

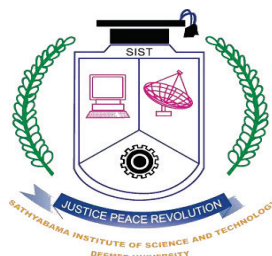
COMPARATIVE STUDY ON NUMERICAL METHODS FOR DIFFERENTIAL EQUATION

Submitted in partial fulfilment of the requirements for the award of
Bachelor of Science Degree in Mathematics

by

V. PRIYADHARSHINI (40060012)

R. RASHIKA (40060013)



**DEPARTMENT OF MATHEMATICS
SCHOOL OF SCIENCE AND HUMANITIES**

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

**Accredited with Grade "A" by NAAC I12B Status by UGC I approved by AICTE
JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600 119**

April - 2023



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited with "A" grade by NAAC | 12B status by UGC | approved by AICTE
Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai – 600 119
www.sathyabama.ac.in

DEPARTMENT OF MATHEMATICS

BONAFIDE CERTIFICATE

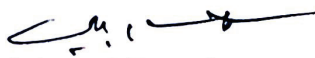
This is to certify that this Project Report is the Bonafide work of **V. PRIYADHARSHINI**
(40060012), **R. RASHIKA** (40060013) who carried out the project entitled
"COMPARATIVE STUDY ON NUMERICAL METHODS FOR DIFFERENTIAL
EQUATION" under our supervision from **JANUARY 2023** to **APRIL 2023**.

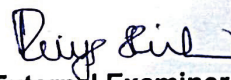

Internal guide

Dr.S.CYNTHIYA MARGARET INDRANI


Head of the Department
Dr. M. NIRMALA, M.Sc., M.Phil., Ph.D.,
Head of the Department
Dept. of Mathematics
Sathyabama Institute of Science and Technology
(Deemed to be University) Chennai-600 119.
(Dr. M. NIRMALA)

Submitted for Viva voce Examination held on 09-05-2023


Internal Examiner


External Examiner

DECLARATION

We **V. PRIYADHARSHINI** and **R. RASHIKA** hereby declare that the Project Report entitled "**COMPARATIVE STUDY ON NUMERICAL METHODS FOR DIFFERENTIAL EQUATION**" done by us under the guidance of **Dr.S.CYNTHIYA MARGARET INDRANI** at **SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY** is submitted in partial fulfilment of the requirements for the award of Bachelor of Science degree in Mathematics.

DATE: 29.04.2023

PLACE: Chennai

V. Priyadharshini

R. Rashika

SIGNATURE OF THE CANDIDATES

(V. PRIYADHARSHINI)

(R. RASHIKA)

ACKNOWLEDGEMENT

We are pleased to acknowledge our sincere thanks to Board of Management of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. We are grateful to them.

We convey our thanks to **Dr. REKHA CHAKRAVARTHI**, Dean, School of Arts and Sciences and **Dr. M. NIRMALA**, Head of the Department, Dept. of Mathematics for providing us necessary support and details at the right time during the progressive reviews.

We would like to express our sincere and deep sense of gratitude to our Project Guide **Dr.S.CYNTHIYA MARGARET INDRANI** for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of our project work.

We wish to express our thanks to all Teaching and Non-teaching staff members of the Department of Mathematics who were helpful in many ways for the completion of the project

ABSTRACT

A comparative study on numerical methods for differential equations is an investigation of the accuracy, stability, and computational efficiency of different numerical techniques used to solve differential equations with MATLAB. The study compares various numerical methods such as Euler's method, Improved Euler's method, Runge-Kutta method. The goal is to identify the strengths and weaknesses of each method and provide insight into the selection of the most appropriate method for specific problems. The study involves both theoretical analysis and practical implementation of numerical methods on various differential equations. The results obtained from the comparative study can be used to guide the selection of the most suitable numerical method for a given differential equation problem.

TABLE OF CONTENT

CHAPTER 1	PAGE NO
ERROR APPROXIMATION	
1.1 INTRODUCTION	
1.2 ERROR TERMS	
1.3 TYPES OF ERROR	1 - 4
1.3.1 ROUNDING ERRORS	
1.3.2 UNCERTAINTY	
1.3.3 TRUNCATION	
CHAPTER 2	
LITERATURE SURVEY	5 - 7
CHAPTER 3	
AIM AND SCOPE	8 - 9
CHAPTER 4	
SOLVING EULER METHOD BY USING MATLAB CODING	
4.1 INTRODUCTION	
4.2 NUMERICAL EXAMPLE	10 - 21
4.3 CODING FOR COMPUTING EULER METHOD	
CHAPTER 5	
SOLVING AN IMPROVED EULER METHOD BY USING MATLAB CODING	
5.1 INTRODUCTION	
5.2 NUMERICAL EXAMPLE	22 - 32
5.3 CODING FOR COMPUTING IMPROVED EULER METHOD	
CHAPTER 6	
SOLVING THE RUNGE-KUTTA METHOD BY USING MATLAB CODING	
6.1 INTRODUCTION	
6.2 NUMERICAL EXAMPLE	33 - 42
6.3 CODING FOR COMPUTING RUNGE-KUTTA METHOD	
CHAPTER 7	
SUMMARY AND CONCLUSION	43 - 44

LIST OF TABLES			
S.NO	TABLE NO.	TITLE	PAGE NO.
1	4.1	Tabulated Values for Exact and Numerical Solutions to Equation with $h = 0.2$	14
2	4.2	Tabulated Values for Exact and Numerical Solutions to Equation with $h = 0.1$	15
3	5.1	Tabulated Values for Exact and Numerical Solutions to Equation with $h = 0.2$	25
4	5.2	Tabulated Values for Exact and Numerical Solutions to Equation with $h = 0.1$	26
5	6.1	Tabulated Values for Exact and Numerical Solutions to Equation with $h = 0.2$	36
6	6.2	Tabulated Values for Exact and Numerical Solutions to Equation with $h = 0.1$	37

CHAPTER 1

ERROR APPROXIMATION

1.1 INTRODUCTION

An Equation involving one dependent variable and its derivatives with respect to one or more independent variables is called Differential Equations. Although there are many analytical method for finding the solution of differential equations. There exists a quite number of differential equations that cannot be solved analytically. (ie) the solutions cannot be expressed on the sum of finite number of elementary follows (polynomial, exponents). Numerical methods provide powerful alternate tool for solving the differential equations under predictable initial conditions.

