Southern Luzon State University

College of Engineering

Computer Engineering Department

TIME - VARIANCE TEST

Time Variant and Time Invariant of Continuous and Discrete Signals

CPE22L – Digital Signal Processing Laboratory
Application of Digital Signal Processing using MATLAB
Finals Project

BY:

Amat, Juliana Louise A.
Bagacina, Alexis O.
Bolo, Lyza April P.
Carreos, Clarice Mae G.
Corpuz, Reimarc G.
Lariba, Aron P.
Mostoles, Ivan M.
Oba, Romina Giane Z.
Reyes, Realyn L.

BSCpE IV – GE

Engr. Rene M. Inson

Course Instructor

A.Y. 2023 - 2024

INTRODUCTION

MATLAB, which stands for Matrix Laboratory, is a high-level language and interactive environment. It is designed primarily for numerical computing, data analysis, signal processing, and visualization, and is used in various fields such as engineering. In digital signal processing, MATLAB is a versatile tool for signal analysis, filter design, and such. It offers a set of functions that make it easier for researchers and engineers to develop, analyze, and implement algorithms for a wide range of signal-processing applications.

Continuous signals and discrete signals are two fundamental types of signals used in the field of signal processing, and they differ primarily in terms of their nature, representation, and mathematical characteristics.

Continuous signals are defined for every point in time within a specified interval and can take any value within a given range at any instant, as they are continuous both in amplitude and time. Continuous signals are crucial in analog systems where signals vary continuously over time, finding applications in analog communication, control systems, and analog signal processing.

On the other hand, discrete signals are defined only at specific, isolated points in time and have discrete values at distinct time instances.Discrete signals are crucial in digital systems where information is processed in a step-by-step manner, playing a vital role in digital communication, digital signal processing, and any application involving digital data.

Discrete-time systems are mathematical models used to describe the relationship between an input signal and an output signal in a discrete-time domain. The importance of discrete-time system properties lies in their role in shaping the behavior, characteristics, and performance of systems in various digital signal processing (DSP) applications.

One of the properties of both discrete and continuous time systems is the time variance. Time variance is a property of systems that describes how their behavior changes with respect to time. Time-invariant systems exhibit consistent behavior over time, simplifying the analysis and allowing for the application of results obtained at one time to different time instances. This property is crucial for developing stable and reliable systems.

Understanding the concepts of the discrete and continuous signals and the property time variance, will provide a strong foundation for applying them in the creation of GUIs.

DESCRIPTION AND FUNCTION OF THE PROGRAM

MATLAB Code for Time Variance Test of Signals

```
function FINALPROJECT
      % Create the main figure
    mainFig = figure('Name', 'Time
Variance Test GUI', 'Position', [400,
100, 1200, 900], 'Color', "#E4E1D8",
'MenuBar', 'none');
    % ---------------------------------
-----------------------------------------
---------------------------- menu bar

    % Create an edit box for user input
expression
    expressionEdit = uicontrol('Style',
'edit', 'Units', 'normalized',
'String', 'Enter Expression (e.g
sin(x))');

    % Create a button group for signal
type selection
    signalTypeGroup =
uibuttongroup('Units', 'normalized');

    % Create radio buttons for signal
type
    continuousButton =
uicontrol(signalTypeGroup, 'Style',
'radiobutton', 'String', 'C O N T I N U
O S', ...
        'Units', 'normalized',
'Position', [0.1, 0.48, 0.8, 0.6],
'Callback', @updateSliderVisibility);

    discreteButton =
uicontrol(signalTypeGroup, 'Style',
'radiobutton', 'String', 'D I S C R E T
E', ...
        'Units', 'normalized',
'Callback', @updateSliderVisibility);

    % Create a slider for input signal
(amplitude)
inputSlider = uicontrol('Style',
'slider', 'Min', 0, 'Max', 1, 'Value',
1, ...
    'Units', 'normalized', 'Position',
[0.1, 0.65, 0.4, 0.05], 'Callback',
@updatePlot);


    % Create a slider for time shift
    timeShiftSlider =
uicontrol('Style', 'slider', 'Min', -
10, 'Max', 10, 'Value', 0, ...
        'Units', 'normalized',
'Position', [0.55, 0.65, 0.4, 0.05],
'Callback', @updatePlot, 'Visible',
'on');

    % Create an edit box for entering
discrete sequences
    sequenceEdit = uicontrol('Style',
'edit', 'Units', 'normalized',
'String', 'Enter Sequence (Space
Separated)', 'Visible', 'off');

    % Create labels for sliders
    amplitudeLabel = uicontrol('Style',
'text', 'String', 'A M P L I T U D E',
'Units', 'normalized');
    timeShiftLabel = uicontrol('Style',
'text', 'String', 'T I M E  S H I F T',
'Units', 'normalized');


    % Create labels for user input
information
    %dispLabel = uicontrol('Style',
'text', 'String', '', ...
    %    'Units', 'normalized',
'Position', [0.75, 0.35, 0.25, 0.15]);


    % Create labels for user input
information
    PrgmTitle = uicontrol('Style',
'text', 'String', 'Time - Variance
Test', ...
        'Units', 'normalized',
'Position', [0.25, 0.9, 0.5, 0.05]);

    % Create axes for input signal
    inputAxes = axes('Parent', mainFig,
'Position', [0.1, 0.33, 0.6, 0.2]);
    title(inputAxes, 'Input Signal');

    outputAxes2 = axes('Parent',
mainFig, 'Position', [0.1, 0.06, 0.6,
0.2], 'Visible', 'off');
    title(outputAxes2, 'Signal after
Shift');

    outputAxes1 = axes('Parent',
mainFig, 'Position', [0.1, 0.06, 0.2,
0.2], 'Visible', 'off');
    title(outputAxes1, 'Signal after
Shift');

    outputAxes3 = axes('Parent',
mainFig, 'Position', [0.35, 0.06, 0.2,
0.2], 'Visible', 'off');
    title(outputAxes3, 'LHS');

    outputAxes4 = axes('Parent',
mainFig, 'Position', [0.60, 0.06, 0.2,
0.2], 'Visible', 'off');
    title(outputAxes4, 'RHS');
```

```
    % Create a button to show the
result
    testButton = uicontrol('Style',
'pushbutton', 'String', 'T E S T', ...
        'Units', 'normalized',
'Callback', @performTest);



set(PrgmTitle,'ForegroundColor',
"#705D3F", 'FontSize', 32,
'BackgroundColor', "#E4E1D8",
'FontWeight', 'bold');

set(expressionEdit,'ForegroundColor',
"#705D3F",'FontSize', 12,
'BackgroundColor', "#B4B897",
'FontWeight', 'bold', 'Position', [0.1,
0.80, 0.40, 0.05]);

set(sequenceEdit,'ForegroundColor',
"#705D3F",'FontSize', 12,
'BackgroundColor', "#B4B897",
'FontWeight', 'bold', 'Position',
[0.32, 0.72, 0.4, 0.05]);


set(signalTypeGroup,'FontSize', 12,
'BackgroundColor', "#B4B897",
'Position', [0.55, 0.80, 0.4, 0.05]);

set(continuousButton,'ForegroundColor',
"#705D3F",'FontSize', 12, 'FontAngle',
'italic', 'FontWeight', 'bold',
'BackgroundColor', "#B4B897",
'Position', [0.1, 0.20, 0.8, 0.6]);

set(discreteButton,'ForegroundColor',
"#705D3F",'FontSize', 12, 'FontAngle',
'italic', 'FontWeight', 'bold',
'BackgroundColor', "#B4B897",
'Position', [0.6, 0.20, 0.8, 0.6]);


set(amplitudeLabel,'ForegroundColor',"#
705D3F", 'FontSize', 12, 'FontWeight',
'bold', 'BackgroundColor', "#E4E1D8",
'Position', [0.2, 0.58, 0.2, 0.05]);

set(timeShiftLabel,'ForegroundColor',"#
705D3F", 'FontSize', 12, 'FontWeight',
'bold', 'BackgroundColor', "#E4E1D8",
'Position', [0.65, 0.58, 0.2, 0.05]);
         set(outputAxes2,'FontSize',
12, 'Color', "#0072BD", 'FontAngle',
'italic');


set(testButton,'ForegroundColor',"#E4E1
D8", 'FontSize', 18, 'BackgroundColor',
"#715C41", 'FontWeight', 'bold',
'Position', [0.85, 0.06, 0.1, 0.07]);
```

```
        % ------------------------------
------------------------------------
--------------------------------set
    % Callback function to update
slider visibility
    % Callback function to update
slider visibility
    function updateSliderVisibility(~,
~)
        % Show/hide the time shift
slider and sequence edit box based on
the selected signal type
        if discreteButton.Value
            set(timeShiftSlider,
'Visible', 'on');
            set(sequenceEdit,
'Visible', 'on');
        else
            set(timeShiftSlider,
'Visible', 'on');
            set(sequenceEdit,
'Visible', 'off');
        end

    end

    % Function to perform the test and
update the output plot
    function performTest(~, ~)
        % Clear the axes before
updating the plots
        cla(inputAxes);
        cla(outputAxes1);

        cla(outputAxes2);
        cla(outputAxes3);
        cla(outputAxes4);

        % Get the amplitude slider
value
        amplitudeValue =
inputSlider.Value;

        % Get the time shift slider
value
        timeShiftValue =
round(timeShiftSlider.Value);

        % Get the selected signal type
        if continuousButton.Value
            signalType = 'Continuous';
        elseif discreteButton.Value
            signalType = 'Discrete';
        end

        % Get the user input expression
        userExpression =
strrep(get(expressionEdit, 'String'),
'', '.');

        % Generate a new input signal
based on the user input expression and
slider values
        if strcmp(signalType,
'Discrete')
```

```
            % For discrete signals,
read the sequence from the edit box
            sequenceString =
get(sequenceEdit, 'String');
            inputSignal =
str2num(sequenceString) *
amplitudeValue;
            x = 1:length(inputSignal);
% Use integers as time values for
discrete signals
        else
            % For continuous signals,
use the original linspace
            x = linspace(0, 1, 1000);
% Increase the number of points for
smoother plots
            inputSignal =
eval(userExpression) * amplitudeValue;
        end

        % Perform the time variance
test
        % Create figure if not already
created
mainFig = gcf;

% Delete existing annotations, if any
existingDispLabelAnnotations =
findall(mainFig, 'Type', 'textbox',
'Tag', 'dispLabelAnnotation');
delete(existingDispLabelAnnotations);

existingTimeVarianceAnnotations =
findall(mainFig, 'Type', 'textbox',
'Tag', 'timeVarianceAnnotation');
delete(existingTimeVarianceAnnotations)
;
if strcmp(signalType, 'Discrete')
    % Update the input signal plot
    stem(inputAxes, x, inputSignal);
    title(inputAxes, 'Input Signal');

    x1 = inputSignal;
    n0 = timeShiftValue;

    % Shift the signal by wrapping
around the values
    x2 = [x1(end-n0+1:end), x1(1:end-
n0)];
    y1 = x1;
    y2 = x2;
    y3 = [zeros(1, n0), y1];

    % Update the output signal plots
    stem(outputAxes1, x2, y2);
    title(outputAxes1, 'Signal after
Shift');

    stem(outputAxes3, y2);
    title(outputAxes3, 'LHS');
    stem(outputAxes4, y3);
    title(outputAxes4, 'RHS');
    set(outputAxes2, 'Visible', 'off');

    % Update user input information
text
    dispLabelString =
sprintf('%s\n%s\n%s\n%s', ...
        ['User Input Expression: '
userExpression], ...
        ['Signal Type: ' signalType],
...
        ['Amplitude: '
num2str(amplitudeValue)], ...
        ['Time Shift: '
num2str(timeShiftValue)]);

    % Create a new annotation
    annotation(mainFig, 'textbox',
[0.72, 0.32, 0.26, 0.2], 'String',
dispLabelString, 'Tag',
'dispLabelAnnotation');

    % Display whether the system is
time-variant or time-invariant
    timeVarianceText = '';
    if isequal(y2, y3)
        timeVarianceText = 'The system
is time-invariant.';
    else
        timeVarianceText = 'The system
is time-variant.';
    end
    % Display the time variance
information
    disp(timeVarianceText);
    disp(y2);

    % Create a new annotation
    annotation(mainFig, 'textbox',
[0.72, 0.32, 0.26, 0.05], 'String',
timeVarianceText, 'Tag',
'timeVarianceAnnotation');

else
    % Continuous case
    % Use the input signal directly for
continuous case

    % Update the input signal plot
    plot(inputAxes, x, inputSignal);
    title(inputAxes, 'Input Signal');
    y1 = inputSignal;
    y2 = eval(userExpression);
    y2 = interp1(x, inputSignal, x,
'linear', 0);

    % Plot the system output for
original and shifted input
    plot(outputAxes2, x +
timeShiftValue, y2);
    title(outputAxes2, 'Signal after
Shift');
    set(outputAxes3, 'Visible', 'off');
    set(outputAxes4, 'Visible', 'off');
    set(outputAxes1, 'Visible', 'off');

    % Update user input information
text
```

```
    displLabelString =
sprintf('%s\n%s\n%s\n%s', ...
        ['User Input Expression: '
userExpression], ...
        ['Signal Type: ' signalType],
...
        ['Amplitude: '
num2str(amplitudeValue)], ...
        ['Time Shift: '
num2str(timeShiftValue)]);

    % Create a new annotation
    annotation(mainFig, 'textbox',
[0.72, 0.32, 0.26, 0.2], 'String',
displLabelString, 'Tag',
'displLabelAnnotation');

    % Display whether the system is
time-variant or time-invariant
    % Display whether the system is
time-variant or time-invariant
% Display whether the system is time-
variant or time-invariant
if isequal(y2, interp1(x,
eval(userExpression), x +
timeShiftValue, 'linear', 0))
    timeVarianceText = 'The system is
time-invariant';
else
    timeVarianceText = 'The system is
time-variant';
end


    % Create a new annotation
    disp(timeVarianceText);
    annotation(mainFig, 'textbox',
[0.72, 0.32, 0.26, 0.05], 'String',
timeVarianceText, 'Tag',
'timeVarianceAnnotation');
end
    end
function updatePlot(~, ~)
    %%%
end
end
```

