A STUDY OF EULER'S METHOD FOR SOLVING FIRST ORDER INITIAL VALUE PROBLEMS USING MATLAB

BY

CONFIDENCE IRUODUNOGHENA ONUMABOR
PSC1809114

UNIVERSITY OF BENIN
BENIN CITY

SEPTEMBER, 2023.

A STUDY OF EULER'S METHOD FOR SOLVING FIRST ORDER INITIAL VALUE PROBLEMS USING MATLAB

BY

CONFIDENCE IRUODUNOGHENA ONUMABOR

PSC1809114

DEPARTMENT OF MATHEMATICS,

FACULTY OF PHYSICAL SCIENCES,

UNIVERSITY OF BENIN, BENIN CITY.

SEPTEMBER, 2023.

A STUDY OF EULER'S METHOD FOR SOLVING FIRST ORDER INITIAL VALUE PROBLEMS USING MATLAB

BY

CONFIDENCE IRUODUNOGHENA ONUMABOR

PSC1809114

A PROJECT WORK SUBMITTED TO THE DEPARTMENT OF MATHEMATICS, FACULTY OF PHYSICAL SCIENCES, UNIVERSITY OF BENIN, IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE AWARD OF THE DEGREE OF BACHELOR OF SCIENCE (B.Sc.) HONOURS IN INDUSTRIAL MATHEMATICS.

SEPTEMBER, 2023.

UNDERTAKING

This project was carried out by me, **CONFIDENCE IRUODUNOGHENA ONUMABOR** with the **MATRICULATION NUMBER PSC1809114.** I have neither copied nor duplicated the work of any author(s). All work used have been duly cited and acknowledged.

---------- ----------

Name of Student Signature/Date

CERTIFICATION

This is to certify that this work was carried out by **CONFIDENCE IRUODUNOGHENA ONUMABOR PSC1809114** of the Department of Mathematics, Faculty of Physical Science, University of Benin, Benin City.

_____	_____
Dr. (Mrs.) G.C. Nwachukwu	Prof. R.I. Okuonghae
(Project Supervisor)	(Head of Department)
Date_____	Date_____

DEDICATION

This work is dedicated to God Almighty for His steadfast love and endless mercies that saw me through my studies.

ACKNOWLEDGEMENT

I begin with heartfelt thanks to God Almighty for His unwavering guidance throughout this project, for seeing me through challenges and filling me with inspiration.

I deeply appreciate my project supervisor, Dr. (Mrs.) G.C. Nwachukwu, for her vital role in this journey. Her guidance, encouragement, advice, and prayers were instrumental to the successful completion of this project work. I am also thankful to my lecturers for the valuable knowledge they imparted.

I extend my gratitude to my dad, Mr. Johnbull Onumabor, for his constant care, support, and provision. My siblings, Fortune, Treasure, and Goodness, have been unwavering in their support.

I also want to thank my friends: Sammy, Kennedy (Erhama), Jonathan (Joe boy) for everything they do for me. They are friends indeed.

Special thanks to Lucky Nweke, a big brother, and my senior colleague, Samuel Charles, for their invaluable support in the final lap of my undergraduate journey.

TABLE OF CONTENT

Title page ---------- i

Undertaking ---------- ii

Certification ---------- iii

Dedication ---------- iv

Acknowledgement ---------- v

Table of Content ---------- vi

Abstract ---------- ix

CHAPTER ONE: INTRODUCTION

1.1 Introduction ---------- 1

1.2 Background of the study ---------- 2

1.3 Statement of the study ---------- 3

1.4 Aims and Objectives of the study ---------- 3

1.5 Significance of the study ---------- 4

1.6 Scope of the study ---------- 4

CHAPTER TWO: LITERATURE REVIEW

2.1 Introduction to Differential Equations ---------- 5

2.2 Introduction to Numerical Methods for Solving Differential Equations ---------- 7

2.3 Numerical Methods for Solving First Order Initial Value Problems (IVPs) ---------- 8

2.3.1 Initial Value Problem ---------- 8

2.3.2 The Elementary Theory of Initial Value Problems ---------- 9

2.3.3 Overview of Numerical Methods for Solving First Order Initial Value Problems ---------- 10

2.4 Euler's Method ---------- 13

2.4.1 The Theoretical Framework of Euler's Method ---------- 14

2.5 Advantages and Limitations of Euler's Method ---------- 17

CHAPTER THREE: METHODOLOGY

3.1 Introduction ---------- 21

3.2 Overview of MATLAB as a Computational Tool ---------- 21

3.2.1 MATLAB Features for Solving Differential Equations ------- 26

3.3 Implementation of Euler's Method in MATLAB ---------- 28

3.3.1 Euler's Method Algorithm ---------- 29

3.4 Manual Computation of Euler's Method vs. MATLAB Implementation ---------- 34

3.4.1 Manual Computation of Euler's Method ---------- 36

3.4.2 MATLAB Implementation ---------- 40

3.5 Conclusion ---------- 46

CHAPTER FOUR: RESULTS AND FINDINGS

4.1 Comparative Analysis: Manual Computation vs. MATLAB Implementation ---------- 47

4.1.1 Challenges and Limitation of Manual Computation ---------- 47

4.1.2 Advantages of MATLAB Implementation ---------- 49

4.2 Computational Efficiency of Euler's Method Implementation in MATLAB ---------- 49

4.3 Conclusion ---------- 53

CHAPTER FIVE: CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion ---------- 54

5.2 Recommendations ---------- 54

References ---------- 56

ABSTRACT

This project work deals with the study of Euler's method for solving initial value problems using MATLAB. It sheds light on the challenges commonly encountered when manually implementing numerical methods for solving differential equations. It also emphasizes MATLAB's advantages, such as time savings, improved speed, and enhanced accuracy, over traditional manual computations.

CHAPTER ONE

1.1 INTRODUCTION

Differential equations are a fundamental part of mathematics used to describe real-life problems. They are equations that relate one or more unknown functions and their derivatives (Zill, 2012). According to Wikipedia, in applications, the functions generally represent physical quantities, the derivatives represent their rates of change, and the differential equation defines a relationship between the two. Such relations are common; therefore, differential equations play a prominent role in many disciplines including engineering, physics, economics, and biology.

The study of differential equations consists mainly of the study of their solutions and the properties of their solution. The solution of differential equations involves two main approaches: analytical methods and numerical methods. Analytical methods aim at finding exact, closed-form solutions using techniques such as separation of variables, integrating factors, etc. While analytical methods are generally straightforward and provide exact, closed-form solutions, they are often limited to simple differential equations. Many differential equations, especially real-world problems, cannot be solved exactly, making analytical methods impractical. It is within this realm of complexity that numerical methods are often preferred because they provide approximate solutions that can

handle a wider range of differential equations without the need for closed-form expressions.

1.2 BACKGROUND OF THE STUDY

Throughout history, mathematicians and scientists have grappled with solving differential equations. In the past, they had to rely heavily on symbolic manipulation, pen-and-paper techniques, and mathematical analysis to craft solutions to these equations. However, the limitations of these methods for tackling differential equations soon became evident. The development of numerical methods marked a significant breakthrough in solving differential equations. Numerical methods have a long history dating back to the 18th century. Pioneers like Leonhard Euler laid the foundation for subsequent developments in numerical methods with his innovation of Euler's method, which is named after him.

Like in the analytical ways of solving differential equations, mathematicians also encountered several challenges in the earliest days of the numerical methods when attempting to solve complex problems manually. Manual computations were extremely laborious, time-consuming and prone to a lot of errors, as mathematicians had to perform lengthy calculations by hand. Despite these challenges, early pioneers in numerical methods persevered and made significant contributions that laid the groundwork for modern numerical analysis.

With the advent of digital computers in the 20th century, numericals methods truly flourished and continue to evolve rapidly today, revolutionising the way we solve differential equations.

1.3 STATEMENT OF THE STUDY

Even though numerical methods were a significant breakthrough in solving differential equations, manually implementing these methods can prove to be challenging. This study was instigated by the need for a more efficient and reliable approach to implementing numerical methods for solving differential equations. Hence, the use of MATLAB, a powerful computational tool, to automate and simplify the process.

1.4 AIM AND OBJECTIVES OF THE STUDY

Aim:

The primary aim of this study is to automate the process of solving differential equations using MATLAB.

Objectives:

To accomplish this aim, the study is guided by the following objectives:

1. To develop and implement a MATLAB program for automating the process of solving differential equations.
2. To assess and compare the accuracy, efficiency, and computational performance of MATLAB in solving differential equations with

manual calculations, highlighting the benefits of computational software for numerical problem-solving.

1.5 SIGNIFICANCE OF THE STUDY

This study holds significance as it streamlines the process of solving differential equations numerically, reducing errors and enhancing efficiency through MATLAB automation. The study will serve as a valuable learning resource for students and researchers seeking to learn or improve their numerical analysis skills, as it offers insights into the use of MATLAB for solving differential equations.

1.6 SCOPE OF THE STUDY

Numerical methods for solving differential equations are really wide. So as to not lose sight of the true motivation for this work, the project study focuses on MATLAB's implementation of Euler's method for solving first order initial value problems (IVPs) in ordinary differential equations. The study will cover discussions on first order IVPs, the application of Euler's method for solving IVPs, the practical implementation of Euler's method using MATLAB, and an assessment of the advantages gained by utilizing MATLAB for numerical computations over hand calculations.

