5. Conclusions

In this work, we proposed a novel blended algorithm that has the advantages of the trisection method and the false position method. The computational results show that the proposed algorithm outperforms the trisection and regula falsi methods. On the other hand, the introduced algorithm outperforms the bisection, Newton–Raphson and secant methods according to the iteration number and the average running time. Finally, the implementation results show the superiority of the proposed algorithm on the blended bisection and false position algorithm, which was proposed by Sabharwal [15]. In future work, we will do more numerical studies using benchmark functions to evaluate the proposed algorithm and ensure that it competes with the traditional algorithms to replace it in software packages such as Matlab and Python. We will also propose some other hybrid algorithms that may be better than the proposed algorithm such as the bisection–Newton–Raphson method and trisection–Newton–Raphson.

Author Contributions: Conceptualization, E.B.; methodology, S.A.; software, A.E.G.; validation, E.B.; formal analysis, E.B.; investigation, S.A.; resources, A.E.G.; data curation, A.E.G.; writing—original draft preparation, E.B.; writing—review and editing, E.B.; visualization, E.B.; supervision, E.B.; project administration, E.B.; funding acquisition, S.A. All authors have read and agreed to the published version of the manuscript.

Funding: The authors extend their appreciation to the Deanship of Scientific Research at Majmaah University for funding this work under project number (R-2021-140).

Acknowledgments: The help from Higher Technological Institute, 10th of Ramadan City, Egypt for publishing is sincerely and greatly appreciated. We also thank the referees for suggestions to improve the presentation of this paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Hasan, A. Numerical Study of Some Iterative Methods for Solving Nonlinear Equations. *Int. J. Eng. Sci. Invent.* **2016**, *5*, 1–10.
2. Hasan, A.; Ahmad, N. Comparative study of a new iterative method with that Newtons Method for solving algebraic and transcensental equations. *Int. J. Comput. Math. Sci.* **2015**, *4*, 32–37.
3. Khirallah, M.Q.; Hafiz, M.A. Solving system of nonlinear equations using family of jarratt methods. *Int. J. Differ. Equ. Appl.* **2013**, *12*, 69–83. [[CrossRef](#)]
4. Remani, C. *Numerical Methods for Solving Systems of Nonlinear Equations*; Lakehead University: Thunder Bay, ON, Canada, 2012; p. 13.
5. Lally, C.H. A faster, high precision algorithm for calculating symmetric and asymmetric. *arXiv* **2015**, arXiv:1509.01831.
6. Ehiwario, J.C.; Aghamie, S.O. Comparative Study of Bisection, Newton-Raphson and Secant Methods of Root-Finding Problems. *IOSR J. Eng.* **2014**, *4*, 1–7.
7. Ait-Aoudia, S.; Mana, I. Numerical solving of geometric constraints by bisection: A distributed approach. *Int. J. Comput. Inf. Sci.* **2004**, *2*, 66.
8. Baskar, S.; Ganesh, S.S. *Introduction to Numerical Analysis*; Department of Mathematics, Indian Institute of Technology Bombay Powai: Mumbai, India, 2016.
9. Srivastava, R.B.; Srivastava, S. Comparison of numerical rate of convergence of bisection, Newton and secant methods. *J. Chem. Biol. Phys. Sci.* **2011**, *2*, 472–479.
10. Moazzam, G.; Chakraborty, A.; Bhuiyan, A. A robust method for solving transcendental equations. *Int. J. Comput. Sci. Issues* **2012**, *9*, 413–419.
11. Nayak, T.; Dash, T. Solution to quadratic equation using genetic algorithm. In Proceedings of the National Conference on AIRES-2012, Vishakhapatnam, India, 29–30 June 2012.
12. Mansouria, P.; Asadya, B.; Guptab, N. The Bisection–Artificial Bee Colony algorithm to solve fixed point problems. *Appl. Soft Comput.* **2015**, *26*, 143–148. [[CrossRef](#)]
13. Karaboga, D.; Basturk, B. A powerful and efficient algorithm for numerical func-tion optimization: Artificial Bee Colony (ABC) algorithm. *J. Glob. Optim.* **2007**, *39*, 459–471. [[CrossRef](#)]
14. Burden, L.R.; Douglas, F.J. *Numerical Analysis, Prindle, Weber & Schmidt*, 3rd ed.; Amazon: Seattle, WA, USA, 1 January 1985.
15. Sabharwal, C.L. Blended Root Finding Algorithm Outperforms Bisection and Regula Falsi Algorithms. *Mathematics* **2019**, *7*, 1118. [[CrossRef](#)]
16. Badr, E.M.; Elgendy, H. A hybrid water cycle-particle swarm optimization for solving the fuzzy underground water confined steady flow. *Indones. J. Electr. Eng. Comput. Sci.* **2020**, *19*, 492–504. [[CrossRef](#)]
17. Chapra, S.C.; Canale, R.P. *Numerical Methods for Engineers*, 7th ed.; McGraw-Hill: Boston, MA, USA, 2015.
18. Harder, D.W. Numerical Analysis for Engineering. Available online: <https://ece.uwaterloo.ca/~dwdharter/NumericalAnaly-sis/10RootFinding/falseposition/> (accessed on 11 June 2019).
19. Calhoun, D. Available online: https://math.boisestate.edu/~calhoun/teaching/matlab-tutorials/lab_16/html/lab_16.html (accessed on 13 June 2019).
20. Mathews, J.H.; Fink, K.D. *Numerical Methods Using Matlab*, 4th ed.; Prentice-Hall Inc.: Upper Saddle River, NJ, USA, 2004; ISBN 0-13-065248-2.
21. Esfandiari, R.S. *Numerical Methods for Engineers and Scientists Using MATLAB*; CRC Press: Boca Raton, FL, USA, 2013.
22. Joe, D.H. *Numerical Methods for Engineers and Scientists*, 2nd ed.; CRC Press: Boca Raton, FL, USA, 2001.
23. Demir, A. Trisection method by k-Lucas numbers. *Appl. Math. Comput.* **2008**, *198*, 339–345. [[CrossRef](#)]