DEPARTMENT OF ELECTRICAL ENGINEERING
College of Electrical and Mechanical Engineering (CEME), NUST-Pakistan
B.E. Electrical Engineering

EE 330 Digital Signal Processing

Project Report

Cleaning and Real-Time Simulation of EEG Data Using Digital Filters

Date: 16-05-2025

Group Members	**Registration Number**	**Syndicate**
Muhammad Umer Sajid	413946	B
Shafi Ullah	423652	B
Saad Ahmed	432225	B

Software Usage (P3)	**Debugging and Results (P3)**	**Individual and Teamwork (A3)**	**Lab Report (A3)**	**Total**

Contents

1 Introduction **3**
- 1.1 Project Overview . . . 3
- 1.2 Objectives . . . 3
- 1.3 Brain Wave Frequency Bands . . . 3

2 Materials and Methods **3**
- 2.1 MATLAB Installation and Setup . . . 3
- 2.2 Dataset Description . . . 4
- 2.3 Digital Filter Design . . . 4
  - 2.3.1 Notch Filter . . . 4
  - 2.3.2 High-Pass Filter . . . 4
  - 2.3.3 Low-Pass Filter . . . 4
  - 2.3.4 Band-Pass Filters . . . 5

3 Implementation **5**
- 3.1 Data Preprocessing . . . 5
- 3.2 Filter Application Pipeline . . . 5
- 3.3 Real-Time Simulation . . . 5
- 3.4 Brain Wave Analysis . . . 6

4 Results and Discussion **6**
- 4.1 Filter Responses . . . 6
  - 4.1.1 Frequency Response Analysis . . . 6
- 4.2 Raw vs. Filtered EEG Data . . . 7
  - 4.2.1 Time Domain Analysis . . . 7
  - 4.2.2 Spectral Analysis . . . 7
- 4.3 Brain Wave Extraction . . . 8
  - 4.3.1 Isolated Frequency Bands . . . 8
- 4.4 Real-Time Simulation Results . . . 9
- 4.5 Brain Wave Distribution Analysis . . . 9

5 Performance Evaluation **10**
- 5.1 Filter Performance . . . 10
- 5.2 Real-Time System Responsiveness . . . 10

6 Conclusion **11**

A MATLAB Code **11**
- A.1 Complete Project Code . . . 11

B References **22**

1 Introduction

1.1 Project Overview

Electroencephalography (EEG) is a non-invasive technique used to monitor brain activity through the recording of electrical signals from the scalp. However, these signals are often contaminated with various types of noise, including line noise (50Hz/60Hz), motion artifacts, and other environmental interferences. This necessitates the application of robust digital signal processing techniques to clean the data and extract meaningful information.

This project focuses on implementing digital filtering techniques in MATLAB to process EEG data, with emphasis on noise removal and signal isolation within specific frequency bands corresponding to various brain activities. Additionally, a simulated real-time environment has been developed to demonstrate how these processing techniques would perform in a live setting.

1.2 Objectives

The primary objectives of this project are:

- To implement effective digital filters for removing noise from EEG signals
- To extract and analyze specific brain wave frequency bands (delta, theta, alpha, sigma, and beta)
- To simulate a real-time EEG processing and visualization environment
- To gain practical experience in biomedical signal processing using MATLAB

1.3 Brain Wave Frequency Bands

The human brain generates electrical activity at various frequencies, which are associated with different mental states and cognitive processes. The main frequency bands of interest in this project are:

Band	Frequency Range (Hz)	Associated Mental States
Delta	0.5 - 4	Deep sleep, unconsciousness
Theta	4 - 7	Drowsiness, meditation, creativity
Alpha	8 - 12	Relaxed wakefulness, closed eyes
Sigma	12 - 16	Sleep spindles, memory consolidation
Beta	13 - 30	Active thinking, focus, alertness

Table 1: Brain Wave Frequency Bands

2 Materials and Methods

2.1 MATLAB Installation and Setup

For this project, MATLAB R2022b was installed with the Signal Processing Toolbox, which provides essential functions for filter design and signal analysis. The installation process involved:

- Downloading the MATLAB installer from MathWorks website
- Running the installer and selecting the Signal Processing Toolbox
- Verifying the installation by checking the availability of key functions such as `designfilt`, `filtfilt`, and `pwelch`

2.2 Dataset Description

The EEG dataset used in this project is from subject s01, containing multi-channel recordings sampled at 256 Hz. The data is structured as a CSV file with each column representing an EEG channel and each row representing a time point. The dataset includes the following characteristics:

- Sampling frequency: 256 Hz
- Multiple channels recording various regions of the brain
- Raw data potentially contaminated with line noise (50 Hz) and other artifacts

For the analysis, 5 channels were selected to focus on different regions of the brain activity.

2.3 Digital Filter Design

2.3.1 Notch Filter

A notch filter was designed to specifically remove the 50 Hz line noise (power line interference) common in EEG recordings. The filter was implemented using a 4th order Butterworth band-stop design with the following specifications:

- Center frequency: 50 Hz
- Bandwidth: 4 Hz (48 Hz - 52 Hz)
- Design method: Butterworth

2.3.2 High-Pass Filter

A high-pass filter was implemented to remove DC offset and slow drifts (electrode drift) that can obscure the actual brain activity. The filter specifications were:

- Filter order: 4
- Passband frequency: 0.5 Hz
- Passband ripple: 0.1 dB

2.3.3 Low-Pass Filter

A low-pass filter was designed to remove high-frequency noise and limit the signal to the frequency range of interest for brain activity. The specifications were:

- Filter order: 4
- Passband frequency: 45 Hz
- Passband ripple: 0.1 dB

2.3.4 Band-Pass Filters

Separate band-pass filters were designed to isolate each frequency band of interest. All filters used 4th order Butterworth design with the following cut-off frequencies:

- Delta band: 0.5 Hz - 4 Hz
- Theta band: 4 Hz - 7 Hz
- Alpha band: 8 Hz - 12 Hz
- Sigma band: 12 Hz - 16 Hz
- Beta band: 13 Hz - 30 Hz

3 Implementation

3.1 Data Preprocessing

The initial preprocessing steps included:

1. Loading the EEG data from the CSV file
2. Transposing the data to have channels as rows (standard EEG format)
3. Selection of 5 specific channels for detailed analysis
4. Visualization of raw data to identify noise patterns

3.2 Filter Application Pipeline

The complete filtering pipeline involved the sequential application of filters in the following order:

1. Notch filter to remove line noise
2. High-pass filter to remove DC offset and slow drifts
3. Low-pass filter to remove high-frequency noise
4. Specific band-pass filters to isolate frequency bands of interest

This sequential approach ensures that the most disruptive noise components are removed first, allowing for better isolation of the brain activity signals in subsequent steps.

