FACULTY OF ENGINEERING AND TECHNOLOGY

COMPUTER PROGRAMMING

REPORT OF ASSIGNMENT THREE

BY

GROUP 10

SUBMITTED BY

1. Apio Laura Oula
2. Muganyizi James Factor
3. Arionget Shamim Egonu
4. Otim Innocent Lemo
5. Kakooza Ian Maurice
6. Bisoboka Jemimah Kairu
7. Ngobi Mark Livingstone
8. Rom Christopher Nyeko

This assignment report is submitted to the lecturer of computer programming Mr. BENEDICTO MASERUKA by Group 10.

Submitted on…/……/……….

DECLARATION

We hereby declare 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

NAME	SIGNATURE
Apio Laura Oula	…………...…………………
Muganyizi James Factor	………………………………
Arionget Shamim Egonu	………………………………
Otim Innocent Lemo	………………………………
Kakooza Ian Maurice	………………………………
Bisoboka Jemimah Kairu	………………………………
Ngobi Mark Livingstone	………………………………
Rom Christopher Nyeko	……………………………….

APPROVAL

This is to confirm that this report has been written and presented by GROUP 10 giving the details for the assignment.

LECTURER'S NAME: ……………………………………………………………
SIGNITURE: …………………………………………………………………………
DATE: ……………………………………………………………………………………

DEDICATION

We dedicate this report to all Group 10 members, who have been there with us in the process of researching and doing and compiling this report. To our lecturer Mr. Maseruka Benedicto whose guidance and expertise have been so needful, your mentorship and lecturing has built our understanding.

ACKNOWLEDGEMENT

First and foremost, we would like to thank the Almighty God for giving us the knowledge and guidance while doing our assignment as group 10.

We extend our gratitude to all the persons with whose help we managed to make it this far

The love of every group member to invest time and provide all they could to see the assignment a success.

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 on how to interact with the matlab interface, import, extract and feeding in data in to MATLAB as per the assignment which consisted of retrieving excel data from Kaggle.com
website, copying variables of each year to tables, converting tables to structural arrays, outputting each variable in to a single work book and generating a MATLAB code that can store each members' affirmation attributes, we managed to achieve this through group work and division of tasks.

APPROVAL

We are presenting this report which has been written and produced under our efforts.

TABLE OF CONTENTS

CHAPTER ONE: INTRODUCTION

1.1 Historical background

MATLAB, which stands for matrix laboratory, is a high-performance programming language and environment designed primarily for technical computing. Its origins trace back to the late 1970s when Cleve Moler, a professor of computer science, developed it to provide his students with easy access to mathematical software libraries without requiring them to learn Fortran.

MATLAB is built around the concept of matrices, making it particularly effective for linear algebra and matrix manipulation. It provides a vast library of built-in functions for mathematical operations, statistics, optimization, and other specialized tasks.

MATLAB offers powerful tools for creating 2D and 3D plots, enabling users to visualize data effectively. Specialized toolboxes extend MATLAB's capabilities, providing functions tailored for specific applications like signal processing, image processing, control systems, and machine learning.

MATLAB can interface with other programming languages (like C, C++, and Python) and software tools, allowing for flexible integration into larger systems. Its interactive environment features a command window, workspace, and editor, making it accessible for both beginners and advanced users.

1.2 Historical Development

The first version of MATLAB was created in Fortran in the late 1970s as a simple interactive matrix calculator. This early iteration included basic matrix operations and was built on top of two significant mathematical libraries: LINPACK and EISPACK, which were developed for numerical linear algebra and eigenvalue problems, respectively.

Recent versions of MATLAB have introduced features like the Live Editor, which allows users to create interactive documents that combine code, output, and formatted text. This evolution reflects MATLAB's ongoing adaptation to meet the needs of its diverse user base across academia and industry.

CHAPTER TWO: STUDY METHODOLOGY

2.1 Introduction

At the start, each member was given a task of making research about the assignment before our first meeting. The research concepts were obtained through watching tutorials on U-tube and also consultations from other continuing students especially those in year three and four.

