BUSITEMA UNIVERSITY
Pursuing excellence

FACULTY OF ENGINEERING AND TECHNOLOGY

A REPORT ON CONTRUCTION OF ALGORITHMS TO SOLVE VARIOUS NUMERICAL MEHOD PROBLEMS IN MATLAB

BY GROUP 19

COURSE UNIT: COMPUTOR PROGRAMING

LECTURER: Mr. MASERUKA BENEDICTO

ACKNOWLEDGEMENT

First and foremost, we would like to thank the Almighty God for giving us the strength to carry on with our research in Group 19. We would love to extend our gratitude to all the persons with whose help we managed to make it this far. The willingness of each one of us to invest time and provide constructive feedback has been immensely valuable in this assignment. We wish to extend our gratitude to our lecturer for his consistent guidance and valuable insights throughout this assignment. His teaching and encouragement made it possible for us to understand and practically apply concepts of data importing, organization, and storage in MATLAB.

We also thank our group members for their cooperation and contribution. Each member actively participated in research, coding, and report writing, which ensured the success of this work. Finally, we would like to express our gratitude to all the sources and references that have been cited in this report

ABSTRACT

We started our first meeting for research on 26th, September, 2025 in the university library out of which we were exposed to various concepts and applied a variety of data programming techniques. These techniques were applied together with our knowledge from numerical methods to make algorithms that can automatically find solutions to said numerical problems if given enough data to work with.

DEDICATION

We dedicate this report to all the individuals especially Group 19 members, who have been there with us in the process of formulating and compiling this report. To our lecturer Mr. Maseruka Benedicto whose guidance and expertise have been invaluable, your mentorship and insightful feedback have shaped our understanding.

DECLARATION

We hereby certify and confirm that the information in this report is out of our own efforts, research and it has never been submitted in any institution for any academic award.

STUDENT NAME	REG.NO	COURSE	SIGNATURE
NTALE JOASH KATEREGA	BU/UG/2024/2595	WAR	
WANYAMA JOSEPH EROGO	BU/UP/2024/1077	WAR	
KAKULU ALFRED	BU/UP/2024/0981	MEB	
OKORI DARIOUS	BU/UP/2024/1061	WAR	
ALITEMA VICTOR	BU/UP/2024/5098	WAR	
AMASO SUSAN	BU/UP/2024/5435	PTI	
MAATE PHILEMON	BU/UP/2024/0830	AMI	
SIKUKU BELIZER RUTH	BU/UP/2024/0846	AMI	
TWICHIRIZE FLORENCE	BU/UP/2022/1836	WAR	
OKWI NICHOLAS	BU/UP/2024/4457	WAR	

APPROVAL

We are presenting this report which has been written and produced under our efforts. We carried out research on visualizing our data into plots and graphs that are well labeled ready for easy interpretation by the final user.

DATE OF SUBMISSION: …………………….

SIGNATURE: …………………………….

Table Of Contents

List OF ACRONYMS/ABBREVIATIONS.

MATLAB –Matrix Laboratory.

GUI – Graphics user interface

1 CHAPTER 1: INTRODUCTION

1.1 Background

Matrix Laboratory, or just MATLAB, is a high-speed programming language and computational environment employed in engineering. It was first developed in the late 1970s by computer science professor, Cleve Moler, who desired to provide his students with access to sets of mathematical software without their having to learn to program in Fortran themselves.

1.2 Historical Development

Early Development: The initial release of MATLAB, in the latter 1970s, as an interactive matrix calculator, was in Fortran. It consisted of rudimentary matrix operations and was built upon two early numerical libraries, LINPACK, for linear algebra computations, and EISPACK, to solve eigenvalue problems.

Commercialization: The program entered commercial status in 1984, when Moler, in conjunction with Jack Little and Steve Bangert, began MathWorks. This release marked an extensive revision, as it was fully implemented in C and considerably increased in features, including user-defined functions, toolboxes, and graphical user interfaces, significantly broadening the ways in which it could be utilized.

Expansion through Toolboxes: Until the late 1980s, MATLAB had expanded considerably beyond its original limits. The introduction of toolboxes enabled having specialist applications in signal processing and control systems, and others. At this point, MathWorks also added Simulink, which also became a graphical environment to model and simulate in a dynamic state system.

Recent Advances: Since its past updates, MATLAB has also evolved to meet researchers', engineers', and educators' needs as it advances in this direction. New versions have added capabilities, including the Live Editor, which supports combining code, visualizations, and descriptive text in interactive documents. These advances demonstrate how MATLAB has been evolving to become an adaptable infrastructure supporting both research in academics and industrial practice.

2 CHAPTER 2: STUDY METHODOLOGY

2.1 Introduction

It is one of the core competencies of engineering and data science to create and manipulate algorithms to solve functions and equations. MATLAB provides a suite of functions and loops through which equations can be manipulated to find solutions using numerical method.

For the first part of this assignment, we were provided equations that required roots or solutions. We were to implement the use of function handles, while, if and for loops to continue the working on the equations in various methods i.e.; Newton Raphson method, Bisection method, Secant Method and Fixed iteration method.

The objectives were:

- To calculate roots of the equations using different methods.
- To apply the algorithms on real world problems that require solutions.
- To compare the average time taken for each method.
- To visualize the time taken for each method on graphs and using the information to see the most suitable method.

In part two, we utilized the knowledge obtained from module 1 to 5.1 to generate algorithms to solve ordinary differential equations using numerical methods like Eulers, Heuns and Runge Kutta methods. We would apply the same knowledge from the part one of the assignment here.

The Objectives were to:

- To solve Ordinary Differential Equations using different methods.
- To apply the algorithms on real world problems that require solutions.
- To compare the average time taken for each method.
- To visualize the time taken for each method on graphs and using the information to see the most suitable method.

This project made us acquainted with MATLAB’s potential in algorithm analysis and data visualization. The project showed us how algorithms can be used to solve problems and equations whether multivariable or single variable and displayed for best choice. These skills provide an avenue for future works in engineering, research, and decision-making activities.

2.2 Design Process

1. As a group we decided to review all our methods of solving equations using numerical approximations.
2. We went through different ideologies, flow charts and pseudocodes. That we would then apply into our final scripts.
3. We organized several meetings during our available time where we went through lecture notes and modules to come up with possible lines of code to put in our script.

4. We inquired from other groups about their progress and refined some of ideas from them.
5. The code for both numbers was written down.
6. Under Visualization plots were created to highlight the time taken difference.
7. Debugging was done in the presence of all members that were available to get a better understanding of how it worked.
8. Documentations was carried out in report making and presentation drafting.

3 CHAPTER 3: METHODOLOGY

3.1 Part A

Solving Roots of Functions Using Numerical methods

We shall solve the given equations using various numerical methods for finding roots of functions.

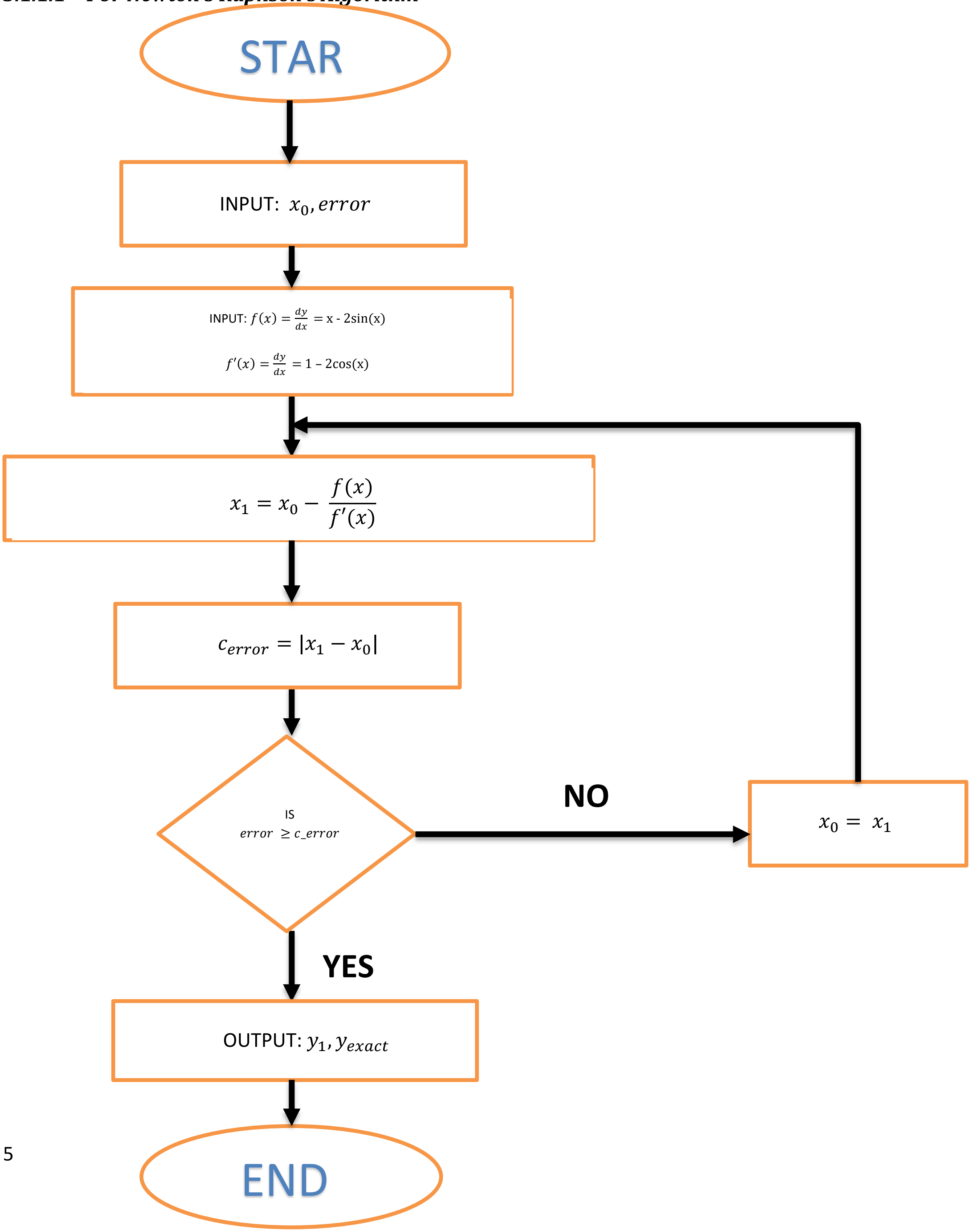
These include the Newton–Raphson Method, the Secant Method, the Bisection Method, and the Fixed-Point Iteration Method.