3.3 Real-Time Simulation

To simulate a real-time processing environment, the following approach was implemented:

1. Selection of a segment of data (10 seconds) for simulation
2. Creation of a moving time window (5 seconds) to display data
3. Sequential processing and visualization of data chunks

4. Updating the display at regular intervals (0.5 seconds) to mimic real-time data acquisition

The simulation incorporates visual elements to distinguish different frequency bands and displays the raw, filtered, and band-specific signals simultaneously.

3.4 Brain Wave Analysis

For quantitative analysis of brain activity, the power contribution of each frequency band was calculated using:

1. Squaring the filtered signals to obtain signal power
2. Summing the power across all time points for each band
3. Calculating the percentage contribution of each band to the total power
4. Visualizing the results using pie charts and bar graphs

4 Results and Discussion

4.1 Filter Responses

4.1.1 Frequency Response Analysis

The frequency responses of the designed filters were analyzed to verify their performance characteristics:

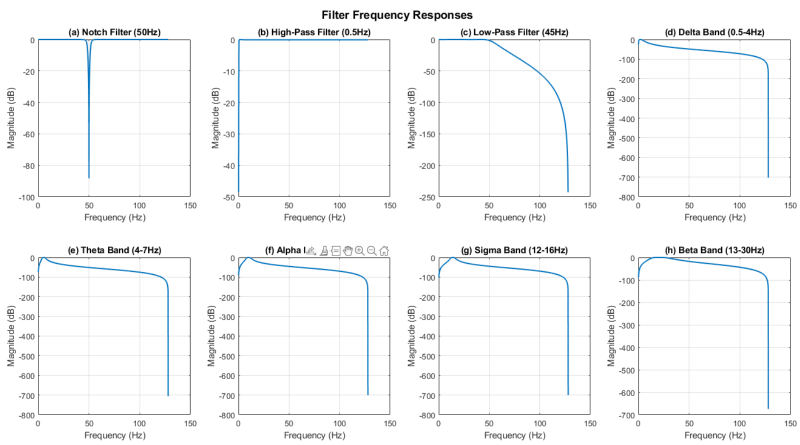

Figure 1: Frequency responses of the designed digital filters. (a) Notch filter, (b) High-pass filter, (c) Low-pass filter, (d) Delta band filter, (e) Theta band filter, (f) Alpha band filter, (g) Sigma band filter, (h) Beta band filter.

The notch filter effectively creates a narrow rejection band centered at 50 Hz, while the high-pass and low-pass filters show characteristic slopes at their respective cutoff frequencies. The band-pass filters demonstrate good selectivity for their designated frequency ranges.

4.2 Raw vs. Filtered EEG Data

4.2.1 Time Domain Analysis

The comparison between raw and filtered EEG signals reveals significant improvements in signal quality:

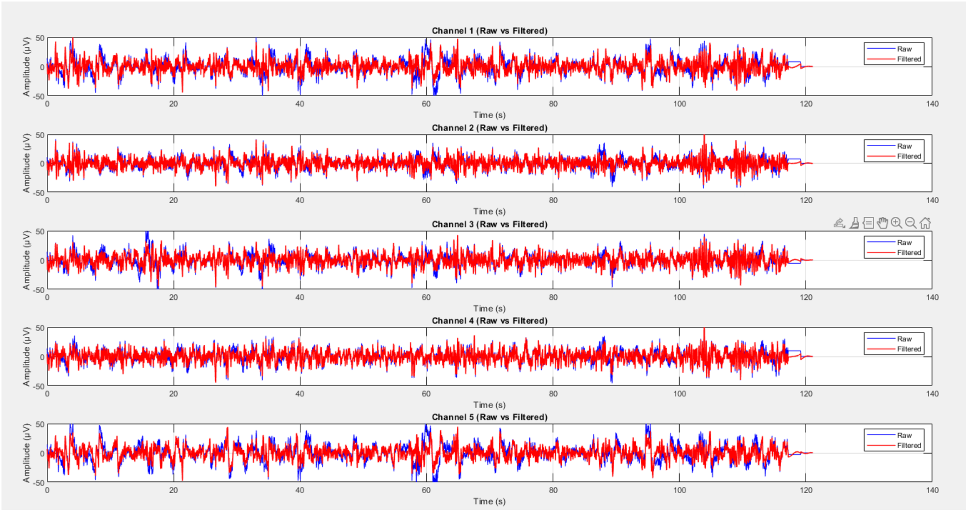

Figure 2: Comparison of raw (blue) and filtered (red) EEG signals for the selected channels. The filtered signals show reduced noise and clearer brain activity patterns.

The filtered signals exhibit reduced baseline drift and significantly less high-frequency noise, making the underlying brain activity patterns more discernible.

