ANCHOR UNIVERSITY LAGOS

Unified Code for Collocation Multistep Methods in Solving Systems of Ordinary Differential Equations

OKWHAROBO, Solomon Monday

Department of Mathematics, Faculty of Science, Anchor University Lagos
In partial fulfillment of the requirement for the award of Bachelor of Science(B.Sc.)
Degree in Mathematics.

July, 2024

DECLARATION

I hereby declare that this Project was written by me and is a correct record of my research work. It has not been presented in any previous application for any degree of this or any other University. All citations and sources of information are acknowledged using references.

_____

OKWHAROBO, SOLOMON M

Date: _____

CERTIFICATION

I hereby certify that this project work entitled **"Unified Code for Collocation Multistep Methods in Solving Systems of Ordinary Differential Equations"** was carried out by **OKWHAROBO Solomon Monday**, with a matric number **AUL/SCI/20/00605**, under the supervision of Supervisor and has not been submitted in part or full to this university or other institutions for the award of a degree.

Prof. J.O Fatokun   **Date**
(Supervisor)

Dr. P.O Evans   **Date**
Head of Department

Prof. S.A Okunuga   **Date**
External Supervisor

Acknowledgments

I would like to express my deepest gratitude to God, the source of all wisdom and strength, for guiding me throughout this academic journey.

My heartfelt thanks go to my mother, whose unwavering support, encouragement, and sacrifices have been the driving force behind my success.

I am immensely grateful to my supervisor, Prof J.O Fatokun, for his invaluable guidance, mentorship, and expertise. His insights and encouragement significantly contributed to the completion of this project.

Special thanks to Dr. H. Akewe, Dr. S.I Okoro, Dr. Evans, and Dr. B Akinnukawe for their support, constructive feedback, and contributions to my academic and personal development.

Your support has been instrumental, and I am deeply appreciative.

Abstract

In this research, we present a Solver developed using Flutter aimed at solving stiff systems of ordinary differential equations (ODEs) through a unified implementation of collocation multistep methods. The application not only provides solutions to differential equations but also offers comprehensive analysis of Linear Multistep Methods (LMM). It calculates error constants, checks for zero stability, and assesses convergence. Numerical examples illustrate the accuracy and functionality of the solver, showcasing its performance across various scenarios. Detailed analyses of methods such as Quade's method, Adams-Bashforth, Adams-Moulton, and Backward Differentiation Formula confirm the theoretical results with computational outputs. The results demonstrate the solver's effectiveness and robustness, establishing it as a valuable tool for numerical analysis in practical applications. solving both non-stiff and stiff differential equations.

Table of Contents

List of Figures

List of Tables

LIST OF ABBREVIATIONS

ODE :   Ordinary Differential Equation

BVP :   Boundary Value Problem

IVP :   Initial Value Problem

BDF: Backward Differentiation Formula

LMM: Linear Multistep Method

UI: User Interface

ILMM: Implicit Linear Multistep Method

ELMM: Explicit Linear Multistep Method

CHAPTER ONE

INTRODUCTION

1.1 Background of Study

Mathematical models in a vast range of disciplines, from science and technology to sociology and business, describe how quantities *change*. This leads naturally to a language of ordinary differential equations (ODEs). Ordinary Differential Equations (ODEs) are a type of differential equation that involves an unknown function and its derivatives.Quantities that change continuously in time or space are often modeled by differential equations. When everything depends on just one independent variable, we call the model an ordinary differential equation (ODE)(*Multistep methods — Fundamentals of Numerical Computation*, n.d.).

ODEs are of paramount significance in mathematical modeling because they provide a concise and powerful way to describe how a quantity changes concerning time or another independent variable. The ability to capture the rate of change of a variable makes ODEs essential in understanding dynamic processes, predicting future states, and optimizing system behavior. In many important cases of differential equations, analytic solutions are difficult or impossible to obtain and time consuming. The mathematical modelling of many problems in physics, engineering, chemistry, biology, and many more give rise to systems of ordinary differential equation. Yet, the number of instances where an exact solution can be found by analytical means is very limited(Lambert, 1977).In many important cases of differential equations, analytic solutions are difficult or impossible to obtain and time consuming, Hence the need for an approximate, or a numerical method.

In contemporary scientific and engineering research, the formulation of complex mathematical models often leads to the generation of differential equations that defy closed-form solutions. This persistent challenge has underscored the growing significance of approximate, or numerical, methods in tackling intricate mathematical problems. Among these methods, numerical techniques for ordinary differential equations (ODEs) stand out as indispensable tools, providing a robust means to compute numerical approximations to the solutions of these challenging equations.This necessity becomes even more pronounced when dealing with stiff systems of differential equations, where rapid variations in solution components pose additional complexities.Classical analytical methods, while powerful and elegant, encounter limitations when confronted with intricate mathematical formulations. Numerical methods step in precisely where analytical methods fall short, offering a practical avenue to obtain solutions when exact expressions are elusive.

Various advanced numerical techniques, such as implicit methods, exponential integrators, and collocation multistep methods, have proven effective in addressing the challenges posed by stiff systems. These methods excel in capturing the dynamics of stiff ODEs by incorporating strategies that adapt to the varying time scales inherent in the system. Implicit methods, for instance, allow for larger time steps, enhancing stability in the presence of stiffness.

With the advent of powerful computing technologies, numerical methods for ODEs have witnessed significant advancements. High-performance computing allows researchers and engineers to tackle more complex problems, simulate intricate physical phenomena, and explore the behavior of systems over extended time-frames. These simulations not only aid in understanding complex systems but also contribute to the optimization and design of real-world applications.

1.2   Problem Statement

Many studies on solving the equations of stiff ordinary differential equations (ODEs) have been done by researchers or mathematicians specifically. With the number of numerical methods that currently exist,extensive research has been done to unveil the comparison between their rate of convergence, number of computations, accuracy, and capability to solve certain type of test problems (Enright, Hull, & Lindberg, 1975) . The well-known numerical methods that are used widely are from

the class of BDFs or commonly understood as Gear's Method (Byrne, Hindmarsh, Jackson, & Brown, 1977). However, many other methods that have evolved to this date are for solving stiff ODEs which arise in many fields of the applied sciences (Yatim, Ibrahim, Othman, & Suleiman, 2013). The class of methods to consider in this project are Linear Multistep methods for the solutions of Initial and Boundary value problems of Ordinary Differential Equations.

In the field of computational mathematics and scientific computing, the effective analysis and numerical solution of ordinary differential equations (ODEs) play a pivotal role in modeling and understanding various real-world phenomena.ODEs arise in diverse contexts, ranging from modeling physical phenomena to simulating engineering systems, often exhibiting a spectrum of behaviors from stiff to non-stiff dynamics. Furthermore, both boundary value problems (BVPs) and initial value problems (IVPs) are prevalent scenarios requiring accurate and efficient numerical solutions.Linear multistep methods (LMMs) stand as prominent numerical techniques extensively utilized for solving ODEs, offering a balance between accuracy and computational efficiency. However, the implementation, analysis, and utilization of LMMs for tackling diverse ODE scenarios, including stiff and non-stiff problems, BVPs, and IVPs, pose significant challenges to researchers and practitioners in computational mathematics (J. Butcher, 2009).

Existing software solutions tailored for LMM analysis and ODE solving often lack robust capabilities to address the complexity and diversity of real-world ODE problems. For instance, proprietary software solutions like MATLAB and Wolfram Mathematica offer built-in functions for ODE solving, but they may not provide specific support for LMMs, potentially limiting the accuracy or efficiency of LMM-based solutions. Open-source libraries like SciPy and GNU Octave offer broader accessibility but may not offer as extensive support for LMM-specific analysis and customization compared to specialized software.

Furthermore, these software solutions may suffer from limitations such as:

- **Proprietary nature:** restricting access for users who cannot afford licenses or prefer open-source solutions.

- **Limited customization or extension capabilities:** particularly in commercial software, which may restrict users from implementing specialized algorithms or analyses.

- **User interface and documentation:** may not be as intuitive or user-friendly compared to specialized software designed specifically for LMM analysis and ODE solving.

The development of such an application requires a deep understanding of the mathematical principles underlying LMMs, including the identification and management of stiffness in ODEs, the selection of appropriate numerical methods, and the implementation of advanced analysis tools for error control and stability analysis. Additionally, the application must be designed to support both boundary value problems (BVPs) and initial value problems (IVPs), necessitating the development of algorithms that can adapt to the specific characteristics of these problem types.

By addressing these challenges, the proposed unified code aims to fill a critical gap in the field of computational mathematics, providing researchers and practitioners with a powerful tool for the analysis of LMM and solution of ODEs. This initiative is expected to contribute significantly to the advancement of computational mathematics education and research, enabling more effective problem-solving and decision-making in various disciplines.

The proposed solution aims to significantly enhance the ease and efficiency with which users can explore, analyze, and apply Linear Multistep Methods (LMMs) across a wide array of Ordinary Differential Equation (ODE) scenarios. This initiative is driven by the recognition that existing software solutions often lack comprehensive support for the complexity and diversity of real-world ODE problems, particularly when addressing stiff and non-stiff problems, as well as boundary value problems (BVPs) and initial value problems (IVPs).

1.3   Aim and Objectives

1.3.1   Aim:

The aim of the application is to develop a robust software tool for analyzing linear multistep methods used in solving stiff ordinary differential equations (ODEs) and also use the method to solve problems of ordinary differential equation (ODEs). This tool will encompass functionalities for assessing numerical properties such as zero-stability, consistency, convergence, and error constants, providing valuable insights

into the behavior and performance of these methods.

1.3.2 Objectives:

1. **Algorithm Development and Software Implementation**: Develop algorithms to analyze numerical properties of linear multistep methods,use the method to also solve stiff ODEs, implement them into a user-friendly software application with intuitive interfaces.

2. **Validation and Error Analysis**: Validate algorithms by comparing results with analytical solutions, incorporate error analysis functionalities to compute error constants, ensuring accuracy and reliability.

3. **Optimization and Documentation**: Optimize software performance for efficient analysis of large-scale ODE problems, provide comprehensive documentation and user support channels for enhanced usability and understanding.

1.4 Scope of Study

The scope of this study encompasses a comprehensive exploration of linear multistep methods as applied to the solution of stiff ordinary differential equations (ODEs). It involves an in-depth analysis and implementation of various linear multistep techniques, including but not limited to Adams-Bashforth and Adams-Moulton methods. The focus is on investigating the numerical properties of these methods, such as stability, accuracy, and convergence, particularly in the context of stiff ODEs.

Additionally, the study involves the development of a software tool tailored for the analysis of linear multistep methods. This entails designing intuitive user interfaces and incorporating features for parameter tuning, result visualization, and interpretation.The validation of the developed software will be conducted through rigorous testing against established benchmarks and analytical solutions to guarantee the accuracy and reliability of results.

Finally, the study will identify any limitations encountered and suggest future research directions and enhancements to the software tool to address these limitations and improve its applicability and functionality.

1.5   Significance of Study

The significance of this study lies in its contributions to both theoretical under-
standing and practical applications in the field of numerical analysis and compu-
tational mathematics, specifically focusing on linear multistep methods for solving
stiff ordinary differential equations (ODEs). It enhances the theoretical knowledge
by providing insights into the numerical properties of these methods, including sta-
bility, accuracy, and convergence behavior. Additionally, it offers valuable tools
and techniques for practitioners in scientific and engineering domains to accurately
and efficiently solve stiff differential equations encountered in their respective fields,
thereby advancing computational techniques used in various research, design, and
decision-making processes.

1.6   Definition of Terms

1. **Ordinary differential equation:** Let y be a function of a single variable x,.
   An ordinary differential equation is an equation of the form:

$$F(x, y, y', y'', ..., y^{(n)}) = 0 \tag{1.1}$$

   Where:

   - F is a given function of the independent variable x, the dependent variable
     y, and its n derivatives with respect to x.
   - y' represents the first derivative of y with respect to x, y'' represents the
     second derivative, and so on until $y^{(n)}$, which represents the nth derivative
     of y with respect to x.
   - The equation is typically defined over some interval in the domain of x
     where the function y is being considered.

   Solving an ODE means finding a function $y(x)$ that satisfies the given differ-
   ential equation over the specified domain. The solution to an ODE may be
   explicit or implicit, and in many cases, there may be multiple solutions or a
   family of solutions.

2. **Numerical method:** A numerical method is a difference equation involving
   a number of consecutive approximations $y_{n+j}, j = 0, 1, 2 \ldots k$ from which it be

possible to compute sequentially the sequence $y_n|n = 0, 1, 2, \ldots N$. The integer k is called a step-number; if $k = 1$, the method is called a one-step method, while if $k > 1$, the method is called a *one-step method*

3. **Step Length (Mesh-Size)**: A point within the solution domain where the solution is approximated or calculated. The **step length** (h) is the size of the interval between consecutive points in the independent variable (e.g., time or space) at which the solution of a differential equation is calculated. It plays a crucial role in determining the granularity of the numerical approximation and impacts the accuracy and efficiency of the solution. A smaller step length typically leads to a more accurate but computationally expensive solution, while a larger step length may sacrifice accuracy for computational efficiency. The choice of an appropriate step length is a critical consideration in the numerical solution of differential equations.

4. **Stiff and Non-Stiff system:** In the context of research, J. D. Lambert characterizes stiffness as follows:

   When employing a numerical method possessing a finite region of absolute stability on a system with arbitrary initial conditions, if the method necessitates the utilization of an exceptionally small step length within a specific integration interval, relative to the smoothness of the exact solution in that range, then the system is identified as stiff during that interval (Lambert, 1977). A system is considered stiff if it contains components or features that vary widely in terms of their natural frequencies or time scales. Stiff systems often involve rapid and slow modes of response, and the stiffness of the system can lead to numerical challenges in solving the associated differential equations.

   It can be deduced that stiffness in a dynamic system refers to the difference in time scales or natural frequencies of its components. Stiff systems require special consideration in numerical simulations due to the challenges associated with solving the corresponding stiff ODEs. Non-stiff systems, on the other hand, are generally easier to simulate numerically.