2.2 Question

a) In your different groups, utilize the knowledge of Algorithm development, control structures and modules 1 -4 on the following problems.

b) All Numerical Approximation Methods for finding the solutions to functions. These include but are not limited to Newton Raphson, Secant, etc.

c) All methods for solving differential equations numerically. These include but are not limited to Range-Kutta, Euler, etc.

d) Ensure to apply a) and b) on a practical real-world problem

e) Note requirements

Ensure the different methods are tested on similar problems (each a minimum of two). This will help you to plot graphs that compare the problems analytical solutions to the solutions obtained by the different methods along with the computation time.

CHAPTER THREE: NUMERICAL APPROXIMATION METHODS

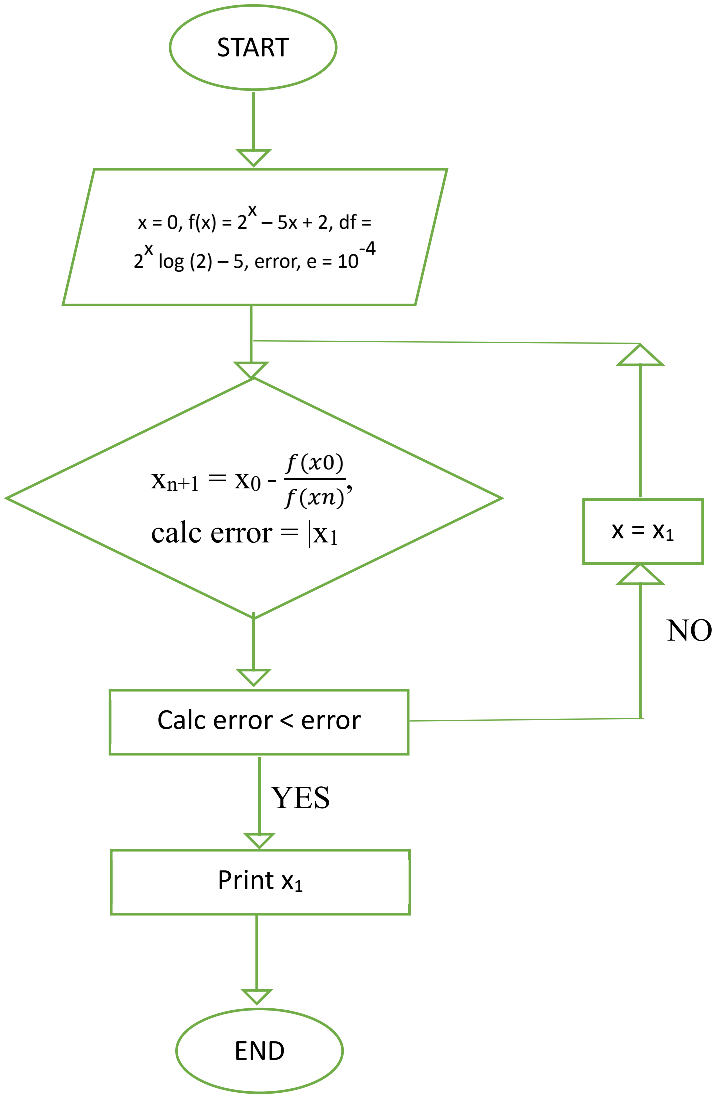
3.1 Question

b) All Numerical Approximation Methods for finding the solutions to functions. These include but are not limited to Newton Raphson, Secant, etc.

3.2 Newton-Raphson Method

Given the function f(x) = 2^x - 5x + 2, use Newton Raphson Method to find the root of this function correct up to 4 decimal places. (e = 10^-4)

Flow Chart


START
x = 0, f(x) = 2^X – 5x + 2, df = 2^X log (2) – 5, error, e = 10^-4
x_{n+1} = x_0 - f(x0)/f(xn)',
calc error = |x_1
x = x_1
NO
Calc error < error
YES
Print x_1
END

Requirements

```
f = @ (x) 2^x - 5*x + 2; % Funtion
df = @ (x) log(2)*(2^x) - 5; %Derivative of function
e = 10^-4; % Tolerance
x0 = 0; % Initial guess
n = 10; % Number of iterations
tic;
if df (x0) ~=0
  for i=1:n
      x1 = x0 - f(x0)/df(x0) %Newton Raphson Formula
      fprintf('x%d = %.10f\n',i,x1)
      if abs(x1-x0)<e
         break
      end
      if df(x1)==0
         disp('Newton Raphson Failed')
      end
      x0 = x1;
  end
else
   disp('Newton Raphson Failed');
end
elapsedTime = toc;
fprintf('Computation time: %.6f seconds\n',elapsedTime);
```