4.2.2 Spectral Analysis

Power spectral density analysis before and after filtering provides insight into the frequency content of the signals:

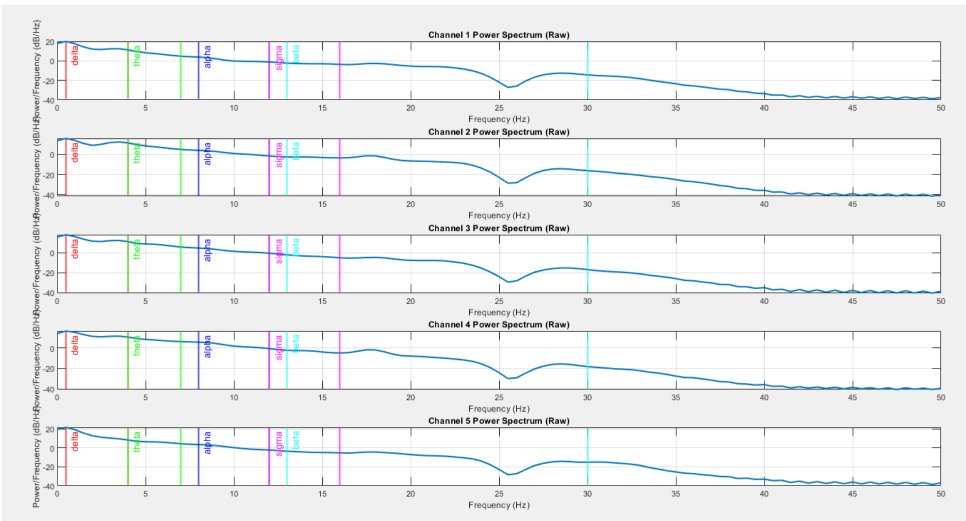

Figure 3: Power spectrum of raw EEG data showing significant line noise at 50 Hz and other noise components.

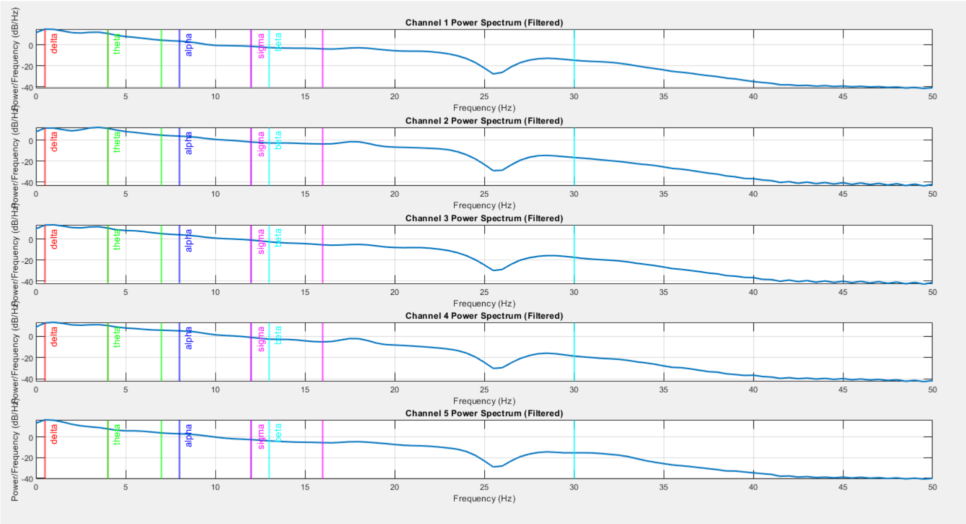

Figure 4: Power spectrum of filtered EEG data showing suppressed line noise and enhanced visibility of brain wave frequency bands.

The power spectra clearly demonstrate the effectiveness of the filtering process, with significantly reduced line noise at 50 Hz and enhanced visibility of the brain wave frequency bands.

4.3 Brain Wave Extraction

4.3.1 Isolated Frequency Bands

The band-pass filtered signals highlight the specific frequency components of brain activity:

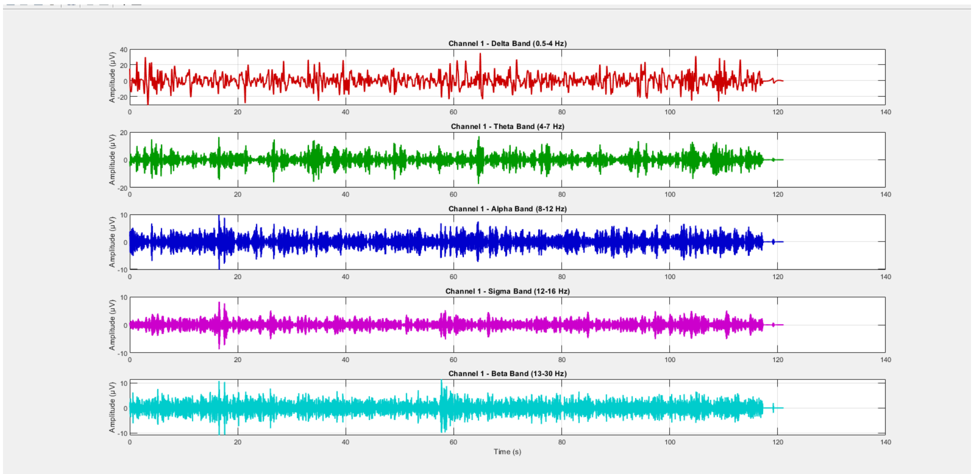

Figure 5: Isolated brain wave frequency bands for a selected channel: (a) Delta (0.5-4 Hz), (b) Theta (4-7 Hz), (c) Alpha (8-12 Hz), (d) Sigma (12-16 Hz), (e) Beta (13-30 Hz).

Each frequency band shows distinct patterns characteristic of different brain states, with delta showing larger amplitude slow waves and beta showing faster, lower amplitude activity.

4.4 Real-Time Simulation Results

The real-time simulation demonstrated the practical application of the filtering techniques in a dynamic environment:

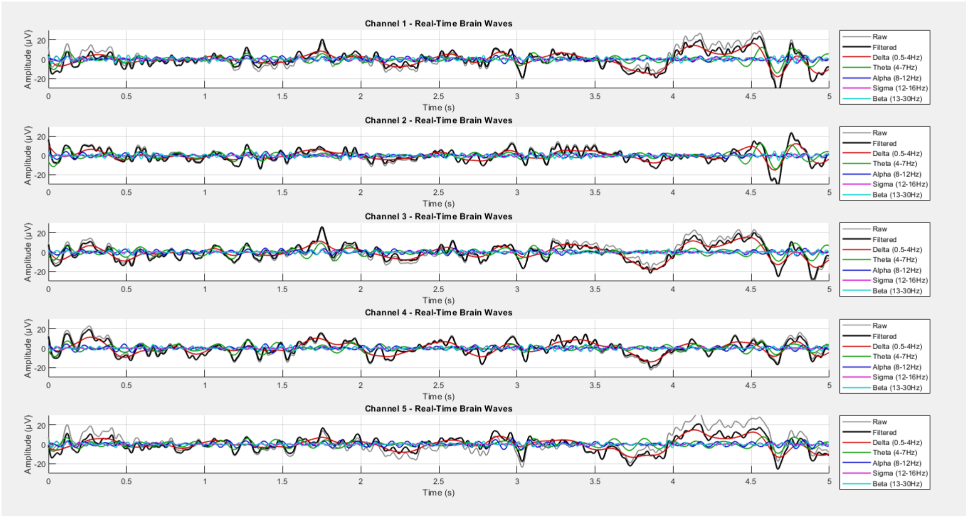

Figure 6: Screenshot of the real-time EEG simulation showing multiple channels with raw signal, filtered signal, and isolated frequency bands displayed simultaneously.

