FACULTY OF ENGINEERING AND TECHNOLOGY

A REPORT ABOUT DESIGNING GRAPHIC USER INTERFACE FOR THE NUMERICAL METHODS.

COURSE UNIT: COMPUTER PROGRAMMING

LECTURER: Mr. MASERUKA BENEDICTO

SUBMITTED BY

GROUP 13

DATE OF SUBMISSION: ………………………………

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

NAME		SIGNATURE
1.	NASASIRA AUSTINE	…………………………
2.	KINTU GADAFI	………………………..
3.	ABUROT MARTHA	…………………………
4.	MUGUME MATHIAS	………………………….
5.	ACHAM LOYCE	………………………….
6.	AINAMATSIKO JEREMIAH	………………………….
7.	NAMUWAWU PATRICIA	…………………………..
8.	BRAVO GIFT	…………………………..
9.	NYEMERA ANNA GRACE	…………………………..
10.	BARASA ELVIS	…………………………..

APPROVAL

We are presenting this report which has been written and produced under our efforts. We carried out research on designing Graphical User Interfaces of numerical methods application for solving computational problems for simplicity.

ACKNOWLEDGEMENT

First and foremost, we would like to thank the Almighty God for giving us the strength to carry on with our research as Group 13. 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.

Finally, we would like to express our gratitude to all the sources and references that have been cited in this report

DEDICATION.

We dedicate this report to all the individuals especially **Group 13** members, who have been there for us in the process of formulating and producing this report.

To our lecturer Mr. Maseruka Benedicto whose guidance and expertise have been invaluable, your mentorship and insightful feedback have shaped our understanding.

ABSTRACT

We started our research from 30th September, 2025 in the university library with division of work for intense research in the different sections, where were obtained the best ways of how to do the assignment and that are used in the engineering field to solve problems.

Information we used was mainly obtained from YouTube tutorials. We carried out research on different concepts on designing the Graphical User Interface.

Contents

LIST OF ACRONYMS/ ABBREVIATIONS.

1. GUI	Graphical user interface.
2. MATLAB	Matrix Laboratory.

CHAPTER ONE: 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 Designing a Graphical User Interface (GUI)

Designing a GUI involves creating an intuitive, visually appealing interface that allows users to interact seamlessly with a digital product. It requires a combination of technical skills, creativity, and adherence to design principles to ensure usability and accessibility.

1.3 Key Principles of GUI Design

Consistency is crucial in GUI design. Maintaining uniform visual elements, layouts, and navigation across the interface ensures users can easily understand and navigate the product. For example, consistent typography and color schemes help users identify content types, while predictable layouts improve usability.

Simplicity enhances user engagement by reducing cognitive load. Minimalistic designs focus on essential elements, avoiding clutter. Clear labels, visual hierarchy, and familiar patterns make the interface intuitive. For instance, Google Search exemplifies simplicity by centering user attention on its search bar.

Feedback mechanisms, such as visual cues, status indicators, and error messages, guide users and reduce confusion. For example, a progress bar during file uploads informs users about the process, while actionable error messages help resolve issues effectively.

1.4 Components of a GUI

A GUI typically includes elements like icons, menus, windows, and scrollbars. Icons represent actions or features, while menus provide shortcuts for efficient navigation. Windows organize content, and scrollbars allow users to access hidden content. These components work together to create a cohesive user experience.

Accessibility and User Testing

Accessibility ensures inclusivity for users with disabilities. Key considerations include proper color contrast, readable typography, and keyboard navigation. For example, using a contrast ratio that meets WCAG standards improves readability for visually impaired users.

User testing is essential to identify usability issues and improve the design. Methods like A/B testing and task analysis provide insights into user behavior, enabling designers to refine the interface for better satisfaction and engagement.

1.5 Designing GUI for integral solver, differential solver and numerical methods

MATLAB is a powerful tool for solving numerical problems. Below are some common numerical methods and how they can be implemented in MATLAB:

- Integral solver
- Numerical method
- Differential solver

These concepts are essential techniques used to solve mathematical problems that are difficult or impossible to solve analytically. MATLAB is a powerful tool that uses numerical analysis with its graphical and computational capabilities, making it an ideal platform for implementing numerical methods.

Some of the real-life applications of numerical methods

- Estimating ocean currents
- Modelling combustion flow in coal power plants
- Simulating airflow over airplane bodies.

CHAPTER 2: STUDY METHODOLOGY

2:1 Introduction

At the start time of the research, we divided ourselves accordingly depending on the numerical methods including the callback functions, classes and the GUI of every numerical method talked about. At least each member had to look for information about a given method and how its GUI interface works.

2.2 Discussion chapter

We were able to run the MATLAB code and use the GUI to find roots using the different numerical methods.

2.3 Steps involved.

Step 1: **Open MATLAB**

Launch MATLAB on your computer.

Step 2: **Create a New Script**

In the MATLAB environment, go to the Home tab.

Click on New Script to open the editor.

Step 3: **Create the Class Code**

Write the numerical method class code.

Save the script with the name numerical method.m. Make sure to save it in a directory that is in your MATLAB path or the current working directory.

Step 4: **Create the GUI Code**

Open another new script in the MATLAB editor

Save this script with the name numerical method name

Step 5: **Run the GUI**

In the MATLAB Command Window, type the following command to run the GUI

We discussed the various numerical methods involving MATLAB code and use of the GUI including;

- Trapezoidal method
- Euler's method
- Simpson method
- Runge Kutta 4

CHAPTER 3: THE WORK FLOW

From the previous assignment “Whilst implementing the concept of classes, encapsulation, inheritance, polymorphism and abstraction, develop and test a high end or high level back end implementation of a numerical methods application for solving computational problems.

For Simplicity apply the codes developed in the previous assignments and ensure that the parent class holds all abstract methods with two subclasses one for differential problems and the other for integral problems.”

Different codes combining the concept of classes, encapsulation, inheritance, polymorphism and abstraction were developed and used in the designing of the numerical methods GUI as shown below.