3.2.1 Output

```
x1 = 0.6965643187
x2 = 0.7321153457
x3 = 0.7322442538
x4 = 0.7322442555
Computation time: 0.027102 seconds
```

3.3 Bisection Method

Given the function f(x) = 2^x - 5x + 2, use Bisection Method to find the root of this function correct up to 4 decimal places.(e = 10^-4)

Requirements

```
f = @ (x) 2^x - 5*x + 2;  % Function
a = 0;  %Leftside of interval
b = 1;  %Rightside of the interval
n = 30;  %Number of iterations
e = 10^-4;  %Tolerance
tic;
if f(a)*f(b)<0
   for i=1:n
      c = (a+b)/2;   %Bisecion method formula
      if abs(c-b)<e || abs(c-a)<e
         break
      end
      fprintf('P%d = %.4f\n',i,c)
      if f(a)*f(c)<0
         b = c;
      elseif f(b)*f(c)<0
         a = c;
      end
   end

else
   disp('No root between given brackets')
end
elapsedTime = toc;
fprintf('Computation time: %.6f seconds\n',elapsedTime);
```

3.3.1 Output

```
P1 = 0.5000
```

```
P2 = 0.7500
P3 = 0.6250
P4 = 0.6875
P5 = 0.7188
P6 = 0.7344
P7 = 0.7266
P8 = 0.7305
P9 = 0.7324
P10 = 0.7314
P11 = 0.7319
P12 = 0.7322
P13 = 0.7323
Computation time: 0.136205 seconds
```

3.4 Secant Method

Given the function f(x) = 2^x - 5x + 2, use Secant Method to find the root of this function correct up to 4 decimal places.(e = 10^-4)

Requirements

```
f = @ (x) 2^x - 5*x + 2;  % Function
x0 = 0; %Initial first guess
x1 = 1; %Second initial guess
e = 10^-4; %Tolerance
n = 10; %Number of iterations
tic;
for i=1:n
   x2 = ((x0)*f(x1)-x1*f(x0))/(f(x1)-f(x0))  % Secant Method Formula
   fprintf('x%d = %.4f\n',i,x2)
   if abs(x2-x1)<e
      break
   end
   x0 = x1;
```

```
    x1 = x2;
end
elapsedTime = toc;
fprintf('Computation time: %.6f seconds\n',elapsedTime);
```

3.4.1 Output

```
x1 = 0.7500
x2 = 0.7317
x3 = 0.7322
x4 = 0.7322
Computation time: 0.032729 seconds
```

3.5 Fixed Point Iteration Method

Given the function f(x) = 2^x - 5x + 2, use Fixed Point Iteration Method to find the root of this function correct up to 4 decimal places.(e = 10^-4). Use x0 = 0 as initial guess.

Requirements

```
g = @(x)(2^x+2)/5';
x0 = 0;
e = 10^-4;
n = 20;
tic;
for i=1:n
    x1 = g(x0)
    fprintf('x%d = %.4f\n',i,x1)
    if abs(x1-x0)<e
        break
    end
    x0 =x1;
end
elapsedTime = toc;
```

```
fprintf('Computation time: %.6f seconds\n',elapsedTime);
```

3.5.1 Output

```
x1 = 0.6000
x2 = 0.7031
x3 = 0.7256
x4 = 0.7307
x5 = 0.7319
x6 = 0.7322
x7 = 0.7322
Computation time: 0.049905 seconds
```

3.6 Cramer's Rule

Solve the following system of linear equations using Cramer's Rule

$10x_1 + 3x_2 + x_3 = 19$

$3x_1 + 10x_2 + 2x_3 = 29$

$x_1 + 2x_2 + 10x_3 = 35$

Requirements

```
A = [10 3 1;3 10 2;1 2 10]   % coefficient matrix
b = [19;29;35] % source vector
N = length(b)
x = zeros(N,1)
d = det(A)
Aold = A;
tic;
if d~=0
  for i=1:N
     A(:,i) = b
     x(i) = det(A)/d
     A = Aold
  end
```

```
   disp('Solution using cramers rule is')
   x
else
    disp('Cramers rule not applicable')
end
elapsedTime = toc;
fprintf('Computation time: %.6f seconds\n',elapsedTime);
```

