Journal of Mathematics and Computing Science, 10 (1). 14-25, 2024

eISSN: 0128-0767

# Comparison of Some Numerical Methods for Solving Real-Life Nonlinear Equations by Using Python Programming

**Annie Gorgey[1]*, Zul Hafiezy Zulkifly[2], Haslinah Hassim[3] and Farah Adilla Azim[4]**

[1,2,3,4]Department of Mathematics, Faculty of Science and Mathematics, Sultan Idris Education University, 35900 Tanjung Malim, Perak, Malaysia

Authors' email: annie_gorgey@fsmt.upsi.edu.my*, d095689@siswa.upsi.edu.my, d095668@siswa.upsi.edu.my and d095688@siswa.upsi.edu.my

*Corresponding author

Received 10 May 2024; Received in revised 5 June 2024; Accepted 20 June 2024
Available online 30 June 2024
DOI: https://doi.org/10.24191/jmcs.v10i1.1856

**Abstract**: This research evaluates the efficiency and accuracy of some iterative methods for solving scalar nonlinear equations. The study focuses on four types of iterative methods, such as Newton-Raphson, Bisection, Secant, and Algorithm 1, a novel fourth-order and derivative-free root-finding algorithm exhibiting different convergence orders. The accuracy of these methods is tested numerically on some real-life nonlinear equations such as the fluid permeability problem, the blood rheology model, Van Der Wall's and Planck's radiation laws, and lastly, the beam design problem. Numerical experiments are conducted using the Python programming language with a tolerance of 10-14 and 10-15. The results indicate that the Secant method requires fewer iterations and CPU time if compared with Newton-Raphson, Bisection and Algorithm 1 in solving fluid permeability problems, blood rheology models, and Van Der Wall's problems. The bisection method converges quickly compared with other methods for solving Planck's radiation law, while Algorithm 1 converges quicker than other methods for solving the beam design problem. In terms of accuracy, the Secant method gives greater accuracy in solving fluid permeability and Van Der Wall's problems. Meanwhile, for the blood rheology model, Newton Raphson's methods overcome other methods. On the other hand, the Bisection method gives greater accuracy for Planck's radiation law and beam design problems. Algorithm 1 performance showed effective convergence to the root, but, in many cases, it encounters a division by zero issue. The study suggests extending investigations to Algorithm 1 refinement, comparative studies on various equation types, exploration of hybrid methods, real-world application and validation, and user-friendly implementation.

**Keywords**: Algorithm 1, Bisection, Newton-Raphson, Numerical Methods, Python, Secant

## 1 Introduction

Scalar nonlinear equations are fundamental in various scientific and engineering applications due to their ability to model complex behaviour and systems. There are many applications involving scalar, nonlinear equations. In physics and engineering problems, the applications involved are heat transfer and fluid dynamics. Scalar nonlinear equations are used to model heat conduction and transfer in various materials and systems. The nonlinear heat conduction equation helps predict temperature distribution over time and space in engineering systems such as reactors, engines, and buildings [1]. Meanwhile, in the study of fluid dynamics, nonlinear equations describe the flow of incompressible fluids, including phenomena like turbulence and boundary layer behaviour. The Navier-Stokes equation, although typically a system of equations, has scalar nonlinear counterparts for specific simplified scenarios [7,3].

This research focuses on comparing different numerical methods for solving nonlinear equations, which are crucial in various scientific and engineering fields [1]. The study aims to identify the most efficient iterative method by numerically investigating the basic iterative methods, which are Newton-Raphson (NR), Secant, and Bisection, including Algorithm 1, which was newly proposed by the previous researchers [7]. The selected methods are implemented using Python, a versatile platform for scientific computing [2]. The performance of each method is evaluated based on metrics such as CPU time, number of iterations, and absolute error of approximations. The findings will contribute to the existing knowledge of numerical methods for solving nonlinear equations, providing valuable insights for practitioners and researchers [3].