3.1 PARENT CODE (NUMERICAL METHODS)

```
classdef (Abstract) NumericalMethod
    % Abstract parent class for all numerical methods

    properties (Access = protected)
        functionHandle  % Function handle for the problem
        lowerLimit      % Start point (t0 or a)
        upperLimit      % End point (tf or b)
        stepSize        % Step size (h)
    end

    methods
        function obj = NumericalMethod(func, a, b, h)
            obj.functionHandle = func;
            obj.lowerLimit = a;
            obj.upperLimit = b;
            obj.stepSize = h;
        end
    end

    % Abstract methods (implemented in subclasses)
    methods (Abstract)
        result = compute(obj)
        plotResult(obj)
    end
end
```

3.2 DIFFERENTIAL SOLVER

```
classdef DifferentialSolver < NumericalMethod
    % Solves differential equations using Euler and RK4 methods

    properties
        initialCondition
    end

    methods
        function obj = DifferentialSolver(func, a, b, h, y0)
            obj@NumericalMethod(func, a, b, h);
            obj.initialCondition = y0;
        end

        function result = compute(obj)
            t = obj.lowerLimit:obj.stepSize:obj.upperLimit;
            yEuler = zeros(size(t));
            yRK4 = zeros(size(t));
            yEuler(1) = obj.initialCondition;
            yRK4(1) = obj.initialCondition;

            % Euler Method
            tic;
            for i = 1:length(t)-1
                yEuler(i+1) = yEuler(i) + obj.stepSize *
obj.functionHandle(t(i), yEuler(i));
            end
            eulerTime = toc;

            % Runge-Kutta 4 Method
            tic;
            for i = 1:length(t)-1
                k1 = obj.functionHandle(t(i), yRK4(i));
                k2 = obj.functionHandle(t(i)+obj.stepSize/2,
yRK4(i)+obj.stepSize*k1/2);
                k3 = obj.functionHandle(t(i)+obj.stepSize/2,
yRK4(i)+obj.stepSize*k2/2);
                k4 = obj.functionHandle(t(i)+obj.stepSize,
yRK4(i)+obj.stepSize*k3);
                yRK4(i+1) = yRK4(i) + obj.stepSize*(k1+2*k2+2*k3+k4)/6;
            end
            rk4Time = toc;

            % Results
            result.t = t;
            result.yEuler = yEuler;
            result.yRK4 = yRK4;
            result.eulerTime = eulerTime;
            result.rk4Time = rk4Time;
        end

        function plotResult(obj)
            result = obj.compute();
            figure;
            plot(result.t, result.yEuler, 'r--', 'LineWidth', 1.5); hold on;
            plot(result.t, result.yRK4, 'b', 'LineWidth', 1.5);
            legend('Euler', 'Runge-Kutta 4');
            xlabel('t'); ylabel('y');
            title('Differential Equation Solver');
            grid on;

            fprintf('\n--- Differential Solver ---\n');
            fprintf('Euler Method Time: %.6f s\n', result.eulerTime);
            fprintf('RK4 Method Time: %.6f s\n', result.rk4Time);
        end
    end
end
```

3.3 INTEGRAL SOLVER

```
classdef IntegralSolver < NumericalMethod
    % Solves integrals using Trapezoidal and Simpson's Rule

    methods
        function obj = IntegralSolver(func, a, b, h)
            obj@NumericalMethod(func, a, b, h);
        end

        function result = compute(obj)
            x = obj.lowerLimit:obj.stepSize:obj.upperLimit;
            y = obj.functionHandle(x);

            % Trapezoidal
            tic;
            trapzVal = obj.stepSize * (sum(y) - (y(1)+y(end))/2);
            trapTime = toc;

            % SimpsonÃ¢â‚¬â„¢s Rule
            tic;
            if mod(length(x)-1, 2) ~= 0
                x = x(1:end-1); y = y(1:end-1);
            end
            n = length(x)-1;
            h = (obj.upperLimit - obj.lowerLimit)/n;
            simpsonVal = (h/3)*(y(1) + 4*sum(y(2:2:end-1)) + 2*sum(y(3:2:end-
2)) + y(end));
            simpTime = toc;

            % Results
            result.x = x;
            result.trapzVal = trapzVal;
            result.simpsonVal = simpsonVal;
            result.trapTime = trapTime;
            result.simpTime = simpTime;
        end

        function plotResult(obj)
            result = obj.compute();
            figure;
            fplot(obj.functionHandle, [obj.lowerLimit obj.upperLimit],
'LineWidth', 1.5);
            title('Numerical Integration');
            grid on;

            fprintf('\n--- Integral Solver ---\n');
            fprintf('Trapezoidal: %.6f | Time: %.6f s\n', result.trapzVal,
result.trapTime);
            fprintf('Simpson: %.6f | Time: %.6f s\n', result.simpsonVal,
result.simpTime);
        end
    end
end
```

