BUSITEMA UNIVERSITY

Pursuing Excellence

FACULTY OF ENGINEERING & TECHNOLOGY

1.1.1.1.1 ASSIGNMENT REPORT ON ALGORITHM DEVELOPMENT, CONTROL STRUCTURES USING THE KNOWLEDGE OF MATLAB MODULES 1-4

COMPUTER PROGRAMMING

COURSE LECTURER: MR. MASERUKA BENDICTO

By GROUP 18

ABSTRACT

This report looks at the numerical approximation methods for finding solutions to functions that is Newton Raphson Method, secant method, bisection method and fixed point iteration. It further more looks at the methods for solving differential equations numerically and they include Euler method and Runge-Kutta. The above methods were designed in the MATLAB environment that is the Live Script. Graphs are also added to compare the problems analytical solutions obtained by the different methods. The project demonstrated fundamental skills in data handling, organization, and problem-solving within the MATLAB environment, providing practical experience in a complete data workflow.

ACKNOWLEGEMENT

By the Grace of GOD we were able to work together as a group to complete the assignment and we acknowledge him for that.

We thank, Mr. Maseruka Bendicto our course lecturer for guiding us in this course which is a vital aspect for our engineering profession.

Appreciation goes to group members for the commitment and team spirit which simplified work and made it easy for us to complete the task and come up with this report.

DECLARATION

We, Group 18 members hereby declare to the best of our knowledge, that this assignment report is a true record of our unending efforts in applying the knowledge we acquired from modules one through four. It is truly an original creation of our own and it has never been used by any other individual for any academic award in any learning institution.

MEMBERS OF GROUP 18

	NAME	**REG NUMBER**	**PROGRAM**	**SIGN**
1	AUMA DIANA	BU/UP/2024/1020	WAR	
2	ENAMU REAGAN EGIMU	BU/UG/2024/2672	APE	
3	MUKHOOLI ELIJAH	BU/UG/2024/2586	MEB	
4	NABAWEESI CLAIRE	BU/UP/2024/1046	WAR	
5	NAKAWEESA LINNET	BU/UP/2024/4327	APE	
6	NANDAULA CATHERINE	BU/UP/2024/4322	AMI	
7	OLUK CHRISTIAN GLEN	BU/UP/2024/3842	WAR	
8	OMARA PASCAL KELLY	BU/UP/2024/1063	WAR	
9	TUMUHAISE SARAH	BU/UG/2024/2674	AMI	
10	UHURU DENISH BRIAN	BU/UP/2024/3841	WAR	

APPROVAL

This is to confirm that this report has been written and presented by Group 18, giving details of the assignment carried out.

Course Lecturer

Signature _____

Date

Table of Contents

ABSTRACT ..... **ii**

ACKNOWLEGEMENT ..... **iii**

DECLARATION ..... **iv**

APPROVAL ..... **v**

CHAPTER 2 : INTRODUCTION **1**

CHAPTER 3 : CONCLUSION **14**

CHAPTER 4 : REFERENCES **15**

CHAPTER 2 : INTRODUCTION

In this assignment we were required to using code within the MATLAB environment. In order to determine a root it is usually essential to have an initial estimate of its value. In some cases you may have more than one root (or none) and you wish to identify which one you are concerned with. The method we will describe now involves user interaction and is used as an illustration. As we are developing the ideas in this report we could consider how they could be generalized to include root selection. In general the methods we have talked about will require either an initial guess for the root or a bracketing interval containing a root. We were required to ensure that the different numerical methods were tested and that this would help us to plot graphs and compare the problem analytical solutions obtained along with the computation time.

TASK GIVEN

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

- All Numerical approximate methods for finding the solutions to functions, these include but are not limited to ; Newton Raphson
- All methods for solving differential equations numerically these include but are not limited to; Euler, Runge-kutta.
- Ensure to apply a and b on a practical real world problem

Note: requirements; ensure that different methods are stated on similar problems each (minimum of 2). This will help u to plot graphs that compare the problems analytical solutions to the solutions obtained by the different methods along with the computation time.

