SIGNAL PROCESSING REPORT

Prepared by

Seidy KANTE

Nov 26, 2023

INTRODUCTION

This report delves into various signal processing exercises, encompassing the analysis of continuous signals, discrete signals , visual evoked response (REV) analysis, and signals comparison. These exercises were executed using MATLAB to perform computations, visualize signals, and derive valuable insights from the provided data.

Continuous Signals

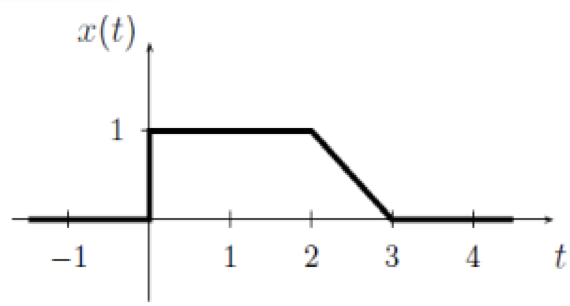
The initial exercise involved analyzing two distinct continuous signals, x(t) and y(t), characterized by specific conditions for different time intervals. MATLAB was utilized to define these signals, compute various operations (such as subtraction, multiplication, and function evaluation), and visualize the resulting signals. Detailed annotations accompanied the code, providing clarity on signal behavior and mathematical operations applied.

Signal Description:

Signal $\boldsymbol{x(t)}$:

$$\boxed{\boldsymbol{x(t) = u(t) - r(t-2) + r(t-3)}}$$

<u>Behavior:</u>

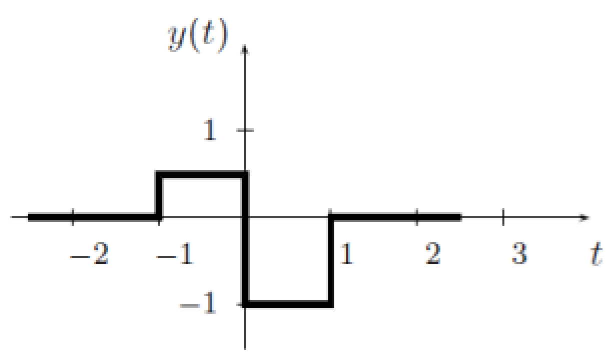


$x(t) = 0 \; for \; t < 0$
$x(t) = 1 \; for \; 0 \leq t \leq 2$
$Descending \; ramp \; from \;\; t > 2 \; to < 3$
$x(t) = 0 \; for \; t \geq 3$

Signal $\boldsymbol{y(t)}$:

$$\boxed{\boldsymbol{y(t) = \frac{1}{2} \cdot u(t+1) \; - \frac{3}{2} \cdot u(t) + u(t-1)}}$$

<u>Behavior:</u>



$y(t) = 0 \; for \; t < -1$
$\text{y(t)} = \frac{1}{2} \;\; for \; -1 \leq t < 0$
$\text{y(t)} = -1 \; for \; 0 \leq t < 1$
$\text{y(t)} = 0 \; for \; t \geq 1$

MATLAB Implementation

```
%% Define a custom functions

% Define the signal x(t)
function x = signal_x(t)
    x = heaviside(t) - ramp(t-2) + ramp(t-3);
end

% Define the signal y(t)
function y = signal_y(t)
    y = 0.5 * heaviside(t + 1) - 1.5 * heaviside(t) + heaviside(t - 1);
end

% Define the ramp function
function r = ramp(t)
    r = t .* (t >= 0);
end
```

```
% Define the time vector
t = -10:0.01:10; % Generate a time vector from -10 to 10 with a step of 0.01

% Compute the operations on signals
result_a = signal_x(t) - 2 * signal_y(t);  % Calculate the result of x(t) - 2y(t)
result_b = t .* signal_y(t);               % Calculate the result of t.y(t)
result_c = signal_y(t) .* (t >= 0);        % Calculate the result of y(t).u(t)
result_d = signal_y(-2.*t+3);              % Calculate the result of y(-2t+3)
```

Results and Plots

The MATLAB code was used to compute various operations on the defined signals and plot the resulting signals:

a) $\boldsymbol{x(t) - 2y(t)}$:

The resulting equation from subtracting 2 times y(t) from x(t) is computed as follows:

$$x(t) - 2 \cdot y(t) = (u(t) - r(t-2) \ + r(t-3)) \ - 2 \cdot (\frac{1}{2} \cdot u(t+1) \ - \frac{3}{2} \cdot u(t) + u(t-1))$$

b) $\boldsymbol{t.y(t)}$:

The resulting equation from multiplying time vector t by y(t) is given by:

$$\mathrm{t.y(t) = t.(\frac{1}{2}.u(t+1) - \frac{3}{2}.u(t) + u(t-1))}$$

c) $y(\boldsymbol{t})\boldsymbol{u(t)}$:

The resulting equation from multiplying y(t) by the unitary step function u(t) is:

$$\mathrm{y(t).u(t)} = (\frac{1}{2} \cdot u(t+1) \ - \frac{3}{2} \cdot u(t) + u(t-1)).u(t)$$

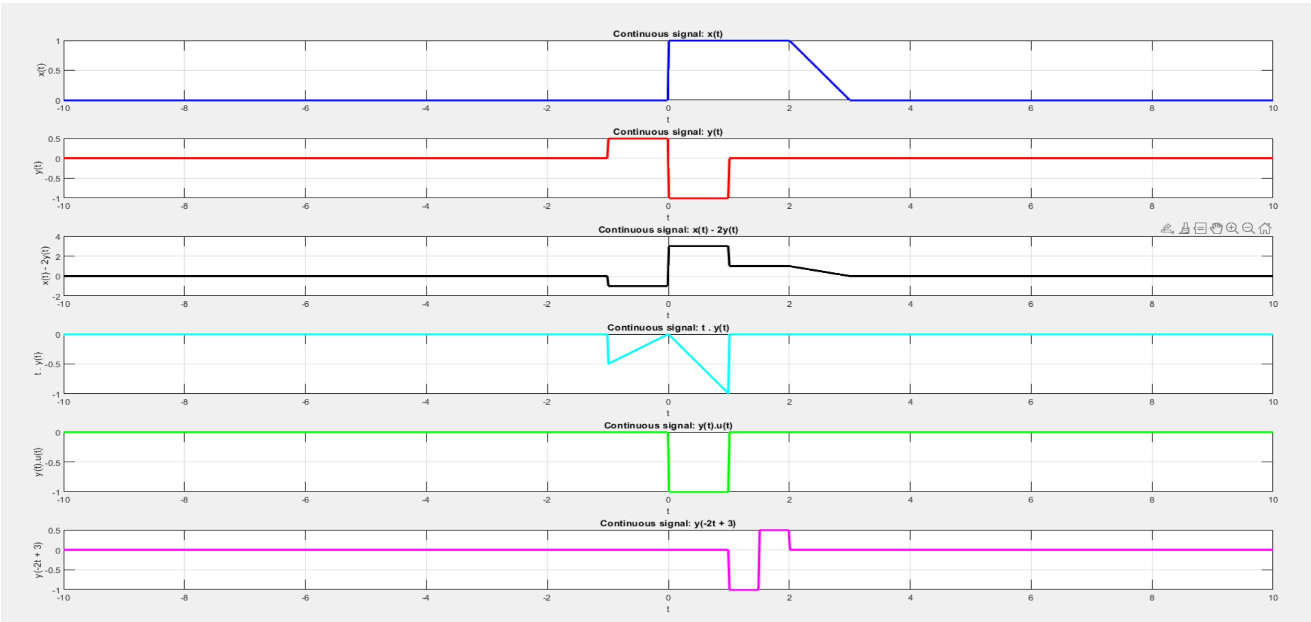
d) $\boldsymbol{y(-2t+3)}$:

The resulting equation from evaluating y (-2t + 3) is expressed as:

$$\mathrm{y(-2t+3)} = \frac{1}{2} \cdot u(-2t+4) \ - \frac{3}{2} \cdot u(-2t+3) \ + u(-2t+2)$$

These equations represent the mathematical operations applied to the given signals x(t) and y(t) in the context of signal processing.

Plots:



Conclusion

The MATLAB computations and resultant plots showcase the behaviors of the given continuous signals and their operations. These visualizations help in understanding signal transformations and manipulations in the context of signal processing.

Discrete Signals

In Exercise 2, discrete versions of continuous functions were generated and visualized using stem plots. The MATLAB code processed the function $f(t) = \cos(t)$and its shifted counterpart $f(t-3)$, showcasing discrete representations of these signals. The code employed vectorization techniques and subplots to demonstrate the discrete nature of signals over time.