3.6.1 Results

```
A = 3×3
    10     3     1
     3    10     2
     1     2    10

b = 3×1
    19
    29
    35
N = 3

x = 3×1
     0
     0
     0
d = 872
A = 3×3
    19     3     1
    29    10     2
    35     2    10
x = 3×1
    1.0000
```

```
        0
        0
A = 3×3
    10     3     1
     3    10     2
     1     2    10
x = 3×1
    1.0000
    2.0000
         0
A = 3×3
    10     3     1
     3    10     2
     1     2    10
x = 3×1
    1.0000
    2.0000
    3.0000
A = 3×3
    10     3     1
     3    10     2
     1     2    10
Solution using cramers rule is
x = 3×1
    1.0000
    2.0000
    3.0000
Computation time: 0.163396 seconds
```

3.7 Lagrange Interpolation

Find the interpolating polynomial that passes through (2,5), (3,7), (5,8) hence manipulate the value of y at x = 4.

Requirements

```
X = [2;3;5]     % list of abscissas
Y = [5;7;8]     % list of ordinates
P0 =  4    % point at which approximation is wanted
n = length(X);
L = zeros(n,n);
tic;
for i=1:n  %for rows
   V = 1;
   for j=1:n  %for making polynomial
      if i~=j
         V = conv(V,poly(X(j)))/(X(i)-X(j))
      end
   end
   L(i,:) = V*Y(i);
end
L;
P = sum(L);
F = flip(P);
disp('Polynomial is:')
for k=n:-1:2
   fprintf('+(%.2fx^%d)',F(k), k-1)
end
elapsedTime = toc;
fprintf('Computation time: %.6f seconds\n',elapsedTime);
fprintf('+(%.2f)',F(1))
A = polyval(P,P0);
fprintf('Approximate value at given data point is: %.4f',A)
x = linspace(X(1),X(n),100);
```

```
y = polyval(P,x);
plot(x,y,'r');
hold on
plot(X,Y,'o');
xlabel('X Values');
ylabel('Y Values');
title('Lagrange Interpolation');
grid on;
```

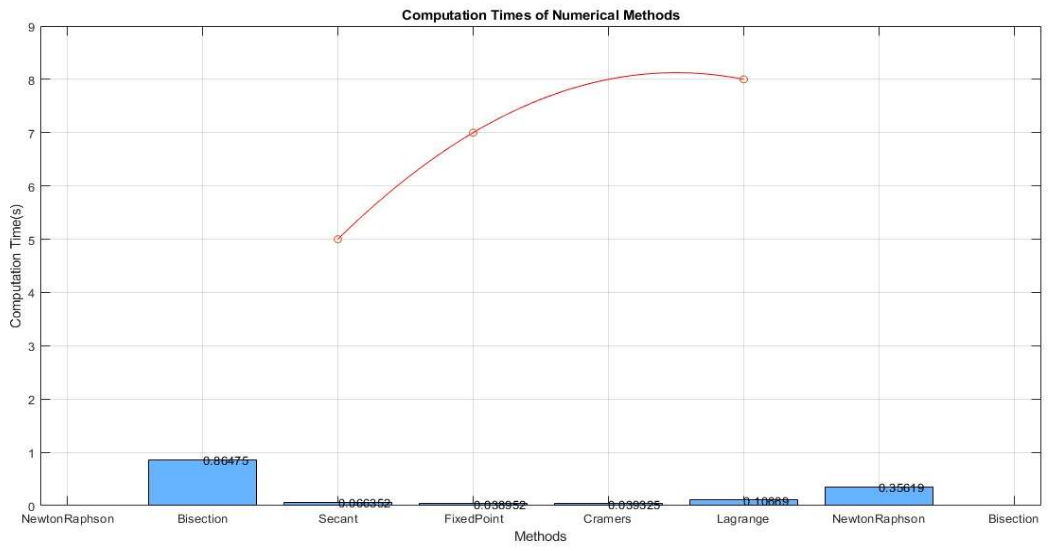
3.7.1 Results

```
X = 3×1
     2
     3
     5
Y = 3×1
     5
     7
     8
P0 = 4
V = 1×2
    -1     3
V = 1×3
    0.3333   -2.6667    5.0000
V = 1×2
     1    -2
V = 1×3
   -0.5000    3.5000   -5.0000
V = 1×2
    0.3333   -0.6667
V = 1×3
    0.1667   -0.8333    1.0000
```

```
Polynomial is: +(-0.50x^2)+(4.50x^1)+(-2.00)
Approximate value at given data point is: 8.0000
Computation time: 0.180388 seconds
```

