International Journal of Latest Engineering and Management Research (IJLEMR)
ISSN: 2455-4847
www.ijlemr.com || Volume 08– Issue 12 || December 2023 || PP. 13-28

Heuristics and Algorithms for Roots of Non-Linear Equations

Chaman Lal Sabharwal
Missouri University of Science and Technology
Rolla, Missouri - 65409, USA

Abstract: Optimization problems lead to solving non-linear equations for calculating the value of a parameter, that is the root of the equation. Numerical iterative techniques are frequently used to find roots of an equation when analytic solution is not available. Nascent hybrid techniques to find roots of an equation is new to the fundamental problem in diverse fields. Recently, there have been some attempts to design hybrid algorithms for efficient solutions. Here we strive for a new hybrid algorithm which is an intuitive approach to hybridization. In this paper, we design and implement two new algorithms: (1) a new Quadsection algorithm extending the Trisection algorithm, (2) a hybrid algorithm, Hybrid 4, that is more efficient than the existing hybrid algorithms. Hybrid 4 algorithm is a blend of Quadsection algorithm and Regula Falsi algorithm. The implementation results validate that the new algorithm, Hybrid4, surpasses the efficiency of existing hybrid algorithms Hybrid 1, Hybrid 2, and Hybrid 3 algorithms. This paper contributes an essential hybrid algorithm to the repertoire of hybrid root finding algorithms.

Keywords: Regula Falsi, Newton-Raphson, Bisection, Trisection, Quadsection, Hybrid Algorithm.

I. INTRODUCTION

Numerical iterative techniques are used to find roots of an equation when analytic solution is not available [Chapra, Traub]. Hybrid algorithm is a new concept to iterative solutions. Envisioning hybrid techniques to find roots of an equation is an inspiring technique to solve the fundamental problem in diverse fields. Recently, there have been three hybrid algorithms designing efficient solutions. Here we strive for a new hybrid algorithm which is a more intuitive approach to hybridization. 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 Hybrid4 that is more efficient hybrid of Quadsection and False Position methods. The implementation results validate that the new algorithm surpasses the existing hybrid algorithms. Thus, we contribute an essential hybrid algorithm to the repertoire of root finding algorithms.

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, Plenary Motion, Social Sciences, Financial Market Stock prediction analysis etc, lend themselves to finding roots of non-linear functional equations [Calhoun], [Thinzar]. There is thorough study by Sapna and Mohan in the financial sector away from mathematics [Sapna].

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 [Chapra,Traub]. Even though classical methods have been developed and used for decades, yet enhancements are progressively made to improve the performance of these methods [Sabh], [Badr], [Thota].

Recently papers are making a ahead way on seeking better performing methods. In response, the better algorithms that are hybrid of classical methods Bisection, Trisection, with False Position, Newton Raphson methods been developed, namely Hybrid1 [Sabh], Hybrid 2 [Badr], and Hybrid 3 [Thota]. Inspired by these three algorithms, we have crafted a new proficient algorithm, Hybrid4, that isa blend of Quadsection and Regula Falsi algorithms. This blended algorithm is comparatively better than Hybrid1, Hybrid2, and Hybrid3 with respectto computational efficiency, solution accuracy (less error) and iteration count required to terminate within the specified error tolerance. This algorithm, Hybrid4, further optimizes these algorithms one step furtherby eliminating some of the computing time and increasing the efficiency of the algorithm.

The paper is organized as follows. Section II is brief description of classical methods Bisection, Regula Falsi, Newton-Raphson, Secant; their strengths and pitfalls. In addition, Trisection and new Quadsection algorithms are included. Section III describes hybrid algorithms and new algorithm, Hybrid4, that blends Quadsection and False position algorithms. Section IV presents experimental results simulating the performance of new algorithm Hybrid4 and validating it by comparing its performance with the previous hybrid algorithms. Section V is conclusion.

II. RELATED BACKGROUND

The classical algorithms Bisection, False Position, Newton-Raphson, Secant methods are readily found in any text book in detail and iterated in most articles [Chapra]. We take close look at the bisection algorithm.

Bisection Method 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)."

Standard algorithm implementation expects that f(a)fb) < 0 to begin with. Because, f(a)f(b)=0 in the following, it will fail to proceed on this root finding problem: " x^2-x-2=0 on interval [2,4], or [0, 2], or [-2,-1] or[-3,-1]. A robust bisection algorithm first confirms that f(a)f(b) < 0 before proceeding to iterate. For 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 succceed on [a,b] only when a,b are not 1,2,3.

In order to be successful on any 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 is not of same sign at the end points, i.e. f(a)f(b) <= 0, then there is a root in the closed interval [a, b]."The Bisection algorithm is generalized from "f(a)f(b) < 0" to "f(a)f(b ≤ 0" to include the interval boundary also in definition. For example, the equation x^2-x-2=0 may show up in some applications with an interval like [2,4], [0, 2]. Though, it may look simple 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 to the algorithm to iterate in search for the true approximate value of actual root, within some acceptable tolerance.

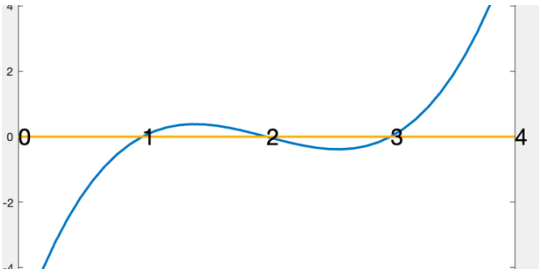

Figure 1. With enhanced version of Bisection on [1,3], it determines a root, at x=2 in one iteration.