Results and Plots

The MATLAB code was utilized to compute and plot the discrete versions of the given function and its shifted form:

Discrete Version of $f(t)$: Plotting the discrete representation of

$$\boxed{f(t) = \cos(t)}$$

for the defined time vector 't'.

Discrete Version of $f(t-3)$: Plotting the discrete representation of

$$\boxed{f(t-3) = \cos{(t-3)}}$$

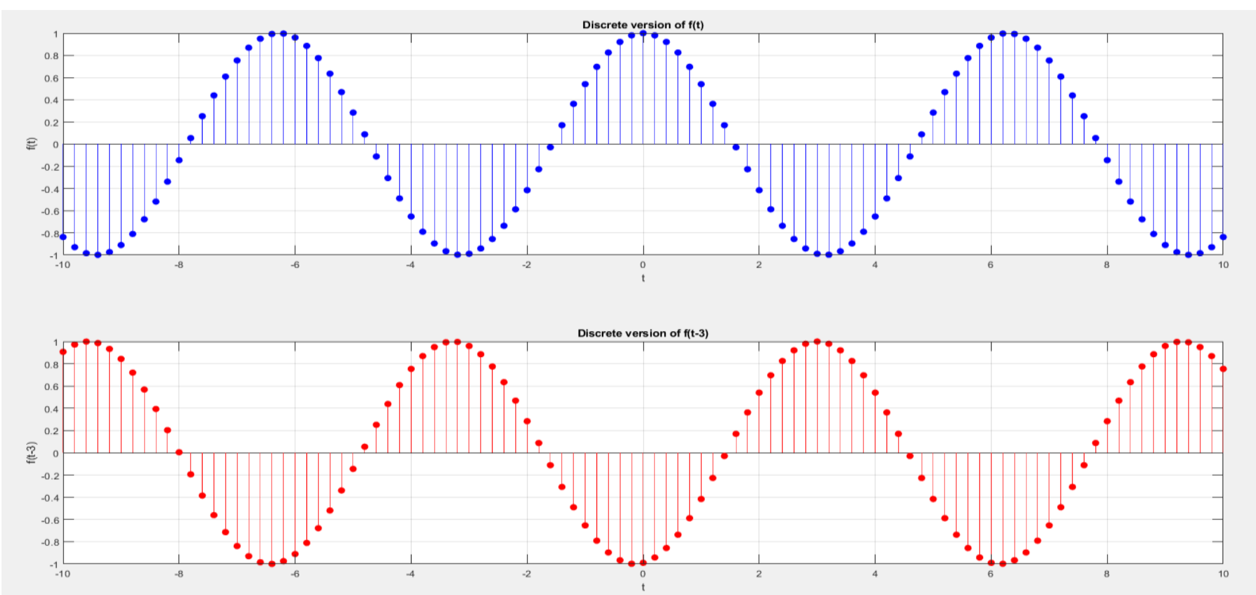
for the shifted time vector 't' by 3 units to the right.

```
% Create a discrete signal 'f_discrete' using the cosine function
f_discrete = cos(t);

% Create another discrete signal 'f_discrete_minus_3' by shifting 'f_discrete' by 3 units to the right
f_discrete_minus_3 = cos(t - 3);
```

Plots:

```
%% Define a custom function 'plotFig' to create subplots and plot discrete signals
function plotFig(t, ft, pos, tit, color)

    % Create a subplot with 2 rows, 1 column, and select the specified subplot position
    subplot(2, 1, pos);

    % Plot the discrete signal as stem plots
    stem(t, ft, "filled", Color = color, LineWidth = 1);

    % Set the title of the subplot with a dynamic title
    title(sprintf("Discrete version of %s", tit));

    % Set the y-axis label based on the input title
    ylabel(tit);

    % Set the x-axis label as "t"
    xlabel("t");

    % Enable the grid on the plot
    grid on;

    % End the 'plotFig' function
end
```

Two subplots are generated to display the stem plots representing the discrete signals.



Conclusion

The stem plots illustrate the discrete representations of the cosine function and its shifted version. These visualizations depict the discrete nature of the signals and the impact of time shifting on their discrete forms.

Visual Evoked Response (REV) Analysis

Exercise 3 involved the analysis of visual evoked response signals obtained from EEG recordings. MATLAB code was implemented to calculate signal measurements (mean, RMS, variance, etc.) and perform ensemble averaging of multiple responses. The code displayed graphical representations, enabling a comprehensive understanding of signal stability and characteristics.