3.8 Plotting the Bar Graph for Computation Time

```
methods = {'NewtonRaphson','Bisection','Secant','FixedPoint','Cramers','Lagrange'};
Computation_times = [0.864748,0.066352,0.038952,0.039325,0.106690,0.356190];
bar(Computation_times,'FaceColor',[0.4 0.7 1]);
set(gca,'XTickLabel',methods);
title('Computation Times of Numerical Methods');
xlabel('Methods');
ylabel('Computation Time(s)');
grid on;
xtips = 1:length(Computation_times);
ytips = Computation_times;
labels = string(Computation_times);
text(xtips,ytips + 0.0002, labels);
box on;
```



CHAPTER FOUR: METHODS OF SOLVING ORDINARY DIFFERENTIAL EQUATIONS

4.1 Euler Algorithm Method

Solve y' = sin(x) - y using Euler Algorithm, with y (-2) = 3 as the initial condition

4.1.1 Requirements

```
f = @(x,y) sin(x) - y;
x0 = -2;
y0 = 3;
c = exp(x0)*(y0 +0.5*(cos(x0) -sin(x0)));
fExact = @(X) c*exp(-X) + sin(X)/2 - cos(X)/2;

xf = 1; % Must be bigger than x0 (final value of x)
h = 0.01; % Domain steps (Step size)
X = x0:h:xf; % Domain value-Vector
figure(1);
plot(X, fExact(X), 'b-');
title('Numerical Solutions for $$\frac{dy}{dx} = \sin(x) - y$$','interpreter','latex');
xlabel(X);
ylabel(fExact(X));
grid on;
```

Numerical solutions

```
n = (xf -x0)/dx +1;
y_euler = zeros(n,1); % Euler
y_euler(1) =y0;
y_midpt = zeros(n,1); % Midpoint
y_midpt(1) = y0;
y_rk4 = zeros(n,1); % Classical RungeKutta 4
y_rk4(1) = y0;
for i=2:n
   %When i = 2, this corresponds to y(k+1) = yk + h*f(xk, yk) k = 1
```

```
    xk = x0 + (i - 2)*h;

    %Euler (RungeKutta 1) Method
    y_euler(i,1) = y_euler(i - 1,1) + h*f(xk, y_euler(i - 1));
    % Midpoint (RungeKutta 2) Method
    L1a = f(xk,y_midpt(i-1,1));
    L2a = f(xk + h/2, y_midpt(i-1,1) + (h/2)*L1a);
    y_midpt(i,1) = y_midpt(i-1,1) + h*L2a;

    % Classical RungeKutta 4 Method
    L1b = f(xk,y_rk4(i-1,1));
    L2b = f(xk + h/2, y_rk4(i-1,1) + (h/2)*L1b);
    L3b = f(xk + h/2, y_rk4(i-1,1) + (h/2)*L2b);
    L4b = f(xk + h, y_rk4(i-1,1) + h*L3b);
    y_rk4(i,1) = y_rk4(i-1,1) + (h/6)*(L1b + 2*L2b + 2*L3b + L4b);
end
hold on;
plot(X,y_euler(:,1),'k-');
plot(X,y_midpt(:,1),'g-');
plot(X,y_rk4(:,1), 'r-');
```

4.2 Runge-Kutta For Order 4

Given that y' +20y = 7e^(-0.5t); y(0) = 5, compute y(0.2) using Runge-Kutta for order 4 by taking h = 0.1.