## 2 Methodology

### *A Newton-Raphson Method*

The Newton-Raphson (NR) method is an iterative technique commonly used to approximate the roots of a nonlinear equation. It relies on the idea of linearising the equation by employing the tangent line at an initial guess and iteratively refining the estimate until a satisfactory approximation to the root is obtained. Graphically, the NR method involves finding the intersection point between the tangent line and the x-axis, which represents an improved approximation to the root. Starting with an initial guess, the tangent line is drawn, and its intersection with the x-axis is determined. This process is repeated iteratively until a desired level of accuracy is achieved. The iteration scheme for the NR method can be summarised as follows [4]:

1. Start with an initial guess, $x_0$, for the root of the equation.
2. Compute the function value, $f(x_0)$, and the derivative value, $f'(x_0)$, at the initial guess.
3. Calculate the next approximation, $x_1$, using the formula:

$$x_1 = x_0 - \frac{f(x_0)}{f\prime(x_0)}. \quad (1)$$

4. Repeat steps 2 and 3 iteratively until the desired level of accuracy is achieved or a predetermined number of iterations is reached.

The iteration scheme exploits the fact that the tangent line to the function at a given point provides a good approximation to the behaviour of the function near that point. By iteratively updating the approximation using the tangent line, the method converges towards the root. The NR method has order-2 accuracy.

### *B Secant Method*

The Secant method (SM) is an iterative numerical technique used to approximate the roots of a nonlinear equation. Unlike the NR method, it does not require the evaluation of derivatives. Instead, it estimates the slope of the function using a finite difference approximation. Graphically, the Secant method approximates the root of an equation by drawing a straight line through two initial guesses and finding the intersection point of the line with the x-axis. This intersection point serves as a new approximation to the root. The process is repeated iteratively, with each new line being drawn using the previous two approximations until the desired level of accuracy is achieved. The iteration scheme for the Secant method can be summarised as follows [5]:

1. Start with two initial guesses, $x_0$ and $x_1$, for the root of the equation.
2. Calculate the function values, $f(x_0)$ and $f(x_1)$.
3. Approximate the slope of the function using the finite difference approximation:

$$\text{Slope } (m) = \frac{(f(x_1) - f(x_0))}{(x_1 - x_0)} \quad (2)$$

4. Compute the next approximation, $x_2$, using the formula:

$$x_2 = x_1 - \frac{f(x_1)}{m} \tag{3}$$

5. Update the values of $x_0$ and $x_1$ as $x_1$ and $x_2$, respectively.
6. Repeat steps 2–5 iteratively until the desired level of accuracy is achieved.

The iteration scheme adapts the slope estimation based on the previous two approximations, gradually refining the approximation to the root.

### *C Bisection Method*

The Bisection method (BM) is a numerical technique used to approximate the roots of a nonlinear equation. It relies on the principle of repeatedly bisecting an interval that contains a root until a desired level of accuracy is achieved. Graphically, the Bisection method involves dividing an interval that contains a root into two equal subintervals and identifying which subinterval the root lies within. The process is iteratively applied to the subinterval that contains the root until a desired level of accuracy is reached. The method exploits the intermediate value theorem, which guarantees the existence of a root within an interval if the function values at the endpoints have opposite signs. The iteration scheme for the Bisection method can be summarised as follows [6]:

1. Start with an interval $[a, b]$ that contains a root of the equation, where $f(a)$ and $f(b)$ have opposite signs.
2. Calculate the midpoint c of the interval:

$$c = \frac{(a + b)}{2} \tag{4}$$

3. Evaluate the function value $f(c)$.
4. If $f(c)$ is sufficiently close to zero or the interval width is below a predetermined tolerance, stop and return c as the approximated root.
5. Otherwise, determine which subinterval $[a, c]$ or $[c, b]$ contains a root based on the sign of $f(c)$.
6. Update the interval by setting either $a$ or $b$ to $c$, depending on which subinterval contains the root.
7. Repeat steps 2–6 iteratively until the desired level of accuracy is achieved.

The iteration scheme bisects the interval at each step, narrowing down the search space for the root. By iteratively refining the interval, the method converges towards the root.

