

FACULTY OF ENGINEERING AND TECHNOLOGY

**REPORT ABOUT USING THE ALGORITHM DEVELOPMENT,**

**CONTROL STRUCTURE AND USING MODULES 1-4 ON**

**PRATICAL REAL WORLD PROBLEM**

COURSE UNIT: COMPUTER POGRAMMING

LECTURER: MR. MASERUKA BENEDICTO

BY GROUP 15

*DATE OF SUBMISSION: ........./.............../........................*

*SUBMITTED TO: ........................................................................*

# ACKNOWLEDGEMENT:

We are grateful to the almighty God for this guidance and strength throughout this work our sincere thanks go to all who supported us and to every group member for their time and effort. Lastly, we acknowledge the sources and references that contributed to this report that have been mentiomed in this report.

# ABSTRACT:

This report presents a research on the algorithm development ,control structures and modules 5.1 for solving mathematical problems using various methods such as Euler, Runge-kutta , Scant, and New Ranpson method which helped us to plot graphs that compare the problems analytically by the different methods .The information used was obtained from Matlab textbooks and Youtube tutorials. We divided group members accordingly to achieve this tasks, different individuals were assigned different nurical methods in order to ease the work and save time.

# DEDICATION:

This report is dedicated to our lecturer, Mr. Masruka Benedicto who’s guidance and support have been instrumental in our learning journey. We also dedicate this work to group 15 members their unwavering encouragement throughout this project.



# DECLARATION:

We, group 15 members hereby declare that this report is our original work, carried out as part of the MATLAB course under the supervision of Mr. Maseruka Benedicto. All sources of information have been duly acknowledged, and this work has not been submitted elsewhere for academic project.

LUGUNGA TIMOTHY................

ODONG ERICK PERRY...................................................

KATUSIME JOEL..............................................................

NABUKWASI SHAKIRA...................................................

NAZIWA PATRICIA............................................................



NANDIJJA LAILA.................................................................



SIDENYA KEVIN ...........................................................…

BULUMA DANIEL………………………………………..

BAHEMUKA GODWINS…………………………………

NAMATA LILLIAN KIZZA………………………

# APPROVAL:

This report titled “The Algorithm Development control Structure and using modues 1-4 on practical real world” has been prepared and submitted by Group 15, in partial fulfillment of the requirements for the course. The work presented here is original and has been carried out under the guidance of the instructor.

We hereby approve this report as a reflection of the student’s effort in applying knowledge from Modules 1 to 4 to analyze, interpret, and visualize real-world data.

Table of Contents

[ACKNOWLEDGEMENT: i](#_Toc32556)

[ABSTRACT: ii](#_Toc6824)

[DEDICATION: iii](#_Toc6495)

[DECLARATION: iv](#_Toc2217)

[APPROVAL: v](#_Toc7086)

[CHAPTER 1: INTRODUCTION: １](#_Toc5733)

[1.1 Background: １](#_Toc18331)

[1.2 Historical Development: １](#_Toc23004)

[2.1 NUMBER ONE: ２](#_Toc21406)

[CHAPTER 3: RECOMMENDATIONS AND CONCLUSION ２６](#_Toc11826)

[3.1 REC0MMENDATION ２６](#_Toc31388)

[3.2 CONCLUSION ２６](#_Toc9696)

[3.3 REFERENCES ２６](#_Toc17341)

# **CHAPTER 1: INTRODUCTION:**

# 1.1 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.

# 1.2 Historical Development:

Initial 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.

Commercial Launch: MATLAB was officially launched as a commercial product in 1984 by MathWorks, a company founded by Moler along with Jack Little and Steve Bangert. This marked the transition from a simple calculator to a comprehensive programming environment. The software was reimplemented in C, enhancing its capabilities with the addition of user-defined functions, toolboxes, and graphical interfaces.

Expansion and Toolboxes: Over the years, MATLAB has expanded significantly. By the late 1980s, it had introduced several specialized toolboxes for various applications, including control systems and signal processing. The introduction of the Simulink environment further allowed users to model and simulate dynamic systems graphically.

