An Analysis of Solutions of Nonlinear Equations Using AI Inspired Mathematical Packages

Isaac Azure

Department of Computer Science, Regentropfen College of Applied Sciences, Bolgatanga, Ghana

Email address:
azureike@yahoo.com

Abstract: In the era of Artificial Intelligence (AI), achieving precise solutions for nonlinear equations has been considerably streamlined, thanks to the advancement of various mathematical tools designed for numerical computations. However, as the utilization of these mathematical software continues to rise, researchers are keen to ascertain the optimal choice among these tools based on their outcome when applied to solving nonlinear equations. This study addresses this question by undertaking a comparative analysis of three prominent mathematical software packages Python, Scilab, and MATLAB using two numerical approaches: Newton-Raphson and Secant. By employing the Newton-Raphson and Secant methods to solve five benchmark problems, this paper assesses the performance of the aforementioned mathematical tools. Notably, the outcomes underscore the competence of all three software options in yielding suitable approximations of the problem's root solutions. In particular, Python stands out for its ability to achieve this while utilizing the fewest iterations and minimizing computational time. As a result, among the three tools investigated, Python emerges as the most favorable choice, considering its efficiency and accuracy. Furthermore, this research validates the robustness of the Newton-Raphson approach over the Secant method, given its capability to efficiently converge to the solutions with the minimal iteration count across the benchmark problems. This finding highlights the superiority of the Newton-Raphson method as a more efficient and reliable technique for solving the considered benchmark problems.

Keywords: Nonlinear Equations, Artificial Intelligence (AI), MATLAB, SCILAB, Python, Secant Method, Newton Raphson Method

1. Introduction

Finding solutions to mathematical problems necessitates the use of the most accurate and robust methods, and now, with the presence of Artificial Intelligence (IA), there are several software tools capable of solving these mathematics problems with ease provided the appropriate code is designed for the tool in question. The following are some of the most important considerations when using mathematical software to solve a mathematical problem:

1. How well is the software able to give the right answer to the problem under consideration?
2. In the case of numerical methods, the interest will be to know how many iterations are needed to arrive at a solution using the given mathematical software?
3. What is the computational time required to reach solution using a given method with the help of the mathematical software?

The difficulties of mathematics are simply due to the wide number of problems that may be modeled into mathematical problems, each of which requires a different strategy to solve. A linear equation, a nonlinear equation, a system of linear equations, or a system of nonlinear equations will invariably come from the equations formulated as a result of modeling a real-life problem [2, 6]. To solve, each of these equations will necessitate the use of the most appropriate mathematical approach and instrument.

The nonlinear equation was the focus of this study. Nonlinear equations are mathematical equations that have at least one nonlinear term, making them more difficult to solve than linear equations. When solving nonlinear equations, which can be highly complicated, numerical methods are used, which normally require an initial guess [3, 16].

A related study examined five numerical approaches for

solving non-linear equations after their answers were manually obtained. The Bisection method, Newton Raphson method, Regula-Falsi method, Secant method, and Fixed-Point Iteration method were all compared. The researchers created a manual computing algorithm for each method and used it to solve a root-finding problem manually using a TI-inspire instrument. All methods converged to an exact solution, but the Bisection method converged at the 14th iteration, the Fixed Point Iterative Method at the 7th iteration, the Secant method at the 5th iteration, and the Regula Falsi and Newton Raphson methods at the 2nd iteration [2, 16].

In another related study, the problem of finding roots of nonlinear equations, which arose in a variety of practical applications in science and engineering, was thoroughly examined. The method of locating a root is known as root-finding, and the value of x that fulfills f(x) = 0 is known as a root of f(x) = 0. The research compared the rate of convergence of two common root-finding methods, Bisection and Newton-Rhapson. The authors utilized MATLAB software to locate the root of a particular function and compare the results of the two methods. The article concludes that Newton's approach is more successful than the Bisection method [1].

Another study presented four numerical methods which were applied to solve nonlinear equations and the Newton Raphson method was recommended as the best method for solving the nonlinear equations of the form $f(x) = 0$ because of its high rate of convergence [5].