Table 1Function f(x) = **(x-1).*(x-2).*(x-3)**
Max Iterations = 40 Error tolerance = 0.000000000001

Method	Interval	Root	Error	Iterations	TimeCPU
Bisection	[1,3]	2.000000000000	0.000000000000	1	0.003404917000

With enhanced version of Bisection on [1,3], it findsa root, at x=2 in one iteration.

Since the Bisection, Regula Falsi, and Newton Raphson methods are readily available in the literature, their standalone derivations are skipped in this background section. However, for the sake of completeness, these algorithms are delegated to an appendix in section VI for reference.

To enhance the performance of Bisection method, Bader et.al [Badr] 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 Quadsection algorithm at no additional compute cost and is at par with Trisection algorithm in computation CPU time, and error in approximate root, but Quadsection requires fewer number of iterations, see examples in this section. These algorithms are blended with False Position and Newton-Raphson methods to construct hybrid 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 heuristics metrics for measuring error, the number iterations and stopping criteria are elaborated here first.

Heuristics for comparing algorithms

2.1 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.

The iterated *root* approximation error can be

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

or

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

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

or

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

or

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 forerror 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 for comparing with tolerance error analysis for all the methods uniformly.

2.2 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$\in_s$ be stopping error and let$\in_a$ be the approximation error at any iteration. If$\in_a < \in_s$, the algorithm stops iterations. With$\in_s = 5/10^{n-1}$, we have n significant digit accuracy in the outcome [Chapra]. The bisection algorithm is trivial [Sabh], Trisection and Quadsection algorithms are described in Section III.

Here we describe two enhancements to bisection algorithm. Trisection algorithm [Badr] and Quadsection [section III] algorithm, each algorithm has four comparison tests and seven function evaluation references. The Quadsection algorithm uses the same amount of computation resources as Trisection algorithm.

But the number of iterations bn (Bisection),tn (Trisection), qn (Quadsection) required by the Bisection, Trisection, and Quadsection algorithms on [a,b] with stopping tolerance eps are

bn =log{(b-a)/tol} tn=(0.63) log{(b-a)/tol} qn= (0.5) log{(b-a)/tol}.

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

Quadesection algorithm takes 13% *fewer* iterations than Trisection algorithm to converge within the desired tolerance. In addition, as observed below in both Trisection and Quadsectionalgorithms, in each iteration, there is no change in the computation time: seven references to function evaluation and four references to compare test.

2.3 How to determine that the proposed algorithm performs better than the existing related algorithms?

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 [Moazzam].

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 [Bader], [Thota], it gets the correct root in one iteration whereas other cited algorithms take more iterative steps, and more CPU time as well. One can infer from this example that the Trisection algorithm 2 and 3 work well and are superior to other algorithms. Of course, it is when input test function is **x^2 - x – 2**andinterval is **[1,4].**

Table 2In this example, algorithm termination upper bound is 40 for iterations and upto 10 decimal digits for approximation in the acceptable root.

Function **f(x) = x^2 - x - 2**,

Method	Interval	Root	Error	Iterations	TimeCPU
Hybrid1: Bisection-FalsePos	[1,4]	2.000000000000	0.000000000000	2	0.008360458000
Hybrid2: Trisection-FalsePos	[1,4]	2.000000000000	0.000000000000	1	0.005007083000
Hybrid3: Trisection-NewtonRaph	[1,4]	2.000000000000	0.000000000000	1	0.004695708000
Hybrid4: Quadsection-FalsePos	[1,4]	1.999999999991	0.000000000026	5	0.010161750000

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

Table 3 Function **f(x) = x^2 - x - 2**,

Method	Interval	Root	Error	Iterations	TimeCPU
Hybrid1: Bisection-FalsePos	[1,3]	2.000000000000	0.000000000000	1	0.003889167000
Hybrid2: Trisection-FalsePos	[1,3]	1.999999999998	0.000000000006	6	0.009600083000
Hybrid3: Trisection-NewtonRaph	[1,3]	2.000000000005	0.000000000014	11	0.015903959000
Hybrid4: Quadsection-FalsePos	[1,3]	2.000000000000	0.000000000000	1	0.006929209000

Table 4Function **f(x) = x^2 - x - 2**,

Method	Interval	Root	Error	Iterations	TimeCPU
Hybrid1: Bisection-FalsePos	[1,5]	2.000000000000	0.000000000000	6	0.022261500000
Hybrid2: Trisection-FalsePos	[1,5]	1.999999999993	0.000000000022	6	0.010815083000
Hybrid3: Trisection-NewtonRaph	[1,5]	2.000000000005	0.000000000015	12	0.052815125000
Hybrid4: Quadsection-FalsePos	[1,5]	2.000000000000	0.000000000000	1	0.003457208000

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 CPU time. The dilemma is which algorithm will work satisfactorily on the average in the majority of 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 example, Tables 5,6,7,8,9 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 Tables 5,6,7,8,9.In all these examples, the algorithm 4 has the lowest number of iterations and lowest amount of CPU 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 upper bound is 40 for iterations and upto 10 decimal digits for approximation in the acceptable root.*

Table 5. Function f(x) = **0.986*x.^3 -5.181* x.^2 +9.067*x-5.289**

Method	Interval	Root	Error	Iterations	TimeCPU
Hybrid1: Bisection-FalsePos	[1,6]	1.929846242840	0.000000000001	10	0.005306959000
Hybrid2: Trisection-FalsePos	[1,6]	1.929846242844	0.000000000000	9	0.004571250000
Hybrid3: Trisection-NewtonRaph	[1,6]	1.929846242850	0.000000000000	11	-0.017164083000
Hybrid4: Quadsection-FalsePos	[1,6]	1.929846242848	0.000000000000	7	0.005307125000