Modern Enhancements: 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 2: STUDY COVERAGE:**

# **2.1 NUMBER ONE:**

**FLOW CHART**

END

yes

START

no

**NEWTON RAPHSON METHOD**

**EXAMPLE ONE**

Start timer

Set initial guess x0 = 2

Define function f(x) = x^3 - x - 2

Define derivative df(x) = 3\*x^2 - 1

Set tolerance err = 0.0005

Compute first Newton–Raphson update:

x = x0 - f(x0)/df(x0)

Compute error = |x - x0|

While error > tolerance:

Update x0 = x

Compute next Newton–Raphson update:

x = x0 - f(x0)/df(x0)

Compute error = |x - x0|

End loop

Record final root value x

Stop timer and record timetaken

Compute high-precision reference root xref using fzero

Save variables (x, timetaken, xref) into file "Newton-Raphson.mat"

tic;

x0 = 2;

f = @(x0) x0^3 - x0 - 2;

df = @(x0) (3\*x0^2)-1;

err = 0.0005;

x = x0-(f(x0)/df(x0))

x = 1.6364

calc\_err = abs(x-x0);

while calc\_err>err

x0 = x;

x = x0-(f(x0)/df(x0))

calc\_err = abs(x-x0);

end

x = 1.5304

x = 1.5214

x = 1.5214

timetaken = toc

timetaken = 0.1385

xref = fzero(@(x0) x0^3 - x0 - 2,1.6);

save("Newton-Raphson.mat","x","timetaken","xref");

**EXAMPLE TWO**

Start timer

Set initial guess x0 = 0

Define function f(x) = x^3 + x - 1

Define derivative df(x) = 3\*x^2 + 1

Set tolerance err = 0.0005

Compute first Newton–Raphson update:

x = x0 - f(x0)/df(x0)

Compute error = |x - x0|

While error > tolerance:

Update x0 = x

Compute next Newton–Raphson update:

x = x0 - f(x0)/df(x0)

Compute error = |x - x0|

End loop

Record final root value x

Stop timer and record timetaken

Compute high-precision reference root xref using fzero

Save variables (x, timetaken, xref) into file "Newton-Raphson\_2.mat"

tic;

x0 = 0;

f = @(x0) x0^3 + x0 - 1;

df = @(x0) (3\*x0^2)+1;

err = 0.0005;

x = x0-(f(x0)/df(x0))

x = 1

calc\_err = abs(x-x0);

while calc\_err>err

x0 = x;

x = x0-(f(x0)/df(x0))

calc\_err = abs(x0-x);

end

x = 0.7500

x = 0.6860

x = 0.6823

x = 0.6823

timetaken = toc

timetaken = 0.0540

xref = fzero(@(x0) x0^3 + x0 - 1,1);

save("Newton-Raphson\_2.mat","x","timetaken","xref");

**SECANT METHOD**

**EXAMPLE ONE**

Start timer

Set initial guesses:

x0 = 1

x1 = 2

Define function f(x) = x^3 - x - 2

Compute slope (approx derivative) between (x0, f(x0)) and (x1, f(x1)):

df = (f(x1) - f(x0)) / (x1 - x0)

Compute first update:

x = x1 - f(x1) / df

Compute error = |x - x1|

While error > tolerance (0.0005):

Update x1 = x

Recompute slope df = (f(x1) - f(x0)) / (x1 - x0)

Compute next update:

x = x1 - f(x1) / df

Update error = |x - x1|

End loop

Record final root value x

Stop timer and record timetaken

Compute high-precision reference root xref using fzero

Save variables (x, timetaken, xref) into file "Secant.mat"

tic;

x0 = 1;

x1 = 2;

f0 = @(x0) x0^3 - x0 - 2;

f = @(x1) x1^3 - x1 - 2;

df = @(x1) (f(x1)-f0(x0))/(x1-x0);

err = 0.0005;

x = x1-(f(x1)/df(x1))

x = 1.3333

calc\_err = abs(x-x1);

while calc\_err>err

x1 = x;

x = x1-(f(x1)/df(x1))

calc\_err = abs(x-x1);

end

x = 1.6429

x = 1.4606

x = 1.5565

x = 1.5026

x = 1.5318

x = 1.5157

x = 1.5245

x = 1.5197

x = 1.5223

x = 1.5209

x = 1.5217

x = 1.5212

timetaken = toc

timetaken = 0.0812

xref = fzero(@(x0) x0^3 - x0 - 2,1.6);

save("Secant.mat","x","timetaken","xref");

**EXAMPLE TWO**

Start timer

Set initial guesses:

x0 = 0

x1 = 1

Define function f(x) = x^3 + x - 1

Compute slope (approx derivative) between (x0, f(x0)) and (x1, f(x1)):

df = (f(x1) - f(x0)) / (x1 - x0)