CHAPTER TWO

LITERATURE REVIEW

2.1 INTRODUCTION TO DIFFERENTIAL EQUATIONS

Differential equations are mathematical equations that relate a function to its derivatives, providing a fundamental framework for expressing relationships between quantities that change continuously over time or space (Stewart, 2008). They involve one or more derivatives of an unknown function with respect to one or more independent variables. Describing the relationship between the unknown function and its derivatives, a differential equation is commonly used to model various physical, engineering, and scientific phenomena.

The order of a differential equation is determined by the highest derivative that appears in the equation. Consider the following differential equations:

$$\frac{dy}{dx} = y - x \qquad (2.1)$$

$$\frac{d^2y}{dx^2} + 5\frac{dy}{dx} + 6y = 0 \qquad (2.2)$$

$$\frac{d^ny}{dx^n} + a_{n-1}\frac{d^{n-1}y}{dx^{n-1}} + \ldots + a_1\frac{dy}{dx} + a_0y = f(x) \qquad (2.3)$$

Equations (2.1) and (2.2) are differential equations of first and second order respectively. Equation (2.3) is an example of a higher-order differential equation; It is the general form of the *n*th order linear ordinary

differential equation. Higher-order differential equations are commonly encountered in various scientific and engineering fields.

Differential equations can be classified into different types based on their characteristics:

1. Ordinary Differential Equations (ODEs):

Ordinary differential equations are mathematical equations involving one or more derivatives of an unknown function with respect to a single independent variable. They express relationships between the unknown function and its derivatives, typically representing how the function changes in response to variations in the independent variable (Stroud, 2001). Equations (2.1), (2.2), and (2.3) are typical examples of ODEs.

2. Partial Differential Equations (PDEs):

Partial differential equations are mathematical equations that involve partial derivatives of an unknown function with respect to multiple independent variables. They describe the behaviour of the unknown function in terms of its partial rates of change with respect to each independent variable, capturing intricate relationships between various variables (Greenberg, 1971).

For example, if $w = f(x, y, z, t)$ is a function of time and the three rectangular coordinates of a point in space, then the following are partial differential equations of the second order:

$$\frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} + \frac{\partial^2 w}{\partial z^2} = 0; \qquad (2.4)$$

$$a^2\left(\frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} + \frac{\partial^2 w}{\partial z^2}\right) = \frac{\partial w}{\partial t}; \qquad (2.5)$$

$$a^2\left(\frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} + \frac{\partial^2 w}{\partial z^2}\right) = \frac{\partial^2 w}{\partial t^2}; \qquad (2.6)$$

These equations, (2.4), (2.5) and (2.6) are called Laplace's equation, the heat equation, and the wave equation, respectively. Each is profoundly significant in theoretical physics, and their study has stimulated the development of many important mathematical ideas. PDEs are essential in describing complex physical systems, such as electric fields, heat conduction, wave propagation, and fluid dynamics. Their theory is different from that of ODE, and is more difficult in almost every respect (Simmons, 1991).

2.2 INTRODUCTION TO NUMERICAL METHODS FOR SOLVING DIFFERENTIAL EQUATIONS

In real-life situations, many complex systems cannot be solved analytically due to their complexity. This is where numerical methods become essential tools. Numerical methods enable us to address complex differential equations that lack exact solutions. By breaking down the equations into smaller parts and employing computational algorithms,

numerical methods offer approximations that help us understand and make predictions about real-life occurrences.

There are several numerical methods used to solve differential equations. Some commonly employed methods include the Euler Method, Runge-Kutta Methods, Adams-Bashforth and Adams-Moulton Methods, etc. Each method has its own strengths and weaknesses. The choice of method depends on the specific characteristics of the differential equation, as well as the desired level of accuracy and computational efficiency.

2.3 NUMERICAL METHODS FOR SOLVING FIRST ORDER INITIAL VALUE PROBLEMS (IVPs)

Before exploring the numerical methods for solving initial value problems, we begin this section of the literature review by establishing the concept of initial value problems in differential equations and the elementary theory of initial value problems.

2.3.1 INITIAL VALUE PROBLEM

Differential equations are used to model problems in science and engineering that involve the change of some variable with respect to another. Most of these problems require the solution of an initial value problem, that is, the solution to a differential equation that satisfies a given initial condition (Burden and Faires, 2011).

Simply put, an initial value problem is a differential equation that requires determining the solution of an unknown function and its associated derivatives with respect to an independent variable, at a specific initial point.

2.3.2 THE ELEMENTARY THEORY OF INITIAL VALUE PROBLEMS

(Matthews and Fink, 1999)

Definition 2.1

A solution to the initial value problem:

$$y' = f(x, y), \qquad y(x_0) = y_0 \qquad (2.7)$$

on an interval $[x_0, b]$ is a differentiable function $y = y(x)$ such that

$$y(x_0) = y_0 \quad \text{and} \quad y'(x) = f(x, y(x)) \quad \text{for all} \quad x \in [x_0, b] \qquad (2.8)$$

Definition 2.2

Given the rectangle $R = \{(x, y): a \leq x \leq b, c \leq y \leq d\}$, assume that $f(x, y)$ is continuous on R. The function f is said to satisfy a **Lipschitz condition** in the variable y on R provided that a constant $L > 0$ exists with the property that

$$|f(x, y_1) - f(x, y_2)| \leq L|y_1 - y_2| \qquad (2.9)$$

whenever $(x, y_1), (x, y_2) \in R$.

The constant L is called a Lipschitz constant for f.

Theorem 2.1

Suppose that $f(x, y)$ is defined on the region R. If there exists a constant $L > 0$ so that

$$|f_y(x, y)| \leq L \quad \text{for all } (x, y) \in R, \qquad (2.10)$$

then f satisfies a Lipschitz condition in the variable y with Lipschitz constant L over the rectangle R.

Theorem 2.2 (Existence and Uniqueness)

Assume that $f(x, y)$ is continuous in a region $R = \{(x, y) : x_0 \leq x \leq b \text{ and } c \leq y \leq d\}$. If f satisfies a Lipschitz condition on R in the variable y and $(x_0, y_0) \in R$, then the initial value problem (2.7) has a unique solution $y = y(x)$ on some subinterval $x_0 \leq x \leq x_0 + \delta$.

2.3.3 OVERVIEW OF NUMERICAL METHODS FOR SOLVING FIRST ORDER INITIAL VALUE PROBLEMS

Numerical methods for solving initial value problems can be classified into two categories: One-step methods and Multi-step methods.

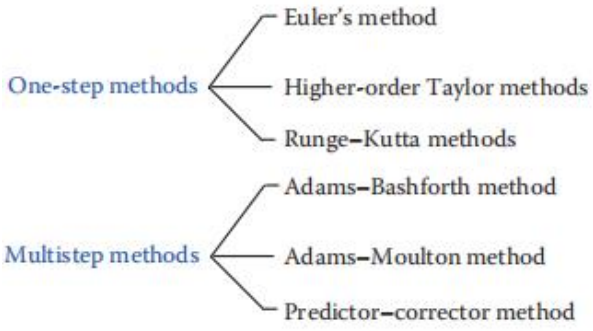


Figure 2.1 Classification of numerical methods to solve an initial value problem (Esfandiari, 2017).

One-step methods efficiently solve first order initial value problems by directly computing the next solution value from the current one in a single computational step, making them suitable for simple problems or memory-limited scenarios. As seen in **Figure 2.1**, some common one-step methods for solving IVPs include the Euler method, High-order Taylor methods, and Runge-Kutta methods.

Mathematically, a one-step method can be expressed as:

$$y_{n+1} = y_n + hf(x_n, y_n) \qquad (2.11)$$

where,

- x_n is the value of the independent variable at the *n*th step.
- y_n is the numerical approximation of the solution at the current step.
- y_{n+1} is the numerical approximation of the solution at the next step.
- h is the step size, representing the interval between successive steps.
- $f(x_n, y_n)$ is the derivative of the function y with respect to x evaluated at x_n and y_n.

In one-step methods, the solution estimate y_{n+1} at x_{n+1} is obtained by using information at a single previous point x_n. Multi-step methods are based on the idea that a more accurate estimate for y_{n+1} at x_{n+1} can be

attained by utilizing information on two or more previous points (or solution values) rather than one.

Mathematically, a general multi-step method can be represented as:

$$y_{n+k} = y_{n+k-1} + h\sum_{j=0}^{k} \alpha_j f(x_{n+j}, y_{n+j}) \tag{2.12}$$

where,

- h is the step size,
- n represents the current step,
- k is the number of previous steps considered,
- y_{n+k} is the approximation of the solution at step $n + k$,
- $f(x_{n+j}, y_{n+j})$ represents the derivative of the function y with respect to x evaluated at the point (x_{n+j}, y_{n+j}),
- The coefficients α_j are determined by the specific method being used.

Figure 2.1 above shows some of the different multi-step methods for solving first order IVPs. It is important to note that these multi-step methods have varying coefficients α_j, which are designed to optimize the trade-off between accuracy and stability. The general form of the equation remains the same, but the specific values of α_j depend on the order and characteristics of the multi-step method to be used.

While various numerical methods, including Runge-Kutta and multi-step methods, contribute significantly to solving IVPs, this project focuses specifically on exploring Euler's Method as a fundamental numerical method for solving first order IVPs.

Focusing on Euler's method not only provides a clear path to understanding the fundamentals of numerical solutions but also makes the implementation in MATLAB more accessible. By concentrating on this method, we also lay the groundwork for an indispensable foundational understanding of MATLAB's functionalities and computational capabilities that can be applied to more advanced numerical methods for solving first order initial value problems.

