Comparison between Newton-Raphson and Bisection Methods of Root-Finding Problems: A Review

A Review Article in

(Mathematical Physics)

Abstract

This review's goal is to the comparison between the output of the two numerical methods, the Newton-Raphson method, and the Bisection method, in terms of rate of convergence. Each method has its manual computational algorithm, which is used to manually solve a root-finding problem using a TI-inspire instrument; both methods converged on 1.56155 as the exact root, according to the calculations. With Newton-Raphson methods converged at the second iteration (2^{nd}) and Bisection methods converged at the fourteenth iteration (14^{th}). Using the software package MATLAB7.6, the Newton's and the Bisection method were both used to discover the root of the function f(x) = cosx-x exp(x) on the close interval [0, 1]. The Newton methods converge to the precise root of 0.5718 with an error of 0.0000 after the second (2^{nd}) iteration, while the Bisection method converges after the fourteenth (14^{th}) iteration. Also, the program Mathematica 9.0 was used to compare the effects by using Newton's method and the Bisection method to find the root of the function f(x) = x-cosx on the close interval [0, 1]. The Bisection method converges at the 52^{nd} iteration, whereas Newton's method converges to the precise root of 0.739085 with an error of 0.000000 at the 8^{th} iteration. The Newton-Raphson method was found to be more effective in computing and arranging the roots of a nonlinear equation than the Bisection method.

Keywords: Manual Computation; MATLAB Code; Mathematica 9.0; Newton-Raphson method; Algorithm; Bisection method.

1. Introduction

Natural equations are often nonlinear or linear since real-world problems are modeled into mathematical equations to answer a problem. The outcome of the problem is determined by the roots of the equations. This means that in mathematical computations, using the most effective numerical method is very significant for root-finding problems, because achieving a precise outcome is critical in problem-solving. The Root - Finding Problem is the problem of discovering a root of the equation $f(x) = 0$, where $f(x)$ is a function of the solitary variable x. Given a function $f(x) = 0$ and $x = \phi$ such that $f(\phi) = 0$, then ϕ is a root of $f(x)$. Root - Finding Problems arise in

many areas of studies including Chemistry, Engineering, Biosciences, Agriculture, and so on. This is since, in problem formulas involving real-life problems, unknown variables will often appear. Finding the quantized energy level of a constrained structure, the potential surface of a field, the equilibrium position of an object, are examples of relevant circumstances in Physics where such problems must be solved (Adegoke et al., 2018) (Ahmad and Technology, 2015).

The Bisection and Newton-Raphson are two numerical methods for solving root-finding problems. Convergence rates may be linear, quadratic, or non-linear. Several studies have been conducted by various scholars to discover the best strategies for resolving root–finding problems. Ehiwario et al(2014), for example, looked into the efficacy of Newton-Raphson, and Bisection in solving root-finding problems (Ehiwario and Aghamie, 2014).

Before the analysis by Ehiwario et al(2014), Srivastava et al(2011) compared the Bisection, Newton-Raphson, and Secant methods to determine which approach needed the fewest iterations when solving a single variable nonlinear equation (Srivastava et al., 2011). Other computational methods for solving root-finding problems, such as Regula–False and Fixed Point Iteration, have been used in previous studies to solve root-finding problems (Ebelechukwu et al., 2018, Ehiwario and Aghamie, 2014).

The review study's goal is to compare the number of iterations required by a given numerical method to find a solution, as well as the rate at which the methods converge for root-finding problems. This review article has been arranged such that; in section two both the Bisection and Newton-Raphson methods are illustrated in terms of their background and Algorithm and the rate at which the methods converge and the MATLAB Code of both Methods are showed. In section three, three examples are illustrated: In the first example Manual computational algorithms for each of these two methods are used to discover the root of the function and achieve the aim of this analysis, in the second example it did by Using The Software Package MATLAB 7.6, and the last example is solved by Using The Software, Mathematica 9.0. The outcomes were delivered to check the appropriateness of the best method. Newton’s method was the most robust for solving the nonlinear equation. Besides, Newton’s method gave a lesser number of iterations compared to the Bisection and it showed less processing time.

2. Materials and Methods

The following methods: Bisection and Newton-Raphson methods are discussed as a result of these works:

2.1 Bisection Method

Many methods are used for root-finding methods; one of these methods is the bisection method that can be used to discover the root of any continuous function with two opposite-sign values. The procedure involves repetitively bisecting the interval identified by these values and afterward choosing the subinterval where the function modifications sign, indicating that it necessarily has an origin. It's an easy and reliable method, but it's also very slow (Richard and Douglas, 1985). This method is also called Bolzano Method and the binary search method (Ahmad and Technology, 2015, Jamil, 2013), and the interval halving method (Bachrathy and Stépán, 2012).

