FACULTY OF ENGINEERING AND TECHONOLOGY

COMPUTER PROGRAMMING

ASSIGNMENT REPORT ON HOW DIFFERENT NUMERICAL METHODS IN LINE WITH GRAPHICAL USER INTERFACE CAN SOLVE REAL LIFE PROBLEMS

BY GROUP SIX

PRESENTED TO MR. MASERUKA BENEDICTO

DATE OF SUBMISSION;

11TH NOVEMBER,2025.

GROUP SIX MEMBERS

NAME	COURSE	REG NUMBER
OPIO SAM ODWORI	WAR	BU/UP/2024/4454
MUWAZA DEOGRATIUS	PTI	BU/UP/2024/5398
KATUNGUKA EMMANUEL	WAR	BU/UP/2024/1030
JOAN RACHAEL LAGU	WAR	BU/UP/2024/1028
KISAKYE PRISCILA PATIANCE	WAR	BU/UP/2024/1034
AYOO LINDA	AMI	BU/UP/2024/5097
WAMBASA GEOFERY	WAR	BU/UP/2024/3259
NAMANYA PRECIOUS	AMI	BU/UP/2024/5255
NANGOLI DERRIK	AMI	BU/UP/2024/3736
APUDA CECILIA LINDI	WAR	BU/UP/2024/3248

DECLARATION

We group six members declare that all the work embodied in this report is of our own efforts and that it was not duplicated from any.

NAMES	SIGNATURE
MANGOLI DERRICK	
OPIO SAM ODWORI	
WAMBASA GEOFREY	
KISAKYE PRISCILA PATIENC	
NAMANYA PRECIOUS	
AYOO LINDA	
KATUNGUKA EMMANUEL	
JOAN RACHEAL LAGU	
APUDA CECILIA LINDI	
MUWAZA DEOGRATIOU	

ACKNOWLEDGEMENT
We thank the Almighty God for the gift of life, knowledge and constant guidance throughout the course of this assignment. We extend our sincere gratitude to our beloved parents for their support in our academics and also our lecturer Mr. Maseruka Benedicto (HOD) for his continuous efforts in ensuring we learn computer programming thoroughly.

We finally thank our fellow group six members for their cooperation and active participation in this assignment through our group leader Wambasa Geofrey.

APPROVAL.

This is to confirm that this report has been prepared and presented by group six without duplication but research.

Date ……

Signature…….

DEDICATION

We dedicate this report to all group SIX members, who worked tirelessly to ensure that its completed.

METHODOLOGY

In this assignment, we utilized our knowledge of numerical methods and graphical user interface in module4 and 5.

We were able to use the Newton-Raphson method, the secant method to find solutions to functions, solve differential equations and also solve real world.

EXECUTIVE SUMMARRY

- ✧ Differential method
- ✧ Integral problem
- ✧ Newton Raphson method
- ✧ Newton graphical user Interface app

TABLE OF CONTENTS

Catalog

CHAPTER ONE
INTRODUCTION

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.

Historical Background

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.

STUDY METHODOLOGY.

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 YouTube, reading the modules and consultations from other continuing students especially those in year three and four.

CHAPTER TWO:

Question

Apply the numerical Methodes with graphical user interface to show how they solve real life problems