Compute first update:

x = x1 - f(x1) / df

Compute error = |x - x1|

While error > tolerance (0.0005):

Update x1 = x

Recompute slope df = (f(x1) - f(x0)) / (x1 - x0)

Compute next update:

x = x1 - f(x1) / df

Update error = |x - x1|

End loop

Record final root value x

Stop timer and record timetaken

Compute high-precision reference root xref using fzero

Save variables (x, timetaken, xref) into file "Secant\_2.mat"

tic;

x0 = 0;

x1 = 1;

f0 = @(x0) x0^3 + x0 - 1;

f = @(x1) x1^3 + x1 - 1;

df = @(x1) (f(x1)-f0(x0))/(x1-x0);

err = 0.0005;

x = x1-(f(x1)/df(x1))

x = 0.5000

calc\_err = abs(x-x1);

while calc\_err>err

x1 = x;

x = x1-(f(x1)/df(x1))

calc\_err = abs(x-x1);

end

x = 0.8000

x = 0.6098

x = 0.7290

x = 0.6530

x = 0.7011

x = 0.6705

x = 0.6899

x = 0.6775

x = 0.6854

x = 0.6804

x = 0.6836

x = 0.6815

x = 0.6828

x = 0.6820

x = 0.6825

x = 0.6822

timetaken = toc

timetaken = 0.1129

xref = fzero(@(x0) x0^3 + x0 - 1,1);

save("Secant\_2.mat","x","timetaken","xref");

**BISECTOR METHOD**

**EXAMPLE ONE**

Start timer

Define function f(x) = x^3 - x - 2

Set initial interval [a, b] = [1, 2]

Set tolerance err = 0.0005

While (b - a) > err:

Compute midpoint c = (a + b) / 2

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

Root lies in [a, c], so set b = c

Else:

Root lies in [c, b], so set a = c

End loop

Set approximate root x = (a + b) / 2

Stop timer and record timetaken

Compute high-precision reference root xref using fzero

Save variables (x, timetaken, xref) into file "Bisection.mat"

tic;

f = @(x) x^3 - x - 2;

a = 1;

b = 2;

err = 5e-4;

while (b-a)>err

c=(a+b)/2;

if f(a)\*f(c)<0

b = c;

else

a = c;

end

end

x = (a+b)/2

x = 1.5212

timetaken = toc

timetaken = 0.0491

xref = fzero(@(x) x^3 - x - 2,1.6);

save("Bisection.mat","x","timetaken","xref");

**EXAMPLE TWO**

Start timer

Define function f(x) = x^3 + x - 1

Set initial interval [a, b] = [0, 1]

Set tolerance err = 0.0005

While (b - a) > err:

Compute midpoint c = (a + b) / 2

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

Root lies in [a, c], so set b = c

Else:

Root lies in [c, b], so set a = c

End loop

Set approximate root x = (a + b) / 2

Stop timer and record timetaken

Compute high-precision reference root xref using fzero

Save variables (x, timetaken, xref) into file "Bisection\_2.mat"

tic;

f = @(x) x^3 + x - 1;

a = 0;

b = 1;

err = 5e-4;

while (b-a)>err

c=(a+b)/2;

if f(a)\*f(c)<0

b = c;

else

a = c;

end

end

x = (a+b)/2

x = 0.6824

timetaken = toc

timetaken = 0.1038

xref = fzero(@(x) x^3 - x - 2,1);

save("Bisection\_2.mat","x","timetaken","xref");

**FIXED POINT ITERATION METHOD**

**EXAMPLE ONE**

Start timer

Define function f(x) = x^3 - x - 2

Rearrange into x = g(x), e.g., g(x) = (x + 2)^(1/3)

Set initial guess x0

Set tolerance err = 0.0005

Initialize iteration list x\_hist with x0

Compute first iterate:

x = g(x0)

Compute error = |x - x0|

While error > tolerance:

Update x0 = x

Compute next iterate:

x = g(x0)

Compute error = |x - x0|

Append x to iteration list

End loop

Record final root value x

Stop timer and record timetaken

Compute high-precision reference root xref using fzero

Save variables (x, timetaken, xref) into file "Fixed\_point\_iteration.mat"

tic;

f = @(x) x^3 - x - 2;

g = @(x) (x+2).^(1/3);

x = 1;

err = 5e-4;

while true

x1 = g(x);

if abs(x1-x)<err

break

end

x = x1

end

x = 1.4422

x = 1.5099

x = 1.5197

x = 1.5211

timetaken = toc

timetaken = 0.0500

xref = fzero(@(x) x^3 - x - 2,1.6);

save("Fixed\_point\_iteration.mat","x","timetaken","xref");

