



An Adaptive Algorithm for Roots of Non-Linear Equations

Chaman Lal Sabharwal

Missouri University of Science and Technology
Rolla, Missouri - 65409, USA

Abstract: Optimization problems lead to non-linear equations for determining their roots efficiently. Finding the roots of a non-linear equation is a fundamental problem in diverse fields. Numerical iterative techniques become necessary when analytic solution is not available. Recently, some attempts Adapt1 combination of Bisection and Regula Falsi; Adapt2 a blend of Trisection and Regula Falsi; and Adapt3 a composite of Trisection and Newton-Raphson methods emerged to design adaptive algorithms. In this paper, we design a new heuristic algorithm Adapt4 which is an intuitive approach. The new algorithm Adapt4 promises to be more efficient than the existing adaptive algorithms. The new adapted algorithm is a blend of BT section algorithm and Regula Falsi algorithm. The algorithm Adapt4 reduces the number of iterations and the computation time of each iteration. The implementation results validate that the new algorithm Adapt4, surpasses the efficiency of existing such algorithms Adapt1, Adapt2, and Adapt3 algorithms. This paper contributes an essential adaptive algorithm for finding roots of frequent engineering equations.

Keywords: Regula False; Newton-Raphson; Bisection; Trisection; BTsection; Adaptive Algorithm

I. Introduction

Numerical iterative techniques are applied to find roots of an equation when analytic solution is not available [1], [2], [3], [4]. Adaptive algorithm is a new concept to iterative solutions. Envisioning /Adaptive techniques to find roots of an equation is an inspiring technique to solve the fundamental problem in diverse fields. Recently, there have been some adaptive algorithms designing efficient solutions. Here we present a new adaptive algorithm which is a more intuitive and heuristic approach to adaptation.

Mostly optimization problems lead to solving non-linear equations for optimizing calculation of the value of a parameter, that is the root of the equation. We design and implement a new algorithm Adapt4 that is more efficient combination of BTsection and False Position methods. The implementation results validate that the new algorithm surpasses the existing such algorithms rooted on standalone Bisection and Trisection algorithms in conjunction with False Position and Newton Raphson algorithms. We designed and implemented an essential algorithm to the root finding algorithms.

First, why do we need another root finding algorithm? Even though the classical methods have been developed and used for decades, enhancements are steadily made to improve the performance of these methods [5]. There are several factors in determining the efficiency of an algorithm: accuracy, run-time efficiency, and generality that it works most of the time on all functions and domain intervals as expected. Accuracy may be measured in terms of number of decimal digits. For iterative methods, efficiency could mean the number of iterations to arrive at a solution coupled with run-times, and computational complexity of iteration etc.

Finding the roots of an equation is a fundamental problem in diverse fields in physical and social sciences including Computer Science, Engineering (Biological, Civil, Electrical, Mechanical), and Social Sciences (Psychology, Economics, Businesses, Stock Analysis) etc. They look for the optimal solution to the recurring non-linear problems. The problems such as minimization, Target Shooting, Orbital Motion, Planary Motion, Social Sciences, Financial Market Stock prediction analysis, lend themselves to finding roots of non-linear functional equations [6], [7]. There is thorough study by Sapna and Mohan in the financial sector away from mathematics [8].

Thus, a root-finding algorithm implies solving an equation defined by a function that may be linear, non-linear or transcendental. Some root-finding algorithms do not guarantee that they will find a root. If such an algorithm does not find a root, that does not mean that a root does not exist. No single classical method outperforms another method on all functions [9]. A method may outperform other methods on one input and may produce inferior results on another input. Different methods have their own strengths/weaknesses.

There are classical root-finding algorithms: Bisection, False Position, Newton-Raphson, Secant, methods for finding the roots of an equation $f(x) = 0$. Every text book on Numerical Techniques has details of these methods [1], [2], [3]. In the presence of classical methods, enhancements are gradually made to improve the performance of these methods [10],[11],[12].

Recently papers are making a headway on seeking better performing methods. In response, the better algorithms that are combination of classical methods Bisection, Trisection, with False Position, Newton Raphson



methods been developed, namely Adapt1 [10], Adapt2 [11], and Adapt3 [12]. Inspired by these three algorithms, we have crafted a new proficient algorithm, Adapt4, that is a blend of BTsection and Regula Falsi algorithms. This blended algorithm is comparatively better than Adapt1, Adapt2, and Adapt3 with respect to computational efficiency, solution accuracy (less error) and iteration count required to terminate within the specified error tolerance. This algorithm, Adapt4, further optimizes these algorithms, first by blending Bisection and Trisection algorithms, then the resulting algorithm is blended with False Position method to evolve into Adapt 4 algorithm, combining one step further by eliminating some of the computing time and increasing the efficiency of the algorithm.

The paper is organized as follows. Section 2 is brief description of classical methods Bisection, Regula Falsi, Newton-Raphson, Secant; their strengths and pitfalls. In addition, Trisection and new BTsection algorithms are included. Section 3 describes adaptive algorithms and new algorithm, Adapt4, that blends BTsection and False position algorithms. Section 4 presents experimental results simulating the performance of new algorithm Adapt4 and validating it by comparing its performance with the previous related algorithms. Section 5 is conclusion.

II. Background

The classical algorithms Bisection, False Position, Newton-Raphson, Secant methods are readily found in any textbook in detail and iterated in most articles [1], [2], [3]. The classical algorithms, that are ubiquitous [16] are available everywhere. They are iteration-based methods. BIS-Bisection method computes a new approximation at the midpoint of the interval. FP-False Position obtains a new approximation by the secant line joining the endpoints of the function. NR- Newton Raphson derives new approximating by the tangent line at the iterated point on the function. All these algorithms are iterative for calculations of better approximations.