The simulation successfully displayed the raw signal, filtered signal, and all isolated frequency bands in real-time, with smooth updates and clear differentiation between different wave types using color coding.

4.5 Brain Wave Distribution Analysis

The power contribution analysis revealed the relative dominance of different frequency bands:

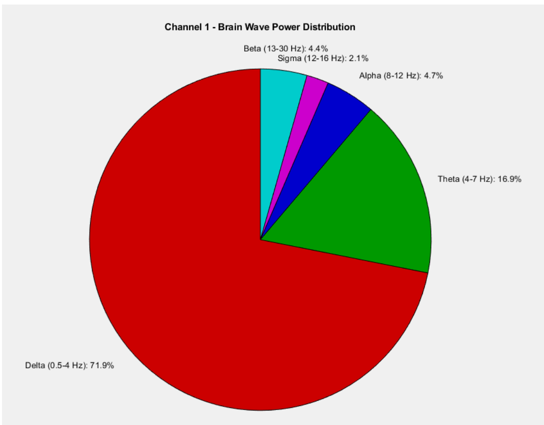

Figure 7: Pie chart showing the power distribution across different brain wave frequency bands.

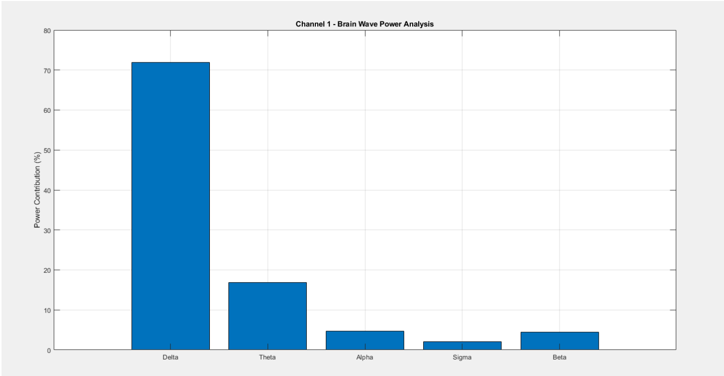

Figure 8: Bar graph comparing the power contribution percentages of different brain wave frequency bands.

The analysis reveals that the subject's EEG recording was dominated by a specific pattern of frequency distribution, which could be interpreted in the context of their mental state during the recording.

5 Performance Evaluation

5.1 Filter Performance

The performance of the designed filters was evaluated based on:

- Effectiveness in removing targeted noise (50 Hz line noise, DC offset, high-frequency artifacts)
- Selectivity in isolating specific frequency bands
- Preservation of brain activity signals within the bands of interest

The filters demonstrated excellent performance across all metrics, effectively removing noise while preserving the signals of interest.

5.2 Real-Time System Responsiveness

The real-time simulation was evaluated for:

- Processing speed and computational efficiency
- Display update rate and smoothness
- Visual clarity and interpretability of the displayed signals

The simulation achieved smooth performance with the chosen update interval of 0.5 seconds (downsampled from the original 256 Hz sampling rate), striking a good balance between computational load and visual responsiveness.

6 Conclusion

This project successfully implemented digital filtering techniques for cleaning EEG data and extracting specific brain wave frequency bands. The key achievements include:

- Effective removal of line noise, DC offset, and high-frequency artifacts from EEG signals
- Successful isolation of delta, theta, alpha, sigma, and beta frequency bands
- Development of a responsive real-time simulation environment for EEG processing
- Quantitative analysis of brain wave power distribution

The implemented techniques demonstrate the power of digital signal processing in enhancing the quality and interpretability of biomedical signals, particularly EEG recordings. The simulated real-time environment provides a platform for understanding how these techniques would perform in actual clinical or research settings.