In all of the above researches carried out, the MATLAB software was used to estimate the root of nonlinear equations. This research aimed at comparing the solutions of nonlinear equations using the Python, Scilab and MATLAB software.

According to Downey, A. B., Python is a widely-used high-level programming language known for its simplicity, readability, and versatility [4]. It was created by Guido van Rossum and first released in 1991. Over the years, Python has evolved into a powerful language with a large and active community of developers. Python's origin can be traced back to the late 1980s when Guido van Rossum began working on a new programming language during Christmas holidays in December 1989. He named it "Python" after the British comedy series "Monty Python's Flying Circus" [14]. With the many literature around the Python software, it was prudent to find out how well this software can perform compared to others such as MATLAB and Scilab.

On the other hand, Scilab is an open-source numerical computing software package that provides a powerful environment for scientific and engineering computations [17]. It is often used for tasks such as mathematical modeling, simulation, data analysis, and visualization [20]. Scilab was developed to provide a free and open alternative to commercial numerical computing environments like MATLAB [13, 17]. Though the Scilab is an open-source mathematical software, many researchers prefer using other software and this research sought to find out how accurate it is in computing the roots of a nonlinear equation.

Unlike the Python software, the MATLAB and Scilab software have a lot of similarities in their features. MATrix LABoratory, commonly known as MATLAB is a powerful and widely used high-level programming language and numerical computing environment [10]. It was initially developed in the late 1970s by Cleve Moler, a professor of mathematics, as a tool to help his students access mathematical and matrix computations more easily. Since then, MATLAB has evolved into an indispensable tool for engineers, scientists, researchers, and educators across various disciplines. MATLAB is designed to facilitate numerical computations and data analysis. It excels in handling matrix operations, linear algebra, optimization, statistics, and other mathematical tasks, making it particularly valuable for solving complex mathematical problems [7, 8, 18].

As this study seek to compare the performance of Python, Scilab and MATLAB with respect to solving the roots of nonlinear equations, a related paper compared the performance and features of the two high-level numerical computing and modeling software environments: the commercial MATLAB and the freeware Scilab. The motivation for the work was to compare these tools for educational use at the college and university level, but with a perspective to their professional and scientific use as well. The paper aimed to provide an objective performance comparison of the two tools and help the reader to choose between them. The paper also provided a benchmarking methodology and original benchmarks to compare the performances of both calculation tools [12].

2. Objectives of the Study

i. To compare the solutions of nonlinear equations using different mathematical tools (software).

ii. To assess the speed with which the selected mathematical tools produce results.

3. Methodology

Two key things were at the core of this research; firstly, the numerical methods under consideration and secondly the mathematical tool(s) were used for the computation. This section gives an overview of numerical methods and mathematical tools that were adopted for this study.

3.1. Numerical Methods

For the purpose of this study, the Secant method and Newton Raphson method were adopted for find the roots of the bench mark nonlinear equations using different mathematical tools.

3.1.1. Newton Raphson Method

The Newton-Raphson method is an iterative numerical technique used to find the roots of a real-valued function. It is based on the idea of approximating the function by its tangent line at an initial guess and then finding the x-intercept of that tangent line as an improved approximation

of the root [11, 15].

Theorem 1: Convergence of the Newton-Raphson Method

Suppose $f(x)$ is a continuous function and $f'(x)$ is continuously differentiable in an open interval containing the root r. If $f(r) = 0$ and $f'(r) \neq 0$, then the Newton-Raphson method converges to r quadratically [19].

Let x_n be the n th approximation obtained from the Newton-Raphson method

r be the exact root

$f(x)$ be the error function

For each iteration, the error function $f(x)$ is approximated by its tangent line at x_n

That is;

$$y = f(x_n) + f'(x_n).(x - x_n) \qquad (1)$$

At x intercept of the tangent line, $y = 0, hence\ set\ x = x_{n+1}$

$$\Rightarrow 0 = f(x_n) + f'(x_n).(x_{n+1} - x_n) \qquad (2)$$

Solving for x_{n+1} in equation (2) above gives the formula for the Newton Raphson method

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \qquad (3)$$

Algorithm

Step 1: Find points $a\ and\ b$ such that $a < b$ and $f(a).f(b) < 0$

Step 2: Take the initial $[a, b]$ and find the next $x_0 = \frac{a+b}{2}$

Step 3: Find $f(x_0)$ and $f'(x_0)$

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

Step 4: If $f(x_1) = 0$ then x_1 is an exact root else $x_0 = x_1$

Step 5: Repeat steps 3 and 4 until $f(x_i) = 0$ and $|f(x_i)| \leq Accuracy$

The above algorithm illustrates the procedure involved in computing the root of a nonlinear equation manually using the Newton Raphson method.