**EXAMPLE TWO**

Start timer

Define function f(x) = x^3 + x - 1

Rearrange into x = g(x), e.g., g(x) = (1 - x)^(1/3)

Set initial guess x0

Set tolerance err = 0.0005

Initialize iteration list x\_hist with x0

Compute first iterate:

x = g(x0)

Compute error = |x - x0|

While error > tolerance:

Update x0 = x

Compute next iterate:

x = g(x0)

Compute error = |x - x0|

Append x to iteration list

End loop

Record final root value x

Stop timer and record timetaken

Compute high-precision reference root xref using fzero

Save variables (x, timetaken, xref) into file "Fixed\_point\_iteration\_2.mat"

tic;

f = @(x) x^3 + x - 1;

g = @(x) (1-x).^(1/3);

x = 0.5;

err = 5e-4;

while true

x1 = g(x);

if abs(x1-x)<err

break

end

x = x1

end

x = 0.7937

x = 0.5909

x = 0.7424

x = 0.6363

x = 0.7138

x = 0.6590

x = 0.6986

x = 0.6704

x = 0.6907

x = 0.6763

x = 0.6866

x = 0.6792

x = 0.6845

x = 0.6807

x = 0.6835

x = 0.6815

x = 0.6829

x = 0.6819

x = 0.6826

x = 0.6821

timetaken = toc

timetaken = 0.0755

xref = fzero(@(x) x^3 + x - 1,1);

save("Fixed\_point\_iteration\_2.mat","x","timetaken","xref");

**GROUPING DATA FOR PLOTTING**

Define list of methods = [Newton-Raphson, Secant, Bisection, Fixed Point]

Define list of problems = [Example1, Example2]

Initialize empty table for computation times

For each problem in problems:

Open New figure

For each method in methods:

Load the saved data file for this method and problem

Extract iterates (x), true root, and time

Compute error at each iteration = |x - true root|

Plot error vs iteration

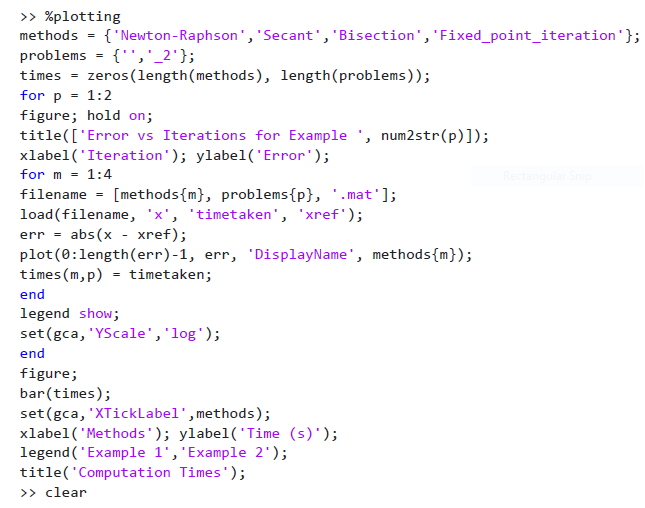
Store time in the times table

Show legend and set Y axis to log scale

Open new figure

Make bar chart of computation times for all methods and both problems

Label axes and add title



**GRAPH COMPARING ANALYTICAL SOLUTIONS TO SOLUTIONS OBTAINED WITH COMPUTATION TIME**

For each method and problem:

Plot error vs iteration curve

Use log scale for error axis

After looping:

Create bar chart comparing times

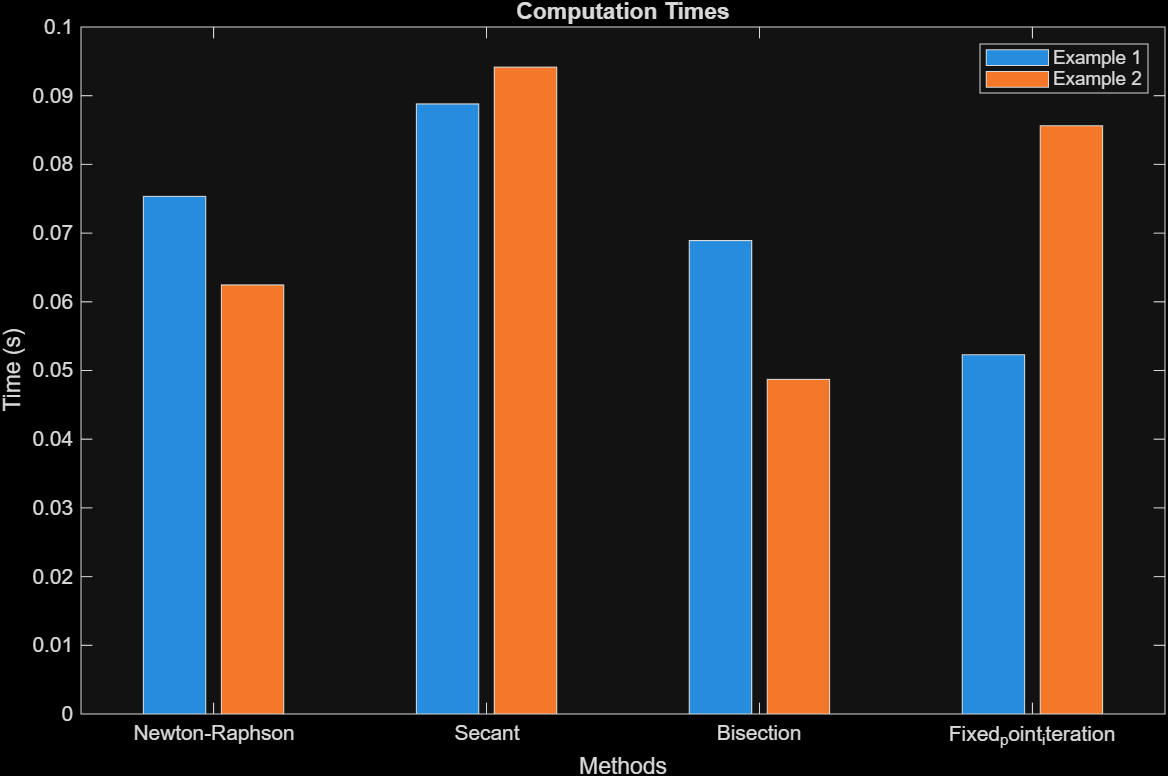
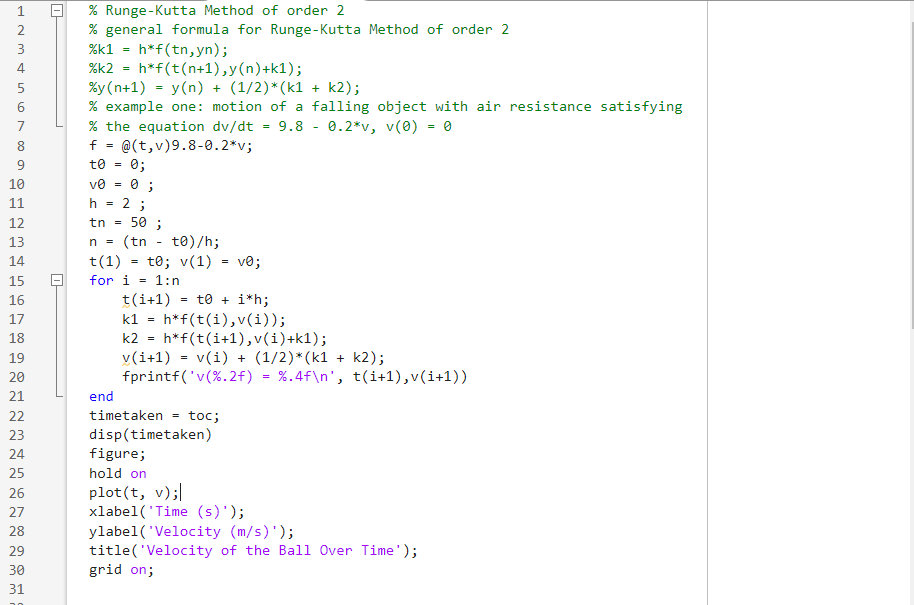