DEVLOPING THE NUMERICAL METHODS GUI(NumericalMethodsApp)

```
 classdef NumericalMethodsApp < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure                  matlab.ui.Figure
        UIAxes                    matlab.ui.control.UIAxes
        ProblemTypeDropDownLabel  matlab.ui.control.Label
        ProblemTypeDropDown       matlab.ui.control.DropDown
        MethodDropDownLabel       matlab.ui.control.Label
        MethodDropDown            matlab.ui.control.DropDown
        LowerLimitEditFieldLabel  matlab.ui.control.Label
        LowerLimitEditField       matlab.ui.control.NumericEditField
        UpperLimitEditFieldLabel  matlab.ui.control.Label
        UpperLimitEditField       matlab.ui.control.NumericEditField
        StepSizeEditFieldLabel    matlab.ui.control.Label
        StepSizeEditField         matlab.ui.control.NumericEditField
        InitialConditionEditFieldLabel  matlab.ui.control.Label
        InitialConditionEditField matlab.ui.control.NumericEditField
        ComputeButton             matlab.ui.control.Button
        ResultsTextAreaLabel      matlab.ui.control.Label
        ResultsTextArea           matlab.ui.control.TextArea
    end


    % App initialization and construction
    methods (Access = private)

        % NRM Implementation Helper Function
        function [root, time] = newtonRaphson(app, x0, tol, maxIter)
            tic; % Start timer
            f = @(x) x^3 - 2*x - 5; % Example function: xÃ‚³ - 2x - 5 = 0
            df = @(x) 3*x^2 - 2;   % Derivative of the function

            x = x0;
            iter = 0;
            history = x0;

            while abs(f(x)) > tol && iter < maxIter
                if abs(df(x)) < 1e-9 % Check for near-zero derivative
```

```
                    root = NaN;
                    time = toc;
                    app.ResultsTextArea.Value = {'Error: Near-zero derivative 
encountered.'};
                    return;
                end
                x = x - f(x) / df(x);
                history = [history, x];
                iter = iter + 1;
            end

            root = x;
            time = toc;

            % Plot the convergence history (optional but helpful)
            plot(app.UIAxes, 0:iter, history, 'm-o', 'LineWidth', 1.5, 
'MarkerSize', 4);
            title(app.UIAxes, 'Newton-Raphson Convergence');
            xlabel(app.UIAxes, 'Iteration'); ylabel(app.UIAxes, 'x_n');
            grid(app.UIAxes, 'on');
            hold(app.UIAxes, 'on');
            plot(app.UIAxes, [0, iter], [root, root], 'k--', 'DisplayName', 
'Final Root');
            legend(app.UIAxes, 'Convergence', 'Root', 'Location', 'best');
            hold(app.UIAxes, 'off');
        end


        % Button pushed function: ComputeButton
        function ComputeButtonPushed(app, event)
            try
                problemType = app.ProblemTypeDropDown.Value;
                methodType = app.MethodDropDown.Value;

                % These fields are only used if not in Root Finding mode
                if ~strcmp(problemType, 'Root Finding')
                    a = app.LowerLimitEditField.Value;
                    b = app.UpperLimitEditField.Value;
                    h = app.StepSizeEditField.Value;
                end

                cla(app.UIAxes);

                if strcmp(problemType, 'Differential')
                    y0 = app.InitialConditionEditField.Value;
                    f = @(t,y) -2*y; % Example differential equation

                    % NOTE: DifferentialSolver Class is assumed to exist and be 
functional
                    solver = DifferentialSolver(f, a, b, h, y0);
                    result = solver.compute();

                    t = result.t;
                    switch methodType
```

```
                        case 'Euler'
                            plot(app.UIAxes, t, result.yEuler, 'r--',
'LineWidth', 1.5);
                            val = result.yEuler(end);
                            time = result.eulerTime;
                            title(app.UIAxes, 'Euler Method')

                        case 'Runge-Kutta 4'
                            plot(app.UIAxes, t, result.yRK4, 'b-', 'LineWidth',
1.5);
                            val = result.yRK4(end);
                            time = result.rk4Time;
                            title(app.UIAxes, 'Runge-Kutta 4');

                        otherwise
                            app.ResultsTextArea.Value = {'Invalid method for
differential problem.'};
                            return;
                    end
                    xlabel(app.UIAxes, 't'); ylabel(app.UIAxes, 'y');
                    grid(app.UIAxes, 'on');

                    app.ResultsTextArea.Value = {
                        sprintf('Final Value: %.6f', val)
                        sprintf('Computation Time: %.6f s', time)
                    };

                elseif strcmp(problemType, 'Integral')
                    f = @(x) sin(x); % Example function

                    % NOTE: IntegralSolver Class is assumed to exist and be
functional
                    solver = IntegralSolver(f, a, b, h);
                    result = solver.compute();

                    x = result.x; y = f(x);
                    plot(app.UIAxes, x, y, 'LineWidth', 1.5);
                    xlabel(app.UIAxes, 'x'); ylabel(app.UIAxes, 'f(x)');
                    grid(app.UIAxes, 'on');

                    switch methodType
                        case 'Trapezoidal'
                            val = result.trapzVal;
                            time = result.trapTime;
                            title(app.UIAxes, 'Trapezoidal Rule');
                        case 'Simpson'
                            val = result.simpsonVal;
                            time = result.simpTime;
                            title(app.UIAxes, 'Simpsons Rule');
                        otherwise
                            app.ResultsTextArea.Value = {'Invalid method for
integral problem.'};
                            return;
                    end

                    app.ResultsTextArea.Value = {
                        sprintf('Integral Value: %.6f', val)
                        sprintf('Computation Time: %.6f s', time)
                    };

                % NEW LOGIC FOR ROOT FINDING
                elseif strcmp(problemType, 'Root Finding')
                    if strcmp(methodType, 'Newton Raphson')
                        % Use the Initial Condition field as the initial guess