5. **Algorithms or Packages:** These are computer code which implements numerical method, in addition to find the approximate/numerical method, it may perform other task such as estimating the error of a particular method, monitoring and updating the value of the step-length h and deciding which of the family of methods to employ at a particular stage in the solution (Lambert,

1977)

6. **Collocation method:** is a numerical technique used to solve ordinary differential equations, partial differential equations, and integral equations. The method involves selecting a finite-dimensional space of candidate solutions, usually polynomials up to a certain degree, and a number of points in the domain called collocation points. The idea is to select the solution that satisfies the given equation at the collocation points. The method provides high order accuracy and globally continuous differentiable solutions. (Wikipedia contributors, 2023a)

7. **Multistep and Singlestep methods:** A single-step method is a numerical method for solving ordinary differential equations that calculates the approximate solution only using the information from the current step.Examples of this are the Taylor algorithm of order K and Runge-Kutta Methods.A multistep method is a numerical method for solving ordinary differential equations that calculates the approximate solution using the information from the current step and one or more previous steps.A crucial characteristic of multistep methods is the necessity to compute prior values of y_n (where n takes on values $1, 2, 3, \ldots, N$) for f_n (where n takes on values $1, 2, 3, \ldots, N$) through alternative methods, such as Runge-Kutta methods. This is essential for obtaining accurate values when starting the utilization of multistep methods. Alternatively, if the exact solution is known, y_n can be directly calculated (Fatokun, 1992).For example:

$$y_{n+1} = y_n + h \cdot f_n \tag{1.2}$$

$$y_{n+2} = y_{n+1} + \frac{h}{2} \left[3f_{n+1} - f_n \right] \tag{1.3}$$

where

$$f_{n+i} = f(x_{n+i}, x_{n+i}), \text{for } i = 0, 1, 2, \ldots \tag{1.4}$$

A disadvantage of multistep methods is that they are not self-starting.But on the other hand, they are faster than the single-step methods. In addition, Multistep methods can be more stable and efficient for certain types of ODEs, especially when dealing with stiff systems (Fatokun, 1992).

8. **Linear Multistep Method (LMM):** A numerical method for approximating

the solution of an ordinary differential equation (ODE) at discrete time points using a linear combination of past and present function values. LMMs typically involve using multiple previous function values to compute the next value, hence the term "multistep".The general linear multistep method is given as

$$\sum_{j=0}^{k} \alpha_j y_{n+j} = h \sum_{j=0}^{k} \beta_j f_{n+j} \tag{1.5}$$

9. **Explicit Method:** A type of linear multistep method in which the value of the unknown function at the next time step is explicitly computed using only known values at previous time steps. The formula for the next value does not involve solving any equations or iterative procedures.

10. **Implicit Method:** A type of linear multistep method in which the value of the unknown function at the next time step is computed using known values at previous time steps, as well as the value at the next time step itself. The formula for the next value involves solving equations or iterative procedures, making implicit methods more computationally intensive than explicit methods.

   Adams-Bashforth Method: A specific family of explicit linear multistep methods used for numerical integration of ordinary differential equations. Adams-Bashforth methods use interpolation of previous function values to approximate the derivative of the function, allowing for the computation of future function values.

   Adams-Moulton Method: A specific family of implicit linear multistep methods used for numerical integration of ordinary differential equations. Adams-Moulton methods involve using interpolation of previous function values, as well as the value at the next time step, to compute the next function value, typically requiring solving equations or iterative procedures.

11. **Stability:** A property of linear multistep methods indicating the behavior of the numerical solution with respect to small perturbations or errors. A stable method produces a solution that does not grow exponentially with time and remains bounded, ensuring accuracy and reliability of the numerical solution.

12. **Order and Error constant**

   In numerical analysis, particularly in the context of solving ordinary differential equations (ODEs), the concepts of **order** and **error constant** are fundamental

in assessing the accuracy and efficiency of numerical methods.

The *order* of a numerical method refers to the rate at which the error decreases as the step size h decreases. More formally, if a numerical method approximates the solution to an ODE with a local truncation error that is proportional to h^{p+1}, where h is the step size and p is a positive integer, then the method is said to be of *order p*.

Mathematically, for a method with step size h, if the local truncation error $\tau(h)$ satisfies

$$\tau(h) = C \cdot h^{p+1} + \mathcal{O}(h^{p+2}) \tag{1.6}$$

where C is a constant, the method is of order p (J. Butcher, 2009).

The order indicates how quickly the global error decreases as the step size decreases. Higher-order methods are generally more accurate but may require more computational effort per step.

Error Constant

The *error constant* is the coefficient C in the leading term of the local truncation error expression.

The error constant C provides a measure of the accuracy of the method for a given step size. While the order p determines the rate at which the error decreases as h decreases, the error constant C affects the absolute magnitude of the error for a given h (Atkinson, 1989).

A smaller error constant means the method is more accurate for the same step size, even if two methods have the same order.

13. **Convergence:** The necessary conditions for a linear multistep method of (1.5) is said to be convergent if and only if it is consistent and zero stable A property of linear multistep methods indicating the behavior of the numerical solution as the step size approaches zero. A convergent method produces a solution that approaches the true solution of the ordinary differential equation as the step size decreases, ensuring accuracy and consistency of the numerical solution (Deborah .I.A, 2021).

14. **Order of Accuracy:** A measure of the accuracy of a linear multistep method, indicating the rate at which the numerical solution approaches the true solution

as the step size decreases. Higher-order methods have higher order of accuracy, meaning they converge to the true solution faster as the step size decreases.

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

There are many processes in science, management and technology that involves the rate of change of one variable in relation to another modeled as differential equations."Most differential equations in science and technology are solved by numerical methods"(S.L & II, 1989) because analytic solution is not possible ot useful (Lambert, 1977).They are many existing algorithm designed for ODE such as Runge-kutta and Euler's Methods, Taylor series method (Lambert, 1977), Hybrid methods by Ademiluyi(1987), Collocation method by Awoyemi(1996,1999 and 2000).However this paper addresses the use of linear multistep method to provide solutions to this differential problems.

One of the more challenging classes of problems in numerical computation is the solution of stiff equations and stiff systems. These problems arise from various physical situations but were likely first identified in chemical kinetics. Finding numerical solutions to stiff systems has been a significant challenge for numerical analysts. A potentially good numerical method for solutions of stiff systems must possess certain qualities in terms of its region of absolute stability and accuracy (Quresh, Ramos, Soomro, Akinfenwa, & Akanbi, 2024).

2.2 Exploring Stiff Systems of Ordinary Differential Equations: Characteristics, Solutions, and Numerical Methods

Stiff ordinary differential equations (ODEs) pose significant challenges in scientific and engineering applications due to their unique properties, which can lead to numerical instability and inaccuracy in traditional numerical methods.Stiff ODEs are characterized by a rapid change in the solution over a short period, followed by a period of slow change. This characteristic can lead to numerical instability in traditional numerical methods, making it difficult to accurately solve these equations over long time scales.

The earliest detection of stiffness in differential equations in the digital computer era, by the two chemists, Curtiss and Hirschfelder (Curtiss & Hirschfelder, 1952), was apparently far in advance of its time. They named the phenomenon and spotted the nature of stiffness (stability requirement dictates the choice of the step size to be very small).To resolve the problem they recommended possible methods such as the Backward Differentiation Formula (S̈uli & Mayers, 2003)

2.2.1 Characteristics

Stiff systems, in the realm of differential equations, possess specific mathematical characteristics that differentiate them from non-stiff systems. These characteristics often arise due to the underlying dynamics and structure of the system, particularly in the context of solving differential equations numerically. some of the main characteristics of stiff systems:

- Multiple Time Scales: Stiff systems typically involve processes evolving at vastly different rates. Some components of the system change rapidly, while others evolve slowly. This time scale separation can lead to numerical stiffness, where traditional integration methods become inefficient or unstable.

- Eigenvalue Spectrum: Stiff systems exhibit eigenvalues with a wide range of magnitudes. This spectrum includes both large and small eigenvalues, with some eigenvalues dominating the behavior of the system. The presence of large eigenvalues relative to the integration step size contributes to stiffness.

- High Condition Number: Stiff systems often have matrices with high condition numbers. A high condition number indicates that the matrix is ill-conditioned, meaning small changes in the input can lead to large changes in the output. Ill-conditioning exacerbates numerical instability and error propagation.

- Exponential Dynamics: Stiff systems may contain components that exhibit rapid exponential growth or decay. These exponential dynamics can lead to numerical challenges, especially when using explicit integration methods that struggle to capture such behavior accurately.

- Implicitness Requirement: Stiff systems often necessitate the use of implicit numerical methods for stability. Implicit methods involve solving equations that relate the future state of the system to its current state. By incorporating future information, implicit methods can better handle stiff dynamics compared to their explicit counterparts.

- Small Stability Region: Stiff systems typically have a small stability region for explicit numerical methods. The stability region represents the range of integration parameters (e.g., step size) for which the numerical solution remains bounded. In stiff systems, the stability region can be severely limited, constraining the choice of integration parameters.

- Slow Transient Behavior: Despite the presence of fast processes, stiff systems may also exhibit slow transient behavior that requires accurate numerical representation. Balancing the resolution of fast and slow dynamics poses a challenge in numerical integration, particularly for stiff systems with widely varying time scales (Wikipedia contributors, 2023b).

In this paper, we will be focusing on linear multistep methods to tackle this stiff property, They are particularly well-suited for stiff systems due to their inherent stability properties and efficiency in handling multiple time scales.

Consider the initial value problem

$$y'(x) = -15y(x), \quad x \geq 0, \quad y(0) = 1. \tag{1}$$

The exact solution (shown in cyan) is

$$y(x) = e^{-15x}, \quad y(x) \to 0 \text{ as } x \to \infty. \tag{2}$$

We seek a numerical solution that exhibits the same behavior.

The figure (right) illustrates the numerical issues for various numerical integrators applied on the equation:

- Euler's method with a step size of $h = \frac{1}{4}$ oscillates wildly and quickly exits the range of the graph (shown in red).

- Euler's method with half the step size, $h = \frac{1}{8}$, produces a solution within the graph boundaries but oscillates about zero (shown in green).

- The trapezoidal method (that is, the two-stage Adams–Moulton method) is given by

$$y_{n+1} = y_n + \frac{1}{2}h\left(f_n + f_{n+1}\right), \tag{2.1}$$

where $y' = f(x, y)$. Applying this method instead of Euler's method gives a much better result (blue). The numerical results decrease monotonically to zero, just as the exact solution does.

2.3 Multistep Methods

Multistep methods are a category of numerical methods used to solve ordinary differential equations (ODEs), including stiff systems. They are called "multistep" because they use information from multiple previous steps to compute the next step. This makes them particularly effective for problems with stiff behavior, where the slope of the solution changes rapidly (Jeon, Bak, & Bu, 2019). One of the main strengths of multistep methods is their ability to handle temporal evolution. Because they use information from previous steps, they can adapt to changes in the behavior of the solution over time. This makes them particularly effective for problems where the solution evolves in a complex way, such as stiff systems (Jeon et al., 2019).

In terms of applications, multistep methods are widely used in various fields, including physics, engineering, and economics. They are used to solve a wide range of problems, from simulating the motion of celestial bodies to modeling economic

growth. In the context of boundary value problems (BVPs), multistep methods can be used to solve problems where the solution varies over time and space (Jeon et al., 2019).

A general linear multistep method can be expressed as:

$$\sum_{j=0}^{k} \alpha_j y_{n+j} = h \sum_{j=0}^{k} \beta_j f(x_{n+j}, y_{n+j}) \qquad (2.2)$$

where:

- k is the number of previous steps to use

- h is the step length

- α_j and β_j are the coefficients of the method

- y_{n+j} is the solution at the previous time steps

and also

$$\alpha_k = 1, |\alpha_0| + |\beta_0| \neq 0$$

Linear multistep methods are generally defined by their coefficients α_j and β_j, and the choice of these coefficients determines the order and stability properties of the method. Common examples of linear multistep methods include the backward Euler method, the Adams-Bashforth methods, and the Adams-Moulton methods (Lambert, 1977).

The solution at the next time step, y_{n+1}, can be obtained by rearranging the terms in the above formula:

$$y_{n+1} = \frac{1}{\alpha_0} \left(h \sum_{j=0}^{k} \beta_j f_{n+j} - \sum_{j=1}^{k} \alpha_j y_{n+j} \right) \qquad (2.3)$$

Linear multistep method can be classified into **Implicit** and **Explicit** LMM.

2.3.1   Explicit Multistep Method

Explicit linear multistep methods are a class of numerical methods used for solving ordinary differential equations (ODEs). These methods are designed to improve computational efficiency by utilizing information from previous steps in the solution process, rather than discarding it as in single-step methods like Euler's method (Wikipedia contributors, 2023b).

The general form of an ELMM is give as:

$$y_{n+1} = \sum_{j=0}^{k} \alpha_j y_{n+j} + h \sum_{j=0}^{k-1} \beta_j f_{n+j} \qquad (2.4)$$

Notice the $\beta_k = 0$, Hence a LMM in which $\beta_k = 0$ is called an Explicit LMM.

The key characteristic of explicit linear multistep methods is that they can directly compute the next value in the sequence without needing to solve for it, making them particularly useful for problems where computational resources are limited or where the solution needs to be obtained quickly.ELMM are also derived by interpolating previously computed points and using this interpolation to compute future solution points (Alexanderian, 2022).

The Adams-Bashforth methods are a popular example of explicit linear multistep methods. They are used to solve ordinary differential equations (ODEs) and are particularly effective for non-stiff problems. The Adams-Bashforth methods are derived from the Lagrange interpolating polynomial and are used to approximate the solution at the next time step based on the solution at previous time steps.The Adams–Bashforth methods were designed by John Couch Adams to solve a differential equation modeling capillary action, and the theory was published by Francis Bashforth in 1883. These methods are widely used due to their efficiency and accuracy, especially for problems where the solution exhibits a strong dependency on the initial conditions (Wikipedia contributors, 2023a). The Adams-Bashforth methods are widely used in scientific and engineering applications, including fluid dynamics, chemical kinetics, and climate modeling (Wong, 2020).

The general form of the Adams-Bashforth method of order s is:

$$y_{n+1} = y_n + h \sum_{j=0}^{s-1} b_j f_{n-j} \tag{2.5}$$

where:

- y_n is the known solution value at time step t_n, - h is the step size, - $f_{n-j} = f(x_{n-j}, y_{n-j})$ are the derivative values at previous time steps, and - b_j are the method-specific coefficients.

The coefficients b_j for the Adams-Bashforth methods are chosen such that the methods have order s. This means that the methods are accurate up to and including terms of order h^s in the local truncation error.

The coefficients b_j for the Adams-Bashforth methods are determined by the requirement that the method be consistent with the underlying differential equation and that it has the desired order of accuracy. Specifically, the coefficients are chosen such that the method reproduces the Taylor series expansion of the exact solution up to order s.