When we create a numerical or discrete model for a differential equation, we make several decisive replacements or substitutions. First, the derivatives in the equation are replaced by differences. Second, the continuous variable x is replaced by a discrete variable. Third, the real number line is replaced by a discrete set of values. Any type of approximation argument involves some sort of loss of information, there will always be an error term. It is also the case that these numerical approximation techniques can give rise to instability phenomena and other unpredictable behaviour.

The spirit of the numerical method can be illustrated with a basic example. Consider the simple differential equation

$$y' = y, \quad y(0) = 1$$

The initial condition tells that the point $(0, 1)$ lies on the graph of the solution y . The equation itself tells that, at that point, the slope of the solution is

$$y' = y = 1$$

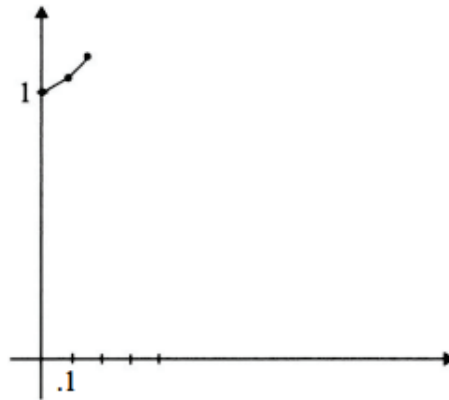
Thus, the graph will proceed to the right with slope 1.

Let us assume the numerical calculation with 0.1. So we proceed to the right to the point (0.1, 1.1). This is the second point on our "**approximate solution graph**".

Now we return to the differential equation to obtain the slope of the solution at this new point. It is

$$y' = y = 1.1$$

Thus, when we proceed to sketch the approximate solution graph to the right of (0.1, 1.1), we draw a line segment of slope 1.1 to the point (0.2, 1.21).



(Fig 1.1)

1.2 ERROR TERM

In numerical methods, the error term refers to the difference between the exact or true solution of a mathematical problem and the approximate solution obtained using a numerical method. Since numerical methods involve approximations and finite precision arithmetic, there is always some level of error in the calculated solution. The error term quantifies this difference between the exact and approximate solutions.

1.3 TYPES OF ERROR

There are three main sources of errors in numerical computation: rounding, data uncertainty, and truncation.

- **1.3.1 Rounding Errors**, also called arithmetic errors, are an unavoidable consequence of working in finite precision arithmetic. It deal with these errors in the context of polynomial evaluation and solving linear equations.
- **1.3.2 Uncertainty** in the data is always a possibility when we solving practical problems. It may arise in several ways: from errors in measuring physical quantities, from errors in storing the data on the computer (rounding errors), or, if the data is itself the solution to another problem, it may be the result of errors in an earlier computation. The effects of errors in the data are generally easier to understand than the effects of rounding errors committed during a computation, because data errors can be analyzed using perturbation theory for the problem, while intermediate rounding errors require an analysis specific to the given method.
- **1.3.3 Truncation Or Discretization Or Approximation Errors** are much harder to analyze. It deal with the context of solving differential equations. Many standard numerical methods can be derived by taking finitely many terms of a Taylor series. The terms omitted constitute the truncation error, and for many methods, the size of this error depends on h , whose appropriate value is a compromise between obtaining a small error and a fast computation. For many scientific problems, such approximations are necessary.

CHAPTER 2

LITERATURE SURVEY

The origin of numerical methods for differential equations can be traced back to the 18th century, when mathematicians began to study the behavior of differential equations. One of the pioneers in this field was Leonhard Euler, who developed a method for solving ordinary differential equations (ODEs) known as Euler's method.

Euler's method involves approximating the solution to an ODE by stepping through the solution at discrete time intervals. This approach forms the basis of many numerical methods for ODEs, including the Runge-Kutta methods, which were developed in the early 20th century by Carl Runge and Martin Kutta.

In the mid-20th century, the development of digital computers led to the development of more sophisticated numerical methods for differential equations. One important development was the finite difference method, which involves approximating derivatives in a differential equation using finite differences. This method can be used to solve both ODEs and partial differential equations (PDEs).

Other numerical methods for differential equations that have been developed include the finite element method, the spectral method, and the boundary element method. These methods have been applied to a wide range of problems in science and engineering, including fluid dynamics, structural analysis, and quantum mechanics. This section would provide an overview of the topic, outlining the importance of differential equations in various fields of science and engineering, and explaining the need for numerical methods to solve these equations. The section would also discuss the objectives and scope of the study.

The section would be divided into subsections based on the different methods, such as:

- The euler method involves approximating the derivative of the solution at a point using the slope of the tangent line at that point. The approximate solution is then found by moving along the tangent line by a small step. It contains numerous studies that apply the Euler method to various differential equations. The accuracy of the method can be improved by reducing the step size, but this increases the computational cost.
- The improved method involves using a midpoint approximation of the solution between the two points used in the Euler method. The derivative is approximated at the midpoint, and the solution is then advanced using this derivative. It contains many studies that apply the Improved Euler method to various differential equations. The method is particularly useful for problems where the solution changes rapidly.
- This subsection would discuss the basics of the Runge-Kutta method, its applications, advantages, and limitations. It would also review various research articles that have used this method in solving differential equations.
- Comparative studies of these methods have been conducted. For example, one study compared the accuracy and efficiency of the Euler method, the Improved Euler method, and the Runge-Kutta method for solving a differential equation using MATLAB coding. The study found that the Runge-Kutta method was the most accurate and efficient method for this problem.

Overall, the literary survey would provide a comprehensive overview of the existing literature on numerical methods for differential equations, and would help the reader to understand the advantages and limitations of each method.

CHAPTER 3

AIM AND SCOPE

The aim of a comparative study on numerical methods for differential equations is to compare different methods for solving differential equations with matlab code and evaluate their effectiveness in terms of accuracy, computational efficiency, and ease of implementation. The study can help researchers and practitioners in the field of numerical analysis to select the most appropriate method for their specific application.

The scope of the study may include a variety of numerical methods for solving ordinary differential equations using matlab coding for the methods such as Method of euler, Improved euler and Runge-Kutta method. The study may also evaluate the performance of different solvers and software packages for implementing the numerical methods.

Overall, the aim and scope of a comparative study on numerical methods for differential equations is to provide a comprehensive evaluation of different methods and solvers for solving differential equations, and to help researchers and practitioners make informed decisions when selecting a numerical method for their specific application.