2.1.1 The Method

The continuous function *f* should have at minimum one root in the range to fulfill the intermediate value theorem, so a, and b are said to bracket a root in this case (a, b). For the real variable *x*, the method can be used to the solution of the equation *f(x)* = 0. The equation, where *f* is a continuous function distinct on the interval [a, b], and the signs of *f (a)* and *f (b)* are opposed. The method divides the interval in half at each step by calculating the interval's midpoint *c* = *(a+b)* / 2 and the value of the function *f(c)* at that point. Unless *c* is a root (which is doubtful but not impossible), there are just two options: *f(a)* and *f(c)* bracket a root with opposite signs, or *f(b)* and *f(c)* bracket a root with opposite signs (Burden and Faires, 1985). The procedure selects the current interval to be utilized in the next phase as the subinterval that is certain to be a bracket. Each stage reduces the width of an interval with a zero of *f* by 50%, and the process repeats until the interval is small enough. As shown in figure 1

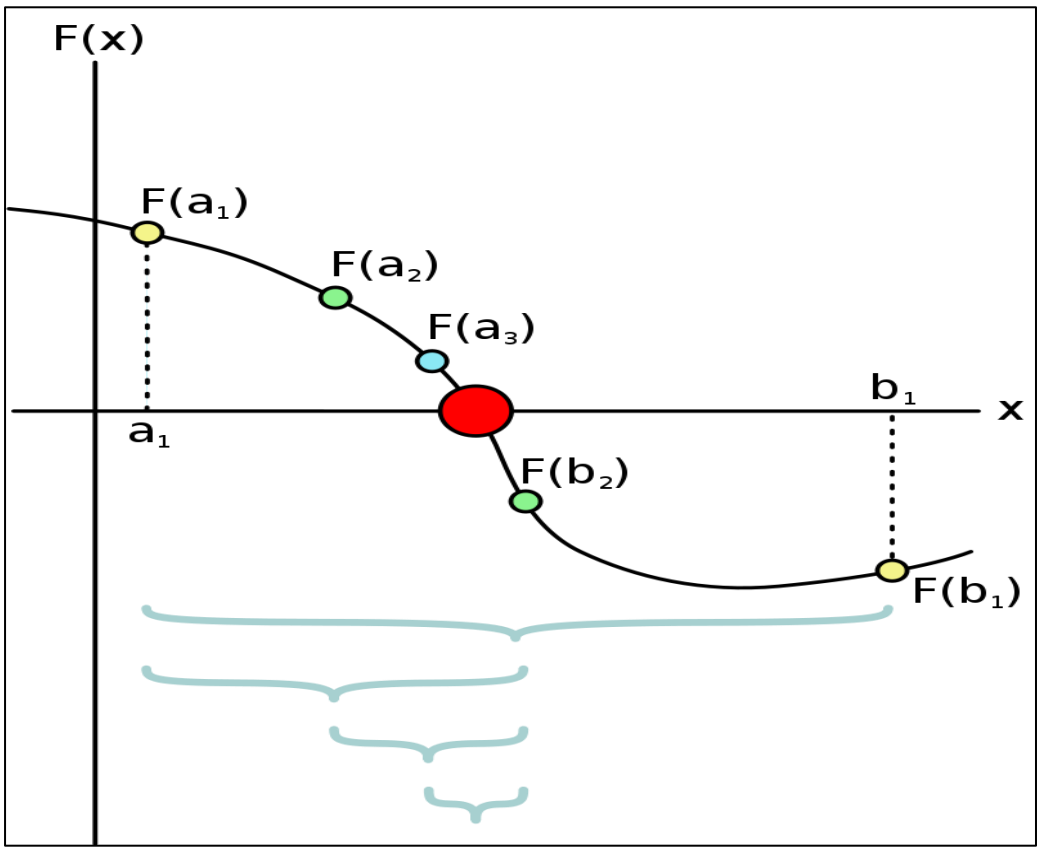

Figure 1: Over the starting range [a1; b1], a few steps of the bisection method were added with the larger red dot representing the function's root.

It sets c as the new value for b if the signs of*f(a)* and *f(c)* are differing, and c as the new *a* if the signs of *f(b)* and *f(c)* are opposed. (If *f(c) = 0*, *c* is assumed to be the solution, and the phase ends.) Since the new *f(a)* and *f(b)* have opposed signs in both cases, it can be applied to this smaller interval (Burden and Faires, 1985).

2.1.2 Background of Bisection Method

The concept of this method is based on Bolzano's theorem on continuity. Given a function *f(x)* = 0 such that its root lies in [*a*, *b*]. If *f*(*x*) is continuous and real and *f*(*a*). *f(b)* < 0,

then it is at minimum one root between *b* and *a*. This method is classified under bracketing methods because two initial estimates for the root are necessary. These estimates necessity "bracket," or be on whichever side of, the root, as the name implies. Different techniques are used in the methods outlined here to systematically minimize the width of the bracket and, as a result, zero in on the correct answer (Chapra and Canale, 2015).

2.1.3 Algorithm for Bisection Method

Step 1: Select *a* and *b* as the initial guesses such that $f(a)\, f(b) < 0$.

Step 2: Compute the midpoint of *a*, and *b* such that $x_m = \frac{a+b}{2}$

Step 3: Find $f(a)f(x_m)$ if $f(a)f(x_m) < 0$, then $a = a$; $b = x_m$ if $f(a)f(x_m) > 0$, then $a = x_m$; $b = b$

if $f(a)f(x_m) = 0$ then the root of the function = x_m

Step 4: Find $x_m = \frac{a+b}{2}$

Step 5: Go back to step 3 (Eiger et al., 1984) (Azure et al., 2019).