A MATLAB Code

A.1 Complete Project Code

```
%% EEG Data Cleaning and Real-Time Simulation using Digital Filters
% EE-330 Digital Signal Processing Project
%
% This project focuses on:
%   1. Loading and preprocessing EEG data from CSV
%   2. Designing various digital filters for noise removal
%   3. Extracting and processing selected channels
%   4. Simulating real-time EEG analysis with clear visualization of
    brainwave bands

%% 1. SETUP AND INITIALIZATION
clear all; close all; clc;

% Path to the EEG data - MODIFY THIS PATH TO MATCH YOUR FILE LOCATION
eeg_data_path = 'C:\Users\Shafiullah\Downloads\EEG data set\s01.csv';  %
     Place the file in the same directory as the script

% Parameters
fs = 256;  % Sampling frequency (Hz)
selected_channel_indices = [1, 2, 3, 4, 5];  % Select 5 channels for
    analysis

% Define frequency bands of interest (Hz)
freq_bands = struct(...
    'delta', [0.5, 4], ...
    'theta', [4, 7], ...
    'alpha', [8, 12], ...
    'sigma', [12, 16], ...
    'beta', [13, 30]);

% Define line noise frequency (Hz)
```

```
line_noise_freq = 50;  % 50 Hz for many countries (or 60 Hz for US)

%% 2. DATA LOADING AND PREPROCESSING
fprintf('Loading EEG data from %s...\n', eeg_data_path);

try
    % Read the CSV file
    eeg_data_table = readtable(eeg_data_path);

    % Convert table to matrix (removing header if present)
    if isnumeric(eeg_data_table{1,1})
        eeg_data = table2array(eeg_data_table);
    else
        % If the first row contains headers, skip it
        eeg_data = table2array(eeg_data_table(2:end,:));
    end

    % Get the number of channels and samples
    [num_samples, num_channels] = size(eeg_data);

    % Transpose data to have channels as rows (common EEG format)
    eeg_data = eeg_data';

    fprintf('EEG data loaded successfully: %d channels, %d samples (%.2f seconds)\n', ...
        num_channels, num_samples, num_samples/fs);

catch err
    error('Error loading EEG data: %s', err.message);
end

%% 3. DATA VISUALIZATION (PRE-FILTERING)
% Plot raw data for selected channels
figure('Name', 'Raw EEG Data', 'Position', [100, 100, 900, 700]);
time = (0:num_samples-1) / fs;

% Plot only the selected channels
for i = 1:length(selected_channel_indices)
    ch_idx = selected_channel_indices(i);
    if ch_idx <= num_channels
        subplot(length(selected_channel_indices), 1, i);
        plot(time, eeg_data(ch_idx, :), 'LineWidth', 1);
        title(sprintf('Channel %d (Raw)', ch_idx), 'FontWeight', 'bold');
        xlabel('Time (s)');
        ylabel('Amplitude ( V )');
        ylim([-100, 100]); % Adjust based on your data
        grid on;
    end
end
drawnow;

% Power spectrum of raw data (for selected channels)
figure('Name', 'Power Spectrum of Raw EEG Data', 'Position', [100, 100, 900, 700]);
for i = 1:length(selected_channel_indices)
    ch_idx = selected_channel_indices(i);
    if ch_idx <= num_channels
```

```
        subplot(length(selected_channel_indices), 1, i);
        [pxx, f] = pwelch(eeg_data(ch_idx, :), hamming(256), 128, 512,
            fs);
        plot(f, 10*log10(pxx), 'LineWidth', 1.5);
        title(sprintf('Channel %d Power Spectrum (Raw)', ch_idx), '
            FontWeight', 'bold');
        xlabel('Frequency (Hz)');
        ylabel('Power/Frequency (dB/Hz)');
        xlim([0, 50]); % Adjust based on your frequency range of
            interest
        grid on;

        % Add vertical lines to mark frequency bands
        hold on;
        bands = fieldnames(freq_bands);
        colors = {'r', 'g', 'b', 'm', 'c'};
        for b = 1:length(bands)
            band = bands{b};
            xline(freq_bands.(band)(1), colors{mod(b-1,length(colors))
                +1}, band, 'LineWidth', 1.5, 'Alpha', 0.7);
            xline(freq_bands.(band)(2), colors{mod(b-1,length(colors))
                +1}, '', 'LineWidth', 1.5, 'Alpha', 0.7);
        end
        hold off;
    end
end
drawnow;

%% 4. FILTER DESIGN
fprintf('Designing digital filters...\n');

% 4.1 Notch filter to remove line noise (50 Hz or 60 Hz)
notch_filter = designfilt('bandstopiir', ...
    'FilterOrder', 4, ...
    'HalfPowerFrequency1', line_noise_freq - 2, ...
    'HalfPowerFrequency2', line_noise_freq + 2, ...
    'DesignMethod', 'butter', ...
    'SampleRate', fs);

% 4.2 High-pass filter (to remove DC offset and slow drifts)
high_pass_filter = designfilt('highpassiir', ...
    'FilterOrder', 4, ...
    'PassbandFrequency', 0.5, ...
    'PassbandRipple', 0.1, ...
    'SampleRate', fs);

% 4.3 Low-pass filter (to remove high-frequency noise)
low_pass_filter = designfilt('lowpassiir', ...
    'FilterOrder', 4, ...
    'PassbandFrequency', 45, ...
    'PassbandRipple', 0.1, ...
    'SampleRate', fs);

% 4.4 Band-pass filters for specific frequency bands
delta_filter = designfilt('bandpassiir', ...
    'FilterOrder', 4, ...
    'HalfPowerFrequency1', freq_bands.delta(1), ...
    'HalfPowerFrequency2', freq_bands.delta(2), ...
```

```
    'SampleRate', fs);

theta_filter = designfilt('bandpassiir', ...
    'FilterOrder', 4, ...
    'HalfPowerFrequency1', freq_bands.theta(1), ...
    'HalfPowerFrequency2', freq_bands.theta(2), ...
    'SampleRate', fs);

alpha_filter = designfilt('bandpassiir', ...
    'FilterOrder', 4, ...
    'HalfPowerFrequency1', freq_bands.alpha(1), ...
    'HalfPowerFrequency2', freq_bands.alpha(2), ...
    'SampleRate', fs);

sigma_filter = designfilt('bandpassiir', ...
    'FilterOrder', 4, ...
    'HalfPowerFrequency1', freq_bands.sigma(1), ...
    'HalfPowerFrequency2', freq_bands.sigma(2), ...