CHAPTER 4

SOLVING EULER METHOD BY USING MATLAB CODING

4.1 INTRODUCTION

The **Euler method** also called the **forward Euler method** is a first-order numerical procedure for solving ordinary differential equations (ODEs) with a given initial value. It is the most basic explicit method for numerical integration of ordinary differential equations. The Euler method is a first-order method, which means that the local error is proportional to the square of the step size, and the global error is proportional to the step size. The Euler method often serves as the basis to construct more complex methods.

Consider an initial value problem of the form

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

We may integrate from x_0 to $x_1 = x_0 + h$ to obtain

$$y(x_1) - y(x_0) = \int_{x_0}^{x_1} f(x, y) dx$$

or

$$y(x_1) = y(x_0) + \int_{x_0}^{x_1} f(x, y) dx$$

Since the unknown function y occurs in the integrand on the right, we cannot proceed unless we have some method of approximating the integral.

The Euler method is obtained from the simplest technique for approximating the integral. Namely, we assume that the integrand does not vary much on the interval $[x_0, x_1]$, and therefore that a rather small error will result if we replace $f(x, y)$ by its value at the left endpoint. Now, we put in place a partition $a = x_0 < x_1 < x_2 < \cdots < x_k = b$ of the interval $[a, b]$. Assume that each interval $[x_{j-1}, x_j]$ has length h . We set $Y_0 = y(x_0)$. Now we take

$$\begin{aligned}
y(x_1) &= y(x_0) + \int_{x_0}^{x_1} f(x_0, y_0) dx \\
&\approx y(x_0) + \int_{x_0}^{x_1} f(x_0, Y_0) dx \\
&= y(x_0) + h \cdot f(x_0, Y_0).
\end{aligned}$$

Based on this calculation, we simply define

$$y_1 = y_0 + h \cdot f(x_0, Y_0).$$

By Continuing, we set $x_k = x_{k-1} + h$ and define

$$y_{k+1} = y_k + h \cdot f(x_k, y_k).$$

Then the points $(x_0, y_0), (x_1, y_1), \dots, (x_k, y_k), \dots$ are the points of our "**approximate solution**" to the differential equation.

It is sometimes convenient to measure the total relative error $\bar{E}_n$ at the n th step; this quantity is defined to be

$$\bar{E}_n = \frac{|y(x_n) - y_n|}{|y(x_n)|}$$

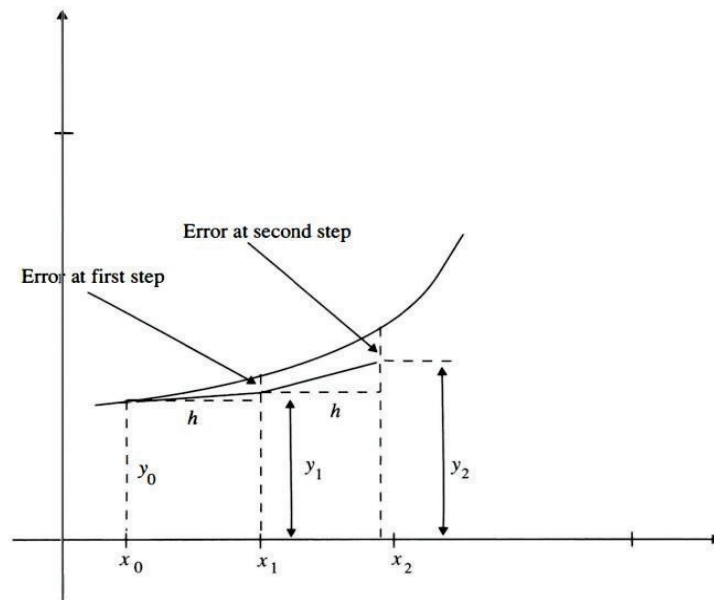
We express this quantity as a percentage, and we obtain thereby a comfortable way of measuring how well the numerical technique under consideration is performing.

To examine the initial value problem

$$y' = x + y, \quad y(0) = 1$$

over and over again using different methods of numerical analysis. We may use the method of first-order linear equations to find the explicit solution

$$y = -1 - x + 2e^x$$



(fig 4.1)

4.2 NUMERICAL EXAMPLE

Apply the Euler technique to the ordinary differential equation

$$y' = x + y, \quad y(0) = 1$$

using increments of size $h = 0.2, 0.1$.

SOLUTION:

In the first line of this table, the initial condition $y(0) = 1$ determines the slope $y' = x + y = 1.00$.

Since, $h = 0.2$ and $Y_1 = Y_0 + h \cdot f(x_0, Y_0)$,

the next value is given by $1.00 + 0.2 \cdot (1.00) = 1.20$. This process is iterated in the succeeding lines. The expression $\overline{E}_n$ represents the percent of error. For instance, in the second line of the table it is calculated as

$$\overline{E}_1 = \frac{1.24281 - 1.2}{1.24281} \approx .034$$

and we represent the quantity in the table as a percent so it is 3.4 percent.

Table 4.1 Tabulated Values for Exact and Numerical Solutions to Equation with $h = 0.2$

x_n	y_n	Exact	$\overline{E}_n$ (%)
0.0	1.00000	1.00000	0.0
0.2	1.20000	1.24281	3.4
0.4	1.48000	1.58365	6.5
0.6	1.85600	2.04424	9.2
0.8	2.34720	2.65108	11.5
1.0	2.97664	3.43656	13.4

For comparison purposes, we also record in Table 9.2 the tabulated values for $h = 0.1$

Table 4.2 Tabulated Values for Exact and Numerical Solutions to Equation with $h=0.1$

x_n	y_n	Exact	$\overline{E}_n$ (%)
0.0	1.00000	1.00000	0.0
0.1	1.10000	1.11034	0.9
0.2	1.28200	1.24281	1.8
0.3	1.36200	1.39972	2.7
0.4	1.52820	1.58365	3.5
0.5	1.72102	1.79744	4.3
0.6	1.94312	2.04424	4.9
0.7	2.19743	2.32751	5.6
0.8	2.48718	2.65108	6.2
0.9	2.81590	3.01921	6.7
1.0	3.18748	3.43656	7.2

The displayed data makes clear that reducing the step size will increase accuracy. But the tradeoff is that significantly more computation is required.