4.2.1 Requirements

```
f = @(t,y) -20*y + 7*exp(-0.5*t); %Function
t0 = 0; % Initial value of independent variable
y0 = 5; % Initial value of dependent variable
h = 0.1; % Step size
tn = 0.2; % Point at which the solution is being evaluated
n = (tn-t0)/h;
```

```
t(1) = t0; y(1) = y0;
for i=1:n
    t(i+1) = t0 + i*h;
    k1 = h*f(t(i),y(i))
    k2 = h*f(t(i) + (h/2),y(i)+(k1/2));
    k3 = h*f(t(i) + (h/2),y(i)+(k2/2));
    k4 = h*f(t(i) + h,y(i)+k3);
    y(i+1) = y(i) + (1/6)*(k1+2*k2+2*k3+k4);
    fprintf('y(%.2f) = %.4f\n',t(i+1),y(i+1))
end
```

4.2.2 Output

```
k1 = -9.3000
y(0.10) = 1.8885
k1 = -3.1112
y(0.20) = 0.8406
```

CHAPTER FIVE: APPLICATION OF NUMERICAL METHODS IN REAL-WORLD PROBLEMS

5.1 Real-World Problem One

Study Of the Transmission of a Disease in a Given Area While Looking at Those Susceptible, Infected and the Recovered

Solution Requirements

```
%Susceptible is S
%Infected is I
%Recovered is R

%The model is described by the following system of differential equations:

%dS/dt = -beta * S * I / N
%dI/dt = beta * S * I / N - gamma * I
%dR/dt = gamma * I

%where beta is the transmission rate,
%gamma is the recovery rate,
%N is the total population.

% Parameters
beta = 0.2;  % transmission rate
gamma = 0.1;  % recovery rate
N = 1000;  % total population
S0 = 900;  % initial susceptible population
I0 = 100;  % initial infected population
R0 = 0;  % initial recovered population
tspan = [0 100];  % time span
```

5.1.1 Method One: Euler Method

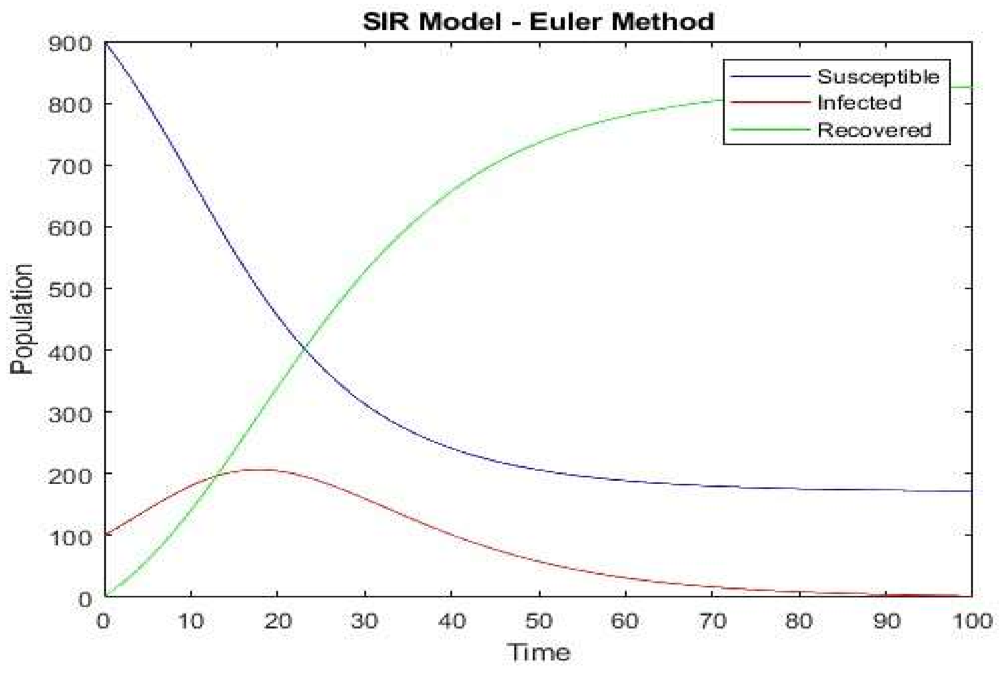
Code

```
dt = 0.1;
t = tspan(1):dt:tspan(2);
S = zeros(size(t));
I = zeros(size(t));
R = zeros(size(t));
S(1) = S0;
I(1) = I0;
R(1) = R0;
for i = 2:length(t)
   S(i) = S(i-1) - beta * S(i-1) * I(i-1) / N * dt;
   I(i) = I(i-1) + (beta * S(i-1) * I(i-1) / N - gamma * I(i-1)) * dt;
   R(i) = R(i-1) + gamma * I(i-1) * dt;
```

```
end

% Plotting
figure;
plot(t, S, 'b-', t, I, 'r-', t, R, 'g-');
xlabel('Time');
ylabel('Population');
legend('Susceptible', 'Infected', 'Recovered');
title('SIR Model - Euler Method');
```