2.1.4 Convergence of Bisection Methods

In the case of the Bisection method, if there is a series y_n that converges to zero and a positive constant K, then assume that an algorithm creates iterates that converge as $\lim_{n\to\infty} x_n = \delta$ such that:

$$\epsilon_n = |x_n - \delta|$$

2.1.5 Bisection Methods Stopping Criteria

The conditions for stopping, as indicated by (Atkinson, 2008). Let *e* be the tolerance error, which means we want to get the root with an error of no more than e. Accept $x = c_k$ as the root of $f(x) = 0$ then. If one or more of the subsequent conditions are met,

i $|f(c_k)| \leq e$.

ii $|c_{k-1} - c_k / c_k| \leq e$.

iii $b - a / 2^k \leq e$.

iv The number of iterations *k* exceeds or equals a fixed number, such as *N*. (Azure et al., 2019).

2.1.6 MATLAB Code of Bisection Method

```
function root = bisection23(fname,a,b,delta,display)
% The bisection method.
%
%input: fname is a string that names the function f(x)
% a and b define an interval [a,b]
% delta is the tolerance
% display = 1 if step-by-step display is desired,
% = 0 otherwise
%output: root is the computed root of f(x)=0
%
fa = feval(fname,a);
fb = feval(fname,b);
if sign(fa)*sign(fb) > 0
    disp('function has the same sign at a and b')
    return
end
if fa == 0,
    root = a;
    return
end
if fb == 0
    root = b;
    return
end
c = (a+b)/2;
fc = feval(fname,c);
e_bound = abs(b-a)/2;
if display,
    disp(' ');
    disp(' a b c f(c) error_bound');
    disp(' ');
    disp([a b c fc e_bound])
    end
    while e_bound > delta
    if fc == 0,
    root = c;
    return
    end
    if sign(fa)*sign(fc) < 0 % a root exists in [a,c].
    b = c;
    fb=fc;
    else % a root exists in [c,b].
    a = c;
    fa = fc;
    end
    c = (a+b)/2;
    fc = feval(fname,c);
    e_bound = e_bound/2;
    if display, disp([a b c fc e_bound]), end
    end
    root = c;
```

Figure 2: represents the MATLAB code of the Bisection method (Ahmad and Technology, 2015).

2.1.7 The Major Benefit of Bisection Method

The bisection method has two main advantages. The first is that it is extremely durable. In a known number of iterations, it is guaranteed to find an approximation to the root within a specified degree of accuracy. To see this, notice that the solution is initially contained within an interval of size |b - a|. If the solution is not found in the second step, it is found in an interval of size |b - a|/2. The solution must be in an interval of size at the nth iteration.

$$|b^{(n)} - a^{(n)}| = |b - a|/ 2^{n-1}.$$

As a result, if our error tolerance is δ, we can be certain that we will be within that tolerance.

$$n = \log_2 (b - a/\delta) + 1 \text{ iterations.}$$

It can thus be used to find the roots of non-smooth functions. While this may be advantageous in some situations, the fact that the method does not use the derivatives of the function means that it does not take advantage of any knowledge about the function's curvature to aid in finding the root. This is what makes the bisection method sluggish in comparison to other methods that take advantage of the curvature of functions (Kopecky, 2007).

2.2 Newton-Raphson Method

2.2.1 Background of Newton-Raphson method

This method can be deduced by a variety of methods, the most notable of which is the definition of finding the slope of a function and the Taylor series.

2.2.2 Concept of Slope of a Function

Derivation of the this method which is based on the principle of slope starts with the premise that a tangent to a curve that cuts the x-axis provides an idea, which leads to the computation of likely solutions to the curve's equation, as shown in the diagram below.

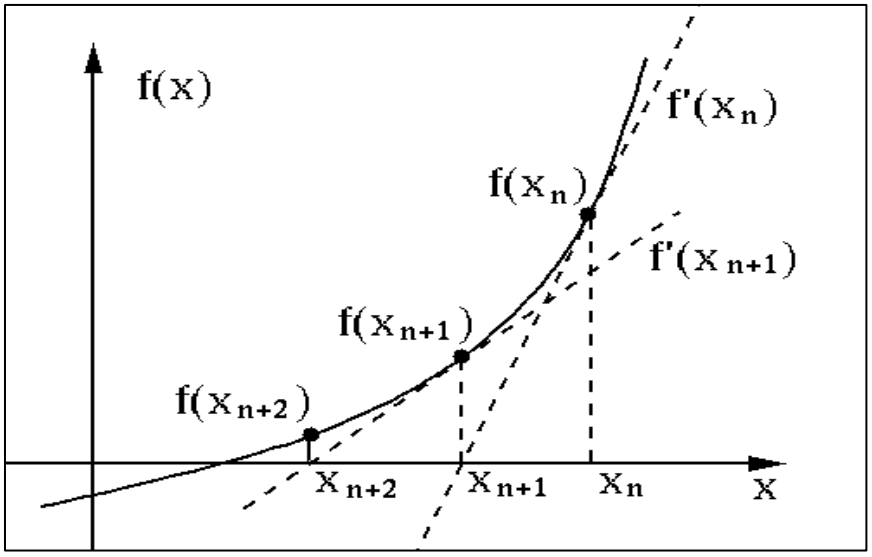

Figure 3: Diagram indicating the slope of a curve

From the figure above, the slope of (x_n, $f(x_n)$) is given by:

$$f'(x_n) = \frac{f(x_n) - 0}{x_{n-x_{n+1}}} \quad (1)$$

Making x_{i+1} the subject in equation (1) yields the Newton-Raphson equation, which is shown below.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad (2)$$

The method makes certain that each iterative solution is modified at every point (Azure et al., 2019).