The Adams-Bashforth methods with $k = 1, 2, 3, 4, 5$ respectively are given by (J. C. Butcher, 2003) and (Hairer, Nørsett, & Wanner, 1993):

$$y_{n+1} = y_n + h \cdot f_n \tag{2.6}$$

$$y_{n+2} = y_{n+1} + h \left(\frac{3}{2} f_{n+1} - \frac{1}{2} f_n \right) \tag{2.7}$$

$$y_{n+3} = y_{n+2} + h \left(\frac{23}{12} f_{n+2} - \frac{4}{3} f_{n+1} + \frac{5}{12} f_n \right) \tag{2.8}$$

$$y_{n+4} = y_{n+3} + h \left(\frac{55}{24} f_{n+3} - \frac{59}{24} f_{n+2} + \frac{37}{24} f_{n+1} - \frac{9}{24} f_n \right) \tag{2.9}$$

$$y_{n+5} = y_{n+4} + h \left(\frac{1901}{720} f_{n+4} - \frac{2774}{720} f_{n+3} + \frac{2616}{720} f_{n+2} - \frac{1274}{720} f_{n+1} + \frac{251}{720} f_n \right) \tag{2.10}$$

2.3.2   Implicit Multistep Method

Implicit Linear Multistep Methods (ILMMs) are a class of numerical methods used for solving ordinary differential equations (ODEs) that are particularly advantageous in situations where the step size in an explicit method has to be chosen based on stability rather than accuracy (Alexandru, Gurnari, Ionescu, & Rădulescu,

2017).ILMMs are a subset of linear multistep methods (LMMs) that are used to solve ODEs by interpolating previously computed points and using this interpolation to compute future solution points. The key characteristic of ILMMs is that they require solving a nonlinear system of equations at each step, which makes them more stable than explicit methods, which can directly compute the next value in the sequence without needing to solve for it (Keller & Du, 2020). In (2.1), a LMM in which $\beta_k \neq 0$ is called an Implicit LMM.

In explicit methods, the step size is typically limited by the stability requirements imposed by the governing equations. For stiff ODEs, where there are rapid changes in the solution, explicit methods often necessitate very small step sizes to maintain stability. This can result in computationally expensive simulations, as a large number of steps are required to cover the solution domain adequately.

On the other hand, ILMMs allow for larger step sizes while maintaining stability, making them particularly advantageous for stiff ODEs. By implicitly incorporating information from future time steps, ILMMs can handle stiff problems more efficiently, reducing the computational cost associated with solving such equations.However, they can be more computationally expensive due to the need to solve a system of equations at each step (Thohura & Rahman, 2013).

One of the mostly widely used ILMMs is the Adams-Moulton method. The Adams-Moulton methods are a family of implicit linear multistep methods (ILMMs) commonly used for solving ordinary differential equations (ODEs). These methods are derived by considering the interpolation of the derivative function $f(x, y)$ over a given time interval.It is particularly advantageous for stiff systems, where explicit methods like the Adams-Bashforth method may require very small step sizes to maintain stability, leading to computationally expensive simulations. The Adams-Moulton method allows for larger step sizes while maintaining stability, making it more efficient for solving stiff ODEs. The Adams-Moulton method is widely used in various scientific and engineering applications where accurate and stable numerical solutions of ODEs are required. It provides a balance between accuracy and computational efficiency, making it a valuable tool for simulating dynamic systems governed by differential equations.

The three-step Adams-Moulton method is given by:

$$y_{n+3} = y_{n+2} + \frac{h}{24}\left(9f_{n+3} + 19f_{n+2} - 5f_{n+1} + f_n\right) \qquad (2.11)$$

This is the Adams-Moulton method of order 3, commonly denoted as AM(3). It is an implicit method because it requires solving for y_{n+1} implicitly in the equation. The method involves using the function values f_{n+3}, f_{n+}, and f_n to compute the value of y_{n+1}, making it suitable for stiff ODEs where explicit methods may be unstable.

Another implicit methods for stiff ODEs is the Backward Differentiation Formula (BDF).Backward Differentiation Formulas (BDFs) are a family of implicit numerical methods commonly used to solve stiff systems of ordinary differential equations (ODEs).BDFs use information from the future (at the next time level) to update the solution at the current time level. The backward nature of the method enables stability for stiff problems (Press, Teukolsky, Vetterling, & Flannery, 2007).

The general form of a BDF of order k is given by:

$$\alpha_0 y_n + \alpha_1 y_{n-1} + \alpha_2 y_{n-2} + \ldots + \alpha_k y_{n-k} = h \cdot f_n \qquad (2.12)$$

It works by approximating the solution at the next step using a polynomial of degree less than or equal to the method order. This makes BDF suitable for stiff ODEs, as it avoids the loss of accuracy associated with steep slopes in the solution.BDFs are widely implemented in numerical software packages for solving stiff ODEs. Popular implementations include ode23s and ode45s (Shampine & Reichelt, 1997),the Livermore Solver for Ordinary Differential Equations (LSODA),the Differential Algebraic System Solver (DASSL),GEAR, DIFSUB, and EPISODE (Yatim et al., 2013) which is a collection of FORTRAN subroutines designed to facilitate the automated resolution of problems, minimizing the level of effort needed when encountering potential challenges in the problem-solving process (Thohura & Rahman, 2013).

2.3.3   Analysis of Multistep Method

For a comprehensive analysis of linear multistep methods (LMMs), the paper titled "Linear Multistep Numerical Methods for Ordinary Differential Equations" by

Nikesh S. Dattani is a valuable resource. This paper provides a review of the most popular LMMs, including the Adams-Bashforth Methods, Adams-Moulton Methods, and Backwards Differentiation Formulas. It discusses these methods in terms of their order, consistency, and various types of stability, offering a general overview that does not require much prior knowledge in numerical ODEs(Dattani, 2008).

The general $k - step$ linear multistep method is given as

$$\alpha_k y_{n+k} + \alpha_{k-1} y_{n+k-1} + \ldots + \alpha_1 y_{n+1} + \alpha_0 y_n = h\left(\beta_k f_{n+k} + \beta_{k-1} f_{n+k-1} + \ldots + \beta_1 f_{n+1} + \beta_0 f_n\right)$$
$$(2.13)$$

which is equivalent to (2.1), where h is the parameter known as the **Step-size**, and $f_0, f_1, f_2 \ldots f_n$ are the values of the given function $f(x, y)$ at an equidistance of $x_n = x_0 + nh$.

The central concepts in the analysis of linear multistep methods, and indeed any numerical method for differential equations, are convergence, order, and stability. These concepts are crucial for understanding the behavior and performance of numerical methods in solving differential equations.

Zero-Stability

Stability is a property that ensures the numerical solution does not explode or oscillate as the step size is reduced. A method is considered stable if it satisfies certain conditions related to the roots of the characteristic polynomial of the method. Specifically, all roots of the characteristic polynomial must lie inside the unit circle in the complex plane. This condition ensures that the method does not amplify errors and remains accurate as the step size is reduced (Wikiversity, 2019) (Alexanderian, 2022). A scheme that is zero-stable will not produce approximations which grow unrealistically with time.The stability of a numerical method is not as tangable as consistency and convergence but when you see an unstable solution it is obvious.

The stability of multistep methods is determined by analyzing the roots of their characteristic polynomials. Methods that satisfy the root condition and have only one root with magnitude one are classified as strongly stable, while those with multiple distinct roots of magnitude one are termed weakly stable. Multistep methods that fail to meet the root condition are considered unstable (Butler, 2022).With this said it is safe to say that All one step methods, Adams-Bashforth and Adams-

Moulton methods are all stongly stable.

Given the **First characteristic polynomial**

$$\rho(z) = \alpha_0 + \alpha_1 z + \alpha_2 z^2 + \cdots + \alpha_k z^k \tag{2.14}$$

where the a_i are the coefficients of the LMM. LMM method are said to be **zero stable** if the zeros of the first characteristic polynomial such that

1. None of the zeros of the first characteristic polynomial are larger than 1 in magnitude.

2. Any zero equal to 1 in magnitude is simple (i.e., not repeated).

Convergence

Convergence refers to the property of a numerical method to produce a solution that approaches the exact solution as the step size approaches zero. A method is said to be convergent if the error between the numerical solution and the exact solution decreases as the step size is reduced. The rate of convergence is often expressed in terms of the order of the method, which indicates how quickly the error decreases with each step (J.O. Fatokun, 2022).

If the Linear multistep method satisfies the zero stable scheme and consistency scheme then it is said to be **convergent** (J.O. Fatokun, 2022).

Consistency and Order

A scheme that is zero-stable will produce approximations that do not grow in size in a way that is not present in the exact, analytic solution. Zero stability is a required property, but it is not enough on its own. There remains the issue of whether the approximations are close to the exact values. The truncation error of the general linear multistep method is a measure of how well the differential equation and the numerical method agree with each other. It is defined by

$$\tau_j = \frac{1}{\beta}\left(c_0 hy(jh) + c_1 y'(jh) + c_2 hy''(jh) + c_3 h^2 y'''(jh) + \ldots\right) = \beta h \sum_{p=0}^{\infty} c_p h^p y^{(p)}(jh)$$
(2.15)

where $\beta = \sum \beta_j$ is a normalizing factor (J.O. Fatokun, 2022). It is the first few terms in this expression that will matter most in what follows, and it helps us that there are formulae for the coefficients which appear:

$$c_0 = \sum \alpha_j, \quad c_1 = \sum (j\alpha_j - \beta_j), \quad c_2 = \sum \left(j^2 \alpha_j - j\beta_j\right), \quad c_3 = \sum \left(\frac{j^3}{3!}\alpha_j - \frac{j^2}{2!}\beta_j\right)$$
(2.16)

and so on. The general formula for $p \geq 2$ is

$$c_p = \sum_{j=0}^{k} \left(\frac{(j^p)!}{p!}\alpha_j - \frac{(j^{p-1})}{(p-1)!}\beta_j\right),$$
(2.17)

Recall that the truncation error is intended to be a measure of how well the differential equation and its approximation agree with each other(HELM Green, Harrison, & Ward, 2008). In summary we can deduct that a linear multistep scheme is **consistent** if $c_0 = 0$ and $c_1 = 0$.

Assuming that the method is consistent, the order of the scheme tells us how quickly the truncation error tends to zero as $h \to 0$. For example, if $c_0 = 0$, $c_1 = 0$, $c_2 = 0$, and $c_3 \neq 0$, then the first nonzero term in τ_j will be the one involving h^2, and the linear multistep method is called second-order. This means that if h is small, then τ_j is dominated by the h^2 term (because the h^3 and subsequent terms will be tiny in comparison), and halving h will cause τ_j to decrease by a factor of approximately $\frac{1}{4}$. The decrease is only approximately known because the h^3 and other terms will have a small effect.In conclusion a linear multistep method is said to be of **order** p if $c_0 = c_1 = c_2 = \ldots = c_p = 0$ and $c_{p+1} \neq 0$ (HELM Green et al., 2008).

Numerical Solvers for Solving Stiff Systems

However, like all numerical methods, multistep methods have their limitations. For example, they can suffer from numerical diffusion, where the solution becomes smoother than expected due to roundoff errors. This can lead to inaccuracies in the solution, especially for problems with stiff behavior. Furthermore, the choice of the number of previous steps to use can significantly affect the performance of the method. More steps can lead to more accurate solutions, but they also increase the computational cost (Jeon et al., 2019).

The `ode15s` and `ode23` solvers in MATLAB are examples of multistep methods used for solving stiff systems of BVPs or IVPs. The `ode15s` solver uses an implicit Runge-Kutta method of order 15, with the embedded 6th order BDF method as a predictor. It is able to handle stiff and nonstiff problems and can be used with either the Jacobian of the system or a numerical approximation. On the other hand, the `ode23` solver uses an implicit Runge-Kutta method of order 2, with the embedded 3rd order BDF method as a predictor. It is also able to handle stiff and nonstiff problems (Wong, 2020).

The `bvp4c` and `bvp5c` solvers in MATLAB are examples of multistep methods used for solving BVPs. They use a collocation method with a finite difference code that implements the Lobatto IIIa formula.

Each of these tools has its strengths and weaknesses; `DIFSUB` has no graphical user interface, which makes it harder for users who are not comfortable working from the command-line or with text-based interfaces, and it is a very old package, therefore finding support becomes challenging; `GEAR` uses a FORTRAN package, users need to have some knowledge of FORTRAN to use it effectively; `EPISODE` is a deprecated package which performs faster than `GEAR` in solving waves or active solutions, but the reverse for linear or decaying problems (Byrne et al., 1977). With these limitations comes the aim of this project.

The Runge-Kutta-Fehlberg (RKF45) method is another widely used implicit method for solving ordinary differential equations, including stiff ODEs. It combines both explicit and implicit methods to achieve high accuracy and stability (Stone, Alferman, & Niemeyer, 2017).

$$k_1 = h \cdot f(x_n, y_n),$$

$$k_2 = h \cdot f\left(x_n + \frac{1}{4}h, y_n + \frac{1}{4}k_1\right),$$

$$k_3 = h \cdot f\left(x_n + \frac{3}{8}h, y_n + \frac{3}{32}k_1 + \frac{9}{32}k_2\right),$$

$$k_4 = h \cdot f\left(x_n + \frac{12}{13}h, y_n + \frac{1932}{2197}k_1 - \frac{7200}{2197}k_2 + \frac{7296}{2197}k_3\right),$$

$$k_5 = h \cdot f\left(x_n + h, y_n + \frac{439}{216}k_1 - 8k_2 + \frac{3680}{513}k_3 - \frac{845}{4104}k_4\right),$$

$$k_6 = h \cdot f\left(x_n + \frac{1}{2}h, y_n - \frac{8}{27}k_1 + 2k_2 - \frac{3544}{2565}k_3 + \frac{1859}{4104}k_4 - \frac{11}{40}k_5\right).$$

$$y_{n+1} = y_n + \frac{16}{135}k_1 + \frac{6656}{12825}k_3 + \frac{28561}{56430}k_4 - \frac{9}{50}k_5 + \frac{2}{55}k_6.$$

$$\text{Error} = \frac{1}{360}h(-127k_1 + 845k_3 - 28561k_4 + 9k_5 - 2k_6).$$

Although there exist no specific stiff system solver that utilizes the RKF45 method exclusively, the RKF45 method is a variant of the Dormand-Prince method, which is a popular implicit Runge-Kutta method used in MATLAB's `ode15s` solver (Burkardt, 2010). It's worth noting that while the RKF45 method is not used by a specific stiff system solver, it is a powerful tool for solving stiff systems of ODEs. Its ability to accurately estimate the local truncation error and adapt the step size accordingly allows it to handle stiff problems effectively (Burkardt, 2010).

2.4 Collocation Methods

Collocation methods are a class of numerical methods used to solve ordinary differential equations (ODEs), including stiff systems. They work by choosing specific points (collocation points) within the domain where the solution is sought. The solution is then approximated as a polynomial at these points, and the differential equation is converted into a set of algebraic equations by enforcing the equations at these points. Numerous studies have demonstrated the efficacy of collocation methods in various scientific and engineering applications. Examples include the modeling of chemical reactions, structural dynamics, and climate phenomena. The ability of collocation methods to efficiently capture rapid changes in the system dy-

namics makes them well-suited for problems characterized by stiff components. In
the realm of stiff systems, collocation methods exhibit notable advantages, offer-
ing enhanced efficiency by directly manipulating the coefficients of the differential
equation, ensuring heightened accuracy in capturing stiff behavior, and providing
increased stability; however, their implementation complexity and sensitivity to the
choice of collocation points present challenges(Faleichik, 2009).