Now, let's delve into Euler's Method, exploring its implementation, advantages, limitations, and the theoretical concepts associated with it.

2.4 EULER'S METHOD

Euler's Method, pioneered by the Swiss mathematician Leonhard Euler in the 18th century, marked a significant breakthrough in solving differential equations numerically. It laid the groundwork for modern numerical analysis by introducing a simple yet effective approach to approximating solutions to initial value problems.

This innovation subsequently inspired the emergence of the Runge-Kutta methods and other advanced numerical methods. These methods built

upon Euler's approach by incorporating multiple steps to enhance accuracy and stability. By leveraging Euler's foundational work, these methods became indispensable tools in diverse fields, delivering enhanced precision in approximating solutions to differential equations.

2.4.1 THE THEORETICAL FRAMEWORK OF EULER'S METHOD (Matthews and Fink, 1999)

Let [*a, b*] be the interval over which we want to find the solution to the well-posed IVP $y' = f(x, y)$, with $y(a) = y_0$. In actuality, we will not find a differentiable function that satisfies the IVP. Instead, a set of points $\{(x_n, y_n)\}$ is generated, and the points are used for an approximation (i.e., $y(x_n) \approx y_n$).

First, we choose the abscissas for the points. For convenience, we subdivide the interval [*a, b*] into *N* equal subintervals and select the mesh points

$$x_n = a + nh \quad \text{for } n = 0, 1, \ldots, N \qquad (2.13)$$

Where, $h = \dfrac{b - a}{N}$

The value *h* is called the ***step size***.

We now proceed to solve approximately

$$y' = f(x, y) \quad \text{over} \quad [x_0, x_N] \quad \text{with} \quad y(x_0) = y_0 \qquad (2.14)$$

Assume that $y(x)$, $y'(x)$, and $y''(x)$ are continuous and use Taylor's theorem to expand $y(x)$ about $x = x_0$. For each value x, there exists a value c_1 that lies between x_0 and x so that

$$y(x) = y(x_0) + y'(x_0)(x - x_0) + \frac{y''(c_1)(x - x_0)^2}{2} \qquad (2.15)$$

When $y'(x_0) = f(x_0, y(x_0))$ and $h = x_1 - x_0$ are substituted in equation (2.15), the result is an expression for $y(x_1)$:

$$y(x_1) = y(x_0) + hf(x_0, y(x_0)) + y''(c_1)\frac{h^2}{2} \qquad (2.16)$$

If the step size h is chosen small enough, then we may neglect the second-order term (involving h^2) and get

$$y_1 = y_0 + hf(x_0, y_0), \qquad (2.17)$$

which is **Euler's approximation**.

The process is repeated and generates a sequence of points that approximates the solution curve $y = y(x)$.

The general step for Euler's method is

$$y_{n+1} = y_n + hf(x_n, y_n), \qquad x_{n+1} = x_n + h \qquad (2.18)$$

for $n = 0, 1, \ldots, N - 1$.

Step Size versus Error

When using any numerical method to approximately solve an initial value problem, there are two main sources of error: truncation and round off.

Definition 2.3 (Truncation Error)

Assume that $\{(x_n, y_n)\}_{n=0}^{N}$ is the set of discrete approximations and that $y = y(x)$ is the unique solution to the initial value problem.

The ***global error*** e_n is defined by

$$e_n = y(x_n) - y_n \quad \text{for } n = 0, 1, \ldots, N - 1 \qquad (2.19)$$

It is the difference between the unique solution and the numerical solution.

The ***local truncation error*** t_{n+1} is defined by

$$t_{n+1} = y(x_{n+1}) - y_n - hf(x_n, y_n) \quad \text{for } n = 0, 1, \ldots, N - 1 \qquad (2.20)$$

It is error committed in the single step from x_n to x_{n+1}.

When we obtained equation (2.18) for Euler's method, the neglected term for each step was $y^{(2)}(c_n)(h^2/2)$. If this was the only error at each step, then at the end of the interval $[a, b]$, after N steps have been made, the accumulated error would be

$$\sum_{n=1}^{N} y^{(2)}(c_n)\frac{h^2}{2} \approx Ny^{(2)}(c)\frac{h^2}{2} = \frac{hN}{2}y^{(2)}(c)h = \frac{(b-a)y^{(2)}(c)}{2}h = O(h^1)$$

There could be more error but this estimate predominates.

Theorem 2.3 (Precision of Euler's Method)

Assume that $y(x)$ is the solution to the IVP given in (2.14). If $y(x) \in C^2[x_0, b]$ and $\{(x_n, y_n)\}_{n=0}^{N}$ is the sequence of approximations generated Euler's method, then

$$|e_n| = |y(x_n) - y_n| = O(h),$$
$$|t_{n+1}| = |y(x_{n+1}) - y_n - hf(x_n, y_n)| = O(h^2) \qquad (2.21)$$

The error at the end of the interval is called the ***final global error (F.G.E.)***

$$E\,(y(b), h) = |y(b) - y_N| = O(h) \qquad (2.22)$$

Remark: The final global error $E\,(y(b), h)$ is used to study the behaviour of the error for various step sizes. It can be used to give us an idea of how much computing effort must be done to obtain an accurate approximation.

2.5 ADVANTAGES AND LIMITATIONS OF EULER'S METHOD

To uncover the advantages and limitations of Euler's Method, this section of the literature review will examine a real-life scenario through a specific initial value problem. By scrutinizing how Euler's Method performs in this context, we gain valuable insights into scenarios where it might be a suitable choice despite its limitations. This exploration will provide a clearer picture of the method's practical implications.

Example 2.1 (Matthews and Fink, 1999)

Use Euler's method to solve approximately the initial value problem

$y' = Ry$ over [0, 1] with $y(0) = y_0$ and R constant (2.23)

The step size must be chosen, and the first formula in (2.18) can be determined for computing the ordinates. This formula is sometimes called a difference equation, and in this case it is

$$y_{n+1} = y_n + hRy_n \quad \text{for } n = 0, 1, \ldots, N - 1 \qquad (2.24)$$

$$y_{n+1} = y_n(1 + hR) \qquad (2.25)$$

If we trace the solution values recursively, we see that

$$\begin{aligned} y_1 &= y_0(1 + hR) \\ y_2 &= y_1(1 + hR) = y_0(1 + hR)^2 \\ &\vdots \\ y_N &= y_{N-1}(1 + hR) = y_0(1 + hR)^N \end{aligned} \qquad (2.26).$$

For most problems, there is no explicit formula for determining the solution points, and each new point must be computed successively from the previous point. However, for the IVP (2.23), we are fortunate; Euler's method has the explicit solution

$$y_n = y_0(1 + hR)^n, \qquad x_n = nh \quad \text{for } n = 0, 1, \ldots, N \qquad (2.27)$$

Formula (2.27) can be viewed as the "compound interest", and the Euler approximation gives the future value of a deposit.

Example 2.2: Suppose that ₦1000 is deposited and earns 10% interest compounded continuously over 5 years. What is the value at the end of 5 years?

We choose the Euler approximations with $h = 1, \frac{1}{12}, \frac{1}{360}$ to approximate $y(5)$ for the IVP

$$y' = 0.1y \quad \text{over } [0, 5] \text{ with } y(0) = 1000$$

Formula (2.27) with $R = 0.1$ produces the table below:

Step size, h	Number of iterations, N	Approximations to $y(5)$, y_N
1	5	$1000(1 + 0.1)^5 = 1610.51$
$\frac{1}{12}$	60	$1000\left(1 + \frac{0.1}{12}\right)^{60} = 1645.31$
$\frac{1}{360}$	1800	$1000\left(1 + \frac{0.1}{360}\right)^{1800} = 1648.61$

Table 2.1 Compound Interest in Example 2.2

Think about the different values y_5, y_{60}, and y_{1800} that are used to determine the future values after 5 years. These values are obtained using different step sizes and reflect different amounts of computing effort to obtain an approximation to $y(5)$.

The exact solution to the IVP is $y(5) = 1000e^{0.5} = 1648.72$

If we did not use the closed-form solution (2.27), then it would have required 1800 manual iterations of Euler's method to obtain y_{1800}, and we still have only five digits of accuracy in the answer!

This is where MATLAB becomes an indispensable computational tool. Its computational power allows us to automate and optimize this iterative process, making it efficient and accurate even for complex problems.

CHAPTER THREE

METHODOLOGY

3.1 INTRODUCTION

As demonstrated in the previous chapters, the motivation for this study arises from the need for a more efficient and reliable approach to implementing numerical methods for solving differential equations. Given the focus on one of the earliest developed numerical methods, this chapter showcases the practical implementation of this method using MATLAB. Additionally, it provides an overview of MATLAB's features that make it an ideal computational tool for tackling mathematical problems, particularly differential equations.

3.2 OVERVIEW OF MATLAB AS A COMPUTATIONAL TOOL

(Chapra and Raymonds, 2015)

MATLAB developed by The MathWorks, Inc., initially as a *matrix laboratory*, remains rooted in matrix computations. This interactive environment excels at mathematical operations involving matrices, making it convenient and user-friendly. Over the years, MATLAB has evolved to encompass various numerical functions, symbolic computations, and powerful visualization tools. These features collectively establish MATLAB as a comprehensive technical computing

environment, well-suited for implementing numerous numerical methods, especially for solving differential equations.