(x0)
                        x0 = app.InitialConditionEditField.Value;
                        tol = 1e-6; % Fixed tolerance
                        maxIter = 50; % Fixed max iterations

                        [root, time] = app.newtonRaphson(x0, tol, maxIter);

                        if isnan(root)
                            % Error handled in the NRM function
                        else
                            app.ResultsTextArea.Value = {
                                sprintf('Root Found: %.6f', root)
                                sprintf('Computation Time: %.6f s', time)
                            };
                        end
                    else
                        app.ResultsTextArea.Value = {'Invalid method for root
finding problem.'};
                    end

                else
                    app.ResultsTextArea.Value = {'Unknown problem type.'};
                end

            catch ME
                app.ResultsTextArea.Value = {['Error: ' ME.message]};
            end
        end
    end


    methods (Access = private)

        % Create components and set layout
        function createComponents(app)
            % Create UIFigure and components
            app.UIFigure = uifigure('Visible', 'off');
            app.UIFigure.Position = [100 100 700 500];
            app.UIFigure.Name = 'Numerical Methods Solver';

            % Create UIAxes
            app.UIAxes = uiaxes(app.UIFigure);
            title(app.UIAxes, 'Output Plot')
            xlabel(app.UIAxes, 'x/t')
            ylabel(app.UIAxes, 'f(x)/y(t)')
            app.UIAxes.Position = [280 100 400 360];

            % Create ProblemTypeDropDownLabel
            app.ProblemTypeDropDownLabel = uilabel(app.UIFigure);
            app.ProblemTypeDropDownLabel.HorizontalAlignment = 'right';
            app.ProblemTypeDropDownLabel.Position = [20 430 90 22];
            app.ProblemTypeDropDownLabel.Text = 'Problem Type';

            % Create ProblemTypeDropDown
            app.ProblemTypeDropDown = uidropdown(app.UIFigure);
            app.ProblemTypeDropDown.Items = {'Differential', 'Integral', 'Root
Finding'}; % Added 'Root Finding'
            app.ProblemTypeDropDown.Position = [120 430 120 22];
            app.ProblemTypeDropDown.Value = 'Differential';

            % Create MethodDropDownLabel
            app.MethodDropDownLabel = uilabel(app.UIFigure);
            app.MethodDropDownLabel.HorizontalAlignment = 'right';
            app.MethodDropDownLabel.Position = [20 390 90 22];
            app.MethodDropDownLabel.Text = 'Method';

            % Create MethodDropDown
            app.MethodDropDown = uidropdown(app.UIFigure);
            app.MethodDropDown.Items = {'Newton Raphson','Euler', 'Runge-Kutta
4', 'Trapezoidal', 'Simpson'}; % Restored 'Newton Raphson'
            app.MethodDropDown.Position = [120 390 120 22];
            app.MethodDropDown.Value = 'Newton Raphson'; % Restored default

            % Create LowerLimitEditFieldLabel
            app.LowerLimitEditFieldLabel = uilabel(app.UIFigure);
            app.LowerLimitEditFieldLabel.HorizontalAlignment = 'right';
            app.LowerLimitEditFieldLabel.Position = [20 340 90 22];
            app.LowerLimitEditFieldLabel.Text = 'Lower Limit (a)';

            % Create LowerLimitEditField
            app.LowerLimitEditField = uieditfield(app.UIFigure, 'numeric');
            app.LowerLimitEditField.Position = [120 340 120 22];
            app.LowerLimitEditField.Value = 0;

            % Create UpperLimitEditFieldLabel
            app.UpperLimitEditFieldLabel = uilabel(app.UIFigure);
            app.UpperLimitEditFieldLabel.HorizontalAlignment = 'right';
            app.UpperLimitEditFieldLabel.Position = [20 300 90 22];
            app.UpperLimitEditFieldLabel.Text = 'Upper Limit (b)';

            % Create UpperLimitEditField
            app.UpperLimitEditField = uieditfield(app.UIFigure, 'numeric');
            app.UpperLimitEditField.Position = [120 300 120 22];
            app.UpperLimitEditField.Value = 2;

            % Create StepSizeEditFieldLabel
            app.StepSizeEditFieldLabel = uilabel(app.UIFigure);
            app.StepSizeEditFieldLabel.HorizontalAlignment = 'right';
            app.StepSizeEditFieldLabel.Position = [20 260 90 22];
            app.StepSizeEditFieldLabel.Text = 'Step Size (h)';

            % Create StepSizeEditField
            app.StepSizeEditField = uieditfield(app.UIFigure, 'numeric');
            app.StepSizeEditField.Position = [120 260 120 22];
            app.StepSizeEditField.Value = 0.1;

            % Create InitialConditionEditFieldLabel
            app.InitialConditionEditFieldLabel = uilabel(app.UIFigure);
            app.InitialConditionEditFieldLabel.HorizontalAlignment = 'right';
            app.InitialConditionEditFieldLabel.Position = [20 220 90 22];
            app.InitialConditionEditFieldLabel.Text = 'Initial Condition
(yÃ¢â€šâ,¬/xÃ¢â€šâ,¬)'; % Updated label

            % Create InitialConditionEditField
            app.InitialConditionEditField = uieditfield(app.UIFigure,
'numeric');
            app.InitialConditionEditField.Position = [120 220 120 22];
            app.InitialConditionEditField.Value = 1;

            % Create ComputeButton
            app.ComputeButton = uibutton(app.UIFigure, 'push');
            app.ComputeButton.Position = [70 170 120 30];
            app.ComputeButton.Text = 'Compute';
            app.ComputeButton.ButtonPushedFcn = createCallbackFcn(app,
@ComputeButtonPushed, true);

            % Create ResultsTextAreaLabel
            app.ResultsTextAreaLabel = uilabel(app.UIFigure);
            app.ResultsTextAreaLabel.HorizontalAlignment = 'right';
            app.ResultsTextAreaLabel.Position = [20 110 90 22];
            app.ResultsTextAreaLabel.Text = 'Results';

            % Create ResultsTextArea
            app.ResultsTextArea = uitextarea(app.UIFigure);
            app.ResultsTextArea.Position = [120 60 120 60];
            app.ResultsTextArea.Value = {'Output will appear here'};

            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end


    methods (Access = public)

        % Construct app
        function app = NumericalMethodsApp
            createComponents(app)
        end

        % Code that executes before app deletion
        function delete(app)
            delete(app.UIFigure)
        end
    end
end
```