Let's consider a simple second-order ordinary differential equation (ODE) as an
example:

$$y''(x) = f(x, y, y')$$

with boundary conditions $y(a) = \alpha$ and $y(b) = \beta$. The objective is to find the
function $y(x)$ that satisfies the differential equation and boundary conditions.

The collocation method involves selecting a set of collocation points $\{x_1, x_2, \ldots, x_n\}$
within the domain $[a, b]$. At these collocation points, the differential equation is en-
forced. This results in a system of algebraic equations that can be solved to obtain
the values of $y(x_i)$ at the collocation points.

Let $y_i = y(x_i)$ and $y'_i = y'(x_i)$. Applying the collocation method to the differential
equation, we have:

$$y''_i = f(x_i, y_i, y'_i)$$

This equation is enforced at each collocation point x_i, resulting in a set of alge-
braic equations:

$$y''_1 = f(x_1, y_1, y'_1)$$
$$y''_2 = f(x_2, y_2, y'_2)$$
$$\vdots$$
$$y''_n = f(x_n, y_n, y'_n)$$

The accuracy and stability of the collocation method depend on the choice of

collocation points and the method used to solve the resulting system of equations.

CHAPTER THREE

METHODOLOGY

3.1 Introduction

The methodology section serves as the backbone of this project, providing a comprehensive understanding of the algorithms approach used to analyze linear multistep methods (LMMs) and solve ordinary differential equation (ODE). It outlines the systematic procedures, techniques, and tools utilized throughout the solver-project lifecycle, shedding light on the intricacies of our analysis and solution methodology.This section plays a pivotal role in elucidating how we approached the analysis of LMMs and their application in solving ODE questions. It provides clarity on the selection of LMMs, the formulation of numerical algorithms, the validation of results, and the integration of computational techniques into a cohesive framework.

3.2 Flutter as a Development tool

A notable aspect of this solver is the utilization of Flutter, Google's open-source UI software development kit, for building the desktop application. Flutter was chosen as the development framework due to its versatility and efficiency in creating cross-platform applications that run seamlessly on various operating systems, including Windows, macOS, and Linux.

The decision to adopt Flutter stems from its numerous advantages for desktop app development. Firstly, Flutter offers a single codebase that can be used to target multiple platforms, eliminating the need to maintain separate codebase for different

operating systems. This not only streamlines the development process but also ensures consistency in the user experience across platforms and it also implies that the solver will run any of the following operating system ranging from the IOS for mobile users to Windows for desktop to users, but currently we will be sticking to only desktop apps preferably the Windows operating system.

Additionally, Flutter provides a rich set of widgets and tools for designing visually appealing and interactive user interfaces. The flexibility of Flutter's UI framework allows for the creation of custom UI components tailored to the specific requirements of our desktop application. This is particularly advantageous for visualizing numerical data and facilitating user interactions with the ODE solver.

Furthermore, Flutter boasts excellent performance characteristics, thanks to its high-performance rendering engine, Dart language optimization, and ahead-of-time compilation. This ensures smooth and responsive user experiences, even when performing complex numerical computations within the application.

By leveraging Flutter for desktop app development and also Dart(*flutter is written in dart*), we aim to deliver a robust and user-friendly application that combines the power of LMM analysis with intuitive UI design. The following sections will delve deeper into the methodology employed, including the design considerations, integration of LMM algorithms, validation techniques, and deployment strategies.

The development of the solver is divided into two modules, the first module which involves the development of the algorithms and UI for the analysis of the linear multistep method, and the second module which involves using the method to solve a particular problem.

3.3 Module 1: Analysis of Linear multistep method

The general $k - step$ linear multistep method takes the form

$$y_{n+k} + \alpha_{k-1}y_{n+k-1} + \cdots + \alpha_0 x_n = h(\beta_k f_{n+k} + \beta_{k-1}f_{n+k-1} + \cdots + \beta_0 f_n) \quad (3.1)$$

which is equal to

$$\sum_{j=0}^{k} \alpha_j y_{n+j} = h \sum_{j=0}^{k} \beta_j f_{n+j} \tag{3.2}$$

(J.O. Fatokun, 2022)

The properties such as **Consistency** ,**Zero Stability** ,**Convergence** are investigated, also the **Error constant** and **Order** is also calculated by the software.

3.3.1   Consistency Analysis Algorithm

Consistency is a crucial property of linear multistep methods (LMMs) used to solve ordinary differential equations (ODEs). A consistent LMM has a local truncation error (LTE) that tends to zero as the step size decreases. This property ensures that numerical approximations are close to the exact solutions.

To determine if a given LMM is consistent, one approach is to evaluate the local truncation error (LTE) and verify that it approaches zero as the step size decreases. Alternatively, the consistency conditions can be used to check whether the leading order term of the LTE is zero. The following formulas allow you to assess the consistency of an LMM:

$$c_0 = \sum_{i=0}^{k} \alpha_i, \tag{3.3}$$

where α_i represents the coefficients for the terms involving the dependent variable y_{n+i}.

The second consistency condition is given by:

$$c_1 = \sum_{i=0}^{k} (i\alpha_i - \beta_i), \tag{3.4}$$

where β_i are the coefficients for the terms involving the function $f(x_{n+i}, y_{n+i})$.

An additional consistency condition for higher orders is defined by:

$$c_p = \sum_{j=0}^{k} \left(\frac{(j^p)!}{p!} \alpha_j - \frac{(j^{p-1})}{(p-1)!} \beta_j \right), \tag{3.5}$$

which applies for $p > 2$. This will be explored in other analysis schemes.

To confirm consistency, check if both c_0 and c_1 are zero. If these conditions are met, then the LMM is consistent. In this case, c_0 can be derived from the sum of the alpha coefficients, and c_1 from the alpha coefficients with indices multiplied by their values minus the beta coefficients.

To ensure the accuracy of user inputs in the software, remember that a one-step linear method should have two alpha coefficients and two beta coefficients, leading to a total of four parameters. For a two-step method, the total number of coefficients should be six, indicating that the total number of alpha and beta coefficients required is $2 + 2 \times k$, where k is the step number. This validation helps prevent runtime errors due to incorrect inputs.

The following section outlines the algorithm used to check for consistency in more detail.

Algorithm 1 Checking Consistency of a Linear Multistep Method
---
1: Initialize the number of steps in the LMM: $kSteps$
2: Initialize the alpha coefficients: α
3: Initialize the beta coefficients: β
4: Ensure the total number of coefficients is $2 \times kSteps + 2$. If not, throw an error.
5: Calculate $c_0 = \sum_{j=0}^{kSteps} \alpha_j$
6: Calculate the sum of beta coefficients: $\sum_{j=0}^{kSteps} \beta_j$
7: Calculate the sum of alpha coefficients multiplied by their indices:

$$sumOfAlphaMultipliedByIndex = \sum_{j=0}^{kSteps} j \times \alpha_j$$

8: Calculate $c_1 = sumOfAlphaMultipliedByIndex - \sum_{j=0}^{kSteps} \beta_j$
9: Check if c_0 and c_1 are approximately zero. If both are zero, return 'true' (consistent). Otherwise, return 'false' (inconsistent).

By following this algorithm, you can determine whether a linear multistep method is consistent. If both $c0$ and $c1$ are approximately zero, then the local truncation error tends to zero, indicating that the method is consistent. If either of these values is not zero, then the method is not consistent.

3.3.2   Zero-Stability Analysis Algorithm

The necessary and sufficient condition for a given LMM to be zero-stable as discussed in (J.O. Fatokun, 2022) is for it to satisfy the root condition which is also known as the **Dahlquist root condition** (Lambert, 1977). The proof for this is discussed in (Keller & Du, 2020).

if the zeros of the first characteristics polynomial are such that

$$\rho(z) = \sum_{j=0}^{k} \alpha_j z^j \tag{3.6}$$

, are such that: *i.* none is greater than 1 in **magnitude**, and *ii.*any zero equal to 1 in magnitude is simple (that is, not repeated). When this properties are satisfied then we say that the LMM is **zero-stable**.

The need for finding the roots of the first polynomial arises, and there exist many root finding method, techniques or algorithm. These root finding method all do have advantages and disadvantages over the other.Methods such as Bisection, Newton's Iteration, Secant methods which are all Algebraic method where consider in finding the roots of the first characteristics polynomial especially for polynomial of degree greater than 2. This method proved to be ineffective since they only consider one of the many possible roots of the characteristics polynomial.

The Durand-Kerner method, also known as the Weierstrass method, is a root-finding algorithm used for solving polynomial equations numerically. It was initially discovered by Karl Weierstrass in 1891 and later rediscovered independently by Durand in 1960 and Kerner in 1966.The Durand-Kerner method operates by iterating through a set of initial guesses for the roots of the polynomial. For each iteration, the method adjusts these guesses based on the polynomial's values at those points, aiming to converge towards the actual roots. The convergence of the Durand-Kerner method is not guaranteed for all polynomials; there are cases where the method fails to converge to the roots, instead converging to periodic cycles or other non-root values, but despite its potential limitations, the Durand-Kerner method can be effective in practice, especially when the roots are well-separated and the initial guesses are reasonably close to the actual roots.

The Durand-Kerner Method algorithms is given as follows:

Algorithm 2 Durand-Kerner Algorithm

Require: Polynomial $f(x) = a_0 + a_1 x + \ldots + a_n x^n$, initial guesses $x_1, \ldots, x_n$, tolerance ϵ, maximum iterations.
Ensure: List of roots $x_1^{(k+1)}, \ldots, x_n^{(k+1)}$.
  Initialize tolerance ϵ and maximum iterations.
  Initialize roots $x_1, \ldots, x_n$.
  for each iteration $k = 1$ to maximum iterations **do**
    for each root x_i **do**
      Compute:
$$x_i^{(k+1)} = x_i^{(k)} - \frac{f(x_i^{(k)})}{\prod_{j \neq i}(x_i^{(k)} - x_j^{(k)})}.$$
    end for
    Compute changes $\Delta x_i = |x_i^{(k+1)} - x_i^{(k)}|$.
    if all $\Delta x_i < \epsilon$ **then**
      Converged.
      break.
    end if
  end for
  if converged **then**
    return $x_1^{(k+1)}, \ldots, x_n^{(k+1)}$
  else
    return Error: Non-convergence
  end if

The Durand-Kerner method is employed to find the roots of the first characteristic polynomial due to its efficiency and accuracy. This method is capable of calculating all the roots of the polynomial simultaneously. For 2-step the quadratic formula is used but for higher order polynomials, the Durand-Kerner method is employed.

After evaluating the roots of the first characteristic polynomial, the next step is to check if the roots satisfy the Dahlquist root condition. If all the roots have magnitudes less than 1 and any root with magnitude equal to 1 is simple (non-repeated), then the LMM is zero-stable. This analysis is crucial for determining the stability of the LMM and ensuring the accuracy of the numerical solutions.

The following algorithm outlines the process of checking the zero-stability of a linear multistep method using both the Durand-Kerner method and the Dahlquist root condition:

Algorithm 3 Algorithm for Zero Stability in Linear Multistep Methods

---

Require: List of coefficients 'alpha', number of steps 'kSteps'
Ensure: Boolean 'isZeroStable' indicating zero stability
  1: Initialize an empty list 'algebraicSolution'
  2: Reverse the list 'alpha' to create 'reversedAlpha'
  3: Initialize a flag 'stable = True'
     {Create the characteristic polynomial based on 'kSteps'}
  4: **if** 'kSteps == 1' **then**
  5:    Create a linear polynomial with 'alpha[1]', 'alpha[0]'
  6: **else if** 'kSteps == 2' **then**
  7:    Create a quadratic polynomial with 'alpha[2]', 'alpha[1]', 'alpha[0]'
  8: **else if** 'kSteps == 3' **then**
  9:    Create a cubic polynomial with 'alpha[3]', 'alpha[2]', 'alpha[1]', 'alpha[0]'
10: **else**
11:    Use Durand-Kerner method to create the characteristic polynomial from 'reversedAlpha'
12: **end if**
     {Solve for roots and check stability}
13: Solve the characteristic polynomial to find roots, storing them in 'algebraicSolution'
14: **for** each 'root' in 'algebraicSolution' **do**
15:    **if** Modulus('root') ¿ 1 **then**
16:      Set 'stable' to 'False'
17:    **else if** Modulus('root') == 1 and there is more than one such root **then**
18:      Set 'stable' to 'False'
19:    **end if**
20: **end for**
     {Return result based on stability check}
21: **if** 'stable' **then**
22:    **return** 'True'
23: **else**
24:    **return** 'False'
25: **end if**

This pseudocode outlines the steps to check zero stability for a Linear Multistep Method (LMM). It includes initialization, creating the characteristic polynomial, solving for roots, and checking the modulus of the roots to determine stability. If any root's modulus exceeds one or if there's more than one root with modulus equal to one, the method is not zero-stable. The algorithm uses conditional checks and loops to examine the roots, ensuring a correct result.

3.3.3 Convergence Analysis Algorithm

The condition for a LMM to be convergent has been discuued already, the scheme must be consistent and zero-stable. The convergence of a LMM is crucial for ensuring that the numerical solutions approach the exact solutions as the step size decreases. By combining consistency and zero-stability, we can determine if a given LMM is convergent.

The zero-stability and consistency algorithms has been discussed already, the convergence algorithm is a combination of the two algorithms. The algorithm for checking the convergence of a linear multistep method is outlined below:

Algorithm 4 Algorithm for Convergence in Linear Multistep Methods

---

Require: List of coefficients 'alpha', List of coefficients 'beta', number of steps 'kSteps'
Ensure: Boolean 'isConvergent' indicating convergence
 1: Initialize a flag 'isConvergent = False'
    {Step 1: Check Zero Stability}
 2: Reverse the list 'alpha' to create 'reversedAlpha'
 3: Create the characteristic polynomial based on 'kSteps'
 4: Solve the characteristic polynomial to find roots
 5: Set 'isZeroStable = True' if all roots lie within or on the unit circle, otherwise 'False'
    {Step 2: Check Consistency}
 6: Initialize 'c0' as the sum of 'alpha'
 7: Initialize 'c1' as the sum of indices multiplied by 'alpha' minus the sum of 'beta'
 8: If 'c0 == 0' and 'c1 == 0', set 'isConsistent = True', otherwise 'False'
    {Step 3: Determine Convergence}
 9: **if** 'isZeroStable' and 'isConsistent' **then**
10:     Set 'isConvergent = True'
11: **end if**
12: **return** 'isConvergent'

In this algorithm:

- **Step 1: Zero Stability** The algorithm begins by checking for zero stability. A characteristic polynomial is created using the coefficients `alpha`. The roots of the polynomial are then evaluated to ensure they lie within or on the unit circle.

