American Journal of Applied Mathematics
2024, Vol. 12, No. 4, pp. 79-90
https://doi.org/10.11648/j.ajam.20241204.11

Research Article

Derivation and Performance Analysis of Composite Trapezoidal Iterative Methods

Isaac Azure*

Department of Mathematics, Kwame Nkrumah University of Science and Technology, Kumasi, Ghana

Abstract

The pursuit of more efficient and reliable numerical methods to solve nonlinear systems of equations has long intrigued many researchers. Among these, the Broyden method has stood out since its introduction, serving as a foundational technique from which various derivative methods have evolved. These derivative methods, commonly referred to as Broyden-like iterative methods, often surpass the traditional Broyden method in terms of both the number of iterations required and the computational time needed. This study aimed to develop new Broyden-like methods by incorporating weighted combinations of different quadrature rules. Specifically, the research focused on leveraging the Composite Trapezoidal rule with n=3n=3, and comparing it against the Midpoint, Trapezoidal, and Simpson quadrature rules. By integrating these approaches, three novel methods were formulated. The findings revealed that several of these new methods demonstrated enhanced efficiency and robustness compared to their established counterparts. In a detailed comparative analysis with the classical Broyden method and other improved versions, the Midpoint–Composite Trapezoidal (MT_3) method emerged as the top performer. This method consistently provided superior numerical outcomes across all benchmark problems examined in the study. The results highlight the potential of these new methods to significantly advance the field of numerical analysis, offering more powerful tools for researchers and practitioners dealing with complex nonlinear systems of equations. Through this innovative approach, the study not only broadens the understanding of Broyden-like methods but also sets the stage for further advancements in the development of efficient numerical solutions.

Keywords

Newton-Raphson Method, Broyden Method, Quadrature Rules, Composite Trapezoidal Rule

1. Introduction

In the realm of mathematical computations, finding solutions to equations is crucial for solving practical problems. These solutions, often represented as the roots of equations, are essential across various fields of study [1]. Consequently, there's a continuous quest to develop efficient numerical methods to obtain accurate results. This pursuit becomes even more challenging when dealing with systems of nonlinear equations, which are prevalent in engineering and scientific applications.

Despite considerable research efforts, solving nonlinear equations remains a complex task [2]. One significant challenge is the selection of suitable initial guesses for solutions, as it greatly influences the effectiveness of traditional numerical methods like Newton's method [3]. To address this,

*Corresponding author: azureike@yahoo.com (Isaac Azure)

Received: 2 June 2024; **Accepted:** 17 June 2024; **Published:** 6 August 2024

researchers have explored combining traditional methods with intelligent algorithms [4]. However, such approaches can be complex and computationally demanding, especially when dealing with multiple nonlinear equations simultaneously [5, 6].

Among the different numerical schemes available, the Newton–Raphson method is widely used but has drawbacks, such as the need to iteratively calculate the inverse Jacobian matrix, making it inefficient for large-scale problems. This limitation has led to the development of alternative methods like the Broyden method, which has undergone significant improvements over time, inspiring further innovations in the field [7].

Researchers have introduced various techniques to enhance the efficiency of solving nonlinear systems of equations. These include using central finite differences to approximate the inverse Jacobian matrix, resulting in improved schemes. Additionally, strategies like the Steepest Descent method have been employed to obtain adequate initial guesses for subsequent Broyden method solutions [8-11].

A recent trend involves formulating iterative schemes using quadrature rules, reflecting efforts to develop more efficient methods for computing solutions to systems of nonlinear equations. Numerous references are available on methods utilizing quadrature rules, highlighting ongoing efforts to improve computational approaches in this area [12, 13].

Newton-Cotes quadrature rules encompass techniques for numerical integration that involve evaluating the integrand at uniformly spaced points [14]. Named after Isaac Newton and Roger Cotes, these rules use polynomial approximations of local order k to fit the data. This method involves assessing the function at k evenly spaced nodes $(x_1, x_2, \dots, x_n)$ and assigning weights $w_1, w_2, \dots, w_n$ to these nodes, the Newton-Cotes quadrature formulas estimate the integral of a function $\int_a^b f(x)dx$ [15, 21]. Among the commonly known Newton-Cotes quadrature formulas are the Mid-point, Trapezoidal, and Simpson's rules. The general structure of the Newton-Cotes formula is expressed as:

$$\int_a^b f(x)dx = \sum_{k=1}^{n} w_k f(x_k) \quad (1)$$

Newton's method can be obtained by expanding a function (which is of a single variable), denoted as $f(x)$, using Taylor's series around a specific point x_1:

$$f(x) = f(x_1) + (x - x_1)f'(x_1) + \frac{1}{2!}(x - x_1)^2 f''(x_1) + \cdots \quad (2)$$

In this scenario, both the function f and its first and second derivatives, denoted as f', and f'', respectively, are assessed at the point x_1. If dealing with a function F that involves multiple variables and maps from $F: R^n \to R^n$, it can be demonstrated [20] that equation (2) is equivalent.

$$F(x) = F(x_k) + \int_{x_k}^{x} F'(t)dt \quad (3)$$

The array of partial derivatives denoted as $F'(t)$ in equation (3) represents the Jacobian matrix, which is denoted as J. Here, the integral from $\int_{x_k}^{x} F'(t)dt$, as described in equation (4), involves multiple integrals.

$$\int_{x_k}^{x} F'(t)dt = \int_{x_k,1}^{1} \int_{x_k,2}^{2} \cdots \int_{x_k,n}^{n} F'(x_1, x_2, \dots, x_n)dx_n dx_{n-1} \dots dx_1 \quad (4)$$

An alternative method treats the multiple integral as a sequence of nested one-dimensional integrals, applying a one-dimensional quadrature rule to each variable in turn [16]. This allows us to estimate $\int_{x_k}^{x} F'(t)dt$ by combining weighted quadrature formulas. Various techniques have been proposed by the authors [16, 17] and others cited therein for approximating the integral in equation (4) using Newton-Cotes formulas ranging from zero to first order. In a related study, a modification of the Broyden-like method was suggested, employing a weighted combination of the Trapezoidal, Simpson, and Midpoint quadrature rules, resulting in the TSMM Broyden-like method [18], as expressed below:

$$m_k = x_k - B_K^{-1}F(x_k)$$

$$x_{k+1} = x_k - 24[5B(x_k) + 14B(z_k) + 5B(m_k)]^{-1}F(x_k) \quad (5)$$

where: $z_k = \frac{x_k + m_k}{2}, k = 0, 1, \dots$

The TSMM method's performance was assessed in comparison to various existing methods, including Classical Broyden (CB), Trapezoidal-Broyden (TB), and Midpoint-Simpson-Broyden (MSB). The TSMM method showed superior performance over all the others [19]. In a follow-up study conducted the next year by the same author, a robust variant of the Broyden-like method was introduced, called the Midpoint-Trapezoidal (MT) method. The iterative process of this method is described as follows:

$$m_k = x_k - B_K^{-1}F(x_k)$$

$$x_{k+1} = x_k - 4[B(x_k) + 2B(z_k) + B(m_k)]^{-1}F(x_k) \quad (6)$$

for $z_k = \frac{x_k + m_k}{2}, k = 0, 1, \dots$

The MT method was evaluated against other existing methods, including Classical Broyden (CB), Trapezoidal-Broyden, and Midpoint-Simpson-Broyden (MSB). The results showed that the MT method significantly outperformed all the others [20]. A significant observation among these Broyden-like methods is their consistent use of three common quadrature rules: Trapezoidal, Midpoint, and Simpson rules. It was suggested that further improvements could be made by refining these quadrature rules.