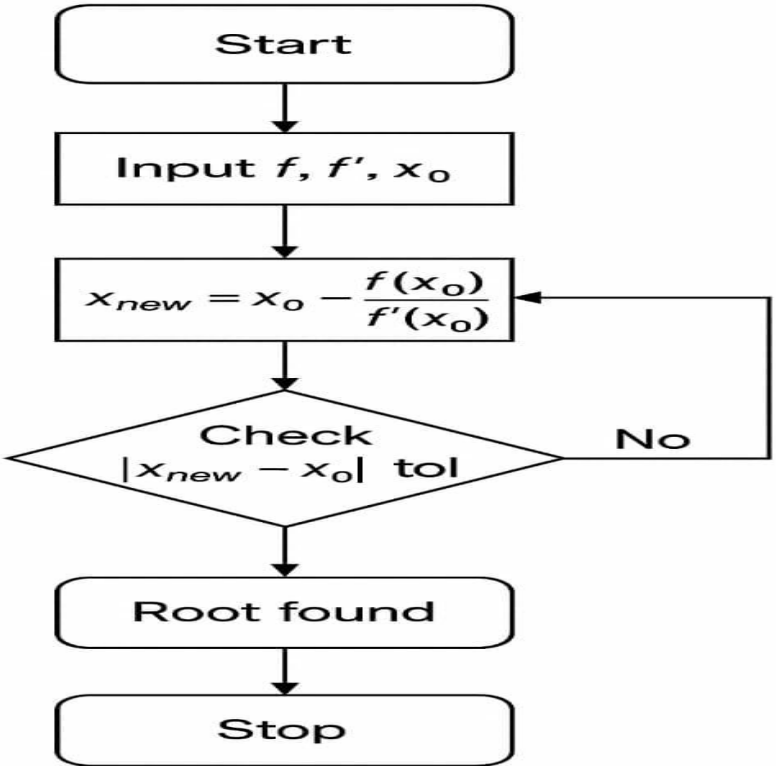
Solution to the question

Here is the question we are trying to solve using Newton's Raphson method, secant method, Bisection method and then fixed iteration

Solve $x^3 - 5x + 3 = 0$

Below are the flow charts;

1. Newton’s Raphson Method



Pseudo Codes for the flow Chart

Input: function f(x), derivative f’(x), initial guess x0, tolerance tol, maxIter

For k = 1 to maxIter

x_new = x0 - f(x0) / f’(x0)