This step is critical because zero stability implies that small perturbations in the initial conditions will not lead to significant errors in the numerical solution.

- **Step 2: Consistency** The algorithm then checks for consistency by calculating the coefficients `c0` and `c1`. Consistency is achieved when both `c0` and `c1` are zero, indicating that the local truncation error tends to zero as the step size decreases. This step determines whether the LMM's approximation aligns with the continuous differential equation it aims to solve.

- **Step 3: Convergence** Finally, the algorithm concludes by determining convergence. If both zero stability and consistency are met, the linear multistep method (LMM) is considered convergent. This is because convergence signifies that the numerical solution approaches the exact solution as the step size tends to zero. If these conditions are satisfied, the algorithm returns `True`. Otherwise, it returns `False`.

This algorithm encompasses the essential components required to determine convergence in linear multistep methods. It can be used as a guide for building an implementation in programming languages such as Dart or Python, providing a robust framework for evaluating convergence in LMMs.

That concludes the analysis of the linear multistep method, the next section will discuss the application of the method in solving a particular problem.

3.3.4   Determination of Order and Error Constant Algorithm

The **order** of the a given LMM tells us how quickly the truncation error tends to zero as h $\rightarrow$ 0. The linear difference operator $\mathcal{L}$ of (3.1) is given as

$$\mathcal{L}[z(x); h] := \sum_{k}^{j=0} [\alpha_j z(x + jh) - h\beta_j z'(x + jh)] \tag{3.7}$$

, where $z(x) \in C^1[a, b]$ is an arbitrary function. From (Lambert, 1977), if we choose $z(x)$ to be differentiated as we need and expand $z(x + jh)$ and $z'(x + jh)$ about x, we need then obtain

$$\mathcal{L}[z(x); h] := C_0 z(x) + C_1 h z^{(1)}(x) + \cdots + C_q h^q z^{(q)}(x) + \dots \tag{3.8}$$

The Linear multistep method (3.1) and it associated difference operator defined by (3.5) is said to to be have an **order p** if in (3.6),

$$C_0 = C_1 = \cdots = C_p = 0$$

, $C_{p+1} \neq 0$, and the error constant is the value of C_{p+1} (Lambert, 1977).

With all this discussed by (Lambert, 1977), the algorithm used in obtaining **error constant** and **order** is outlined below

---
Algorithm 5 Order and Error Constant Calculation for Linear Multistep Method

---
Require: $kSteps$, $\alpha[]$, $\beta[]$
Ensure: Order of convergence and error constant.
  $c_0 \leftarrow$ empty list
  sumOfC0 $\leftarrow 0$
  errorConstant $\leftarrow 0$
  cp $\leftarrow 2$
  while True **do**
    for $i = 0$ to $kSteps$ **do**
      term $\leftarrow \frac{(i^{cp}) \cdot \alpha[i]}{(cp).factorial()} - \frac{(i^{cp-1}) \cdot \beta[i]}{(cp-1).factorial()}$
      sumOfC0 $\leftarrow$ sumOfC0 + term
      if $i = kSteps$ **then**
        approximatedR $\leftarrow$ sumOfC0.approximate(6)
        $c_0 \leftarrow c_0 +$ approximatedR
      end if
    end for
    if c_0 is not empty **then**
      errorConstant $\leftarrow$ errorConstant + sumOfC0
      if errorConstant $= 0$ **then**
        cp $\leftarrow$ cp + 1
        sumOfC0 $\leftarrow 0$
        Continue
      end if
      Break
    else
      Break
    end if
  end while
  return c_0.length, errorConstant

3.4 Module 2: Application of Linear Multistep Method in Solving ODEs

The application of linear multistep methods (LMMs) in solving ordinary differential equations (ODEs) is critical for numerical analysis. These methods offer efficient and accurate solutions for a wide range of ODE problems, including initial value

problems (IVPs) and boundary value problems (BVPs). This module outlines how to implement LMMs in a computational environment, with a focus on explicit algorithms.

Characterization of ODE Problems and Selection of Numerical Methods Algorithm

In the process of employing linear multistep methods, it is crucial to ascertain the characteristics of the Ordinary Differential Equation (ODE) problem at hand. This entails discerning whether the problem exhibits stiffness or non-stiffness and determining the most appropriate approach, be it implicit or explicit. Explicit methods typically find utility in non-stiff scenarios, offering straightforward implementations, whereas implicit techniques are favored for stiff problems owing to their inherent stability.

3.4.1   Explicit Method Algorithm

Explicit linear multistep methods (LMMs) calculate new values in a sequence using known data, avoiding the need to solve complex systems of equations. These methods are particularly useful for non-stiff problems due to their simplicity and computational efficiency. Below is an outline of the explicit linear multistep algorithm used for solving ordinary differential equations (ODEs), derived from the provided code.

When starter values are required, the fourth-order Runge-Kutta method is employed to generate these initial values. The algorithm for the fourth-order runge-kutta method is described below.

Following the generation of starter values, the explicit linear multistep method proceeds to compute subsequent values based on the specified coefficients and step sizes. The algorithm iterates through the coefficients to calculate the next value, updating the variables accordingly. The process continues until the desired number of steps is reached, generating a numerical solution to the ODE.

Algorithm 6 4-Stage Runge-Kutta Method for Starter Values

Require: Function 'func', initial values 'y0' and 'x0', 'stepSize', and number of steps 'N'
Ensure: A list 'result' containing starter values calculated using the 4-stage Runge-Kutta method

  0: Initialize an empty list 'result' with 'N' elements.
  0: Set 'x' to 'x0' and 'y' to 'y0'.
  0: **For each step** from '0' to 'N - 1':

- Calculate 'k1' as 'stepSize * func(x, y)'.

- Calculate 'k2' as 'stepSize * func(x + stepSize * 0.5, y + k1 * 0.5)'.

- Calculate 'k3' as 'stepSize * func(x + stepSize * 0.5, y + k2 * 0.5)'.

- Calculate 'k4' as 'stepSize * func(x + stepSize, y + k3)'.

- Determine the next value of 'y' using the average of 'k1', 'k2', 'k3', and 'k4', with weights: $y + (k1 + 2 \times k2 + 2 \times k3 + k4)/6$.

- Store the calculated value in the 'result[i]'.

- Update 'x' by adding 'stepSize'.

- Update 'y' to the newly calculated value for the next iteration.

  0: Return the 'result' list containing the computed starter values.

- **Initialize the Result List**: Begin by initializing a list of the desired size, filled with zeros. This list will store the computed results for the ODE solution.

- **Step-by-Step Computation**: Depending on the number of steps specified ('stepNumber'), perform the following calculations:

  - For a single-step method ($k = 1$), compute the next value based on the coefficients ('alpha', 'beta') and the step size. Update x_0 and y_0 after each step.

  - For a two-step method ($k = 2$), use a fourth-order Runge-Kutta method to generate initial values, then calculate subsequent results by iterating through the coefficients and step sizes, updating x_0 and y_0 accordingly.

  - For multi-step methods, generate the characteristic polynomial based on the coefficients, and compute results using appropriate summation and approximation techniques.

- **Error Handling and Special Cases**: Incorporate logic for handling various scenarios, such as implicit values, special cases, and error checks. This helps ensure stability and consistency during computation.

- **Return the Result**: Once all computations are complete, return the 'result' list containing the numerical solution to the ODE.

Algorithm 7 Explicit Linear Multistep Method

Require: $stepNumber$, $alpha$, $beta$, $func$, y_0, x_0, $stepSize$, N
Ensure: $result$
 1: Initialize $result$ as a list of size N filled with 0s
 2: $result[0] \leftarrow y_0$
 3: **if** $stepNumber = 1$ **then**
 4:     **for** $i = 0$ to $N - 1$ **do**
 5:         $y \leftarrow y_0 + stepSize \cdot func(x_0, y_0)$
 6:         $result[i] \leftarrow y$
 7:         $x_0 \leftarrow x_0 + stepSize$
 8:         $y_0 \leftarrow y$
 9:     **end for**
10: **else if** $stepNumber = 2$ **then**
11:     Use Runge-Kutta to compute initial values
12:     $result[1] \leftarrow y_{\mathrm{RK}}$
13:     **for** $i = 2$ to N **do**
14:         $y \leftarrow$ compute using multistep formula
15:         $result[i] \leftarrow y$
16:         Update x_0, y_0, y_1 for next iteration
17:     **end for**
18: **else**
19:     Compute initial values using Runge-Kutta
20:     Initialize x and y with initial values
21:     **for** $i = stepNumber$ to N **do**
22:         $y \leftarrow$ compute using multistep formula
23:         $result[i] \leftarrow y$
24:         Update x and y for next iteration
25:     **end for**
26: **end if**
27: **return** $result$

Summary of the Explicit Linear Multistep Method Process

1. **Initialization**:

   - Create a list `result` of size N filled with 0s.

   - Set the initial condition: `result`$[0] = y_0$.

2. **Single-Step Method (if `stepNumber` is 1)**:

   - Loop from $i = 0$ to $N - 1$:

     – Calculate y using the formula: $y = y_0 + h \cdot \text{func}(x_0, y_0)$.

     – Update `result`$[i]$ with the new y.

     – Increment x_0 by `stepSize`.

     – Update y_0 with the new y.

3. **Two-Step Method (if `stepNumber` is 2)**:

   - Use Runge-Kutta method to compute the initial value y_{RK} and set `result`$[1]$.

   - Loop from $i = 2$ to N:

     – Compute the new y using the explicit linear multistep formula.

     – Update `result`$[i]$ with the new y.

     – Update x_0, y_0, and y_1 for the next iteration.

4. **Multi-Step Method (if `stepNumber` ¿ 2)**:

   - Compute initial values using the Runge-Kutta method.

   - Initialize x and y with these initial values.

   - Loop from $i = $ `stepNumber` to N:

     – Compute the new y using the explicit linear multistep formula.

     – Update `result`$[i]$ with the new y.

     – Update x and y for the next iteration.

5. **Return the `result` list**.

This explicit linear multistep algorithm serves as a practical guide for implementing LMMs in a computational environment, providing a robust foundation for solving a variety of ODE problems. It emphasizes step-by-step computation, consistent updating of variables, and careful consideration of stability and consistency.

3.4.2   Implicit method Algorithm

The PECE (Prediction-Evaluation-Correction-Evaluation) algorithm is a specific implementation of the predictor-corrector method used in numerical analysis to solve ordinary differential equations (ODEs). It enhances the accuracy of the solution by iterating through a cycle of prediction and correction steps. The following outlines how the PECE algorithm is applied in the provided code:

- Prediction: The initial set of values (starter values) for y are computed using an explicit linear multistep method (predictor method). This step generates an initial approximation for the next values of y. In the code, this is achieved by calling the `explicitLinearMultistepMethod` with the predictor coefficients (`predictorAlpha` and `predictorBeta`).

- Evaluation: Using the predicted values, the function values $f(x_j, y_j)$ are calculated. These function values are used to compute the predictor sum (βF) and the corrector sum (αY). The code calculates these function values in a loop and stores them in the `fValues` list.

- Correction: The corrector method refines the predicted values by applying the implicit linear multistep formula. This involves calculating a new y using the corrector coefficients (`correctorAlpha` and `correctorBeta`) and the previously computed function values. The correction formula used in the code is:

$$y_{\text{next}} = \text{stepSize} \cdot \beta F - \alpha Y$$

- Evaluation: After correction, the new y value is evaluated again to ensure the accuracy and stability of the solution. This step may involve additional iterations of correction to improve the solution further. The corrected value of y is then added to the `result` list, and the algorithm updates the values of x and y for the next iteration.

Summary of the Code Implementation

The code implements the PECE algorithm as follows:

- **Initialization**: It initializes the `result` list and computes starter values using the explicit linear multistep method.

- **Prediction**: The explicit linear multistep method provides initial approximations for y.

- **Evaluation**: Function values are computed for these predicted y values.

- **Correction**: The corrector method refines these predictions using the function values and corrector coefficients.

- **Evaluation**: The corrected values are evaluated again to update the solution.

This cycle ensures that each step improves the accuracy of the numerical solution to the ODE, leveraging both predictor and corrector steps for enhanced precision and stability.

CHAPTER FOUR

IMPLEMENTATION AND NUMERICAL RESULTS

Illustrative Numerical Examples Demonstrating Solver Performance

In this section, we provide numerical examples to offer a comprehensive illustration of the accuracy and functionality of the proposed solver. These examples serve to demonstrate the solver's capabilities across various scenarios and shed light on its performance under different conditions. We carefully select three main examples that effectively showcase the solver's effectiveness and robustness in solving differential equations encountered in practical applications. Through these examples, we aim to provide insights into the solver's behavior, its accuracy in approximating solutions, and its suitability for diverse problem types.

4.1 Module 1: Analysis of Linear multistep

4.1.1 The Quade's Method

Consider the Quade's method of the form (Lambert, 1977)

$$y_{n+4} - \frac{8}{19}y_{n+3} + \frac{8}{19}y_{n+1} - y_n = \frac{6}{19}h\left(f_{n+4} + 4f_{n+3} + 4f_{n+1} + f_n\right) \qquad (4.1)$$

from the above equation, we can see that the method is a 4-step method,

where:

$$\alpha_0 = -1, \alpha_1 = \frac{8}{19}, \alpha_2 = 0, \alpha_3 = -\frac{8}{19}, \alpha_4 = 1$$
$$\beta_0 = \frac{6}{19}, \beta_1 = \frac{24}{19}, \beta_2 = 0, \beta_3 = \frac{24}{19}, \beta_4 = \frac{6}{19}$$

in other to determine the order of the Quade's method, we use (3.4), we obtain the following values:

$$c_0 = \sum_{i=0}^{4}(\alpha_i) = \frac{8}{19} - \frac{8}{19} = 0 \tag{4.2}$$

$$c_1 = \sum_{i=0}^{4}(i\alpha_i - \beta_i) = \frac{60}{19} - \frac{60}{19} = 0 \tag{4.3}$$

$$c_2 = \sum_{i=0}^{4}(\frac{i^2}{2!}\alpha_i - i\beta_i) = \frac{120}{19} - \frac{120}{19} = 0 \tag{4.4}$$

$$c_3 = \sum_{i=0}^{4}(\frac{i^3}{3!}\alpha_i - \frac{i^2}{2!}\beta_i) = \frac{168}{19} - \frac{168}{19} = 0 \tag{4.5}$$

$$c_4 = \sum_{i=0}^{4}(\frac{i^4}{4!}\alpha_i - \frac{i^3}{3!}\beta_i) = \frac{176}{19} - \frac{176}{19} = 0 \tag{4.6}$$

$$c_5 = \sum_{i=0}^{4}(\frac{i^5}{5!}\alpha_i - \frac{i^4}{4!}\beta_i) = \frac{146}{19} - \frac{146}{19} = 0 \tag{4.7}$$

$$c_6 = \sum_{i=0}^{4}(\frac{i^6}{6!}\alpha_i - \frac{i^5}{5!}\beta_i) = \frac{100}{19} - \frac{100}{19} = 0 \tag{4.8}$$

$$c_7 = \sum_{i=0}^{4}(\frac{i^7}{7!}\alpha_i - \frac{i^6}{6!}\beta_i) = 3.0682 - 3.0772 = -0.0090 \tag{4.9}$$

From the aforementioned results, it is evident that all coefficients $c_0, c_1, c_2, c_3, c_4, c_5, c_6$ are found to be zero, while c_7 evaluates to -0.0090. This analysis reveals that Quade's method exhibits a sixth-order convergence and possesses an error constant of -0.0090. Notably, these findings corroborate those reported in the study by (Emmanuel, 2018).