3.1.2. The Secant Method

The Secant Method is an iterative numerical method used to approximate the root of a real-valued function. It is an improvement over the Bisection method as it does not require an initial interval with opposite signs. Instead, it approximates the root using two initial points on the graph of the function [9, 11].

Theorem 2: Convergence of the Secant Method

Suppose $f(x)$ is a continuous function and $f(x)$ is continuously differentiable in an open interval containing the root r. If $f(r) = 0$ and the initial approximations x_0 and x_1 are sufficiently close to r, then the Secant method generates a sequence $\{x_n\}$ that converges to r quadratically [19].

Let x_n be the n th approximation obtained from the Newton-Raphson method

r be the exact root

$f(x)$ be the error function

For each iteration, the error function $f(x)$ is approximated by its tangent line at x_n

The secant method aims to find the root of the function $f(x)$ by using the secant line that passes through the two points $((x_{n-1} - f(x_{n-1}))$ and $(x_n, f(x_n))$ on the graph of $f(x)$. The equation of the secant line is given by:

$$y = f(x_{n-1}) + \frac{f(x_n).f(x_{n-1})}{x_n - x_{n-1}}.(x - x_{n-1}) \qquad (4)$$

Since we want to find the value of x when $y = 0$ (i.e. the $x\ intercept\ of\ the\ secant\ line$) we set $y = 0$ and solve for x:

$$0 = f(x_{n-1}) + \frac{f(x_n).f(x_{n-1})}{x_n - x_{n-1}}.(x - x_{n-1}) \qquad (5)$$

Simplifying the above equation gives us the formula for x_{n+1} which represents the secant method formula

$$y = x_n + \frac{f(x_n).f(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})} \qquad (6)$$

Algorithm
1. Choose two initial approximations x_0 and x_1 such that $x_0 \neq x_1$.
2. Evaluate the function at the initial approximations: $f(x_0)$ and $f(x_1)$.
3. Calculate the next approximation x_2 using the Secant method formula:

$$x_2 = x_1 - \frac{f(x_1).f(x_1 - x_0)}{f(x_1) - f(x_0)}$$

4. Set $x_0 = x_1$ and $x_1 = x_2$.
5. Repeat steps 2 to 4 until the desired level of accuracy is achieved.

3.2. Mathematical Tools (Software) Inspired by Artificial Intelligence (AI)

The study considered three common mathematical software mostly used for the computation of solutions of mathematical problems namely; Python, Scilab and Matlab. The 2023 versions of these software were adopted for the study. Codes were written for each of the software to solve the bench mark nonlinear equations questions listed below:

Problem 1: $f(x) = x^3 - x - 1$; $x_0 = 1, x_1 = 2$
Problem 2: $f(x) = 2x^3 - 2x - 5$; $x_0 = 1, x_1 = 2$
Problem 3: $f(x) = x^3 + 2x^2 + x - 1$; $x_0 = 0, x_1 = 1$
Problem 4: $f(x) = 2\cos(x) - x$; $x_0 = 1, x_1 = 2$
Problem 5: $f(x) = 2x^2 \cos(x) + 5\sin(x) + 3$; $x_0 = -1, x_1 = 0$