Output



5.1.2 Method Two: Runge-Kutta Method

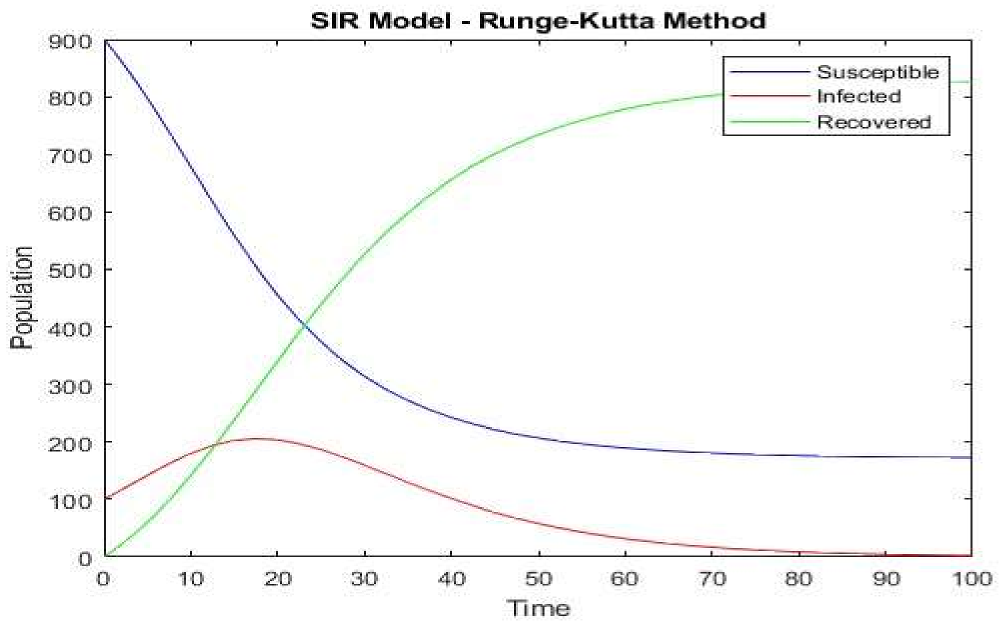
Code

```
[t_rk, y_rk] = ode45(@(t, y) sir_model(t, y, beta, gamma, N), tspan, [S0 I0 R0]);
S_rk = y_rk(:, 1);
I_rk = y_rk(:, 2);
R_rk = y_rk(:, 3);
function dydt = sir_model(~, y, beta, gamma, N)
   % y(1) = S (susceptible), y(2) = I (infected), y(3) = R (recovered)
```

```
    S = y(1);
    I = y(2);
    %R = y(3);
    dS = -beta * S * I / N;
    dI = beta * S * I / N - gamma * I;
    dR = gamma * I;
    dydt = [dS; dI; dR];
end

% Plotting
figure;
plot(t_rk, S_rk, 'b-', t_rk, I_rk, 'r-', t_rk, R_rk, 'g-');
xlabel('Time');
ylabel('Population');
legend('Susceptible', 'Infected', 'Recovered');
title('SIR Model - Runge-Kutta Method');
```

Output



5.2 Real-World Problem Two

Finding the Equilibrium Point of a Predator-Prey System

Solution Requirements

% we shall use the following equation.

% f(x) = x - (r / a) * (1 - x / K)

% where x is the prey population,

% r is the growth rate of the prey,

% a is the predation rate,

% K is the carrying capacity of the prey.

5.2.1 Method One: Newton-Raphson Method

% x_{n+1} = x_n - f(x_n) / f'(x_n)

%where f'(x) is the derivative of f(x).