It scheme is also consistent since $c_0 = 0$ **and** $c_1 = 0$. The characteristics equation

of the scheme is

$$\lambda^4 - \frac{8}{19}\lambda^3 + \frac{8}{19}\lambda - 1 = 0 \tag{4.10}$$

$$19x^4 - 8x^3 + 8x - 19 = 0 \tag{4.11}$$

$$x = -1, \quad x = 1, \quad x = \frac{4}{19} + i\frac{\sqrt{345}}{19}, \quad x = \frac{4}{19} - i\frac{\sqrt{345}}{19} \tag{4.12}$$

For $x = \frac{4}{19} + i\frac{\sqrt{345}}{19}$:

$$|x| = \sqrt{\left(\frac{4}{19}\right)^2 + \left(\frac{\sqrt{345}}{19}\right)^2} = \sqrt{\frac{16}{361} + \frac{345}{361}} = \sqrt{\frac{361}{361}} = 1$$

For $x = \frac{4}{19} - i\frac{\sqrt{345}}{19}$:

$$|x| = \sqrt{\left(\frac{4}{19}\right)^2 + \left(-\frac{\sqrt{345}}{19}\right)^2} = \sqrt{\frac{16}{361} + \frac{345}{361}} = \sqrt{\frac{361}{361}} = 1$$

In this context, some of the roots are complex. Zero-stability necessitates that the absolute values have magnitudes less than or equal to 1. Consequently, we affirm that the method demonstrates **zero stability**.

Figure 4.1: α β - value collector

Figure 4.2: Result of Quade's method analysis

4.1.2 Adams Family

Method	Steps	Error Constant	β Coefficients
Adams-Bashforth	2	$\frac{5}{12}$	$\beta_1 = \frac{3}{2}, \beta_0 = -\frac{1}{2}$
Adams-Bashforth	4	$\frac{251}{720}$	$\beta_3 = \frac{55}{24}, \beta_2 = -\frac{59}{24}, \beta_1 = \frac{37}{24}, \beta_0 = -\frac{9}{24}$
Adams-Bashforth	6	$\frac{19087}{60480}$	$\beta_5 = \frac{4277}{1440}, \beta_4 = -\frac{7923}{1440}, \beta_3 = \frac{9982}{1440}, \beta_2 = -\frac{7298}{1440}, \beta_1 = \frac{2877}{1440}, \beta_0 = -\frac{475}{1440}$
Adams-Moulton	3	$\frac{1}{24}$	$\beta_0 = \frac{5}{12}, \beta_1 = \frac{2}{3}, \beta_2 = -\frac{1}{12}$
Adams-Moulton	7	$\frac{36799}{120960}$	$\beta_0 = \frac{198721}{60480}, \beta_1 = \frac{18637}{2520}, \beta_2 = -\frac{235183}{20160}, \beta_3 = \frac{10754}{945}, \beta_4 = -\frac{135713}{20160}, \beta_5 = \frac{5603}{2520}, \beta_6 = -\frac{19087}{60480}$

Table 4.1: Adams-Bashforth and Adams-Moulton Methods with Error Constants and β Coefficients

The table (4.1) shows the Error constant for The Adams-Bashforth 2,4, and 6 Step, and also that of the Adams-Moulton 3 and 7 step.

Using the JF-SM Solver we obtain the same result

For the Implicit case, that is the Adam Moulton:

Figure 4.3: α β - value collector for AB(2) method

Figure 4.4: Results of the AB(2) method analysis

Figure 4.5: Results of the AB(4) method analysis

Figure 4.6: Results of the AB(2) method analysis

Figure 4.7: α β - value collector for AM(3) method

Figure 4.8: Results of the AM(3) method analysis

Figure 4.9: α β - value collector for AM(7) method

Figure 4.10: Results of the AM(7) method analysis

4.1.3   3-Step Backward Differentiation Formula

The 3-Step Backward Differentiation Formula is given as:

$$y_{n+3} = \frac{18}{11}y_{n+2} - \frac{9}{11}y_{n+1} + \frac{2}{11}y_n + \frac{6}{11}hf_{n+3}$$

from the above equation, we can see that the method is a 3-step method,

where:

$$\alpha_0 \quad = -\frac{2}{11}, \alpha_1 = \frac{9}{11}, \alpha_2 = -\frac{18}{11}, \alpha_3 = 1$$
$$\beta_0 \qquad = 0, \beta_1 = 0, \beta_2 = 0, \beta_3 = \frac{6}{11}$$

in other to determine the order of the 3-Step BDF's method, we use (3.4), we obtain the following values:

$$c_0 = \sum_{i=0}^{3} \alpha_i = -\frac{2}{11} + \frac{9}{11} - \frac{18}{11} + 1 = 0 \tag{4.13}$$

$$c_1 = \sum_{i=0}^{3} (i\alpha_i - \beta_i) = 0 \cdot \alpha_0 + 1 \cdot \alpha_1 + 2 \cdot \alpha_2 + 3 \cdot \alpha_3 - (\beta_0 + \beta_1 + \beta_2 + \beta_3)$$

$$= 0 + 1 \cdot \frac{9}{11} + 2 \cdot \left(-\frac{18}{11}\right) + 3 \cdot 1 - \left(0 + 0 + 0 + \frac{6}{11}\right)$$

$$= \frac{9}{11} - \frac{36}{11} + 3 - \frac{6}{11} = 0 \tag{4.14}$$

$$c_2 = \sum_{i=0}^{3} \left(\frac{i^2}{2!}\alpha_i - i\beta_i\right) = \left(\frac{0^2}{2!}\alpha_0 - 0 \cdot \beta_0\right) + \left(\frac{1^2}{2!}\alpha_1 - 1 \cdot \beta_1\right) + \left(\frac{2^2}{2!}\alpha_2 - 2 \cdot \beta_2\right) + \left(\frac{3^2}{2!}\alpha_3 - 3 \cdot \beta_3\right)$$

$$= 0 + \frac{1}{2} \cdot \frac{9}{11} - 0 + \frac{4}{2} \cdot \left(-\frac{18}{11}\right) - 0 + \frac{9}{2} \cdot 1 - \frac{18}{11}$$

$$= \frac{9}{22} - \frac{36}{11} + \frac{9}{2} - \frac{18}{11} = 0 \tag{4.15}$$

$$c_3 = \sum_{i=0}^{3} \left(\frac{i^3}{3!}\alpha_i - \frac{i^2}{2!}\beta_i\right) = \left(\frac{0^3}{3!}\alpha_0 - \frac{0^2}{2!}\beta_0\right) + \left(\frac{1^3}{3!}\alpha_1 - \frac{1^2}{2!}\beta_1\right) + \left(\frac{2^3}{3!}\alpha_2 - \frac{2^2}{2!}\beta_2\right) + \left(\frac{3^3}{3!}\alpha_3 - \frac{3^2}{2!}\beta_3\right)$$

$$= 0 + \frac{1}{6} \cdot \frac{9}{11} - \frac{1}{2} \cdot 0 + \frac{8}{6} \cdot \left(-\frac{18}{11}\right) - \frac{4}{2} \cdot 0 + \frac{27}{6} \cdot 1 - \frac{9}{2} \cdot \frac{6}{11}$$

$$= \frac{9}{66} - \frac{144}{66} + \frac{27}{6} - \frac{54}{11} = 0 \tag{4.16}$$

$$c_4 = \sum_{i=0}^{3} \left(\frac{i^4}{4!}\alpha_i - \frac{i^3}{3!}\beta_i\right) = \left(\frac{0^4}{4!}\alpha_0 - \frac{0^3}{3!}\beta_0\right) + \left(\frac{1^4}{4!}\alpha_1 - \frac{1^3}{3!}\beta_1\right) + \left(\frac{2^4}{4!}\alpha_2 - \frac{2^3}{3!}\beta_2\right) + \left(\frac{3^4}{4!}\alpha_3 - \frac{3^3}{3!}\beta_3\right)$$

$$= 0 + \frac{1}{24} \cdot \frac{9}{11} - \frac{1}{6} \cdot 0 + \frac{16}{24} \cdot \left(-\frac{18}{11}\right) - \frac{8}{6} \cdot 0 + \frac{81}{24} \cdot 1 - \frac{27}{6} \cdot \frac{6}{11}$$

$$= \frac{9}{264} - \frac{288}{264} + \frac{81}{24} - \frac{162}{22} = -\frac{3}{22} = -0.13636364 \tag{4.17}$$

The characteristic polynomial $P(z)$ associated with the BDF-3 method is given by:

$$P(z) = 11z^3 - 18z^2 + 9z - 2 \tag{4.18}$$

Solving the Algebraic Equation

$$11z^3 - 18z^2 + 9z - 2 = 0 \tag{4.19}$$

$$11z^3 - 18z^2 + 9z - 2 = (z - 1)(11z^2 - 7z + 2) \tag{4.20}$$

$$11z^2 - 7z + 2 = 0 \tag{4.21}$$

$$(z - 1) = 0 \tag{4.22}$$

$$z = \frac{7 \pm \sqrt{(-7)^2 - 4(11)(2)}}{2(11)} = \frac{7 \pm \sqrt{49 - 88}}{22} = \frac{7 \pm i\sqrt{39}}{22} \tag{4.23}$$

Therefore, the roots of the polynomial $11z^3 - 18z^2 + 9z - 2 = 0$ are:

$$z = 1 \tag{4.24}$$

$$z = \frac{7 + i\sqrt{39}}{22} \tag{4.25}$$

$$z = \frac{7 - i\sqrt{39}}{22} \tag{4.26}$$

We take the modulus of the roots to check for zero-stability:

$$|z| = |1| = 1 \tag{4.27}$$

$$|z| = \sqrt{\left(\frac{7}{22}\right)^2 + \left(\frac{\sqrt{39}}{22}\right)^2} = \sqrt{\frac{49}{484} + \frac{39}{484}} = \sqrt{\frac{88}{484}} = \sqrt{\frac{22}{121}} = \frac{\sqrt{22}}{11} < 1 \tag{4.28}$$

$$|z| = \sqrt{\left(\frac{7}{22}\right)^2 + \left(-\frac{\sqrt{39}}{22}\right)^2} = \sqrt{\frac{49}{484} + \frac{39}{484}} = \sqrt{\frac{88}{484}} = \sqrt{\frac{22}{121}} = \frac{\sqrt{22}}{11} < 1 \tag{4.29}$$

Using Equation (3.4), we can see that the method is a 3-step method of order 3. The method is consistent since $c_0 = 0$ and $c_1 = 0$. Furthermore, the method is zero-stable since the roots of the characteristic equation are all inside the unit circle. Lastly, the method is convergent since the error constant is $-\frac{3}{22}$ or approximately -0.1363634.

Using the JF-SM Solver to analyze the 3-step Backward Differentiation Formula (BDF(3)), we obtain the following results as shown below:

Additionally, the JF-SM Solver software, which was developed as part of this research, has been used to analyze the BDF(3) method. The software successfully

Figure 4.11: α β - value collector for 3-step BDF method

Figure 4.12: Results of the 3-step BDF method analysis

replicated the theoretical results, providing the same error constant and stability characteristics. Moreover, the JF-SM Solver demonstrated superior performance in terms of computation speed and efficiency, making it a valuable tool for numerical analysis of differential equations.

4.2   Module 2: Solving Differential Equations

4.2.1   Explicit Method: Adams-Bashforth Method 3-Step Method

Consider the problem

$$f(x, y) = 3x^2 y \tag{4.30}$$

, with the initial condition $y(0) = 1$, the step-size also given as $h = 0.1$.

The Adams-Bashforth 3-step method is given as:

$$y_{n+3} = y_{n+2} + h \left(\frac{23}{12} f_{n+2} - \frac{4}{3} f_{n+1} + \frac{5}{12} f_n \right) \tag{4.31}$$

The exact solution to the problem is given as:

$$y(x) = e^{x^3} \tag{4.32}$$

Step n	Adams-Bashforth 3-step Method	JF-SM Solver	Exact Value
y_0	1.000000	1.000000	1.000000
y_1	1.001000	1.001000	1.001001
y_2	1.008032	1.008032	1.008032
y_3	1.027213	1.027213	1.027398
y_4	1.065494	1.065494	1.066492
y_5	1.131580	1.131580	1.133148
y_6	1.237609	1.237609	1.241857
y_7	1.401946	1.401946	1.409637
y_8	1.654090	1.654090	1.669859
y_9	2.043706	2.043706	2.073784

Table 4.2: Comparison of Results: Adams-Bashforth 3-step Method, JF-SM Solver, and Exact Values

The table illustrates the numerical values obtained at each step n. The Adams-Bashforth 3-step method and the JF-SM Solver results are compared against the exact solution $y = e^{x^3}$.