In another study, the integral in Equation (4) was approximated using a weighted combination of quadrature rules including Trapezoidal, Midpoint, Simpson, Simpson's 1/3,

and Simpson's 3/8 quadrature rules, resulting in the development of MS-1/3, MS-3/8, TS-1/3, TS-3/8, SS-1/3, and SS-3/8 methods. Among these, the MS-3/8 method was found to be superior to all existing methods [21].

This study aims to assess the effectiveness of methods formed through a weighted combination of the composite trapezoidal rule when compared to the midpoint, trapezoidal, and Simpson rules. The table below illustrates how the quadrature rules were combined:

Table 1. *Combination of Quadrature Rules to Yield New Broyden-like Methods.*

Quadrature rules	**Midpoint (M)**	**Trapezoidal (T)**	**Simpson (S)**
Composite Trapezoidal (T_3)	MT_3	TT_3	ST_3

In a study by the authors [21], a Broyden-like method named the Trapezoidal-Simpson's 3/8 method was devised, utilizing a weighted combination of the Trapezoidal and Simpson's 3/8 quadrature rules. While this method showed superiority in certain benchmark problems compared to other Broyden-like methods, there were instances where alternative methods yielded better results.

This study aims to achieve the following objectives: (i) Develop three new Broyden-like methods by utilizing weighted combinations of quadrature rules; (ii) Evaluate the new methods by comparing their number of iterations, CPU time, and robustness index using selected systems of nonlinear equations as test problems. The paper is structured as follows: Section 2.0 outlines the general formula of the composite trapezoidal quadrature rule, Section 3.0 details the derivation of the newly developed methods, Section 4.0 presents numerical tests and results, and Section 5.0 provides a summary conclusion of the research findings.

2. The General Formula of Composite Trapezoidal Rule

The Composite Trapezoidal Rule is a numerical integration method used to approximate the definite integral of a function over a specified interval. The Composite Trapezoidal Rule is based on approximating the area under the curve of a function by dividing the interval into smaller subintervals and approximating each subinterval's area using trapezoids.

For a function $F(x)$ over the interval $[a, b]$, the Composite Trapezoidal Rule formula is:

$$\int_a^b F(x)dx \approx \frac{h}{2}[F(a) + 2\sum_{i=1}^{n-1} F(x_i) + F(b)] \quad (7)$$

Where h is the width of each n subinterval and x_i are the partition points.

For the purpose of this research, the composite trapezoidal rule with $n = 3$ is what is considered for the study. This special composite trapezoidal rule is therefore denoted as T_3.

This means we are dividing the interval [a, b] into three equal subintervals. Denoting h as the width of each subinterval, means $h = \frac{b-a}{3}$. The points where the function is evaluated within each subinterval as:

$$x_0 = a$$

$$x_1 = a + h$$

$$x_2 = a + 2h$$

$$x_3 = b$$

Using these points, we can apply the composite Trapezoidal rule formula:

$$\int_a^b F(x)dx \approx \frac{h}{2}[F(a) + 2F(x_1) + 2F(x_2) + F(b)] \quad (8)$$

Substituting the expressions x_1 *and* x_2 in terms of a *and* h.

$$\int_a^b F(x)dx \approx \frac{h}{2}[F(a) + 2F(a+h) + 2F(a+2h) + F(b)] \quad (9)$$

Given that $h = \frac{b-a}{3}$

$$\int_a^b F(x)dx \approx \frac{\left(\frac{b-a}{3}\right)}{2}\left[F(a) + 2F\left(a + \frac{b-a}{3}\right) + 2F\left(a + 2\frac{b-a}{3}\right) + F(b)\right] \quad (10)$$

$$\int_a^b F(x)dx \approx \frac{(b-a)}{6}\left[F(a) + 2F\left(a + \frac{b-a}{3}\right) + 2F\left(a + 2\frac{b-a}{3}\right) + F(b)\right] \quad (11)$$

$$\int_a^b F(x)dx \approx \left(\frac{b-a}{6}\right)\left[F(a) + 2F\left(\frac{2a+b}{3}\right) + 2F\left(\frac{a+2b}{3}\right) + F(b)\right] \quad (12)$$

3. Derivation of New Methods

This section aims to develop a novel Broyden-like method formed through the weighted combination of the composite trapezoidal quadrature rule, compared against the midpoint, trapezoidal, and Simpson quadrature rules. The weighted combinations yield the MT_3, TT_3 and ST_3 methods as outlined below.

3.1. Derivation of MT_3 Method

Given the Composite Trapezoidal (T_3) quadrature rule

$$\int_a^b F(x)dx \approx \left(\frac{b-a}{6}\right)\left[F(a) + 2F\left(\frac{2a+b}{3}\right) + 2F\left(\frac{a+2b}{3}\right) + F(b)\right] \quad (13)$$

And Midpoint quadrature rule

$$\int_a^b F(x) \approx (b-a)F\left(\frac{a+b}{2}\right) \quad (14)$$

Approximating the integral in equation (3) by the average of Midpoint and Composite Trapezoidal (MT_3) quadrature rules yields:

$$\int_{x_k}^{x} F'(t)dt = \frac{x-x_k}{2}\left(F'\left(\frac{x_k+x}{2}\right)\right) + \left(\frac{x-x_k}{12}\right)\left[F'(x_k) + 2F'\left(\frac{2x_k+x}{3}\right) + 2F'\left(\frac{x_k+2x}{3}\right) + F'(x)\right] \quad (15)$$

Substituting equation (15) into (3), we have

$$F(x) = F(x_k) + \frac{x-x_k}{2}\left[F'\left(\frac{x_k+x}{2}\right)\right] + \left(\frac{x-x_k}{12}\right)\left[F'(x_k) + 2F'\left(\frac{2x_k+x}{3}\right) + 2F'\left(\frac{x_k+2x}{3}\right) + F'(x)\right] \quad (16)$$

$F(x) = 0$, hence

$$0 = F(x_k) + \frac{x-x_k}{2}\left[F'\left(\frac{x_k+x}{2}\right)\right] + \left(\frac{x-x_k}{12}\right)\left[F'(x_k) + 2F'\left(\frac{2x_k+x}{3}\right) + 2F'\left(\frac{x_k+2x}{3}\right) + F'(x)\right] \quad (17)$$