4.3 CODING FOR COMPUTING EULER METHOD:

```
% Prompt user to input step size and differential equation
h = input('Enter the step size (h): ');
eqn = input('Enter the differential equation (dy/dx): ','s');

% Convert input equation string to function handle
f = str2func(['@(x,y) ', eqn]);

% Prompt user to input initial condition
x0 = input('Enter the initial value of x: ');
y0 = input('Enter the initial value of y: ');

% Prompt user to input final value of x for which y is to be calculated
xf = input('Enter the final value of x: ');

% Initialize arrays to store the numerical solutions and errors
x = x0:h:xf;
y = zeros(size(x));
error = zeros(size(x));

% Set the initial value
y(1) = y0;

% Compute the numerical solution and error for each step
for i = 1:length(x)-1
    y(i+1) = y(i) + h*f(x(i), y(i));
    exact_sol = exp(x(i+1)) - x(i+1) - 1;
    error(i+1) = abs(exact_sol - y(i+1));
end
```

```
% Plot the numerical solution and exact solution
plot(x, y, '-o', x, exp(x) - x - 1, 'LineWidth', 1.5);
legend('Numerical Solution', 'Exact Solution');
xlabel('x');
ylabel('y');
title('Numerical and Exact Solutions using Euler"s Method');
```

```
% Plot the error at each step
figure;
plot(x, error, '-o', 'LineWidth', 1.5);
xlabel('x');
ylabel('Error');
title('Error in Numerical Solution using Euler"s Method');
```

```
disp(['x: ', num2str(x)]);
disp(['y: ', num2str(y)]);
disp(['Error: ', num2str(error)]);
```

OUTPUT:

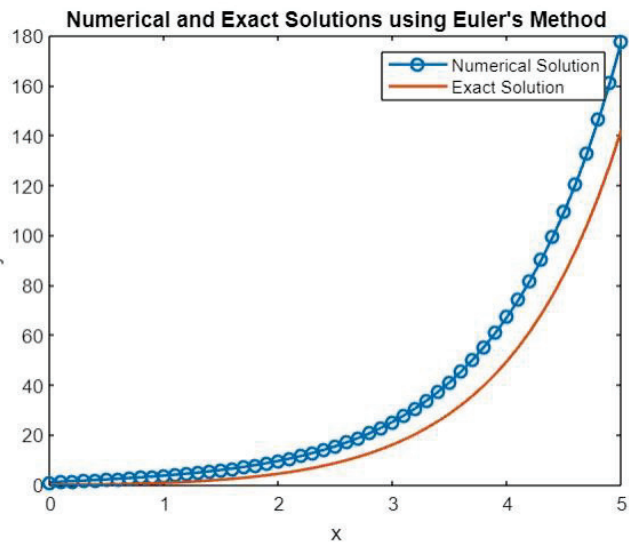
```
>>Eulermethod
Enter the step size (h):
0.1
Enter the differential equation (dy/dx): y-sin(x)+1
Enter the initial value of x: 0
Enter the initial value of y:
1
Enter the final value of x:
5
```

x	y	Error
0	1	0
0.1	1.2	1.19483
0.2	1.410017	1.38861
0.3	1.631151	1.58129
0.4	1.864715	1.77289
0.5	2.112244	1.96352
0.6	2.375526	2.15341
0.7	2.656614	2.34286
0.8	2.957854	2.53231
0.9	3.281904	2.7223
1	3.631761	2.91348
1.1	4.010791	3.10662
1.2	4.422749	3.30263
1.3	4.87182	3.50252
1.4	5.362646	3.70745
1.5	5.900366	3.91868
1.6	6.490653	4.13762
1.7	7.139761	4.36581
1.8	7.85457	4.60492
1.9	8.642642	4.85675
2	9.512277	5.12322
2.1	10.47257	5.4064
2.2	11.53351	5.7085

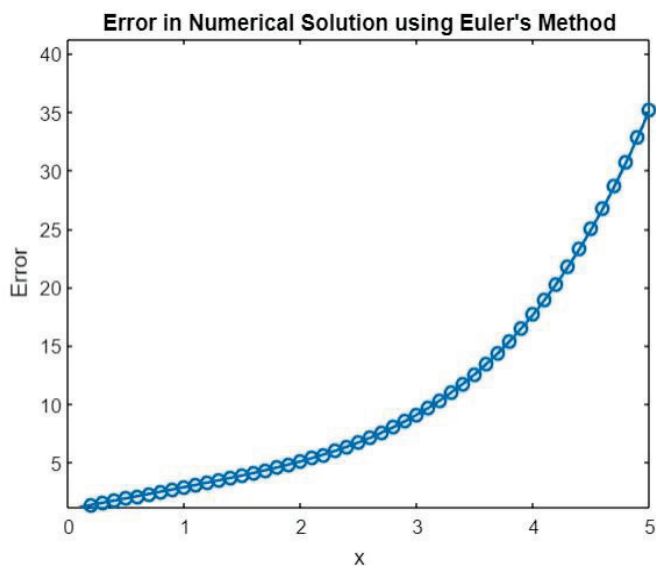
2.3	12.70601	6.03183
2.4	14.00204	6.37887
2.5	15.4347	6.75221
2.6	17.01832	7.15459
2.7	18.76861	7.58887
2.8	20.70273	8.05808
2.9	22.8395	8.56536
3	25.19953	9.11399
3.1	27.80537	9.70742
3.2	30.68175	10.3492
3.3	33.85576	11.0431
3.4	37.35711	11.793
3.5	41.21838	12.6029
3.6	45.47529	13.4771
3.7	50.16707	14.4198
3.8	55.33676	15.4356
3.9	61.03163	16.5292
4	67.30357	17.7054
4.1	74.2096	18.9693
4.2	81.81239	20.3261
4.3	90.18079	21.781
4.4	99.39048	23.3396
4.5	109.5247	25.0076
4.6	120.6749	26.7906

4.7	132.9418	28.6946
4.8	146.4359	30.7255
4.9	161.2792	32.8894
5.0	177.6053	35.1922

GRAPH:



(Graph 4.3.1)



(Graph 4.3.2)

CHAPTER 5

SOLVING AN IMPROVED EULER METHOD BY USING MATLAB CODING

5.1 INTRODUCTION

In the Improved Euler method, the solution is approximated by computing two estimates for the next point in time, one based on the slope of the ODE at the current point, and another based on the slope at a predicted point using the first estimate. The final approximation is then obtained by taking the average of these two estimates.