3.2.1. Python Code for Newton Raphson Method

```
import time
def F(x):
    return x**3 - x - 1
def dF(x):
    return 3*x**2 - 1
def newton_raphson(x0, tol=1e-6, max_iter=1000):
    start_time = time.time()

    x_n = x0
    iterations = 0
```

```
    while abs(F(x_n)) > tol and iterations < max_iter:
      x_n = x_n - F(x_n) / dF(x_n)
      iterations += 1

    end_time = time.time()
    computational_time = end_time - start_time

    if iterations == max_iter:
      print("Newton-Raphson method did not converge
within the maximum number of iterations.")
      return None, None
    else:
      print("Root:", x_n)
      print("Number of Iterations:", iterations)
      print("Computational        Time        (seconds):",
computational_time)
      return x_n, iterations

  if __name__ == "__main__":
    x0 = 1
    x1 = 2
    root, num_iterations = newton_raphson(x0)
```

3.2.2. Scilab Code for Newton Raphson Method

```
  function y = F(x)
    y = x^3 - x - 1;
  endfunction

  function dy = dF(x)
    dy = 3*x^2 - 1;
  endfunction

  function [root, num_iterations] = newton_raphson(x0, tol,
max_iter)
    tic();

    x_n = x0;
    iterations = 0;

    while abs(F(x_n)) > tol & iterations < max_iter
      x_n = x_n - F(x_n) / dF(x_n);
      iterations = iterations + 1;
    end

    computational_time = toc();

    if iterations == max_iter
      disp("Newton-Raphson method did not converge
within the maximum number of iterations.");
      root = NaN;
    else
      disp("Root:");
      disp(x_n);
      disp("Number of Iterations:");
      disp(iterations);
      disp("Computational Time (seconds):");
```

```
      disp(computational_time);
      root = x_n;
    end
  endfunction

  x0 = 1;
  tolerance = 1e-6;
  max_iterations = 1000;

  [root, num_iterations] = newton_raphson(x0, tolerance,
max_iterations);
```

3.2.3. Python Code for Secant Method

```
  import time
  def F(x):
    return x**3 - x - 1
  def secant_method(F, x0, x1, tol, max_iter):
    iter_count = 0
    start_time = time.time()

    while abs(x1 - x0) >= tol and iter_count < max_iter:
      x_next = x1 - F(x1) * (x1 - x0) / (F(x1) - F(x0))
      x0 = x1
      x1 = x_next
      iter_count += 1

    elapsed_time = time.time() - start_time

    if abs(x1 - x0) < tol:
      root = x1
    else:
      root = None

    return root, iter_count, elapsed_time

  # Initial values
  x0 = 1
  x1 = 2
  tolerance = 1e-6
  max_iterations = 100

  # Find the root using the secant method and measure
computational time
  root, iter_count, elapsed_time = secant_method(F, x0, x1,
tolerance, max_iterations)

  # Display the result
  if root is not None:
    print(f"Root: {root:.6f}")
    print(f"F(Root): {F(root):.6f}")
    print(f"Number of Iterations: {iter_count}")
    print(f"Computational        Time        (seconds):
{elapsed_time:.6f}")
  else:
    print("Secant method did not converge within the
specified maximum iterations.")
```

3.2.4. Scilab Code for Secant Mehtod

```
function y = F(x)
    y = x^3 - x - 1;
endfunction

function [root, iter_count, elapsed_time] =
secant_method(x0, x1, tol, max_iter)
    iter_count = 0;
    tic();

    while abs(x1 - x0) >= tol && iter_count < max_iter
        x_next = x1 - F(x1) * (x1 - x0) / (F(x1) - F(x0));
        x0 = x1;
        x1 = x_next;
        iter_count = iter_count + 1;
    end

    elapsed_time = toc();

    if abs(x1 - x0) < tol
        root = x1;
    else
        root = nan;
    end
endfunction

// Initial values
x0 = 1;
x1 = 2;
tolerance = 1e-6;
max_iterations = 100;

// Find the root using the secant method and measure
computational time
[root, iter_count, elapsed_time] = secant_method(x0, x1,
tolerance, max_iterations);

// Display the result
disp("Root: " + string(root));
disp("F(Root): " + string(F(root)));
disp("Number of Iterations: " + string(iter_count));
disp("Computational    Time    (seconds):    "    +
string(elapsed_time));
```