If |x_new - x0| < tol

Root = x_new

Stop

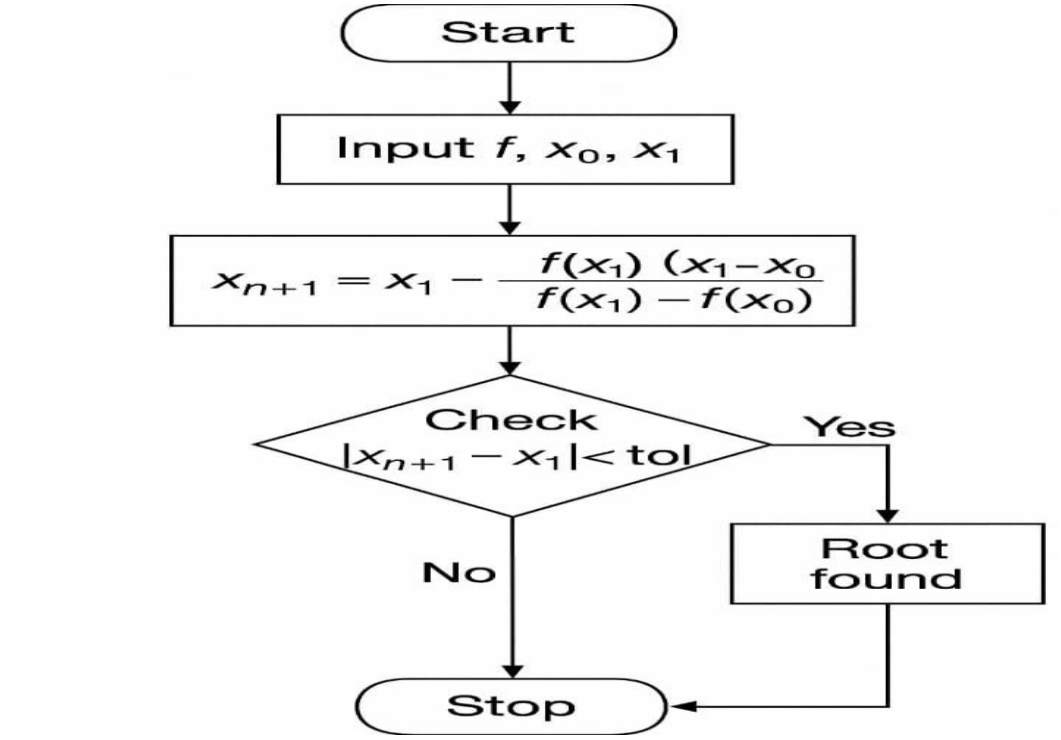
End If

x0 = x_new

End For

Output: Root

2. Secant Method



Pseudo code for the flow chart above

Input: f(x), two guesses x0, x1, tolerance tol, maxIter

For k = 1 to maxIter

x_new = x1 - f(x1)(x1 - x0)/(f(x1) - f(x0))*

If |x_new - x1| < tol

Root = x_new

Stop

End If

x0 = x1

x1 = x_new

End For

Output: Root

Combine codes for Newton and Secant Methods

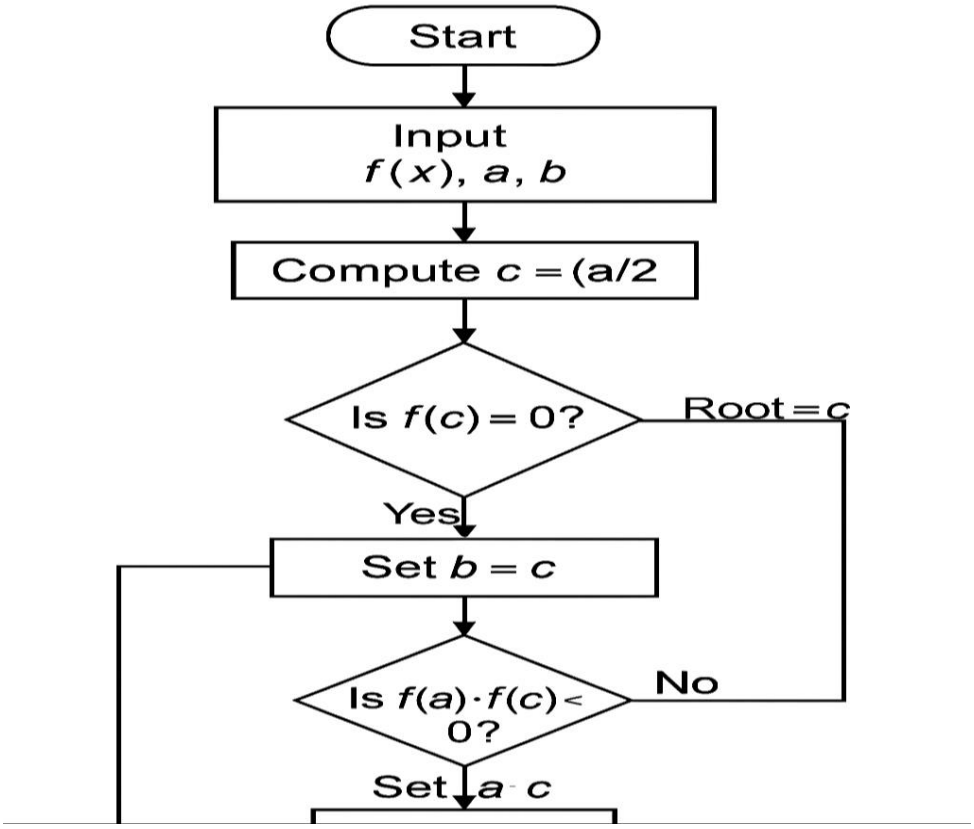
```
%%Root Finding using Newton-Raphson and Secant Methods
```

```
clc; clear; close all;


f = @(x) x.^3 - 5*x + 3;      % Function
df = @(x) 3*x.^2 - 5;         % Derivative for Newton-Raphson


% Initial guesses
x0 = 1;  % initial guess for Newton-Raphson
x1 = 0;  % first guess for Secant
x2 = 1;  % second guess for Secant


tol = 1e-6;
maxIter = 50;


% Newton-Raphson
tic;
for k = 1:maxIter
    x_new = x0 - f(x0)/df(x0);
    if abs(x_new - x0) < tol
        break;
    end
    x0 = x_new;
end
NR_root = x_new;
NR_time = toc;


% Secant Method
tic;
for k = 1:maxIter
    x_new = x2 - f(x2)*(x2-x1)/(f(x2)-f(x1));
```

```
    if abs(x_new - x2) < tol
        break;
    end
    x1 = x2;
    x2 = x_new;
end
Sec_root = x_new;
Sec_time = toc;


fprintf('Newton-Raphson Root: %.6f (time = %.6f s)\n', NR_root,
NR_time);
fprintf('Secant Method Root: %.6f (time = %.6f s)\n', Sec_root,
Sec_time);
```