Multiplying through (17) by $\frac{12}{x-x_k}$

$$0 = \frac{12}{x-x_k}F(x_k) + 6\left[F'\left(\frac{x_k+x}{2}\right)\right] + \left[F'(x_k) + 2F'\left(\frac{2x_k+x}{3}\right) + 2F'\left(\frac{x_k+2x}{3}\right) + F'(x)\right] \quad (18)$$

$$0 = \frac{12}{x-x_k}F(x_k) + 6F'\left(\frac{x_k+x}{2}\right) + F'(x_k) + 2F'\left(\frac{2x_k+x}{3}\right) + 2F'\left(\frac{x_k+2x}{3}\right) + F'(x) \quad (19)$$

$$\frac{-12}{x-x_k}F(x_k) = 6F'\left(\frac{x_k+x}{2}\right) + F'(x_k) + 2F'\left(\frac{2x_k+x}{3}\right) + 2F'\left(\frac{x_k+2x}{3}\right) + F'(x) \quad (20)$$

$$x - x_k = \frac{-12F(x_k)}{6F'\left(\frac{x_k+x}{2}\right)+F'(x_k)+2F'\left(\frac{2x_k+x}{3}\right)+2F'\left(\frac{x_k+2x}{3}\right)+F'(x)} \quad (21)$$

$$x = x_k - 12\left[6F'\left(\frac{x_k+x}{2}\right) + F'(x_k) + 2F'\left(\frac{2x_k+x}{3}\right) + 2F'\left(\frac{x_k+2x}{3}\right) + F'(x)\right]^{-1}F(x_k) \quad (22)$$

Setting $x = x_{k+1}$ and $x_k = x_k$ in equation (22), we have

$$x_{k+1} = x_k - 12\left[6F'\left(\frac{x_k+x_{k+1}}{2}\right) + F'(x_k) + 2F'\left(\frac{2x_k+x_{k+1}}{3}\right) + 2F'\left(\frac{x_k+2x_{k+1}}{3}\right) + F'(x_{k+1})\right]^{-1}F(x_k) \quad (23)$$

Setting $F'\left(\frac{2x_k+x_{k+1}}{3}\right) \approx F'\left(\frac{x_k+2x_{k+1}}{3}\right) \approx F'\left(\frac{x_k+x_{k+1}}{2}\right)$

$$x_{k+1} = x_k - 12\left[6F'\left(\frac{x_k+x_{k+1}}{2}\right) + F'(x_k) + 2F'\left(\frac{x_k+x_{k+1}}{2}\right) + 2F'\left(\frac{x_k+x_{k+1}}{2}\right) + F'(x_{k+1})\right]^{-1}F(x_k) \quad (24)$$

$$x_{k+1} = x_k - 12\left[F'(x_k) + 10F'\left(\frac{x_k+x_{k+1}}{2}\right) + F'(x_{k+1})\right]^{-1}F(x_k) \quad (25)$$

Which is an implicit equation because the presence of x_{k+1} at both sides of the equation, hence to avoid its implicit nature we use the $(k+1)^{th}$ iteration of the Broyden's method in the right hand side. Thus we have;

$$x_{k+1} = x_k - 12[F'(x_k) + 10F'(z_k) + F'(m_k)]^{-1}F(x_k) \quad (26)$$

Let $B_k = B(x_k) + 10B(z_k) + B(m_k)$

$$\Longrightarrow x_{k+1} = x_k - 12B_K^{-1}F(x_k) \quad (27)$$

3.2. Derivation of TT_3 Method

Given the Composite Trapezoidal (T_3) quadrature rule as in equation (13).

And Trapezoidal quadrature rule:

$$\int_a^b F(x) \approx \left(\frac{b-a}{2}\right)(F(a) + F(b)) \quad (28)$$

Approximating the integral in equation (3) using the average of the Midpoint and Composite Trapezoidal (MT_3) quadrature rules results in:

$$\int_{x_k}^{x} F'(t)dt = \frac{x-x_k}{4}\left(F'(x) + F'(x_k)\right) + \left(\frac{x-x_k}{12}\right)\left[F'(x_k) + 2F'\left(\frac{2x_k+x}{3}\right) + 2F'\left(\frac{x_k+2x}{3}\right) + F'(x)\right] \quad (29)$$

Substituting equation (29) into (3), we have

$$F(x) = F(x_k) + \frac{x-x_k}{4}[F'(x) + F'(x_k)] + \left(\frac{x-x_k}{12}\right)\left[F'(x_k) +$$

$$2F'\left(\frac{2x_k+x}{3}\right)+2F'\left(\frac{x_k+2x}{3}\right)+F'(x)\Big] \quad (30)$$

$F(x)=0$, hence

$$0=F(x_k)+\frac{x-x_k}{4}[F'(x)+F'(x_k)]+\left(\frac{x-x_k}{12}\right)\Big[F'(x_k)+2F'\left(\frac{2x_k+x}{3}\right)+2F'\left(\frac{x_k+2x}{3}\right)+F'(x)\Big] \quad (31)$$

Multiplying through (31) by $\frac{12}{x-x_k}$

$$0=\frac{12}{x-x_k}F(x_k)+3[F'(x)+F'(x_k)]+\Big[F'(x_k)+2F'\left(\frac{2x_k+x}{3}\right)+2F'\left(\frac{x_k+2x}{3}\right)+F'(x)\Big] \quad (32)$$

$$0=\frac{12}{x-x_k}F(x_k)+3F'(x)+3F'(x_k)+F'(x_k)+2F'\left(\frac{2x_k+x}{3}\right)+2F'\left(\frac{x_k+2x}{3}\right)+F'(x) \quad (33)$$

$$0=\frac{12}{x-x_k}F(x_k)+4F'(x)+4F'(x_k)+2F'\left(\frac{2x_k+x}{3}\right)+2F'\left(\frac{x_k+2x}{3}\right) \quad (34)$$

$$x-x_k=\frac{-12F(x_k)}{4F'(x)+4F'(x_k)+2F'\left(\frac{2x_k+x}{3}\right)+2F'\left(\frac{x_k+2x}{3}\right)} \quad (35)$$

$$x=x_k-12\Big[4F'(x)+4F'(x_k)+2F'\left(\frac{2x_k+x}{3}\right)+2F'\left(\frac{x_k+2x}{3}\right)\Big]^{-1}F(x_k) \quad (36)$$

Setting $x=x_{k+1}$ and $x_k=x_k$ in equation (36), we have

$$x_{k+1}=x_k-12\Big[4F'(x_{k+1})+4F'(x_k)+2F'\left(\frac{2x_k+x_{k+1}}{3}\right)+2F'\left(\frac{x_k+2x_{k+1}}{3}\right)\Big]^{-1}F(x_k) \quad (37)$$

Setting $F'\left(\frac{2x_k+x_{k+1}}{3}\right)\approx F'\left(\frac{x_k+2x_{k+1}}{3}\right)\approx F'\left(\frac{x_k+x_{k+1}}{2}\right)$,

$$x_{k+1}=x_k-12\Big[4F'(x_k)+2F'\left(\frac{x_k+x_{k+1}}{2}\right)+2F'\left(\frac{x_k+x_{k+1}}{2}\right)+4F'(x_{k+1})\Big]^{-1}F(x_k) \quad (38)$$

$$x_{k+1}=x_k-12\Big[4F'(x_k)+4F'\left(\frac{x_k+x_{k+1}}{2}\right)+4F'(x_{k+1})\Big]^{-1}F(x_k) \quad (39)$$

Which is an implicit equation because the presence of x_{k+1} at both sides of the equation, hence to avoid its implicit nature we use the $(k+1)^{th}$ iteration of the Broyden's method in the right hand side. Thus we have;

$$x_{k+1}=x_k-12[4F'(x_k)+4F'(z_k)+4F'(m_k)]^{-1}F(x_k) \quad (40)$$

Let $B_k=4B(x_k)+4B(z_k)+4B(m_k)$

$$\Longrightarrow x_{k+1}=x_k-12B_K^{-1}F(x_k) \quad (41)$$

3.3. Derivation of ST_3 Method

Given the Simpson quadrature rule:

$$\int_a^b F(x)\approx\left(\frac{b-a}{2}\right)\left(f(a)+3f\left(\frac{a+b}{2}\right)+f(b)\right) \quad (42)$$

And the Composite Trapezoidal (T_3) quadrature rule as in equation (13).