### *D Algorithm 1 by Naseem et al. [7]*

The Algorithm 1 method (A1M) is a newly designed algorithm developed by Naseem et al. [7] for solving nonlinear equations. They designed a novel fourth-order and derivative-free root-finding algorithm by applying the finite difference scheme to the well-known Ostrowski method. This method offers a unique approach to finding the roots of nonlinear equations and has shown promising results in various real-life applications. The graphical description of the Algorithm 1 method involves analysing the behaviour of the function and its dynamics via computer tools. Algorithm 1 offers a novel perspective on solving nonlinear equations. The iteration scheme for the Algorithm 1 method can be summarised as follows:

1. Start with an initial guess, $u_0$, for the root of the equation $\psi(u) = 0$.
2. Compute the predictor, $v_i$, using the formula:

$$v_i = u_i - \frac{\psi(u_i)}{\psi'(u_i)} \tag{5}$$

3. Compute the intermediate point, $w_i$, using the formula:
$$w_i = v_i - \frac{\psi(v_i)}{\psi'(v_i)}. \tag{6}$$

4. Compute the next approximation, $u_{i+1}$, using the formula:

$$u_{i+1} = w_i - \frac{[\psi(w_i)\psi(v_i)]}{[\psi'(v_i)\psi(v_i) - 2\psi(w_i)]}. \tag{7}$$

5. Repeat steps 2–4 until the desired convergence is achieved or the maximum number of iterations is reached.

### *E Real-Life Nonlinear Equations*

The nonlinear equations selected for experimentation represent diverse real-world problems, each posing unique challenges for numerical methods. The following examples elucidate the nature of these equations.

#### i. Equation 1: Fluid Permeability Problem

In the field of fluid dynamics, the concept of hydraulic permeability is important [8]. This parameter quantifies the ease with which a fluid can traverse through a porous medium. The mathematical representation of this concept is given by the equation:

$$\kappa = \frac{r_e u^3}{20(1-u)^2} \tag{8}$$

where κ denotes the specific hydraulic permeability, $r_e$ represents the radius of the pores, and $u$ is the porosity, defined as the fraction of the volume of the material that is occupied by pores. The porosity value lies within the range of 0 to 1 [7]. An additional equation:

$$r_e u^3 - 20k(1-u)^2 = 0, \tag{9}$$

is also presented, which may be a rearranged form, or a related equation used in the computation process.

A specific instance of this concept is explored where the radius $r_e$ is 100 and the specific hydraulic permeability $k$ is 0.4655. Substituting these values into the equation results in a new nonlinear function:

$$\psi_1(u) = 100u^3 - 9.31(1-u)^2 \tag{10}$$

The solution $\psi_1$ is obtained through an iterative process, which involves making an initial guess for u (in this case, $u_0 = 1.0$) with a tolerance $10^{-15}$ for the $[0.30\,, 0.35]$ interval and then repeatedly applying a method to refine this guess until the true solution is reached.

#### ii. Equation 2: Blood Rheology Model

Blood rheology, a specialised field within the scientific community, focuses on the study of the physical and flow properties of blood [9]. Blood is characterised as a non-Newtonian fluid and is often modelled as a Caisson fluid [10]. The Caisson fluid model elucidates those simple fluids in a tube flow in such a

manner that the central core of the fluid moves akin to a plug with minimal deformation, and the velocity gradient is predominantly observed near the wall [7]. To analyse the plug flow of Caisson fluids, we consider the following nonlinear equation:

$$H = 1 - \frac{16}{7}\sqrt{u} + \frac{4}{3}u - \frac{1}{21}u^4, \tag{11}$$

where $H$ computes the reduction in flow rate. By substituting $H$=0.40 into equation (11), the following nonlinear function is derived [11]:

$$\psi_2(u) = \frac{1}{441}u^8 - \frac{8}{63}u^5 - 0.05714285714u^4 + \frac{16}{9}u^2 - 3.624489796u + 0.3. \tag{12}$$

To solve for $\psi_2$, an initial guess of $u_0 = 2.0$ with tolerance of $10^{-15}$ for the $[0.00\ , 0.10]$ interval is used to commence the iterative process.