Moreover, MATLAB allows the creation of M-files, which are commonly used for numerical calculations. Here's how you can create one:

Starting MATLAB:

To launch MATLAB in a PC environment (assuming it's properly installed and licensed), locate the MATLAB icon on your desktop or in your Start Menu and double-click it. MATLAB typically opens with a "MATLAB Start" window offering various options and resources, including the ability to create new scripts, access MATLAB's extensive library, and explore its features.

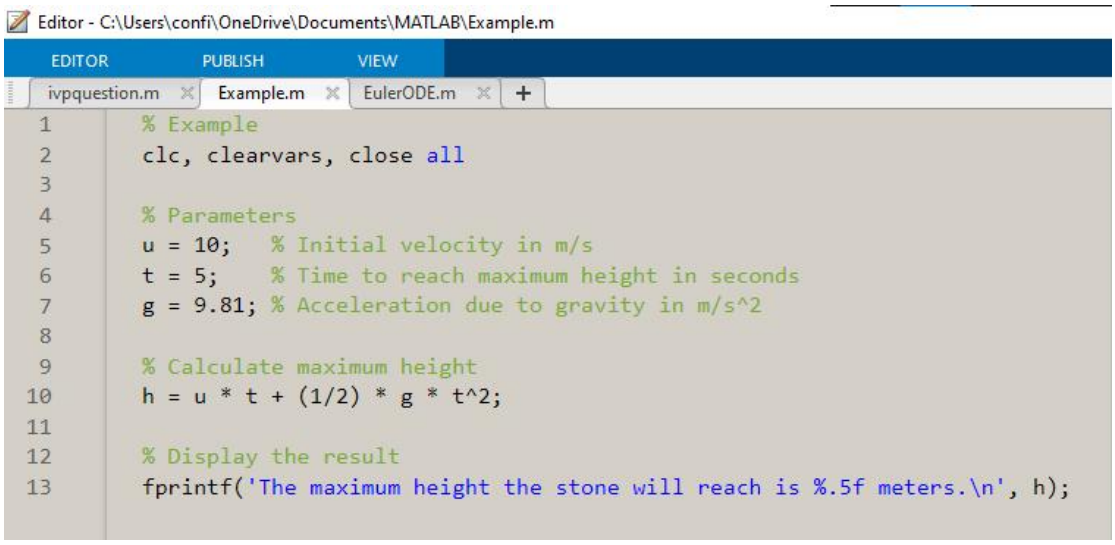
Example 3.1

In this simple example, we would recognize that MATLAB's utility is closely tied to programming.

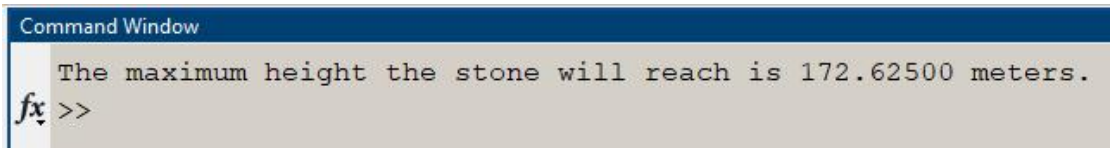
Imagine you are standing in the open field at the Faculty of Physical Science in the University of Benin, and you decide to throw a ball vertically upward from the ground with an initial velocity of 10 m/s. Your goal is to determine the maximum height the ball will reach before falling back to the ground, given that it takes 5 seconds to reach its maximum

height. Take the acceleration due to gravity, '*g*', to be approximately 9.81 m/s^2.

The MATLAB code presented in Program 3.1 analytically calculates the maximum height using the parameters from the example given.



Program 3.1: *Example.m*



Display 3.1: *Command window display of the result of Program 3.1*

In this example, we break down each part of the code in Program 3.1 above and explain their functionality, thus providing insights into the high-level programming language and built-in functions in MATLAB.

Creating an M-File:

Start by creating a MATLAB document called an M-file. To do this, navigate to the menu and select

File > New Script.

A new window, titled "MATLAB Editor", will open allowing you to type and edit MATLAB programs. In the MATLAB code in Program 3.1 above, the program is saved with the name "**Example**". MATLAB automatically attaches the "**.m**" extension to denote it as an M-file, resulting in "**Example.m**".

Code Explanation:

- `% Example`: This line includes the comment (%) to indicate that the subsequent lines are part of an example. Comments in MATLAB are text that are not executed as code; they provide explanations.
- `clc`: This command clears the command window displayed below the MATLAB Editor, providing a clean workspace for the upcoming calculations.
- The `clearvars` command clears all variables from the MATLAB workspace, ensuring that no existing variables might interfere with the calculations.
- The `close all` command closes all open figure windows, ensuring a clean slate for displaying any new figures.

- `% Parameters`: This comment indicates that the following lines 5 to 7 define the parameters for the problem.
- `u = 10;`: This line assigns the initial velocity `u` a value of 10 m/s. It is the speed at which the object is thrown upward.
- `t = 5;`: This line assigns the time `t` a value of 5 seconds. This is the time taken for the object to reach its maximum height.
- `g = 9.81;`: This line assigns the acceleration due to gravity `g` a value of 9.81 m/s^2. This value is approximately equal to the acceleration due to gravity on Earth.
- `% Calculate maximum height`: This comment indicates that line 10 calculates the maximum height.
- `h = u * t + (1/2) * g * t^2;`: This line performs the calculation for maximum height `h`. It uses Newton's second equation for motion which calculates the maximum height reached by an object thrown upward.
- `% Display the result`: This is a comment indicating that the following line displays the result of the calculation.
- `fprintf('The maximum height the stone will reach is %.5f meters.\n', h);`: This line uses the `fprintf` function to print a formatted message to the command window. It displays the calculated maximum height `h` with a precision of 5

decimal places. The message is enclosed in single quotes and includes a placeholder `%.5f` to insert the value of `h`. The `\n` character is used to create a new line after displaying the result as seen in Display 3.1.

- In the code, note the semicolon (;) at the end of some lines, which suppresses the output display for those specific lines when executed. This prevents MATLAB from displaying intermediate results.

Thus, when you run the MATLAB code in Program 3.1, it will clear the command window, set the parameters for the problem, calculate the maximum height using the provided formula, and then display the result in the command window.

3.2.1 MATLAB FEATURES FOR SOLVING DIFFERENTIAL EQUATIONS (MathWorks Inc., 2023)

MATLAB, developed by MathWorks Inc., is renowned for its exceptional capabilities in tackling a wide array of mathematical and computational challenges. Within the context of differential equations, MATLAB stands as a powerful ally, offering a suite of features and functions tailored to facilitate the numerical solution of these equations. These include:

1. **Extensive ODE Solvers**: One of the most notable strengths of MATLAB is its arsenal of Ordinary Differential Equation (ODE) solvers. MATLAB provides an array of ODE solvers catering to various types of differential equations, from initial value problems to boundary value problems. These solvers are rigorously tested and optimized, ensuring high accuracy and efficiency in solving complex differential equations.
2. **Robust Documentation**: MATLAB's documentation is a treasure trove of resources for both beginners and experienced users. It offers detailed explanations, examples, and reference materials for all functions, including those relevant to differential equations. This comprehensive documentation simplifies the learning curve and serves as a reliable reference when implementing numerical methods for solving differential equations.
3. **Visualization Tools and Interactive Environment**: MATLAB excels in data visualization and graphical representation. Its visualization tools empower users to create insightful plots and graphs, enabling a deeper understanding of the behaviour of solutions to differential equations. The interactive environment of MATLAB allows for real-time adjustments and exploration, facilitating experimentation with different parameters and initial conditions.

4. **Thriving Community and Support**: MATLAB benefits from a robust user community and extensive support resources. Users have access to forums, tutorials, and expert guidance, creating a collaborative environment where challenges can be overcome and methodologies optimized. This strong community and support network are invaluable when seeking solutions to complex problems involving differential equations.

These features collectively make MATLAB a versatile and powerful tool for tackling a wide range of differential equation problems, from simple initial value problems to complex systems of partial differential equations.

Now, let us delve into the practical implementation of Euler's method for solving initial value problems in MATLAB.

3.3 IMPLEMENTATION OF EULER'S METHOD IN MATLAB

As seen in the literature review in Chapter Two, Euler's Method stands as a fundamental numerical technique for solving first order initial value problems in differential equations. It operates on a straightforward principle that approximates the solution by iteratively stepping forward from the initial point. The key idea is to divide the interval over which we seek a solution into small steps, and at each step, estimate the change in the function's value based on its derivative.

3.3.1 EULER'S METHOD ALGORITHM

In this subsection, we will describe the step-by-step process of implementing Euler's Method in MATLAB, along with code snippets illustrating the algorithmic steps.

The user-defined function `EulerODE` given in Program 3.2 below, employs Euler's method to approximate the solution of the initial value problem (Equation 2.14) by computing (Equation 2.18)

for $n = 0, 1, \ldots, N - 1$.