2.2.3 Algorithm for Newton-Raphson Method

To find the root of the equation using Newton's method:

Step 1: Write $f(x) = 0$ is form of the equation. And find $f'(x)$

Step 2: Using the function $f(x)$, test values of x to find the range(x_0, x_1) where the real root is located.

Step 3: Select x_0 in step 2 as the initial value and solve for $f(x_0)$ and $f'(x_0)$

Step 4: Use values obtained in step 3 to compute h_0 that is $h_0 = \frac{f(x_0)}{f'(x_0)}$ and hence $x_1 = x_0 - h_0$

Step 5: Use the new x_1 value to repeat the process in steps 3 and 4 to obtain the value for x_2. Stop when values for x_{n+1} in two different iterations are the same. That value is the real root of the function (Abbasbandy and computation, 2003).

2.2.4 Convergence of Newton-Raphson Method

Assuming that x_1, x_2, x_3 … xi is a sequence of approximations to a root δ obtained by a numerical method, where $\lim_{n\to\infty} |x_n - \delta| = 0$

$$\epsilon_n = x_n - \delta \quad \text{if} \lim_{n\to\infty} \frac{|\epsilon_{n+1}|}{|\epsilon_n|^p} = k$$

for some p and some non-zero constant K, the numerical method has an order of convergence p and K defines the asymptotic error constant. The value of p is directly proportional to the rate of convergence. Discarding ϵ_n^3 and the higher power of n, we get and the higher power of n, we get

$$\epsilon_{n+1} = k\epsilon_n^2 \quad (3)$$

Where $K = \frac{1}{2}\frac{f''(\delta)}{f'(\delta)}$, Since $\frac{|\epsilon_{n+1}|}{|\epsilon_n|^2} = \frac{|f''(\delta)|}{2|f'(\delta)|}$

By taking a limit as n → ∞ in the case of Newton's method, assume that the function $f(x)$ can be differentiated twice, and δ is a root of the function. Substituting

$\epsilon_n + \delta = x_n$ into the function $f(xn)$ and expanding $f(\delta + \epsilon_n)$ and $f'(\delta + \epsilon_n)$ using Taylor's series about the point δ where $f(\delta) = 0$ we get

$$\epsilon_{n+1} = \epsilon_n - \frac{\left[\epsilon_n f'(\delta) + \frac{1}{2}\epsilon_n^2 f''(\delta) + \cdots\right]}{f'(\delta) + \epsilon_n f''(\delta) + \cdots}$$

$$\epsilon_{n+1} = \epsilon_n - \frac{\left[\epsilon_n + \frac{1}{2}\epsilon_n^2 f''(\delta) + \cdots\right]}{f'(\delta) + \epsilon_n f''(\delta) + \cdots}$$

$$\epsilon_{n+1} = \frac{1}{2}\frac{f''(\delta)}{f'(\delta)}\epsilon_n^2 + O(\epsilon_n^3)$$

$$\lim_{n\to\infty} \frac{|\epsilon_{n+1}|}{|\epsilon_n|^2} = \lim_{n\to\infty} \frac{|f''(\delta)|}{2|f'(\delta)|}$$

If $x_n \to \delta$ as $n \to \infty$. Hence for the root δ, the Newton-Raphson method has second-order - quadratic convergence and an asymptotic error constant of $\frac{|f''(\delta)|}{2|f'(\delta)|}$. This means that the square of the previous error $_n$ is proportional to the subsequent error ϵ_{n+1} (Azure et al., 2019).

2.2.5 Termination Condition for Newton-Raphson Method

The iterative process of Newton's method is terminated at the point when approximate relative error ϵ_a is less than a certain threshold. Hence the relative error formula is given by

$$\epsilon_a = \frac{x_{n-1} - x_i}{x_{n+1}} \times 100 \qquad (4)$$

2.2.6 MATLAB code of Newton's Raphson

```
function root=newton24(fname,fdname,x,xtol,ftol,n_max,display)
% Newton's method.
%
%input:
% fname string that names the function f(x).
% fdname string that names the derivative f'(x).
% x the initial point
% xtol and ftol termination tolerances
% n_max the maximum number of iteration
% display = 1 if step-by-step display is desired,
% = 0 otherwise
%output: root is the computed root of f(x)=0
%
n = 0;
fx = feval(fname,x);
if display,
    disp(' n x f(x)')
    disp('------------------------------------')
    disp(sprintf('%4d %23.4e %23.4e', n, x, fx))
end
if abs(fx) <= xtol
    root = x;
    return
end
for n = 1:n_max
    fdx = feval(fdname,x);
    d = fx/fdx;
    x = x - d;
    fx = feval(fname,x);
    if display,
        disp(sprintf('%4d %23.4e %23.4e',n,x,fx))
    end
    if abs(d) <= xtol | abs(fx) <= ftol
        root = x;
        return
    end
end
```

Figure 4: Represents the MATLAB code of the Newton-Raphson Method (Ahmad and Technology, 2015).