### iii. Equation 3: Van Der Wall's Equation

Van Der Wall's equation is a renowned mathematical model that provides a comprehensive understanding of the behaviour of both real and ideal gases [12]. The equation is expressed as follows:

$$(P + \frac{C_1 n^2}{V^2})(V - nC_2) = iRT \tag{13}$$

In this equation:

- $P$ denotes the pressure of the gas.
- $C_1$ and $C_2$ are constants.
- $n$ represents the number of moles of the gas.
- $V$ is the volume of the gas.
- $i$ signifies the number of degrees of freedom of the gas molecules.
- $R$ is the ideal gas constant.
- $T$ is the temperature of the gas.

By substituting specific values into this equation, it can be transformed into a nonlinear function [7]:

$$\psi_3(u) = 0.986u^3 - 5.181u^2 + 9.067u - 5.289 \tag{14}$$

In this equation, $u$ represents the volume of the gas. This equation is cubic, so it should have three roots. However, only one of these roots is physically meaningful because the volume of a gas cannot be negative. This root is approximately 1.9298462428.

To find this root, an iterative process is used. This involves starting with an initial guess $u$ (in this case, $u_0 = 2.0$) with tolerance of $10^{-14}$ for the $[1.50\ , 2.00]$ interval and then repeatedly applying a method to refine this guess until the true solution is reached.

### iv. Equation 4: Planck's Radiation Law

The Law of Planck's Radiation is a pivotal principle in quantum mechanics that provides a comprehensive understanding of the energy density within a black isothermal body [13]. The equation is expressed as follows:

$$\phi(\gamma) = \frac{8\pi Pc}{\gamma^5 e^{Pc/\gamma Tk} - 1} \tag{15}$$

In this equation:

- P denotes the pressure of the gas.
- c represents the speed of light.
- Tk is the temperature of the black body.
- γ is the wavelength

The objective is to compute the wavelength $\gamma_1$ at which the energy density $\phi(\gamma_1)$ reaches its peak. To achieve this, equation (15) is transformed into the following nonlinear function [7]:

$$\psi_4(u) = -1 + \frac{u}{5} + e^{-u} \tag{16}$$

In this equation, $u$ represents the wavelength. This equation is used to find the value of $u$ that maximizes the energy density. One of the estimated roots of $\psi_4$ is -0.00, which signifies the maximum wavelength of the radiation.

To find this root, an iterative process is used. This involves starting with an initial guess for u (in this case, $u_0 = 0.5$) with a tolerance $10^{-15}$ for the $[-1\,, 1]$ interval and then repeatedly applying a method to refine this guess until the true solution is reached.

### v. Equation 5: Beam Designing Problem

The problem of beam design is a significant challenge in the fields of Physics and Engineering [14]. This problem pertains to the determination of the embedment, denoted as $u$, a sheet pile wall [15]. A sheet pile wall is a structural element used in construction to retain soil or water. The embedment of the wall is mathematically represented as a scalar nonlinear function:

$$\psi_5(u) = \frac{u^3 + 2.87u^2 - 4.62u - 10.28}{4.62} \tag{17}$$

This equation $u$ represents the embedment of the sheet pile wall. This equation is used to find the value $u$ that satisfies the conditions of the problem.

To find this value, an iterative process is used. This involves starting with an initial guess for $u$ (in this case, $u_0 = 1.0$) with tolerance $10^{-14}$ for the $[2.00\,, 2.10]$ interval and then repeatedly applying a method to refine this guess until the true solution is reached. The maximum number of iterations used for all numerical experiments is 100.

## 3 Results and Discussion

The analysis of the numerical experiments reveals a nuanced perspective on the efficiency, accuracy and reliability of different iterative methods applied to a variety of nonlinear equations. The discussion is structured around the results obtained for each equation.

### *A The Most Efficient Method*

In this section, we analyse the efficiency of the numerical methods based on the CPU time and iteration numbers required to converge to a solution. The following results were obtained from the experiments.