We will discuss the recent algorithms Trisection and BTsection. As Bisection method divides an interval into two equal parts, the Trisection [11] algorithm subdivides the interval in three equal parts to get better estimate at the next iteration. BTsection is new algorithm that subdivides the interval adaptively into three parts: first Bisection into two, then the promising one half is adaptively trisected into parts to get better approximation, see algorithm in section3 [14].

Bisection algorithm states that “If f is continuous on a closed interval $[a, b]$ and f is of opposite signs at the end points, i.e. $f(a)f(b) < 0$, then there is a root in the open interval (a, b) .”

A robust implementation checks that $f(a)f(b) < 0$ before beginning to iterate to find a root. It will fail to proceed on a root finding problem where $f(a)f(b)=0$. In the following problem: $x^2-x-2=0$ on interval $[2,4]$, or $[0, 2]$, or $[-2,-1]$ or $[-3,-1]$. The bisection algorithm first confirms that $f(a)f(b) < 0$ before proceeding to iterate. For another example, for $f(x) = (x-1)(x-2)(x-3)$ on the interval $[1,3]$, the original version of algorithm will not start, but will succeed on $[a,b]$ only when a, b are not 1, 2, or 3, see Table 1.

In order to be successful on an interval including special cases, we reconsider the wording in this theorem and reformulate this theorem to include closed interval all the way. “If f is continuous on a closed interval $[a, b]$ and $f(a)f(b) < 0$, or $f(a)f(b)=0$, then there is a root in the *closed* interval $[a, b]$. For this paper, the Bisection algorithm is rephrased from “ $f(a)f(b) < 0$ ” to “ $f(a)f(b) \leq 0$ ” to include the interval boundary case in definition. For example, the algorithm will succeed to find root $x= 2$ for the equation $x^2-x-2=0$ with $f(a)f(b)=0$, no matter which of the two intervals $[1,2]$ or $[2,3]$ is given. Though, it may look trivial but if $f(a)f(b) = 0$, then a or b must be a root, but we still want to know which one of a and b ?

For approximate solutions using iterative methods, we have some idea about where the root may be. We should have initial guess as close to the location of root as far as possible. That is, we provide a start point or a guess or initial bracketing interval for the algorithm to iterate in search for the approximate value of actual root, within some acceptable tolerance.

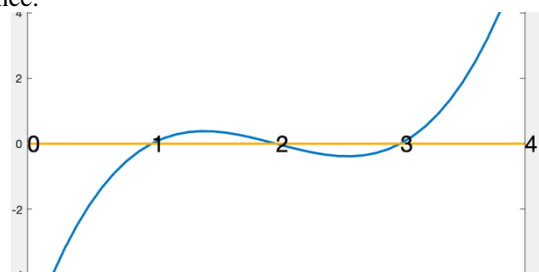


Figure 1. The value of the function at the end points is zero, meaning, $f(a)f(b) < 0$ does not hold good. Modified Bisection algorithm determines a root $x=1$ of equation $(x-1)(x-2)(x-3)=0$, on $[1,3]$, in one iteration

The Table 1 depicts the numerical output of Matlab R2024a64-bit simulation on the function $f(x) = (x-1)(x-2)(x-3)$ on interval $[1, 3]$. The modified Bisection algorithm uses max mum iteration and error tolerance to terminate the algorithm. The Table 1 has name of the function in mathematical form. The upper bound on the



number of iterations is 40 with 12 decimal digits error tolerance. Bisection algorithm is followed by the interval of definition of function, the root computed by the algorithm, the error in function value at the root, the number of iterations required by the algorithm, and computer run-time.

Method	Interval	Root	Error	Iterations	Run-Time
Bisection	[1,3]	1.000000000000	0.000000000000	1	0.003404917

Since the Bisection, Regula Falsi, and Newton Raphson methods are readily available in the literature, their standalone derivations are skipped in this background section.

To enhance the performance of Bisection method, Bader et.al [11] designed a Trisection method that supersedes Bisection method in finding an approximate root. Trisection method reduces the number of iterations performed, computation time, and error of approximation at a small cost on number of function computations. It inspired us to enhance the Trisection algorithm to a BTsection algorithm at no additional compute cost and is at par with Trisection algorithm in computation run-time, and error in approximate root, but BTsection requires fewer number of iterations, see examples in this section. These algorithms are blended with False Position and Newton-Raphson methods to construct new algorithms. The effectiveness and efficacy of root approximation is measured by number of iterations in root calculation and the approximation accuracy of the root at the termination of algorithms. The heuristic metrics for measuring the error and the number iterations and stopping criteria are elaborated here first.

II.1. Heuristics for Comparing Algorithms

II.1.1. Metric for Approximation Error Measurement

There are various ways to measure an error in approximate root of an equations, $f(x)=0$, at successive iterations to continue to a relatively more accurate approximation to the actual root. At iteration n , to determine r_n for which $f(r_n) \cong 0$, we proceed to analyze as follows.

<https://www.lehigh.edu/~ineng2/clipper/notes/bisection.htm>

The iterated root approximation error can be [6]

$$\text{RelativeRootError} = \left| \frac{r_n - r_{n-1}}{r_n} \right|$$

or

$$\text{AbsoluteRootError} = |r_n - r_{n-1}|$$

Since a root can be zero, in order to avoid division by small numbers, it is preferable to use absolute error $|r_n - r_{n-1}|$ for convergence test. Another reason for this is that if $r_n = 2^{-n}$, then $\left| \frac{r_n - r_{n-1}}{r_n} \right|$ is always 1, it can never be less than 1, so the root-error tolerance test cannot be satisfied effectively, this test does not work. Since function value is expected to be zero at the root, an alternate cognitively more appealing error test is to use $f(r_n)$ for error consideration instead of r_n . There are three versions for this concept, for comparison criteria, they are [3]

$$\text{RelativeValueError} = \left| \frac{f(r_n) - f(r_{n-1})}{f(r_n)} \right|$$

or

$$\text{AbsoluteVlaueError} = |f(r_n) - f(r_{n-1})|$$

or

$$\text{TrueValueError} = |f(r_n)|$$

Since $f(r_n)$ is to be close to zero near the root, in order to avoid divide by small numbers, we discard using $\left| \frac{f(r_n) - f(r_{n-1})}{f(r_n)} \right|$. Further, since $|f(r_n) - f(r_{n-1})|$ can be close to zero without $|f(r_n)|$ being close to zero, we discard using $|f(r_n) - f(r_{n-1})|$ also in favor of using only $|f(r_n)|$, trueValue error. For example, $f(r_n) = (n-1)/n$ is such an example. We avoid using the first two criteria for this reason and exploit the last one, $|f(r_n)|$.