Flow Chart

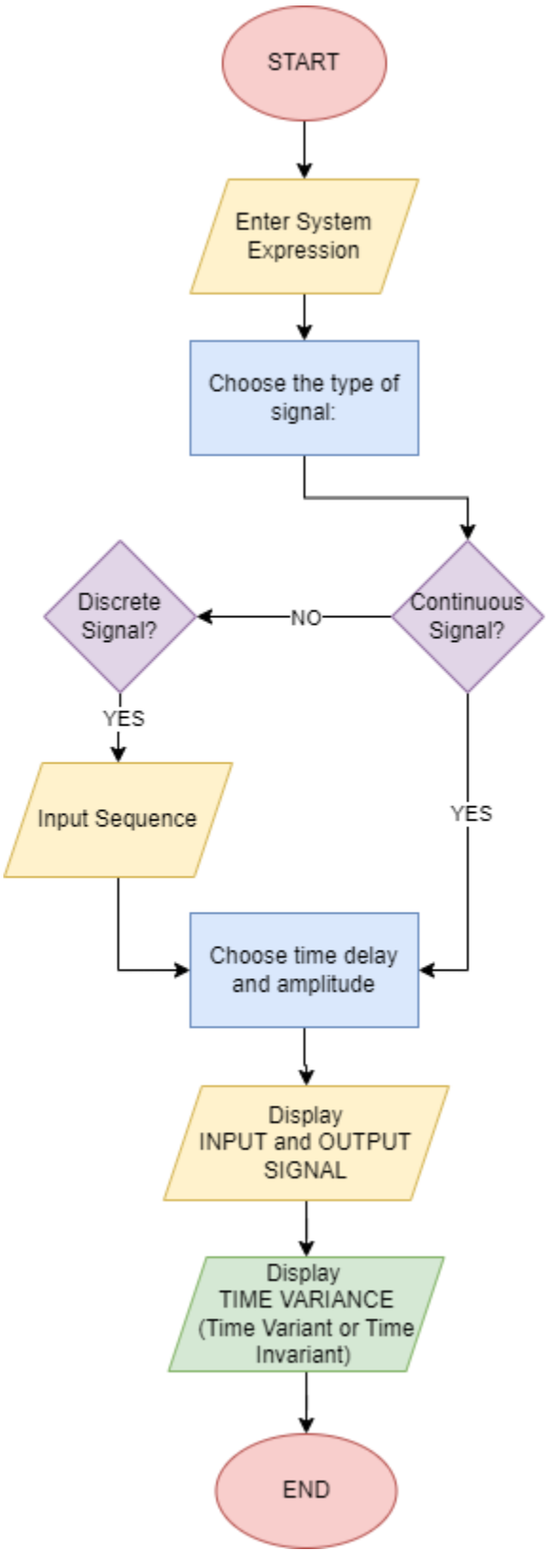


Figure 1. Flowchart of the GUI.