Euler's method was to replace the integrand by $f(x_0, y_0)$. This generated the iterative scheme of the last section. Now we propose to instead replace the integrand with $[f(x_0, y_0) + f(x_1, y(x_1))]/2$. Thus, we find that

$$y_1 = y_0 + 2 [f(x_0, y_0) + f(x_1, y(x_1))]$$

This proposed equation is that $y(x_1)$ is unknown-just because we do not know the exact solution y . What we can do instead is to replace $y(x_1)$ by its approximate value as found by the Euler method. Denote this new value by $z_1 = y_0 + h \cdot f(x_0, y_0)$. Then Equation becomes

$$y_1 = y_0 + 2 \cdot [f(x_0, y_0) + f(x_1, z_1)]$$

To verify that each quantity on the right-hand side can be calculated from information that we have-without knowledge of the exact solution of the differential equation. More generally, our iterative scheme is

$$y_{j+1} = y_j + h/2 \cdot [f(x_j, y_j) + f(x_{j+1}, z_{j+1})],$$

Where,

$$z_{j+1} = y_j + h \cdot f(x_j, y_j)$$

and $j = 0, 1, 2, \dots$

This new method, usually called the improved Euler method or **Heun's method**, first predicts and then corrects an estimate for y_j . It is an example of numerical techniques called predictor-corrector methods. It is possible, using subtle Taylor series arguments, to show that the local discretization error is

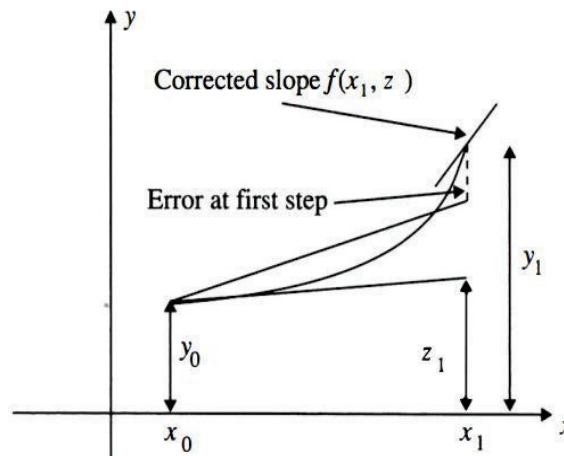
$$\epsilon_j = -y'''(\xi) \cdot h^3/12$$

for some value of ξ between x_0 and x_n . Thus, in particular, the total discretization error is proportional to h^2 , so we can get accuracy for the same step size.

To Visualize the improved Euler method. First, the point at (x_1, z_1) is predicted using the original Euler method, then this point is used to estimate the slope of the solution curve at x_1 . This result is then averaged with the original slope estimate at (x_0, y_0) to make a better prediction of the solution-namely, (x_1, y_1) .

We shall continue to examine,

$$y' = x + y, \quad y(0) = 1$$



(fig 5.1)

5.2 NUMERICAL EXAMPLE

Apply the improved Euler method to the differential equation

$$y' = x + y, \quad y(0) = 1$$

with step size 0.2 and gauge the improvement in accuracy over the ordinary Euler method

SOLUTION:

We see that

$$z_{k+1} = y_k + 0.2 \cdot f(x_k, y_k) = y_k + 0.2(x_k + y_k)$$

And

$$y_{k+1} = y_k + 0.1 [(x_k + y_k) + (x_{k+1} + z_{k+1})].$$

We begin the calculation by setting $k = 0$ and using the initial values $x_0 = 0.0000$, $y_0 = 1.0000$.

Thus,

$$z_1 = 1.0000 + 0.2(0.0000 + 1.0000) = 1.2000$$

And

$$y_1 = 1.0000 + 0.1 [(0.0000 + 1.0000) + (0.2 + 1.2000)] = 1.2400.$$

We continue this process and obtain the values.....

Table 5.1 Tabulated Values for Exact and Numerical Solutions to Equation with $h=0.2$

x_n	y_n	Exact	$\overline{E_n}$ (%)
0.0	1.00000	1.00000	0.0
0.2	1.24000	1.24281	0.23
0.4	1.57680	1.58365	0.43
0.6	2.03170	2.04424	0.61
0.8	2.63067	2.65108	0.77
1.0	3.40542	3.43656	0.91

We see that the resulting approximate value for $y(1)$ is 3.40542. The aggregate error is about 1 percent, whereas with the former Euler method it was more than 13 percent. This is a substantial improvement. Therefore, a smaller step size results in even more dramatic improvement in accuracy.

Table 5.2 Tabulated Values for Exact and Numerical Solutions to Equation with $h = 0.1$

x_n	y_n	Exact	$\overline{E}_n$ (%)
0.0	1.00000	1.00000	0.0
0.1	1.11000	1.11034	0.0
0.2	1.24205	1.24281	0.1
0.3	1.39847	1.39972	0.1
0.4	1.58180	1.58365	0.1
0.5	1.79489	1.79744	0.1
0.6	2.04086	2.04424	0.2
0.7	2.32315	2.32751	0.2
0.8	2.64558	2.65108	0.2
0.9	3.01236	3.01921	0.2
1.0	3.42816	3.43656	0.2

equation using a step size of $h = 0.1$. The relative error at $x = 1.00000$ is now about 0.2 percent, which is another order of magnitude of improvement in accuracy. We have predicted that halving the step size will decrease the aggregate error by a factor of 4. These results bear out that prediction.

5.3 CODING FOR COMPUTING AN IMPROVED EULER METHOD:

```
% Get user input for the differential equation  $y' = f(x, y)$ 
prompt = 'Enter the differential equation y" = ';
f_str = input(prompt, 's');
f = str2func(['@(x, y) ', f_str]);

% Get user input for the initial conditions
x0 = input('Enter the initial value of x: ');
y0 = input('Enter the initial value of y: ');

% Get user input for the step size and the maximum value of x
h = input('Enter the step size: ');
xmax = input('Enter the maximum value of x: ');

% Initialize arrays to store x, y, and the error
x = x0:h:xmax;
y = zeros(size(x));
err = zeros(size(x));

% Set initial values
y(1) = y0;

% Implement Improved Euler's Method
for i = 1:length(x)-1
    % Compute  $y_{\text{hat}}$  using Euler's method
    y_hat = y(i) + h*f(x(i), y(i));

    % Compute the next value of y using Improved Euler's method
    y(i+1) = y(i) + (h/2)*(f(x(i), y(i)) + f(x(i+1), y_hat));
```

```

    % Compute the exact solution at x(i+1)
    exact = exp(x(i+1)^2/2);

    % Compute the error
    err(i+1) = abs(exact - y(i+1));

    % Check if the maximum value of x has been reached
    if x(i+1) >= xmax
        break;
    end
end

% Plot the numerical solution and the exact solution
plot(x, y, '-o', x, exp(x.^2/2), '-');
legend('Improved Euler', 'Exact');
xlabel('x');
ylabel('y');
title(['Numerical Solution for y'' = ', f_str]);

% Plot the error
figure;
plot(x, err, '-o');
xlabel('x');
ylabel('Error');
title(['Error in Numerical Solution for y'' = ', f_str]);

disp(['x: ', num2str(x)]);
disp(['y: ', num2str(y)]);