Approximating the integral in equation (3) by the average of Simpson and Composite Trapezoidal quadrature rules yields:

$$\int_{x_k}^{x}F'(t)dt=\frac{x-x_k}{4}\left(F'(x_k)+3F'(\tfrac{x+x_k}{2})+F'(x)\right)+\left(\frac{x-x_k}{12}\right)\Big[F'(x_k)+2F'\left(\frac{2x_k+x_k}{3}\right)+2F'\left(\frac{x_k+2x}{3}\right)+F'(x)\Big] \quad (43)$$

Substituting equation (43) into (3), we have

$$F(x)=F(x_k)+\frac{x-x_k}{4}\left(F'(x_k)+3F'(\tfrac{x+x_k}{2})+F'(x)\right)+\left(\frac{x-x_k}{12}\right)\Big[F'(x_k)+2F'\left(\frac{2x_k+x}{3}\right)+2F'\left(\frac{x_k+2x}{3}\right)+F'(x)\Big] \quad (44)$$

$F(x)=0$, hence

$$0=F(x_k)+\frac{x-x_k}{4}\left(F'(x_k)+3F'(\tfrac{x+x_k}{2})+F'(x)\right)+\left(\frac{x-x_k}{12}\right)\Big[F'(x_k)+2F'\left(\frac{2x_k+x}{3}\right)+2F'\left(\frac{x_k+2x}{3}\right)+F'(x)\Big] \quad (45)$$

Multiplying through (45) by $\frac{12}{x-x_k}$

$$0=\frac{12}{x-x_k}F(x_k)+4\left(F'(x_k)+3F'(\tfrac{x+x_k}{2})+F'(x)\right)+\Big[F'(x_k)+2F'\left(\frac{2x_k+x}{3}\right)+2F'\left(\frac{x_k+2x}{3}\right)+F'(x)\Big]+F(x) \quad (46)$$

$$0=\frac{12}{x-x_k}F(x_k)+4F'(x_k)+12F'(\tfrac{x+x_k}{2})+4F'(x)+F'(x_k)+2F'\left(\frac{2x_k+x}{3}\right)+2F'\left(\frac{x_k+2x}{3}\right)+F'(x) \quad (47)$$

$$\frac{-12}{x-x_k}=12F'\left(\frac{x+x_k}{2}\right)+5F'(x)+5F'(x_k)+3F'\left(\frac{2x_k+x}{3}\right)+3F'\left(\frac{x_k+2x}{3}\right) \quad (48)$$

$$\frac{-12}{x-x_k}=5F'(x_k)+12F'\left(\frac{x+x_k}{2}\right)+3F'\left(\frac{2x_k+x}{3}\right)+3F'\left(\frac{x_k+2x}{3}\right)+5F'(x) \quad (49)$$

$$x-x_k=\frac{-12F(x_k)}{5F'(x_k)+12F'\left(\frac{x+x_k}{2}\right)+3F'\left(\frac{2x_k+x}{3}\right)+3F'\left(\frac{x_k+2x}{3}\right)+5F'(x)} \quad 50)$$

$$x = x_k - 12\left[5F'(x_k) + 12F'\left(\frac{x+x_k}{2}\right) + 3F'\left(\frac{2x_k+x}{3}\right) + 3F'\left(\frac{x_k+2x}{3}\right) + 5F'(x)\right]^{-1} F(x_k) \quad (51)$$

Setting $x = x_{k+1}$ and $x_k = x_k$ in equation (51), we have

$$x_{k+1} = x_k - 12\left[5F'(x_k) + 12F'\left(\frac{x_{k+1}+x_k}{2}\right) + 3F'\left(\frac{2x_k+x_{k+1}}{3}\right) + 3F'\left(\frac{x_k+2x_{k+1}}{3}\right) + 5F'(x_{k+1})\right]^{-1} F(x_k) \quad (52)$$

Setting $F'\left(\frac{2x_k+x_{k+1}}{3}\right) \approx F'\left(\frac{x_k+2x_{k+1}}{3}\right) \approx F'\left(\frac{x_k+x_{k+1}}{2}\right)$, equation (52) becomes

$$x_{k+1} = x_k - 12\left[5F'(x_k) + 18F'\left(\frac{x_k+x_{k+1}}{2}\right) + 5F'(x_{k+1})\right]^{-1} F(x_k) \quad (53)$$

Which is an implicit equation because the presence of x_{k+1} at both sides of the equation, hence to avoid its implicit nature we use the $(k+1)^{th}$ iteration of the Broyden's method in the right hand side. Thus we have;

$$x_{k+1} = x_k - 12[5(x_k) + 18F'(z_k) + 5F'(m_k)]^{-1}F(x_k) \quad (54)$$

with $m_k = x_k - B_K^{-1}F(x_k)$ and $z_k = \frac{x_k+m_k}{2}$

Now replacing $F'(x_k)$, $F'(m_k)$ and $F'(z_k)$ by $B(x_k)$, $B(m_k)$ and $B(z_k)$ respectively and use the same procedure as prescribed in [4-9], we get

$$x_{k+1} = x_k - 12[5(x_k) + 18F'(z_k) + 5F'(m_k)]^{-1}F(x_k) \quad (55)$$

$$\text{Let } B_k = 5B(x_k) + 18B(z_k) + 5B(m_k)$$

$$\Longrightarrow x_{k+1} = x_k - 12\,B_K^{-1}F(x_k) \quad (56)$$

4. Numerical Test

To evaluate the effectiveness of the new methods, they were tested on three benchmark problems described by Osinuga et al. in 2018. These tests involved varying dimensions, ranging from 5 to 1065 variables. The results were compared using three primary metrics: the number of iterations (NI), CPU time in seconds, and the Robustness Index (RI). The computations were conducted using Python (Anaconda Navigator 2.4.0) on a computer with the following specifications: Intel® Core™ i5-3427U CPU @ 1.80GHz 2.30GHz processor, 8.00GB RAM (7.87GB usable), and a 64-bit Operating System, ×64-based processor.

The program was set to stop execution once the number of iterations reached 500. Any method that failed to meet these convergence criteria is denoted by a dash ('-'), as shown in the results Table 1.

The problems that were used for the test are:

Problem One

$$F_i(x) = x_i x_{i+1} - 1,\ F_n(x) = x_n x_1 - 1,\ i = 1,2,\dots,n-1 \text{ and } x_0 = (0.8, 0.8, \dots, 0.8)^T$$

Problem Two

$$F_i(x) = x_i x_{i+1} - 1,\ F_n(x) = x_n x_1 - 1, i = 1,2,\dots,n-1 \text{ and } x_0 = (0.5, 0.5, \dots, 0.5)^T$$

Problem Three

$$F_i(x) = x_i^2 - \cos(x_1 - 1), i = 1,2,\dots,n \text{ and } x_0 = (0.5, 0.5, \dots, 0.5)^T$$

5. Results and Discussion

The performance of three newly developed methods was evaluated by applying them to solve three benchmark problems. Python code was crafted for each method to compute three key parameters: CPU time, Number of Iterations (NI), and the Robustness Index (RI). CPU time indicates the duration, in seconds, required by the method to solve the problem, while the number of iterations represents how many iterations were necessary to approximate a solution. The Robustness Index indicates the method's consistency in solving the problem.