Editor - C:\Users\confi\OneDrive\Documents\MATLAB\EulerODE.m

EDITOR PUBLISH VIEW

ivpquestion.m | Example.m | EulerODE.m

```
function y = EulerODE(f, x, y0)
    %
    % EulerODE uses Euler's method to solve a first-order initial value
    % problem in the form y' = f(x, y), y(x0) = y0.
    %
    %  y = EulerODE(f, x, y0), where
    %
    %   Inputs:
    %       f is an anonymous function representing f(x,y),
    %       x is a vector representing the mesh points,
    %       y0 is a scalar representing the initial value of y,
    %
    %   Output:
    %       y is the vector of solution estimates at the mesh points.
    %

    % Pre-allocate
    y = 0*x;
    y(1) = y0;
    N = length(x);

    % Perform Euler's Method
    for n = 1:N-1
        h = x(n+1) - x(n);   %step size
        y(n+1) = y(n) + h*f(x(n), y(n));
    end

end
```

Program 3.2: User-defined function `EulerODE`

The step-by-step process of implementing Euler's method in MATLAB using the provided `EulerODE` function in Program 3.2 is as follows:

1. **Define the ODE Function**:

Start by defining the ordinary differential equation (ODE) function, **`f(x, y)`**, that represents the derivative of **`y`** with respect to **`x`**. This function should be defined using MATLAB's anonymous function syntax, for example:

```
f = @(x, y) (x + y);
```

Here, **`f`** represents the ODE **`dy/dx = x + y`**.

2. **Specify the Range of x-values**:

- Define the initial point **`x0`**.
- Define the final point **`xN`**.
- Specify the step size **`h`** between consecutive points.

For example:

```
x0 = 0;     % Initial x-value
xN = 1;     % Final x-value
h = 0.1;   % Step size
```

3. **Create the Vector `x`**:

Use a loop to generate the vector `x` with the specified step size. Starting from `x0`, keep adding `h` until you reach or slightly exceed `xN`. This ensures that the final point is included. For example:

```
x = x0:h:xN;
```

This creates a vector `x` containing values `[0, 0.1, 0.2, ..., 0.9, 1]` with a step size of `0.1`.

4. **Specify the Initial Condition**:

Set the initial value of `y` at `x0`. This is the value of `y` at the initial point `x0`. For example:

```
y0 = 1;    % Initial value of y at x0
```

5. **Call the `EulerODE` Function**:

Use the `EulerODE` function to solve the ODE. Provide the ODE function `f`, the vector `x`, and the initial condition `y0` as arguments. The function will return the vector `y` containing the estimated values of `y` at each point in `x`.

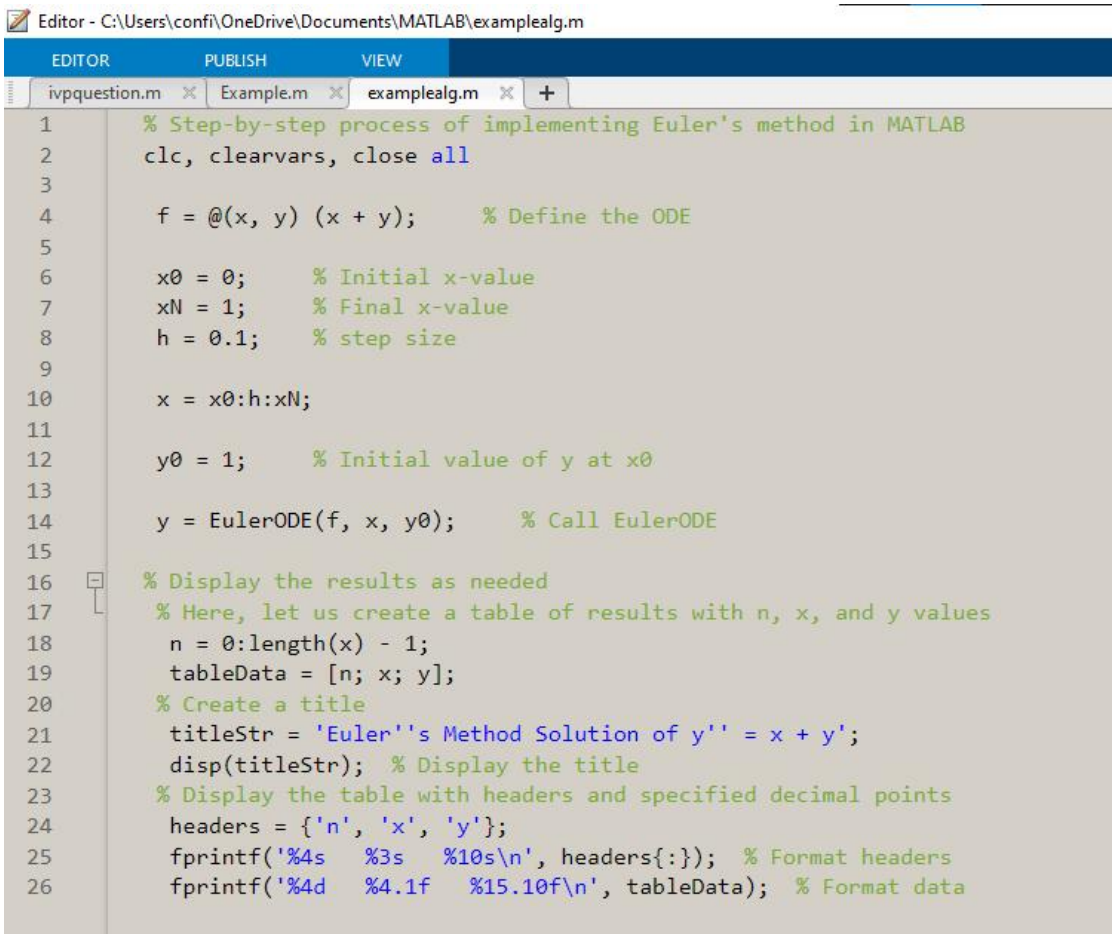
```
y = EulerODE(f, x, y0);
```

6. **Display or Analyse the Results**:

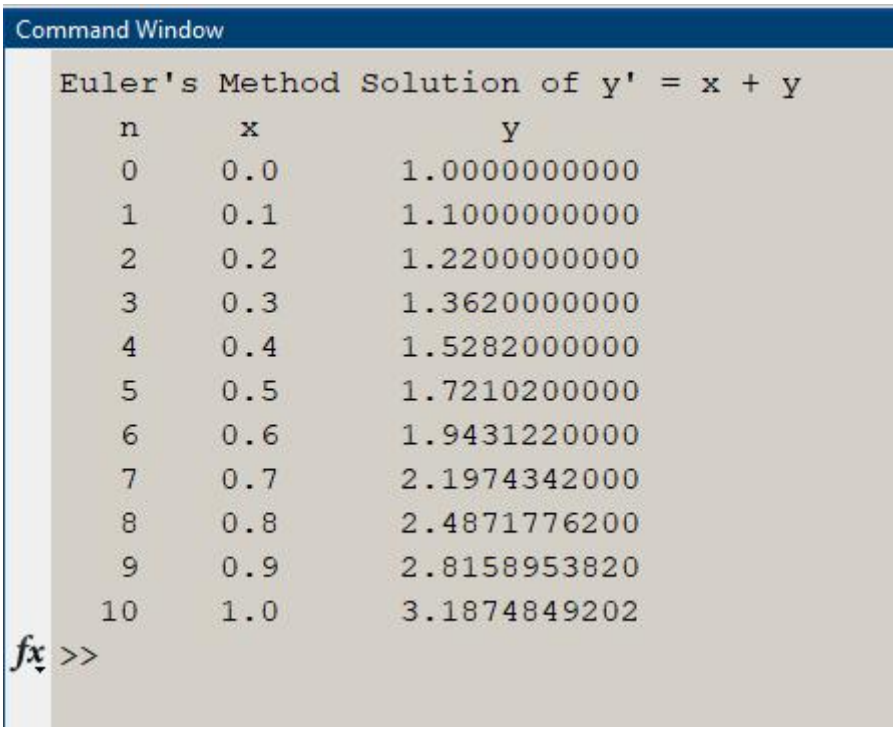
Once you have the solution vector `y`, you can display the results, plot them, or analyse them further based on the specific problem. For example:

```
% Create a table with n, x, and y values
    n = 0:length(x) - 1;
    tableData = [n; x; y];

 % Create a title
    titleStr = 'Euler''s Method Solution of dy/dx = x + y';
    disp(titleStr); % Display the title

 % Display the table with headers and specified decimal points
    headers = {'n', 'x', 'y'};
    fprintf('%4s %3s %10s\n', headers{:}); % Format headers
    fprintf('%4d %4.1f %15.10f\n', tableData); % Format data
```

Here's the complete MATLAB code that implements Euler's method for the ODE `dy/dx = x + y` over the specified range of `x` values with the given initial condition:



Program 3.3: *MATLAB code illustrating the algorithmic steps of implementing Euler's method.*



Display 3.2: *Command window display of Program 3.3*

This therefore demonstrates how MATLAB can implement Euler's Method for solving initial value problems.

3.4 MANUAL COMPUTATION OF EULER'S METHOD VS. MATLAB IMPLEMENTATION

In this section, we will compare the manual computation of Euler's method with its implementation in MATLAB by solving the IVP $y' = x^2 - y$ with the initial condition $y(0) = 1$ over the interval $[0, 10]$ and evaluate $y(10)$.

First, let's start by finding the exact solution for the IVP $y' = x^2 - y$ with the initial condition $y(0) = 1$. Although MATLAB can easily

provide the exact solution for the IVP by a simple code, we will instead solve this analytically, by hand.