Table 1computation time

Part b

Runge-Kutta method



v(2.00) = 15.6800

v(4.00) = 26.3424

v(6.00) = 33.5928

v(8.00) = 38.5231

v(10.00) = 41.8757

v(12.00) = 44.1555

v(14.00) = 45.7057

v(16.00) = 46.7599

v(18.00) = 47.4767

v(20.00) = 47.9642

v(22.00) = 48.2956

v(24.00) = 48.5210

v(26.00) = 48.6743

v(28.00) = 48.7785

v(30.00) = 48.8494

v(32.00) = 48.8976

v(34.00) = 48.9304

v(36.00) = 48.9526

v(38.00) = 48.9678

v(40.00) = 48.9781

v(42.00) = 48.9851

v(44.00) = 48.9899

v(46.00) = 48.9931

v(48.00) = 48.9953

v(50.00) = 48.9968

4.4456e+03

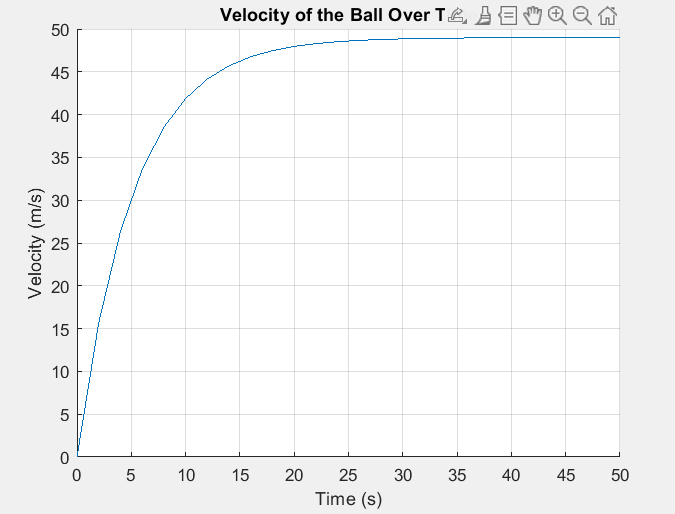
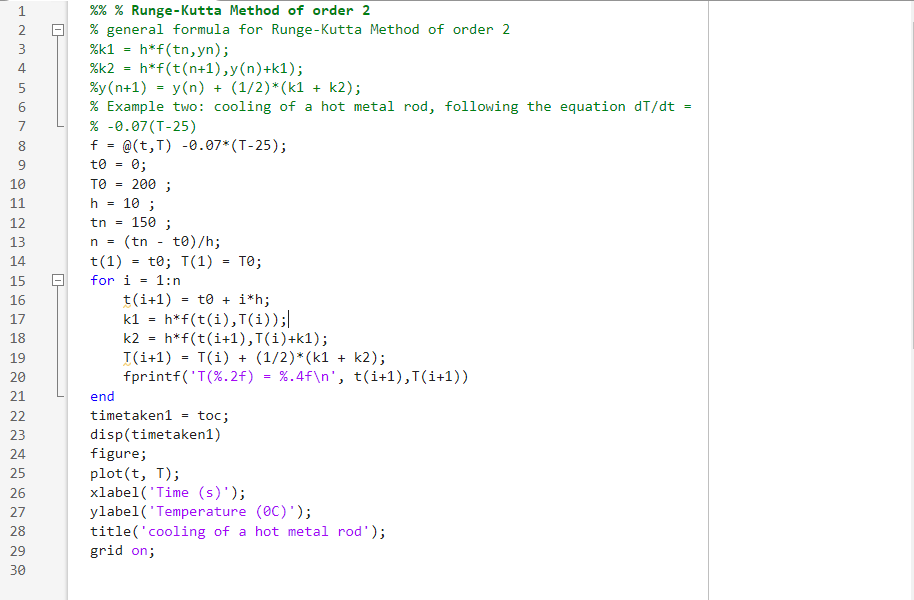