For the MT_3 method, testing it on the first benchmark problem revealed a consistent four (4) iterations required to reach an approximation, irrespective of the value of 'n.' The CPU time ranged from 0.0000 to 0.7856 seconds, with the lowest recorded at 'n=15' and 'n=35' and the highest at 'n=1065.' A Robustness Index of 0.5000 was consistent across all 'n' values, indicating its stability. For the second and third problems, the MT_3 method consistently required five (5) iterations for all 'n' values. CPU times ranged from 0.0000 to 1.1484 for problem two and were 0.9542 for problem three. Robustness indices ranged between 0.9357 to 0.9999 for problem two and 0.5015 to 0.8242 for problem three.

The TT_3 method, when applied to the first benchmark problem, consistently required five (5) iterations for all 'n' values. CPU times ranged from 0.0000 to 0.9572 seconds, with a Robustness Index ranging from 0.5000 to 0.7417. For the second problem, it required five (5) iterations, and for the third, six (6). CPU times ranged from 0.0156 to 1.0197 for problem two and from 0.0221 to 1.1879 for problem three. The method showed a consistent Robustness Index of 0.5000 for problem two, and for problem three, it recorded 0.9637 for 'n=665' and 0.8852 for 'n=1065' only.

Regarding the ST_3 method, it exhibited a significantly higher number of iterations for all benchmark problems (48, 50, and 49 for problems one, two, and three, respectively). CPU time ranges were 0.0156 to 10.3777, 0.0313 to 9.6954, and 0.0312 to 10.6786 for problems one, two, and three, re-

spectively. The method consistently recorded a Robustness Index of 0.5000 for all 'n' values across all problems. Table 2 summarizes all results in the study.

Table 2. *Comparison of Newly Developed Methods.*

Problem	n	1 MT_3			2 TT_3			3 ST_3		
		NI	CPU	RI	NI	CPU	RI	NI	CPU	RI
1	5	4	0.0156	0.5000	5	0.0000	0.5000	48	0.0156	0.5000
	15	4	0.0000	0.5000	5	0.0000	0.5000	48	0.0377	0.5000
	35	4	0.0000	0.5000	5	0.0156	0.5000	48	0.0156	0.5000
	65	4	0.0156	0.5000	5	0.0000	0.5000	48	0.0468	0.5000
	165	4	0.0555	0.5000	5	0.0312	0.5000	48	0.1536	0.5000
	365	4	0.0685	0.5000	5	0.0846	0.6024	48	0.8701	0.5000
	665	4	0.2944	0.5000	5	0.3542	0.7417	48	3.4445	0.5000
	1065	4	0.7856	0.5000	5	0.9572	0.8403	48	10.3777	0.5000
2	5	5	0.0156	-	5	0.0156	0.5000	50	0.0313	0.5000
	15	5	0.0000	0.9357	5	0.0000	0.5000	50	0.0221	0.5000
	35	5	0.0000	0.9945	5	0.0000	0.5000	50	0.0469	0.5000
	65	5	0.0156	0.9957	5	0.0156	0.5000	50	0.0534	0.5000
	165	5	0.0156	0.9999	5	0.0156	0.5000	50	0.1784	0.5000
	365	5	0.1159	0.9999	5	0.0864	0.5000	50	1.1726	0.5000
	665	5	0.4351	0.9999	5	0.3320	0.5000	50	4.0700	0.4999
	1065	5	1.1484	0.9999	5	1.0197	0.5000	50	9.6954	0.4999
3	5	5	0.0000	0.5015		0.0221	-	49	0.0312	0.5000
	15	5	0.0156	0.5023		0.0000	-	49	0.0377	0.5000
	35	5	0.0000	0.5023	6	0.0156	-	49	0.0469	0.5000
	65	5	0.0156	0.5077	6	0.0156	-	49	0.0846	0.5000
	165	5	0.0377	0.5335	6	0.0312	-	49	0.3363	0.5000
	365	5	0.1158	0.6010	6	0.1381	-	49	1.2304	0.5000
	665	5	0.3320	0.7218	6	0.4825	0.9637	49	3.5947	0.5000
	1065	5	0.9542	0.8242	6	1.1879	0.8852	49	10.6786	0.5000

In Figure 1 below, it is evident that regardless of the values of n, the MT_3 method consistently exhibited the shortest computational time (CPU time), closely trailed by TT_3. Consequently, it can be inferred that for problem one, MT_3 achieved the lowest CPU time, as depicted in Figure 1 ST_3 on the other hand, recorded the highest CPT time.

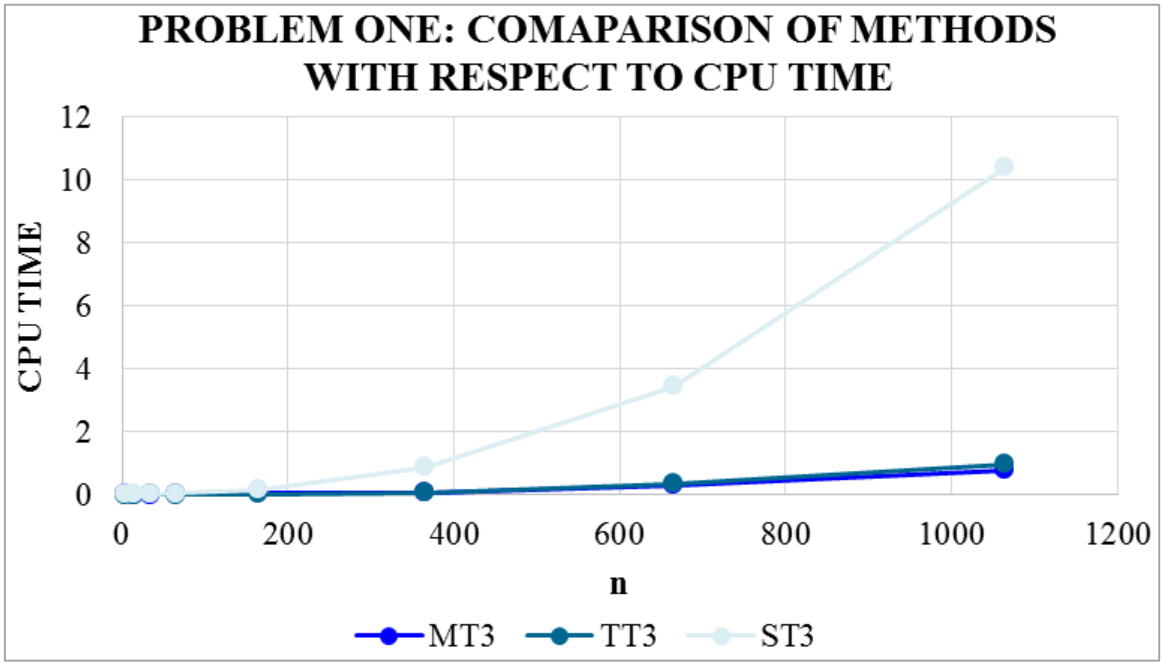

Figure 1. *Problem One: Comparison of Methods with Respect to CPU Time.*

Regarding problem one, Figure 2 distinctly illustrates that all methods exhibited robustness, as they demonstrated minimal variation in response to changes in parameters.