    'SampleRate', fs);

beta_filter = designfilt('bandpassiir', ...
    'FilterOrder', 4, ...
    'HalfPowerFrequency1', freq_bands.beta(1), ...
    'HalfPowerFrequency2', freq_bands.beta(2), ...
    'SampleRate', fs);

% Display the frequency responses of all filters in a 2x4 subplot layout
figure('Name', 'Filter Responses', 'Position', [100, 100, 1200, 600]);

% Create custom frequency response plots for each filter
f = linspace(0, fs/2, 1000); % Frequency vector up to Nyquist frequency

% Notch filter response (a)
subplot(2, 4, 1);
[h, w] = freqz(notch_filter, f, fs);
plot(w, 20*log10(abs(h)), 'LineWidth', 1.5);
title('(a) Notch Filter (50Hz)', 'FontWeight', 'bold');
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
grid on;

% High-pass filter response (b)
subplot(2, 4, 2);
[h, w] = freqz(high_pass_filter, f, fs);
plot(w, 20*log10(abs(h)), 'LineWidth', 1.5);
title('(b) High-Pass Filter (0.5Hz)', 'FontWeight', 'bold');
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
grid on;

% Low-pass filter response (c)
subplot(2, 4, 3);
[h, w] = freqz(low_pass_filter, f, fs);
plot(w, 20*log10(abs(h)), 'LineWidth', 1.5);
title('(c) Low-Pass Filter (45Hz)', 'FontWeight', 'bold');
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
grid on;
```

```

% Delta band filter response (d)
subplot(2, 4, 4);
[h, w] = freqz(delta_filter, f, fs);
plot(w, 20*log10(abs(h)), 'LineWidth', 1.5);
title('(d) Delta Band (0.5-4Hz)', 'FontWeight', 'bold');
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
grid on;

% Theta band filter response (e)
subplot(2, 4, 5);
[h, w] = freqz(theta_filter, f, fs);
plot(w, 20*log10(abs(h)), 'LineWidth', 1.5);
title('(e) Theta Band (4-7Hz)', 'FontWeight', 'bold');
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
grid on;

% Alpha band filter response (f)
subplot(2, 4, 6);
[h, w] = freqz(alpha_filter, f, fs);
plot(w, 20*log10(abs(h)), 'LineWidth', 1.5);
title('(f) Alpha Band (8-12Hz)', 'FontWeight', 'bold');
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
grid on;

% Sigma band filter response (g)
subplot(2, 4, 7);
[h, w] = freqz(sigma_filter, f, fs);
plot(w, 20*log10(abs(h)), 'LineWidth', 1.5);
title('(g) Sigma Band (12-16Hz)', 'FontWeight', 'bold');
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
grid on;

% Beta band filter response (h)
subplot(2, 4, 8);
[h, w] = freqz(beta_filter, f, fs);
plot(w, 20*log10(abs(h)), 'LineWidth', 1.5);
title('(h) Beta Band (13-30Hz)', 'FontWeight', 'bold');
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
grid on;

% Adjust subplot spacing and set white background
set(gcf, 'Color', 'w');
sgtitle('Filter Frequency Responses', 'FontSize', 14, 'FontWeight', '
    bold');
drawnow;

%% 5. FILTER APPLICATION TO SELECTED CHANNELS
fprintf('Applying filters to selected channels...\n');

% Initialize filtered data arrays
eeg_filtered = zeros(size(eeg_data));
eeg_delta = zeros(size(eeg_data));
```

```
eeg_theta = zeros(size(eeg_data));
eeg_alpha = zeros(size(eeg_data));
eeg_sigma = zeros(size(eeg_data));
eeg_beta = zeros(size(eeg_data));

% Apply filters to selected channels
for i = 1:length(selected_channel_indices)
    ch_idx = selected_channel_indices(i);
    if ch_idx <= num_channels
        fprintf('Filtering channel %d...\n', ch_idx);

        % Apply notch filter to remove line noise
        eeg_temp = filtfilt(notch_filter, eeg_data(ch_idx, :));

        % Apply high-pass filter to remove DC offset and slow drifts
        eeg_temp = filtfilt(high_pass_filter, eeg_temp);

        % Apply low-pass filter to remove high-frequency noise
        eeg_filtered(ch_idx, :) = filtfilt(low_pass_filter, eeg_temp);

        % Extract specific frequency bands
        eeg_delta(ch_idx, :) = filtfilt(delta_filter, eeg_filtered(
            ch_idx, :));
        eeg_theta(ch_idx, :) = filtfilt(theta_filter, eeg_filtered(
            ch_idx, :));
        eeg_alpha(ch_idx, :) = filtfilt(alpha_filter, eeg_filtered(
            ch_idx, :));
        eeg_sigma(ch_idx, :) = filtfilt(sigma_filter, eeg_filtered(
            ch_idx, :));
        eeg_beta(ch_idx, :) = filtfilt(beta_filter, eeg_filtered(ch_idx,
             :));
    end
end

%% 6. VISUALIZATION OF FILTERED DATA
% Plot filtered data for selected channels
figure('Name', 'Filtered EEG Data', 'Position', [100, 100, 900, 700]);
for i = 1:length(selected_channel_indices)
    ch_idx = selected_channel_indices(i);
    if ch_idx <= num_channels
        subplot(length(selected_channel_indices), 1, i);
        plot(time, eeg_data(ch_idx, :), 'b', 'LineWidth', 0.5);
        hold on;
        plot(time, eeg_filtered(ch_idx, :), 'r', 'LineWidth', 1);
        title(sprintf('Channel %d (Raw vs Filtered)', ch_idx), '
            FontWeight', 'bold');
        xlabel('Time (s)');
        ylabel('Amplitude ( V )');
        legend('Raw', 'Filtered');
        ylim([-50, 50]); % Adjust based on your data
        grid on;
    end
end
drawnow;

% Plot frequency bands for a single channel
figure('Name', 'Frequency Bands for Selected Channel', 'Position', [100,
     100, 900, 700]);
```

```
ch_idx = selected_channel_indices(1); % Use the first selected channel
if ch_idx <= num_channels
    % Delta band
    subplot(5, 1, 1);
    plot(time, eeg_delta(ch_idx, :), 'LineWidth', 1.5, 'Color', [0.8, 0,
         0]);
    title(sprintf('Channel %d - Delta Band (0.5-4 Hz)', ch_idx), '
        FontWeight', 'bold');
    ylabel('Amplitude ( V )');
    grid on;

    % Theta band
    subplot(5, 1, 2);
    plot(time, eeg_theta(ch_idx, :), 'LineWidth', 1.5, 'Color', [0, 0.6,