Table 2velocity of ball over time



T(10.00) = 120.3750

T(20.00) = 76.9794

T(30.00) = 53.3288

T(40.00) = 40.4392

T(50.00) = 33.4143

T(60.00) = 29.5858

T(70.00) = 27.4993

T(80.00) = 26.3621

T(90.00) = 25.7423

T(100.00) = 25.4046

T(110.00) = 25.2205

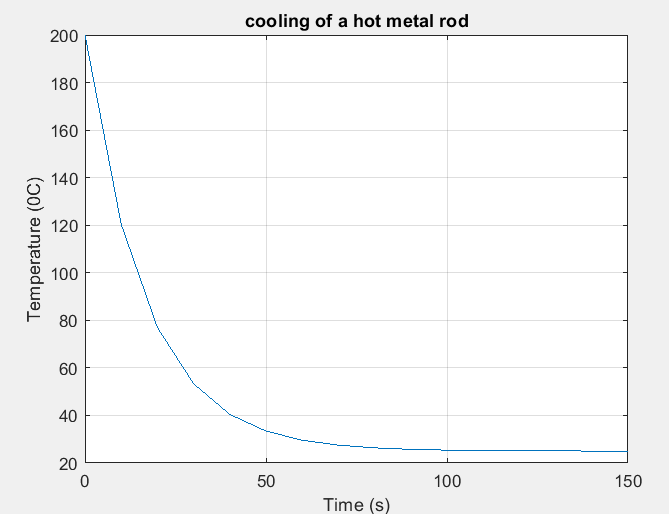
T(120.00) = 25.1202

T(130.00) = 25.0655

T(140.00) = 25.0357

T(150.00) = 25.0195

timetaken1 = 5.1043e+03



**EULER METHOD**

PART (A);

%cooling function:T=25 +(85-25)\*exp(-o.1\*t)

f = @(t) 25 + ( 85-25) \*exp(-0.1\*t) - 30;

% Find when T=30

df = @(t) -(85-25)\*0.1\*exp(-0.1\*t);

PART (B);

% Newton's law of cooling

k = 0.1; T\_env = 25; T0 = 85;

t0 = 0; tf = 50; h= 1;

t=t0:h:tf;

% Exact analytical solution

T-exact = T\_env + (TO -T\_env)\*exp(-k.\*t);

% Euler method

T\_exact = zeros(size(t)); T\_eular(1) = T0;

for i = 1:length(t)-1

T-euler(i+1) = T-euler(i) + h\*(-k\*(T\_euler(i) - T\_env));

end

% RK4 Method

T\_rk4 = zeros(size(t)); T-rk4(1) = T0;

for i = 1:length(t) -1

k1 = -k\*(T\_rk4(i) - T\_env);

k2 = -k\*((T\_rk4(i) + 0.5\*h\*k1) - T\_env);

k3 = -k\*((T\_rk4(i) + 0.5\*h\*k2) - T\_env);

k4 = -k\*((T\_rk4(i) + h\*k3) - T\_env);

T-rk4(i+1) = T-rk4(i) + (h/6)\*(k1 + 2\*k2 + 2\*k3 +4);

end

% Display Results for part B

fprinntf('\n--- DIFFRENTIAL EQUATION RESULTS ---\n');

fprintf('Euler Time: %.6f s\n', euler\_time);

fprintf('Runge-Kutta 4th order Time: %.6f s\n', rk4\_time);

% Plot comparison

figure;

plot(t, y\_exact, 'k-','LineWdth', 2);

hold on

plot(t, y\_euler,'r--','LineWidth',1.5);

plot(t, y\_rk4,'b-.','LineWidth',1.5);

xlabel('Time (t)');

ylabel('population(y)');

legend('Exact solution', 'Euler Method','RK4 Method');

title('Comparison of Numerical Solutions for Population Growth');

grid on;

%Compute and Display Error

error\_euler = abs(y\_exact - y\_euler);

error\_rk4 = abs(y\_exact - y\_rk4);

figure;

plot(t, error\_euler,'r-- ', t, error\_rk4, 'b-','LineWdth', 1.5);

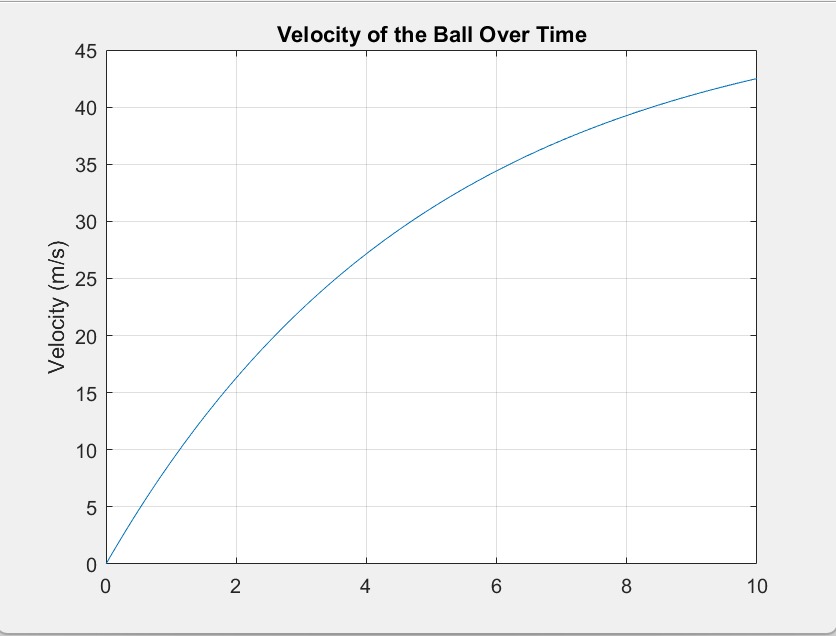
xlabel('Time (t)');