2.2.7 The Major Benefit of Newton-Raphson Methods

The main advantage of Newton over bisection is the significant increase in the convergence rate. As Newton works, he works at a breakneck pace. However, this increase in productivity does not come without a price. To use Newton's method at all, the function must be differentiable and the derivative must be given. The function can be differentiable, but computing the derivative, i.e. the Jacobean matrix of partial derivatives, can be a nightmare. Human error becomes a very real possibility. It is possible to substitute a finite difference approximation for the analytical Jacobean, but this results in a slower overall rate of convergence (Kopecky, 2007).

3. Some Examples

In the following section some examples are solved to compare both the Bisection and Newton-Raphson Methods:

3.1 Manual Computational

Example: The solution of the equation $x^2 + x - 4 = 0$ using Bisection method, Newton-Raphson method is computed using the preamble which starts with writing the given equation in form $f(x) = 0$ as shown below (Azure et al., 2019).

$$f(x) = x^2 + x - 4 \quad (5)$$

This implies that

$$f'(x) = 2x + 1 \quad (6)$$

Testing for the possible roots of equation (5), we have $f(1) = -2$, $f(1.5) = -0.25$, $f(1.6) = 0.16$, implying that the real root of equation (1) lies between 1.5 and 1.6, which are adopted as initial guesses for the rest of the computations

Table 1: Root of Equation (5) Using Bisection Method.

N	a_n (- ve)	b_n (+ ve)	$X_{n+1}=\frac{a_n+b_n}{2}$	$f(x_{n+1})$
0	1.5	1.6	1.55	-0.0475
1	1.55	1.6	1.575	0.055625
2	1.55	1.575	1.5625	0.003906
3	1.55	1.5625	1.5525	-0.037244
4	1.5525	1.5625	1.5525	-0.016694
5	1.5575	1.5625	1.56	-0.0064
6	1.56	1.5625	1.56125	-0.0001248
7	1.56125	1.5625	1.56188	-0.001349
8	1.56125	1.56188	1.56157	-0.000071
9	1.56125	1.56157	1.56141	-0.000589
10	1.56141	1.56157	1.56149	-0.000259
11	1.56149	1.56157	1.56153	-0.000094
12	1.56153	1.56157	1.56155	-0.000012
13	1.56155	1.56157	1.56156	-0.00003
14	1.56155	1.56156	1.56156	-0.00003

Table 1 shows the effects by using the Bisection approach to solve a nonlinear equation (5). It produced an exact root of 1.56156 as the final production, but it converged after the 14th iteration.

Table 2: Root of Equation (5) Using Newton Method.

N	x_n	$f(x_n)$	$f'(x_n)$	$h_n = -\frac{f(x_n)}{f'(x_n)}$	$X_{n+1} = x_n + h_n$
1	1.5	-0.25	4	0.00625	1.5625
2	1.5625	0.003906	4.125	-0.000947	1.56155
3	1.56155	-0.000012	4.1231	0.000003	1.56155

Table 2 shows the output results of equation (5). When the Newton-Raphson method is used, converged to a precise solution of 1.56155 after the 2nd iteration.

3.2 Using the Software Package MATLAB 7.6

Example: solving $f(x) = cosx - xexp(x) = 0$ at [0, 1] using Bisection and Newton Raphson method with help of the MATLAB Code (Ahmad and Technology, 2015).

1. Using Bisection Method

Solution: Input from the MATLAB command window.

```
>> fname = @(x)(cos(x)-x*exp(x));
>> a=0;
>> b=1;
>> display=1;
>> delta=10^-4;
>> root = bisection23(fname,a,b,delta,display)
```

Table 3: Root of $f(x) = cosx - x * exp(x)$ using Bisection methods

n	a	b	c	f(c)	error - bound
1	0	1.0000	500×10^{-3}	53.2×10^{-3}	500×10^{-3}
2	500×10^{-3}	1.0000	750×10^{-3}	-856.1×10^{-3}	250×10^{-3}
3	500×10^{-3}	750×10^{-3}	625×10^{-3}	-356.7×10^{-3}	125×10^{-3}
4	500×10^{-3}	625×10^{-3}	562.5×10^{-3}	-141.3×10^{-3}	62.5×10^{-3}
5	500×10^{-3}	562.5×10^{-3}	531.3×10^{-3}	-41.5×10^{-3}	31.3×10^{-3}
6	500×10^{-3}	531.3×10^{-3}	515.6×10^{-3}	6.5×10^{-3}	15.6×10^{-3}
7	515.6×10^{-3}	531.3×10^{-3}	523.4×10^{-3}	-17.4×10^{-3}	7.8×10^{-3}
8	515.6×10^{-3}	523.4×10^{-3}	519.5×10^{-3}	-5.4×10^{-3}	3.9×10^{-3}
9	515.6×10^{-3}	519.5×10^{-3}	517.6×10^{-3}	$0.55x\ 10^{-3}$	2×10^{-3}
10	517.6×10^{-3}	519.5×10^{-3}	518.6×10^{-3}	-2.4×10^{-3}	1×10^{-3}
11	517.6×10^{-3}	518.6×10^{-3}	518.1×10^{-3}	-0.9×10^{-3}	0.5×10^{-3}
12	517.6×10^{-3}	518.1×10^{-3}	517.8×10^{-3}	-0.2×10^{-3}	0.2×10^{-3}
13	517.6×10^{-3}	517.8×10^{-3}	517.7×10^{-3}	0.2×10^{-3}	0.1×10^{-3}