3. Bisection Method

Flow Chart



Pseudo code

Input: f(x), interval [a,b], tolerance tol, maxIter

*If f(a)*f(b) > 0*

Print "No root in [a,b]"

Stop

End If

For k = 1 to maxIter

c = (a+b)/2

If |f(c)| < tol

Root = c

Stop

End If

*If f(a)*f(c) < 0*

b = c

Else

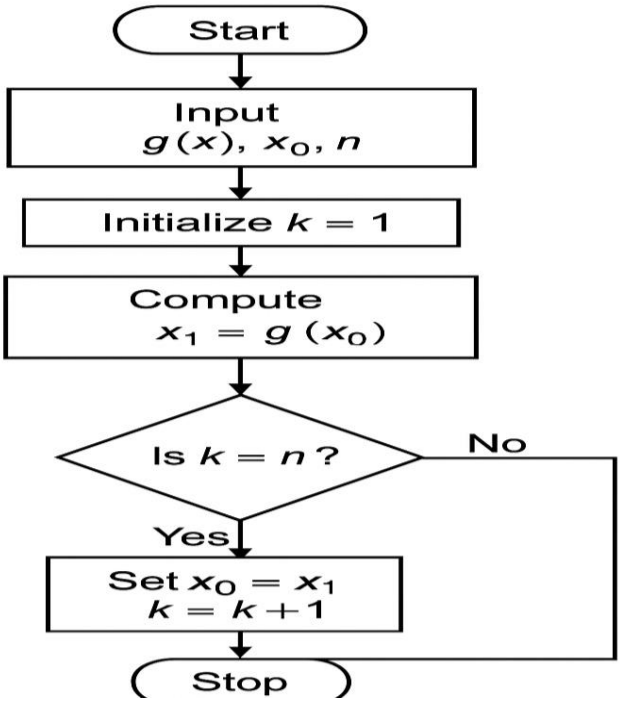
a = c

End If

End For

Output: Root

4. Fixed Iteration



Pseudo Code

Input: f(x), interval [a,b], tolerance tol, maxIter

*If f(a)*f(b) > 0*

Print "No root in [a,b]"