CHAPTER 4: GUI CODE OUTPUT

4.1 EXPLANATION OF OUR GUI COMPONENTS

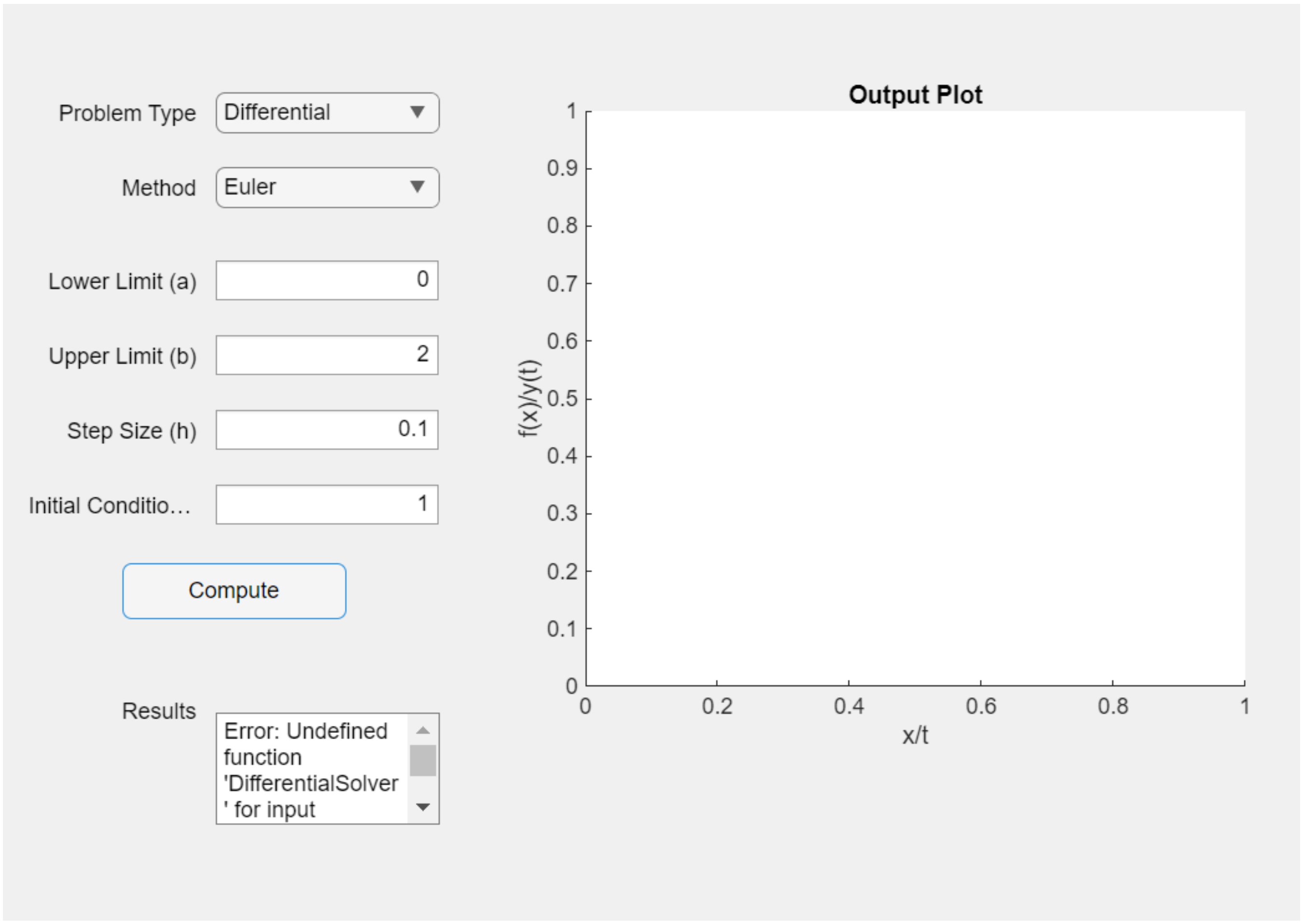
In the course of our code build up, we used the following GUI features to see that on whatever input, regardless of the equation, the numerical problem is solved.

- Drop down:

This is used to select an option from many on a single tap.we used two drop downs for the design.

- Edit field (numerical): it is used to alphabetically describe what is numerically input
- Edit field (text): it is used to alphabetically describe the output
- GUI Axes: it is used to plot the results.

The above is well represented in the GUI figure below.