The work was done by constructing flowcharts, pseudocodes, and then algorithms for each of the methods. Using the constructed algorithms, we solved different equations and compared the methods.

We applied the algorithms on two test equations followed by one practical real-world application while also recording the time taken for each solution. This enabled us to plot graphs showing the accuracy of the solutions against the time required for computation.

3.1.1 FLOW CHARTS

3.1.1.1 For Newton's Raphson's Algorithm


STAR
INPUT: $x_0, error$
INPUT: $f(x) = \frac{dy}{dx}$ = x - 2sin(x)
$f'(x) = \frac{dy}{dx}$ = 1 – 2cos(x)
$x_1 = x_0 - \frac{f(x)}{f'(x)}$
$c_{error} = |x_1 - x_0|$
IS
$error \geq c\_error$
NO
$x_0 = x_1$
YES
OUTPUT: y_1, y_{exact}
END

3.1.2 PSEUDOCODE

3.1.2.1 Using Newton Raphson Formula

```
1. Initialize:
       • Check how many d.p are required
       • initial approximation says, $x = 5$
       • state the error(tolerance)
2. Compute the function at initial approximation:
   $f(x) = x - 2sin(x)$
3. Compute derivative: $f'(x) = 1 - 2cos(x)$
4. Compute predicted next value from $x_1 = x_0 + f(x_0)/f'(x_0)$
5. Compute the calculated error $c_{error} = |x_1 - x_0|$
6. While $c_{error} > error$:
     • Update $x_0 = x_1$
     • Compute next value $x_1$
     • Update new $c_{error}$
7. Output stored final results in table and computation time.
```

3.1.3 Number One

Use the Different algorithms with an initial approximation of $x = 2$ to find an approximate solution of one of the roots of the equation correct to 5((d.p);

$$f(x) = x - 2\sin(x)$$

3.1.3.1 Using Newton Raphson's Formulae method

```
X = 2; %Initial Approximation
ERR = 0.00005; %Error(depends on d.p required)
F = @(X) X - 2*sin(X); %FUNCTION
F_D = @(X) 1 - 2*cos(X); %functions derivative
X1 = X - (F(X)/F_D(X)); %Finding the next root
R_ERROR = abs(X1 - X);
%%Using a while loop to repeat for n iterations
while R_ERROR > ERR
    X = X1;
    X1 = X - (F(X)/F_D(X))
    R_ERROR = abs(X1 - X);
end
```

```
X1 = 1.8955

X1 = 1.8955
```

```
disp(["Final answer is:", num2str(X1)])
```

```
    "Final answer is:"    "1.8955"
```

3.1.3.2 Using Secant Method

```
x0 = 2; %Initial Approximation
x1 = 1.9; %Next Approximation
err = 0.00005;
F = @(x) x - 2*sin(x);
T_ERROR = abs(x1 - x0);
while T_ERROR > err
    x2 = x1 - (F(x1)*(x1 - x0)) / (F(x1) - F(x0))
    T_ERROR = abs(x2 - x1);
    x0 = x1;
    x1 = x2;
end
```

```
x2 = 1.8957

x2 = 1.8955

x2 = 1.8955
```

```
disp(["The Final answer is" num2str(x2)])
```

```
  "The Final answer is"    "1.8955"
```

3.1.3.3 Bisection Method

```
a = 1; % Lower boundary
b = 3; % Upper boundary
ERR = 0.00005;

F = @(x) x - 2*sin(x);
%Checking for existance of root
if F(a)*F(b) > 0
    error('No root in the interval')
end

c = (a+b)/2; %Bisecting to find the next root
R_ERROR = abs(b-a);

%Using a while loop to repeat for many iterations
while R_ERROR > ERR
    if F(a)*F(c) < 0
        b = c;
    else
        a = c;
    end
    c_new = (a+b)/2;
    R_ERROR = abs(c_new - c);
    c = c_new
end
```

```
c = 1.5000

c = 1.7500

c = 1.8750

c = 1.9375

c = 1.9062

c = 1.8906

c = 1.8984

c = 1.8945

c = 1.8965

c = 1.8955
```

```
c = 1.8950

c = 1.8953

c = 1.8954

c = 1.8954

c = 1.8955
```

```
disp(["Final answer is: ", num2str(c)])
```

```
    "Final answer is: "    "1.8955"
```

3.1.3.4 Fixed Point Iteration

```
x0 = 2;%Initial Approximation
ERR = 0.00005;

g = @(x) 2*sin(x); %Rearraneged f(x), g(x) used to find x

x1 = g(x0);
R_ERROR = abs(x1 - x0);

while R_ERROR > ERR
    x0 = x1;
    x1 = g(x0)
    R_ERROR = abs(x1 - x0);
end
```

```
x1 = 1.9389

x1 = 1.8660

x1 = 1.9135

x1 = 1.8837

x1 = 1.9029

x1 = 1.8907

x1 = 1.8985

x1 = 1.8936

x1 = 1.8967

x1 = 1.8947

x1 = 1.8960
```

```
x1 = 1.8952

x1 = 1.8957

x1 = 1.8954

x1 = 1.8956

x1 = 1.8954

x1 = 1.8955

x1 = 1.8955

x1 = 1.8955
```

```
disp(["Final answer is: ", num2str(x1)])
```

```
    "Final answer is: "    "1.8955"
```

3.1.4 Number Two

Use the Different algorithms with an initial approximation of $x = 3$ to find an approximate solution of one of the roots of the equation correct to 5((d.p);

$$f(x) = x^3 - 6x^2 + 11x - 6.1$$

3.1.4.1 Newton-Raphson Method

```
X = 3.5;
ERR = 0.00005;

F = @(X) X^3 - 6*X^2 + 11*X - 6.1;
F_D = @(X) 3*X^2 - 12*X + 11;

X1 = X - F(X)/F_D(X);
R_ERROR = abs(X1 - X);

while R_ERROR > ERR
    X = X1;
    X1 = X - F(X)/F_D(X);
    R_ERROR = abs(X1 - X);
end

disp(["Final answer is: ", num2str(X1)])
```

```
    "Final answer is: "    "3.0467"
```

3.1.4.2 Secant Method

```
x0 = 3;
x1 = 2.8;
err = 0.00005;

F = @(x) x^3 - 6*x^2 + 11*x - 6.1;
T_ERROR = abs(x1 - x0);

while T_ERROR > err
    x2 = x1 - (F(x1)*(x1 - x0)) / (F(x1) - F(x0));
    T_ERROR = abs(x2 - x1);
    x0 = x1;
    x1 = x2;
end

disp(["Final answer is: ", num2str(x2)])
```

```
    "Final answer is: "    "3.0467"
```