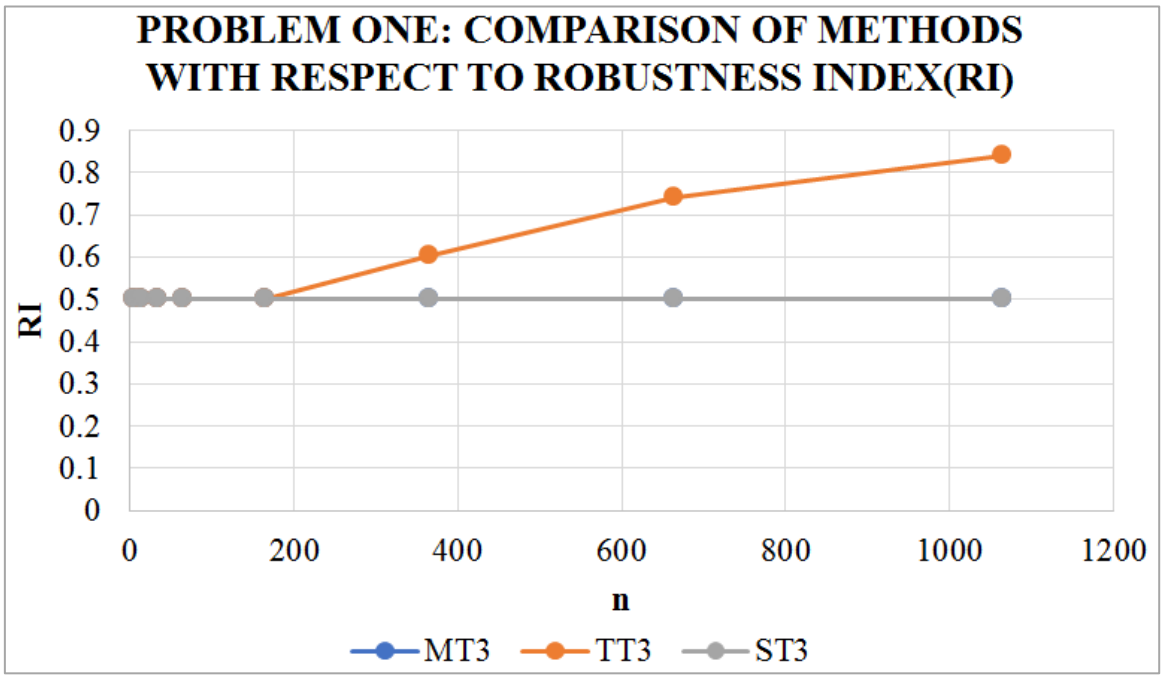

Figure 2. *Problem One: Comparison of Methods with Respect to Robustness Index (RI).*

Similar to Figure 1 above, both MT_3 and TT_3 methods displayed the lowest recorded CPU time values for problem two, highlighting their efficiency. Conversely, ST_3 once again exhibited significantly higher CPU time, as depicted in Figure 3.

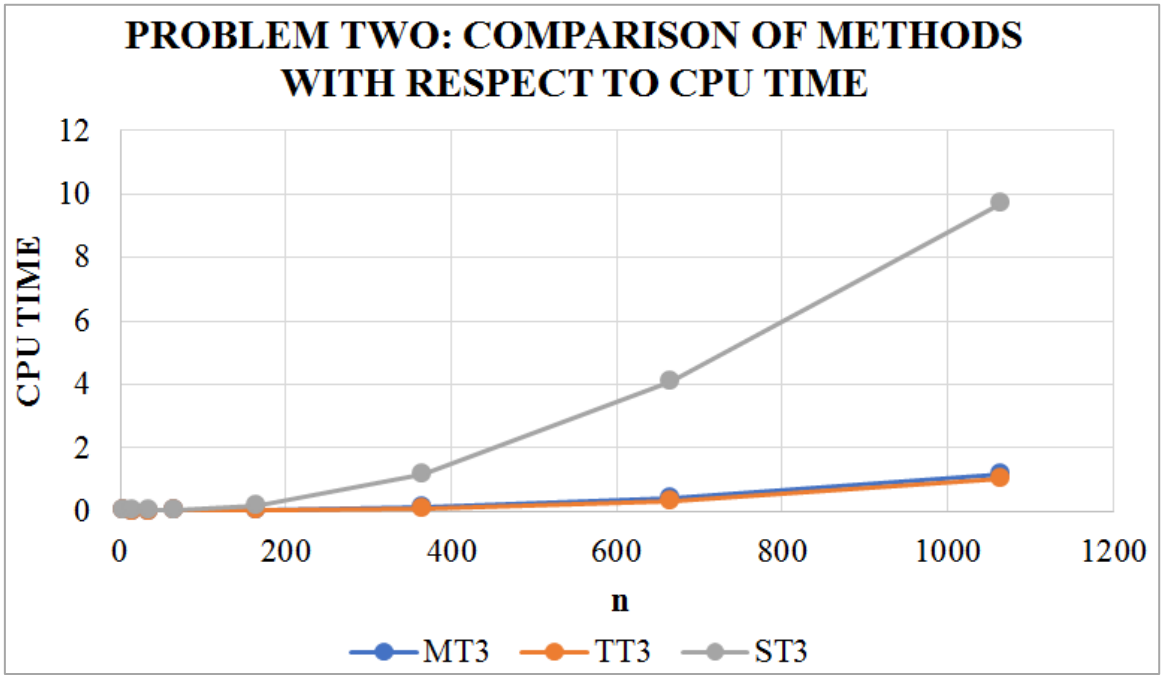

Figure 3. *Problem Two: Comparison of Methods with Respect to CPU Time.*

In Figure 4, it is evident that there was a fluctuation in the robustness index for the initial two values of 'n' when employing the MT_3 method. Nonetheless, the robustness index remained consistent for the subsequent 'n' values. Conversely, the TT_3 and ST_3 methods exhibited a comparable, unchanging robustness index across all 'n' values.