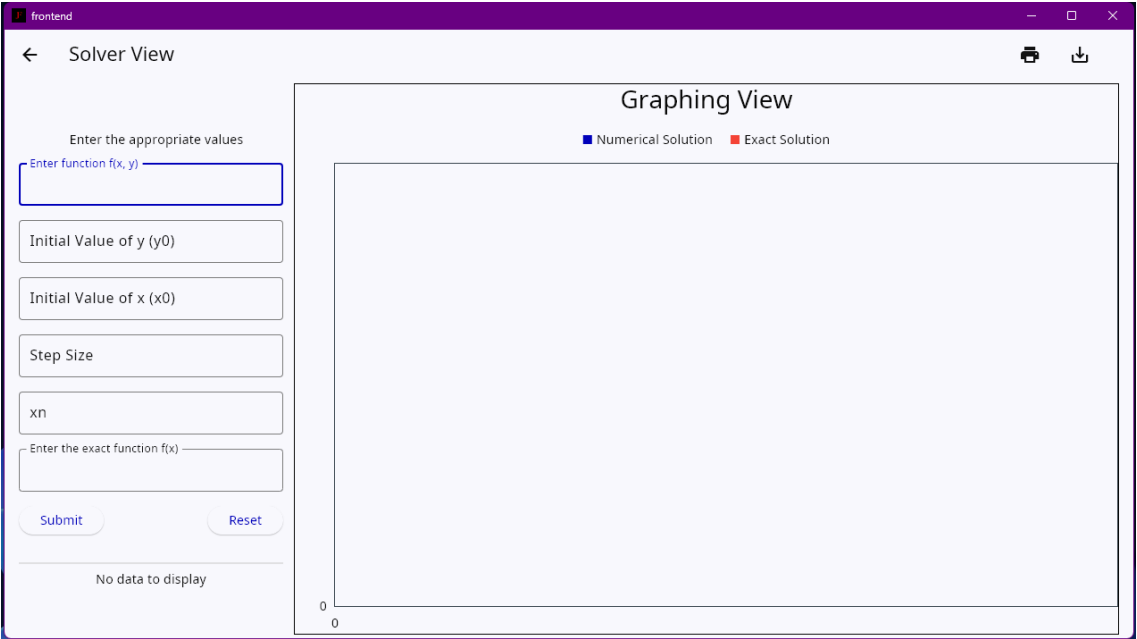The images below shows that output of the JF-SM Solver:
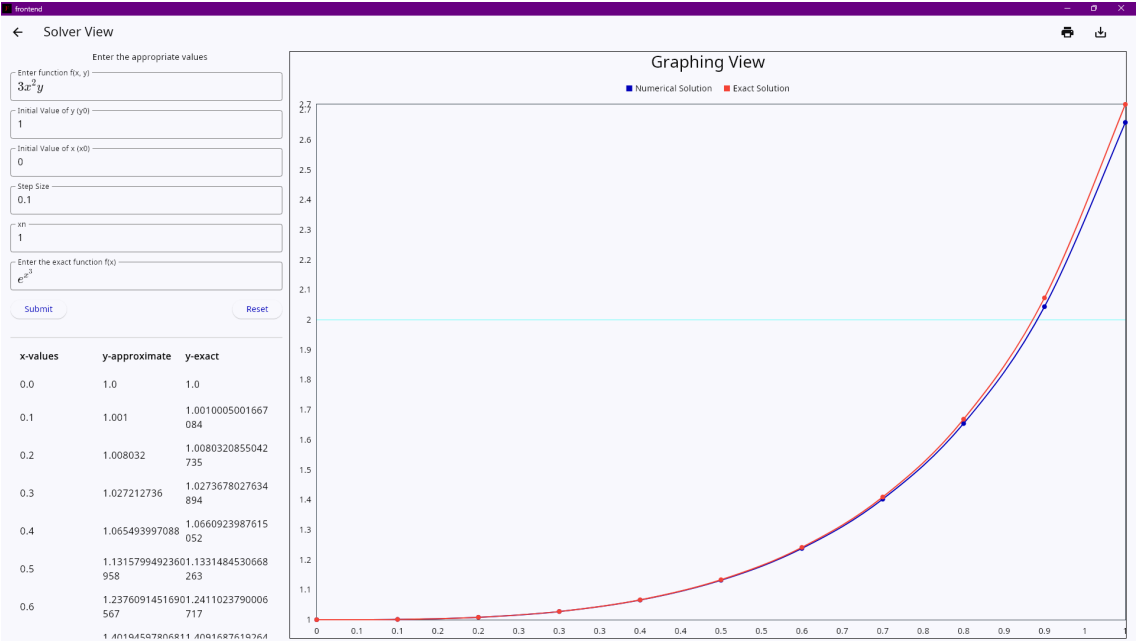
Figure 4.13: The JF-SM Solver view

Figure 4.14: output result of the JF-SM Solver for (4.30)

From the comparison, it is evident that both the Adams-Bashforth 3-step method and the JF-SM Solver produce results that closely approximate the exact solution. The JF-SM Solver results are rounded to six decimal places, showing a high degree of accuracy.

This comparison demonstrates the effectiveness, accuracy, and reliability of the JF-SM Solver in approximating the solution of explicit differential equations using the Linear Multistep Method (LMM) given. The application works by accepting the coefficients of the LMM and then solves the differential equation using these coefficients.

4.3 Implicit Method(Predictor-Corrector Algorithm): Adams-Moulton Method 4-Step Method (Non-Stiff problem)

Consider the problem

$$f(x, y) = x + y \tag{4.33}$$

, with the initial condition $y(0) = 0$, the step-size also given as $h = 0.2$.

We use two LMM methods that are of the same order, the predictor method which is an explicit method, and the corrector method which is an Implicit method, for this example, the predictor method is the 4-step Adam Bashforth given as

$$y_{n+4} = y_{n+3} + h \left(\frac{55}{24} f_{n+3} - \frac{59}{24} f_{n+2} + \frac{37}{24} f_{n+1} - \frac{9}{24} f_n \right) \tag{4.34}$$

The corrector method is the 3-step Adam-Moulton method, which is given as:

$$y_{n+3} = y_{n+2} + h \left(\frac{9}{24} f_{n+3} + \frac{19}{24} f_{n+2} - \frac{5}{24} f_{n+1} + \frac{1}{24} f_n \right) \tag{4.35}$$

both method are of the same order, that is order 4, this is determined by the JF-SM Solver, and this is discussed in (Advanced Engineering Mathematics, Erwin Kreyszig,Table 21.9, Pg 914)

we use the JF-SM Solver to get the solution (4.33) and also analysis the Predictor-Corrector method used,we get the following result:

n	x_n	Starting y_n	Predicted y_n^*	Corrected y_n	Exact Values	$10^6 \cdot$ Error of y_n
0	0.0	0.000000			0.000000	0
1	0.2	0.021400			0.021403	3
2	0.4	0.091818			0.091825	7
3	0.6	0.222107			0.222119	12
4	0.8		0.425361	0.425529	0.425541	12
5	1.0		0.718066	0.718270	0.718282	12
6	1.2		1.119855	1.120106	1.120117	11
7	1.4		1.654885	1.655191	1.655200	9
8	1.6		2.352653	2.353026	2.353032	6
9	1.8		3.249190	3.249646	3.249647	1
10	2.0		4.388505	4.389062	4.389056	-6

Figure 4.15: Predicted and Corrected Values of (4.33)

Figure 4.16: entering α and β values into the JF-SM Solver
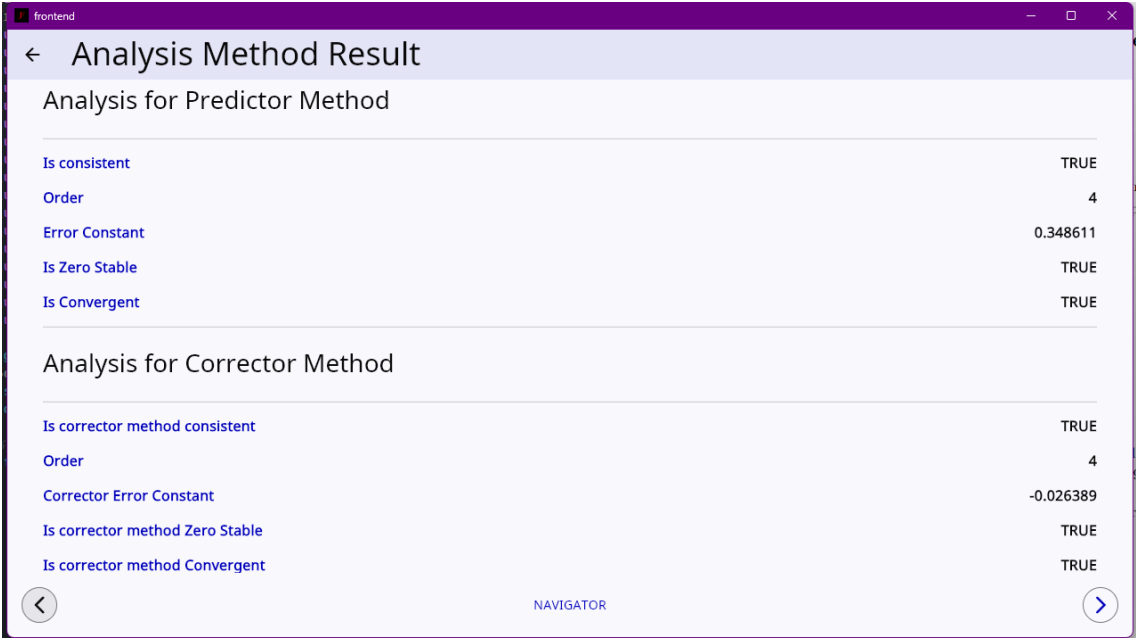
Figure 4.17: Analysis result of (4.33)
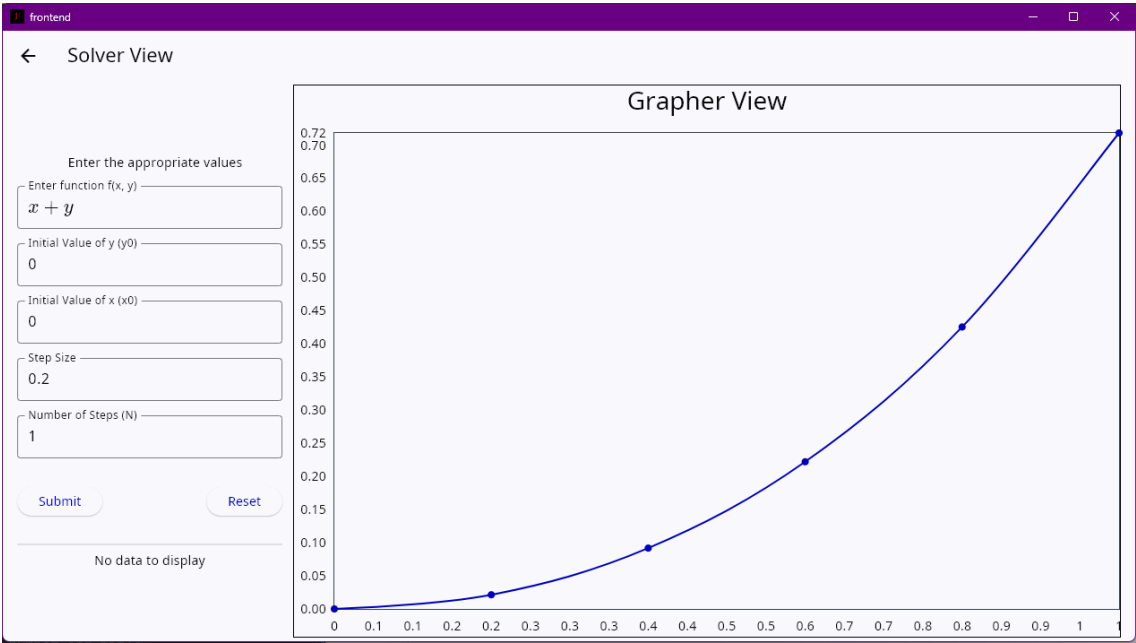
Figure 4.18: solution of (4.33) by JF-SM Solver

4.4 Implicit Method: 3-Step Backward Differentiation Formula Method (Stiff problem)

The Dahlquist Test Equation is given as:

$$y'(x) = Ky(x) \tag{4.36}$$

Where K is a constant, this equation becomes a stiff problem when K is negative. The JF-SM Solver is used to solve this equation using the 3-step Backward Differentiation Formula Method, which has been proven to be effective for solving stiff problems.

The solver is used to analyze the scheme and to solve the stiff problem:

$$y'(x) = -16y(x) \tag{4.37}$$

The solution to the problem is shown below:
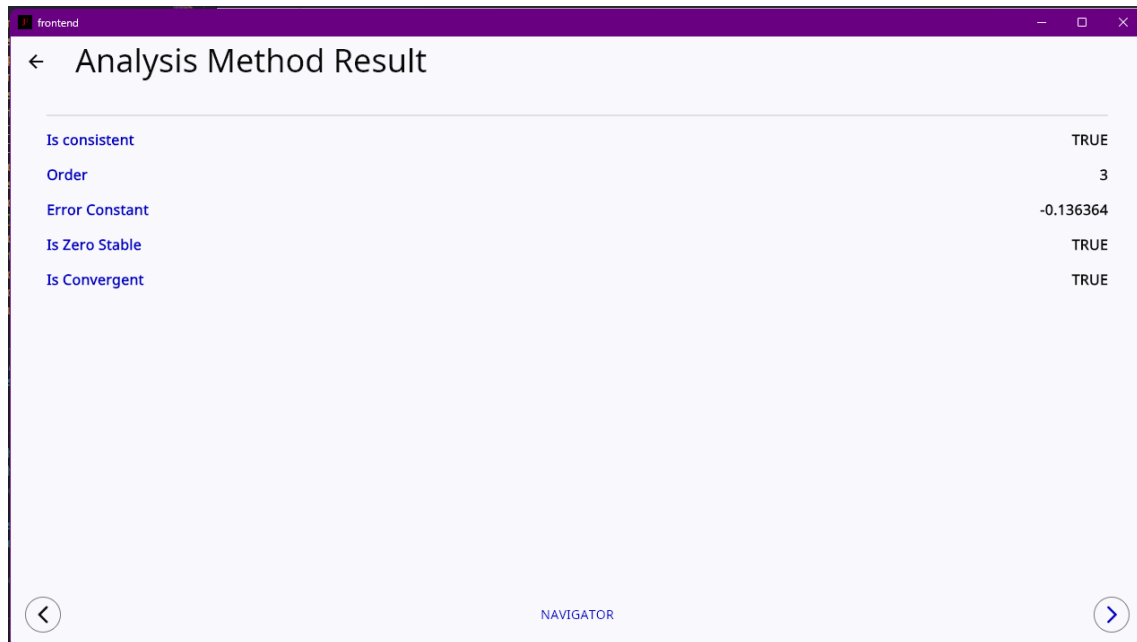
Figure 4.19: Analysis result of (4.37)
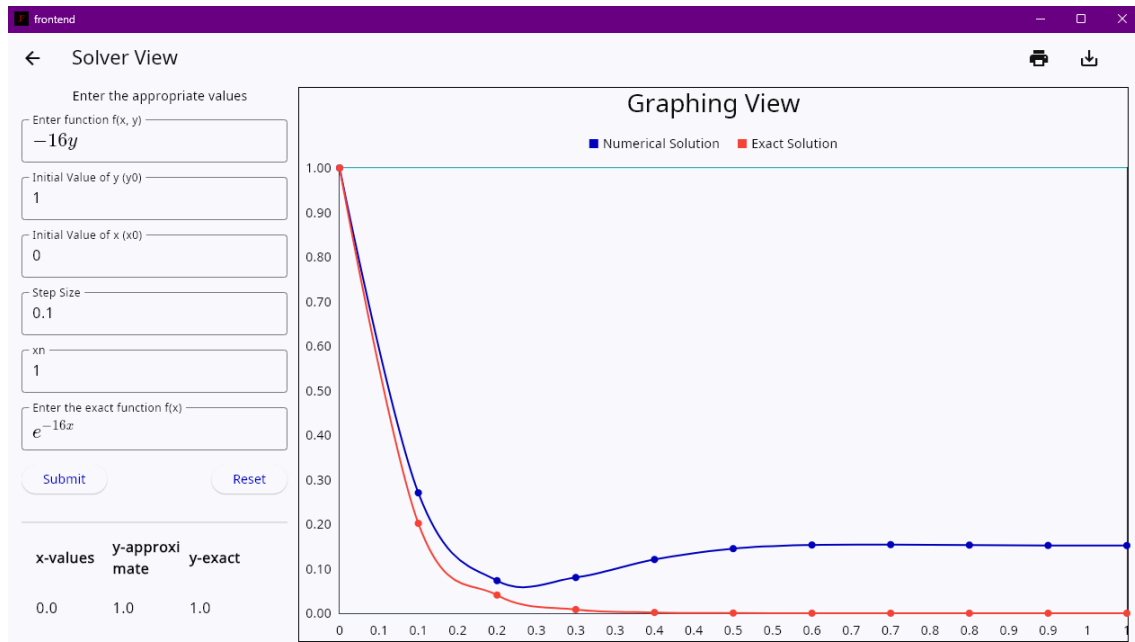
Figure 4.20: solution of (4.37) by JF-SM Solver

In conclusion, the numerical examples provided in this chapter effectively demonstrate the accuracy and robustness of the proposed solver across various methods and scenarios. The detailed analysis of Quade's method whcih is shown in Figure 4.2, Adams family methods as shown in Figure 4.3,Figure 4.4,Figure 4.5,Figure 4.6,Figure 4.7,Figure 4.8,Figure 4.9, and Figure 4.10, and the 3-step Backward Differentiation Formula Figure 4.12, showcases the solver's capability in handling both explicit and implicit differential equations. The results, corroborated by theoretical analysis and the use of the JF-SM Solver software, affirm the solver's reliability and efficiency in practical applications. This comprehensive evaluation highlights the solver's suitability for a wide range of differential equation problems, validating its potential as a valuable tool in numerical analysis.