14 517.7×10^{-3} 517.8×10^{-3} 517.8×10^{-3} -0×10^{-3} 0.1×10^{-3}

Root = 0.5178

2. Using Newton-Raphson Methods

Solution: Input from the MATLAB command window.

```
>> fname = @(x)(cos(x)-x*exp(x));
>> fdname = @(x)(-sin(x)-(x*exp(x)+exp(x)));
>> x=0.5;
>> xtol=10^-4;
>> ftol=10^-4;
>> n_max=10;
>> display=1;
>> root=newton24(fname,fdname,x,xtol,ftol,n_max,display)
```

Table 4: Root of f(x) = cosx−x∗ exp(x) using Newton-Raphson method

n	x	f(x)
0	5.0000e-001	5.3222e-002
1	5.1803e-001	-8.1744e-004
2	5.1776e-001	-1.8378e-007

Root = 0.5178

3.3 Using the Software, Mathematica 9.0

Example: Using the program Mathematica9.0, the Bisection and Newton-Raphson methods were performed to a single-variable function: *f (x) x - cosx* on [0, 1]. Tables 3 to 4 present the findings (Ehiwario and Aghamie, 2014).

Table 5: Bisection Methods Iteration Data

Steps	A	Function values	B	Function values
0	0	-1	10×10^{-1}	4.59698×10^{-1}
1	5×10^{-1}	-3.77583×10^{-1}	10×10^{-1}	4.59698×10^{-1}
2	5×10^{-1}	-3.77583×10^{-1}	7.5×10^{-1}	0.183111×10^{-1}

3	6.25×10^{-1}	-1.85963×10^{-1}	7.5×10^{-1}	0.183111×10^{-1}
4	6.875×10^{-1}	-0.853349×10^{-1}	7.5×10^{-1}	0.183111×10^{-1}
5	7.1875×10^{-1}	-0.338794×10^{-1}	7.5×10^{-1}	0.183111×10^{-1}
6	7.34375×10^{-1}	-0.787473×10^{-2}	7.5×10^{-1}	0.183111×10^{-1}
7	7.34375×10^{-1}	-0.787473×10^{-2}	7.421875×10^{-1}	0.519571×10^{-2}
8	7.3828125×10^{-1}	-0.134515×10^{-2}	7.421875×10^{-1}	0.519571×10^{-2}
9	7.3828125×10^{-1}	-0.134515×10^{-2}	$7.40234375 \times 10^{-1}$	0.192387×10^{-2}
10	7.3828125×10^{-1}	-0.134515×10^{-2}	$7.392578125 \times 10^{-1}$	0.289009×10^{-3}
11	$7.3876953125 \times 10^{-1}$	-0.528158×10^{-3}	$7.392578125 \times 10^{-1}$	0.289009×10^{-3}
12	$7.39013671875 \times 10^{-1}$	-0.119597×10^{-3}	$7.392578125 \times 10^{-1}$	0.289009×10^{-3}
13	$7.39013671875 \times 10^{-1}$	-0.119597×10^{-3}	$7.391357421875 \times 10^{-1}$	0.847007×10^{-4}
14	$7.3907470703125 \times 10^{-1}$	-0.174493×10^{-4}	$7.391357421875 \times 10^{-1}$	0.847007×10^{-4}
15	$7.3907470703125 \times 10^{-1}$	-0.174493×10^{-4}	$7.39105224609375 \times 10^{-1}$	0.336253×10^{-4}
16	$7.3907470703125 \times 10^{-1}$	-0.174493×10^{-4}	$7.390899658203125 \times 10^{-1}$	0.808791×10^{-5}
17	$7.390823364257813 \times 10^{-1}$	-0.468074×10^{-5}	$7.390899658203125 \times 10^{-1}$	0.808791×10^{-5}
18	$7.390823364257813 \times 10^{-1}$	-0.468074×10^{-5}	$7.390861511230469 \times 10^{-1}$	0.170358×10^{-5}
19	$7.390842437744141 \times 10^{-1}$	-0.148858×10^{-5}	$7.390861511230469 \times 10^{-1}$	0.170358×10^{-5}
20	$7.390842437744141 \times 10^{-1}$	-0.148858×10^{-5}	$7.390851974487305 \times 10^{-1}$	0.107502×10^{-6}
21	$7.390847206115723 \times 10^{-1}$	-0.690538×10^{-6}	$7.390851974487305 \times 10^{-1}$	0.107502×10^{-6}
22	$7.390849590301514 \times 10^{-1}$	-0.291518×10^{-6}	$7.390851974487305 \times 10^{-1}$	0.107502×10^{-6}
23	$7.390850782394409 \times 10^{-1}$	-0.92008×10^{-7}	$7.390851974487305 \times 10^{-1}$	0.107502×10^{-6}
24	$7.390850782394409 \times 10^{-1}$	-0.92008×10^{-7}	$7.390851378440857 \times 10^{-1}$	0.774702×10^{-8}
25	$7.390851080417633 \times 10^{-1}$	-0.421305×10^{-7}	$7.390851378440857 \times 10^{-1}$	0.774702×10^{-8}
26	$7.390851229429245 \times$	$-0.171917 \times 10^{-}$	$7.390851378440857 \times$	0.774702×10^{-8}