Problem Type: Differential

Method: Euler

Lower Limit (a): 0

Upper Limit (b): 2

Step Size (h): 0.1

Initial Conditio…: 1

Results: Error: Undefined function 'DifferentialSolver' for input

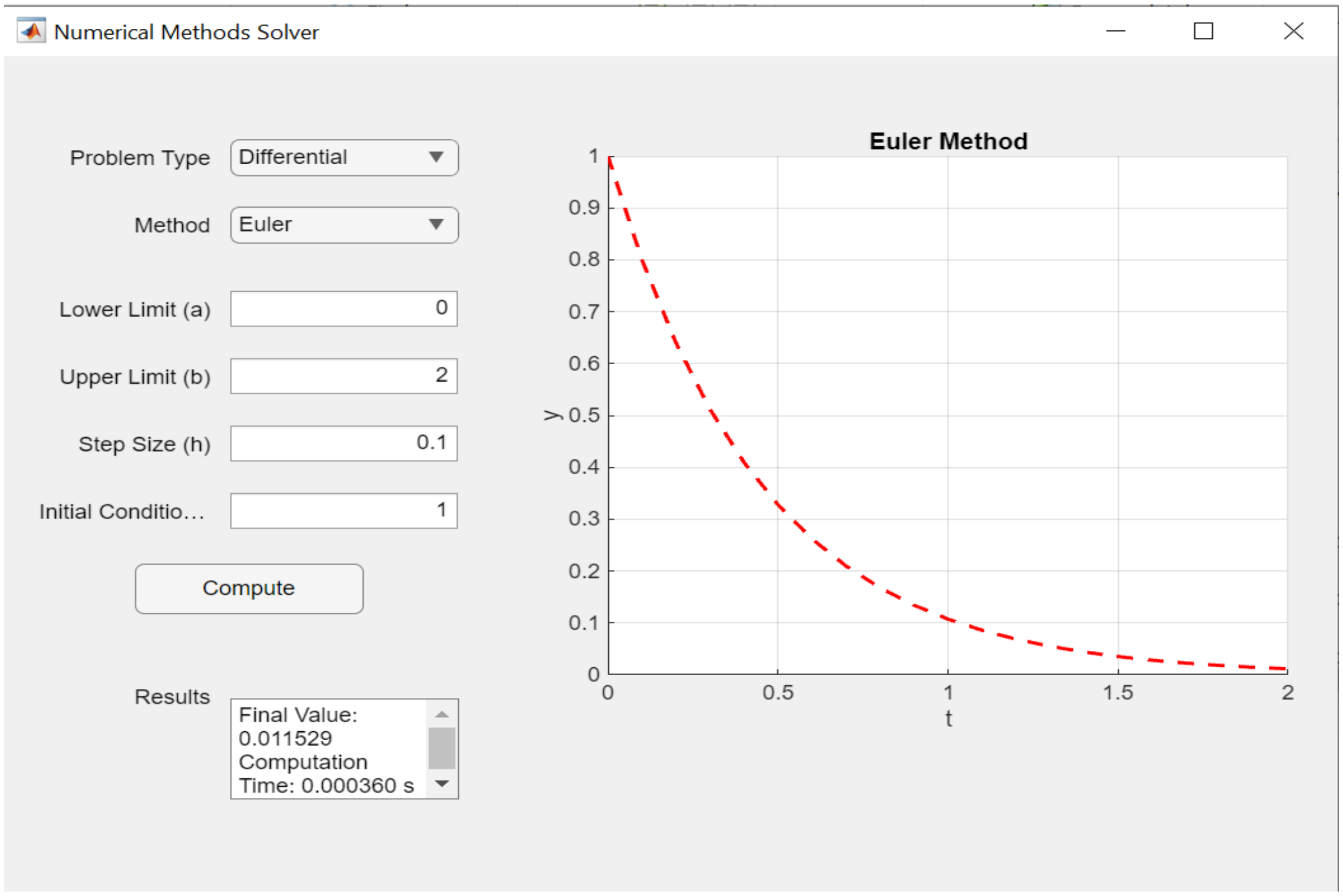
4.2 DIFFERENTIAL SOLVER

We focused on two methods

- Euler’s method

Here on input of the lower, upper limits, step size and initial conditions with the compute button pressed, the final answer or root will be printed together with the computation time. The action is completed by a plot of y (function) values against t (computation time) giving us a figure as shown below.

A figure showing the numerical computation using Euler’s method


Numerical Methods Solver

Problem Type: Differential

Method: Euler

Lower Limit (a): 0

Upper Limit (b): 2

Step Size (h): 0.1

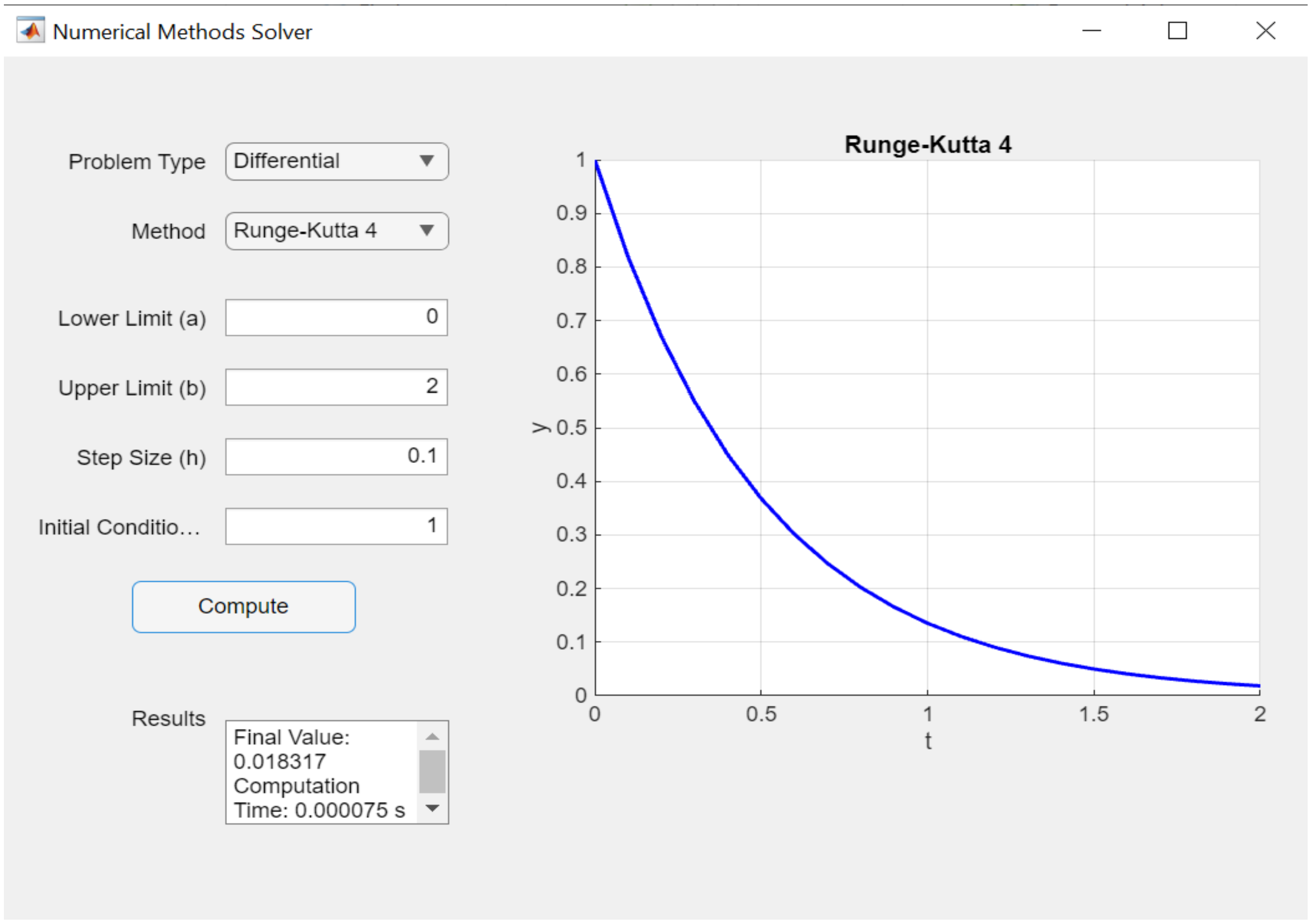
Initial Conditio…: 1

Results: Final Value: 0.011529 Computation Time: 0.000360 s

- Runge Kutta 4

Here on input of the lower, upper limits, step size and initial conditions with the compute button pressed, the final answer or root will be printed together with the computation time. The action is completed by a plot of y (function) values against t (computation time) giving us a figure as shown below.