CHAPTER FIVE

SUMMARY, CONCLUSION AND RECOMMENDATIONS

5.1 Summary

The aim of this study was to develop, validate, and optimize algorithms for the numerical analysis of linear multistep methods, particularly for solving stiff ordinary differential equations (ODEs). This was accomplished through a structured approach comprising three main objectives: Algorithm Development and Software Implementation, Validation and Error Analysis, and Optimization and Documentation. The achievements in each of these areas are thoroughly discussed in Chapter 3 of this study, and the accuracy is examined in Chapter 4.

This project focuses on the development of a desktop application called JF-SM Solver, designed to analyze and solve ordinary differential equations (ODEs) using Linear Multistep Methods (LMM). The application is built using Flutter and Dart, ensuring a modern, responsive, and cross-platform user experience. The key features of JF-SM Solver include the analysis of LMMs for any number of steps, the ability to solve both stiff and non-stiff ODEs using appropriate methods, and the implementation of graphing functionalities for visualizing solutions.

5.2   Key Features and Implementation

5.2.1   Linear Multistep Method (LMM) Analysis

JF-SM Solver provides comprehensive tools for analyzing the numerical properties of LMMs. Users can specify the number of steps for the LMM, and the application performs detailed analyses, including stability and accuracy assessments. This feature is crucial for researchers and engineers who need to understand the behavior and performance of different LMM configurations.

5.2.2   Solving Ordinary Differential Equations

The core functionality of JF-SM Solver is its ability to solve ODEs using both implicit and explicit LMMs:

- **Implicit LMM for Stiff ODEs**: For stiff ODE problems, JF-SM Solver employs implicit LMMs, utilizing the Predictor-Corrector Algorithm. This approach ensures stability and accuracy in the numerical solutions of stiff equations, which are often challenging to solve using explicit methods.

- **Explicit LMM for Non-Stiff ODEs**: For non-stiff ODEs, the solver uses explicit LMMs, providing efficient and accurate solutions. This flexibility allows users to tackle a wide range of ODE problems within a single application.

5.2.3   Graphing Functionality

To aid in the interpretation and analysis of solutions, JF-SM Solver includes robust graphing functionalities. After solving an ODE, users can visualize the results through interactive graphs. This feature allows users to see the behavior of the solution over time, facilitating a deeper understanding of the underlying dynamics of the ODE.

5.3 Development and User Experience

Developed using Flutter and Dart, JF-SM Solver benefits from a modern, responsive user interface that works seamlessly across various desktop platforms. Flutter's rich set of widgets and Dart's efficient performance contribute to a smooth and intuitive user experience, making advanced numerical analysis accessible to both novice and experienced users.

5.4 Conclusion

JF-SM Solver represents a significant advancement in desktop applications for solving and analyzing ODEs using LMMs. By combining the power of implicit and explicit LMMs with the flexibility of the Predictor-Corrector Algorithm and enhanced graphing capabilities, JF-SM Solver provides a comprehensive tool for both academic and practical applications in numerical analysis. The use of Flutter and Dart ensures that the application is both user-friendly and highly functional, catering to the needs of a diverse user base.

Future enhancements may include expanding the range of numerical methods available, integrating additional visualization tools, and refining the user interface based on user feedback. JF-SM Solver sets a strong foundation for ongoing development and application in the field of numerical ODE solving.

5.5 Limitations of the Study

- **Stability Issues with Stiff Equations**: The JF-SM Solver's performance in solving stiff equations relies heavily on the stability properties of the Linear Multistep Methods (LMM) that are analyzed and used. In particular, the A-stability and L-stability of the LMMs are crucial for effectively managing stiff problems. If the chosen LMM does not possess these stability characteristics, the app may struggle to handle stiff equations properly, which could result in inaccuracies or failures in the solutions.

- **Predictor-Corrector Algorithm**: The predictor and corrector must have the same order, which can limit the flexibility of the solver in some cases.

5.6   Recommendations

Based on the findings and developments in this study, the following recommendations are proposed for future work:

- **Expanding Numerical Methods**: Incorporate additional numerical methods beyond LMMs to broaden the range of solvable ODE problems.

- **Enhanced Visualization Tools**: Develop more advanced visualization tools, such as 3D plotting and dynamic animations, to provide deeper insights into the behavior of solutions.

- **User Interface Improvements**: Continuously refine the user interface based on user feedback to enhance usability and functionality.

- **Performance Optimization**: Further optimize the performance of the application to handle even larger and more complex ODE problems efficiently.The implementation of variable step size control and adaptive methods can improve the efficiency of the solver.

- **Integration with Other Tools**: Explore the integration of JF-SM Solver with other mathematical and engineering software to create a more comprehensive toolkit for researchers and practitioners.

- **Educational Resources**: Develop educational resources and tutorials to help users, especially those new to numerical analysis, to effectively use JF-SM Solver.

- **Implementing a Step-Size Control System**: To enhance the app's capability in handling stiff equations, it is recommended to integrate an adaptive step-size control system. This system would dynamically adjust the time step based on the stiffness of the problem, ensuring stability and accuracy. By implementing step-size control, the app can better manage the challenges posed by stiff equations, improving overall performance and reliability.

By addressing these recommendations, future versions of JF-SM Solver can become even more powerful and versatile tools for the numerical analysis and solution of ODEs.

REFERENCES

Alexanderian, A. (2022). A brief note on linear multistep methods.
(Last revised: April 27, 2022)

Alexandru, A., Gurnari, L., Ionescu, C., & Rădulescu, V. (2017). Linear multistep methods. *Journal of Mathematical Sciences*, *185*(3), 282–290. doi: 10.1007/s10958-012-1179-3

Atkinson, K. E. (1989). *An introduction to numerical analysis* (2nd ed.). New York, NY, USA: John Wiley & Sons.

Burkardt, J. (2010, March). *Rkf45 - a fortran90 library which implements the runge-kutta-fehlberg ode solver.* `https://people.math.sc.edu/Burkardt/f_src/rkf45/rkf45.html`.

Butcher, J. (2009). General linear methods for ordinary differential equations. *Mathematics and Computers in Simulation*, *79*(6), 1834-1845. Retrieved from `https://www.sciencedirect.com/science/article/pii/S0378475407001462` (Applied and Computational MathematicsSelected Papers of the Sixth PanAmerican WorkshopJuly 23-28, 2006, Huatulco-Oaxaca, Mexico) doi: https://doi.org/10.1016/j.matcom.2007.02.006

Butcher, J. C. (2003). *Numerical methods for ordinary differential equations.* John Wiley.

Butler, J. S. (2022). *Numerical Analysis Book - Chapter 05 - IVP Consistent Convergence Stability.* Online. Retrieved from `https://john-s-butler-dit.github.io/NumericalAnalysisBook/Chapter%2005%20-%20IVP%20Consistent%20Convergence%20Stability/503_Stability.html`

Byrne, G. D., Hindmarsh, A. C., Jackson, K. R., & Brown, H. G. (1977). A comparison of two ode codes: Gear and episode. *Computers & Chemical Engineering*, *1*(2), 125–131. Retrieved from `https://www.sciencedirect.com/science/article/pii/0098135477800185` doi: 10.1016/0098-1354(77)80018-5

Curtiss, C. F., & Hirschfelder, J. O. (1952, Jan). Integration of stiff system. *Proc. Nat. Acad. Sci.*. Retrieved from `https://null.com`

Dattani, N. S. (2008). *Linear multistep numerical methods for ordinary differential equations.*

Deborah .I.A, J. F. (2021). Development of a generalized online computer program for mathematical analysis of linear multistep methods. *Anchor University Journal of Science and Technology*, *1*, 4.

Emmanuel, F. S. (2018, August). Determination of the order and the error constant of an implicit linear four-step method. *Department of Mathematics, Ekiti State University, Ado Ekiti, Nigeria*, *1*, 7.

Enright, W. H., Hull, T. E., & Lindberg, B. (1975). Comparing numerical methods for stiff systems of o.d.e:s. *BIT Numerical Mathematics*, *15*(1), 10–48. Retrieved from `https://doi.org/`

10.1007/BF01932994 doi: 10.1007/BF01932994

Faleichik, B. (2009, April). *Explicit implementation of collocation methods for stiff systems with complex spectrum 1.* Retrieved from `https://api.semanticscholar.org/CorpusID: 9177554`

Fatokun, J. O. (1992, August). Power series collocation methods for the initial value problems. *Unilorin*, *1*, 10.

Hairer, E., Nørsett, S. P., & Wanner, G. (1993). *Solving ordinary differential equations i: Nonstiff problems* (2nd ed.). Berlin: Springer Verlag.

HELM Green, D., Harrison, M., & Ward, J. (2008). Workbook 32: Numerical initial value problems.
(Accessed on April 16, 2024)

Jeon, Y., Bak, S., & Bu, S. (2019). Reinterpretation of multi-stage methods for stiff systems: A comprehensive review on current perspectives and recommendations. *Mathematics*, *7*(12). Retrieved from `https://www.mdpi.com/2227-7390/7/12/1158` doi: 10.3390/math7121158

J.O. Fatokun, J. L., S.I. Okoro. (2022). A class of power series collocation multistep methods with legendre interpolant for the integration of initial value problems. *Anchor University Journal of Science and Technology*, *143*, 1-156. (This document presents a new approach for the derivation of a class of collocation linear multistep methods for the integration of some initial value problems of ordinary differential equations.)

Keller, R., & Du, Q. (2020, Jan). Discovery of dynamics using linear multistep methods. *arXiv preprint arXiv:1912.12728v2*. Retrieved from `https://arxiv.org/abs/1912.12728v2`

Lambert, J. D. (1977). The initial value problem for ordinary differential equations. In D. Jacobs (Ed.), *The state of the art in numerical analysis* (pp. 451–501). New York: Academic Press.

Multistep methods — fundamentals of numerical computation. (n.d.). Retrieved from `https:// fncbook.github.io/fnc/ivp/multistep.html` (Accessed: 2023-09-03)

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (2007). *Numerical recipes: The art of scientific computing* (3rd ed.). Cambridge University Press.

Quresh, S., Ramos, H., Soomro, A., Akinfenwa, O. A., & Akanbi, M. A. (2024). Numerical integration of stiff problems using a new time-efficient hybrid block solver based on collocation and interpolation techniques. *Mathematics and Computers in Simulation*. Retrieved from `https://www.sciencedirect.com/science/article/pii/S0378475424000016` doi: https://doi.org/10.1016/j.matcom.2024.01.001

Shampine, L. F., & Reichelt, M. W. (1997). The matlab ode suite. *SIAM Journal on Scientific Computing*, *18*, 1–22.

S.L, R., & II, R. S. (1989). *Introduction to ordinary differential equations* (3rd ed.). John Wiley and sons,NY.

Stone, C. P., Alferman, A. T., & Niemeyer, K. E. (2017). Accelerating finite-rate chemical kinetics with coprocessors: Comparing vectorization methods on gpus, mics, and cpus. *Computational Science and Engineering, LLC*.

S̈uli, E., & Mayers, D. F. (2003, Jan). An introduction to numerical analysis. *Cambridge University Press, first edition*. Retrieved from `https://null.com`

Thohura, S., & Rahman, A. (2013). Numerical approach for solving stiff differential equations: A comparative study. *Global Journal of Science Frontier Research. Mathematics and Decision Sciences*, *13*(6), Version 1.0. (Type: Double Blind Peer Reviewed International Research

Journal)

Wikipedia contributors. (2023a). *Collocation method — Wikipedia, the free encyclopedia.* Retrieved from `https://en.wikipedia.org/w/index.php?title=Collocation_method&oldid=1166346639` ([Online; accessed 14-January-2024])

Wikipedia contributors. (2023b). *Linear multistep method — Wikipedia, the free encyclopedia.* Retrieved from `https://en.wikipedia.org/w/index.php?title=Linear_multistep_method&oldid=1182900519` ([Online; accessed 12-March-2024])

Wikiversity. (2019). *Numerical analysis/stability of multistep methods — wikiversity,.* Retrieved from `\url{https://en.wikiversity.org/w/index.php?title=Numerical_Analysis/stability_of_Multistep_methods&oldid=1982169}` ([Online; accessed 3-March-2019])

Wong, J. (2020, April 12). *Math 563 lecture notes: Numerical methods for boundary value problems.* Lecture Notes.

Yatim, S. A. M., Ibrahim, Z. B., Othman, K. I., & Suleiman, M. B. (2013). A numerical algorithm for solving stiff ordinary differential equations. *Mathematical Problems in Engineering*, *2013*, Article ID 989381, 11. Retrieved from `https://doi.org/10.1155/2013/989381` doi: 10.1155/2013/989381