The IVP is a first order ordinary differential equation. To solve it, we begin by rewriting the equation:

$$\frac{dy}{dx} = x^2 - y$$

Next, we separate the variables so that we have:

$$\frac{dy}{dx} + y = x^2$$

Now, we will use an integrating factor to solve this linear ODE. The integrating factor is given by:

$$\text{IF}(x) = e^{\int (1\,dx)}$$

$$\therefore \quad \text{IF}(x) = e^x$$

Multiplying both sides of the equation by the integrating factor, we have:

$$e^x \frac{dy}{dx} + e^x y = e^x(x^2)$$

Observe that the left-hand side of the equation is the derivative of $(e^x y)$ with respect to x. So from the product rule, it implies that:

$$\frac{d}{dx}(e^x y) = e^x(x^2)$$

Integrating both sides with respect to x:

$$\int \frac{d}{dx}(e^x y)\,dx = \int e^x(x^2)\,dx$$

On the left side, we have the integral of a derivative, which simplifies to:

$$e^x y = \int e^x (x^2)\, dx$$

Now, by integrating the right side integration by parts, we have:

$e^x y = e^x(x^2 - 2x + 2) + \mathrm{C}$, where C is the constant of integration

Finally, we solve for y. From the equation, we divide through by e^x so that:

$$y = x^2 - 2x + 2 + \mathrm{C}e^{-x}$$

Applying the initial condition $y(0) = 1$, we have:

$$1 = (0)^2 - 2(0) + 2 + \mathrm{C}e^{-(0)}$$

$$1 = 2 + \mathrm{C}$$

$$\therefore \mathrm{C} = -1$$

So, the exact solution to the IVP $y' = x^2 - y$ with the initial condition $y(0) = 1$ is:

$$y = x^2 - 2x + 2 - e^{-x}$$

Suppose $x = 10$, we would have the exact solution as:

$$y = (10)^2 - 2(10) + 2 - e^{-(10)}$$

$$y = 81.9999546$$

3.4.1 MANUAL COMPUTATION OF EULER'S METHOD

Let the step size be $h = 0.1$, given $y' = x^2 - y$ with $y(0) = 1$.

From (Equation 2.18), the Euler method is given as:,

$y_{n+1} = y_n + hf(x_n, y_n),\ x_{n+1} = x_n + h$, for $n = 0, 1, \ldots, N-1$

Where, $f(x_n, y_n) = y' = {x_n}^2 - y_n$

Iteration 1 (for $n = 0$):

$x_0 = 0,\ y_0 = 1$

Using Euler's method,

$$y_1 = y_0 + h * ({x_0}^2 - y_0)$$

$$y_1 = 1 + 0.1 * (0^2 - 1)$$

$$y_1 = 1 + 0.1 * (-1)$$

$$y_1 = 1 - 0.1$$

$$y_1 = 0.9$$

But, $x_{n+1} = x_n + h$

It implies that,

$$x_1 = x_0 + h$$

$$x_1 = 0 + 0.1$$

$$x_1 = 0.1$$

Iteration 2 (for $n = 1$):

Values from iteration 1: $x_1 = 0.1,\ y_1 = 0.9$

Using Euler's method,

$$y_2 = y_1 + h * ({x_1}^2 - y_1)$$

$$y_2 = 0.9 + 0.1 * (0.1^2 - 0.9)$$

$$y_2 = 0.9 + 0.1 * (0.01 - 0.9)$$

$$y_2 = 0.9 + 0.1 * (-0.89)$$

$$y_2 = 0.9 - 0.089$$

$$y_2 = 0.811$$

And, $x_{n+1} = x_n + h$

Therefore,

$$x_2 = x_1 + h$$

$$x_2 = 0.1 + 0.1$$

$$x_2 = 0.2$$

Iteration 3 (for *n* = 2):

Values from iteration 2: $x_2 = 0.2,\ y_2 = 0.811$

Using Euler's method,

$$y_3 = y_2 + h * ({x_2}^2 - y_2)$$

$$y_3 = 0.811 + 0.1 * (0.2^2 - 0.811)$$

$$y_3 = 0.811 + 0.1 * (0.04 - 0.811)$$

$$y_3 = 0.811 + 0.1 * (-0.771)$$

$$y_3 = 0.811 - 0.0771$$

$$y_3 = 0.7339$$

And, $x_{n+1} = x_n + h$

Therefore,

$$x_3 = x_2 + h$$

$$x_3 = 0.2 + 0.1$$

$$x_3 = 0.3$$

Iteration 4 (for *n* = 3):

Values from iteration 3: $x_3 = 0.3,\ y_3 = 0.7339$

Therefore,

$$y_4 = y_3 + h * (x_3{}^2 - y_3)$$

$$y_4 = 0.7339 + 0.1 * (0.3^2 - 0.7339)$$

$$y_4 = 0.7339 + 0.1 * (0.09 - 0.7339)$$

$$y_4 = 0.7339 + 0.1 * (-0.6439)$$

$$y_4 = 0.7339 - 0.06439$$

$$y_4 = 0.66951$$

Iteration 5 (for *n* = 4):

Following the step length interval, we can assume that $x_4 = 0.4$.

And from iteration 4, $y_4 = 0.66951$

Therefore,

$$y_5 = y_4 + h * (x_4{}^2 - y_4)$$

$$y_5 = 0.66951 + 0.1 * (0.4^2 - 0.66951)$$

$$y_5 = 0.66951 + 0.1 * (0.16 - 0.66951)$$

$$y_5 = 0.66951 + 0.1 * (-0.50951)$$

$$y_5 = 0.66951 - 0.050951$$

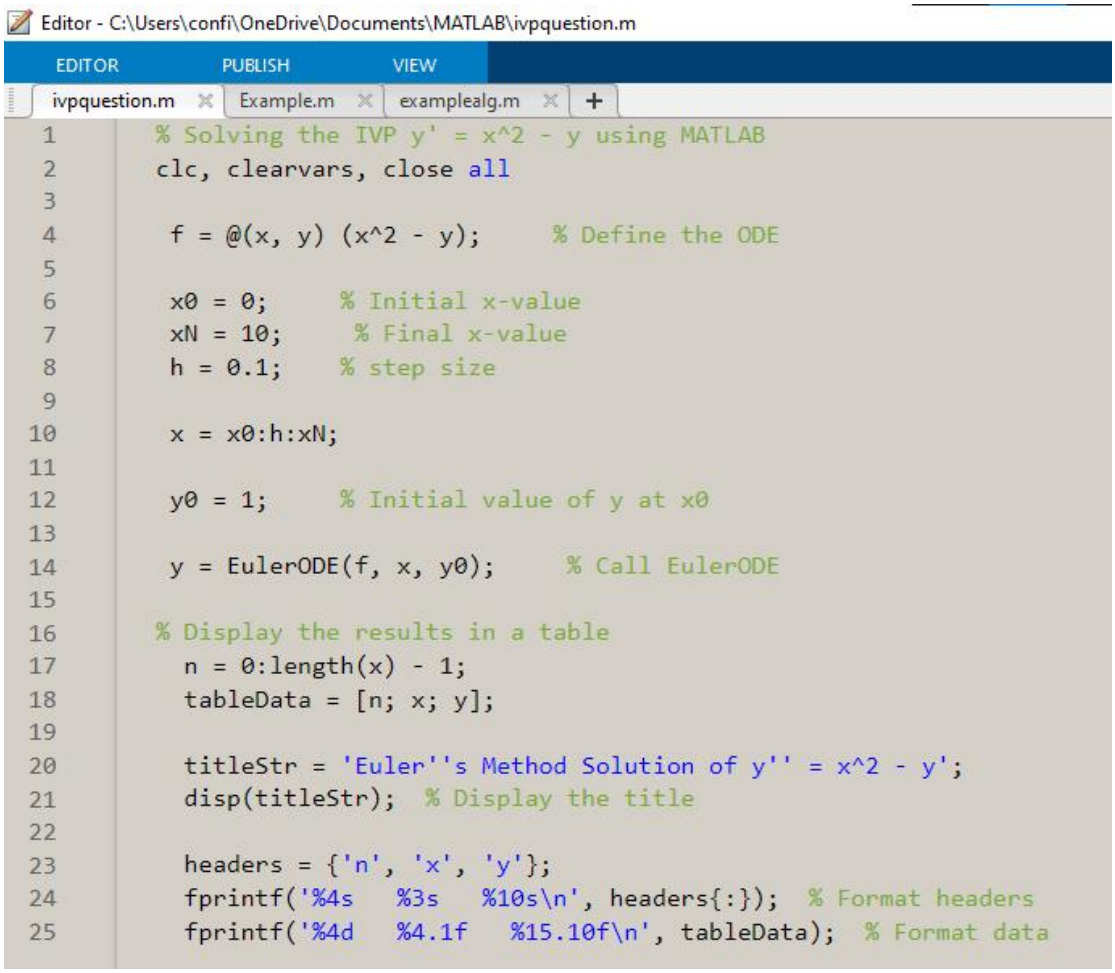
$$y_5 = 0.618559$$

After just 5 iterations of Euler's method, we have approximated $y(0.5)$ to be approximately 0.618559. To approximate $y(10)$, we would need to do a significant number of iterations. This could prove really tedious to manually compute.

3.4.2 MATLAB IMPLEMENTATION

In the previous section, we saw that approximating $y(10)$ by hand would be a daunting task as it would require 100 iterations with a step size of 0.1. However, this task can be effortlessly handled by MATLAB, which excels in repetitive calculations with high efficiency.