ylabel('Absolute Error');

legend('Euler Error', 'RK4 Error');

title('Error Comparison Between Euler and RK4 Methods');

grid on



% Parameters

T\_initial = 200; % Initial temperature of the rod

T\_env = 25; % Environment temperature

k = 0.07; % Cooling constant

dt = 0.1; % Time step

t\_final = 60; % Total time for simulation

N = t\_final / dt; % Number of time steps

% Preallocate arrays for time and temperature

time = zeros(1, N);

temperature = zeros(1, N);

% Initial conditions

temperature(1) = T\_initial;

time(1) = 0;

% Start computation time

tic;

% Euler method for solving the differential equation

for n = 1:N-1

dT = -k \* (temperature(n) - T\_env); % Calculate the change in temperature

temperature(n+1) = temperature(n) + dT \* dt; % Update temperature

time(n+1) = time(n) + dt; % Update time

end

% End computation time

elapsed\_time = toc;

% Plotting the results

figure;

plot(time, temperature, 'LineWidth', 2);

xlabel('Time (seconds)');

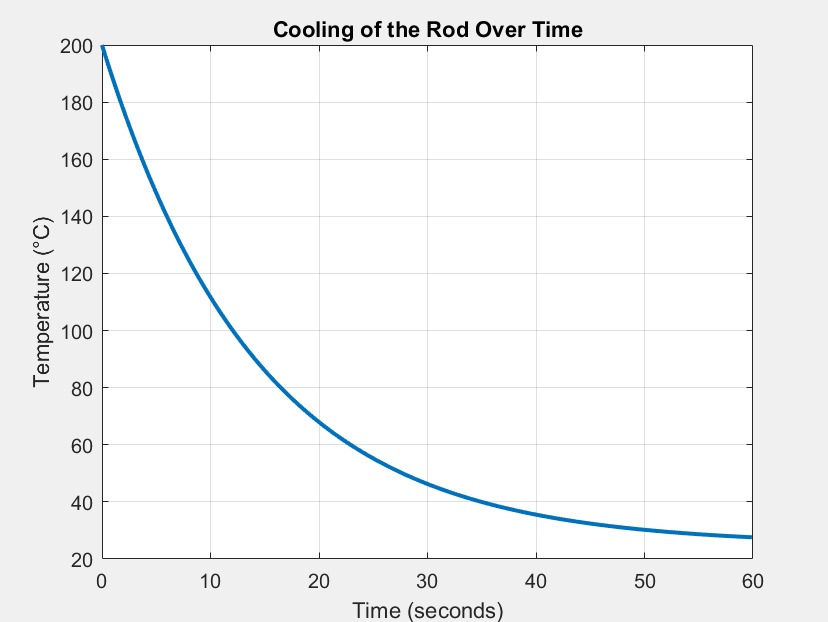
ylabel('Temperature (°C)');

title('Cooling of the Rod Over Time');

grid on;

% Display computation time

fprintf('Computation time: %.4f seconds\n', elapsed\_time);

****

# CHAPTER 3: RECOMMENDATIONS AND CONCLUSION

### 3.1 REC0MMENDATION

Use multiple methods on the same problem to compare accuracy.

Apply higher-order methods like Runge-Kutta for precision.

Use MATLAB for implementation and visualization.

Extend applications to more complex real-world systems.

### 3.2 CONCLUSION

Numerical methods such as Newton-Raphson, Secant, Euler, and Runge-Kutta provide practical solutions when analytical methods are difficult. While simpler methods are easier to apply, higher-order ones give faster and more accurate results. Visualization confirmed the accuracy and limitations of each approach in real-world problems.

### 3.3 REFERENCES

Chapra, S. C., & Canale, R. P. (2015). Numerical Methods for Engineers. McGraw-Hill.

Burden, R. L., & Faires, J. D. (2010). Numerical Analysis. Brooks/Cole.

Mathews, J. H., & Fink, K. D. (2004). Numerical Methods Using MATLAB. Pearson.

MathWorks. (2025). MATLAB Documentation