	10^{-1}	^{7}	10^{-1}	
27	7.390851303935051 x 10^{-1}	-0.472236 x 10^{-8}	7.390851378440857 x 10^{-1}	0.774702 x 10^{-8}
28	7.390851303935051 x 10^{-1}	-0.472236 x 10^{-8}	7.390851341187954 x 10^{-1}	0.151233 x 10^{-8}
29	7.390851322561502 x 10^{-1}	-0.160501 x 10^{-8}	7.390851341187954 x 10^{-1}	0.151233 x 10^{-8}
30	7.390851331874728 x 10^{-1}	-0.463387 x 10^{-10}	7.390851341187954 x 10^{-1}	0.151233 x 10^{-8}
31	7.390851331874728 x 10^{-1}	-0.463387 x 10^{-10}	7.390851336531341 x 10^{-1}	0.732998 x 10^{-9}
32	7.390851331874728 x 10^{-1}	-0.463387 x 10^{-10}	7.390851334203035 x 10^{-1}	0.343329 x 10^{-9}
33	7.390851331874728 x 10^{-1}	-0.463387 x 10^{-10}	7.390851333038881 x 10^{-1}	0.148495 x 10^{-9}
34	7.390851331874728 x 10^{-1}	-0.463387 x 10^{-10}	7.390851332456805 x 10^{-1}	0.510784 x 10^{-10}
35	7.390851331874728 x 10^{-1}	-0.463387 x 10^{-10}	7.390851332165767 x10^{-1}	0.236988 x 10^{-11}
36	7.390851332020247 x 10^{-1}	-0.219844 x 10^{-10}	7.390851332165767 x 10^{-1}	0.236988 x 10^{-11}
37	7.390851332093007 x 10^{-1}	-0.980727 x 10^{-11}	7.390851332165767 x 10^{-1}	0.236988 x 10^{-11}
38	7.390851332129387 x 10^{-1}	-0.371869 x 10^{-11}	7.390851332165767 x 10^{-1}	0.236988 x 10^{-11}
39	7.390851332147577 x 10^{-1}	-0.67446 x 10^{-12}	7.390851332165767 x 10^{-1}	0.236988 x 10^{-11}
40	7.390851332147577 x 10^{-1}	-0.67446 x 10^{-12}	7.390851332156672 x 10^{-1}	0.847655 x 10^{-12}
41	7.390851332147577 x 10^{-1}	-0.67446 x 10^{-12}	7.390851332152124 x 10^{-1}	0.865974 x 10^{-13}
42	7.390851332149850 x 10^{-1}	-0.293876 x 10^{-12}	7.390851332152124 x 10^{-1}	0.865974 x 10^{-13}
43	7.390851332150987 x 10^{-1}	-0.103584 x 10^{-12}	7.390851332152124 x 10^{-1}	0.865974 x 10^{-13}
44	7.390851332151556 x 10^{-1}	-0.854872 x 10^{-14}	7.390851332152124 x 10^{-1}	0.865974 x 10^{-13}
45	7.390851332151556 x 10^{-1}	-0.854872 x 10^{-14}	7.390851332151840 x 10^{-1}	0.390799 x 10^{-13}
46	7.390851332151556 x 10^{-1}	-0.854872 x 10^{-14}	7.390851332151698 x 10^{-1}	0.153211 x 10^{-13}
47	7.390851332151556 x 10^{-1}	-0.854872 x 10^{-14}	7.390851332151627 x 10^{-1}	0.344169 x 10^{-14}
48	7.390851332151591 x 10^{-1}	-0.255351 x 10^{-14}	7.390851332151627 x 10^{-1}	0.344169 x 10^{-14}
49	7.390851332151591 x	-0.255351 x 10^{-}	7.390851332151609 x	0.444089 x 10^{-}

	10^{-1}	^{14}	10^{-1}	^{15}
50	7.390851332151600 x 10^{-1}	-0.111022 x 10^{-14}	7.390851332151609 x 10^{-1}	0.444089 x 10^{-15}
51	7.390851332151605 x 10^{-1}	-0.333067 x 10^{-15}	7.390851332151609 x 10^{-1}	0.444089 x 10^{-15}
52	7.390851332151607 x 10^{-1}	0	7.390851332151609 x 10^{-1}	0

Table 5 displays the iteration data obtained using Mathematica 9.0 for the Bisection process. The function $f(x) = x\text{-}cosx = 0$ at the interval [0,1] converges to 0.7390851332151607 with an error level of 0.000000 at the 52th iterations using the Bisection method, as shown in Table 5.