3.1.4.3 Bisection Method

```
a = 2.5;
b = 3.5;
ERR = 0.00005;

F = @(x) x^3 - 6*x^2 + 11*x - 6.1;

if F(a)*F(b) > 0
    error('No root in the interval')
end

c = (a+b)/2;
R_ERROR = abs(b-a);

while R_ERROR > ERR
    if F(a)*F(c) < 0
        b = c;
    else
        a = c;
    end
    c_new = (a+b)/2;
    R_ERROR = abs(c_new - c);
    c = c_new;
end
```

```
disp(["Final answer is: ", num2str(c)])
```

```
  "Final answer is: "    "3.0467"
```

3.1.4.4 Fixed Point Iteration

```
x0 = 3;
ERR = 0.00005;

g = @(x) (6.1 + 6*x^2 - x^3)/11;   % Rearranged form x = g(x)

x1 = g(x0);
R_ERROR = abs(x1 - x0);

while R_ERROR > ERR
    x0 = x1;
    x1 = g(x0);
    R_ERROR = abs(x1 - x0);
end

disp(["Final answer is: ", num2str(x1)])
```

```
  "Final answer is: "    "3.0465"
```

3.1.5 Real World Application

A projectile is fired with initial speed $v_0 = 50m$ at an angle θ with the horizontal. We want the projectile to hit a target at horizontal distance R=200m. Neglect air resistance.
The Formula for Horizontal Range is;

$$R(\theta) = \frac{{v_0}^2 \sin(2\theta)}{g} = 200$$

where:

- $\boldsymbol{g} = 9.81\, m/s^2$
- $\boldsymbol{v_0} = \mathbf{50}\, m/s$
- $\boldsymbol{R} = \mathbf{200}\, m$

Find the angle of projection for these initial conditions.

Use;

$$f(\theta) = \frac{{v_0}^2 \sin(2\theta)}{g} - R(\theta)\ as\ the\ function$$

3.1.5.1 Newton-Raphson Method (Projectile Motion)

```
tic
theta = 0.5;
ERR = 5e-6;
%Substituting given data from the question
v0 = 50; g = 9.81; R = 200;

F = @(theta) (v0^2 * sin(2*theta))/g - R;
F_D = @(theta) (2*v0^2 * cos(2*theta))/g;

theta1 = theta - F(theta)/F_D(theta);
R_ERROR = abs(theta1 - theta);

while R_ERROR > ERR
    theta = theta1;
    theta1 = theta - F(theta)/F_D(theta)
    R_ERROR = abs(theta1 - theta);
end
```

```
theta1 = 0.4512

theta1 = 0.4512

theta1 = 0.4512
```

```
disp(["Newton-Raphson: Launch angle = ", num2str(rad2deg(theta1))])
```

```
    "Newton-Raphson: Launch angle = "    "25.8511"
```

```
timetaken_NR_ = toc
```

```
timetaken_NR_ = 0.1056
```

3.1.5.2 Secant Method (Projectile Motion)

```
tic
theta0 = 0.4;
theta1 = 0.6;
err = 5e-6;
%Substituting given data from the question
F = @(theta) (v0^2 * sin(2*theta))/g - R;
T_ERROR = abs(theta1 - theta0);

while T_ERROR > err
    theta2 = theta1 - (F(theta1)*(theta1 - theta0)) / (F(theta1) - F(theta0))
    T_ERROR = abs(theta2 - theta1);
    theta0 = theta1;
    theta1 = theta2;
end
```

```
theta2 = 0.4628

theta2 = 0.4482

theta2 = 0.4512

theta2 = 0.4512

theta2 = 0.4512
```

```
disp(["Secant Method: Launch angle = ", num2str(rad2deg(theta2))])
```

```
    "Secant Method: Launch angle = "    "25.8511"
```

```
time_taken_Secant = toc
```

```
time_taken_Secant = 0.1479
```

3.1.5.3 Bisection Method (Projectile Motion)

```
tic
a = 0.2;
b = 0.6;
ERR = 5e-6;
%Substituting given data from the question
v0 = 50; g = 9.81; R = 200;
F = @(theta) (v0^2 * sin(2*theta))/g - R;

if F(a)*F(b) > 0
    error('No root in the interval')
end

c = (a+b)/2;
R_ERROR = abs(b-a);

while R_ERROR > ERR
    if F(a)*F(c) < 0
        b = c;
    else
        a = c;
    end
    c_new = (a+b)/2;
    R_ERROR = abs(c_new - c);
    c = c_new;
end
```

```
disp(["Bisection Method: Launch angle = ", num2str(rad2deg(c))])
```

```
    "Bisection Method: Launch angle = "    "25.8511"
```

```
time_taken_Bisection = toc
```

```
time_taken_Bisection = 0.1379
```

3.1.5.4 Fixed Point Iteration (Projectile Motion)

```
tic
theta0 = 0.4;
ERR = 5e-6;
%Substituting given data from the question
v0 = 50; g = 9.81; R = 200;

% Rearranged form
gfun = @(theta) 0.5 * asin((R*g)/(v0^2));

theta3 = gfun(theta0);
R_ERROR = abs(theta3 - theta0);

while R_ERROR > ERR
    theta0 = theta3;
    theta3 = gfun(theta0);
    R_ERROR = abs(theta3 - theta0);
end

disp(["Fixed Point Iteration: Launch angle = ", num2str(rad2deg(theta3))])
```

```
    "Fixed Point Iteration: Launch angle = "    "25.8511"
```

```
time_taken_FPI = toc
```

```
time_taken_FPI = 0.0674
```

3.1.6 Data Analysis

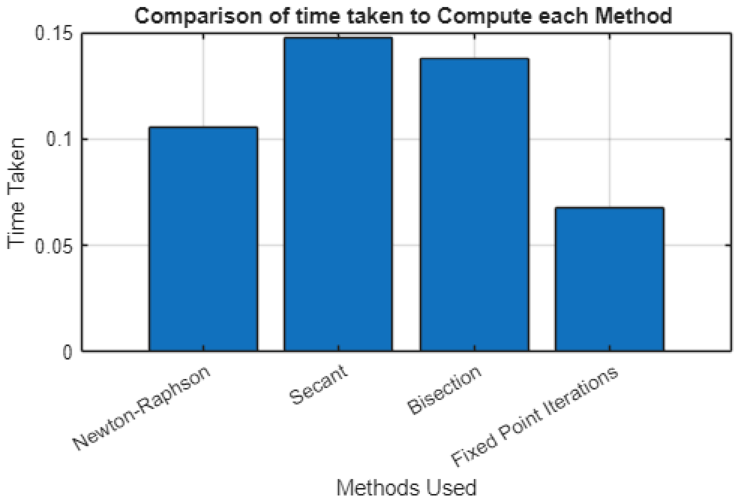
```
 Method = {'Newton-Raphson'; 'Secant'; 'Bisection'; 'Fixed Point Iterations'};
 Computational_Time = [timetaken_NR_; time_taken_Secant; time_taken_Bisection;
time_taken_FPI];
 Final_Answers = [rad2deg(theta1); rad2deg(theta2); rad2deg(c);
rad2deg(theta3)];

 Results_Table = table(Method, Final_Answers, Computational_Time)
```

```
Results_Table = 4×3 table
```

	Method	Final_Answers	Computational_Time
1	'Newton-Raphson'	25.8511	0.1056
2	'Secant'	25.8511	0.1479
3	'Bisection'	25.8511	0.1379
4	'Fixed Point Iterations'	25.8511	0.0674

```
%Plots
figure
bar(Method,Results_Table.Computational_Time)
grid on
xlabel('Methods Used');
ylabel('Time Taken');
title('Comparison of time taken to Compute each Method ');
```



3.2 Part B

Solving Ordinary Differential Equations with Numerical methods

We shall solve the Ordinary Differential Equations using various methods. These include;