The MATLAB code for implementing Euler's method for the same IVP is given as:



Program 3.4: MATLAB code for Euler's method solution of

$y' = x^2 - y$

The table of results for Euler's method solution of $y' = x^2 - y$ from Program 3.4 are displayed below in Table 3.1 to Table 3.4:

Command Window

```
Euler's Method Solution of y' = x^2 - y
   n      x            y
   0     0.0      1.0000000000
   1     0.1      0.9000000000
   2     0.2      0.8110000000
   3     0.3      0.7339000000
   4     0.4      0.6695100000
   5     0.5      0.6185590000
   6     0.6      0.5817031000
   7     0.7      0.5595327900
   8     0.8      0.5525795110
   9     0.9      0.5613215599
  10     1.0      0.5861894039
  11     1.1      0.6275704635
  12     1.2      0.6858134172
  13     1.3      0.7612320755
  14     1.4      0.8541088679
  15     1.5      0.9646979811
  16     1.6      1.0932281830
  17     1.7      1.2399053647
  18     1.8      1.4049148282
  19     1.9      1.5884233454
  20     2.0      1.7905810109
  21     2.1      2.0115229098
  22     2.2      2.2513706188
  23     2.3      2.5102335569
  24     2.4      2.7882102012
  25     2.5      3.0853891811
```

Table 3.1: *Command Window display of the MATLAB code in Program 3.4*

Command Window

26	2.6	3.4018502630
27	2.7	3.7376652367
28	2.8	4.0928987130
29	2.9	4.4676088417
30	3.0	4.8618479576
31	3.1	5.2756631618
32	3.2	5.7090968456
33	3.3	6.1621871611
34	3.4	6.6349684450
35	3.5	7.1274716005
36	3.6	7.6397244404
37	3.7	8.1717519964
38	3.8	8.7235767967
39	3.9	9.2952191171
40	4.0	9.8866972054
41	4.1	10.4980274848
42	4.2	11.1292247363
43	4.3	11.7803022627
44	4.4	12.4512720364
45	4.5	13.1421448328
46	4.6	13.8529303495
47	4.7	14.5836373146
48	4.8	15.3342735831
49	4.9	16.1048462248
50	5.0	16.8953616023

Table 3.2

Command Window

51	5.1	17.7058254421
52	5.2	18.5362428979
53	5.3	19.3866186081
54	5.4	20.2569567473
55	5.5	21.1472610726
56	5.6	22.0575349653
57	5.7	22.9877814688
58	5.8	23.9380033219
59	5.9	24.9082029897
60	6.0	25.8983826907
61	6.1	26.9085444217
62	6.2	27.9386899795
63	6.3	28.9888209815
64	6.4	30.0589388834
65	6.5	31.1490449950
66	6.6	32.2591404955
67	6.7	33.3892264460
68	6.8	34.5393038014
69	6.9	35.7093734213
70	7.0	36.8994360791
71	7.1	38.1094924712
72	7.2	39.3395432241
73	7.3	40.5895889017
74	7.4	41.8596300115
75	7.5	43.1496670104

Table 3.3

Command Window

76	7.6	44.4597003093
77	7.7	45.7897302784
78	7.8	47.1397572506
79	7.9	48.5097815255
80	8.0	49.8998033729
81	8.1	51.3098230357
82	8.2	52.7398407321
83	8.3	54.1898566589
84	8.4	55.6598709930
85	8.5	57.1498838937
86	8.6	58.6598955043
87	8.7	60.1899059539
88	8.8	61.7399153585
89	8.9	63.3099238227
90	9.0	64.8999314404
91	9.1	66.5099382963
92	9.2	68.1399444667
93	9.3	69.7899500200
94	9.4	71.4599550180
95	9.5	73.1499595162
96	9.6	74.8599635646
97	9.7	76.5899672081
98	9.8	78.3399704873
99	9.9	80.1099734386
100	10.0	81.8999760947

Table 3.4

3.5 CONCLUSION

In this chapter, we have studied MATLAB as a powerful tool for solving differential equations. We have provided insights into MATLAB's capabilities and features, along with practical steps for the implementation of Euler's method in MATLAB.

In the next chapter, we will conduct a comparative analysis of manually computing Euler's method and the MATLAB implementation illustrated in this chapter. This analysis will reveal the challenges of manual computation and highlight the advantages of using MATLAB.

CHAPTER FOUR

RESULTS AND FINDINGS

4.1 COMPARATIVE ANALYSIS: MANUAL COMPUTATION VS. MATLAB IMPLEMENTATION

Manual computation of Euler's method, as demonstrated in the previous chapter, involves iteratively calculating values for each point within the specified range. While this approach may work for small-scale problems, it could become increasingly challenging for the following reasons:

4.1.1 CHALLENGES AND LIMITATIONS OF MANUAL COMPUTATION

1. **Tedious and Time-Consuming**: As seen in the methodology, continuing the iterations manually quickly becomes a tedious and time-consuming task because, with each iteration, we have to perform calculations, update values, and record results. As the number of iterations increases, so does the time required to compute the solution.
2. **Cumulative Errors**: Manual calculations are susceptible to human errors. A small mistake in one iteration could lead to significant deviations from the true solution as the process continues. And because manual computation lack the automation provided by computational tools like MATLAB, the likelihood of oversight in

calculations is high. Errors can accumulate, thereby causing inaccuracies in the final results.

3. **Limited Scalability**: As the complexity of the problem or the required precision increases, manually performing iterations becomes impractical as seen in the example in the previous chapter. There is a limit to the number of iterations one can reasonably complete, and exceeding this limit risks compromising the accuracy of the solution.
4. **Dependent on Mathematical Proficiency**: It is noteworthy that manual computation heavily relies on mathematical proficiency. Users (students or researchers) must possess a deep understanding of both the numerical method and the underlying differential equation. This dependence on expertise can be a barrier for those who are not well-versed in the mathematical intricacies of the problem.
5. **Impact on Accuracy**: The challenges mentioned above not only make manual computation time-consuming but also introduce the potential for inaccuracies. Cumulative errors, even if small individually, can lead to significant deviations from the true solution. Additionally, the limited scalability of manual computation restricts the level of precision that can be achieved.

4.1.2 ADVANTAGES OF MATLAB IMPLEMENTATION

In contrast to the challenges and limitations of manual computation, MATLAB's implementation of Euler's method offers an automated, efficient, and highly accurate approximate solution to the initial value problem tackled in the previous chapter.

Implementing MATLAB is straightforward: we define the problem and initial conditions, specify the step size, and then use the user-defined **`EulerODE`** function to efficiently perform Euler's method and solve the initial value problem, generating a table of results. The MATLAB code automates the process, reducing the chances of manual calculation errors and making it easier to visualize the results.

4.2 COMPUTATIONAL EFFICIENCY OF EULER'S METHOD IMPLEMENTATION IN MATLAB

To further illustrate MATLAB's advantages over manual computation, we will compare the numerical solutions and exact solutions for the IVP $y' = x^2 - y$ with the initial condition $y(0) = 1$ over the interval $[0, 10]$.

Using the same MATLAB code in Program 3.4, with additional details for calculating and plotting the exact solution alongside the numerical solutions from Euler's method, we have a new MATLAB code in Program 4.1 below, providing a visual analysis of the results.

plotgraph.m *

```
clc, clearvars, close all
f = @(x, y) (x^2 - y);     % Define the ODE

 x0 = 0;      % Initial x-value
 xN = 10;      % Final x-value
 h = 0.1;     % step size
 x = x0:h:xN;    % Create the vector x
 y0 = 1;      % Initial value of y at x0

 y = EulerODE(f, x, y0);     % Call EulerODE function

% Define the exact solution
exact_solution = @(x) (x.^2 - 2*x + 2 - exp(-x));
y_exact = exact_solution(x);

% Create a figure for plotting
figure;
hold on;
 % Plot the Euler Method
  plot(x, y, '-b.','LineWidth', 3, 'MarkerSize',5,  ...
      'DisplayName', 'Euler Method');
 % Plot the Exact solution
  plot(x, y_exact, 'r.', 'MarkerSize', 10, ...
      'DisplayName', 'Exact Solution');
 % Add labels
  xlabel('x');
  ylabel('y');
  title('Comparison of Numerical Solution and Exact Solution');
  legend('Location', 'NorthEast');  % Add legend

hold off;
```

Program 4.1:*MATLAB code for Numerical solutions and Exact solutions of* $y' = x^2 - y$