Table 6: Newton's Method with x_o= 0.5 iterations.

Steps	x_k	$f(x_k+1)$
1	$5x10^{-1}$	$-96.2771x10^{-1}$
2	$-96.2771x10^{-1}$	$-24.3009x10^{-1}$
3	$-24.3009x10^{-1}$	$23.9002x10^{-1}$
4	$23.9002x10^{-1}$	$5.35581x10^{-1}$
5	$5.35581x10^{-1}$	$7.50361x10^{-1}$
6	$7.50361x10^{-1}$	$7.39113x10^{-1}$
7	$7.39113x10^{-1}$	$7.39085x10^{-1}$
8	$7.39085x10^{-1}$	$7.39085x10^{-1}$

Table 6 discovered that with x_o = 0.5, the function converges to 0.739085 with error 0.000000 after 8th iteration.

4. Result and Discussion

In the manual computational algorithm to find the solution of the equation $x^2 + x - 4 = 0$, both methods converged on 1.56155 as the exact root, the Newton-Raphson methods converged at the second iteration while the Bisection method converged at the fourteenth iteration. To find the root of $f(x) = cosx - x\ exp(x)$ by using the software package MATLAB7.6, the Newton methods converge to the precise root of 0.5718 with an error of 0.0000 after the second iteration, while the Bisection method converges after the fourteenth iteration. Also, for the $f(x) = x - cosx$ by using the Mathematica 9.0 the Bisection method converges at the 52nd iteration, whereas Newton's method converges to the precise root of 0.739085 at the 8th iteration. According to these results of both methods in terms of order of convergence, Newton's method is appropriately the more effective method.

5. Conclusion

Based on the results of both methods (Newton-Raphson and the Bisection methods), In terms of order of convergence, Newton's method is properly the most efficient as compared to the

Bisection method. (Newton method will converge faster to the solution of the function compared to the Bisection method), the number of iterations necessary by each method to conclude demonstrated this. Since the root of the interval is bracketed in the Bisection method, the method is certain to converge; however, it is extremely slow. This is because it converges at a rate similar to Newton's method, however only involves a solitary function assessment per iteration. In comparison to Newton's Raphson method, it was also concluded that while Bisection's convergence is assured, its rate of convergence is very slow, making it hard to expand to use for systems of equations.

6. References

ABBASBANDY, S. J. A. M. & COMPUTATION 2003. Improving Newton–Raphson method for nonlinear equations by modified Adomian decomposition method. 145**,** 887-893.

ADEGOKE, T., ADEGOKE, G., ODUWOLE, H. & YAHYA, A. 2018. Comparative Study of Some Numerical Iterations Using Zero Truncated Poisson Distribution.

AHMAD, A. G. J. I. J. O. M. T. & TECHNOLOGY 2015. Comparative Study of Bisection and Newton-Raphson Methods of Root-Finding Problems. 19.

ATKINSON, K. E. 2008. *An introduction to numerical analysis*, John wiley & sons.

AZURE, I., ALOLIGA, G. & DOABIL, L. J. M. L. 2019. Comparative study of numerical methods for solving non-linear equations using manual computation. 5**,** 41-46.

BACHRATHY, D. & STÉPÁN, G. J. P. P. M. E. 2012. Bisection method in higher dimensions and the efficiency number. 56**,** 81-86.

BURDEN, R. L. & FAIRES, J. D. J. N. A. 1985. 2.1 The bisection algorithm. 46-52.

CHAPRA, S. C. & CANALE, R. P. 2015. Numerical Methods for Engineers, ISBN: 978 0073397924. McGraw-Hill Publishers.

EBELECHUKWU, O. C., JOHNSON, B. O., MICHAEL, A. I., FIDELIS, A. T. J. I. J. O. T. & MATHEMATICS, A. 2018. Comparison of Some Iterative Methods of Solving Nonlinear Equations. 4**,** 22.

EHIWARIO, J. & AGHAMIE, S. J. I. J. O. E. 2014. Comparative study of bisection, Newton-Raphson, and secant methods of root-finding problems. 4**,** 01-07.

EIGER, A., SIKORSKI, K. & STENGER, F. J. A. T. O. M. S. 1984. A bisection method for systems of nonlinear equations. 10**,** 367-377.

JAMIL, N. J. I. J. E. S. 2013. A comparison of iterative methods for the solution of non-linear systems of equations. 3**,** 119-130.

KOPECKY, K. A. J. E. 2007. Root-finding methods. 613**,** 614.

RICHARD, B. & DOUGLAS, J. 1985. The bisection algorithm. *Numerical Analysis.* PWS Publishers Boston.

SRIVASTAVA, R., SRIVASTAVA, S. J. J. O. C., BIOLOGICAL & SCIENCES, P. 2011. Comparison of Numerical Rate of Convergence of Bisection, Newton-Raphson's and Secant Methods. 2**,** 472.