```

```
disp(['Error: ', num2str(err)]);
```

OUTPUT:

```
>>improvedeulermethod
Enter the differential equation y' = y-
sin(x)+1
Enter the initial value of x:
0
Enter the initial value of y:
1
Enter the step size:
0.1
Enter the maximum value of x:
5
```

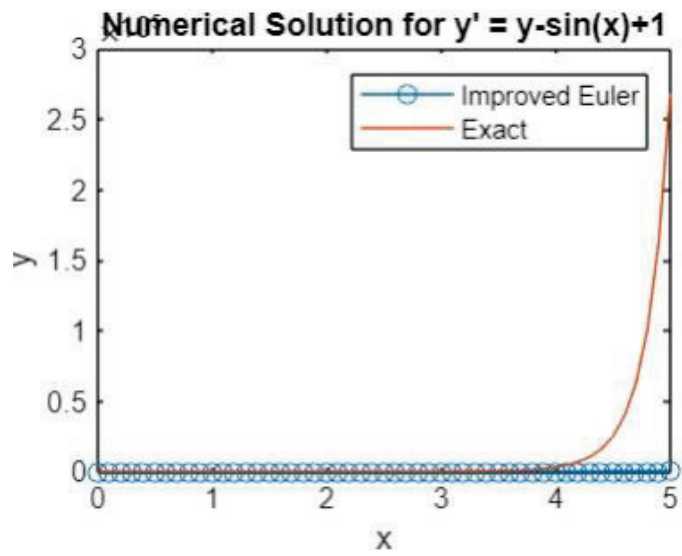
x	y	Error
0	1	0
0.1	1.205008	0.199996
0.2	1.421110	0.400909
0.3	1.649624	0.603596
0.4	1.892110	0.808822
0.5	2.150392	1.017243

0.6	2.426582	1.229365
0.7	2.723107	1.445486
0.8	3.042734	1.665606
0.9	3.388600	1.889297
1.0	3.764246	2.115525

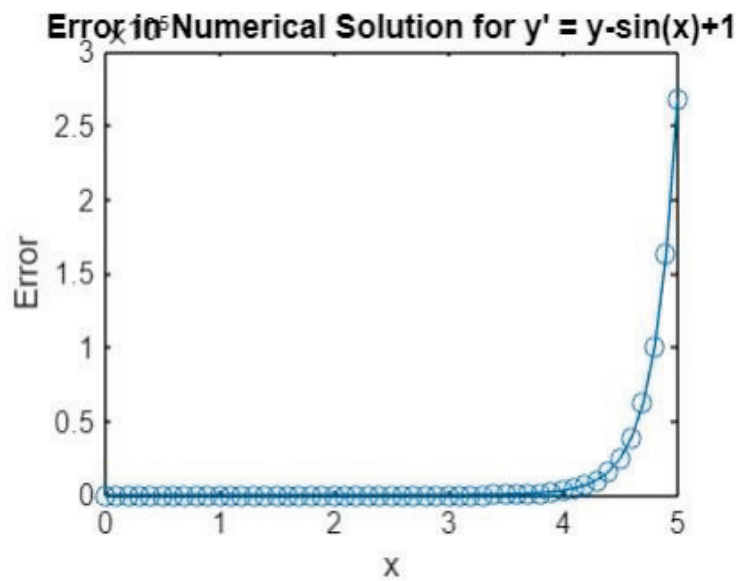
1.1	4.173651	2.342399
1.2	4.621266	2.566833
1.3	5.112059	2.784081
1.4	5.651557	2.987101
1.5	6.245896	3.165679
1.6	6.901874	3.305234
1.7	7.627011	3.385159
1.8	8.429613	3.376523
1.9	9.318846	3.238875
2.0	10.304810	2.915757
2.1	11.398650	2.328395
2.2	12.612600	1.366744
2.3	13.960170	0.123271
2.4	15.456210	2.358067
2.5	17.117030	5.642862
2.6	18.960630	10.410140
2.7	21.006780	17.275997
2.8	23.277230	27.123213
2.9	25.795950	41.224644
3.0	28.589310	61.427817

3.1	31.686350	90.433139
3.2	35.119050	132.216319
3.3	38.9227	192.6746309
3.4	43.1360	280.6232105
3.5	47.8019	409.3428615
3.6	52.9675	599.003481
3.7	58.6849	880.4884014
3.8	65.0115	1301.477535
3.9	72.0108	1936.201117
4.0	79.7526	2901.205412
4.1	88.3141	4381.043629
4.2	97.7807	6670.483923
4.3	108.246	10244.42631
4.4	119.815	15874.68166
4.5	132.602	24826.65356
4.6	146.734	39193.38065
4.7	162.350	62467.66988
4.8	179.607	100530.3548
4.9	219.742	163371.9285
5.0	219.742	268117.544

GRAPH:



(Graph 5.3.1)



(Graph 5.3.2)

CHAPTER 6

SOLVING THE RUNGE-KUTTA METHOD USING MATLAB CODING

6.1 INTRODUCTION

Runge–Kutta method is an effective and widely used method for solving the initial-value problems of differential equations. Runge–Kutta method can be used to construct high order accurate numerical method by functions' self without needing the high order derivatives of functions. It attempts to overcome the problem of the Euler's method, as far as the choice of a sufficiently small step size is concerned, to reach a reasonable accuracy in the problem resolution. On the other hand, the minus of the Runge-Kutta method is that we need to calculate the increments k_i involved in the numerical algorithm, enlarging the utilized space of the Excel worksheet, generating a loss in efficiency, especially when, like in the Optimal Control theory, more variables are involved. Just as the trapezoid rule provides an improvement over the rectangular method for approximating integrals, so Simpson's rule gives an even better means for approximating integrals. With Simpson's rule we approximate not by rectangles or trapezoids but by parabolas.