Table 1: Numerical Results based on computational time (Efficiency Table)

| Equations | NR | BM | SM | A1M |
|---|---|---|---|---|
| 1 | 0.3426482058114499<br>Iteration: 7<br>(0.0000000000 sec) | 0.34264820581145017<br>Iteration: 45<br>(0.0000000000 sec) | 0.3426482058114499<br>**Iteration: 5**<br>**(0.0000000000 sec)** | None<br>(Encounter division by zero) |
| 2 | 0.08643355805246679<br>Iteration: 7<br>(0.0000000000 sec) | 0.08643355805246672<br>Iteration: 46<br>(0.0000000000 sec) | 0.08643355805246677<br>**Iteration: 5**<br>**(0.0000000000 sec)** | None<br>(Encounter division by zero) |
| 3 | None<br>Iteration: 100<br>(0.0000000000 sec) | None<br>Iteration: 100<br>(0.0000000000 sec) | 1.9298462428478675<br>**Iteration: 11**<br>**(0.0000000000 sec)** | None<br>(Encounter division by zero) |
| 4 | -1.16014740694848E-16<br>Iteration: 6<br>(0.0090053082 sec) | 0.0<br>**Iteration: 1**<br>**(0.0019958019 sec)** | 3.49819278439754E-17<br>Iteration: 10<br>(0.0080032349 sec) | -2.18542946146473E-16<br>Iteration: 2<br>(0.0039973259 sec) |
| 5 | 2.0021187789538275<br>Iteration: 7<br>(0.0000000000 sec) | 2.0021187789538306<br>Iteration: 42<br>(0.0000000000 sec) | 2.0021187789538266<br>Iteration: 4<br>(0.0000000000 sec) | 2.002118778953827<br>**Iteration: 3**<br>**(0.0000000000 sec)** |

The analysis reveals that, in terms of efficiency, the Secant method (SM) often exhibits superior performance, requiring fewer iterations and CPU time to converge compared to the Newton-Raphson (NR), Bisection (BM) and Algorithm 1(A1M) in solving Equations 1 to 3. Notably, the Bisection methods provided rapid convergence for well-behaved functions, such as Equation 4, and Algorithm 1 converges quicker than other methods for Equation 5. Efficiency in terms of iterations is a crucial metric for real-world applications, especially when computational resources are limited. However, it is essential to note that the choice of the most efficient method may vary depending on the nature of the function and the initial guess.

### *B The Most Accurate Method*

In this section, we evaluate the accuracy of the numerical methods concerning the absolute value of the roots obtained. The absolute values for each method in solving the equations are as follows:

Table 2: Numerical Results based on errors (Accuracy Table)

| Equations | NR | BM | SM | A1M |
|---|---|---|---|---|
| 1 | 1.1102230246251565e-16<br>Iteration: 7<br>(0.0000000000 sec) | 1.6653345369377348e-16<br>Iteration: 45<br>(0.0000000000 sec) | **1.1102230246251565e-16**<br>Iteration: 5<br>(0.0000000000 sec) | (Encounter division by zero) |
| 2 | **1.3877787807814457e-17**<br>Iteration: 7<br>(0.0000000000 sec) | 8.326672684688674e-17<br>Iteration: 46<br>(0.0000000000 sec) | 2.7755575615628914e-17<br>Iteration: 5<br>(0.0000000000 sec) | (Encounter division by zero) |
| 3 | 3.3084646133829665e-14<br>Iteration: 100<br>(0.0000000000 sec) | 6.972200594645983e-14<br>Iteration: 100<br>(0.0000000000 sec) | **7.549516567451064e-15**<br>Iteration: 11<br>(0.0000000000 sec) | (Encounter division by zero) |
| 4 | 1.16014740694848e-16<br>Iteration: 6<br>(0.0090053082 sec) | **0.0**<br>Iteration: 1<br>(0.0019958019 sec) | 3.49819278439754e-17<br>Iteration: 10<br>(0.0080032349 sec) | 2.18542946146473e-16<br>Iteration: 2<br>(0.0039973259 sec) |
| 5 | 2.6645352591003757e-15<br>Iteration: 7<br>(0.0000000000 sec) | **4.440892098500626e-16**<br>Iteration: 42<br>(0.0000000000 sec) | 3.552713678800501e-15<br>Iteration: 4<br>(0.0000000000 sec) | 3.1086244689504383e-15<br>Iteration: 3<br>(0.0000000000 sec) |

The absolute error analysis provides insight into the precision of each method in determining the roots. Smaller absolute errors indicate greater accuracy. The second method gives greater accuracy in solving Equations 1 and 3. Meanwhile, for Equation 2, the Newton-Raphson (NR) method overcomes other methods. On the other hand, the Bisection method (BM) gives greater accuracy for Equations 4 and 5. Algorithm 1 (A1M) performances showed effective convergence to the root, but, in many cases, it encountered a division by zero issue. However, it is crucial to acknowledge that the choice of the most accurate method depends on various factors, including the characteristics of the function and the specific requirements of the application.