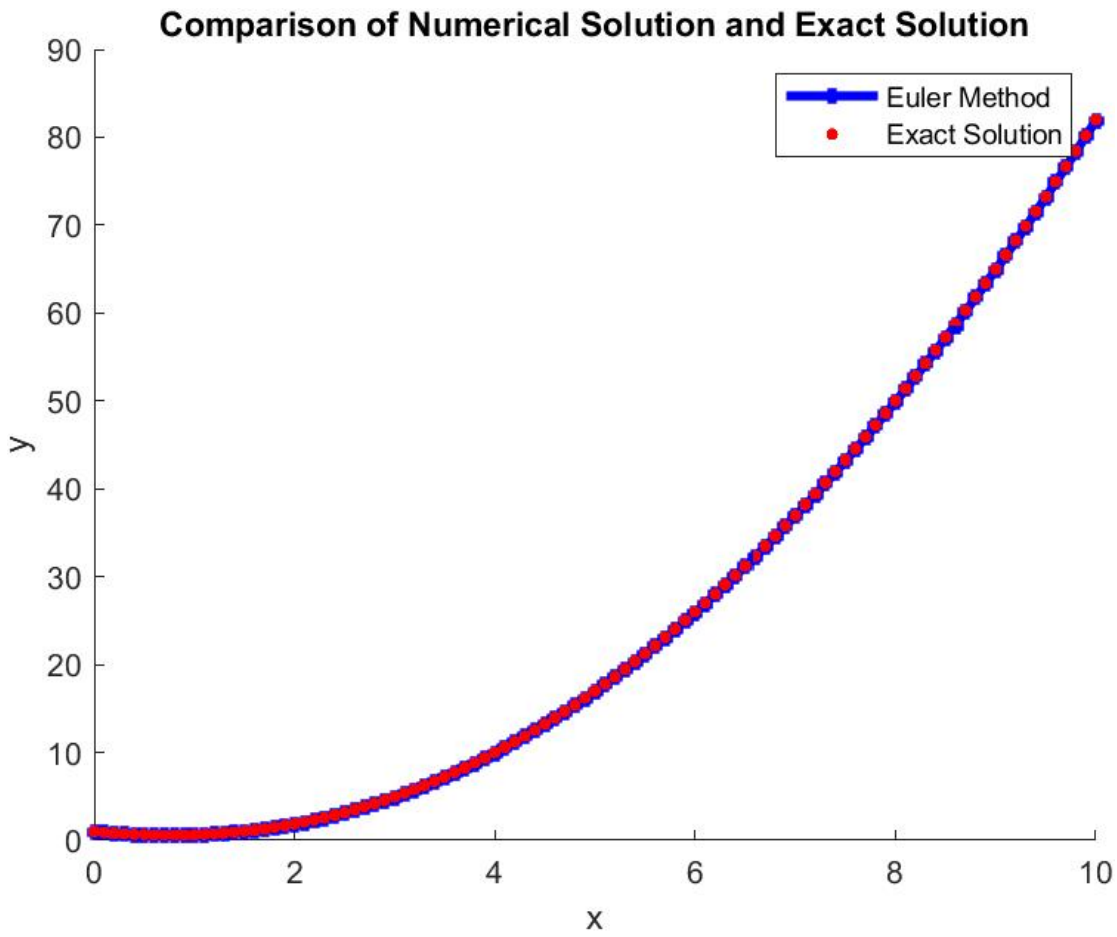


Figure 4.1: *Plot comparing the exact solutions of* $y' = x^2 - y$ *and the numerical solutions from Euler's method*

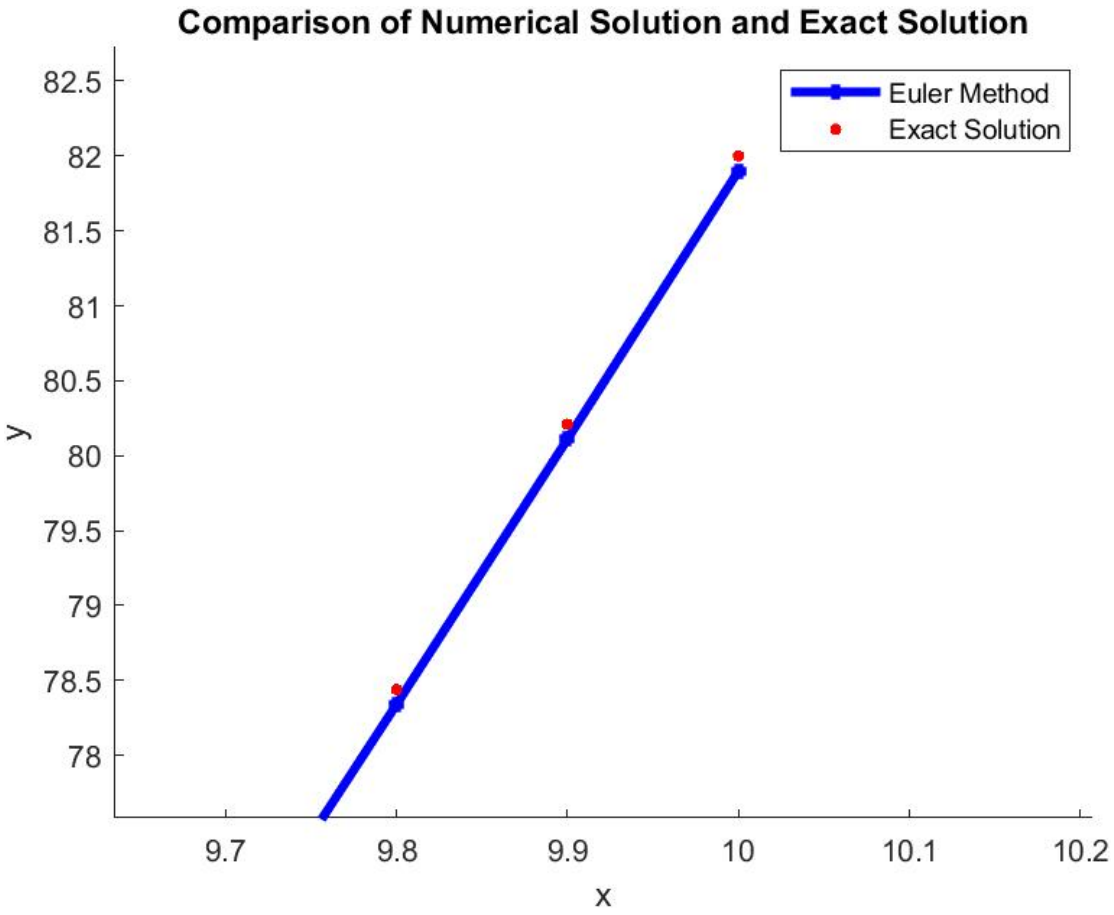


Figure 4.2

Result Discussion

Here, we analyse the results obtained from our MATLAB code in Program 4.1. The plots represented in Figure 4.1 and 4.2 above display the comparison between the exact solutions, represented by the red dots, and the numerical solutions from Euler's method using MATLAB, shown as the dotted blue line.

Upon examining the plots, we make these key observations:

1. Remarkable Accuracy: Notice the remarkable accuracy of the numerical solutions (the dotted blue line) in approximating the exact

solutions (the red dots). The close alignment of these two solutions demonstrates MATLAB's computational efficiency in solving differential equations. This accuracy is a testament to the effectiveness of numerical methods like Euler's method implemented in MATLAB.

2. Slight Deviation: While the numerical solutions do follow the same trend as the exact solutions, there are slight deviations as we see in a close up view of the plots in Figure 4.2.

4.3 CONCLUSION

In this chapter, we conducted a comprehensive comparative analysis between manual computation and MATLAB implementation of Euler's method for solving initial value problems. The results of this analysis are clear: MATLAB is a powerful tool that significantly streamlines the process of solving differential equations. The plot showcased in this chapter underscores this fact, demonstrating that even with a relatively simple numerical method like Euler's method and a moderate step size, MATLAB can yield solutions that closely align with the mathematically derived exact solution.

CHAPTER FIVE

CONCLUSION AND RECOMMENDATIONS

5.1 CONCLUSION

In the course of this project, we embarked on a journey to explore the application of Euler's method in solving initial value problems, using the powerful computational tool MATLAB. The motivation for this study arose from the need for a more efficient and reliable approach to implementing numerical methods for solving differential equations, beyond manual computations.

Through this study, we discovered that manual numerical computations can be exceptionally time-consuming, prone to errors, and insufficient for tackling complex problems. In stark contrast, MATLAB exhibited remarkable computational capabilities, significantly reducing time requirements and vastly improving accuracy.

5.2 RECOMMENDATION

One significant insight gained from this project, is the extensive applicability of MATLAB’s efficiency in solving mathematical problems, particularly in fields where numerical solutions are crucial.

I strongly recommend that educational institutions consider integrating computational software, such as MATLAB, into their curricula. This step

is vital for equipping students with the computational skills demanded by the modern world.

I hope that this work serves as a valuable resource, assisting students and researchers in enhancing their numerical analysis skills, and as a stepping stone towards more efficient and accurate numerical problem-solving across various domains.

REFERENCES

Burden, R.L. and Faires, J.D. (2011). *Numerical Analysis (9th ed.)*. Brooks/Cole, Boston, Massachusetts.

Chapra, S.C. and Raymonds, R.D. (2015). *Numerical Methods for Engineers (7th ed.)*. McGraw-Hill, New York, New York.

Esfandiari, R.S. (2017). *Numerical Methods for Engineers & Scientists using MATLAB (2nd ed.)*. CRC Press, Boca Raton, Florida.

Greenberg, M.D. (1971). *Advanced Engineering Mathematics (1st ed.)*. Prentice-Hall, Englewood Cliffs, New Jersey.

MathWorks Inc. (2023). *MATLAB*. MathWorks. https://www.mathworks.com/ (Accessed on September 13, 2023).

Matthews, J.H. and Fink, K.D. (1999). *Numerical Methods using MATLAB (3rd ed.)*. Prentice-Hall, Upper Saddle River, New Jersey.

Simmons, G.F. (1991). *Differential Equations with Applications and Historical Notes (2nd ed.)*. McGraw-Hill, New York, New York.

Stewart, J. (2008). *Calculus: Early Transcendentals.* Cengage Learning, Boston, Massachusetts.

Stroud, K.A. (2001). *Engineering Mathematics (5th ed.)*. Palgrave, Basingstoke, United Kingdom.

Wikipedia. (2023). *Differential Equations*. https://en.wikipedia.org/wiki/Differential_equation (Accessed on September 15, 2023).

Zill, D.G. (2012). *A First Course in Differential Equations with Modeling Applications*. Brooks/Cole, Boston, Massachusetts.