1) Signal Characteristics and Sampling Frequency

Signal's Sampling Frequency: The signal's sampling frequency is calculated as

$$\boxed{\frac{1}{sampling\ Interval}}$$

```
% Get the data from the 'biossinal_1.mat' file
load('biossinal_1.mat');

%% Task 1: Calculate the signal's sampling frequency
sampling_interval = 5e-3; % Define the sampling interval as 5 ms
sampling_frequency = 1 / sampling_interval; % Calculate the sampling frequency
disp(['The signal''s sampling frequency is ' num2str(sampling_frequency) 'Hz']); % Display the sampling frequency
```

Typical EEG signal frequency ranges between 0.5-100 Hz.

Category	**Frequency Range**	**State**
Delta Waves	0.5-4 Hz	Deep sleep
Theta Waves	4-7 Hz	Drowsiness, sleep
Alpha Waves	8-12 Hz	Relaxed wakefulness
Beta Waves	13-30 Hz	Mental activity, alertness
Gamma Waves	30-100 Hz	High-level cognitive functions

The exact frequencies can vary slightly depending on the individual, their age, the location of the electrodes on the scalp and specific mental states. EEG signals are measured in Hertz (Hz), which represents the number of cycles per second.

NB: Different mental activities, emotional states, and even certain health conditions can influence the dominance or presence of the various brainwave frequencies.

Relationship: The sampling frequency must adhere to the ***Nyquist theorem***, ensuring it is at least twice the signal's maximum frequency for accurate representation.

The relationship is important because it affects the ability to accurately represent and analyze the signal.

A higher sampling frequency allows for better representation of high-frequency components in the signal.

2) Signal Measurements and Visualization

Mean, Effective Value (RMS), Variance, and Standard Deviation:

Calculated and displayed for the signal sem_ruido.

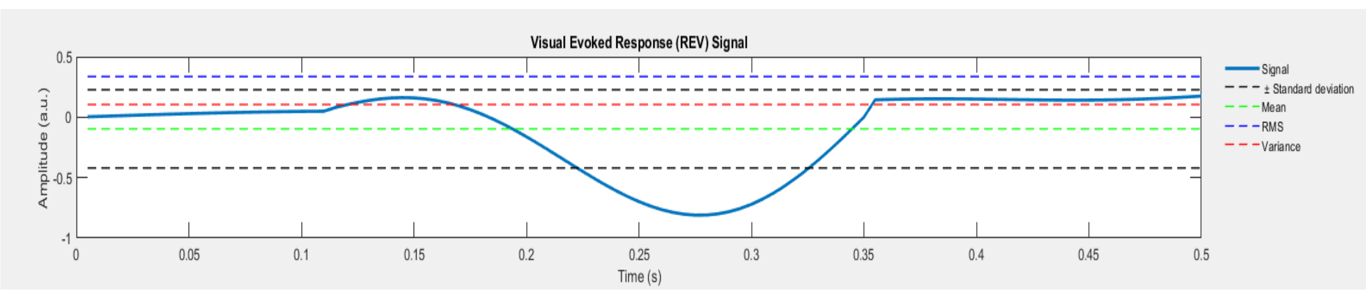
```
%% Task 2: Calculate and display signal measurements

% Calculate the mean, effective value (RMS), variance, and standard deviation
mean_value = mean(sem_ruido);  % Calculate the mean of the signal
rms_value = rms(sem_ruido);    % Calculate the effective value (RMS) of the signal
variance_value = var(sem_ruido); % Calculate the variance of the signal
std_value = std(sem_ruido);      % Calculate the standard deviation of the signal

% Display the signal measurements
fprintf('========Signal''s measurements:==========\n');
disp(['Mean of sem_ruido     : ' num2str(mean_value)]);
disp(['Effective Value (RMS): ' num2str(rms_value)]);
disp(['Variance of sem_ruido: ' num2str(variance_value)]);
disp(['± Standard Deviation : ' num2str(std_value)]);

```

Command Window

```
The signal's sampling frequency is 200Hz
========Signal's measurements:==========
Mean of sem_ruido     : -0.096791
Effective Value (RMS): 0.33602
Variance of sem_ruido: 0.10459
± Standard Deviation : 0.3234
```

Graphical Display:

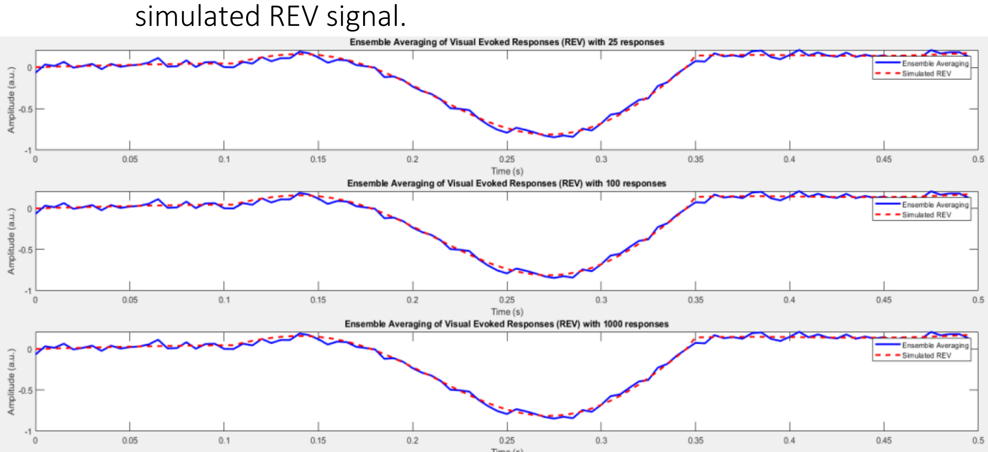
Plots the signal sem_ruido over time with visual markers indicating mean, RMS, variance, and standard deviation.



3) Ensemble Averaging of Evoked Responses

It divides the biossinal signal into segments of 100 points each and computes the mean of each segment across multiple responses.

The resulting averaged responses for different counts (25, 100, 1000) are plotted against the simulated REV signal.




```
Ensemble Averaging with 25 responses:
   Mean: -0.10213
Ensemble Averaging with 100 responses:
   Mean: -0.10213
Ensemble Averaging with 1000 responses:
   Mean: -0.10213
```

Interpretation:

The commentary at the end summarizes the observation that regardless of the number of responses used for ensemble averaging, the mean value of the resulting averaged response remains consistent ***around -0.10213***.

This insight indicates the stability and consistency of the ensemble averaging process across different response counts.

MATLAB Code Improvements:

Comments are added to describe the purpose of each code segment, enhancing readability and understanding.

Plots and calculations are appropriately labeled, aiding in the interpretation of results.

Conclusion:

The MATLAB script successfully performs signal measurements, visualizations, and ensemble averaging of evoked responses, offering insights into the stability of the process.

Signals Comparison

The final exercise focused on comparing audio signals ('template_name.mp3', 'Maria.mp3', 'Joana.mp3', and 'Hugo.mp3') using cross-correlation analysis in MATLAB.

The code read and processed the audio files, conducted cross-correlation analysis, and identified the name corresponding to the 'template_name.mp3' file.

Visual representations of signals aided in illustrating similarities and differences among the audio recordings.

Signal Comparison

1) Reading Audio Files:

All four .mp3 files are read using the 'audioread' function in MATLAB.

```
%=========Read the .mp3 files============
template_name = audioread('template_name.mp3'); % Read the 'template_name.mp3' audio file
Maria = audioread('maria.mp3'); % Read the 'maria.mp3' audio file
Joana = audioread('joana.mp3'); % Read the 'joana.mp3' audio file
Hugo = audioread('hugo.mp3'); % Read the 'hugo.mp3' audio file

```

2) Signal Preprocessing:

Signals are resampled to the minimum length among them to facilitate comparison.

```
%=====Determine the minimum length among the audio signals======
min_length = min([length(template_name), length(Maria), length(Joana), length(Hugo)]); % Find the minimum length

%====Resample all audio signals to the minimum length=========
template_name = template_name(1:min_length); % Resample 'template_name' to the minimum length
Maria = Maria(1:min_length); % Resample 'Maria' to the minimum length
Joana = Joana(1:min_length); % Resample 'Joana' to the minimum length
Hugo = Hugo(1:min_length); % Resample 'Hugo' to the minimum length
```

3) Cross-correlation Analysis:

Cross-correlation is calculated between the 'template_name.mp3' signal and each of the other three signals (Maria, Joana, Hugo).