Code

```
% Parameters
r = 0.5;  % growth rate of prey
a = 0.02;  % predation rate
K = 100;  % carrying capacity of prey
% Function
f = @(x) x - (r / a) * (1 - x / K);
% Derivative of function
df = @(x) 1 + (r / a) * (x / K);
x0 = 50;  % initial guess
tol = 1e-8;  % tolerance
max_iter = 1000;  % maximum number of iterations
x_nr = x0;
x_nr_history = x_nr;
for i = 1:max_iter
    x_nr_new = x_nr - f(x_nr) / df(x_nr);
    x_nrhistory = [x_nr_history, x_nr_new];
    if abs(x_nr_new - x_nr) < tol
```

```
        break;
    end
    x_nr = x_nr_new;
end

if abs(f(x_nr)) > tol
    fprintf('Newton-Raphson Method did not converge within %d iterations.\n', 
max_iter);
end
```

5.2.2 Method Two: Secant Method

```
%The Secant method uses the formula:
% x_{n+1} = x_nr - f(x_nr) * (x_nr - x_{n-1}) / (f(x_nr) - f(x_{n-1}))
```

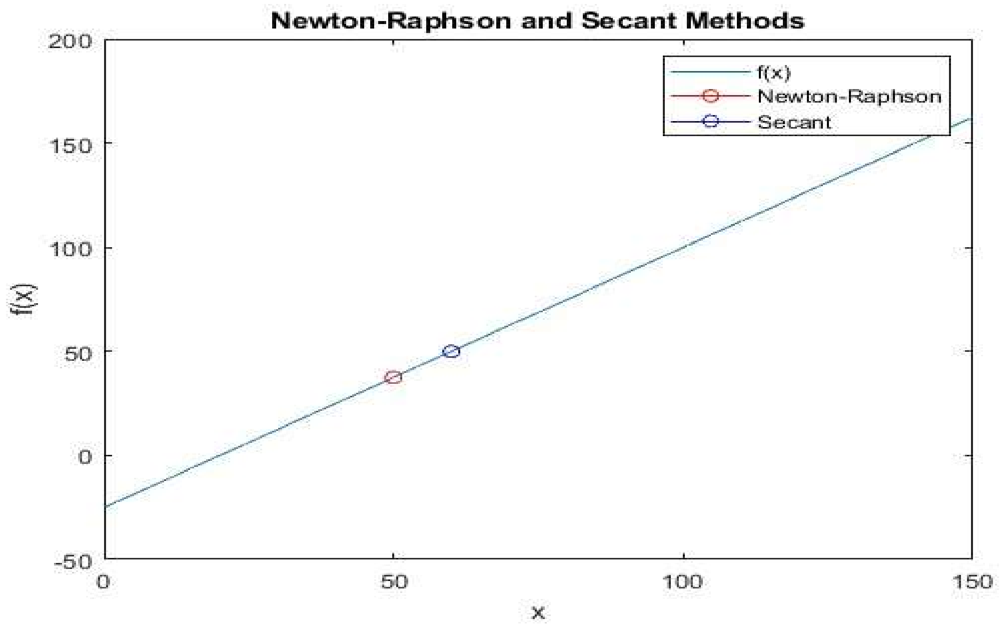
Code

```
x0 = 40;  % initial guess 1
x1 = 60;  % initial guess 2
tol = 1e-8;  % tolerance
max_iter = 1000;  % maximum number of iterations
x_secant = x1;
x_prev = x0;
x_secant_history = x_secant;
for i = 1:max_iter
    x_secant_new = x_secant - f(x_secant) * (x_secant - x_prev) / (f(x_secant) - 
f(x_prev));
    x_secanthistory = [x_secant_history, x_secant_new];
    if abs(x_secant_new - x_secant) < tol
        break;
    end
    x_prev = x_secant;
    x_secant = x_secant_new;
end
```

```
if abs(f(x_secant)) > tol
    fprintf('Secant Method did not converge within %d iterations.\n', max_iter);
end

% Plot the function
x = linspace(0, 150, 400);
y = f(x);
plot(x, y);
hold on;
plot(x_nr_history, f(x_nr_history), 'ro-');
plot(x_secant_history, f(x_secant_history), 'bo-');
legend('f(x)', 'Newton-Raphson', 'Secant');
xlabel('x');
ylabel('f(x)');
title('Newton-Raphson and Secant Methods');
fprintf('Newton-Raphson Method: x = %.4f\n', x_nr);
fprintf('Secant Method: x = %.4f\n', x_secant);
```

Output



CONCLUSION

The assignment was successful since there was maximum cooperation among group 10 members.

The project applied numerical methods like Newton Raphson, Euler, Runge-Kutta to solve functions and differential equations for real-world problems.

Comparing analytical and numerical solutions showed the importance of choosing the right method considering accuracy, stability, and computation time.