3.2.5. Matlab Code for Secant Method

```
% Initial values
x0 = 1;
x1 = 2;
tolerance = 1e-6;
max_iterations = 100;

% Find the root using the secant method and measure
computational time
[root, iter_count, elapsed_time] = secant_method(@F, x0,
x1, tolerance, max_iterations);
```

```
% Display the result
fprintf('Root: %.6f\n', root);
fprintf('F(Root): %.6f\n', F(root));
fprintf('Number of Iterations: %d\n', iter_count);
fprintf('Computational    Time    (seconds):    %.6f\n',
elapsed_time);


function y = F(x)
    y = x^3 - x - 1;
end

function [root, iter_count, elapsed_time] =
secant_method(F, x0, x1, tol, max_iter)
    iter_count = 0;
    tic();

    while abs(x1 - x0) >= tol && iter_count < max_iter
        x_next = x1 - F(x1) * (x1 - x0) / (F(x1) - F(x0));
        x0 = x1;
        x1 = x_next;
        iter_count = iter_count + 1;
    end

    elapsed_time = toc();

    if abs(x1 - x0) < tol
        root = x1;
    else
        root = NaN;
    end
end
```

3.2.6. MATLAB Code for Newton-Raphson Method

```
function y = F(x)
    y = x^3 - x - 1;
end

function dy = dF(x)
    dy = 3*x^2 - 1;
end

function [root, num_iterations] = newton_raphson(x0, tol,
max_iter)
    tic;

    x_n = x0;
    iterations = 0;

    while abs(F(x_n)) > tol && iterations < max_iter
        x_n = x_n - F(x_n) / dF(x_n);
        iterations = iterations + 1;
    end

    computational_time = toc;

    if iterations == max_iter
```

```
    disp('Newton-Raphson method did not converge
within the maximum number of iterations.');
    root = NaN;
  else
    disp('Root:');
    disp(x_n);
    disp('Number of Iterations:');
    disp(iterations);
    disp('Computational Time (seconds):');
    disp(computational_time);
    root = x_n;
  end
end

x0 = 1;
tolerance = 1e-6;
max_iterations = 1000;

[root, num_iterations]  =  newton_raphson(x0, tolerance,
max_iterations);
```

4. Results and Discussions

The results from this research were obtained using a computer with the following specification:
1. Processor: Intel® Core™ i5 – 3427U CPU®1.80GHz 2.30GHz
2. Installed RAM: 8.00GB (787 GB usable)
3. System type: 64 – bit operating system, 64 – based processor

A summary of results is displayed in the tables 1.0 and 2.0 below, with Table 1.0 showing the numerical results of roots of nonlinear equations using the Secant method with the help of the Python, Scilab and MATLAB softwares. Included in the tables are results of manual computations of the same nonlinear equations, the time required for by each software to produce answers and the number of iterations needed to arrive at an approximated solution(root).

The results in Table 1.0 below showed that given the bench mark problems considered in this study, the Python, Scilab, MATLAB and manual computation will produce the same estimated root and number of iterations for each problem. However, the computational time made the difference between the three mathematical tools. In Table 1.0 the Secant method was used and the computational time required to reach solution for the Python software was approximately zero for all the bench mark problems under consideration while that of the Scilab and MATLAB varied. For example, from the table, the Python software produced result in zero second for Problem 1 while the Scilab amd MATLAB produced results in 0.0021639 seconds and 0.002297seconds respectively for the same problem.

In the case of Problem 2, the Python software solved it in zero seconds while MATLAB and Scilab solved the same problem in 0.000402 seconds and 0.0010286 seconds respectively. An observation from Table 1.0 shows that the Scilab software was able to solve problems 1, 3 and 5 with less computational time compared to MATLAB while MATLAB on the other hand was able to solve problems 2 and 3 with less computational time compared to Scilab.

Another observation made from the Table 1.0 is that Problems 1, 2 and 3 which are algebraic in nature where all solved in seven iterations while that of Problems 4 and 5 which are trigonometric in nature where solved in 4 and 5 iterations respectively.