DIFFERENTIAL PROBLEM

```
classdef DifferentialProblem < NumericalMethodBase
% Subclass for Newton-Raphson and Secant methods.
methods
function obj = DifferentialProblem()
% Inheritance: Automatically calls NumericalMethodBase constructor
obj = obj@NumericalMethodBase();
end

function f_handle = defineFunction(obj)
% Defines the function f(Q) = p*Q - (fc - vc*Q) (p*A changed to p*Q for Q)
% p=25, fc=1000, vc=15
% Encapsulation: Accessing protected properties
p_val = obj.p;
fc_val = obj.fc;
vc_val = obj.vc;

% Function handle: Q is the variable
f_handle = @(Q) p_val * Q - (fc_val - vc_val * Q);
end
function df_handle = defineDerivative(obj)
% Derivative of f(Q) is f'(Q) = p + vc
df_handle = @(Q) obj.p + obj.vc;
end
% Polymorphism: Implements the abstract method
function solveProblem(obj)
f = obj.defineFunction();
df = obj.defineDerivative();
% Newton-Raphson Method (NRM)
Q0_nrm = 10; % Initial guess
```

```
Q_nrm = obj.newtonRaphson(f, df, Q0_nrm);
fprintf('NRM Result (Q): %.4f\n', Q_nrm);

% Secant Method
Q0_sec = 0;
Q1_sec = 10; % Initial guesses
Q_sec = obj.secantMethod(f, Q0_sec, Q1_sec);
fprintf('Secant Result (Q): %.4f\n', Q_sec);
end
end
methods (Access = public)
% Encapsulation: Helper methods are kept private
function Q_next = newtonRaphson(~, f, df, Q0)
% High-level implementation of NRM
Q_next = Q0 - f(Q0) / df(Q0);
end
function Q_next = secantMethod(~, f, Q0, Q1)
% High-level implementation of Secant
Q_next = Q1 - f(Q1) * (Q1 - Q0) / (f(Q1) - f(Q0));
end
end
End
```

INTEGRAL PROBLEM

```
classdef Intergrationproblem < NumericalMethodBase
% Subclass for Euler and Runge-Kutta methods.

methods
function obj = Intergrationproblem()
% Inheritance: Calls NumericalMethodBase constructor
obj = obj@NumericalMethodBase();
end
```

```
function dydt_handle = defineFunction(obj)
% Defines the ODE: dA/dt = r*A (or y' = r*y)
% r=0.05
r_val = obj.r;
% Function handle: t is time, A is the variable (y)
dydt_handle = @(t, A) r_val * A;
end
% Polymorphism: Implements the abstract method
function solveProblem(obj)
dydt = obj.defineFunction();
% Encapsulation: Accessing protected properties
t_span = obj.t_span;
y0 = obj.Ao;
% Euler Method
h = 1; % Step size
[t_eul, y_eul] = obj.eulerMethod(dydt, t_span, y0, h);
fprintf('Euler Result at t=%.0f: %.4f\n', t_eul(end), y_eul(end));

% Runge-Kutta Method (RK4)
[t_rk, y_rk] = obj.rungeKutta(dydt, t_span, y0, h);
fprintf('Runge Kutta Result at t=%.0f: %.4f\n', t_rk(end), y_rk(end));
% (Note: The exact solution is 1000*exp(0.05*10) = 1648.7213)
end
end
methods (Access = public)
% Encapsulation: Helper methods are kept private
function [t_out, y_out] = eulerMethod(~, dydt, t_span, y0, h)
% High-level implementation of Euler
t_out = t_span(1):h:t_span(2);
y_out = zeros(size(t_out));
y_out(1) = y0;
for i = 1:(length(t_out)-1)
y_out(i+1) = y_out(i) + h * dydt(t_out(i), y_out(i));
end
```

```
end

function [t_out, y_out] = rungeKutta(~, dydt, t_span, y0, h)
% High-level implementation of RK4
t_out = t_span(1):h:t_span(2);
y_out = zeros(size(t_out));
y_out(1) = y0;
for i = 1:(length(t_out)-1)
t = t_out(i);
y = y_out(i);
k1 = dydt(t, y);
k2 = dydt(t + h/2, y + h*k1/2);
k3 = dydt(t + h/2, y + h*k2/2);
k4 = dydt(t + h, y + h*k3);
y_out(i+1) = y + (h/6) * (k1 + 2*k2 + 2*k3 + k4);
end
end
end
End
```

NEWTON RAPHSON BASE

```
classdef (Abstract) NumericalMethodBase < handle
% NUMERICALMETHODBASE Abstract base class for all numerical methods.
properties (Access = public)
% Encapsulation: Protected properties for shared data
p; % Price (for root-finding)
fc; % Fixed Cost (for root-finding)
vc; % Variable Cost (for root-finding)
Ao; % Initial Amount (for ODEs)
r; % Rate (for ODEs)
t_span; % Time span (for ODEs)
end
methods
```

```
function obj = NumericalMethodBase()
% Constructor: Initialize properties (Encapsulation)
obj.p = 25;
obj.fc = 1000;
obj.vc = 15;
obj.Ao = 1000;
obj.r = 0.05;
obj.t_span = [0, 10];
end
end
methods (Abstract)
% Abstraction: All subclasses MUST implement these methods (Polymorphism)
solveProblem(obj);
defineFunction(obj);
end
end
```