Now we are left with two options $|r_n - r_{n-1}|$ and $|f(r_n)|$ to consider for error analysis. Again, r_n and r_{n-1} can be closer to each other without $f(r_n)$ being closer to zero. For example, $r_n = 1 + 1/n$, $f(r_n) = r_n$. Between the options $|r_n - r_{n-1}|$ and $|f(r_n)|$, we find that $|f(r_n)|$ is the only reliable metric for analyzing the approximation error. Hence, we use, $|f(r_n)|$, as the criteria for comparing with tolerance error analysis for all the methods uniformly.

II.2. Metric for Iterations Stopping Criteria, Halting Condition

Stopping criteria plays a major role in simulations. The iteration termination (stopping) criteria for False Position method is different from Bisection method. Tradeoff between accuracy and efficiency is accuracy of the



outcome. In order to obtain n significant digit accuracy, let ϵ_s be stopping error and let ϵ_a be the approximation error at any iteration. If $\epsilon_a < \epsilon_s$, the algorithm stops iterations. With $\epsilon_s = 5/10^{n-1}$, we have n significant digit accuracy in the outcome [1]. The Bisection algorithm is trivial [1], [2], the Trisection and BTsection algorithms are described in section 3.

Here we describe two enhancements to bisection algorithm. Trisection algorithm [11] and BTsection [section 3] algorithm, each algorithm has five comparison tests and reference to four function values. The BTsection algorithm uses the same amount of computation resources as Trisection algorithm.

Since the root is bracketed, smaller the computed bracket, smaller the error in the root approximation. The Bisection method divides the interval in $1/2$ at each iteration. The Trisection method divides the interval in $1/3$ at each iteration. The BTsection method divides the interval in $1/6$ at each iteration. Using the number of iterations bn (Bisection), tn (Trisection), btn (BTsection) required by the Bisection, Trisection, and BTsection algorithms on $[a,b]$ with stopping tolerance tol are:

$$\frac{b-a}{2^{bn}} < tol, \quad \frac{b-a}{3^{tn}} < tol, \quad \frac{b-a}{6^{btn}} < tol,$$

$$bn = \log \frac{b-a}{tol}, \quad tn = (0.63) \log \frac{b-a}{tol}, \quad btn = (0.39) \log \frac{b-a}{tol}$$

Trisection algorithm takes 37% fewer iterations than Bisection algorithm to converge within the desired tolerance.

BTsection algorithm takes 24% fewer iterations than Trisection algorithm to converge to the root within the desired tolerance. In addition, as observed below in both Trisection and BTsection algorithms, in each iteration, there is no change in the computation time: five comparison tests and references to four function values.

II.3. Criteria for Performance

This is an experimental science. The methods are numerical and iterative, not analytic solutions. A method may perform well in one case, and fail miserably in another case, see examples below. No one method outperforms all other methods all the time on all the intervals of definition [9], [15]. We determine that the proposed algorithm performs better than the existing related algorithms is questionable.

There is no easy way to declare that an algorithm superior to other exiting algorithms. There are several factors that must be taken into consideration for distinguishing between the competing algorithms for a root finding problem.

Smart way is to represent new algorithm as to how the proposed algorithm differs from the other algorithms. To emphasize the new idea, it is also desirable that the new solution be applicable to a larger spectrum, devoid of incomplete domain. A simple customized example does not give a quantitative or qualitative view. We need to check the performance and applicability of the proposed approach in a broader scope. To comprehend this further, let us take the following example.

For example, the following problem shows that Trisection algorithm outperforms the other algorithms, see Table2 [11], [12], it gets the correct root in one iteration whereas other cited algorithms take more iterative steps, and more run-time as well. One can infer from this example that the Trisection algorithms 2 and 3 work well and are superior to other algorithms. Of course, it is when input test function is $x^2 - x - 2$ and the interval is [1,4].

The Tables 2-7 depict the Matlab 2024R64bit numerical outputs of the simulation on the functions. In these examples four algorithms are used. In the next sections, we will describe in detail what these algorithms are. Each table has name of the function in mathematical form. Each algorithm has name with the methods used to combine, followed by the interval of definition of function, the root computed by the algorithm, the error in function value at the root, the number of iterations required by the algorithm, and computer run-time.

Table 2 Function $f(x) = x^2 - x - 2$, interval [1,4]				
Method	Root	Error	Iterations	Run-Time
Bisection-FalsePos	2.000000000000	0.000000000000	2	0.020951834000
Trisection-FalsePos	2.000000000000	0.000000000000	1	0.006400042000
Trisection-NewtonR	2.000000000000	0.000000000000	1	0.010491333000
BTsection-FalsePos	2.000000000000	0.000000000000	7	0.007028250000

Now, consider the same function, and same four test algorithms. The initial input interval [1, 4] is changed to [1,3] or [1,5], the Table3 indicates that the inference from Table 2 does not hold good, the tables are turned, see Table 3. Here *algorithm4* solves the same problem in one iteration and uses the lowest amount of run-time. Again, from this example, it is unfair to declare that the algorithm 4 outperforms Algorithms 1, 2, 3. Considering Table 3,



it gives a clue to inference. Even though algorithm4 doesn't terminate in one iteration, it does terminate before the other algorithms terminate.