Stop

End If

For k = 1 to maxIter

c = (a+b)/2

If |f(c)| < tol

Root = c

Stop

End If

*If f(a)*f(c) < 0*

b = c

Else

a = c

End If

End For

Output: Root

Combined codes for Bisection and Fixed iteration

BISECTION

```
Input: f(x), interval [a,b], tolerance tol, maxIter
If f(a)*f(b) > 0
    Print "No root in [a,b]"
    Stop
End If
For k = 1 to maxIter
    c = (a+b)/2
    If |f(c)| < tol
        Root = c
        Stop
    End If
    If f(a)*f(c) < 0
        b = c
    Else
        a = c
    End If
End For
Output: Root
```

FIXED ITERATION

```
Input: g(x), initial guess x0, tolerance tol, maxIter
For k = 1 to maxIter
    x_new = g(x0)
    If |x_new - x0| < tol
        Root = x_new
        Stop
    End If
    x0 = x_new
End For
Output: Root
```

Codes on differential equation

```
% Solving ODE using Euler and Runge-Kutta



clc; clear; close all;
```

```
f = @(x,y) -2*y;  % dy/dx = -2y
x0 = 0; y0 = 1;   % initial condition
h = 0.1;          % step size
xf = 5;           % final x
N = (xf-x0)/h;


% Arrays
x = x0:h:xf;
y_euler = zeros(1,length(x));
y_rk4 = zeros(1,length(x));
y_exact = exp(-2*x);


% Initial values
y_euler(1) = y0;
y_rk4(1) = y0;


% Euler Method
tic;
for i = 1:N
    y_euler(i+1) = y_euler(i) + h*f(x(i), y_euler(i));
end
Euler_time = toc;


% RK4 Method
tic;
for i = 1:N
    k1 = h*f(x(i), y_rk4(i));
    k2 = h*f(x(i)+h/2, y_rk4(i)+k1/2);
    k3 = h*f(x(i)+h/2, y_rk4(i)+k2/2);
    k4 = h*f(x(i)+h, y_rk4(i)+k3);
```

```
    y_rk4(i+1) = y_rk4(i) + (k1+2*k2+2*k3+k4)/6;
end
RK4_time = toc;


% Plot
figure;
plot(x, y_exact, 'k-', 'LineWidth', 2); hold on;
plot(x, y_euler, 'r--o');
plot(x, y_rk4, 'b--s');
legend('Exact','Euler','RK4');
xlabel('x'); ylabel('y');
title('Numerical vs Analytical Solutions');


fprintf('Euler Method time: %.6f s\n', Euler_time);
fprintf('RK4 Method time: %.6f s\n', RK4_time);
```

Practical question used in daily life to apply the above methods

Qn. A metal rod heated at 90^{o}c is left to cool in a workshop where the surrounding temperature is 25^{o}c if the cooling constant is 0.1per min, predict the rod's temperature for the next 60minutes using;

a) The analytical solution of newton's law of cooling.