## 4 Python Coding

Below is the code of Python programming for the numerical experiment.

### *A Importing Libraries*

```
import math
from prettytable import PrettyTable
import matplotlib.pyplot as plt
import numpy as np
from sympy import symbols, Eq, solve, exp, sqrt, sin, cos, tan
import time
```

### *B Defining the Function*

```
x = symbols('x')
def f(x):
    return (x**3 + 2.87*x**2 - 4.62*x - 10.28)/4.62
```

### *C Solving the Equation Symbolically*

```
equation = Eq(f(x), 0)
roots = solve(equation, x)
print("Roots found:", roots)
```

### *D Plotting the Function*

```
x_vals = np.linspace(-5, 5, 1000)
f_numeric = np.vectorize(lambda x: float(f(x)))
y_vals = f_numeric(x_vals)
plt.plot(x_vals, y_vals)
plt.axhline(0, color='black', linestyle='--', linewidth=0.8)
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Graph of the Function')
plt.grid(True)
plt.show()
```

### *E Calulating the Derivative of the Function*

```
def df(x, epsilon=1e-6):
    return (f(x + epsilon) - f(x - epsilon)) / (2 * epsilon)
```

### *F Intermediate Value Theorem Check*

```
def ivt_check(a, b):
    fa, fb = f(a), f(b)
    if fa * fb > 0:
        print("Bisection Root\t   : IVT does not guarantee a root in the
interval. Choose a different interval.")
        return False
    return True
```

### *G Bisection Method*

```
def bisection_auto(a, b, tolerance, max_iterations):
    B_iter = 0
    iterations = []
    if not ivt_check(a, b):
        return None, B_iter, iterations

    for i in range(max_iterations):
        c = (a + b) / 2
        B_iter += 1
        error = abs(c - desired_root)
        iterations.append((B_iter, c, f(c), error))
```

```
            if error < tolerance:
                return c, B_iter, iterations
            if f(c) * f(a) < 0:
                b = c
            else:
                a = c
        return None, B_iter, iterations
```

### *H Newton-Raphson Method*

```
    def newton_raphson_auto(initial_guess, tolerance, max_iterations):
        x = initial_guess
        NR_iter = 0
        iterations = []
        for i in range(max_iterations):
            x = x - f(x) / df(x)
            NR_iter += 1
            error = abs(x-desired_root)
            iterations.append((NR_iter, x, f(x), error))
            if error < tolerance:
                return x, NR_iter, iterations
        return None, NR_iter, iterations
```

### *I Secant Method*

```
    def secant_auto(x0, x1, tolerance, max_iterations):
        S_iter = 0
        iterations = []
        for i in range(max_iterations):
            f_x0, f_x1 = f(x0), f(x1)
            denominator = f_x1 - f_x0

            # Check for zero in the denominator
            if denominator == 0:
                print("\nSecant Method\t   : Division by zero encountered. Cannot
    continue.")
                return None, S_iter, iterations

            x2 = x1 - (f_x1 * (x1 - x0)) / denominator
            S_iter += 1
            error = abs(x2 - desired_root)
            iterations.append((S_iter, x2, f(x2), error))
            if error < tolerance:
                return x2, S_iter, iterations
            x0, x1 = x1, x2
        return None, S_iter, iterations
```