Table 3 Function $f(x) = x^2 - x - 2$				
Method	Root	Error	Iterations	Run-Time
<u>Interval [1,3]</u>				
Adapt1: Bisection-FalsePos	2.000000000000	0.000000000000	1	0.003802500000
Adapt2: Trisection-FalsePos	2.000000000000	0.000000000000	7	0.006985792000
Adapt3: Trisection-NewtonR	2.000000000000	0.000000000000	13	0.026495625000
Adapt4: BTsection-FalsePos	2.000000000000	0.000000000000	1	0.003495625000
<u>Interval [1,5]</u>				
Adapt1: Bisection-FalsePos	2.000000000000	0.000000000000	6	0.028778667000
Adapt2: Trisection-FalsePos.	2.000000000000	0.000000000000	7	0.014464750000
Adapt3: Trisection-NewtonR	2.000000000000	0.000000000000	14	0.031316708000
Adapt4: BTsection-FalsePos	2.000000000000	0.000000000000	1	0.010769959000

We have seen that in one case algorithms 2, 3 and in the other two cases algorithms 1, 4 come out ahead because they solve the problem in one iteration and use the lowest run-time. The dilemma is to determine which algorithm will work satisfactorily in most cases, if not, all the cases. We have to consider not only this function but other functions as well; not one interval, but other related intervals as well. Let us first see the next examples in Table 4, where none of these four algorithms succeed in finding the approximation in one iteration. We used different functions and different intervals for these examples in Table 5. In all these examples, the algorithm 4 has the lowest number of iterations and lowest amount of run-time. We have tested on number other examples as well to validate the same phenomena. The following examples use different functions and different initial intervals [1, 6], [1,7], [1, 8], even [0,1] as well to make a intuitive heuristic that algorithm 4 may be more efficient than the others. In all the ensuing examples in this paper, algorithm halting upperbound is 40 for iterations and upto 10 decimal digits for approximation in the acceptable root.

Table 4				
Method	Root	Error	Iterations	Run-Time
$f(x) = x - \cos(x)$ on [0, 1]				
Adapt1: Bisection-FalsePos	0.739085133215	0.000000000000	7	0.008797250000
Adapt2: Trisection-FalsePos	0.739085133215	0.000000000000	6	0.004603500000
Adapt3: Trisection-NewtonR	0.739085133215	0.000000000001	11	0.011723000000
Adapt4: BTsection-FalsePos	0.739085133215	0.000000000001	5	0.002825583000
$f(x) = x - \exp(-x)$ on [0, 2]				
Adapt1: Bisection-FalsePos	0.567143290410	0.000000000000	8	0.046667333000
Adapt2: Trisection-FalsePos	0.567143290410	0.000000000000	7	0.017032042000
Adapt3: Trisection-NewtonR	0.567143290409	0.000000000001	11	0.015795000000
Adapt4: BTsection-FalsePos	0.567143290410	0.000000000000	6	0.014784125000
$f(x) = x^2 - 2$ on [0,3]				
Adapt1: Bisection-FalsePos	1.414213562373	0.000000000000	8	0.003360958000
Adapt2: Trisection-FalsePos	.4142135623731	0.000000000000	7	0.002914916000
Adapt3: Trisection-NewtonR	1.414213562373	0.000000000000	13	0.005550834000
Adapt4: BTsection-FalsePos	1.414213562373	0.000000000000	6	0.007887542000



Table 5				
Method	Root	Error	Iterations	Run-Time
$f(x) = 0.986x^3 - 5.181x^2 + 9.067x - 5.289$ on $[1, 6]$,				
Adapt1: Bisection-FalsePos	1.929846242840	0.000000000001	10	0.005306959000
Adapt2: Trisection-FalsePos	1.929846242844	0.000000000000	9	0.004571250000
Adapt3: Trisection-NewtonR	1.929846242850	0.000000000000	11	0.017164083000
Adapt4: BTsection-FalsePos	1.929846242848	0.000000000000	7	0.005307125000
$f(x) = x^2 - x - 2$ on $[1, 7]$				
Adapt1: Bisection-FalsePos	2.000000000000	0.000000000000	9	0.003208750000
Adapt2: Trisection-FalsePos	2.000000000000	0.000000000000	7	0.003186292000
Adapt3: Trisection-NewtonR	2.000000000000	0.000000000001	13	0.018284292000
Adapt4: BTsection-FalsePos	2.000000000000	0.000000000000	6	0.002163208000

II.4. Algorithms

II.4.1. Bisection, Regula Falsi, Newton-Raphson algorithm

These algorithms are ubiquitous standard [1], [2], [3]. We use Regula Falsi method for analysis of these algorithms. There are multiple reasons for neglecting Newton Raphson from this analysis: (1) it requires that function be differentiable, (2) it depends heavily on the start point, (3) iterated approximations are not bracketed, (4) it fails if start point is not close to the root. Although False Position method is preferably used in these algorithms, it does not fit well in the category of Bisection, Trisection and BTsection, we will first compare these three algorithms.

II.4.2. Trisection Algorithm [11].

Bader [11] extended bisection algorithm to trisection in order to create a better algorithm to combine it with false position algorithm. The algorithm is as follows.

- Input: Function $f(x)$, Initial approximations $[a, b]$ and absolute error eps .
- Output: Approximate root r , *bracketing interval*, and number of iterations k

```

for k=1 to n
p := (2*a + b)/3; q := (a + 2*b)/3;
if |f(p)| < |f(q)|
    r := p
else
    r := q
endif;
if |f(r)| < eps
    return r, a, b, k;
else if f(a)*f(p) < 0
    b:=p;
else if f(p)*f(q) < 0
    a:=p;
    b:=q;
else
    a:=q;
end if;
end for
    
```



II.4.3. BTsection Algorithm

In this paper, we adapted both Bisection and Trisection algorithms to craft a BTsection algorithm to combine, more heuristically, with False Position method [14]. BTsection algorithm incurs no more computational cost than the Trisection algorithm, but requires fewer iterations, in general, to reach an approximate solution. The algorithm is as follow.