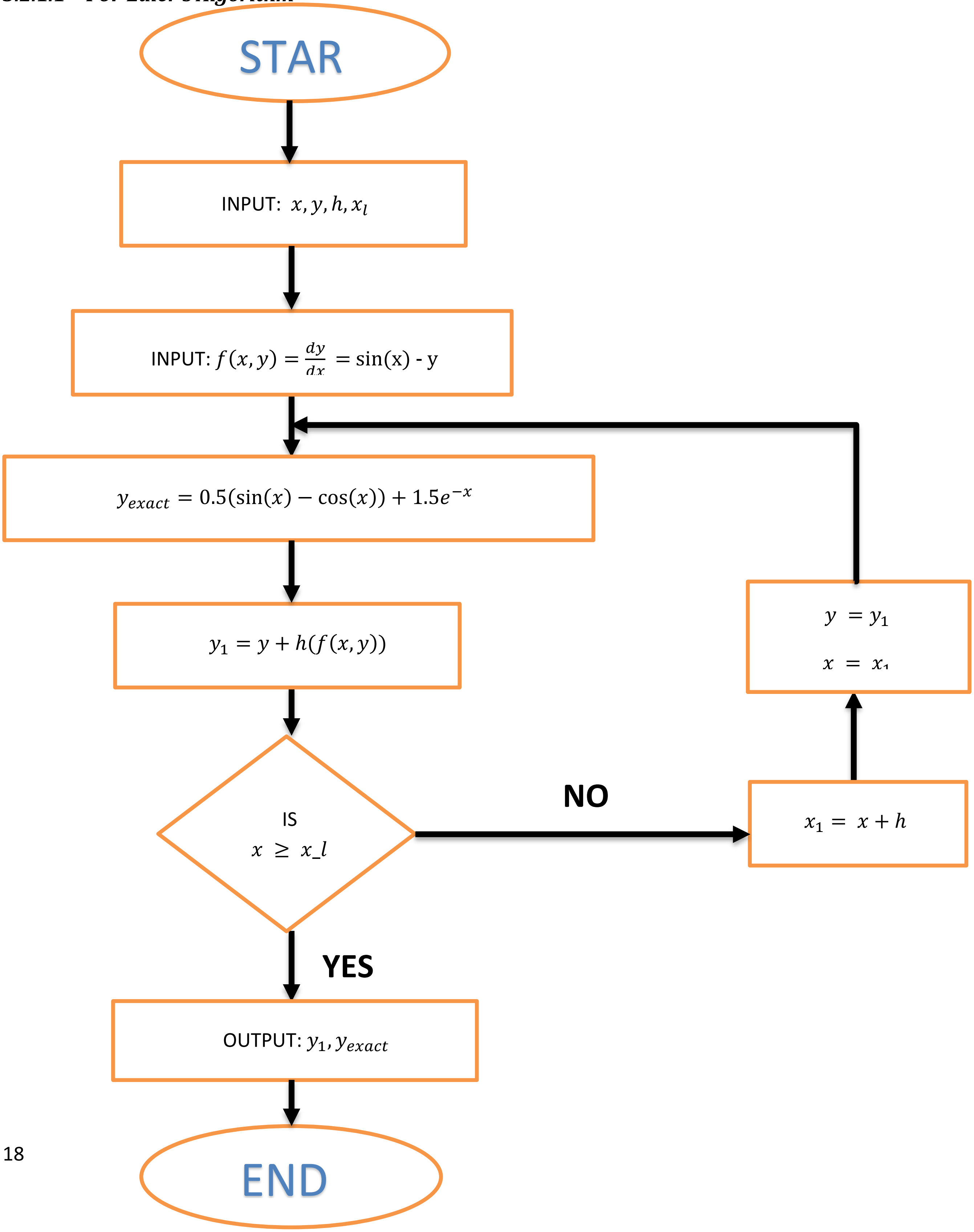
- Euler's Method
- Heun's Method
- Runge – Kutta's Method

The work was done by constructing flowcharts, pseudocodes then algorithms for each of the methods. And then using the constructed algorithms to solve the different equations.

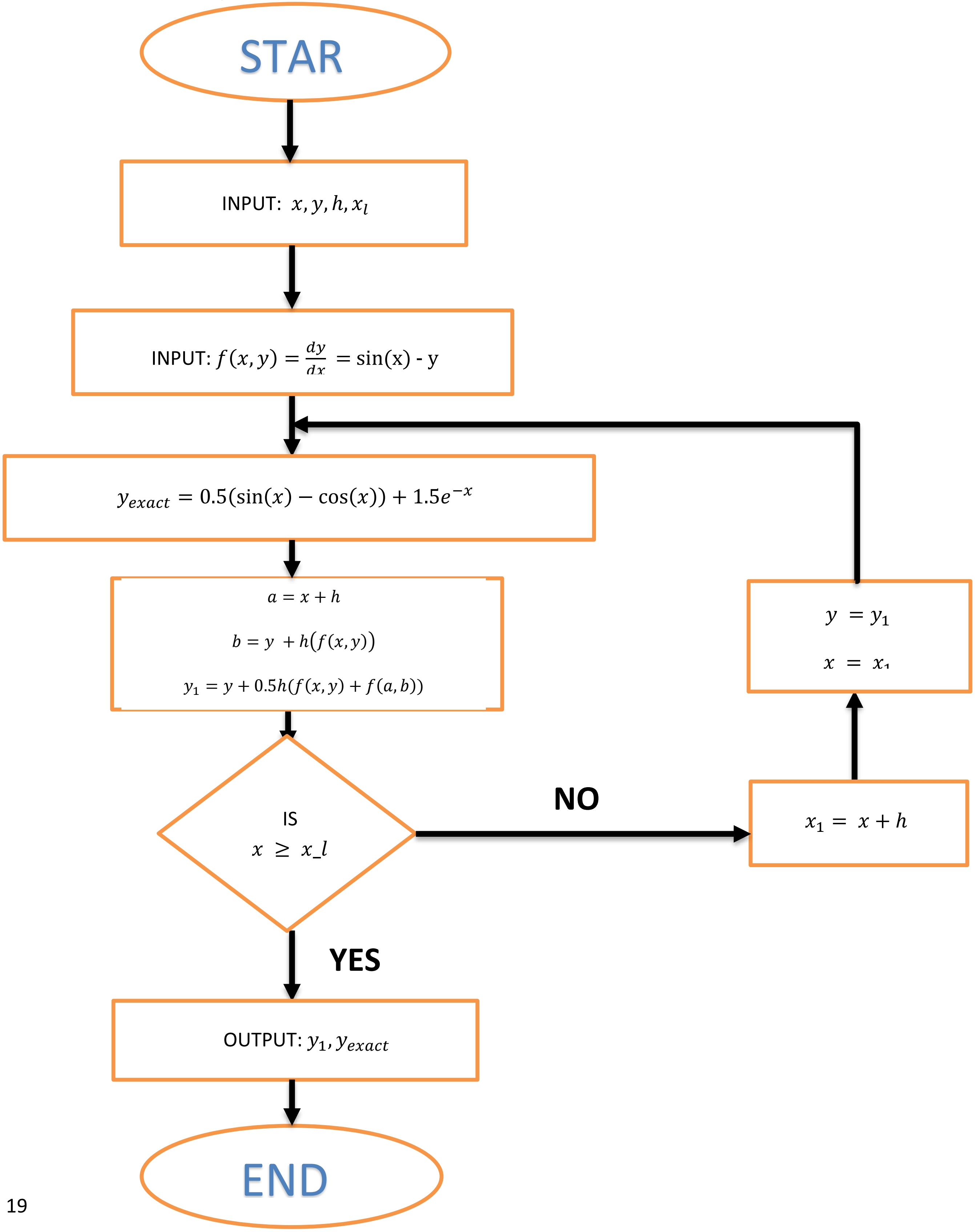
We used 2 test equations followed buy one practical real-world application while obtaining the time taken for each solution on different methods such that we could plot graphs of accuracy of the solution against time taken for it to be calculated.