### *J Algorithm 1 Method*

```
    def algorithm1_auto(initial_guess, max_iterations, tolerance):
        al_iter = 0
        x = initial_guess
        iterations = []
        for i in range(max_iterations):
            vi = x - f(x) / df(x)
            wi = vi - f(vi) / df(vi)

            # Check if the denominator is zero
            denominator = df(vi) * (f(vi) - 2 * f(wi))
            if denominator == 0:
                print("\nAlgorithm1 Root\t   : Division by zero encountered.
    Cannot continue.")
                return None, al_iter, iterations

            x_plus_1 = wi - (f(wi) * f(vi)) / denominator

            x = x_plus_1
            al_iter += 1
```

```
            error = abs(x - desired_root)
            iterations.append((al_iter, x, f(x), error))
            if error < tolerance:
                return x, al_iter, iterations

        return None, al_iter, iterations
```

### *K Printing Iteration Results*

```
def print_iterations_table(iterations, method_name):
    table = PrettyTable()
    table.field_names = ["Iteration", "Root", f"{method_name} f(Root)",
"Absolute Error"]
    table.add_rows(iterations)
    print(f"\n{method_name} Iterations:")
    print(table)

if __name__ == "__main__":
    initial_guess = float(input("Enter the initial guess: "))
    tolerance = float(input("Enter the tolerance: "))
    max_iterations = int(input("Enter the maximum number of iterations: "))

    ivt_a = float(input("Enter the lower bound for IVT (ivt_a): "))
    ivt_b = float(input("Enter the upper bound for IVT (ivt_b): "))
    desired_root = float(input("Enter the desired exact root: "))

    start_newton_raphson = time.time()
    root_newton_auto, NR_iter, iterations_newton =
newton_raphson_auto(initial_guess, tolerance, max_iterations)
    end_newton_raphson = time.time()

    start_bisection = time.time()
    root_bisection_auto, B_iter, iterations_bisection = bisection_auto(ivt_a,
ivt_b, tolerance, max_iterations)
    end_bisection = time.time()

    start_secant = time.time()
    root_secant_auto, S_iter, iterations_secant = secant_auto(ivt_a, ivt_b,
tolerance, max_iterations)
    end_secant = time.time()

    start_algorithm1 = time.time()
    root_algorithm1_auto, al_iter, iterations_algorithm1 =
algorithm1_auto(initial_guess, max_iterations, tolerance)
    end_algorithm1 = time.time()

    print("\n--------------ITERATIONS-----------------\n")

    print_iterations_table(iterations_newton, "Newton-Raphson")

    if ivt_check(ivt_a, ivt_b):
        print_iterations_table(iterations_bisection, "Bisection")
    else:
        print("Bisection Root\t   : IVT does not guarantee a root in the
interval. Choose a different interval.")

    print_iterations_table(iterations_secant, "Secant")
    print_iterations_table(iterations_algorithm1, "Algorithm1")

    print("\n--------------RESULTS-----------------\n")

    print(f"Newton-Raphson Root: {root_newton_auto} \t(Iterations: {NR_iter},
Time: {end_newton_raphson - start_newton_raphson:.10f} seconds)")
    if ivt_check(ivt_a, ivt_b):
        print(f"Bisection Root\t   : {root_bisection_auto} \t(Iterations:
{B_iter}, Time: {end_bisection - start_bisection:.10f} seconds)")
    print(f"Secant Root\t   : {root_secant_auto} \t(Iterations: {S_iter},
Time: {end_secant - start_secant:.10f} seconds)")
```

```
    print(f"Algorithm1 Root\t   : {root_algorithm1_auto} \t\t(Iterations:
{al_iter}, Time: {end_algorithm1 - start_algorithm1:.10f} seconds)")

    print("\n----------------END-------------------\n")
```