A figure showing the numerical computation using Runge Kutta 4 method



Final Value: 0.018317
Computation Time: 0.000075 s

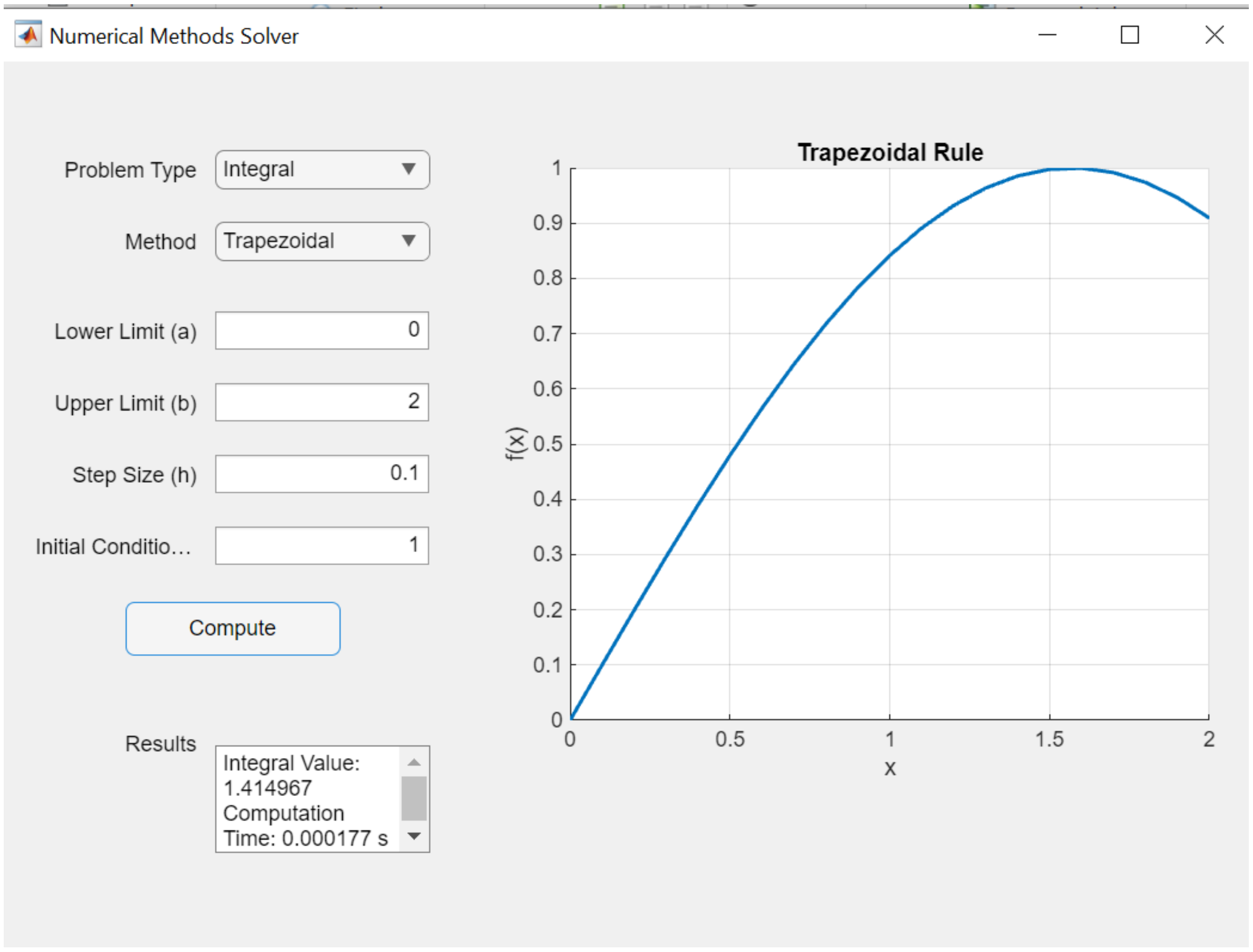
4.3 INTEGRAL SOLVER

We focused on the two first methods

- Trapezoidal method

Just like in the differential solver, an input of the lower, upper limits, step size and initial conditions with the compute button pressed, the final answer or root will be printed together with the computation time. The action is completed by a plot of y (function) values against t (computation time) giving us a figure as shown below.