- Input: function $f(x)$, Initial interval $[a,b]$ and absolute error eps .
- Output: Approximate root r , *bracketing interval*, and number of iterations k

for $i=1:i_{max}$

 bisection step

$r=(a+b)/2;$

 if $(f(a)*f(r)) < 0$

$b=r; r=(a+2*b)/3;$

 elseif $(f(r)*f(b)) < 0$

$a=r; r=(2*a+b)/3;$

 else

 return i, r, a, b

 endif

 trisection step

 if $(f(a)*f(r)) < 0$

$b=r;$

 elseif $(f(r)*f(b)) < 0$

$a=r;$

 else

 return i, r, a, b

 endif

if $|f(r)| < eps$

 return $i,r,a,b ;$

endif

endfor

The Trisection algorithm [11] and a new BTsection algorithm are conceptually equivalent in each iteration. From the following tables, it is clear that BTsection algorithm requires fewer iterations to converge. It shows that BTsection algorithm competes successfully with the other algorithms with respect to number of loop iterations, run-time, and accuracy in approximation of the root. These benchmark functions appear in recent papers in the literature. See Tables 10-15 for comparing the performance of the sectioning algorithms. The functions and the interval of definition are generic of any algorithm. In all the ensuing examples in this paper, algorithm halting upperbound is 40 for iterations and upto 10 decimal digits for approximation in the acceptable root.

The tables 5-6 depict the Matlab numerical outputs of the simulation on the functions. Each table has name of the function in mathematical form. Each algorithm has name followed by the interval of definition of function, the root computed by the algorithm, the error in function value at the root, the number of iterations required by the algorithm, and computer Run-time.



Table 6				
Method	Root	Error	Iterations	Run-Time
$f(x) = x - \exp(-x)$ on [0, 1]				
Bisection	0.567143290409	0.000000000001	38	0.009934417000
Trisection	0.567143290410	0.000000000000	25	0.008384250000
BTsection	0.567143290409	0.000000000001	19	0.013742375000
$f(x) = x - \cos(x)$ on [0, 1]				
Bisection	0.739085133215	0.000000000001	39	0.026511958000
Trisection	0.739085133215	0.000000000000	24	0.017229750000
BTsection	0.739085133215	0.000000000000	21	0.021565208000
$f(x) = (x-1)(x-2)(x-3)$ on [1,3]				
Bisection	2.000000000000	0.000000000000	1	0.003182459000
Trisection	2.000000000000	0.000000000000	26	0.003927291000
BTsection	2.000000000000	0.000000000000	1	0.003374833000
$f(x) = \exp(x)(x-1)$ on [0, 4]				
Bisection	1.000000000000	0.000000000000	2	0.012940750000
Trisection	1.000000000000	0.000000000001	29	0.016367250000
BTsection	1.000000000000	0.000000000001	2	0.017235792000
$f(x) = 0.986 x^3 - 5.181 x^2 + 9.067 x - 5.289$ on [1, 5]				
Bisection	1.929846242856	0.000000000001	37	0.002006000000
Trisection	1.929846242855	0.000000000001	24	0.002049208000
BTsection	1.929846242843	0.000000000000	20	0.003908708000
$f(x) = x^2 - x - 2$ on [1,6]				
Bisection	1.000000000000	0.000000000003	40	0.006641375000
Trisection	2.000000000000	0.000000000001	27	0.007940666000
BTsection	2.000000000000	0.000000000001	17	0.006606125000

III. Adaptive Algorithms

Now we present recent algorithms and propose a newblended algorithm. The existing blended algorithms have one thing in common. At each iteration, they compute the bracketing interval for each of the two algorithms and compute the interval common to the two algorithms to use at the next iteration. It incurs a computation step. Here in the new algorithm, there is no need to perform such computation because we can have the common interval readily available without performing this computation.

First, we describe the original algorithm, namely, Adapt1 [10] based on classical Bisection and False Position algorithms. Since the classical algorithms can be found in any textbook, those algorithms are not described here.

In Adapt1 algorithm, at each iteration, more promising root between the Bisection and False Position approximate roots is selected, and common interval is computed for the next iteration. This curtails the unnecessary iterations in either method. It was succeeded by more efficient algorithms using Trisection method in place of Bisection algorithm: Adapt2 [11] and Adapt3 [12]. These algorithms lead the way for us to discover more heuristics to design a new blended algorithm Adapt4 which is more efficient than the previous three algorithms. All the four algorithms are described here for reference. For ease in readability, the prefixed variables use symbols: b for Bisection, p for False Position, t for Trisection, and bt for BTsection. The variables without prefix are algorithms local working variable.



Adapt1: Bisection and False Position Algorithm [10]

Input: f , $[a, b]$, ϵ_s , maxIterations

Output: root r , k -number of iterations, error of approximation ϵ_a , bracketing interval $[a_{k+1}, b_{k+1}]$

//initialize

$k = 0$; $a_1 = a$, $b_1 = b$

Initialize bounded interval for bisection and false position.

$pa_{k+1} = ba_{k+1} = a_1$; $pb_{k+1} = bb_{k+1} = b_1$

repeat

$pa_{k+1} = ba_{k+1} = a_k$; $pb_{k+1} = bb_{k+1} = b_k$

compute the midpoint and error at m .