Table 6. Function f(x) = **x^2 - x - 2**

Method	Interval	Root	Error	Iterations	TimeCPU
Hybrid1: Bisection-FalsePos	[1,7]	2.000000000000	0.000000000000	9	0.033662625000
Hybrid2: Trisection-FalsePos	[1,7]	2.000000000000	0.000000000000	7	0.013277458000
Hybrid3: Trisection-NewtonRaph	[1,7]	2.000000000000	0.000000000001	13	0.020348459000
Hybrid4: Quadsection-FalsePos	[1,7]	2.000000000000	0.000000000000	6	0.009984417000

Table 7. Function f(x) = **x^2 - 2**

Method	Interval	Root	Error	Iterations	TimeCPU
Hybrid1: Bisection-FalsePos	[1,8]	1.414213562373	0.000000000001	10	0.003816208000
Hybrid2: Trisection-FalsePos	[1,8]	1.414213562373	0.000000000001	7	0.002297375000
Hybrid3: Trisection-NewtonRaph	[1,8]	1.414213562373	0.000000000000	13	0.007534041000
Hybrid4: Quadsection-FalsePos	[1,8]	1.414213562373	0.000000000001	6	0.001220875000

Table 8. Function f(x) = **x-exp(-x)**

Method	Interval	Root	Error	Iterations	TimeCPU
Hybrid1: Bisection-FalsePos	[0,1]	0.567143290410	0.000000000001	5	0.027407458000
Hybrid2: Trisection-FalsePos	[0,1]	0.567143290410	0.000000000000	6	0.012605625000
Hybrid3: Trisection-NewtonRaph	[0,1]	0.567143290409	0.000000000001	11	0.027166625000
Hybrid4: Quadsection-FalsePos	[0,1]	0.567143290410	0.000000000000	5	0.011773583000

Table 9. Function f(x) = **x-cos(x)**

Method	Interval	Root	Error	Iterations	TimeCPU
Hybrid1: Bisection-FalsePos	[0,1]	0.739085133215	0.000000000000	7	0.015696125000
Hybrid2: Trisection-FalsePos	[0,1]	0.739085133215	0.000000000000	6	0.010849875000
Hybrid3: Trisection-NewtonRaph	[0,1]	0.739085133215	0.000000000001	11	0.019732500000
Hybrid4: Quadsection-FalsePos	[0,1]	0.739085133215	0.000000000000	4	0.015507833000

2.4Algorithms

2.4.1 Bisection, Regula Falsi, Newton algorithm

These algorithms are unbiquous standard [appendix section VI]. We will use Bisection and Regula Falsi methods for comparison analysis of hybrid algorithms. There are multiple reasons for neglecting Newton Raphson from this analysis: (a) 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. Also False Position method does not fit well in the category of Bisection, Triscetion and Quadsection, we will first compare these three algorithms.

2.4.2 Trisection Algorithm [Badr].

Bader [Badr] extended bisection algorithm to trisection in oreder to create a better algorithm to hybridize 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*, *enclosing 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
```

2.4.3 Quadsection Algorithm (new here)

In this paper, we adapted trisection algorithm to craft aQuadsection algorithm to hybridize, more heuristically, with false position method. Quadsection 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, enclosing interval,* and number of iterations *k*