Declared Variables

Variable	Description
mainFig	Represents the main GUI figure.
expressionEdit	Allows the user to input mathematical expressions
signalTypeGroup	Groups radio buttons for selecting signal type (continuous or discrete).
continuousButton, discreteButton	Allows the user to choose the signal type (continuous or discrete)
inputSlider	Adjusts the amplitude of the input signal.
timeShiftSlider	Adjusts the time shift applied to the input signal.
sequenceEdit	Allows the user to input discrete sequences (visible only for discrete signals).
dispLabel	Displays user input information, such as expression, signal type, amplitude, and time shift.
PrgmTitle	Displays the program title.
inputAxes, outputAxes2, outputAxes3, outputAxes4	Represent the axes for plotting input and output signals.
testButton	Triggers the test function (performTest) when clicked.
performTest	Performs the time variance test and updates the output plots based on user input.
amplitudeValue	Stores the current value of the amplitude slider.
timeShiftValue	Stores the current value of the time shift slider.
signalType	Stores the selected signal type (continuous or discrete).
userExpression	Stores the user-input mathematical expression.
x, x1, x2, y1, y2, y3	Store time and signal values for plotting and performing time variance test.
sequenceString	Stores the user-input discrete sequence.
dispLabelString	Stores the formatted string for displaying user input information.
timeVarianceText	Stores the result of the time variance test (time-invariant or time-variant).
annotation	Displays the time variance information on the main figure.

Table 1. List of Variables declared in the program.

Code Blocks Description

```
% Create the main figure
mainFig = figure('Name', 'Time Variance Test GUI', 'Position', [100, 100, 900, 500]);
```

Code Block 1. GUI Initialization

In this code snippet, we initiate the main GUI figure (mainFig). The figure function creates a new figure window, and parameters like 'Name' set the title of the window ('Time Variance Test GUI'). The 'Position' property determines the position and size of the GUI window on the screen.

```
% Create an edit box for user input expression
   expressionEdit = uicontrol('Style', 'edit', 'Units', 'normalized',
'Position', [0.1, 0.80, 0.40, 0.05], 'String', 'Enter Expression (e.g
sin(x))');
   % Create a button group for signal type selection
   signalTypeGroup = uibuttongroup('Units', 'normalized', 'Position', [0.55,
0.80, 0.35, 0.05]);
```

Code Block 2. User Input Controls

In this segment, the emphasis is on establishing controls to facilitate user input. The expressionEdit component is designed as an editable text box, configured with the style 'edit'. This feature enables users to input mathematical expressions, and its 'Position' property determines its location and size within the graphical user interface (GUI) window. Additionally, an initial string is provided to guide users in utilizing this input box. Concurrently, the signalTypeGroup is implemented as a button group of type uibuttongroup. This group is designated to contain radio buttons that facilitate the selection of the type of signal. Positioned within the GUI window, the properties assigned to signalTypeGroup govern its appearance and behavior, contributing to an organized and user-friendly interface.

```
 % Create radio buttons for signal type
   continuousButton = uicontrol(signalTypeGroup, 'Style', 'radiobutton',
'String', 'Continuous', ...
       'Units', 'normalized', 'Position', [0.1, 0.48, 0.8, 0.6], 'Callback',
@updateSliderVisibility);
   discreteButton = uicontrol(signalTypeGroup, 'Style', 'radiobutton',
'String', 'Discrete', ...
       'Units', 'normalized', 'Position', [0.6, 0.48, 0.8, 0.6], 'Callback',
@updateSliderVisibility);
   % Create a slider for input signal (amplitude)
   inputSlider = uicontrol('Style', 'slider', 'Min', 0, 'Max', 1, 'Value',
0.5, ...
       'Units', 'normalized', 'Position', [0.08, 0.65, 0.4, 0.05],
'Callback', @updatePlot);
```

Code Block 3. Signal Type Selection Controls Setup

This code snippet is dedicated to establishing a button group within a graphical user interface (GUI) to facilitate the selection of signal types. The `signalTypeGroup` is created using the `uibuttongroup` function, serving as a container for radio buttons. Two radio buttons, namely `continuousButton` and `discreteButton`, are embedded within this

group to represent the options 'Continuous' and 'Discrete'. The 'Position' property specifies their placement and dimensions within the GUI window. Notably, the callback function `@updateSliderVisibility` is assigned to both radio buttons, indicating that upon selection, this function will be triggered to handle the associated logic—presumably involving updates to the visibility or behavior of other GUI elements based on the chosen signal type. Overall, this code plays a crucial role in creating an interactive interface for users to designate the type of signal, enhancing the overall usability of the graphical application.

```
% Create a slider for input signal (amplitude)
   inputSlider = uicontrol('Style', 'slider', 'Min', 0, 'Max', 1, 'Value',
0.5, ...
       'Units', 'normalized', 'Position', [0.08, 0.65, 0.4, 0.05],
'Callback', @updatePlot);
% Create a slider for time shift
   timeShiftSlider = uicontrol('Style', 'slider', 'Min', -10, 'Max', 10,
'Value', 0, ...
       'Units', 'normalized', 'Position', [0.55, 0.65, 0.4, 0.05],
'Callback', @updatePlot, 'Visible', 'on');
```

Code Block 4. Slider Controls

This section introduces sliders and an edit box. inputSlider represents the amplitude slider, allowing users to adjust the amplitude of the input signal. timeShiftSlider represents the time shift slider, enabling users to shift the input signal in time. Additionally, there's an edit box (sequenceEdit) initially hidden for entering discrete sequences. These controls facilitate customization of the signals under examination.

```
% Create an edit box for entering discrete sequences
sequenceEdit = uicontrol('Style', 'edit', 'Units', 'normalized', 'Position',
[0.32, 0.72, 0.4, 0.05], 'String', 'Enter Sequence (Space Separated',
'Visible', 'off');
```

Code Block 5. Slider Controls

This code segment creates an editable text box within the GUI, specifically designed for entering discrete sequences. The 'Style' parameter is set to 'edit,' enabling user input, and the 'Position' property defines its location and size. The 'String' property provides an initial prompt, guiding users to input a space-separated sequence. The 'Visible' property is initially set to 'off,' implying that the text box becomes visible based on user interactions or specific conditions, enhancing the interactive nature of the GUI for discrete sequence input.

```
% Create labels for sliders
uicontrol('Style', 'text', 'String', 'Amplitude', 'Units', 'normalized',
'Position', [0.22, 0.58, 0.1, 0.05]);
   uicontrol('Style', 'text', 'String', 'Time Shift', 'Units', 'normalized',
'Position', [0.73, 0.58, 0.1, 0.05]);
% Create labels for user input information
dispLabel = uicontrol('Style', 'text', 'String', '', ...
       'Units', 'normalized', 'Position', [0.75, 0.2, 0.25, 0.15]);
% Create labels for program title
PrgmTitle = uicontrol('Style', 'text', 'String', 'Time - Variance Test', ...
       'Units', 'normalized', 'Position', [0.40, 0.9, 0.25, 0.05]);
```

Code Block 6. Labels

This code section creates labels in the GUI for sliders ('Amplitude' and 'Time Shift'), user input information ('dispLabel'), and the program title ('Time - Variance Test'). These labels enhance the user interface by providing clear descriptions, dynamic feedback, and a program title, contributing to a well-structured and informative graphical environment.

```
% Create axes for input signal
   inputAxes = axes('Parent', mainFig, 'Position', [0.1, 0.33, 0.6, 0.2]);
   title(inputAxes, 'Input Signal');
% Create axes for output signals
   outputAxes2 = axes('Parent', mainFig, 'Position', [0.1, 0.06, 0.6, 0.2]);
   title(outputAxes2, 'Signal after Shift');
   outputAxes3 = axes('Parent', mainFig, 'Position', [0.35, 0.06, 0.2,
0.2]);
   title(outputAxes3, 'LHS');
   outputAxes4 = axes('Parent', mainFig, 'Position', [0.60, 0.06, 0.2,
0.2]);
   title(outputAxes4, 'RHS');
```

Code Block 7. Plotting Initialization

This code establishes axes within the main GUI figure to visualize input and output signals. Four axes are created, each designated for specific purposes. The `inputAxes` represent the input signal, while `outputAxes2` displays the signal after a time shift. Additionally, `outputAxes3` and `outputAxes4` are dedicated to visualizing specific components ('LHS' and 'RHS'). The positioning parameters determine the location and size of each axis within the GUI window. Titles are assigned to these axes, providing a descriptive label for better interpretation of the displayed signals, enhancing the overall clarity and user experience of the graphical interface.

```
% Create a button to show the result
   testButton = uicontrol('Style', 'pushbutton', 'String', 'Test', ...
       'Units', 'normalized', 'Position', [0.83, 0.06, 0.1, 0.07],
'Callback', @performTest);
```