Table 2.0 is a summary of results when the Newton-Raphson method was applied to solve the bench mark problems with the help of Python, Scilab and Matlab softwares. Data in Table 2.0 showed that all three mathematical tools where able to solve the bench mark problems accurately. The Python software recorded the least number of iterations for Problems 1 and 2 while Scilab and MATLAB recorded the same number of iterations for all five problems. The Python softeware had the least computational time of approximately 0 seconds. MATLAB solved Problems 1, 2 and 4 faster than Scilab while Scilab was able to solve Problems 3 and 5 faster than MATLAB.

Comparing the results in Table 1.0 and Table 2.0 it is observed that the Newton-Raphson method solved the bench mark problems faster than the Secant method.

Table 1. Results of mathematical softwares using the secant method.

Problem	Solution Approach	Approximated Root	Number of Iterations	Computational Time	Initial Guess Values
Problem 1	Manual Computation	1.32471	7		(1, 2)
	Python	1.324718	7	0.000000	(1, 2)
	Scilab	1.324718	7	0.0021639	(1, 2)
	Matlab	1.324718	7	0.002297	(1, 2)
Problem 2	Manual Computation	1.6006	7		(1, 2)
	Python	1.600599	7	0.000000	(1, 2)
	Scilab	1.6005985	7	0.0010286	(1, 2)
	Matlab	1.600599	7	0.000402	(1, 2)
Problem 3	Manual Computation	0.4655	7		(0, 1)
	Python	0.465571	7	0.000000	(0, 1)
	Scilab	0.4655712	7	0.0005137	(0, 1)
	Matlab	0.465571	7	0.001279	(0, 1)
Problem 4	Manual Computation	1.0299	4		(0, 1)
	Python	1.029867	4	0.000000	(1, 2)
	Scilab	1.029867	4	0.0014337	(1, 2)
	Matlab	1.029867	4	0.000883	(1, 2)

Problem	Solution Approach	Approximated Root	Number of Iterations	Computational Time	Initial Guess Values
Problem 5	Manual Computation	-0.9421	5		(-1, 0)
	Python	-0.942076	5	0.000000	(-1, 0)
	Scilab	-0.942076	5	0.0005727	(-1, 0)
	Matlab	-0.942076	5	0.001345	(-1, 0)

Table 2. Results of mathematical softwares using the newton raphson method.

Problem	Solution Approach	Approximated Root	Number of Iterations	Computational Time	Initial Guess Value
Problem 1	Manual Computation	1.3247	4		1.5
	Python	1.324718	3	0.000000	1.5
	Scilab	1.324718	4	0.0050916	1.5
	Matlab	1.324718	4	0.002314	1.5
Problem 2	Manual Computation	1.6006	4		1.5
	Python	1.600599	3	0.000000	1.5
	Scilab	1.600599	5	0.0004007	1.5
	Matlab	1.600599	5	0.000156	1.5
Problem 3	Manual Computation	0.4656	3		0.5
	Python	0.465571	3	0.000000	0.5
	Scilab	0.465571	3	0.0006141	0.5
	Matlab	0.465571	3	0.003154	0.5
Problem 4	Manual Computation	1.0299	4		0.5
	Python	1.029867	4	0.0000000	0.5
	Scilab	1.029867	4	0.0007773	0.5
	Matlab	1.029867	4	0.000312	0.5
Problem 5	Manual Computation	-0.9421	3		-0.5
	Python	-0.942076	3	0.000000	-0.5
	Scilab	-0.942076	3	0.0004461	-0.5
	Matlab	-0.942076	3	0.001140	-0.5

5. Conclusion and Recommendations

From the results obtained in this research, it can be concluded that the three mathematical tools namely; Python, Scilab and MATLAB gave an accurate estimation of the roots of the nonlinear equations. The algorithms for both the Secant and Newton-Raphson methods considered in this study worked perfectly well using the mathematical tools, with the Newton-Raphson method recording the least number of iterations and the least computational time. This is indeed an indication that the Newton-Raphson method is more robust compared to the Secant method.

Another interesting discovery in this research is that Python, Scilab and MATLAB are able to solve algebraic nonlinear equations faster than that of trigonometric nonlinear equations.