```
for k=1 to n
p:= (3*a +b)/4; m := (2*a +2*b)/4; q := (a +3*b)/4;
   r=m;
    if f(a)*f(m)< 0
         b=m;   r=p;
         if f(a)*f(p)<0
              b=p
         else
             a=p
          endif
    else
         a=m;r=q;
         if f(a)*f(q)<0
              b=q
         else
             a=q
         endif
    endif
    if |f(r)| < eps
      return r, a,b, k;
    endif
endfor
```

The Trisection algorithm [Badr] and anew Quadsection algorithm are conceptually equivalent in each iteration. From the following tables, it is clear that Quadsection algorithm requires fewer iterations to converge. It shows that Quadsection algorithm competes successfully with the other algorithms with respect to number of loop iterations, CPU 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 upper bound is 40 for iterations and upto 10 decimal digits for approximation in the acceptable root.

Table 10. Function **x^2 - x - 2**

Method	Interval	Root	Error	Iterations	TimeCPU
Bisection	[1,6]	1.000000000000	0.000000000003	40	0.007535792000
Trisection	[1,6]	2.000000000000	0.000000000001	27	0.008417209000
Quadsection	[1,6]	2.000000000000	0.000000000001	21	0.011262208000

_

Table 11. Function **0.986*x.^3 -5.181* x.^2 +9.067*x-5.289**

Method	Interval	Root	Error	Iterations	TimeCPU
Bisection	[1,5]	1.929846242856	0.000000000001	37	0.008264875000

Trisection	[1,5]	1.929846242855	0.000000000001	24	0.009037791000
Quadsection	[1,5]	1.929846242841	0.000000000001	19	0.008326791000

Table 12. Function **exp(x).*(x-1)**

Method	Interval	Root	Error	Iterations	TimeCPU
Bisection	[1,4]	1.000000000000	0.000000000119	40	0.012896875000
Trisection	[1,4]	4.000000000000	2620.711201590	40	0.016012917000
Quadsection	[1,4]	1.000000000000	0.000000000000	1	0.012987334000

Table 13. Function**(x-1).*(x-2).*(x-3)**

Method	Interval	Root	Error	Iterations	TimeCPU
Bisection	[1,3]	2.000000000000	0.000000000000	1	0.005182542000
Trisection	[1,3]	2.000000000000	0.000000000000	26	0.003394209000
Quadsection	[1,3]	2.000000000000	0.000000000000	1	0.004560500000

Table 14. Function **x-cos(x)**

Method	Interval	Root	Error	Iterations	TimeCPU
Bisection	[0,1]	0.739085133215	0.000000000001	39	0.002875750000
Trisection	[0,1]	0.739085133215	0.000000000000	24	0.001766334000
Quadsection	[0,1]	0.739085133216	0.000000000001	20	0.002702500000

Table 15. Function **x-exp(-x)**

Method	Interval	Root	Error	Iterations	TimeCPU
Bisection	[0,1]	0.567143290409	0.000000000001	38	0.009934417000
Trisection	[0,1]	0.567143290410	0.000000000000	25	0.008384250000
Quadsection	[0,1]	0.567143290409	0.000000000001	19	0.013742375000

III. HYBRID ALGORITHMS

Now we present recent hybrid algorithms and a new proposed hybrid algorithm. The existing hybrid algorithms have one thing in common. At each iteration, they compute the bracketing interval for each of the two hybridizing 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 hybrid algorithm, namely, Hybrid1 [Sabh] based on classical Bisection and False Position algorithms. Since the classical algorithms can be found in any text book, those algorithms are not described here. For reference in hybrid algorithm, and for the sake of completeness, these algorithms are delegated to an appendix Section VI.

In Hybrid1 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: Hybrid2 [Badr] and Hybrid3 [Thota]. These algorithms lead the way for us to discover more heuristics to design a new blended algorithm Hybrid4 which is more efficient than the previous three hybrid algorithms. All the four algorithms are described here for reference.

Hybrid1: **Bisection and False Position Algorithm** [Sabh]

Input: f , [a, b], $\in_s$, maxIterations
Output: root r, k-number of iterations, error of approximation $\in_a$, 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 mid point the error
$m= \frac{a_k+b_k}{2}$, and $\in_m = |f(m)|$
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 $\in_p = |f(s)|$
if $|f(m)| < |f(s)|$,
f(m) is closer to zero, Bisection method determines bracketing interval $[ba_{k+1}, bb_{k+1}]$
r= m
$\in_a = \in_m$
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
$\in_a = \in_p$
if $f(a_k) \cdot f(r) > 0$,
$pa_{k+1} = r$; $pb_{k+1} = b_k$;
else
$pa_{k+1} = a_k$; $b_{k+1} = r$;
endif
endif
Since the root is bracketed by both $[ba_{k+1}, bb_{k+1}]$ and $[pa_{k+1}, pb_{k+1}]$, set
$[a_{k+1}, b_{k+1}] = [ba_{k+1}, bb_{k+1}] \cap [pa_{k+1}, pb_{k+1}]$ 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 = $\in_a = |f(r)|$
k = k + 1
until $|f(r)| < \in_s$ or k >maxIterations

Hybrid2: Trisection and False Position Algorithm [Badr]
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; 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)| <= 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)
```

Hybrid 3: Trisection and Newton-Raphson Algorithm [Thota]

This algorithm is along the same lines as Hybrid 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 hybridization 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 to coordinate the two algorithms to hybridize. At each iteration, they determine (1) the promising approximation root out of the two roots (2) the common interval bracketing the approximate root. In Hybrid1 and Hybrid 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 Hybrid3 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.

New Hybrid Algorithm

Hybrid4algorithm provides a more proficient approach to optimization: (1) Quadsection algorithm is used instead of bisection or trisection, (2) it eliminates the computation of common interval required by the foregoing algorithms used to hybridize, 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 [Occam], Fig 2. The Occam's razor principle is a heuristic, not a proof. That is, when faced with competing choices, we use the simplest from what we have. It will be shown that Occam's Razor Principle works quite well in this case.

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

Hybrid4. Quadsection and False Position methods

Input a_0, b_0, r_o, eps, imax, f
Output k, a_k, b_k, r_k
for k=1:imax
quadsection iteration step determines
qa_k, qb_{k-}, qr_{k-} from qa_{k-1},qb_{k-1}, qr_{k-1}
relable
qa_k, qb_k, qr_ktoa, b, r_-
False-position iteration step
input isa, b, r instead of oldpa_{k-1}, pb_{k-1}, pr_{k-1}
false position iteration step determines
pa_k,pb_k, pr_{k-} from a, b, rinstead of old pa_{k-1}, pb_{k-1}, pr_{k-1}
This makes [pa_k,pb_k] as the common interval [a,b] without any computation.
At the same time, using a, b, r instead of old pa_{k-1}, pb_{k-1}, pr_{k-1}, it makes this step more optimal
if $f(pr_k)$<eps
returnk,pa_k,pb_k, pr_k
end
relabel pa_k,pb_k, pr_k to qa_k, qb_{k-},qr_{k-}
endFor

Summarizing the foregoing algorithms, succinctly the *iteration step* in the algorithms are:

Hybrid1
[ba_k,bb_k,br_k]= Bisection(a_{k-1}, b_{k-1}, r_{k-1},f)
[pa_k,pb_k,pr_k]=FalsePosition(a_{k-1},b_{k-1}, r_{k-1},f)
The results of hybridization step are:
r_k is better of br_kpr_k,
[a_k,b_k] is common to [ba_k,bb_k], [pa_k,pb_k],
r_k belongs to [a_k,b_k]