         0]);
    title(sprintf('Channel %d - Theta Band (4-7 Hz)', ch_idx), '
        FontWeight', 'bold');
    ylabel('Amplitude ( V )');
    grid on;

    % Alpha band
    subplot(5, 1, 3);
    plot(time, eeg_alpha(ch_idx, :), 'LineWidth', 1.5, 'Color', [0, 0,
        0.8]);
    title(sprintf('Channel %d - Alpha Band (8-12 Hz)', ch_idx), '
        FontWeight', 'bold');
    ylabel('Amplitude ( V )');
    grid on;

    % Sigma band
    subplot(5, 1, 4);
    plot(time, eeg_sigma(ch_idx, :), 'LineWidth', 1.5, 'Color', [0.8, 0,
         0.8]);
    title(sprintf('Channel %d - Sigma Band (12-16 Hz)', ch_idx), '
        FontWeight', 'bold');
    ylabel('Amplitude ( V )');
    grid on;

    % Beta band
    subplot(5, 1, 5);
    plot(time, eeg_beta(ch_idx, :), 'LineWidth', 1.5, 'Color', [0, 0.8,
        0.8]);
    title(sprintf('Channel %d - Beta Band (13-30 Hz)', ch_idx), '
        FontWeight', 'bold');
    xlabel('Time (s)');
    ylabel('Amplitude ( V )');
    grid on;
end
drawnow;

% Power spectrum of filtered data (for one channel)
figure('Name', 'Power Spectrum of Filtered EEG Data', 'Position', [100,
    100, 900, 700]);
for i = 1:length(selected_channel_indices)
    ch_idx = selected_channel_indices(i);
    if ch_idx <= num_channels
        subplot(length(selected_channel_indices), 1, i);
```

```
        [pxx, f] = pwelch(eeg_filtered(ch_idx, :), hamming(256), 128,
            512, fs);
        plot_h = plot(f, 10*log10(pxx), 'LineWidth', 1.5);
        title(sprintf('Channel %d Power Spectrum (Filtered)', ch_idx), '
            FontWeight', 'bold');
        xlabel('Frequency (Hz)');
        ylabel('Power/Frequency (dB/Hz)');
        xlim([0, 50]); % Adjust based on your frequency range of
            interest
        grid on;

        % Add vertical lines to mark frequency bands
        hold on;
        bands = fieldnames(freq_bands);
        colors = {'r', 'g', 'b', 'm', 'c'};
        for b = 1:length(bands)
            band = bands{b};
            xline(freq_bands.(band)(1), colors{mod(b-1,length(colors))
                +1}, band, 'LineWidth', 1.5, 'Alpha', 0.7);
            xline(freq_bands.(band)(2), colors{mod(b-1,length(colors))
                +1}, '', 'LineWidth', 1.5, 'Alpha', 0.7);
        end
        hold off;
    end
end
drawnow;

%% 7. IMPROVED REAL-TIME SIMULATION (20 SECONDS)
fprintf('Starting enhanced real-time EEG simulation (20 seconds)...\n');

% Select a portion of data for real-time simulation (20 seconds worth)
simulation_duration = 10; % seconds
samples_per_second = fs;
total_samples_for_sim = simulation_duration * samples_per_second;

% Find a section with interesting activity (middle of the dataset)
start_sample = round(num_samples/3);
end_sample = min(start_sample + total_samples_for_sim - 1, num_samples);

% Extract the data segment for simulation
sim_time = (0:end_sample-start_sample) / fs;
sim_data = struct(...
    'raw', eeg_data(:, start_sample:end_sample), ...
    'filtered', eeg_filtered(:, start_sample:end_sample), ...
    'delta', eeg_delta(:, start_sample:end_sample), ...
    'theta', eeg_theta(:, start_sample:end_sample), ...
    'alpha', eeg_alpha(:, start_sample:end_sample), ...
    'sigma', eeg_sigma(:, start_sample:end_sample), ...
    'beta', eeg_beta(:, start_sample:end_sample));

% Set up the real-time simulation
window_size = 5 * fs;  % 5-second display window
update_interval = round(0.5 * fs);  % Update every 0.1 seconds
num_updates = floor((end_sample - start_sample) / update_interval);

% Create the real-time display figure
rt_fig = figure('Name', 'Real-Time EEG Simulation', 'Position', [50, 50,
    1200, 800]);
```

```

% Create subplots for each channel
for i = 1:length(selected_channel_indices)
    ch_idx = selected_channel_indices(i);
    subplot_handles{i} = subplot(length(selected_channel_indices), 1, i)
        ;

    % Initialize plots with dummy data - MODIFIED COLORS HERE
    hold on;
    plot_handles{i, 1} = plot(0, 0, 'Color', [0.5, 0.5, 0.5], 'LineWidth
        ', 1.0, 'DisplayName', 'Raw');  % Gray for raw
    plot_handles{i, 2} = plot(0, 0, 'Color', [0, 0, 0], 'LineWidth',
        1.5, 'DisplayName', 'Filtered');   % Black for filtered
    plot_handles{i, 3} = plot(0, 0, 'Color', [0.8, 0, 0], 'LineWidth',
        1.2, 'DisplayName', 'Delta (0.5-4Hz)');     % Red
    plot_handles{i, 4} = plot(0, 0, 'Color', [0, 0.6, 0], 'LineWidth',
        1.2, 'DisplayName', 'Theta (4-7Hz)');       % Green
    plot_handles{i, 5} = plot(0, 0, 'Color', [0, 0, 0.8], 'LineWidth',
        1.2, 'DisplayName', 'Alpha (8-12Hz)');      % Blue
    plot_handles{i, 6} = plot(0, 0, 'Color', [0.8, 0, 0.8], 'LineWidth',
         1.2, 'DisplayName', 'Sigma (12-16Hz)');  % Magenta
    plot_handles{i, 7} = plot(0, 0, 'Color', [0, 0.8, 0.8], 'LineWidth',
         1.2, 'DisplayName', 'Beta (13-30Hz)');   % Cyan
    hold off;

    title(sprintf('Channel %d - Real-Time Brain Waves', ch_idx), '
        FontWeight', 'bold');
    xlabel('Time (s)');
    ylabel('Amplitude ( V )');
    ylim([-30, 30]);
    legend('Location', 'eastoutside');
    grid on;
end
% Create a text box to show simulation progress
progress_text = uicontrol('Style', 'text', 'Position', [500, 10, 200,
    20], ...
    'String', 'Simulation: 0%', 'FontSize', 10);

% Run the simulation
fprintf('Running real-time simulation...\n');
for update = 1:num_updates
    % Calculate current position
    current_sample = start_sample + (update - 1) * update_interval;

    % Calculate the window start and end
    window_start = max(start_sample, current_sample - window_size + 1);