Code Block 8. Test Button

This code generates a button labeled 'Test' within the GUI. The button is created using the ‘uicontrol’ function with a 'pushbutton' style. The button's appearance and behavior, such as its position and dimensions, are defined by various properties like 'Units' and 'Position'. The 'Callback' property specifies that the ‘performTest’ function should be executed when the button is pressed. This button likely serves as a trigger for initiating a test or computation, and its location is specified within the normalized coordinates of the GUI window.

```
% Function to perform the test and update the output plot
function performTest(~, ~)
```

Code Block 9. Test Execution Function

The `performTest` function in this MATLAB code executes the time variance test and dynamically updates the graphical user interface (GUI) output plots. It clears previous

plots and annotations, retrieves user input (amplitude, time shift, signal type, and mathematical expression), generates the input signal, performs the time variance test, and updates the GUI to display the results. This function enhances modularity and maintainability within the overall program.

```
% Clear the axes before updating the plots
       cla(inputAxes);
       cla(outputAxes2);
       cla(outputAxes3);
       cla(outputAxes4);
       % Clear previous annotations
       delete(findall(mainFig,'type','annotation'));
```

Code Block 10. Clearing Previous Plots and Annotations

This code snippet serves the purpose of preparing the graphical user interface (GUI) for updated information by clearing existing content. The four `cla` commands target different axes within the GUI—`inputAxes`, `outputAxes2`, `outputAxes3`, and `outputAxes4`—ensuring a clean slate for updated signal plots. Simultaneously, the `delete(findall(mainFig,'type','annotation'));` statement removes any previous annotations associated with the main figure (`mainFig`). This systematic clearing process helps maintain the visual integrity of the GUI, preventing overlap or clutter when displaying new input and output signals after each test iteration.

```
       % Get the amplitude slider value
       amplitudeValue = inputSlider.Value;
       % Get the time shift slider value
       timeShiftValue = timeShiftSlider.Value;
```

Code Block 11. Fetching Slider Values

This code segment is responsible for acquiring the current positions of two sliders, namely `inputSlider` and `timeShiftSlider`. The `amplitudeValue` variable captures the selected amplitude level from the `inputSlider`, providing a key parameter for determining the magnitude of the input signal in subsequent computations. Similarly, the `timeShiftValue` variable retrieves the current position of the `timeShiftSlider`, representing the amount of time by which the signal will be shifted. Both of these values are crucial parameters influencing the characteristics of the input signal and the applied time shift, contributing to the overall analysis of time variance in the system.

```
% Get the selected signal type
       if continuousButton.Value
           signalType = 'Continuous';
       elseif discreteButton.Value
           signalType = 'Discrete';
       end
```

Code Block 12. Determining Signal Type

This code section determines the type of signal selected by the user through the radio buttons `continuousButton` and `discreteButton`. The conditionals check the boolean values associated with each button, and based on the active state, it assigns a corresponding label to the `signalType` variable. If the `continuousButton` is selected, the

`signalType` is set to 'Continuous', and if the `discreteButton` is chosen, the `signalType` is assigned the value 'Discrete'. This categorization is vital for subsequent processes, guiding the system in handling either continuous or discrete signals accordingly during the time variance analysis.

```
% Get the user input expression
       userExpression = get(expressionEdit, 'String');
       % Generate a new input signal based on the user input expression and
slider values
       if strcmp(signalType, 'Discrete')
           % For discrete signals, read the sequence from the edit box
           sequenceString = get(sequenceEdit, 'String');
           inputSignal = str2num(sequenceString) * amplitudeValue;
           x = 1:length(inputSignal);  % Use integers as time values for
discrete signals
       else
           % For continuous signals, use the original linspace
           x = linspace(0, 1, 1000);  % Increase the number of points for
smoother plots
           inputSignal = eval(userExpression) * amplitudeValue;
       end
```

Code Block 13. User Input Processing

This section retrieves and processes the user's mathematical expression from the `expressionEdit` box, storing it as `userExpression`. It dynamically generates an input signal based on this expression and the amplitude slider (`amplitudeValue`). For discrete signals, it reads the sequence from `sequenceEdit`, converts it, and assigns it to `inputSignal` with corresponding time values `x`. For continuous signals, it evaluates the expression over a linspace and scales it accordingly. This versatile approach accommodates various signal types for subsequent analysis.

```
% Perform the time variance test
       if strcmp(signalType, 'Discrete')
           % Update the input signal plot
           stem(inputAxes, x, inputSignal);
           title(inputAxes, 'Input Signal');
           x1 = inputSignal;  % Assuming the inputSignal is a discrete
signal
           n0 = round(timeShiftValue);
           x2 = x - timeShiftValue;
           y1 = x1;
           y2 = interp1(x, [zeros(1, n0), x1], x2, 'linear', 0);
           y3 = [zeros(1, n0), y1];
```

Code Block 14. Performing Time Variance Test

This segment executes the time variance test specifically tailored for discrete signals. It begins by updating the input signal plot on `inputAxes` using the stem plot function. Assuming `inputSignal` represents a discrete signal, it then calculates the shifted signal (`y2`) by interpolating and shifting the original signal based on the time shift value (`timeShiftValue`). The results (`y2` and `y3`) are plotted on `outputAxes2`, `outputAxes3`, and `outputAxes4`. These plots depict the signal after the shift, the left-hand side (LHS) of the time variance equation, and the right-hand side (RHS) of the equation, respectively. This process forms the core of the time variance test for discrete signals in the graphical user interface (GUI).

```
% Update the output signal plots
            stem(outputAxes2, x2, y2);
            title(outputAxes2, 'Signal after Shift');
            stem(outputAxes3, x, y2);
            title(outputAxes3, 'LHS');
            stem(outputAxes4, x, y3);
            title(outputAxes4, 'RHS');
```

Code Block 15. Updating the Output Signal Plots

In this section, the code updates the output signal plots in the graphical user interface (GUI) after performing the time variance test for discrete signals. It employs the `stem` function to plot the shifted signal (`y2`) on `outputAxes2` and labels it as the "Signal after Shift." Subsequently, it plots the LHS (left-hand side) of the time variance equation (`y2`) against the original time values (`x`) on `outputAxes3` and labels it as "LHS." Lastly, it plots the RHS (right-hand side) of the equation (`y3`) against the original time values (`x`) on `outputAxes4` and labels it as "RHS." These visualizations aid in the interpretation and assessment of the time variance characteristics of the discrete signal under consideration.

```
% Update user input information text
            dispLabelString = sprintf('%s\n%s\n%s\n%s', ...
                ['User Input Expression: ' userExpression], ...
                ['Signal Type: ' signalType], ...
                ['Amplitude: ' num2str(amplitudeValue)], ...
                ['Time Shift: ' num2str(timeShiftValue)]);
            set(dispLabel, 'String', dispLabelString);
```

Code Block 16. Updating User Input Information

This code segment is responsible for updating the text displayed in the user input information area (`dispLabel`) of the GUI. It constructs a string (`dispLabelString`) containing information about the user's input, including the expression, signal type, amplitude, and time shift. The `sprintf` function is used to format this information neatly. Subsequently, the `set` function is employed to update the text displayed in the `dispLabel` with the newly formatted string. This ensures that the user is informed about the parameters and characteristics of the signal being analyzed in the GUI.

```
% Display whether the system is time-variant or time-invariant
            timeVarianceText = '';
            if isequal(y2, y3)
                timeVarianceText = 'The system is time-invariant.';
            else
                timeVarianceText = 'The system is time-variant.';
            end
% Display the time variance information
            disp(timeVarianceText);
            annotation(mainFig, 'textbox', [0.75, 0.1, 0.8, 0.1], 'String',
timeVarianceText, 'FitBoxToText', 'on');
```

Code Block 17. Displaying Time Variance Information

This section determines whether the system is time-variant or time-invariant by comparing two signals, `y2` and `y3`. The result is stored in `timeVarianceText`. It is displayed using both the `disp` function in the command line and a textbox annotation in

the GUI. The textbox dynamically adjusts its size to fit the text, providing users with clear feedback on the system's time characteristics.

```
else % Continuous case
          % Use the input signal directly for continuous case
          % Update the input signal plot
          plot(inputAxes, x, inputSignal);
          title(inputAxes, 'Input Signal');
          y1 = inputSignal;
          % Interpolate the input signal for shifted time
          x2 = x + timeShiftValue;
          y2 = interp1(x, inputSignal, x, 'nearest', 0);
```

Code Block 18. Continuous Signal

In the continuous signal processing branch, the input signal is directly utilized without discretization. The code updates the input signal plot in the GUI using the `plot` function and sets the plot's title. The variable `y1` stores the input signal, and `x2` is computed by shifting the time values in `x` by `timeShiftValue`. The resulting `y2` is obtained by interpolating the shifted time values in `x2`. This continuous case is visualized in the GUI, showcasing the impact of the time shift on the continuous input signal.