NEWTON RAPHSON GRAPHICAL USER INTERFACE APP

```
% NumericalSolverGUIApp.m
function NumericalSolverGUIApp()
% Main GUI setup function
% Check if the required class files are in the path
required_classes = {'NumericalMethodBase', 'DifferentialProblem',
'Intergrationproblem'};
for i = 1:length(required_classes)
if ~exist(required_classes{i}, 'class')
error('Class file %s.m not found. Ensure all class files are in the MATLAB path.',
required_classes{i});
end
end


% 1. Create the Main Figure
fig = uifigure('Name', 'Numerical Method Solver', 'Position', [100 100 900 650]);
% 2. Initialize Solver Class Instances (Crucial for property management)
try
```

```
dp_solver = DifferentialProblem(); % Root-finding solver (p=25, fc=1000, vc=15)
ip_solver = Intergrationproblem(); % ODE solver (Ao=1000, r=0.05, t_span=[0, 10])
catch ME
uialert(fig, ['Failed to initialize solver classes. Check that '...
'NumericalMethodBase.m, DifferentialProblem.m, and Intergrationproblem.m '...
'are in the path and correctly defined.'], 'Class Initialization Error', 'Icon',
'error');
rethrow(ME);
end

% 3. Create UI Components
% Panel for Method Selection and Inputs
inputPanel = uipanel(fig, 'Title', 'Select Method and Input Parameters', ...
'Position', [20 20 280 610]);
% Method Selection Dropdown
uilabel(inputPanel, 'Text', 'Select Method:', 'Position', [15 560 100 22]);
methodDropDown = uidropdown(inputPanel, 'Items', ...
{'Newton-Raphson', 'Secant', 'Euler', 'Runge-Kutta (RK4)'}, ...
'Position', [15 530 250 22], ...
'ValueChangedFcn', @methodSelectionCallback);
% Dynamic Input Fields (Labels and Edits)
inputData.labels = cell(1, 4);
inputData.edits = cell(1, 4);
start_y = 480;
y_step = 35;

for i = 1:4
inputData.labels{i} = uilabel(inputPanel, 'Text', 'Input Name:', ...
'Position', [15 start_y - (i-1)*y_step 120 22], 'Visible', 'off');
inputData.edits{i} = uieditfield(inputPanel, 'numeric', 'Value', 0, ...
'Position', [145 start_y - (i-1)*y_step 120 22], 'Visible', 'off');
end

% Set initial visibility and values based on the default selected method
```

```
setMethodInputs('Newton-Raphson', inputData);
% Solve Button
uibutton(inputPanel, 'Text', 'Solve Problem', 'Position', [15 150 250 35], ...
'ButtonPushedFcn', @solveButtonCallback);
% Output Field
uilabel(inputPanel, 'Text', 'Output Result:', 'Position', [15 110 100 22]);
outputArea = uitextarea(inputPanel, 'Value', 'Results will appear here.', ...
'Position', [15 20 250 80], 'Editable', 'off');

% Axis for Plotting (FIX APPLIED HERE)
plotAxes = uiaxes(fig, 'Position', [320 120 550 510]);
plotAxes.Title.String = 'Numerical Solution Plot';
plotAxes.XLabel.String = 'Time (t) or Iteration Index';
plotAxes.YLabel.String = 'Result (Q or A)';
grid(plotAxes, 'on');

% Save Plot Button
uibutton(fig, 'Text', 'Save Plot as PNG', 'Position', [320 50 150 35], ...
'ButtonPushedFcn', @savePlotButtonCallback);
% Store handles, data, and solver objects in the figure's UserData
fig.UserData = struct('methodDropDown', methodDropDown, 'inputData', inputData, ...
'outputArea', outputArea, 'plotAxes', plotAxes, ...
'dp_solver', dp_solver, 'ip_solver', ip_solver);

% Callback Functions

% Callback for method selection dropdown
function methodSelectionCallback(src, ~)
selectedMethod = src.Value;