Hybrid 2
[ta_k,tb_k,tr_k]=Trisection(a_{k-1},b_{k-1}, r_{k-1},f)
[pa_k,pb_k,pr_k]=FalsePosition(a_{k-1},b_{k-1}, r_{k-1},f)
The outcomes of hybridization are:
r_k is better of tr_kpr_k,
[a_k,b_k] is common to [ta_k,tb_k], [pa_k,pb_k],
r_k belongs to [a_k,b_k]

Hybrid3
[ta_k,tb_k,tr_k]=Trisection(a_{k-1},b_{k-1}, r_{k-1},f)
[na_k,nb_k,nr_k]=NewtonRaphson(a_{k-1},b_{k-1}, r_{k-1},f)
The upshot of hybridization is:
r_k as better of tr_knr_k,
[a_k,b_k] is common from [ba_k,bb_k], [pa_k,pb_k] and nr_k by analyzing { $f(ba_k)$, $f(bb_k)$, $f(na_k)$,$f(nb_k)$, $f(nr_k)$}}then finding from {{ ba_k,bb_k,pa_k,pb_k, nr_k}}.
r_k belongs to [a_k,b_k]

Hybrid4
[a,b,r]=Quadsection(a_{k-1},b_{k-1}, r_{k-1},f)
[pa_k,pb_k,pr_k]=FalsePosition(a,b,r,f)
The conclusion of hybridization is: there is no need to do any work calculation: [a_k,b_k] is the same interval [pa_k,pb_k] containingr_k is the same as pr_k, the desired root. This algorithm is optimal in 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 hybrid methods are compared. Error analysis is performed. It is validated here that Hybrid4 algorithm preferable than the existing algorithms Hybrid1, Hybrid2, and Hybrid3.

A. Empirical Evidence Testing

We have tested our new algorithm Hybrid4 with other hybrid algorithms on diverse examples found in articles in the literature to validate that new algorithm outperforms each of them.

B. Experiments in Matlab

Moaazzam [MOAA]used Mirosoft 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. Overall we have validated that the new algorithm performs better than all the hybrid algorithms. These tests indicate that hybrid with false position algorithm is still preferable to hybridization with Newton-Raphson possibly because the NR method requires that the function be differentiable.

For all these Tables 16-22, the functions frequently occurring in the literature. For all the functions the upper bound on the number of iterations is 40 and acceptable approximation error bound is 10^{-10}.These tables are for comparision of four algorithms number of iterations, CPU compute time, accuracy of the solution.

Table 16. Function **x-exp(-x)**

Method	Interval	Root	Error	Iterations	TimeCPU
Hybrid1: Bisection-FalsePos	[0,1]	0.567143290410	0.000000000001	5	0.023543625000
Hybrid2: Trisection-FalsePos	[0,1]	0.567143290410	0.000000000000	6	0.013356459000
Hybrid3: Trisection-NewtonRaph	[0,1]	0.567143290409	0.000000000001	11	0.027260750000
Hybrid4: Quadsection-FalsePos	[0,1]	0.567143290410	0.000000000000	5	0.011438583000

Table 17. Function **0.986*x.^3 -5.181* x.^2 +9.067*x-5.289**

Method	Interval	Root	Error	Iterations	TimeCPU
Hybrid1: Bisection-FalsePos	[1,5]	1.929846242844	0.000000000000	10	0.023045459000
Hybrid2: Trisection-FalsePos	[1,5]	1.929846242848	0.000000000000	9	0.012193917000
Hybrid3: Trisection-NewtonRaph	[1,5]	1.929846242850	0.000000000000	12	0.021333875000
Hybrid4: Quadsection-FalsePos	[1,5]	1.929846242848	0.000000000000	7	0.011543250000

Table 18. Function **(x-1).*(x-2).*(x-3)**

Method	Interval	Root	Error	Iterations	TimeCPU
Hybrid1: Bisection-FalsePos	[1,3]	1.000000000000	0.000000000000	1	0.005420333000
Hybrid2: Trisection-FalsePos	[1,3]	1.000000000000	0.000000000000	1	0.002555584000
Hybrid3: Trisection-NewtonR	[1,3]	1.000000000000	0.000000000000	2	0.018167000000
Hybrid4: Quadsection-FalsePos	[1,3]	1.000000000000	0.000000000000	1	0.002319417000

Table 19. Function **x^2 - x - 2**

Method	Interval	Root	Error	Iterations	TimeCPU
Hybrid1: Bisection-FalsePos	[1,6]	2.000000000000	0.000000000000	8	0.002879208000
Hybrid2: Trisection-FalsePos	[1,6]	2.000000000000	0.000000000000	8	0.003052041000
Hybrid3: Trisection-NewtonR	[1,6]	2.000000000000	0.000000000000	14	0.004807417000
Hybrid4: Quadsection-FalsePos	[1,6]	2.000000000000	0.000000000000	6	0.003639750000

Table 20. Function **x^2 - 2**

Method	Interval	Root	Error	Iterations	TimeCPU
Hybrid1: Bisection-FalsePos	[1,8]	1.414213562373	0.000000000001	10	0.007323334000
Hybrid2: Trisection-FalsePos	[1,8]	1.414213562373	0.000000000001	7	0.007474583000
Hybrid3: Trisection-NewtonRaph	[1,8]	1.414213562373	0.000000000000	13	0.008903292000
Hybrid4: Quadsection-FalsePos	[1,8]	1.414213562373	0.000000000001	6	0.006177333000