b) Euler's method

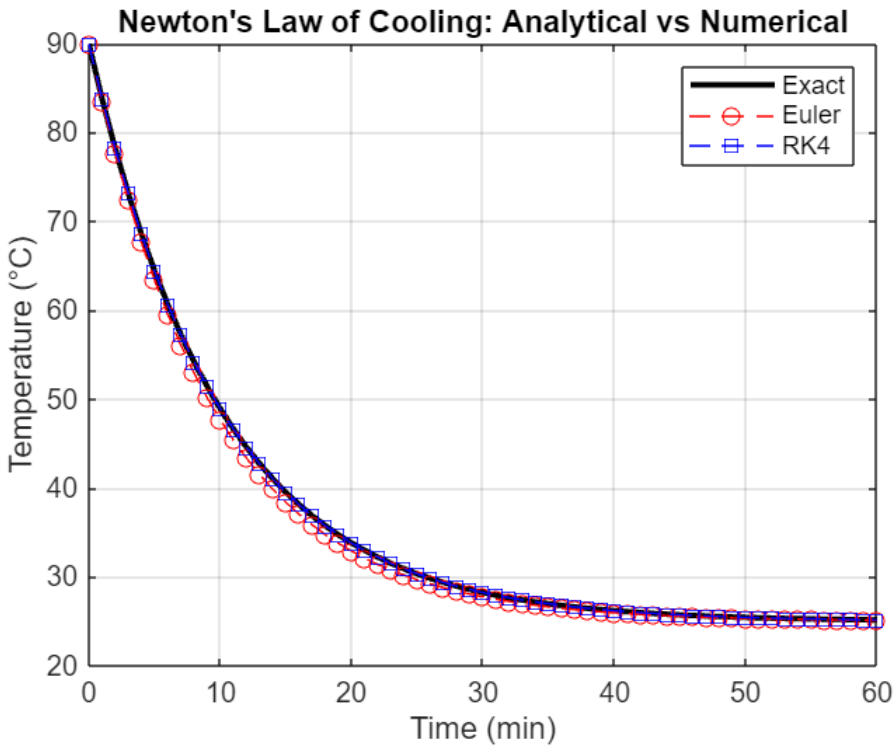
c) Runge-Kutta 4 method with a step size of 1min. compare the results.

CODES FOR THE SOLUTION

```
%% Newton's Law of Cooling
clc; clear; close all;
```

```
% Parameters
T0 = 90;          % Initial temperature (°C)
T_room = 25;      % Room temperature (°C)
k = 0.1;          % Cooling constant (1/min)
t_final = 60;     % Simulate 60 minutes
h = 1;            % Step size (1 min)


% Time vector
t = 0:h:t_final;


% Analytical solution
T_exact = T_room + (T0 - T_room)*exp(-k*t);


%% Euler Method
T_euler = zeros(size(t));
T_euler(1) = T0; % initial condition
for i = 1:length(t)-1
    T_euler(i+1) = T_euler(i) + h*(-k*(T_euler(i)-T_room));
end


%% Runge-Kutta 4 (RK4) Method
T_rk4 = zeros(size(t));
T_rk4(1) = T0; % initial condition
for i = 1:length(t)-1
    f = @(T) -k*(T - T_room);
    k1 = h*f(T_rk4(i));
    k2 = h*f(T_rk4(i) + k1/2);
    k3 = h*f(T_rk4(i) + k2/2);
    k4 = h*f(T_rk4(i) + k3);
    T_rk4(i+1) = T_rk4(i) + (k1 + 2*k2 + 2*k3 + k4)/6;
```

```
end


%% Plot Results
figure;
plot(t, T_exact, 'k-', 'LineWidth', 2); hold on;
plot(t, T_euler, 'ro--');
plot(t, T_rk4, 'bs--');
xlabel('Time (min)');
ylabel('Temperature (°C)');
legend('Exact','Euler','RK4');
title('Newton's Law of Cooling: Analytical vs Numerical');
grid on;
```



The graph above consists of three curves . the plot demonstrates the way newton's law of cooling works in practice and shows how the different numerical methods change. Euler's method, is a bit simple but less accurate RK4 gives a better approximation which is almost similar to the exact solution. The methods above show the same physical behavior which is an exponential cooling of the object as it approaches room temperature of 25^{o}c over time.

CHAPTER 3 : CONCLUSION

For the various numerical methods of finding roots, we realized that the number of iterations for a given method vary according to the initial value; x_0 chosen. Newton is very fast. Secant methods are intermediate in speed. Newton and Secant can fail if x_0 is not close to $x*$. We got hardships in coming up with the codes which required us to utilize the programming and data visualization skills with also visiting the internet for the tutorials.

CHAPTER 4 : REFERENCES

- MATLAB Documentation
- MATLAB lecture notes by Mr. Maseruka Bendicto
- You tube MATLAB tutorials