$m = \frac{a_k + b_k}{2}$, and $\epsilon_m = |f(m)|$

compute the False Position point and error,

$s = a_k - \frac{f(a_k)(b_k - a_k)}{f(b_k) - f(a_k)}$ and $\epsilon_p = |f(s)|$

if $|f(m)| < |f(s)|$,

$f(m)$ is closer to zero, Bisection method determines bracketing interval $[ba_{k+1}, bb_{k+1}]$

$r = m$

$\epsilon_a = \epsilon_m$

if $f(a_k) \cdot f(r) > 0$,

$ba_{k+1} = r$; $bb_{k+1} = b_k$;

else

$ba_{k+1} = a_k$; $bb_{k+1} = r$;

endif

else

$f(s)$ is closer to zero, False Position method determines bracketing interval $[pa_{k+1}, pb_{k+1}]$

$r = s$

$\epsilon_a = \epsilon_p$

if $f(a_k) \cdot f(r) > 0$,

$pa_{k+1} = r$; $pb_{k+1} = b_k$;

else

$pa_{k+1} = a_k$; $pb_{k+1} = r$;

endif

endif

Since the root is bracketed by both $[ba_{k+1}, bb_{k+1}]$ and $[pa_{k+1}, pb_{k+1}]$,

$[a_{k+1}, b_{k+1}] = [ba_{k+1}, bb_{k+1}] \cap [pa_{k+1}, pb_{k+1}]$ or

$a_{k+1} = \max(ba_{k+1}, pa_{k+1})$;

$b_{k+1} = \min(bb_{k+1}, pb_{k+1})$;

outcome: iteration complexity, root, and error of approximation

iterationCount = k

$r = r_k$

error = $\epsilon_a = |f(r)|$

$k = k + 1$

until $|f(r)| < \epsilon_s$ or $k > \text{maxIterations}$

Adapt2: Trisection and False Position Algorithm[11]

This function implements a blend of trisection and false position methods.

Input: The function f ; the interval $[a, b]$ where $f(a)f(b) < 0$ and the root lies in $[a, b]$,

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

$n = 0$; $a_1 := a$; $a_2 := a$; $b_1 := b$, $b_2 := b$

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)| <= 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)
    
```

Adapt 3: Trisection and Newton-Raphson Algorithm [12]

This algorithm is along the same lines as Adapt 2, but with (1) instead of false position method, it uses Newton-Raphson algorithm which requires differentiability of the function, (2) improved iteration count and accuracy in the combination step: namely, the common interval in each iteration is computed by analyzing the five function values and then mapped to parameter values for the optimal interval.

The Algorithm is as follows.

- Input: Function $f(x)$, an Initial approximations x_0 and absolute error eps .
- Output: Root x and number of iterations n

```

df(x):=f'(x); k:=0;
for k=1:n
    p := (2*a + b)/3;
    q := (a + 2*b)/3;
    if |f(p)| < |f(q)|
        then r := p - f(p)/df(p);
        else r := q - f(q)/df(q);
        end if;
    if |f(r)| < eps
        then
            return r, k;
        else
    