A figure showing the numerical computation using Trapezoidal method


Numerical Methods Solver

Problem Type: Integral

Method: Trapezoidal

Lower Limit (a): 0

Upper Limit (b): 2

Step Size (h): 0.1

Initial Conditio…: 1

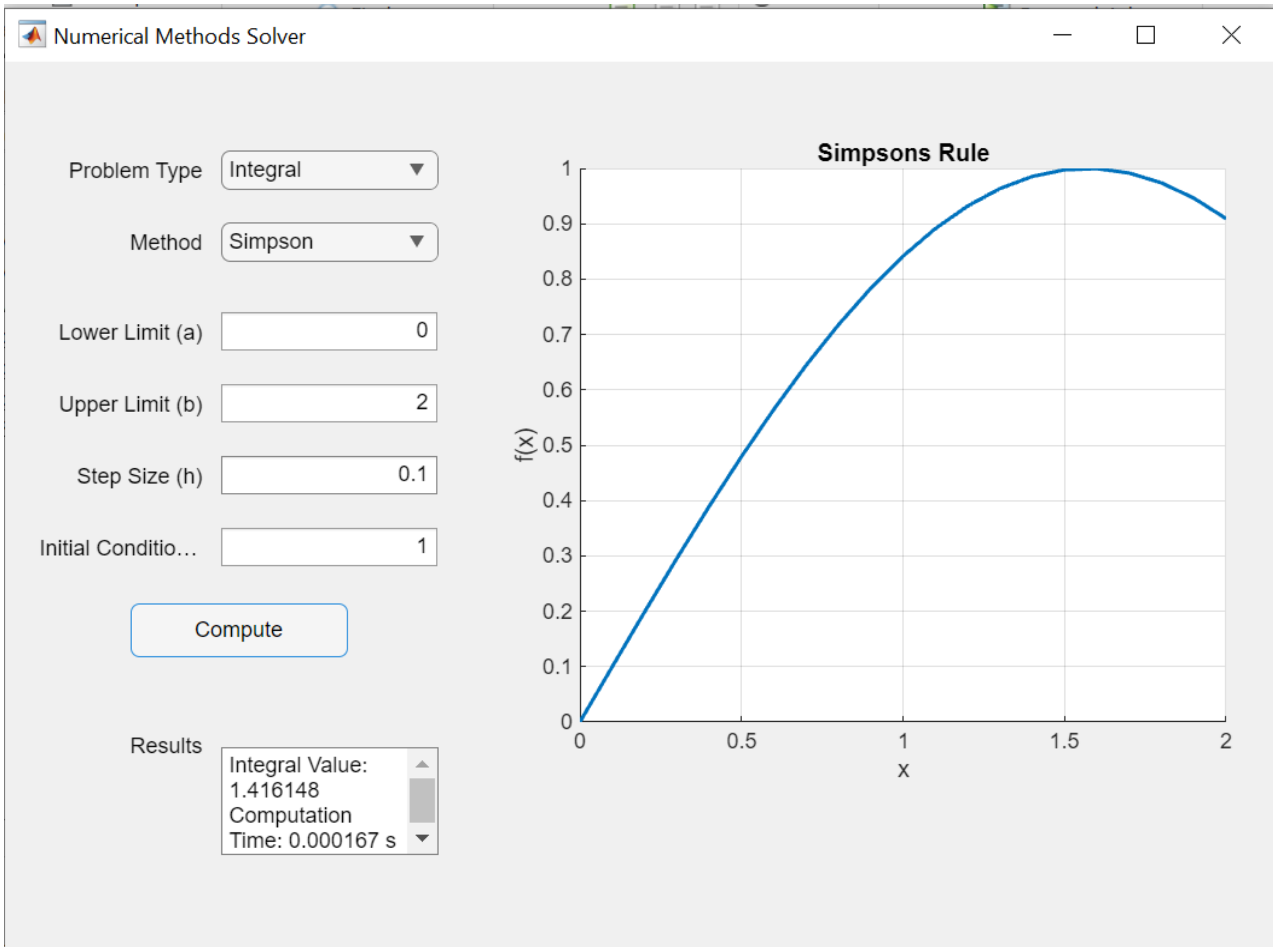
Compute

Results: Integral Value: 1.414967 Computation Time: 0.000177 s

- simpson method

Just like in the differential solver, an input of the lower, upper limits, step size and initial conditions with the compute button pressed, the final answer or root will be printed together with the computation time. The action is completed by a plot of y (function) values against t (computation time) giving us a figure as shown below.

A figure showing the numerical computation using simpson method


Numerical Methods Solver

Problem Type: Integral

Method: Simpson

Lower Limit (a): 0

Upper Limit (b): 2

Step Size (h): 0.1

Initial Conditio…: 1

Results:

```
Integral Value:
1.416148
Computation
Time: 0.000167 s
```

CHAPTER 5: CONCLUSION AND RECOMMENDATION

5.1 Recommendations for GUI Design

Prioritize Usability: Ensure that the interface is intuitive and easy to navigate. Conduct user testing to identify pain points and areas for improvement.

Consistency is Key: Maintain a consistent layout, color scheme, and typography throughout the application. This helps users build familiarity and reduces cognitive load.

Responsive Design: Design GUIs that adapt to different screen sizes and resolutions. This is crucial for applications accessed on various devices, including desktops, tablets, and smartphones.

Feedback Mechanisms: Provide clear feedback for user actions (e.g., button clicks, form submissions). This can include visual cues, sounds, or notifications to confirm actions.

Accessibility: Ensure that the GUI is accessible to all users, including those with disabilities. Follow accessibility guidelines (like WCAG) to make your application usable for everyone.

5.2 Conclusions on GUI Design

User-Centric Approach: The success of a GUI hinges on understanding user needs and behaviors. A user-centric design process leads to more effective and satisfying interfaces.

Importance of Testing: Regular usability testing is essential to identify issues and validate design choices. Engaging real users in the testing process provides invaluable insights.

Evolving Standards: GUI design is not static; it evolves with technology and user expectations. Staying updated with the latest trends and best practices is crucial for creating relevant and effective interfaces.

Integration of Technology: As technology advances (e.g., AI, voice interfaces), GUIs must adapt to incorporate new interaction methods, enhancing user experience and accessibility.

In summary effective GUI design is a blend of usability, aesthetics, and continuous improvement, all centered around the user experience. By following best practices and remaining adaptable to change, designers can create interfaces that are not only functional but also enjoyable to use.

References

Books

1. "The Design of Everyday Things" by Don Norman

A classic book that discusses design principles applicable to GUIs and everyday objects.

2. "About Face: The Essentials of Interaction Design" by Alan Cooper, Robert Reimann, and David Cronin

This book provides insights into interaction design and user experience, focusing on GUI design.

3. "Don't Make Me Think" by Steve Krug

A practical guide to web usability that emphasizes intuitive design in GUIs.

4. "Designing Interfaces: Patterns for Effective Interaction Design" by Jenifer

Tidwell A comprehensive resource on interface design patterns and best practices

for GUIs.