3.2.1 FLOW CHARTS

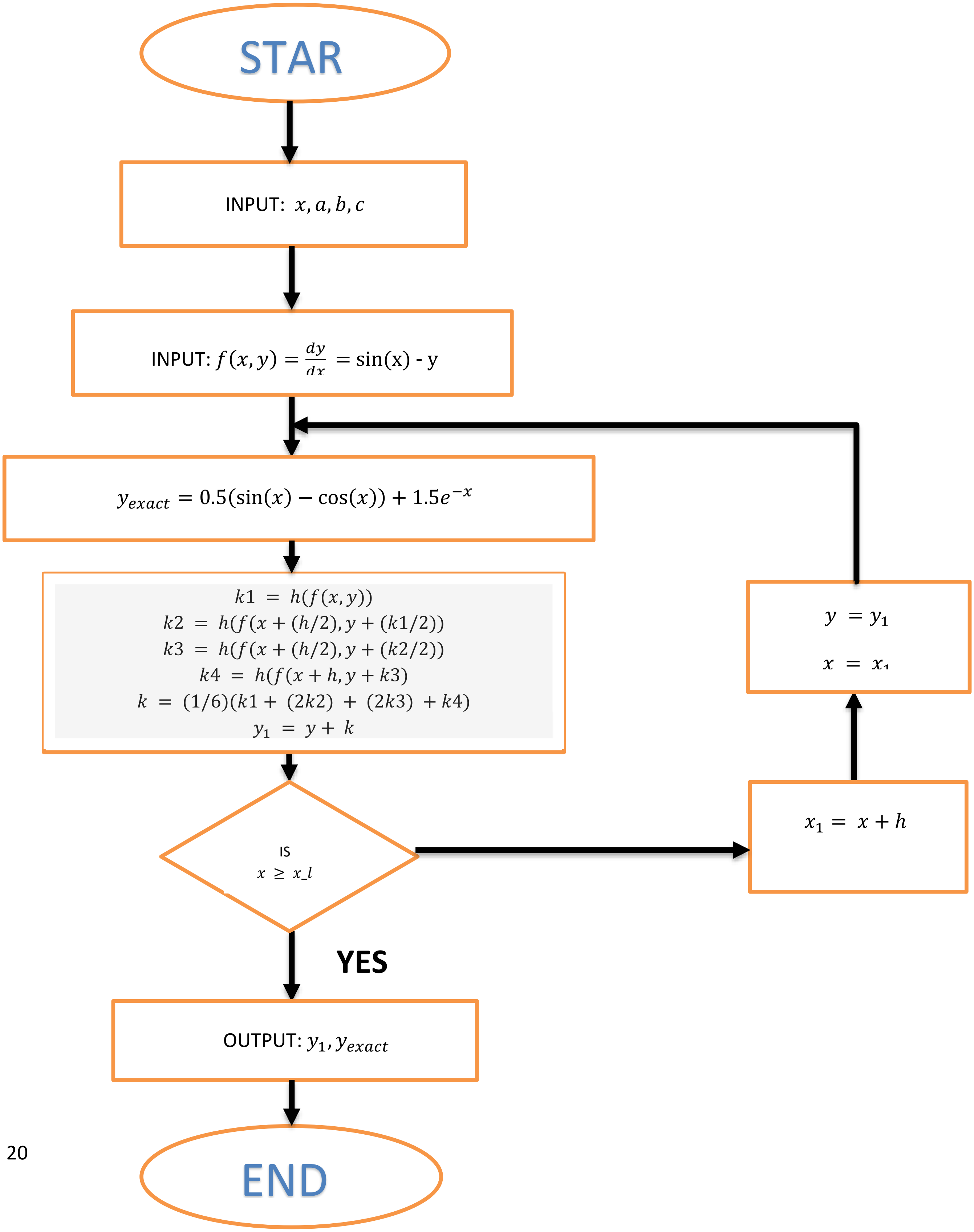
3.2.1.1 For Euler's Algorithm



3.2.1.2 For Heun's Algorithm



3.2.1.3 *For Runge Kutta's Algorithm*


STAR
INPUT: x, a, b, c
INPUT: $f(x,y) = \frac{dy}{dx}$ = sin(x) - y
$y_{exact} = 0.5(\sin(x) - \cos(x)) + 1.5e^{-x}$
$k1 = h(f(x,y))$
$k2 = h(f(x + (h/2), y + (k1/2))$
$k3 = h(f(x + (h/2), y + (k2/2))$
$k4 = h(f(x + h, y + k3)$
$k = (1/6)(k1 + (2k2) + (2k3) + k4)$
$y_1 = y + k$
IS
$x \geq x\_l$
YES
$x_1 = x + h$
$y = y_1$
$x = x_1$
OUTPUT: y_1, y_{exact}
END

3.2.2 PSEUDOCODE

3.2.2.1 Using Eulers Algorithim

```
1. Initialize:
       • step size $h = 0.2$
       • initial values $x = 0, y = 1$
       • upper limit $x_l = 1.8$
2. Compute exact solution at initial point:
   $y_{exact} = 0.5 * (sin(x) - cos(x)) + 1.5 * exp(-x)$
3. Compute derivative: $y\_d = sin(x) - y$
4. Compute predicted next value: $y_1 = y + h * y_d$
5. Store $(x, y, y_{exact}, error)$ in results.
6. While $x < x\_l$:
     • Update $x = x + h$
     • Set $y = y_1$
     • Compute derivative: $y\_d = sin(x) - y$
     • Update predicted next value: $y\_1 = y + h * y_d$
     • Compute exact solution $y_{exact}$
     • Store $(x, y, y_{exact}, error)$ in results
7. Output results table and computation time.
```

3.2.2.2 Heun's Method

```
1. Initialize:
     • step size $h = 0.2$
     • initial values $x = 0, y = 1$
     • upper limit $x_l = 1.8$
2. Compute derivative: $y_d = sin(x) - y$
3. Predict with Euler step:
     • $a = x + h$
     • $b = y + h*y_d$
     • $ab = sin(a) - b$
```

4. Compute improved next value:
   $y_1 = y + 0.5*h*(y_d + ab)$
5. Compute exact solution and store initial results $(x, y, y_{exact}, \text{error})$
6. While $x <= x_l$:

   - Update $x = x + h$
   - Set $y = y_1$
   - Compute derivative: $y_d = \sin(x) - y$
   - Predict: $a = x + h$, $b = y + h*y_d$, $ab = \sin(a) - b$
   - Correct: $y_1 = y + 0.5*h*(y_d + ab)$
   - Compute exact solution y_{exact}
   - Store $(x, y, y_{exact}, \text{error})$ in results

7. Output results table and computation time.

3.2.2.3 Runge-Kutta Method (4th Order)

1. Initialize:

   - $step\ size\ h\ =\ 0.2$
   - $range\ x\ =\ [0{:}\,h{:}\,2]$
   - $initial\ condition\ y(1)\ =\ 1$
   - $function\ f(x,y)\ =\ sin(x)\ -\ y$

2. For each step i from 1 to (length(x)-1):

   - $k1\ =\ h\ *\ f(x(i), y(i))$
   - $k2\ =\ h\ *\ f(x(i)\ +\ h/2, y(i)\ +\ k1/2)$
   - $k3\ =\ h\ *\ f(x(i)\ +\ h/2, y(i)\ +\ k2/2)$
   - $k4\ =\ h\ *\ f(x(i)\ +\ h, y(i)\ +\ k3)$
   - $k\ =\ (1/6) * (k1\ +\ 2 * k2\ +\ 2 * k3\ +\ k4)$
   - $Update\ y(i + 1)\ =\ y(i)\ +\ k$

3. Compute exact solution at all x
4. Compute $errors\ =\ abs(y\ -\ y\_exact)$
5. Output results table and computation time.

3.2.3 Number One

Use the Different algorithms with a step size $h = 0.2$ to find an approximate solution of the first order initial value problem;

$$\frac{dy}{dx} = sinx - y \qquad with \;\; y(0) = 1$$

in the interval $0 \leq x \leq 2$ and compare it with the exact solution;

$$y = \frac{1}{2}(sinx - cosx) + \frac{3}{2}e^{-x}$$

3.2.3.1Euler Method

```
tic
%DATA INPUT
x = 0;
h = 0.2;
y = 1;
x_l = 1.8;
y_exact = 0.5*(sin(x) - cos(x)) + 1.5*exp(-x);
y_d = sin(x) - y;
y_1 = y + h*y_d;

X = [];
Y = [];
Y_exact = [];
Error = [];

X(end+1,1) = x;
Y(end+1,1) = y;
Y_exact(end+1,1) = y_exact;
Error(end+1,1) = abs(y_exact - y);

%USING A WHILE LOOP TO REPEAT THE PROCESS FOR VARIOUS X VALUES
while x < x_l
    x = x + h;
    y = y_1;
    y_d = sin(x) - y;
    y_1 = y + h*y_d;
    y_exact = 0.5*(sin(x) - cos(x)) + 1.5*exp(-x);
    X(end+1,1) = x;
    Y(end+1,1) = y;
    Y_exact(end+1,1) = y_exact;
    Error(end+1,1) = abs(y_exact - y);
end
Results_Eulers1 = table(X, Y, Y_exact, Error)
```

```
Results_Eulers1 = 11×4 table
```

	X	Y	Y_exact	Error
1	0	1	1	0
2	0.2000	0.8000	0.8374	0.0374
3	0.4000	0.6797	0.7397	0.0599
4	0.6000	0.6217	0.6929	0.0712
5	0.8000	0.6103	0.6843	0.0741
6	1	0.6317	0.7024	0.0707
7	1.2000	0.6736	0.7366	0.0630
8	1.4000	0.7253	0.7776	0.0523
9	1.6000	0.7773	0.8172	0.0399
10	1.8000	0.8218	0.8485	0.0267
11	2.0000	0.8522	0.8657	0.0135

```
timetaken_Eulers1 = toc
```

```
timetaken_Eulers1 = 0.1050
```