```



```

find fv:={f(a),f(b),f(r),f(p),f(q)};
a := xa where fv max -ve;
b := xb where fv min +ve
end if;
end.

```

Recall, in these three algorithms, there are two steps in common to coordinate the two algorithms to blend. At each iteration, they determine (1) the promising approximation root out of the two roots (2) the common interval bracketing the approximate root. In Adapt1 and Adapt 2 algorithms, this simply reduces to intersection of two intervals so that common interval contains the approximate root. No function evaluation is involved in the search for common interval to contain the predicted approximate root. In the Adapt3 algorithm, it searches among five function values used to determine two function values pertaining the common interval. From these two function values, the function parameters are determined to create the common bracketing interval.

III.1. NewAdaptove Algorithm

Adapt4 algorithm provides a more proficient approach to optimization: (1) BTsection algorithm is used instead of bisection or trisection, (2) it eliminates the computation of common interval required by the foregoing algorithms used to combine, there is no work needed to determine the better of the two roots. This leads to more efficiency for optimal root approximation and readily available common interval. It is based on common sense Occam’s razor principle [13], Fig 2. The Occam’s razor principle is a heuristic, not a proof. That is, when faced with competing choices, we use the simplest one from what we have. It is shown that Occam’s Razor Principle works quite well in this caseAdapt4 by eliminating the unnecessary computational steps in the existing algorithms.



Figure 2. https://conceptually.org/concepts/occams-razor_

Adapt4

This algorithm is a blend of BTsection and False Position methods to find iteration approximations with minimal effort.

Input $a_0, b_0, r_0, \text{eps}, \text{imax}, f$

Output k, a_k, b_k, r_k

for $k=1:\text{imax}$

BTsection iteration step determines:

$b_{t_k}, b_{t_{k-}}, b_{t_{r_{k-}}}$ from $b_{t_{k-1}}, b_{t_{k-1-}}, b_{t_{r_{k-1-}}}$

relabel

$b_{t_k}, b_{t_{k-}}, b_{t_{r_{k-}}}$ to a_k, b_k, r_k

False-position iteration step

input isa_k, b_k, r_k instead of old $pa_{k-1}, pb_{k-1}, pr_{k-1}$

false position iteration step determines:

pa_k, pb_k, pr_k from a_k, b_k, r_k instead of from old $pa_{k-1}, pb_{k-1}, pr_{k-1}$

This makes $[pa_k, pb_k]$ as the common interval $[a_k, b_k]$ without additional computation.

The use of a_k, b_k, r_k instead of old $pa_{k-1}, pb_{k-1}, pr_{k-1}$, makes this step more optimal for next approximation of root.

if $f(pr_k) < \text{eps}$

return k, pa_k, pb_k, pr_k .

end



relabel pa_k, pb_k, pr_k to qa_k, qb_k, qr_k .

endFor

Summarizing

As described in algorithms Adapt1 [10], Adapt2 [11], Adapt3 [12] and our Adapt4, each algorithm has the similar implementation steps except for the major difference explained here. Succinctly, the iteration step in each algorithm is:

Adapt1 step

Apply Bisection criteria

$[ba_k, bb_k, br_k] = \text{Bisection}(a_{k-1}, b_{k-1}, r_{k-1}, f)$

Apply FalsePosition criteria

$[pa_k, pb_k, pr_k] = \text{FalsePosition}(a_{k-1}, b_{k-1}, r_{k-1}, f)$

Apply blending criteria

r_k is better of br_k, pr_k ,

$[a_k, b_k]$ is common to $[ba_k, bb_k], [pa_k, pb_k]$. It requires intersection of $[ba_k, bb_k], [pa_k, pb_k]$

r_k is better of br_k, pr_k , with smaller of values $f(br_k)$ and $f(pr_k)$

r_k belongs to $[a_k, b_k]$

Adapt 2 step

$[ta_k, tb_k, tr_k] = \text{Trisection}(a_{k-1}, b_{k-1}, r_{k-1}, f)$

$[pa_k, pb_k, pr_k] = \text{FalsePosition}(a_{k-1}, b_{k-1}, r_{k-1}, f)$

The outcome of blending step is:

$[a_k, b_k]$ is common to $[ta_k, tb_k], [pa_k, pb_k]$. It requires intersection of $[ta_k, tb_k], [pa_k, pb_k]$

r_k is better of tr_k, pr_k ,

r_k belongs to $[a_k, b_k]$

Adapt3 step

$[ta_k, tb_k, tr_k] = \text{Trisection}(a_{k-1}, b_{k-1}, r_{k-1}, f)$

$[na_k, nb_k, nr_k] = \text{NewtonRaphson}(a_{k-1}, b_{k-1}, r_{k-1}, f)$

The upshot of blending is:

$[a_k, b_k]$ is common from $[ba_k, bb_k], [pa_k, pb_k]$ and nr_k . It requires intersection of $[ba_k, bb_k], [pa_k, pb_k]$. It involves analyzing $\{f(ba_k), f(bb_k), f(na_k), f(nb_k), f(nr_k)\}$, then finding the common interval from $\{ba_k, bb_k, pa_k, pb_k, nr_k\}$.

r_k as better of tr_k, nr_k ,

r_k belongs to $[a_k, b_k]$

Adapt4 step

$[a, b, r] = \text{BTsection}(a_{k-1}, b_{k-1}, r_{k-1}, f)$

$[pa_k, pb_k, pr_k] = \text{FalsePosition}(a, b, r, f)$

The conclusion of this blending is that it requires no calculation: $[a_k, b_k]$ is the same interval $[pa_k, pb_k]$ containing r_k is the same as pr_k , the desired root. This algorithm is optimal in run-time, the number of iterations and the accuracy in approximate root.



$$[a_k, b_k, r] = \text{BTsection}(a_{k-1}, b_{k-1}, r_{k-1}, f)$$

$$[pa_k, pb_k, pr_k] = \text{FalsePosition}(a_k, b_k, r, f)$$

It is automatically blended. The object of this blending is *that it requires no calculation*: the final $[a_k, b_k]$ is the same interval $[pa_k, pb_k]$ containing r_k is the same as pr_k , the desired root. This algorithm is optimal in run-time, the number of iterations and the accuracy in approximate root.

IV. Discussion

Many researchers focused their attention toward using such methods to solve their problems. The roots are calculated iteratively, along with the number of iterations within a specified tolerance. In this section, all the existing adaptive methods are compared. Error analysis is performed. It is validated here that Adapt4 algorithm preferable than the existing algorithms Adapt1, Adapt2, and Adapt3.

IV.1. Empirical Evidence Testing.

We have tested our new algorithm Adapt4 with other algorithms on diverse examples of functions found in articles in the literature to validate that new algorithm outperforms the other three algorithms.

IV.2. Experiments in Matlab