A comparison of the three mathematical tools showed that the Python software required approximately zero seconds to solve any of the bench mark problems using either the Secant or Newton Raphson methods. Though all the three mathematical tools solved problems with the same number of iterations, Python was able to solve Problem 1 and Problem 2 with the least number of iterations which signifies its ability solve nonlinear equations with the least computational time and number of iterations. It can therefore be concluded that based on the bench mark problems considered in this research, the Python software is the most recommended for the estimation of the roots of nonlinear equations.

References

[1] Ahmad, A. G. (2015). Comparative Study of Bisection and Newton-Raphson Methods of Root-Finding Problems. *International Journal of Mathematics Trends and Technology, 19 (2)*.

[2] Azure, I., Aloliga, G., & Doabil, L. (2019). Comparative Study of Numerical Methods for Solving Non-linear Equations Using Manual Computation. *Mathematics Letters, 5 (4)*, 41-46. doi: 10.11648/j.ml.20190504.11.

[3] Biswa, N. D. (2012). Lecture Notes on Numerical Solution of Root-Finding Problems MATH 435.

[4] Downey, A. B. (2015). *Think Python: How to Think Like a Computer Scientist*. Green Tea Press. Available online: http://greenteapress.com/thinkpython2/html/index.html

[5] Ebelechukwu, O. C., Johnson, B. O., Michael, A. I., & Fidelis, A. T. (2018). Comparison of Some Iterative Methods of Solving Nonlinear Equations. *International Journal of Theoretical and Applied Mathematics, 4 (2)*, 22.

[6] Ehiwario, J. C., & Aghamie, S. O. (2014). Comparative Study of Bisection, Newton-Raphson and Secant Methods of Root-Finding Problems. *IOSR Journal of Engineering (IOSRJEN), 4 (04)*.

[7] Hanselman, D. C., & Littlefield, B. L. (2018). *The Art of MATLAB*. Cambridge University Press.

[8] Hahn, B., & Valentine, D. T. (2020). *Essential MATLAB for Engineers and Scientists*. Academic Press.

[9]   Kazemi, M., Deep, A., & Nieto, J. (2023). An existence result with numerical solution of nonlinear fractional integral equations. *Mathematical Methods in the Applied Sciences*.

[10]   King, A. P., & Aljabar, P. (2017). *MATLAB Programming for Biomedical Engineers and Scientists*. Academic Press.

[11]   Mahdy, A. M. S. (2022). A numerical method for solving the nonlinear equations of Emden-Fowler models. *Journal of Ocean Engineering and Science*.

[12]   Mikac, M., Logožar, R., & Horvatić, M. (2022). Performance Comparison of Open Source and Commercial Computing Tools in Educational and Other Use—Scilab vs. MATLAB. *Tehnički glasnik, 16 (4)*, 509-518.

[13]   Nagar, S. (2021). *Introduction to Scilab*. Notion Press.

[14]   Python Software Foundation. (2021). *Python 3.10.0 Documentation*. Retrieved from https://docs.python.org/3/

[15]   RASHEED, M., Rashid, A., Rashid, T., Hamed, S. H. A., & AL-Farttoosi, O. A. A. (2021). Application of Numerical Analysis for Solving Nonlinear Equation. *Journal of Al-Qadisiyah for computer science and mathematics, 13 (3)*, Page-70.

[16]   RASHEED, M., SHIHAB, S., Rashid, A., Rashid, T., Hamed, S. H. A., & Aldulaimi, M. A. H. (2021). An Iterative Method to Solve Nonlinear Equation. *Journal of Al-Qadisiyah for Computer Science and Mathematics, 13 (2)*, Page-87.

[17]   Sheth, T. (2018). *Scilab: A Practical Introduction to Programming and Problem Solving*. CRC Press.

[18]   Sizemore, J., & Mueller, J. P. (2019). *MATLAB For Dummies*. Wiley.

[19]   Srivastava, R. B., & Srivastava, S. (2011). Comparison of Numerical Rate of Convergence of Bisection, Newton-Raphson's and Secant Methods. *Journal of Chemical, Biological and Physical Sciences (JCBPS), 2 (1)*, 472.

[20]   Vishwanatha, J. S., Swamy, R. S., Mahesh, G., & Gouda, H. V. (2023). A toolkit for computational fluid dynamics using spectral element method in Scilab. *Materials Today: Proceedings*.