3.2.3.2 Using Heuns Method

```
tic
%DATA INPUT
x = 0;
h = 0.2;
y = 1;
x_l = 1.8;
y_d = sin(x) - y;
```

```
a = x + h;
b = y + h*y_d;
ab = sin(a) - b;
y_exact = 0.5*(sin(x) - cos(x)) + 1.5*exp(-x);
y_1 = y + 0.5*h*(y_d + ab);

X_H = [];
Y_H = [];
Y_Exact_H = [];
Error_H = [];

X_H(end+1,1) = x;
Y_H(end+1,1) = y;
Y_Exact_H(end+1,1) = y_exact;
Error_H(end+1,1) = abs(y_exact - y);

%USING A WHILE LOOP TO REPEAT THE PROCESS FOR VARIOUS X VALUES
while x <= x_l
    x = x + h;
    y = y_1;
    y_d = sin(x) - y;
    a = x + h;
    b = y + h*y_d;
    ab = sin(a) - b;
    y_1 = y + 0.5*h*(y_d + ab);
    y_exact = 0.5*(sin(x) - cos(x)) + 1.5*exp(-x);

    X_H(end+1,1) = x;
    Y_H(end+1,1) = y;
    Y_Exact_H(end+1,1) = y_exact;
    Error_H(end+1,1) = abs(y_exact - y);
end
Results_Heuns1 = table(X_H, Y_H, Y_Exact_H, Error_H)
```

```
Results_Heuns1 = 11×4 table
```

	X_H	Y_H	Y_Exact_H	Error_H
1	0	1	1	0
2	0.2000	0.8399	0.8374	0.0025
3	0.4000	0.7435	0.7397	0.0039

	X_H	Y_H	Y_Exact_H	Error_H
4	0.6000	0.6973	0.6929	0.0044
5	0.8000	0.6887	0.6843	0.0044
6	1	0.7063	0.7024	0.0039
7	1.2000	0.7397	0.7366	0.0030
8	1.4000	0.7796	0.7776	0.0020
9	1.6000	0.8181	0.8172	8.6007e-04
10	1.8000	0.8482	0.8485	2.8758e-04
11	2.0000	0.8643	0.8657	0.0014

```
timetaken_Heuns1 = toc
```

```
timetaken_Heuns1 = 0.1167
```

3.2.3.3 Using Runge Kutta method

```
tic
%DATA INPUT
h = 0.2;
a = 0;
b = 2;
c = 1;

% list the values of x
x = (a:h:b);
y(1) = c;
y_exact = 0.5*(sin(x) - cos(x)) + 1.5*exp(-x);
func = @(x,y) sin(x) - y;

%VALUES OF K ARE FOUND USING FOR LOOP
for i=1:(length(x)-1)
    k1 = h.*func(x(i),y(i));
    k2 = h.*func(x(i)+h/2,y(i)+k1/2);
    k3 = h.*func(x(i)+h/2,y(i)+k2/2);
    k4 = h.*func(x(i)+h,y(i)+k3);
    k = (1/6).*(k1+ (2*k2) + (2*k3) +k4);
    y(i+1) = y(i)+ k;
end
%Tabulate data
y_exact = transpose(y_exact);
y1 = transpose(y);
x1 = transpose(x);
Error_R = abs(y1-y_exact);
Results_Runge1 = table(x1,y1,y_exact,Error_R)
```

```
Results_Runge1 = 11×4 table
```

	x1	y1	y_exact	Error_R
1	0	1	1	0
2	0.2000	0.8374	0.8374	4.7039e-06
3	0.4000	0.7397	0.7397	7.4596e-06
4	0.6000	0.6929	0.6929	8.7300e-06
5	0.8000	0.6843	0.6843	8.8872e-06
6	1	0.7024	0.7024	8.2337e-06
7	1.2000	0.7366	0.7366	7.0173e-06
8	1.4000	0.7776	0.7776	5.4430e-06
9	1.6000	0.8172	0.8172	3.6810e-06
10	1.8000	0.8485	0.8485	1.8725e-06
11	2	0.8657	0.8657	1.3406e-07

```
timetaken_Runge1 = toc
```

```
timetaken_Runge1 = 0.0894
```

3.2.4 Number Two

Use the Different algorithms with a step size $h = 0.1$ to find an approximate solution of the first order initial value problem;

$$\frac{dy}{dx} + 2y = \sin(3x) \qquad with\ \ y(0) = 1$$

in the interval $0 \leq x \leq 2.4$ and compare it with the exact solution;

$$y = \frac{2\sin(3x) - 3\cos(3x)}{13} + \frac{16}{13}e^{-2x}$$

3.2.4.1Euler Method

```
tic
% DATA INPUT
x = 0;
h = 0.2;
y = 1;
x_l = 2.4;

y_exact = (2*sin(3*x) - 3*cos(3*x))/13 + (16/13)*exp(-2*x);
y_d = sin(3*x) - 2*y;
y_1 = y + h*y_d;

X = [];
Y = [];
Y_exact = [];
Error = [];

X(end+1,1) = x;
Y(end+1,1) = y;
Y_exact(end+1,1) = y_exact;
Error(end+1,1) = abs(y_exact - y);

% WHILE LOOP for Euler
while x < x_l
    x = x + h;
    y = y_1;
    y_d = sin(3*x) - 2*y;
    y_1 = y + h*y_d;
    y_exact = (2*sin(3*x) - 3*cos(3*x))/13 + (16/13)*exp(-2*x);

    X(end+1,1) = x;
    Y(end+1,1) = y;
    Y_exact(end+1,1) = y_exact;
    Error(end+1,1) = abs(y_exact - y);
end
Results_Eulers_2 = table(X, Y, Y_exact, Error)
```

```
Results_Eulers_2 = 13×4 table
```

	X	Y	Y_exact	Error
1	0	1	1	0
2	0.2000	0.6000	0.7214	0.1214
3	0.4000	0.4729	0.6128	0.1399
4	0.6000	0.4702	0.5730	0.1028
5	0.8000	0.4769	0.5226	0.0457
6	1	0.4212	0.4167	0.0045
7	1.2000	0.2810	0.2505	0.0304
8	1.4000	0.0801	0.0539	0.0262
9	1.6000	-0.1263	-0.1233	0.0030
10	1.8000	-0.2750	-0.2317	0.0433
11	2.0000	-0.3196	-0.2420	0.0775
12	2.2000	-0.2476	-0.1562	0.0914
13	2.4000	-0.0863	-0.0082	0.0781

```
timetaken_Eulers_2 = toc
```

```
timetaken_Eulers_2 = 0.0722
```

3.2.4.2 Using Heuns Method

```
tic
```

```
% DATA INPUT
x = 0;
h = 0.2;
y = 1;
x_l = 2.4;

y_d = sin(3*x) - 2*y;
a = x + h;
b = y + h*y_d;
ab = sin(3*a) - 2*b;
y_exact = (2*sin(3*x) - 3*cos(3*x))/13 + (16/13)*exp(-2*x);
y_1 = y + 0.5*h*(y_d + ab);

X_H = [];
Y_H = [];
Y_Exact_H = [];
Error_H = [];

X_H(end+1,1) = x;
Y_H(end+1,1) = y;
Y_Exact_H(end+1,1) = y_exact;
Error_H(end+1,1) = abs(y_exact - y);

% WHILE LOOP for Heun
while x < x_l
    x = x + h;
    y = y_1;
    y_d = sin(3*x) - 2*y;
    a = x + h;
    b = y + h*y_d;
    ab = sin(3*a) - 2*b;
    y_1 = y + 0.5*h*(y_d + ab);
    y_exact = (2*sin(3*x) - 3*cos(3*x))/13 + (16/13)*exp(-2*x);

    X_H(end+1,1) = x;
    Y_H(end+1,1) = y;
    Y_Exact_H(end+1,1) = y_exact;
    Error_H(end+1,1) = abs(y_exact - y);
end
Results_Heuns_2 = table(X_H, Y_H, Y_Exact_H, Error_H)
```

```
Results_Heuns_2 = 13×4 table
```

	X_H	Y_H	Y_Exact_H	Error_H
1	0	1	1	0
2	0.2000	0.7365	0.7214	0.0150
3	0.4000	0.6279	0.6128	0.0151
4	0.6000	0.5803	0.5730	0.0073
5	0.8000	0.5206	0.5226	0.0020
6	1	0.4086	0.4167	0.0081
7	1.2000	0.2421	0.2505	0.0084
8	1.4000	0.0509	0.0539	0.0030
9	1.6000	-0.1173	-0.1233	0.0060
10	1.8000	-0.2168	-0.2317	0.0149
11	2.0000	-0.2217	-0.2420	0.0203
12	2.2000	-0.1364	-0.1562	0.0199
13	2.4000	0.0053	-0.0082	0.0135

```
timetaken_Heuns_2 = toc
```

```
timetaken_Heuns_2 = 0.1292
```