setMethodInputs(selectedMethod, inputData);
end

% Helper function to set the labels, visibility, and default values for inputs
function setMethodInputs(method, data)
```

```
% Hide all fields first
for k = 1:4
data.labels{k}.Visible = 'off';
data.edits{k}.Visible = 'off';
end
% Set fields based on method
switch method
case 'Newton-Raphson'
data.labels{1}.Text = 'Initial Guess Q0:';
data.edits{1}.Value = 10;
data.labels{1}.Visible = 'on'; data.edits{1}.Visible = 'on';
case 'Secant'
data.labels{1}.Text = 'Initial Guess Q0:';
data.edits{1}.Value = 0;
data.labels{2}.Text = 'Second Guess Q1:';
data.edits{2}.Value = 10;
data.labels{1}.Visible = 'on'; data.edits{1}.Visible = 'on';
data.labels{2}.Visible = 'on'; data.edits{2}.Visible = 'on';
case {'Euler', 'Runge-Kutta (RK4)'}
data.labels{1}.Text = 'Initial Amount (A0):';
data.edits{1}.Value = 1000;
data.labels{2}.Text = 'Time Span End (t_end):';
data.edits{2}.Value = 10;
data.labels{3}.Text = 'Step Size (h):';
data.edits{3}.Value = 1;
% Rate (r)
data.labels{4}.Text = 'Rate (r):';
data.edits{4}.Value = 0.05;

for k = 1:4
data.labels{k}.Visible = 'on';
data.edits{k}.Visible = 'on';
end
end
```

```
end

% Callback for the Solve Button
function solveButtonCallback(~, ~)
% Clear previous plot and output
cla(plotAxes);
outputArea.Value = 'Processing...';
userData = fig.UserData;
selectedMethod = userData.methodDropDown.Value;
inputEdits = userData.inputData.edits;
try
switch selectedMethod
case 'Newton-Raphson'
% Get input
Q0 = inputEdits{1}.Value;
dp = userData.dp_solver;

f = dp.defineFunction();
df = dp.defineDerivative();
max_iter = 10;
tolerance = 1e-6;
Q = zeros(1, max_iter + 1);
Q(1) = Q0;
% Iterative use of the single-step helper method
for inter = 1:max_iter
Q(inter+1) = dp.newtonRaphson(f, df, Q(inter));
if abs(Q(inter+1) - Q(inter)) < tolerance
Q = Q(1:inter+1);
break;
end
end
final_Q = Q(end);
outputArea.Value = sprintf('NRM Result (Q): %.6f\n(Converged in %d steps)',
final_Q, length(Q)-1);
```

```
plot(plotAxes, 0:length(Q)-1, Q, '-o', 'DisplayName', 'NRM Iterations');
% FIX: Use .Title.String
plotAxes.Title.String = 'Newton-Raphson Convergence';
plotAxes.XLabel.String = 'Iteration';
plotAxes.YLabel.String = 'Q value';
case 'Secant'
% Get inputs
Q0 = inputEdits{1}.Value;
Q1 = inputEdits{2}.Value;
dp = userData.dp_solver;

f = dp.defineFunction();

max_iter = 10;
tolerance = 1e-6;
Q = zeros(1, max_iter + 2);
Q(1) = Q0;
Q(2) = Q1;
% Iterative use of the single-step helper method
for inter = 1:max_iter
Q(inter+2) = dp.secantMethod(f, Q(inter), Q(inter+1));
if abs(Q(inter+2) - Q(inter+1)) < tolerance
Q = Q(1:inter+2);
break;
end
end
final_Q = Q(end);

outputArea.Value = sprintf('Secant Result (Q): %.6f\n(Converged in %d steps)',
final_Q, length(Q)-2);
plot(plotAxes, 0:length(Q)-1, Q, '-o', 'DisplayName', 'Secant Iterations');
% FIX: Use .Title.String
plotAxes.Title.String = 'Secant Method Convergence';
plotAxes.XLabel.String = 'Iteration';
```

```
plotAxes.YLabel.String = 'Q value';
case {'Euler', 'Runge-Kutta (RK4)'}
% Get inputs and update class properties
A0 = inputEdits{1}.Value;
t_end = inputEdits{2}.Value;
h = inputEdits{3}.Value;
r = inputEdits{4}.Value;
ip = userData.ip_solver;
% Crucial step: Update the class properties based on user input