User Interface

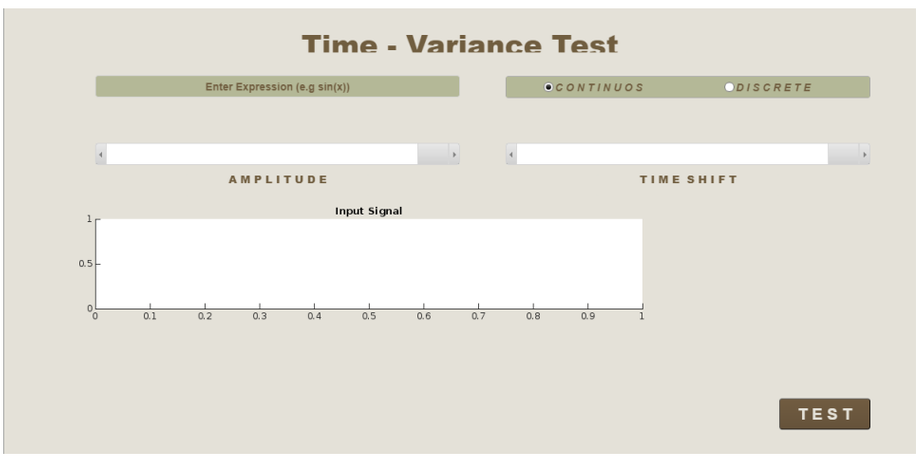


Figure 2. Initial Interface of the GUI

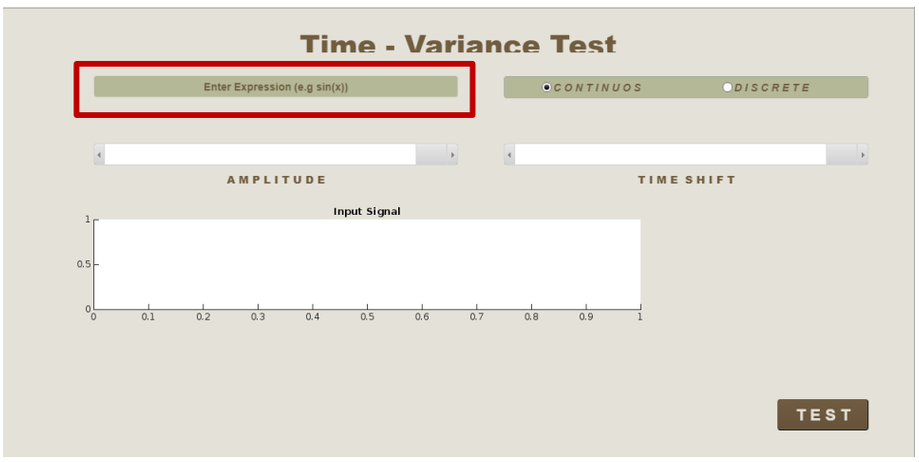


Figure 3. Edit Box for Entering System Expression

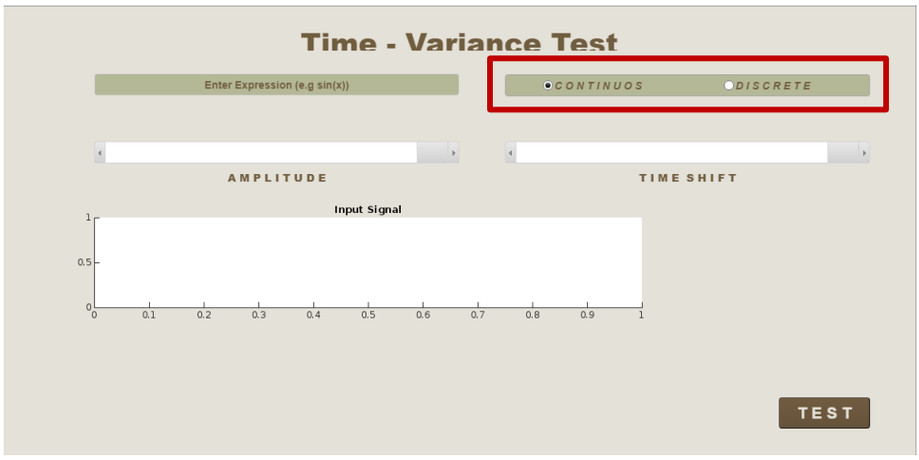


Figure 4. Radio Buttons to Choose the Signal Type

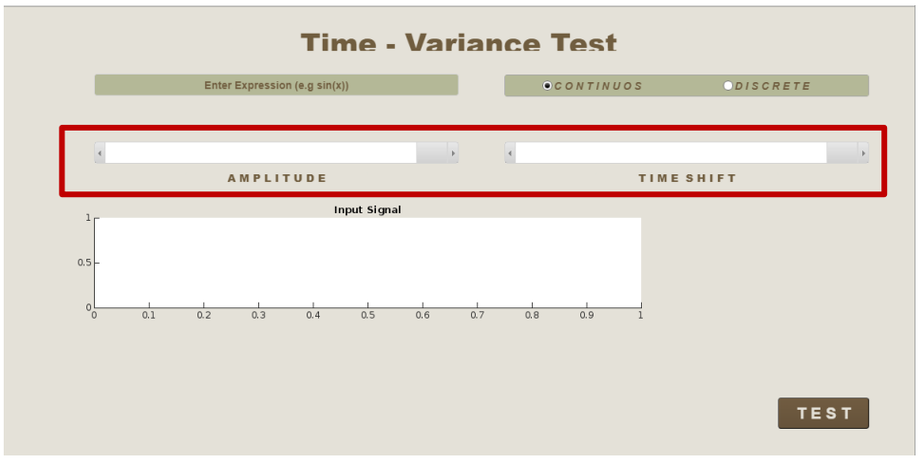


Figure 5. Slides Panels for Amplitude and Time Shift

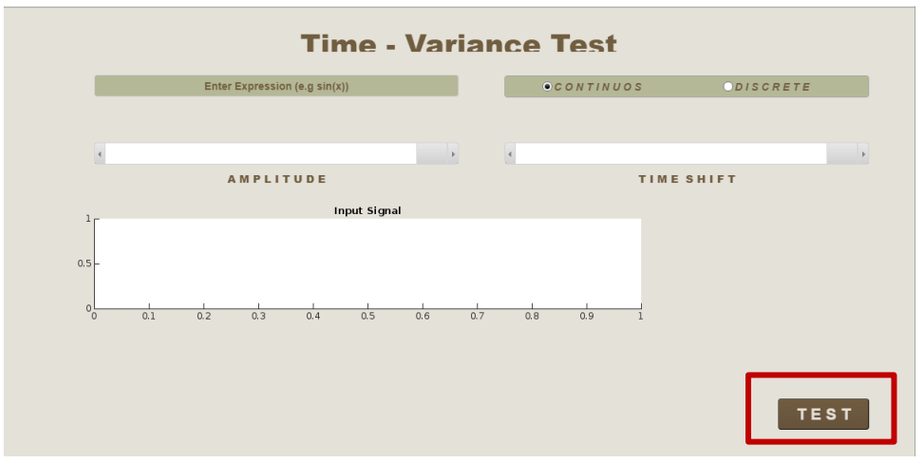


Figure 6. Test Button

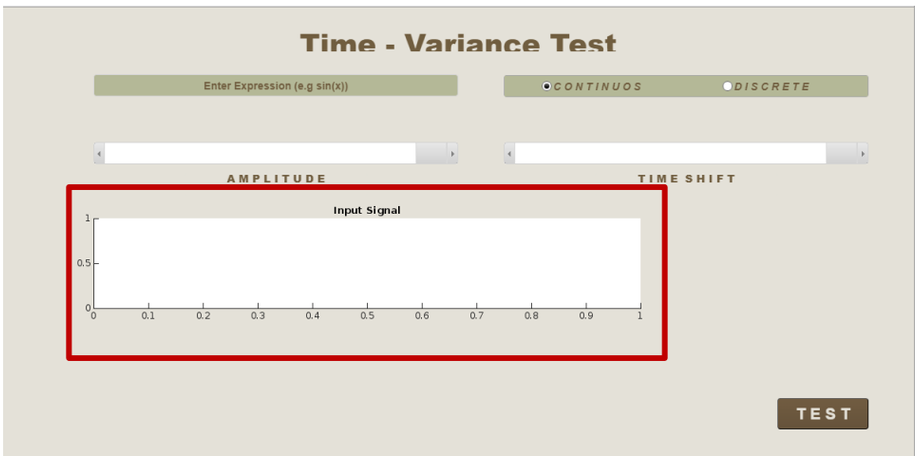


Figure 7. Box plot for input signal

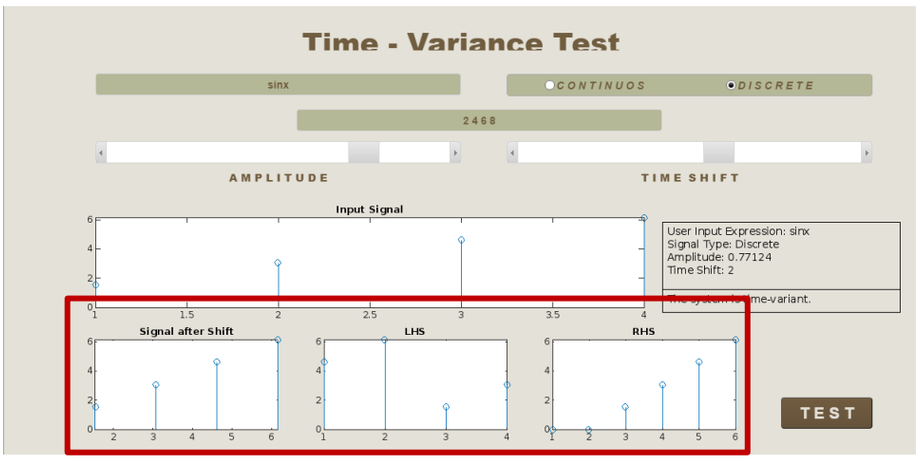


Figure 8. Box plots for output signal

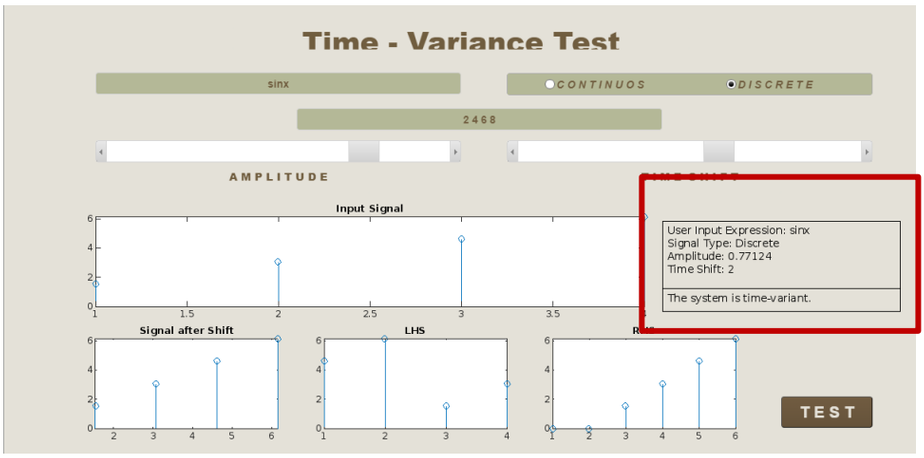


Figure 9. Display Box for Output and Input

Sample Output

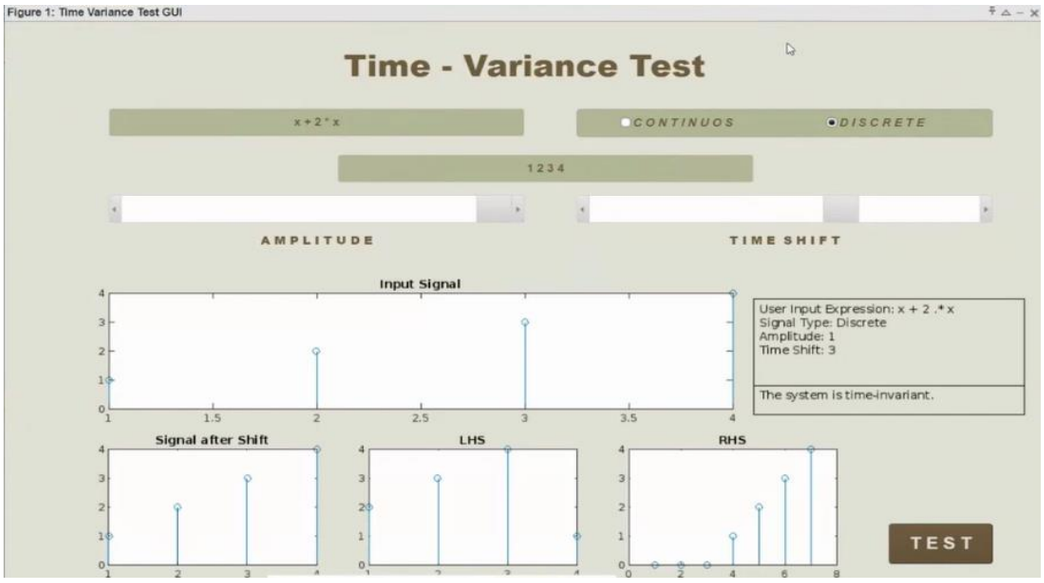


Figure 10. Sample 1 for Discrete Time Invariant

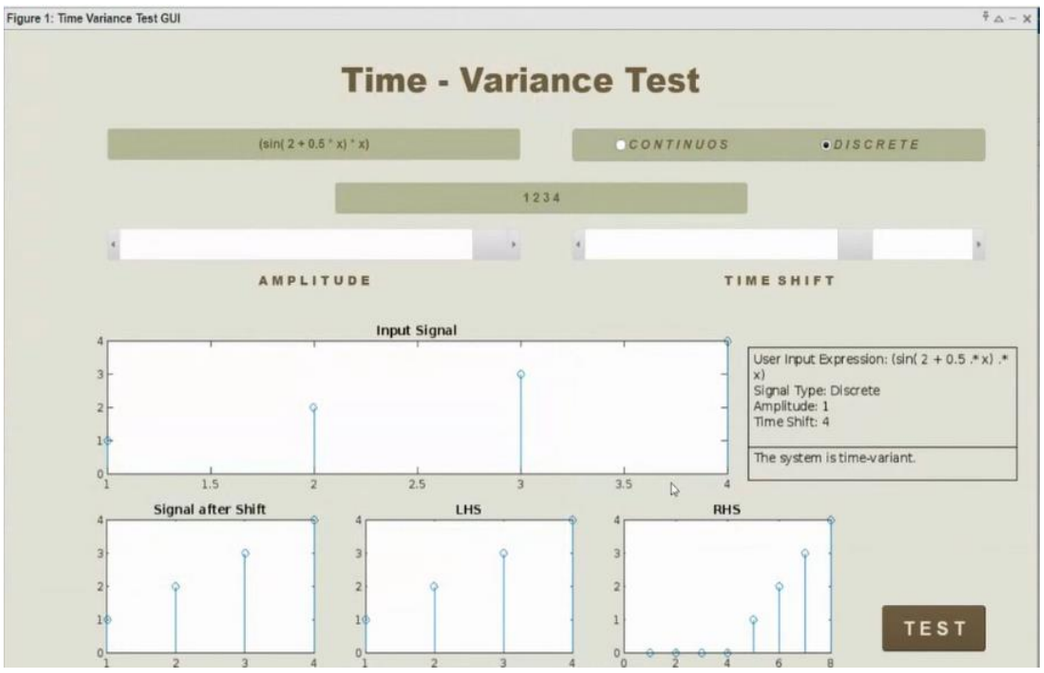


Figure 11:Sample 2 for Discrete Time-Variant

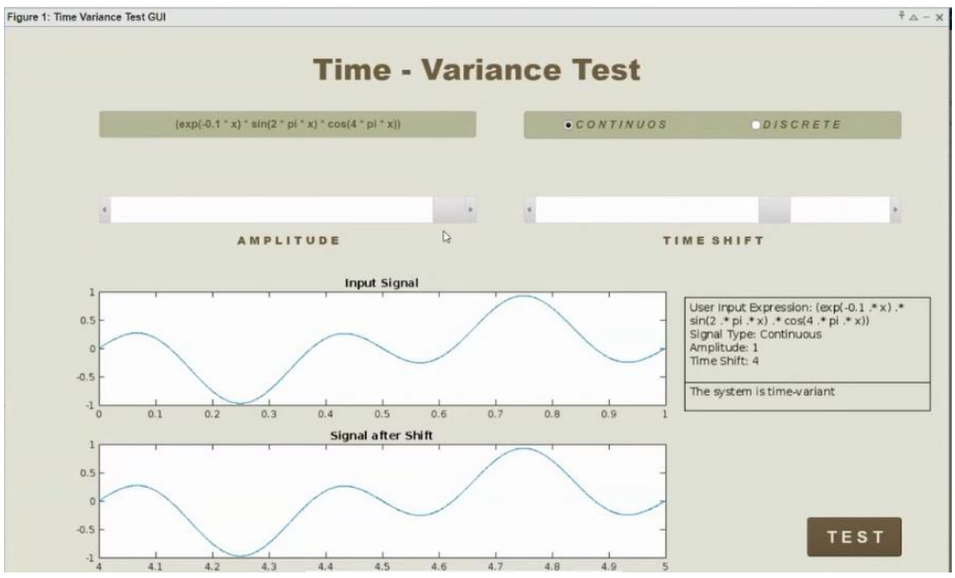


Figure 12: Sample 3 for Continuous Time-Variant

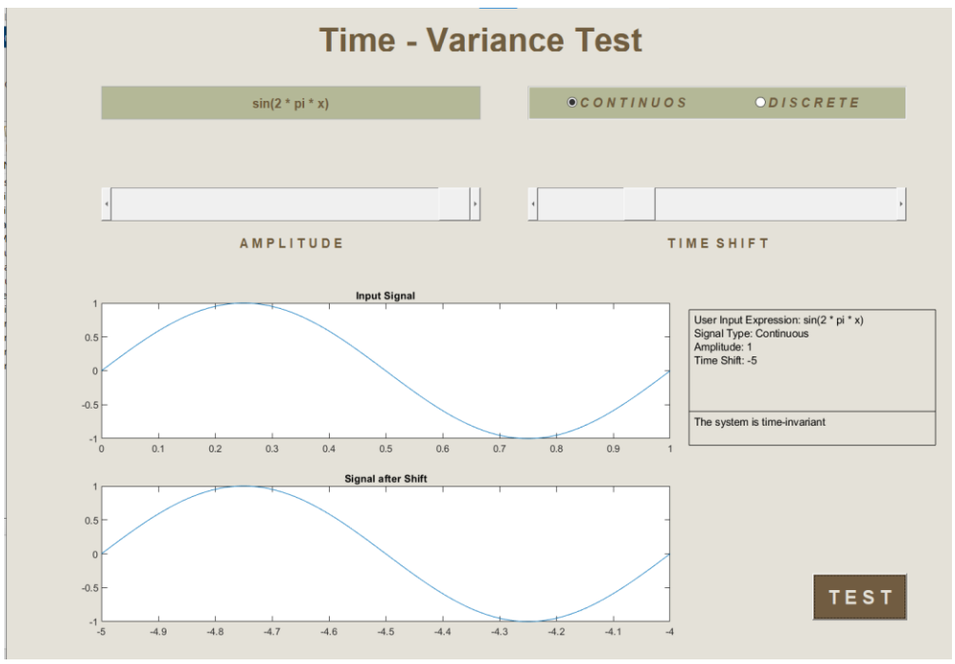


Figure 13: Sample 4 for Continuous Time-Invariant

CONCLUSION

The system initiates by prompting the user to input an expression, allowing them to determine whether the signal is discrete or continuous. For discrete signals, the user provides the input sequence, while for continuous signals, they specify the time delay and amplitude. The system then proceeds to exhibit both the input and output signals, alongside determining the time variance—clarifying whether the signal is time variant or time invariant.

The presented flowchart offers a concise and lucid overview of the entire process involved in creating and visualizing a time delay signal. This tool proves valuable for individuals seeking comprehension in this domain.