3.2.4.3 Using Runge Kutta method

```
tic
% DATA INPUT
h = 0.2;
a = 0;
b = 2.4;
c = 1;

% list the values of x
```

```
x = (a:h:b);
y(1) = c;
y_exact = (2*sin(3*x) - 3*cos(3*x))/13 + (16/13)*exp(-2*x);
func = @(x,y) sin(3*x) - 2*y;

% FOR LOOP for RK4
for i=1:(length(x)-1)
    k1 = h.*func(x(i),y(i));
    k2 = h.*func(x(i)+h/2,y(i)+k1/2);
    k3 = h.*func(x(i)+h/2,y(i)+k2/2);
    k4 = h.*func(x(i)+h,y(i)+k3);
    k = (1/6).*(k1+ (2*k2) + (2*k3) +k4);
    y(i+1) = y(i)+ k;
end

% Tabulate results
y_exact = transpose(y_exact);
y1 = transpose(y);
x1 = transpose(x);
Error_R = abs(y1-y_exact);
Results_Runge_2 = table(x1,y1,y_exact,Error_R)
```

```
Results_Runge_2 = 13×4 table
```

	x1	y1	y_exact	Error_R
1	0	1	1	0
2	0.2000	0.7215	0.7214	1.1633e-04
3	0.4000	0.6129	0.6128	1.3406e-04
4	0.6000	0.5731	0.5730	9.9843e-05
5	0.8000	0.5226	0.5226	4.9930e-05
6	1	0.4167	0.4167	1.0595e-05
7	1.2000	0.2505	0.2505	3.4937e-06
8	1.4000	0.0539	0.0539	9.8551e-06
9	1.6000	-0.1232	-0.1233	4.1903e-05

	x1	y1	y_exact	Error_R
10	1.8000	-0.2316	-0.2317	7.7218e-05
11	2	-0.2419	-0.2420	9.9614e-05
12	2.2000	-0.1561	-0.1562	9.8009e-05
13	2.4000	-0.0081	-0.0082	7.0327e-05

```
timetaken_Runge_2 = toc
```

```
timetaken_Runge_2 = 0.0706
```

3.2.5 Real World Application

A hot metal rod at **200 °C** is suddenly immersed in a coolant bath maintained at **25 °C**. The cooling process follows **Newton's law of cooling**, Modeled by the differential equation:

$$\frac{dT}{dt} = -k(T - T_\infty)$$

where:

- $\boldsymbol{T(t)} = temperature\ of\ the\ rod\ at\ time\ \boldsymbol{t}\ (°C),$
- $\boldsymbol{T_\infty} = \mathbf{25}\ °C\ (coolant\ temperature)$
- $\boldsymbol{k = 0.1\ min^{-1}}\ (cooling\ coefficient).$

Initial condition:

$$\boldsymbol{T(0) = 200\ °C}$$

Consider

$0 \le t \le 30$ minutes with a step size $h = 1$.

Compare your numerical solutions to the analytical solution

$$\boldsymbol{T(t) = T_\infty + (T_0 - T_\infty)e^{-kt}}$$

3.2.5.1 Using Eulers method

```
tic
% DATA INPUT
x = 0;
h = 1;
y = 200;
x_l = 30;

y_exact = 25 + (200-25)*exp(-0.1*x);
y_d = -0.1*(y - 25);
y_1 = y + h*y_d;

X = [];
Y = [];
Y_exact = [];
Error = [];
```

```
X(end+1,1) = x;
Y(end+1,1) = y;
Y_exact(end+1,1) = y_exact;
Error(end+1,1) = abs(y_exact - y);

% USING A WHILE LOOP
while x < x_l
    x = x + h;
    y = y_1;
    y_d = -0.1*(y - 25);
    y_1 = y + h*y_d;
    y_exact = 25 + (200-25)*exp(-0.1*x);

    X(end+1,1) = x;
    Y(end+1,1) = y;
    Y_exact(end+1,1) = y_exact;
    Error(end+1,1) = abs(y_exact - y);
end
Results_Eulers_Cooling = table(X, Y, Y_exact, Error)
```

```
Results_Eulers_Cooling = 31×4 table
```

	X	Y	Y_exact	Error
1	0	200	200	0
2	1	182.5000	183.3465	0.8465
3	2	166.7500	168.2779	1.5279
4	3	152.5750	154.6432	2.0682
5	4	139.8175	142.3060	2.4885
6	5	128.3357	131.1429	2.8071
7	6	118.0022	121.0420	3.0399
8	7	108.7020	111.9024	3.2005
9	8	100.3318	103.6326	3.3008
10	9	92.7986	96.1497	3.3511

	X	Y	Y_exact	Error
11	10	86.0187	89.3789	3.3602
12	11	79.9169	83.2524	3.3356
13	12	74.4252	77.7090	3.2838
14	13	69.4827	72.6931	3.2104
15	14	65.0344	68.1545	3.1201
16	15	61.0309	64.0478	3.0168
17	16	57.4279	60.3319	2.9040
18	17	54.1851	56.9696	2.7845
19	18	51.2666	53.9273	2.6607
20	19	48.6399	51.1745	2.5346
21	20	46.2759	48.6837	2.4078
22	21	44.1483	46.4299	2.2816
23	22	42.2335	44.3906	2.1571
24	23	40.5101	42.5453	2.0352
25	24	38.9591	40.8756	1.9165
26	25	37.5632	39.3649	1.8017
27	26	36.3069	37.9979	1.6910
28	27	35.1762	36.7610	1.5848
29	28	34.1586	35.6418	1.4832
30	29	33.2427	34.6291	1.3863

	X	Y	Y_exact	Error
31	30	32.4185	33.7127	1.2943

```
timetaken_Eulers_Cooling = toc
```

```
timetaken_Eulers_Cooling = 0.0687
```

```
timetaken_Eulers3 = toc
```

```
timetaken_Eulers3 = 0.0871
```

3.2.5.2 Using Heuns Method

```
tic
% DATA INPUT
x = 0;
h = 1;
y = 200;
x_l = 30;

y_d = -0.1*(y - 25);
a = x + h;
b = y + h*y_d;
ab = -0.1*(b - 25);
y_exact = 25 + (200-25)*exp(-0.1*x);
y_1 = y + 0.5*h*(y_d + ab);

X_H = [];
Y_H = [];
Y_Exact_H = [];
Error_H = [];

X_H(end+1,1) = x;
Y_H(end+1,1) = y;
Y_Exact_H(end+1,1) = y_exact;
Error_H(end+1,1) = abs(y_exact - y);

% USING A WHILE LOOP
while x < x_l
    x = x + h;
    y = y_1;
    y_d = -0.1*(y - 25);
    a = x + h;
    b = y + h*y_d;
    ab = -0.1*(b - 25);
    y_1 = y + 0.5*h*(y_d + ab);
    y_exact = 25 + (200-25)*exp(-0.1*x);

    X_H(end+1,1) = x;
    Y_H(end+1,1) = y;
    Y_Exact_H(end+1,1) = y_exact;
    Error_H(end+1,1) = abs(y_exact - y);
end
Results_Heuns_Cooling = table(X_H, Y_H, Y_Exact_H, Error_H)
```

```
Results_Heuns_Cooling = 31×4 table
```

	X_H	Y_H	Y_Exact_H	Error_H
1	0	200	200	0
2	1	183.3750	183.3465	0.0285
3	2	168.3294	168.2779	0.0515
4	3	154.7131	154.6432	0.0699
5	4	142.3903	142.3060	0.0843
6	5	131.2383	131.1429	0.0954
7	6	121.1456	121.0420	0.1036
8	7	112.0118	111.9024	0.1094
9	8	103.7457	103.6326	0.1131
10	9	96.2648	96.1497	0.1151
11	10	89.4947	89.3789	0.1158
12	11	83.3677	83.2524	0.1152
13	12	77.8227	77.7090	0.1138

	X_H	Y_H	Y_Exact_H	Error_H
14	13	72.8046	72.6931	0.1115
15	14	68.2632	68.1545	0.1087
16	15	64.1532	64.0478	0.1054
17	16	60.4336	60.3319	0.1017
18	17	57.0674	56.9696	0.0978
19	18	54.0210	53.9273	0.0937
20	19	51.2640	51.1745	0.0895
21	20	48.7689	48.6837	0.0853
22	21	46.5109	46.4299	0.0810
23	22	44.4673	44.3906	0.0768
24	23	42.6179	42.5453	0.0727
25	24	40.9442	40.8756	0.0686
26	25	39.4295	39.3649	0.0647
27	26	38.0587	37.9979	0.0609
28	27	36.8182	36.7610	0.0572
29	28	35.6954	35.6418	0.0537
30	29	34.6794	34.6291	0.0503
31	30	33.7598	33.7127	0.0471

```
timetaken_Heuns_Cooling = toc
```

```
timetaken_Heuns_Cooling = 0.0772
```