Moazzam [9] used Microsoft visual C++ to find roots, we used Matlab R2022A 64 bit (maci64) on MacBook Pro MacOS Sonoma 14.1.1, 16 GB Apple M1Pro. Along with numerous different functions and varying intervals, we use the same frequently cited functions. We have validated that the new algorithm performs better than all the adaptive algorithms. These tests indicate that blending with false position algorithm is still preferable to blending with Newton-Raphson possibly because the Newton-Raphson method requires that (1) the function be differentiable, (2) meet additional constraints on the derivative for guaranteed convergence.

For all these Tables 1-7, the functions are frequently occurring in the literature. For all the functions the upperbound on the number of iterations is 40 and acceptable approximation error bound is 10^{-12} . These tables are for comparison of four algorithms number of iterations, run-time, accuracy of the solution.

Frequently used twenty functions are compared for iterations efficiency. These functions are listed in the table. This table lists functions used in the simulations. Columns represent the functions, the interval of function's domains, the number of iterations required by each of four algorithm to reach an approximate root within tolerance. Figure3 is a visual representation of the Table 7.

Table 7. Listing of functions, Intervals and Iterations in four Adaptive algorithms.

Function	interval	Iterations			
		Adapt1	Adapt2	Adapt3	Adapt4
$x^2 - 2$	[1,3]	8	7	13	6
$x^2 - 3$	[1,2]	7	6	12	6
$x^2 - 5$	[2,7]	10	8	13	7
$x^2 - 10$	[3,4]	7	6	12	5
$x^2 - x - 2$	[1,5]	6	7	14	5
$x^2 + 2x - 7$	[1,3]	5	6	12	5
$x^3 - 2$	[1,8]	11	8	14	8
$(x-1)(x-2)(x-3)$	[1,3]	1	1	2	1
$x^{10} - 1$	[0,1.4]	11	9	12	8
$x - \exp(-x)$	[0,2]	8	7	11	6
$x \exp(x) - 7$	[0,2]	10	6	13	7
$x \exp(x) - 7$	[0,3]	11	7	15	7
$\exp(x) - 3x - 2$	[2,3]	9	6	12	6
$x - \exp(-x)$	[1,2]	8	6	13	5
$\sin(x) - x^2$	[0.6,1]	7	7	10	6
$x \cos(x) - 1$	[0,2]	5	5	11	4
$x - \cos(x)$	[0,1]	7	6	11	5
$\sin(x) \sinh(x) + 1$	[3,4]	8	7	14	6
$\exp(x) (x-1)$	[0,4]	8	8	13	7



In the chart below, functions are along x-axis, number of iterations along y-axis, legend color indicates algorithms 1,2,3,4. Four algorithms are in the body of the chart.

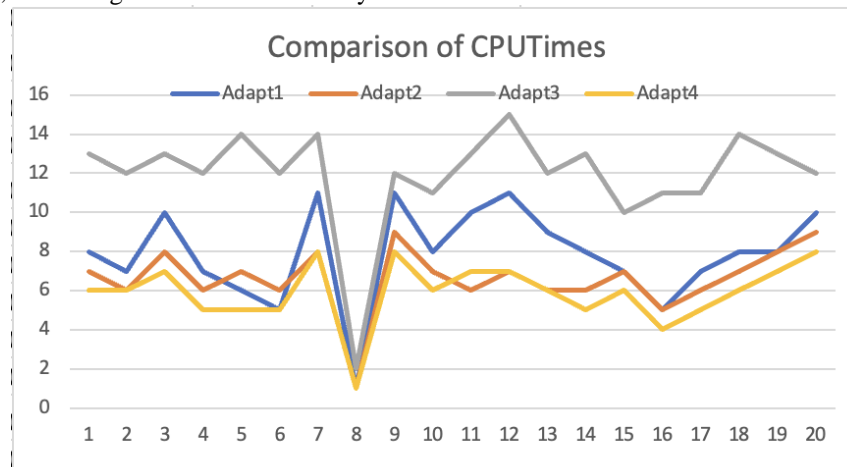


Figure 3. This figure compares the CPU times if iterations in each function tested with four algorithms.

The horizontal x-axis indicates twenty functions used in the experiments. The left vertical y-axis is the axis of number of iterations. Four curves represent the number of seconds, iterations used by each of function to arrive at a root. The algorithm Adapt4 shows it used fewer iterations than Adapt1, Adapt2, Adapt3 algorithms. The actual run times are shown in the tables below. The curve Adapt4 validates that it is better performing than other three algorithms.

V. Conclusions

We have designed and implemented a new algorithm Adapt4, an efficient algorithm blending of BT section and Regula Falsi methods. The algorithm was implemented in Matlab R2024A 64 bit (maci64) on MacBook Pro MacOS Sonoma 14.1.1 16 GB Apple M1Pro. The implementation tests, numerical and visual iteration count table 15 indicates that the algorithm Adapt4 outperforms the Adapt1, Adapt2, and Adapt3 algorithms. The experiments on numerous datasets used in the literature validate that the new algorithm Adapt4 is effective both conceptually and computationally. This algorithm is applicable to both differentiable and non-differentiable functions. This paper provides an essential algorithm for the practitioners and a contribution to the repertoire of adaptive algorithms.

VI. References

- [1]. Steven C Chapra and Raymond P Canale, Numerical Methods for Engineers, 7th Edition, McGraw-Hill Publishers, 2015.
- [2]. John H. Mathews and Kurtis K. Fink Numerical Methods Using Matlab, 4th Edition, 2004 ISBN: 0-13-065248-2 Prentice-Hall Inc. Upper Saddle River, New Jersey, USA
- [3]. J. F. Traub: Iterative Methods for the Solution of Equations, Chelsea Publishing company, New York, NY, USA, 1982.
- [4]. Douglas Wilhelm Harder, <https://ece.uwaterloo.ca/~dwharder/NumericalAnalysis/10RootFinding/falseposition/>, accessed Jun 2019.
- [5]. Ehiwario, J.C., Aghamie, S.O.; Comparative Study of Bisection, Newton-Raphson and Secant Methods of Root-Finding Problems; IOSR Journal of Engineering (IOSRJEN) www.iosrjen.org; ISSN (e): 2250-3021, ISSN (p): 2278-8719; Vol. 04, Issue 04 (April. 2014), ||V1|| PP 01-07.
- [6]. Donna Calhoun https://math.boisestate.edu/~calhoun/teaching/matlab-tutorials/lab_16/html/lab_16.html accessed December 2023
- [7]. C. Thinzar, and N. Aye, "Detection the storm movement by sub pixel registration approach of Newton Raphson method" International Journal of e-Education, e-Business, e-Management and e-Learning, Vol. 4, No. 1, 2014.
- [8]. S. Sapna, Biju R. Mohan, Comparative Analysis of Root Finding Algorithms for Implied Volatility Estimation of Ethereum Options, Computational Economics, <https://doi.org/10.1007/s10614-023-10446-8>, springer, August 2023, pp.1-37.
- [9]. G. Moazzam, A. Chakraborty, A. Bhuiyan: A robust method for solving transcendental equations. Int. J. Comput. Sci., 9 (2012), 413–419.



-
- [10]. Sabharwal, C. L. (2019). Blended root finding algorithm outperforms bisection and regula falsi algorithms. *Mathematics*, 7(11), 1118.
- [11]. Badr, E., Almotairi, S., & Ghamry, A. E. (2021). A comparative study among new hybrid root finding algorithms and traditional methods. *Mathematics*, 9(11), 1306.
- [12]. Srinivasarao, Thota A Blended Root-Finding Algorithm for Solving Transcendental Equations SNAPP, *International Journal of Applied and Computational Mathematics*, <https://doi.org/10.21203/rs.3.rs-2956091/v1>, May 2023, pp.1-9.
- [13]. Occam'Razor: <https://conceptually.org/concepts/occams-razor>, accessed December 2023.
- [14]. G.E. Collins and A.G. Akritas Trisection using Bisection, https://www.lamath.org/journal/Vol1/trisection_using_bisection.pdf, 1976.
- [15]. Srivastava, R.B and Srivastava, S (2011), Comparison of Numerical Rate of Convergence of Bisection, Newton and Secant Methods. *Journal of Chemical, Biological and Physical Sciences*. Vol 2(1) pp 472-479, 2011.
- [16]. Wikipedia https://en.wikipedia.org/wiki/False_position_method#Two_historical_types accessed December 2023.