When we apply it to the integral of f , we find that

$$\int_{x_1}^{x_2} f(x,y)dx = 1/6 [f(x_0,y_0) + 4f(x_{1/2}, y(x_{1/2})) + f(x_1, y(x_1))]$$

Here $x_{1/2} = x_0 + h/2$, the midpoint of x_0 and x_1 . We cannot provide all the rigorous details of the derivation of the fourth-order Runge-Kutta method. We instead provide an intuitive development.

we must now estimate both $y_{1/2}$ and y_1 . The first estimate of $y_{1/2}$ comes from Euler's method. Thus

$$Y_{1/2} = y_0 + m_1/2$$

Here

$$m_1 = h \cdot f(x_0, Y_0).$$

To correct the estimate of $y_{1/2}$, we calculate it again in this manner:

$$Y_{1/2} = y_0 + m_2 / 2$$

Where,

$$m_2 = h \cdot f(x_0 + h/2, y_0 + m_1/2)$$

Now, to predict y_1 , we use the expression for $y_{1/2}$ and the Euler method:

$$y_1 = y_{1/2} + m_3/2,$$

where

$$m_3 = h \cdot f(x_0 + h/2, y_0 + m_2/2)$$

Finally, let $m_4 = h \cdot f(x_0 + h, y_0 + m_3)$. The Runge-Kutta scheme is then obtained by substituting each of these estimates into Equation to yield

$$y_1 = y_0 + 1/6 (m_1 + 2m_2 + 2m_3 + m_4)$$

This algorithm can be applied to any number of mesh points in a natural way. At each step of the iteration, we first compute the four numbers m_1, m_2, m_3, m_4 given by

$$m_1 = h \cdot f(x_k, y_k)$$

$$m_2 = h \cdot f(x_k + h/2, y_k + m_1/2)$$

$$m_3 = h \cdot f(x_k + h/2, y_k + m_2/2)$$

$$m_4 = h \cdot f(x_k + h, y_k + m_3)$$

Then y_{k+1} is given by, $y_{k+1} = y_k + 1/6(m_1 + 2m_2 + 2m_3 + m_4)$

This new analytic paradigm, the Runge-Kutta technique, is capable of giving extremely accurate results without the need for taking very small values of h .

The local truncation error is

$$\epsilon_k = - y^{(5)}(\xi) \cdot h^5/180$$

where ξ is a point between x_0 and x_n . The total truncation error is thus of the order of magnitude of h^4 .

6.2 NUMERICAL EXAMPLE

Apply the Runge-Kutta method to the differential equation

$$y' = x + y, \quad y(0) = 1.$$

Take $h = 0.1$, so that the process has only a single step.

SOLUTION:

We determine that

$$m_1 = 1 \cdot (0 + 1) = 1,$$

$$m_2 = 1 \cdot (0 + 0.5 + 1 + 0.5) = 2,$$

$$m_3 = 1 \cdot (0 + 0.5 + 1 + 1) = 2.5,$$

$$m_4 = 1 \cdot (0 + 1 + 1 + 2.5) = 4.5$$

Thus,

$$y_1 = 1 + 1/6(1 + 4 + 5 + 4.5) = 3.417$$

Observe that this approximate solution is even better than that obtained with the improved Euler method for $h = 0.2$. And the amount of computation involved was absolutely minimal

Table 6.1 Tabulated Values for Exact and Numerical Solutions to Equation with $h = 0.2$

x_n	y_n	Exact	$\overline{E}_n(\%)$
0.0	1.00000	1.00000	0.00000
0.2	1.24280	1.24281	0.00044
0.4	1.58364	1.58365	0.00085
0.6	2.04421	2.04424	0.00125
0.8	2.65104	2.65108	0.00152
1.0	3.43650	3.43656	0.00179

the result of applying Runge-Kutta to our differential equation with $h = 0.2$. Notice that our approximate value for $y(1)$ is 3.43650, which agrees with the exact value to four decimal places. The relative error is less than 0.002 percent.

Table 6.2 Tabulated Values for Exact and Numerical Solutions to Equation with $h = 0.1$

x_n	y_n	Exact	$\overline{E}_n$ (%)
0.0	1.00000	1.00000	0.0
0.1	1.11034	1.11034	0.00002
0.2	1.24281	1.24281	0.00003
0.3	1.39972	1.39972	0.00004
0.4	1.58365	1.58365	0.00006
0.5	1.79744	1.79744	0.00007
0.6	2.04424	2.04424	0.00008
0.7	2.32750	2.32751	0.00009
0.8	2.65108	2.65108	0.00010
0.9	3.01920	3.01921	0.00011
1.0	3.43656	3.43656	0.00012

So, If we cut the step size in half-to 0.1 then the accuracy is increased dramatically. Now the relative error is less than 0.0002 percent.

6.3 CODING FOR COMPUTING RUNGE-KUTTA METHOD:

```
% Prompt user to input step size and differential equation
h = input('Enter the step size (h): ');
eqn = input('Enter the differential equation (dy/dx): ','s');

% Convert input equation string to function handle
f = str2func(['@(x,y) ', eqn]);

% Prompt user to input initial condition
x0 = input('Enter the initial value of x: ');
y0 = input('Enter the initial value of y: ');

% Prompt user to input final value of x for which y is to be calculated
xf = input('Enter the final value of x: ');

% Initialize arrays to store the numerical solutions and errors
x = x0:h:xf;
y = zeros(size(x));
error = zeros(size(x));

% Set the initial value
y(1) = y0;

% Compute the numerical solution and error for each step
for i = 1:length(x)-1
    k1 = h*f(x(i), y(i));
    k2 = h*f(x(i) + h/2, y(i) + k1/2);
    k3 = h*f(x(i) + h/2, y(i) + k2/2);
    k4 = h*f(x(i) + h, y(i) + k3);
```

```

y(i+1) = y(i) + (k1 + 2*k2 + 2*k3 + k4)/6;
exact_sol = exp(x(i+1)) - x(i+1) - 1;
error(i+1) = abs(exact_sol - y(i+1));
end

% Plot the numerical solution and exact solution
plot(x, y, '-o', x, exp(x) - x - 1, 'LineWidth', 1.5);
legend('Numerical Solution', 'Exact Solution');
xlabel('x');
ylabel('y');
title('Numerical and Exact Solutions using Runge-Kutta Method');

% Plot the error at each step
figure;
plot(x, error, '-o', 'LineWidth', 1.5);
xlabel('x');
ylabel('Error');
title('Error in Numerical Solution using Runge-Kutta Method');

disp(['x: ', num2str(x)]);
disp(['y: ', num2str(y)]);
disp(['Error: ', num2str(error)]);