Moreover, the provided code adeptly illustrates time variance concepts through interactive visualization and practical testing. Its utility extends to facilitating an in-depth exploration of time variance within signal processing systems.

To further enhance its functionality, considerations for additional features could include support for more intricate signal types, options to analyze diverse system behaviors, and potential integration with other signal processing tools.

In summary, understanding the concepts of continuous and discrete signals, discrete-time systems, and time variance holds paramount importance across various applications, such as digital signal processing and GUI development. Continuous signals find suitability in analog systems, while discrete signals are indispensable for their digital counterparts. The discrete-time systems, pivotal in modeling the relationship between input and output signals within the digital domain, showcase properties like time variance that significantly influence their behavior and performance. Notably, time-invariant systems are favored for their consistent behavior and simplified analytical procedures.

APPENDICES

First Meeting

Date: January 5, 2023

Time Started: 7:30 pm

Time Ended: 11:30 pm

Venue: Google Meet

Meeting Attendees:

- Amat, Juliana Louise A.

- Bagacina, Alexis O.
- Bolo, Lyza April P.
- Carreos, Clarice Mae G.
- Corpuz, Reimarc G.
- Lariba, Aron P.
- Mostoles, Ivan M.
- Oba, Romina Giane Z.
- Reyes, Realyn L.

Agenda:

- Topic Selection
- Task Distribution

Discussion:

At exactly 7:30 PM on January 5, 2023, the meeting commenced. Realyn Reyes initiated the discussion. She asked everyone which among the topics or experiments studied in the laboratory we would like to create a GUI for. Each topic was deliberated, with each one discussed in detail. In the end, discrete-time signal properties were chosen. The properties to be used and incorporated into the GUI were determined to be linearity, time variance, and causality of a signal.

With the topic selection finalized, the discussion moved on to assigning tasks to the remaining members. Here is the list of tasks and the individuals assigned to each one:

- Back-end Creation

  Assigned to: Realyn L. Reyes

  Romina Giane Z. Oba

- Front-end or GUI Creation

  Assigned to: Clarice Mae G. Carreos

  Aaron P. Lariba

- Documentation:

  Assigned to: Lyza April P. Bolo

  Juliana Louise A. Amat

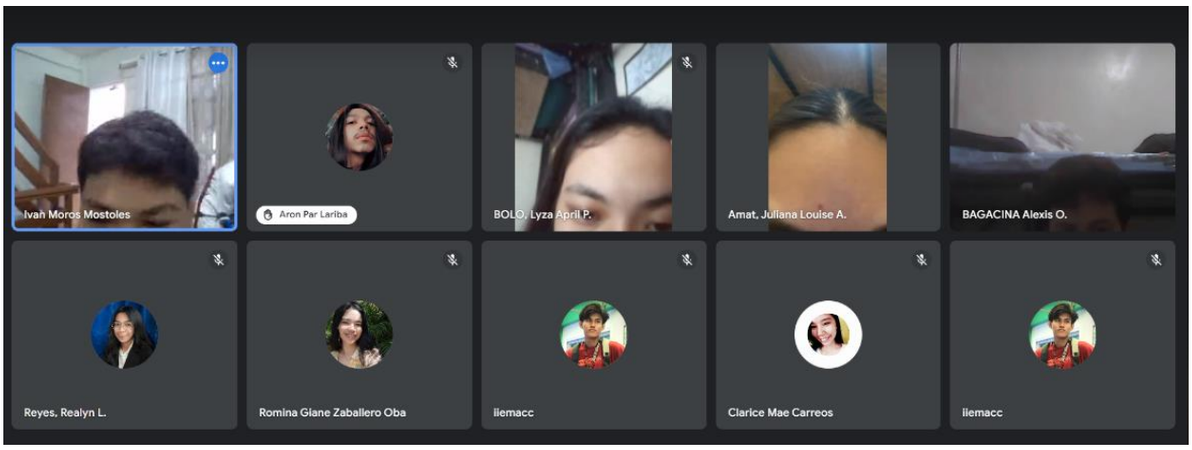
  Alexis O. Bagacina

- Presentation

  Assigned to: Reimarc G. Corpuz

  Ivan M. Mostoles

After everything was settled, and after further reminders, the meeting was concluded by Romina Giane Ob, who mentioned that the details of the next meeting would be announced in the group chat.



Second Meeting

Date: January 12, 2023

Time Started: 7:30 pm

Time Ended: 9:30 pm

Venue: Google Meet

Meeting Attendees:

- Amat, Juliana Louise A.
- Bagacina, Alexis O.
- Bolo, Lyza April P.
- Carreos, Clarice Mae G.
- Corpuz, Reimarc G.
- Lariba, Aron P.
- Mostoles, Ivan M.
- Oba, Romina Giane Z.
- Reyes, Realyn L.

Agenda:

- Task Update for Each Assigned Member

Discussion:

The main discussion of this meeting revolves around updating everyone on the progress of the tasks assigned to them. Here the updates of each of them:

- Back-end Creation

  Realyn L. Reyes:

Started working on making things function. Currently putting different parts together.