3.2.5.3 *Using Runge Kutta method*

```
tic
% DATA INPUT
h = 1;
a = 0;
b = 30;
c = 200;

% list the values of x
x = (a:h:b);
y(1) = c;
y_exact = 25 + (200-25)*exp(-0.1*x);
func = @(x,y) -0.1*(y - 25);

% VALUES OF K ARE FOUND USING FOR LOOP
for i=1:(length(x)-1)
    k1 = h.*func(x(i),y(i));
    k2 = h.*func(x(i)+h/2,y(i)+k1/2);
    k3 = h.*func(x(i)+h/2,y(i)+k2/2);
    k4 = h.*func(x(i)+h,y(i)+k3);
    k = (1/6).*(k1+ (2*k2) + (2*k3) +k4);
    y(i+1) = y(i)+ k;
end

% Tabulate data
y_exact = transpose(y_exact);
y1 = transpose(y);
x1 = transpose(x);
Error_R = abs(y1-y_exact);
Results_Runge_Cooling = table(x1,y1,y_exact,Error_R)
```

```
Results_Runge_Cooling = 31×4 table
```

	x1	y1	y_exact	Error_R
1	0	200	200	0
2	1	183.3466	183.3465	1.4344e-05
3	2	168.2779	168.2779	2.5957e-05
4	3	154.6432	154.6432	3.5231e-05

	x1	y1	y_exact	Error_R
5	4	142.3061	142.3060	4.2504e-05
6	5	131.1429	131.1429	4.8074e-05
7	6	121.0421	121.0420	5.2199e-05
8	7	111.9025	111.9024	5.5104e-05
9	8	103.6326	103.6326	5.6983e-05
10	9	96.1497	96.1497	5.8005e-05
11	10	89.3790	89.3789	5.8317e-05
12	11	83.2525	83.2524	5.8044e-05
13	12	77.7090	77.7090	5.7295e-05
14	13	72.6931	72.6931	5.6163e-05
15	14	68.1545	68.1545	5.4728e-05
16	15	64.0478	64.0478	5.3057e-05
17	16	60.3319	60.3319	5.1208e-05
18	17	56.9697	56.9696	4.9231e-05
19	18	53.9274	53.9273	4.7166e-05
20	19	51.1746	51.1745	4.5049e-05
21	20	48.6837	48.6837	4.2907e-05
22	21	46.4299	46.4299	4.0765e-05
23	22	44.3906	44.3906	3.8643e-05
24	23	42.5453	42.5453	3.6555e-05

	x1	y1	y_exact	Error_R
25	24	40.8757	40.8756	3.4514e-05
26	25	39.3649	39.3649	3.2531e-05
27	26	37.9979	37.9979	3.0613e-05
28	27	36.7610	36.7610	2.8765e-05
29	28	35.6418	35.6418	2.6991e-05
30	29	34.6291	34.6291	2.5295e-05
31	30	33.7128	33.7127	2.3677e-05

```
timetaken_Runge_Cooling = toc
```

```
timetaken_Runge_Cooling = 0.0701
```

3.2.6 Data Analysis

```
%TIME COMPARISON
Method = {'Euler'; 'Heun'; 'RK4'};
CompTime = [timetaken_Eulers_Cooling; timetaken_Heuns_Cooling;
timetaken_Runge_Cooling];
Time_Table = table(Method, CompTime)
```

```
Time_Table = 3×2 table
```

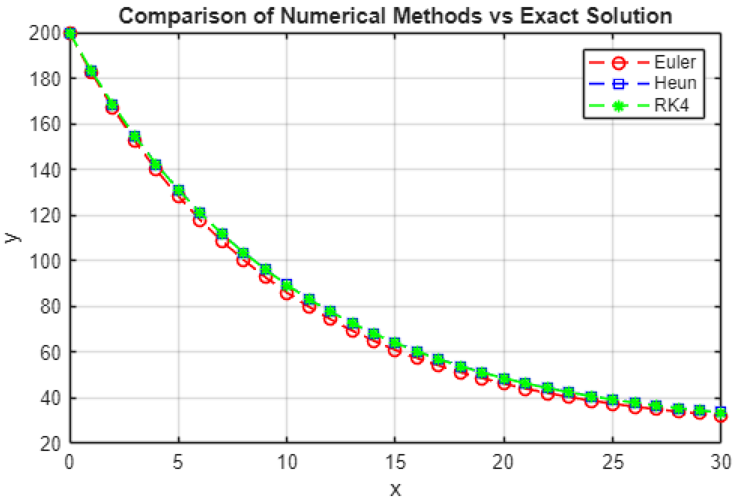
	Method	CompTime
1	'Euler'	0.1859
2	'Heun'	0.1217
3	'RK4'	0.1493

```
%%PLOTS
```

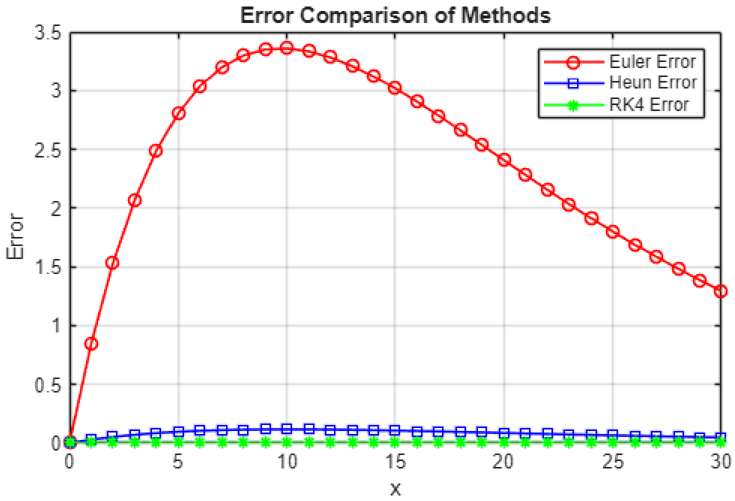
```
 figure;
 plot(Results_Eulers_Cooling.X, Results_Eulers_Cooling.Y, 'r--
o','LineWidth',1.2);
 hold on;
 plot(Results_Heuns_Cooling.X_H, Results_Heuns_Cooling.Y_H, 'b--
s','LineWidth',1.2);
 plot(Results_Runge_Cooling.x1, Results_Runge_Cooling.y1, 'g--
*','LineWidth',1.2);
 legend('Euler','Heun','RK4','Exact');
```

```
Warning: Ignoring extra legend entries.
```

```
 xlabel('x');
 ylabel('y');
 title('Comparison of Numerical Methods vs Exact Solution');
 grid on;
```



```
 figure;
 plot(Error_Table.x, Error_Table.Euler_Error, 'r-o','LineWidth',1.2);
 hold on;
 plot(Error_Table.x, Error_Table.Heun_Error, 'b-s','LineWidth',1.2);
 plot(Error_Table.x, Error_Table.RK4_Error, 'g-*','LineWidth',1.2);
 legend('Euler Error','Heun Error','RK4 Error');
 xlabel('x');
 ylabel('Error');
 title('Error Comparison of Methods');
 grid on;
```


Error Comparison of Methods
Euler Error
Heun Error
RK4 Error
Error
x
0
0.5
1
1.5
2
2.5
3
3.5
0
5
10
15
20
25
30

4 CONCLUSION

The project was successfully executed through collaboration and step-by-step implementing of ideas in MATLAB. Both numbers demonstrated the ability of transforming functional data given in question and converting it into logical and thematical algorithms that could be run by MATLAB to get a final output.

Through plotting of graphs, we were able to come to a stable conclusion that fixed point iterations may be the fastest method of finding roots of equations but due to the difficulty involved in forming a secondary equation, the second fastest option, Newton Raphson Formula may be preferred.

These graphics also facilitated achieving better perception of the errors that may be attained when using specific methods for example the most favorable method of doing the solving Ordinary Differential Equations is realized to be the 4th Order Runge Kutta Method.

This is because while it may not be the absolute fastest method of solving the equations, it is the most accurate next to the exact solutions. If speed was someone's concern, they would opt for the fastest method, Heuns Method here.

In assignment two, personal descriptive details of our group were maintained systematically and programmatically assessed under MATLAB struct array data types. Age distribution, enrollment in courses, religion, and personal interests were analyzed and represented graphically. This assignment enabled us to comprehend how descriptive characteristics can be combined and compared.

We faced a challenge on trying to find specific loops to use for each different methods operation. Luckily this is where we applied knowledge from documentation function from MATLAB better understand how each is done.

5 REFERENCES

- *MATLAB Documentation* Retrieved from: https://www.mathworks.com/help/matlab/
- Engineering Math II Lecture notes provided by Mr. Igga
- Numerical Methods and Optimization by Éric Walter
- Kirani Singh, T., & Chanduri, B. B. *MATLAB Programming*. PHI Learning Pvt. Ltd.
- Course Notes of Modules 1-5: Computer Programming with MATLAB, Mr. Maseruka Benedicto