```

OUTPUT:

```

>> Rungekuttamethod
Enter the step size (h):
0.1
Enter the differential equation (dy/dx): y-
sin(x)+1
Enter the initial value of x:
0
Enter the initial value of y:

```

1

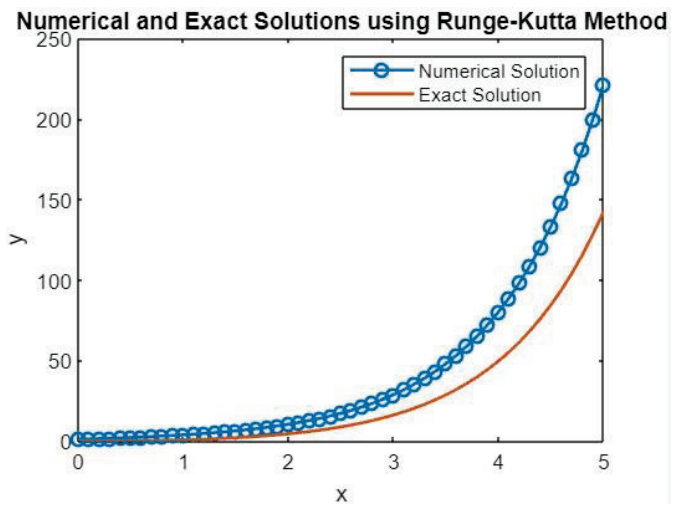
Enter the final value of

x: 5

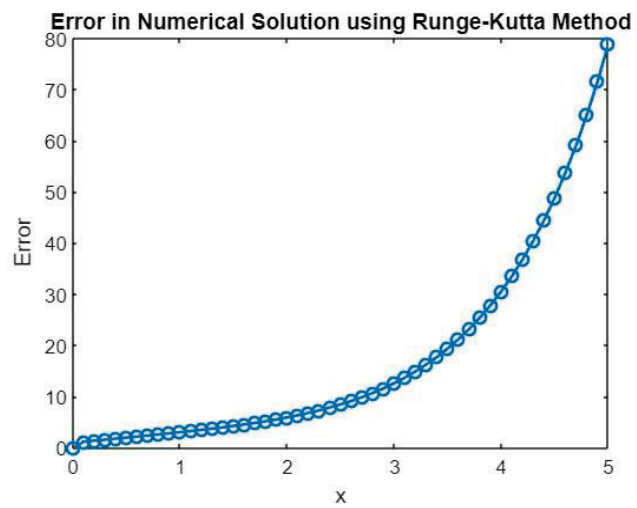
x	y	Error
0	1	0
0.1	1.205175	1.2
0.2	1.421472	1.40007
0.3	1.650216	1.60036
0.4	1.892976	1.80115
0.5	2.151585	2.00286
0.6	2.428166	2.20605
0.7	2.725158	2.41141
0.8	3.045341	2.6198
0.9	3.391871	2.83227
1	3.768307	3.05003
1.1	4.178648	3.27448
1.2	4.62737	3.50725
1.3	5.119469	3.75017
1.4	5.660503	4.0053
1.5	6.256644	4.27495
1.6	6.914728	4.5617
1.7	7.642323	4.86838
1.8	8.447784	5.19814
1.9	9.340335	5.55444
2	10.33015	5.94109
2.1	11.42842	6.36225
2.2	12.6475	6.82249
2.3	14.00097	7.32678

2.4	15.50377	7.8806
2.5	17.17238	8.48988
2.6	19.02488	9.16114
2.7	21.08121	9.90148
2.8	23.36331	10.7187
2.9	25.89531	11.6212
3	28.70381	12.6183
3.1	31.81808	13.7201
3.2	35.27038	14.9379
3.3	39.09625	16.2836
3.4	43.33488	17.7708
3.5	48.02944	19.414
3.6	53.22758	21.2293
3.7	58.98183	23.2345
3.8	65.35019	25.449
3.9	72.39662	27.8942
4	80.19177	30.5936
4.1	88.81362	33.5733
4.2	98.34828	36.862
4.3	108.8909	40.4911
4.4	120.5465	44.4956
4.5	133.4311	48.914
4.6	147.6731	53.7888
4.7	163.4141	59.1669
4.8	180.8107	65.1003
4.9	200.036	71.6462
5	221.2813	78.8682

GRAPH:



(Graph 6.3.1)



(Graph 6.3.2)

CHAPTER 7

SUMMARY AND CONCLUSION

SUMMARY

A Brief introduction to Error Approximation, Numerical solution of differential equation and error types were discussed in 1st chapter. In second chapter, Literature review of Numerical methods for differential equation and In third chapter, Aim and scope of our investigation were introduced. In Fourth chapter Method of Euler, In Fifth chapter an Improved Euler method ,In sixth chapter Runge-Kutta method were discussed by using matlab coding.

CONCLUSION

A comparative study might involve implementing each of these methods for a set of test problems and comparing their accuracy, efficiency, and stability. The comparative study on numerical methods for differential equations provides a comprehensive analysis of various numerical techniques used to solve differential equations. From the study, it is evident that different numerical methods have varying levels of accuracy, stability, and computational efficiency. For instance, Euler's method is simple to implement but has low accuracy and stability, while higher-order methods such as the Runge-Kutta method offer better accuracy and stability but require more computational resources. Overall, the study provides useful insights into the selection of the most appropriate numerical method for solving differential equation problems with matlab.

REFERENCES

- A. C. King, J. Billingham, and S. R. Otto, Differential Equations: Linear, Nonlinear, Ordinary, Partial, Cambridge: Cambridge Univ. Press, 2003.
- Akanbi, M.A, Propagation of Errors in Euler Method, Scholars Research Library, 2010.
- F. B. Hildebrand, Introduction to Numerical Analysis, New York: Dover, 1987.
- George F. Simmons Colorado College and Steven G. Krantz Washington University in St. Louis, Differential Equations Theory, Technique, and Practice , 2006
- Hossain, Md.S., Bhattacharjee, P.K. and Hossain, Md.E. Numerical Analysis, 2013
- M.K Jain, S.R.K Iyengar, R.K Jain, Numerical methods for scientific and engineering computing ,1985.
- L. G. Ixaru, Numerical Methods for Differential Equations and Applications, Dordrecht, Holland: D. Reidel Publishing Co., 1984.
- Ogunrinde, R.B, Fadugba, S.E and Okunlola, J.T On Some Numerical Methods for Solving Initial Value Problems in Ordinary Differential Equations, 2012.