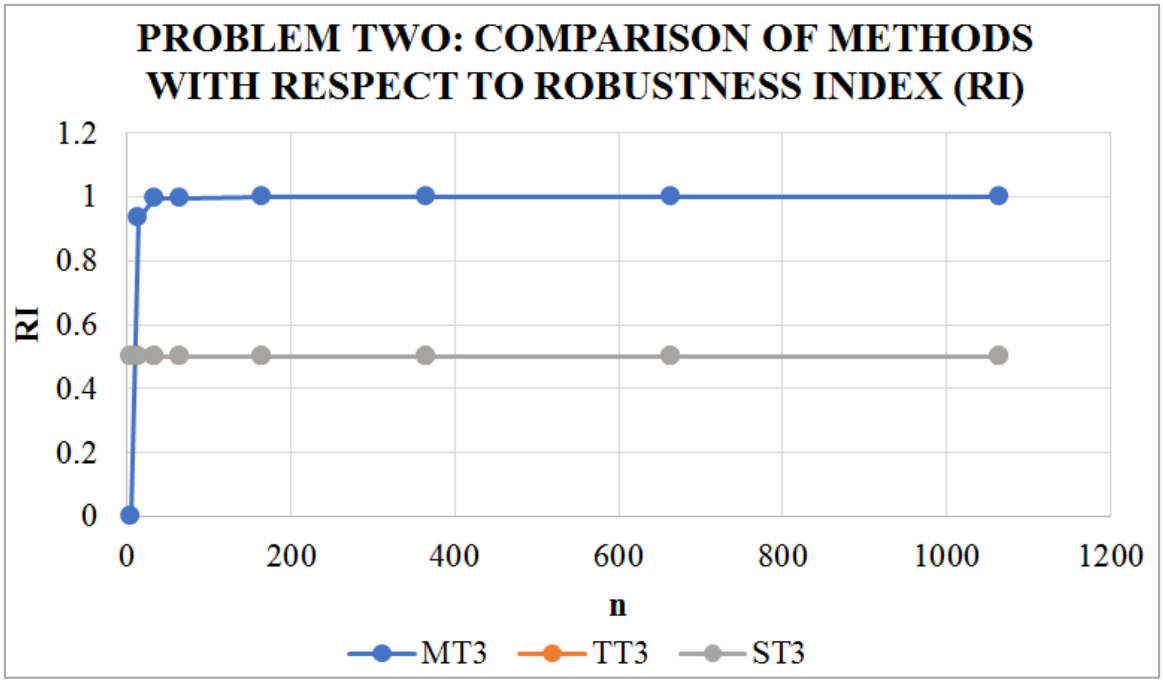

Figure 4. *Problem Two: Comparison of Methods with Respect to Robustness Index (RI).*

Once more, problem three exhibited notably low and comparable CPU times for MT_3 and TT_3, whereas ST_3 continued to display elevated CPU time, as depicted in Figure 5.

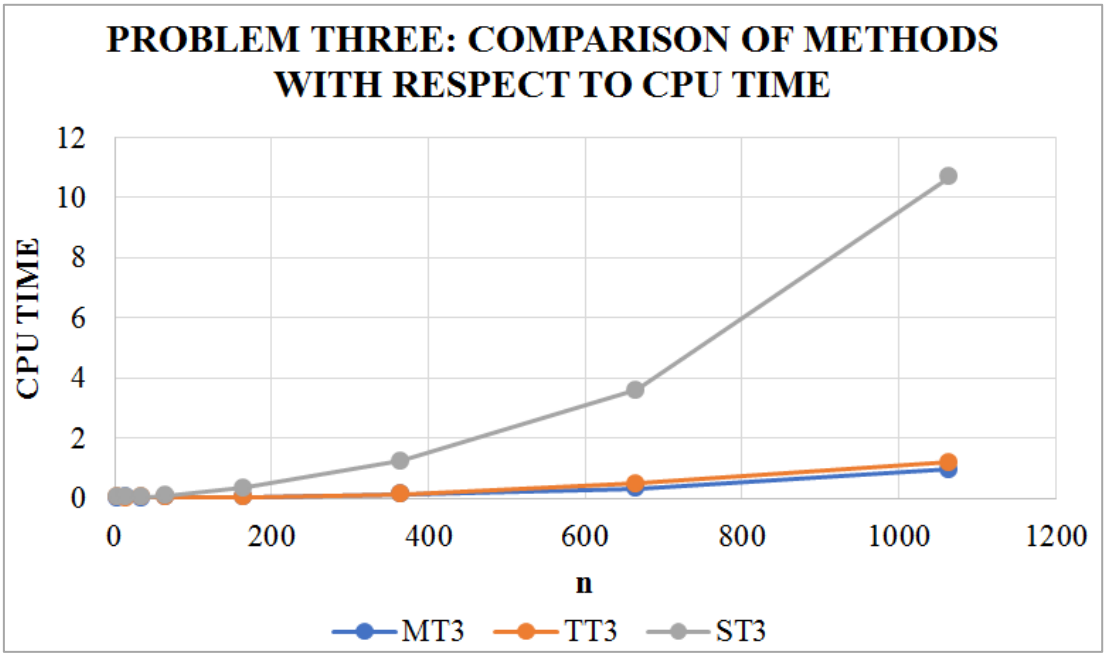

Figure 5. *Problem Three: Comparison of Methods with Respect to CPU Time.*

In Figure 6, it's noticeable that MT_3 and ST_3 displayed consistent robustness indices, whereas that of TT_3 varied significantly with an increase in the n values for problem three.

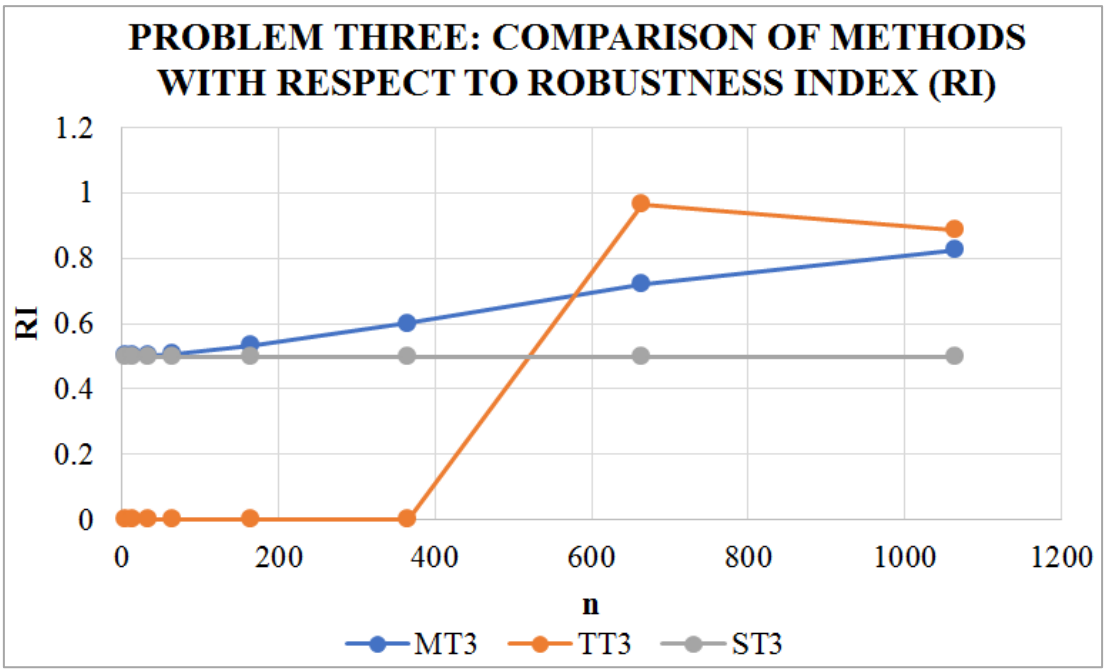

Figure 6. *Problem Three: Comparison of Methods with Respect to Robustness Index (RI).*

6. Conclusion

In this manuscript, we embarked on a journey to enhance the efficacy of numerical methods for solving systems of nonlinear equations, focusing on the development and analysis of Broyden-like iterative schemes. Inspired by the success of methods integrating quadrature rules, we introduced three novel Broyden-like methods MT_3, TT_3, and ST_3 formed through weighted combinations of the Composite Trapezoidal rule with the Midpoint, Trapezoidal, and Simpson quadrature

rules, respectively.