Romina Giane Z. Oba:

Figured out which tools to use and is now setting up the place where everything will happen.

- Front-end or GUI Creation

Clarice Mae G. Carreos:

Decided how the app will look and started building the basic structure.

Aaron P. Lariba:

Make sure the app looks good on different screens and is talking with the back-end team.

- Documentation

Lyza April P. Bolo:

Started writing down what needs to be in the project documents. Planning to finish the first draft soon.

Juliana Louise A. Amat:

Wrote down all the technical details about how the back-end part works.

Alexis O. Bagacina:

Putting together information for users about how to use the GUI.

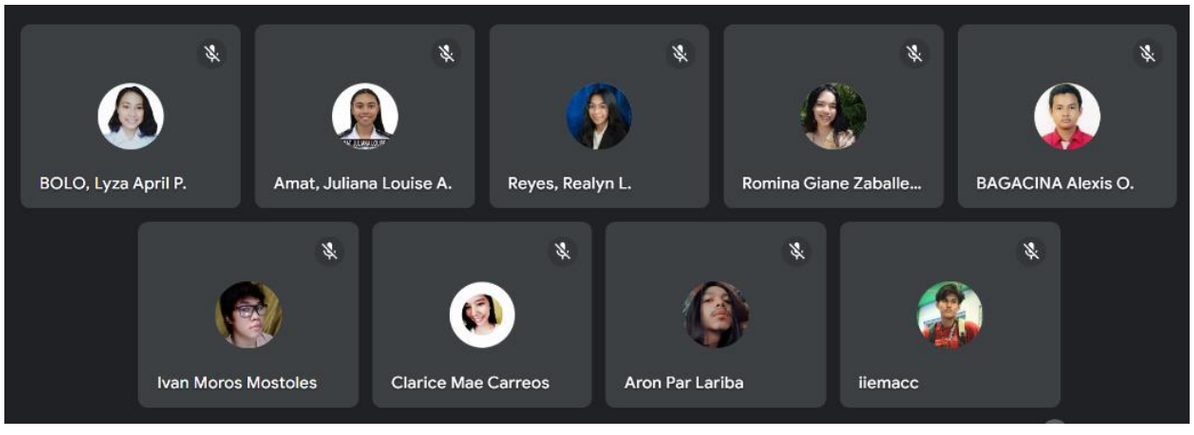
- Presentation

Reimarc G. Corpuz:

Decided on key visuals for the presentations.

Ivan M. Mostoles:

Created slides for the presentation but not yet done because the content in the document is not finished.



Third Meeting

Date: January 18, 2023

Time Started: 1:00 pm

Time Ended: 10:00 pm

Venue: Google Meet

Meeting Attendees:

- Amat, Juliana Louise A.
- Bagacina, Alexis O.
- Bolo, Lyza April P.
- Carreos, Clarice Mae G.
- Corpuz, Reimarc G.
- Lariba, Aron P.
- Mostoles, Ivan M.
- Oba, Romina Giane Z.
- Reyes, Realyn L.

Agenda:

- Finalization of the Entire Project

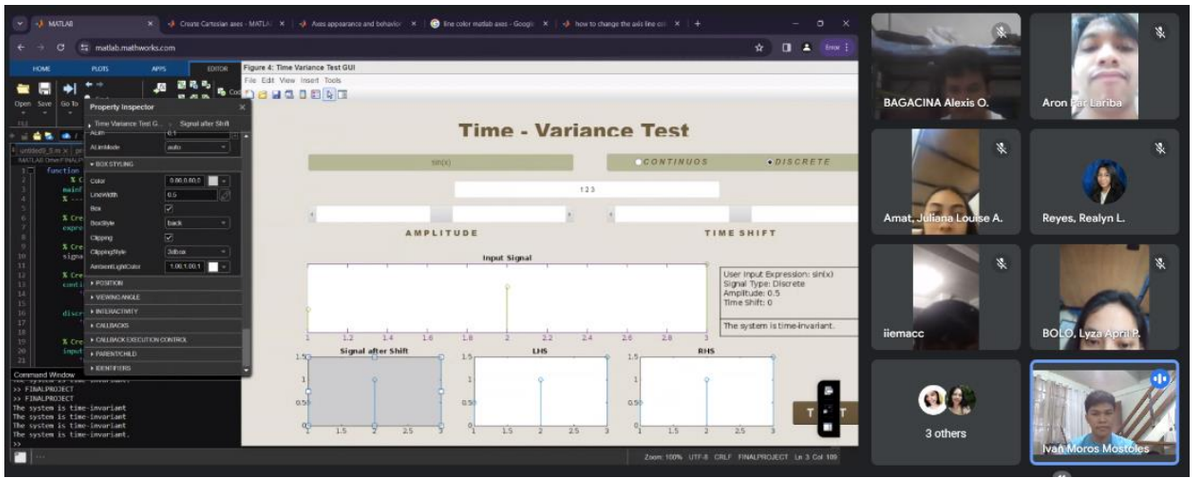
Discussion:

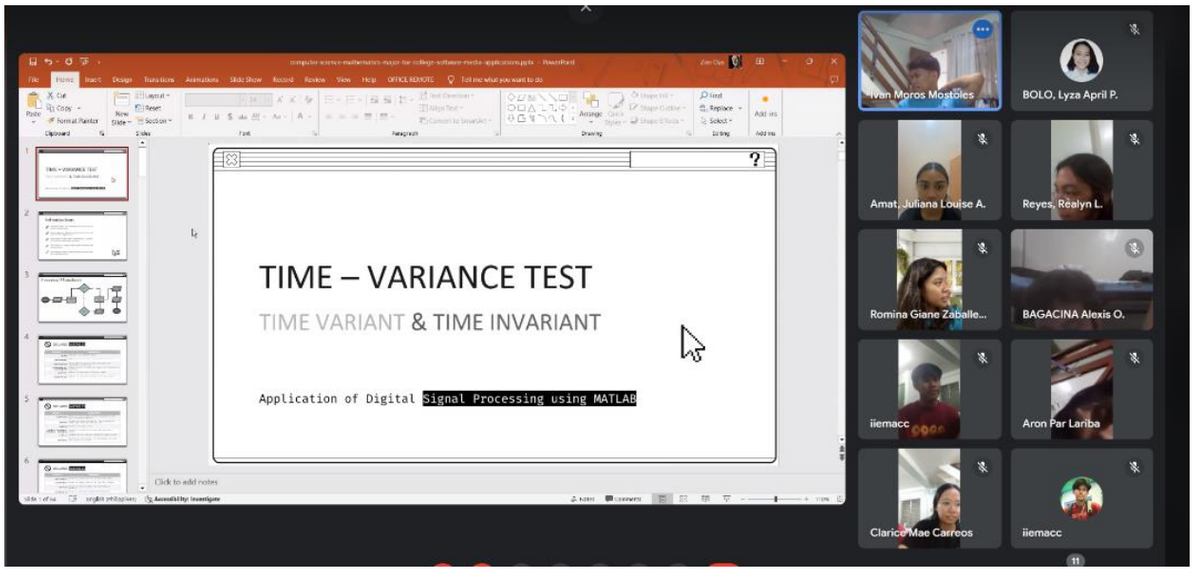
Since the deadline is approaching, everyone is busy finishing their tasks. To monitor the progress, this meeting was organized. It was already around 8 in the evening when the two individuals assigned to back-end creation completed their task. They demonstrated to everyone that the code is fully functional.

Afterward, the two individuals assigned for GUI creation took over. They are currently refining the GUI, enhancing its appearance, as it is the only remaining issue to consider the app project officially completed.

Regarding the documentation, they are waiting for the complete MATLAB code to explain each function. In the meantime, they are studying other snippets of the code to understand how it functions.

The meeting concluded as only a few tasks are left to finish on the GUI part. Everyone decided to continue and complete the remaining tasks the next day.


Time - Variance Test
CONTINUOS
DISCRETE
AMPLITUDE
TIME SHIFT
Input Signal
Signal after Shift
LHS
RHS
BAGACINA Alexis O.
Aron Par Lariba
Amat, Juliana Louise A.
Reyes, Realyn L.
iiemacc
BOLO, Lyza April P.
3 others
Ivan Moros Mostoles


TIME – VARIANCE TEST
TIME VARIANT & TIME INVARIANT
Application of Digital Signal Processing using MATLAB
Ivan Moros Mostoles
BOLO, Lyza April P.
Amat, Juliana Louise A.
Reyes, Realyn L.
Romina Giane Zaballe...
BAGACINA Alexis O.
iiemacc
Aron Par Lariba
Clarice Mae Carreos
iiemacc