ip.Ao = A0; % Initial Amount
ip.r = r; % Rate
ip.t_span = [0, t_end]; % Time Span

dydt = ip.defineFunction();
t_span = ip.t_span; % Get the updated t_span

if strcmp(selectedMethod, 'Euler')
% Solve using Euler method
[t_eul, y_eul] = ip.eulerMethod(dydt, t_span, ip.Ao, h);
final_A = y_eul(end);
% Calculate exact solution for comparison
exact_A = ip.Ao * exp(ip.r * t_end);

outputArea.Value = sprintf('Euler Result at t=%.0f: %.6f\nExact Value: %.6f',
t_end, final_A, exact_A);
plot(plotAxes, t_eul, y_eul, '-o', 'DisplayName', 'Euler Solution');
% FIX: Use .Title.String
plotAxes.Title.String = 'Euler Method Solution';
else % Runge-Kutta (RK4)
% Solve using RK4 method
[t_rk, y_rk] = ip.rungeKutta(dydt, t_span, ip.Ao, h);
final_A = y_rk(end);

% Calculate exact solution for comparison
```

```
exact_A = ip.Ao * exp(ip.r * t_end);
outputArea.Value = sprintf('RK4 Result at t=%.0f: %.6f\nExact Value: %.6f', t_end,
final_A, exact_A);
plot(plotAxes, t_rk, y_rk, '-o', 'DisplayName', 'RK4 Solution');

% FIX: Use .Title.String
plotAxes.Title.String = 'Runge-Kutta (RK4) Solution';
end
% Plot the exact solution for comparison (only for ODEs)
t_exact = linspace(0, t_end, 100);
y_exact = ip.Ao * exp(ip.r * t_exact);
hold(plotAxes, 'on');
plot(plotAxes, t_exact, y_exact, '--r', 'DisplayName', 'Exact Solution');
hold(plotAxes, 'off');
legend(plotAxes, 'show', 'Location', 'northwest');
% FIX: Use .XLabel.String and .YLabel.String
plotAxes.XLabel.String = 'Time (t)';
plotAxes.YLabel.String = 'Amount (A)';
end
catch ME
% Display error message if calculation fails
outputArea.Value = sprintf('Error in calculation:\n%s', ME.message);
cla(plotAxes);
end
end

% Callback for the Save Plot Button
function savePlotButtonCallback(~, ~)
userData = fig.UserData;
plotAxes = userData.plotAxes;
if isempty(plotAxes.Children)
uialert(fig, 'No plot to save. Run a solver first.', 'Plot Error', 'Icon',
'warning');
return;
end
```

```
% Get file name from user

[filename, filepath] = uiputfile('*.png', 'Save Plot As',
'Numerical_Solution.png');

if filename ~= 0

% Create a temporary figure to copy the axes content

tempFig = figure('Visible', 'off');

tempAx = copyobj(plotAxes, tempFig);

set(tempAx, 'Units', 'normalized', 'Position', [0.1 0.1 0.8 0.8]);

% Save the plot

saveas(tempFig, fullfile(filepath, filename));

close(tempFig); % Close the temporary figure

uialert(fig, ['Plot saved successfully to: ' fullfile(filepath, filename)], 'Save
Successful', 'Icon', 'success');

end

end


end
```

CHAPTER THREE:

3.1 CHALLENGES

- Limited time given for the assignment to be completed.
- Referencing errors at times made the work hectic
- Lack of concentration due to the different course units being handled at the same time

3.2 RECOMMENDATIONS

- We recommend that the lecturer to always give us ample time to complete the assignment.

3.3 CONCLUSION AND LEARNING EXPERIENCE

Upon assignment completion, we really appreciated the MATLAB especially from the introduction part to the coverage. This embedded a real-life application of the software into the different engineering aspects. We gained a deeper rhythm on NUMERICAL METHODS and also the entire coverage of module 4-5. This experience was of utmost importance to all of us.