Through meticulous derivation and performance analysis, we unveiled the potential of these methods in achieving superior convergence rates and computational efficiency compared to established counterparts. Particularly, the MT_3 method emerged as the standout performer, consistently outperforming other methods across various benchmark problems. Its robustness and numerical efficacy make it a promising candidate for practical applications in engineering and scientific domains.

The derivation process elucidated the theoretical foundation of these methods, emphasizing the integration of quadrature rules to approximate the Jacobian matrix. By leveraging the Composite Trapezoidal rule alongside other quadrature rules, we effectively balanced accuracy and computational overhead, leading to enhanced convergence behavior.

7. Recommendation

Based on the findings and insights gleaned from this study, several recommendations emerge to guide future research and practical applications in computational mathematics and engineering:

1) Further Theoretical Analysis: Conduct rigorous theoretical investigations into the convergence properties and stability of the proposed methods under varying conditions. Analyze the impact of different weighting schemes and quadrature rule combinations on method performance to deepen our understanding of their behavior.
2) Experimental Validation: Validate the proposed methods through extensive numerical experiments across a diverse range of nonlinear systems. Assess their performance against real-world problems to ascertain their practical utility and robustness in different application scenarios.
3) Algorithm Optimization: Explore optimization strategies to further enhance the computational efficiency of the proposed methods, particularly for large-scale systems. Investigate parallelization techniques and algorithmic refinements to expedite convergence and reduce computational overhead.
4) Extension to Multivariate Systems: Extend the proposed methods to handle multivariate systems of nonlinear equations, catering to a broader range of applications in engineering and scientific computing. Investigate the scalability and efficacy of these methods in high-dimensional problem domains.
5) Integration with Machine Learning Techniques: Explore synergies between iterative numerical methods and machine learning techniques to develop hybrid approaches capable of adaptively adjusting algorithm parameters and improving convergence behavior. Investigate the potential of reinforcement learning and neural network-based approaches in optimizing method performance.

By pursuing these recommendations, researchers can advance the state-of-the-art in numerical methods for solving nonlinear equations, paving the way for more efficient and reliable computational tools with diverse applications across various disciplines.

Abbreviations

MT_3	Midpoint-Composite Trapezoidal Method
TT_3	Trapezoidal-Composite Trapezoidal Method
ST_3	Simpson-Composite Trapezoidal Method
RI	Robustness Index
CPU Time	Computational Time

Author Contributions

Isaac Azure is the sole author. The author read and approved the final manuscript.

Conflicts of Interest

The author declares no conflict of interest.

References

[1] Azure, I., Aloliga, G., & Doabil, L. (2020). Comparative Study of Numerical Methods for Solving Non-linear Equations Using Manual Computation. Mathematics Letters, 5(4), 41. https://doi.org/10.11648/j.ml.20190504.11

[2] Li, Y., Wei, Y., & Chu, Y. (2015). Research on solving systems of nonlinear equations based on improved PSO. Mathematical Problems in Engineering, 2015(1). http://dx.doi.org/10.1155/2015/727218

[3] Al-Towaiq, M. H., & Abu Hour, Y. S. (2017). Two improved classes of Broyden's methods for solving nonlinear systems of equations. JOURNAL OF MATHEMATICS AND COMPUTER SCIENCE-JMCS, 17(1), 22-31.

[4] Kou, J., Li, Y., & Wang, X. (2007). A composite fourth-order iterative method for solving non-linear equations. Applied Mathematics and Computation, 184(2), 471-475.

[5] Luo, Y. Z., Tang, G. J., & Zhou, L. N. (2008). Hybrid approach for solving systems of nonlinear equations using chaos optimization and quasi-Newton method. Applied Soft Computing, 8(2), 1068-1073.

[6] Mo, Y., Liu, H., & Wang, Q. (2009). Conjugate direction particle swarm optimization solving systems of nonlinear equations. Computers & Mathematics with Applications, 57(11-12), 1877-1882.

[7] Osinuga, I. A., & Yusuff, S. O. (2017). Construction of a Broyden-like method for Nonlinear systems of equations. Annals. Computer Science Series, 15(2), 128-135.

[8] Weerakoon, S., & Fernando, T. G. I. (2000). A variant of Newton's method with accelerated third-order convergence. Applied Mathematics Letters, 13(8), 87-93.

[9] Frontini, M. A. R. C. O., & Sormani, E. (2003). Some variant of Newton's method with third-order convergence. Applied Mathematics and Computation, 140(2-3), 419-426.

[10] Dhamacharoen, A. (2014). An efficient hybrid method for solving systems of nonlinear equations. Journal of Computational and Applied Mathematics, 263, 59-68.

[11] Mahwash, K. N., & Gyang, G. D. (2018). Numerical Solution of Nonlinear Systems of Algebriac Equations. https://doi.org/10.11648/j.ijdsa.20180401.14

[12] Mohammad, H., & Waziri, M. Y. (2015). On Broyden-like update via some quadratures for solving nonlinear systems of equations. Turkish Journal of Mathematics, 39(3), 335-345. https://doi.org/10.3906/mat-1404-41

[13] Isaac, A., Stephen, T. B., Seidu, B., Maza, D. D. A., Olukotun, S. F., & Akinlade, G. O. (2021). A New Trapezoidal-Simpson 3/8 Method for Solving Systems of Nonlinear Equations. American Journal of Mathematical and Computer Modelling, 6(1), 1-8. https://doi.org/10.11648/J.AJMCM.20210601.11

[14] Osinuga, I. A., & Yusuff, S. O. (2018). Quadrature based Broyden-like method for systems of nonlinear equations. Statistics, Optimization & Information Computing, 6(1), 130-138. https://doi.org/10.19139/soic.v6i1.471

[15] Darvishi, M. T., & Shin, B. C. (2011). High-order Newton-Krylov methods to solve systems of nonlinear equations. Journal of the Korean Society for Industrial and Applied Mathematics, 15(1), 19-30.

[16] Hafiz, M. A., & Bahgat, M. S. (2012). An efficient two-step iterative method for solving system of nonlinear equations. Journal of Mathematics Research, 4(4), 28. https://doi.org/10.5539/jmr.v4n4p28

[17] Jain, M. K. (2003). Numerical methods for scientific and engineering computation. New Age International.

[18] Kelley, C. T. (1995). Iterative methods for linear and nonlinear equations. Society for Industrial and Applied Mathematics.

[19] Van de Rotten, B., & Lunel, S. V. (2005). A limited memory Broyden method to solve high-dimensional systems of nonlinear equations. In EQUADIFF 2003 (pp. 196-201).

[20] Cordero, A., Torregrosa, J. R., & Vassileva, M. P. (2012). Pseudocomposition: a technique to design predictor–corrector methods for systems of nonlinear equations. Applied Mathematics and Computation, 218(23), 11496-11504.

[21] Isaac, A., Stephen, T. B., & Seidu, B. (2021). A Comparison of Newly Developed Broyden–Like Methods for Solving System of Nonlinear Equations. International Journal of Systems Science and Applied Mathematics, 6(3), 77-94. https://doi.org/10.11648/j.ijssam.20210603.11