Table 21. Function **x-cos(x)**

Method	Interval	Root	Error	Iterations	TimeCPU
Hybrid1: Bisection-FalsePos	[0,1]	0.739085133215	0.000000000000	7	0.015696125000
Hybrid2: Trisection-FalsePos	[0,1]	0.739085133215	0.000000000000	6	0.010849875000
Hybrid3: Trisection-NewtonR	[0,1]	0.739085133215	0.000000000001	11	0.019732500000
Hybrid4: Quadsection-FalsePos	[0,1]	0.739085133215	0.000000000000	4	0.015507833000

Table 22. Function **exp(x).*(x-1)**

Method	Interval	Root	Error	Iterations	TimeCPU
Hybrid1: Bisection-FalsePos	[0,4]	1.000000000000	0.000000000000	9	0.029480000000
Hybrid2: Trisection-FalsePos	[0,4]	1.000000000000	0.000000000000	9	0.027570208000
Hybrid3: Trisection-NewtonR	[0,4]	0.000000000000	16.000000000000	40	0.030620416000
Hybrid4: Quadsection-FalsePos	[0,4]	1.000000000000	0.000000000000	1	0.012339708000

V. CONCLUSION

We have designed and implemented a new algorithm Hybrid4, a proficient algorithm hybrid of Quadsection and Regula Falsi methods. The algorithm was implemented in Matlab R2022A 64 bit (maci64) on MacBook Pro MacOS Sonoma 14.1.116GB Apple M1Pro. The implementation tests Tables16-22 indicate that Algorithm 4 outperforms all above cited algorithms all the time by a considerable margin. The experiments on numerous datasets used in the literature validate that the new algorithm is effective both conceptually and computationally. Thus, this paper provides a fastest algorithm to the repertoire of hybrid algorithms.

VI. APPENDIX

There is no universal algorithm optimal for root approximation that works on all the functions on all the domain intervals. We provide a summary of classical methods here for reference. In the paper, we have provided a new hybrid algorithm that is based on the classical methods and outperforms both the classical and hybrid methods.

There are four classical methods for finding roots of non-linear equations: Bisection, Regula Falsi, Newton-Raphson, Secant. For completeness, we describe these methods briefly for (1) root approximation, (2) error calculation, and (3) termination criteria. Then we use Occam's razor principle to select the optimal method for error calculation and termination criteria.

We constrain this discussion to finding a single root instead of all the roots of an equation. In case, an equation has several roots, we can delineate an interval where the desired root is to be found.

A. Bisection Method

The Bisection method is (1) based on binary chopping of irrelevant subintervals, (2) virtually binary search, and (3) guaranteed to converge to the root. Bisection method is static, the *length* of the subinterval at each iteration is independent of the function. No matter what the function is, the root-error upper bound is fixed at each iteration and can be determined a priori. By specifying the root-error tolerance, the upper bound on the number of iterations can be predetermined quickly[Mathews].

Problem If a function f:[a, b]→R (1) is continuous and (2) f(a) and f(b) are of opposite signs, i.e., $f(a)\bullet f(b) < 0$, then there exists a root r∈ [a, b] such that f(r) = 0.

Let $[a_1,b_1] = [a, b]$ be the initial interval of continuity of f. The first approximate root is

$r_1 = \frac{a_1+b_1}{2}$, middle point of the interval $[a_1,b_1]$

the actual root lies in the interval $[a_1, r_1]$ or $[r_1,b_1]$,

if $f(r_1)=0$, r_1 is the root.

if $f(a_1)$ and $f(r_1)$ are of the opposite sign, $f(a_1)\bullet f(r_1) < 0$, true root lies in $[a_1, r_1]$

if $f(r_1)$ and $f(b_1)$ are of the opposite sign, $f(r_1)\bullet f(b_1) < 0$, true root lies in $[r_1,b_1]$,

The new interval is denoted by $[a_2,b_2]$

At each iteration, new root and next sub-interval is generated.

In general, for each iteration k, the approximation

$r_k=\frac{a_k+b_k}{2}$ is middle point of $[a_k, b_k]$,

Either r_k is the root or

if $f(a_k)\bullet f(r_k) < 0$, the root lies in $[a_k, r_k]$

else if $f(r_k)\bullet f(b_k) < 0$, the root lies in $[r_k, b_k]$

Then the new interval is denoted by $[a_{k+1},b_{k+1}]$ with r_k as one of its end points

A.1 Advantages of Bisection Method

Since the method brackets the root, at each iteration the length of root-bracketing interval is scaled to half the length. Thus, the method guarantees the decrease in the error in the approximate root at each iteration. The convergence of Bisection method is certain as it is simply based on halving the bracketing interval containing the root.

A.2 Drawbacks of Bisection Method

Though the convergence of Bisection method is guaranteed, its rate of convergence is too slow and as such it is quite difficult to extend to use for systems of equations. If one of the initial endpoints is closer to the root, Bisection method does not take advantage of this information, it will take larger number of iterations to reach the root [Mathew].

B. False Position (Regula Falsi) Method[wiki], [Harder]

Motivation for innovative methods arises from the poor performance of Bisection method. The False Position method is known by various names: Double False Position, Regula Falsi method, linear interpolation method. It is a very old method for solving equations in one unknown. This method differs from Bisection in the way the estimates are calculated. False Position method is a dynamic method, it takes advantage of the location of the root to make a conceivably better appropriate selection. Unfortunately, this method is not satisfactory as expected, for all functions, see Figure 3,4,5,6.