## 5 Conclusion

This research embarked on a comprehensive exploration of numerical methods, including Newton-Raphson, Bisection, Secant, and Algorithm 1, to solve nonlinear equations representing diverse real-world problems. The study revealed the behaviour of each method across varied problem domains. The results indicate that the Secant method requires fewer iterations and CPU time if compared to Newton-Raphson, Bisection and Algorithm 1 in solving Equation 1. For Equation 4, the Bisection method converges quickly compared with other methods, and for Equation 5, Algorithm 1 converges quicker than other methods. In terms of accuracy, the Secant method gives greater accuracy in solving Equations 1 and 3. Meanwhile, for Equation 2, the Newton-Raphson method overcomes other methods. On the other hand, the Bisection method gives greater accuracy for Equations 4 and 5. Algorithm 1's performance showed effective convergence to the root, but, in many cases, it encounters a division by zero issue. The research contributes to the practical understanding of solving nonlinear equations and the academic dialogue surrounding numerical methods. It also opens avenues for further refinement and exploration of Algorithm 1. The study concludes with recommendations for future research, emphasising algorithmic refinement, exploration of hybrid methods, real-world applications, and the importance of user-friendly implementations. The research journey encapsulates the essence of scientific exploration and the symbiosis between theory and computation.

### Acknowledgements

The authors would like to express gratitude to Universiti Pendidikan Sultan Idris and the Research Management and Innovation Centre (RMIC) for providing a GPUF research grant (2021-0215-103-01) that enables the team to complete some parts of the research objectives through this article.

### Conflict of Interest Statement

The authors agree that this research was conducted in the absence of any self-benefits, commercial or financial conflicts and declare the absence of conflicting interests with the funders.

### References

[1] S. R. Vadyala, S. N. Betgeri, J. C. Matthews, and E. Matthews, “A review of physics-based machine learning in civil engineering,” *Results Eng.*, vol. 13, no. 100316, 2022.

[2] A. Lavanya *et al.*, “Assessing the performance of python data visualization libraries: a review,” *Int J Comput Eng Res Trends*, vol. 10, no. 1, pp. 29–39, 2023.

[3] F. Yu, L. Liu, L. Xiao, K. Li, and S. Cai, “A robust and fixed-time zeroing neural dynamics for computing time-variant nonlinear equation using a novel nonlinear activation function,” *Neurocomputing*, vol. 350, pp. 108–116, 2029.

[4] S. Akram and Q. U. Ann, “Newton raphson method,” *Int. J. Sci. Eng. Res.*, vol. 6, no. 7, pp. 1748–1752, 2015.

[5] I. K. Argyros and S. K. Khattri, “On the Secant method,” *J. Complex.*, vol. 29, no. 6, pp. 454–471, 2013.

[6] C. Solanki, P. Thapliyal, and K. Tomar, “Role of bisection method,” *Int. J. Comput. Appl. Technol. Res.*, vol. 3, no. 8, pp. 535–535, 2014.

[7] A. Naseem, M. A. Rehman, J. Younis, and V. T. Pham, “Some Real-Life Applications of a Newly Designed Algorithm for Nonlinear Equations and Its Dynamics via Computer Tools,” *Complexity*, vol. 2021, pp. 1–9, 2021.

[8] S. G. Solazzi, J. G. Rubino, D. Jougnot, and K. Holliger, “Dynamic permeability functions for

partially saturated porous media," *Geophys. J. Int.*, vol. 221, no. 2, pp. 1182–1189, 2020.
[9] J. S. Horner, *An experimental and theoretical investigation of blood rheology*. University of Delaware, 2020.
[10] J. S. Tulshiram, "A Study of Free Convection Flow of a Chemically Reacting Fluid," 2023.
[11] S. Qureshi, A. Soomro, A. A. Shaikh, E. Hincal, and N. Gokbulut, "A novel multistep iterative technique for models in medical sciences with complex dynamics," *Comput. Math. Methods Med.*, vol. 2022, no. 1, p. 7656451, 2022.
[12] C. Della Volpe and S. Siboni, "From van der Waals equation to acid-base theory of surfaces: a chemical-mathematical journey," *Rev. Adhes. Adhes.*, vol. 47–97, no. 1, pp. 47–97, 2022.
[13] H. J. Geesink, "Evidence for a quantum distribution of Planck's black-body radiation law," 2023.
[14] J. Chen, S. Hu, S. Zhu, and T. Li, "Metamaterials: from fundamental physics to intelligent design," *Interdiscip. Mater.*, vol. 2, no. 1, pp. 5–29, 2023.
[15] J. L. Wilt, *Design of Bending Moment and Load Capacity Test for FRP Sheet Piles*. West Virginia University, 2021.