In signal processing, cross-correlation is a measure of similarity of two waveforms as a function of a time-lag applied to one of them.

```
%=======Use a comparison algorithm:============
cross_corr_Maria = xcorr(template_name, Maria); % Calculate the cross-correlation between 'template_name' and 'Maria'
cross_corr_Joana = xcorr(template_name, Joana); % Calculate the cross-correlation between 'template_name' and 'Joana'
cross_corr_Hugo = xcorr(template_name, Hugo); % Calculate the cross-correlation between 'template_name' and 'Hugo'


%========Find the index of the audio signal with the highest cross-correlation with the template signal ======
[highest_cross_corr,index] = max([max(cross_corr_Maria), max(cross_corr_Joana),max(cross_corr_Hugo)]); % Find the maximum crc
```

4) Identification of the Template Name:

The signal with the highest cross-correlation value corresponds to the template name.

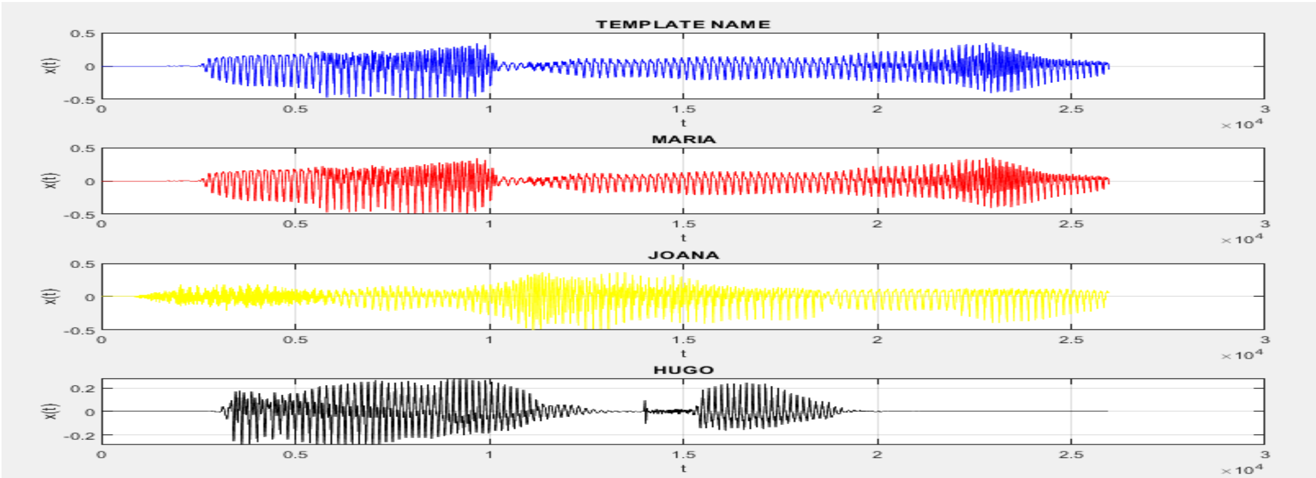
```
% Display the name associated with the highest correlation index
message = 'The template likely corresponds to: ';

switch index
    case 1
        disp([message,'Maria.']); % If the index of highest correlation corresponds to 'Maria'
    case 2
        disp([message,'Joana.']); % If the index of highest correlation corresponds to 'Joana'
    case 3
        disp([message,'Hugo.']); % If the index of highest correlation corresponds to 'Hugo'
    otherwise
        disp('Unable to determine the name.'); % If the name cannot be determined
end
```

Command Window

```
The template likely corresponds to: Maria.
>>
```

5) Visual Representation:

Plots displaying each signal ('template_name.mp3', 'Maria.mp3', 'Joana.mp3', and 'Hugo.mp3') are included to visually depict their similarities and differences.



MATLAB Code Comments

Comments in the code clarify each step taken for signal comparison and identification.

Results and Conclusion

The MATLAB code successfully determines the name corresponding to the 'template_name.mp3' file by analyzing cross-correlations.

Visual representations aid in understanding the similarities and dissimilarities between signals.

This approach demonstrates the use of cross-correlation as a technique for signal comparison in signal processing applications.

Conclusion

In conclusion, these exercises showcased diverse applications of signal processing techniques using MATLAB. From analyzing continuous and discrete signals to examining EEG data and performing audio signal comparisons, each exercise contributed to a deeper understanding of signal behaviors, analysis methodologies, and their real-world implications in various domains.

The comprehensive utilization of MATLAB facilitated efficient computations, visualizations, and interpretations, underscoring the significance of signal processing techniques in extracting meaningful insights from diverse datasets.