Here also the function f:[a, b]→R (1) is continuous and (2) f(a) and f(b) are of opposite signs, i.e., $f(a)\bullet f(b)<0$. The algorithm uses a, b as the two initial estimates a_1, b_1 of the root. The False Position method uses two start values $r_o = a_1$, $r_1 = b_1$, to compute successive values

$$r_n = r_{n-1} - \frac{f(r_{n-1})(r_{n-1}-r_{n-2})}{f(r_{n-1})-f(r_{n-2})} \text{ for natural number } n \geq 2$$

the approximations, r_n , are ensured to be bracketed in $[a_n, b_n]$ depending on $f(a_{n-1})\bullet f(r_n)<0$ for $[a_{n-1}, r_n]$ $[r_n, b_{n-1}]$ depending on as in the case of Bisection method,

B.1 Justification: Error in Bisection method is straight forward; in each iteration root approximation error is halved, i.e. root approximation error in the nth step is no larger than $\frac{b-a}{2^n}$. This is not the case for *False Position* method. If a is sufficiently close to the root r, then f(a) is close to f(r) = 0 due to continuity of f. With step size h = r-a, slope of the secant line slope, $\frac{f(b)-f(a)}{b-a}$, is approximately equal to $\frac{f(b)}{b-r}$. The closer b is to r, the closer the secant line is to tangent, f'(r). Though, it is not required that f be differentiable. The closer b is to r, faster the convergence of iterations [Harder].

B.2 Advantages of False Position Method

It is guaranteed to converge due to decreasing length of root-bracketing interval. It is fast when you know the linear nature of the function.

B.3 Drawbacks of False Position Method

For False Position method, there is no way to tell the number of iterations needed for convergence. The False Position method is expected to be faster than Bisection method. If we cannot ensure that the function can be interpolated by a linear function, then applying the False Position method can result *in worse* results than the Bisection method. The problem occurs when the function is convex, concave up or concave down According to [10], for concave down function, left end point remains stationary and right end point updates in each iteration. For concave up function, right end point remains stationary and left end point updates in each iteration. This is not an accurate statement, it works some of the functions, not all the functions. Figures 3, 4, 5,6 contradict this statement. It depends on the convexity of the function, not concavity up or down. When the root is very close to the end points of the interval, convergence can become extremely slow. Visuals are helpful insight to comprehend the behavior of algorithms.

In these examples, four functions are used with same tolerance and the upper limit of 10 iterations for display purposes. The purpose is to show how the algorithms work for False Position method. In the interest of simplicity of plots, we terminated the algorithms before reaching the error-tolerance.

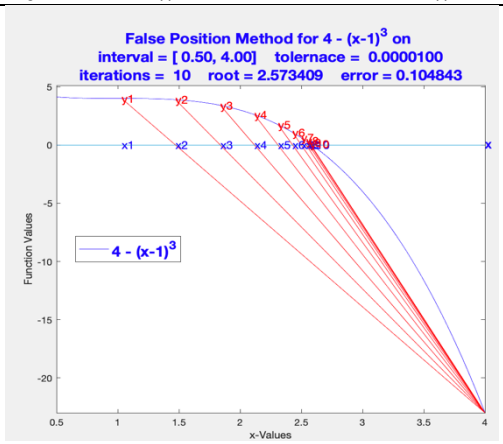

Figure 3. Convex Function Concave up, right end point fixed

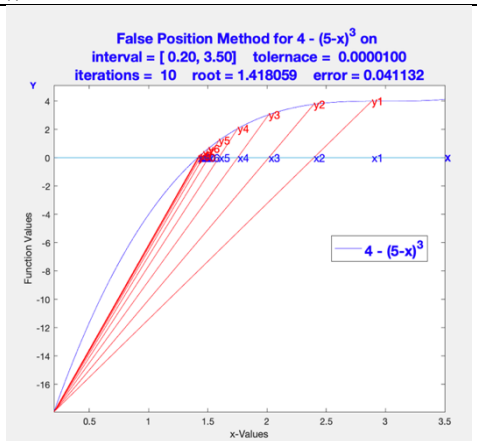

Figure 4. Convex Function Concave up, right end point fixed

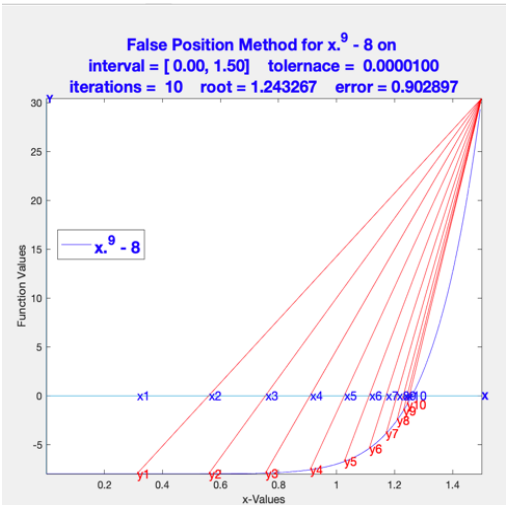

Figure 5. Convex Function Concave down, right end point fixed

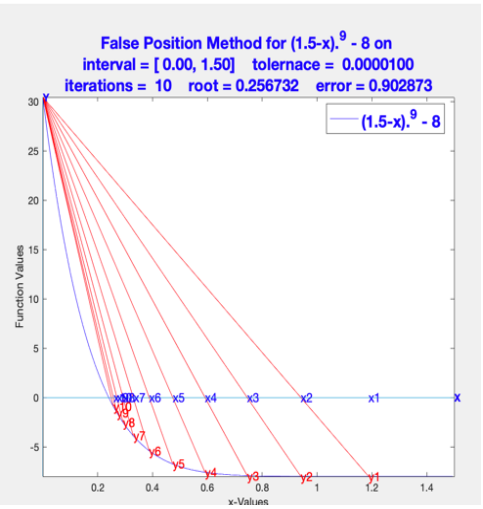

Figure 6. Convex Function Concave down, left end point fixed

C. Newton-Raphson Method

Newton-Raphson method is also called a fixed point iteration method. This method requires that the function be differentiable. If the function f(x) is differentiable on the domain of the function, and r_0 is initial guess, then the first approximation r_1 is defined by

$$r_1 = r_0 - f(r_0)/f'(r_0)$$

and successive approximations are

$$r_n = r_{n-1} - f(r_{n-1})/f'(r_{n-1}) \text{ for natural numbers } n \geq 2.$$