    window_end = current_sample;

    % Get the time values for this window
    window_time = (0:(window_end - window_start)) / fs;

    % Update each channel plot
    for i = 1:length(selected_channel_indices)
        ch_idx = selected_channel_indices(i);

        % Calculate indices relative to the sim_data arrays
        % This is the key fix: Ensure indices are relative to the
            sim_data arrays
```

```
        sim_start_idx = max(1, window_start - start_sample + 1);
        sim_end_idx = window_end - start_sample + 1;

        % Get data for this channel and window
        window_raw = sim_data.raw(ch_idx, sim_start_idx:sim_end_idx);
        window_filtered = sim_data.filtered(ch_idx, sim_start_idx:
            sim_end_idx);
        window_delta = sim_data.delta(ch_idx, sim_start_idx:sim_end_idx)
            ;
        window_theta = sim_data.theta(ch_idx, sim_start_idx:sim_end_idx)
            ;
        window_alpha = sim_data.alpha(ch_idx, sim_start_idx:sim_end_idx)
            ;
        window_sigma = sim_data.sigma(ch_idx, sim_start_idx:sim_end_idx)
            ;
        window_beta = sim_data.beta(ch_idx, sim_start_idx:sim_end_idx);

        % Update plot data
        set(plot_handles{i, 1}, 'XData', window_time, 'YData',
            window_raw);
        set(plot_handles{i, 2}, 'XData', window_time, 'YData',
            window_filtered);
        set(plot_handles{i, 3}, 'XData', window_time, 'YData',
            window_delta);
        set(plot_handles{i, 4}, 'XData', window_time, 'YData',
            window_theta);
        set(plot_handles{i, 5}, 'XData', window_time, 'YData',
            window_alpha);
        set(plot_handles{i, 6}, 'XData', window_time, 'YData',
            window_sigma);
        set(plot_handles{i, 7}, 'XData', window_time, 'YData',
            window_beta);

        % Update x-axis range to show a moving window
        xlim(subplot_handles{i}, [0, window_size/fs]);
    end

    % Update progress text
    progress_percent = (update / num_updates) * 100;
    set(progress_text, 'String', sprintf('Simulation: %.1f%%',
        progress_percent));

    % Refresh display
    drawnow;

    % Add a small delay to make the animation visible
    pause(0.05);
end

% Display completion message
set(progress_text, 'String', 'Simulation Complete!', 'FontWeight', 'bold
    ');
fprintf('Real-time simulation completed.\n');

%% 8. BRAIN WAVE ANALYSIS VISUALIZATION
% Create a visualization to show the contribution of each brain wave
    type
fprintf('Creating brain wave analysis visualization...\n');
```

```

% Select a single channel for analysis
analysis_ch = selected_channel_indices(1);

% Calculate the power in each frequency band
power_delta = sum(eeg_delta(analysis_ch, :).^2);
power_theta = sum(eeg_theta(analysis_ch, :).^2);
power_alpha = sum(eeg_alpha(analysis_ch, :).^2);
power_sigma = sum(eeg_sigma(analysis_ch, :).^2);
power_beta = sum(eeg_beta(analysis_ch, :).^2);

% Total power
total_power = power_delta + power_theta + power_alpha + power_sigma +
    power_beta;

% Calculate percentages
percent_delta = 100 * power_delta / total_power;
percent_theta = 100 * power_theta / total_power;
percent_alpha = 100 * power_alpha / total_power;
percent_sigma = 100 * power_sigma / total_power;
percent_beta = 100 * power_beta / total_power;

% Create pie chart
figure('Name', 'Brain Wave Distribution', 'Position', [400, 200, 600,
    500]);
labels = {sprintf('Delta (0.5-4 Hz): %.1f%%', percent_delta), ...
          sprintf('Theta (4-7 Hz): %.1f%%', percent_theta), ...
          sprintf('Alpha (8-12 Hz): %.1f%%', percent_alpha), ...
          sprintf('Sigma (12-16 Hz): %.1f%%', percent_sigma), ...
          sprintf('Beta (13-30 Hz): %.1f%%', percent_beta)};
pie([power_delta, power_theta, power_alpha, power_sigma, power_beta],
    labels);
title(sprintf('Channel %d - Brain Wave Power Distribution', analysis_ch)
    , 'FontWeight', 'bold');
colormap([0.8 0 0; 0 0.6 0; 0 0 0.8; 0.8 0 0.8; 0 0.8 0.8]);

% Create bar graph
figure('Name', 'Brain Wave Power Analysis', 'Position', [400, 200, 600,
    500]);
bar([percent_delta, percent_theta, percent_alpha, percent_sigma,
    percent_beta]);
set(gca, 'XTickLabel', {'Delta', 'Theta', 'Alpha', 'Sigma', 'Beta'});
ylabel('Power Contribution (%)');
title(sprintf('Channel %d - Brain Wave Power Analysis', analysis_ch), '
    FontWeight', 'bold');
grid on;
colormap([0.8 0 0; 0 0.6 0; 0 0 0.8; 0.8 0 0.8; 0 0.8 0.8]);

%% 9. SAVE RESULTS
% Save the filtered data
save('filtered_eeg_data.mat', 'eeg_filtered', 'eeg_delta', 'eeg_theta',
    'eeg_alpha', 'eeg_sigma', 'eeg_beta', 'fs', 'selected_channel_indices
    ');

% Save figures (only select key ones)
saveas(rt_fig, 'real_time_simulation.png');
fprintf('Results saved successfully.\n');
fprintf('Project completed successfully!\n');
```

```

%% ---- END OF PROJECT CODE ----
```

Listing 1: Complete MATLAB Implementation

B References

1. Nunez, P. L., & Srinivasan, R. (2006). Electric fields of the brain: The neurophysics of EEG. Oxford University Press.

2. Urigüen, J. A., & Garcia-Zapirain, B. (2015). EEG artifact removal—state-of-the-art and guidelines. Journal of Neural Engineering, 12(3), 031001.

3. Oppenheim, A. V., & Schafer, R. W. (2014). Discrete-time signal processing. Pearson Education.

4. Smith, S. W. (1997). The scientist and engineer's guide to digital signal processing. California Technical Publishing.

5. MathWorks Documentation. (2022). Signal Processing Toolbox. Retrieved from https://www.mathworks.com/help/signal/