This method converges provided $|x - f(x)/f'(x)| < 1$ for x in the domain of definition. For functions where there is a singularity and it reverses sign at the singularity, Newton-Raphson method may converge on the singularity.

C.1 Advantages of the Newton-Raphson method

The convergence rate is linear and this method is very fast as compared to Bisection and False Position methods. If we know the *multiplicity*, m, of the root, it can be further improved with faster convergence to the root [3]. The updated iteration formula becomes,

$$r_n = r_{n-1} - m\, f(r_{n-1})/f'(r_{n-1}) \text{ for natural numbers } n \geq 1.$$

Note. If $f^{(k)}(x) = 0$, for integers $k < m$, then multiplicity of root is m.

C.2 Disadvantages of the Newton-Raphson method;

The only pitfall is that it fails if the derivative, f'(x), is near zero at some iteration. For example, Newton-Raphson method fails to compute $r_1$for $f(x) = x^2 - 1$ where $r_0 = 0$. But it does not create a problem in some cases where singularity in f'(x) cancel with f(x). for example, $f(x) = x^3$ with $r_0 = 0$ does result in $r_1 = 0$.

D. Secant Method, Modified Secant Method[EHI]

The Secant method also requires that the function be differentiable. When it is not easy to compute the derivative of the function, the Secant method approximates the derivative with the slope of a secant line. That is, in the absence of derivative of the function, Secant method is a modification of the Newton-Raphson method. In fact, it does not need differentiability as well as bracketing. It is similar to False Position method. The only difference is that False Position method ensures that approximations are bracketed and Secant method simply uses the last two values to approximate the tangent. Thus, the Secant method is not guaranteed to converge.

The convergence rate of the Secant method is super linear. Thus, the convergence rate is between that of the Bisection method and the Newton-Raphson's method. The Secant method requires two initial values, whereas Newton-Raphson required only one start values. If the function f(x) is differentiable, and r_0, r_1 are two is initial guess, then the approximations of the secant method can be written as

$r_n = r_{n-1} - \frac{f(r_{n-1})(r_{n-1}-r_{n-2})}{f(r_{n-1})-f(r_{n-2})}$ for natural numbers $n \geq 2$

For better estimate of the slope of the Secant line, we can define a small constant delta, δ, so that it can uniformly replace $r_{n-2}$with $r_{n--1} - \delta$ as

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

The success of this method is questionable if δ is not chosen sufficiently small.

D.1 Advantages and Shortcomings

It has the advantage for finding a bracketing-interval quickly where the root lies, but choice of delta must be made adaptively, else the algorithm runs the risk of missing the root.

Some researchers, experimented on $f(x) = x - \cos(x)$ on a close interval [0,1] and concluded that secant method is better than Bisection and Newton-Raphson method [Nayak], [srivastava]. It is not accurate to make a general conclusion statement from one function.

VII. REFERENCES

[1]. [Badr] Badr, E., Almotairi, S., & Ghamry, A. E. (2021). A comparative study among new hybrid root finding algorithms and traditional methods. Mathematics, 9(11), 1306.

[2]. [Calhoun] Donna Calhoun https://math.boisestate.edu/~calhoun/teaching/matlab-tutorials/lab_16/html/lab_16.html accessed December 2023

[3]. [Chapra] Steven C Chapra and Raymond P Canale, Numerical Methods for Engineers, 7th Edition, McGraw-Hill Publishers, 2015.

[4]. [Ehi] 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.

[5]. [Harder] Douglas Wilhelm Harder, https://ece.uwaterloo.ca/~dwharder/NumericalAnalysis/10RootFinding/falseposition/, accessed Jun 2019.

[6]. [Mathews] 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

[7]. [Moaa] G. Moazzam, A. Chakraborty, A. Bhuiyan: A robust method for solving transcendental equations. Int. J. Comput. Sci., 9 (2012), 413–419.

[8]. [Nayak] TanisthaNayak ,Tirtharaj Dash Solution To Quadratic Equation Using Genetic Algorithm , Proceedings Of National Conference On AIRES-2012, Andhra University Https://Arxiv.Org/Pdf/1306.4622

[9]. [Occam] Occam'Razor: https://conceptually.org/concepts/occams-razor

[10]. [Sabh] Sabharwal, C. L. (2019). Blended root finding algorithm outperforms bisection and regulafalsi algorithms. Mathematics, 7(11), 1118.

[11]. [Sapna] 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.

[12]. [Srivast] 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.

[13]. [Thinzar] 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.

[14]. [Thota] 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

[15]. [Traub] J. F. Traub: Iterative Methods for the Solution of Equations, Chelsea Publishing company, New York, NY, USA, 1982.

[16]. [Wiki] Wikipedia https://en.wikipedia.org/wiki/False_position_method#Two_historical_types accessed December 2023