Faculty of Engineering & Technology Electrical & Computer Engineering Department

DIGITAL SIGNAL PROCESSING (DSP)

-ENCS4310-
Course Project

Filtering a real electrocardiographic signal

REPORT

Prepared By:

Mohamad Abu Saleh	1203331
Abdulhameed Awad	1212478

Instructor: Dr. Qadri Mayyala

Section: 2 **Date:** 22/May/2024

Abstract

This project focuses on enhancing the readability and facilitating the automatic processing of real electrocardiographic (ECG) signals through digital signal processing techniques. The ECG data, sampled at 360 Hz, is provided in two files, ECG1.xlsx and ECG2.xlsx, which undergo low-pass and high-pass filtering. Initially, the project involves visualizing the ECG data by inserting real-time columns, optimizing signal display, and determining the necessary filtering types.

Subsequently, the project applies high-pass and low-pass filters to the ECG signals, analyzing their transfer functions, frequency responses, and classifying them as FIR or IIR filters. The filters are applied individually and in combination to examine their effects on the signals. An optional task includes using adaptive filters for noise removal. The final report includes the methodology, analysis, results, and conclusions, with code appendices provided. The project uses MATLAB or Python, with an emphasis on clear and organized presentation of results.

Table of Content

Table of Figures

Theory

The project revolves around digital signal processing (DSP) techniques, particularly focusing on filtering electrocardiographic (ECG) signals. ECG signals, which represent the electrical activity of the heart, are inherently prone to noise and artifacts. Filtering these signals is crucial for accurate analysis and interpretation, especially for tasks such as peak detection and classification.

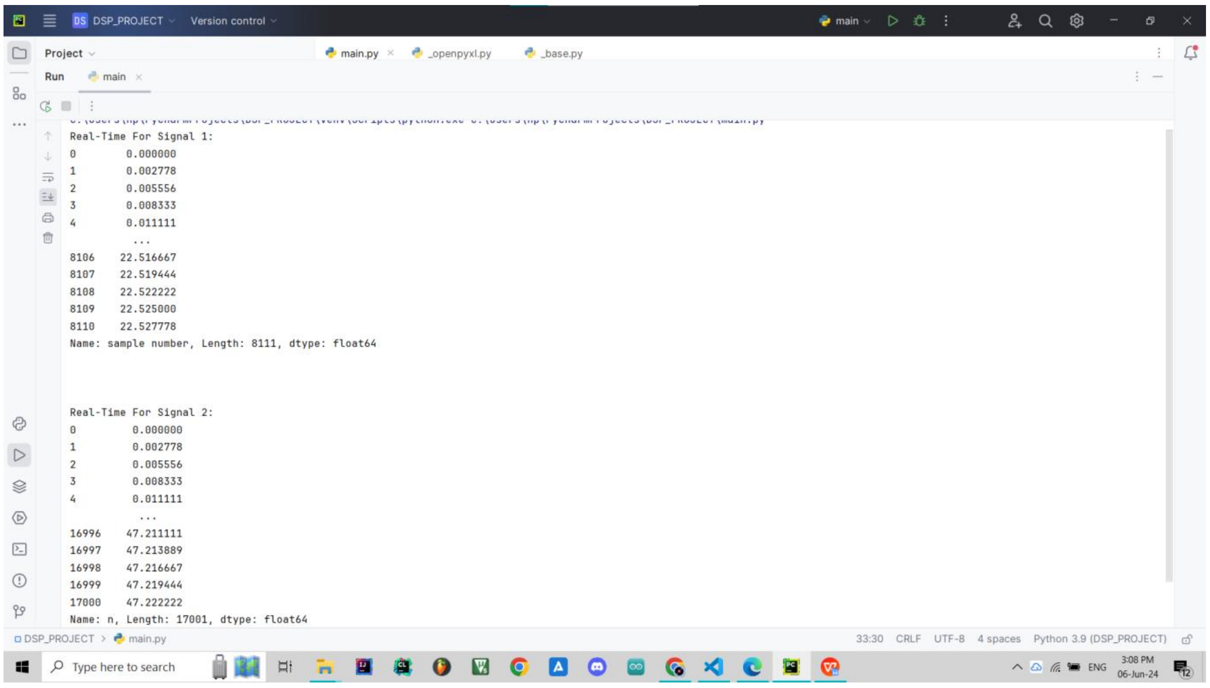
Procedure

1. Data Visualization:

-These data are recordings of actual cardiac electrical signals. They are sampled at 360 Hz.

1.1 Insert the real-time column for each signal:

We can find the real time signal by multiply the sample number by the frequency (360 Hz) and we can implement this using python code



```
Real-Time For Signal 1:
0        0.000000
1        0.002778
2        0.005556
3        0.008333
4        0.011111
           ...
8106    22.516667
8107    22.519444
8108    22.522222
8109    22.525000
8110    22.527778
Name: sample number, Length: 8111, dtype: float64



Real-Time For Signal 2:
0         0.000000
1         0.002778
2         0.005556
3         0.008333
4         0.011111
            ...
16996    47.211111
16997    47.213889
16998    47.216667
16999    47.219444
17000    47.222222
Name: n, Length: 17001, dtype: float64
```

Figure 1

1.2 Display each of these signals (reduce the width of the display lines for better readability)

In this part of the project, we visualize the ECG data by plotting ECG Signal 1 and ECG Signal 2 with reduced line widths for better readability. These visualizations help us observe the signal characteristics and identify the necessary preprocessing steps, such as applying low-pass and high-pass filters to enhance signal quality and facilitate automatic processing.

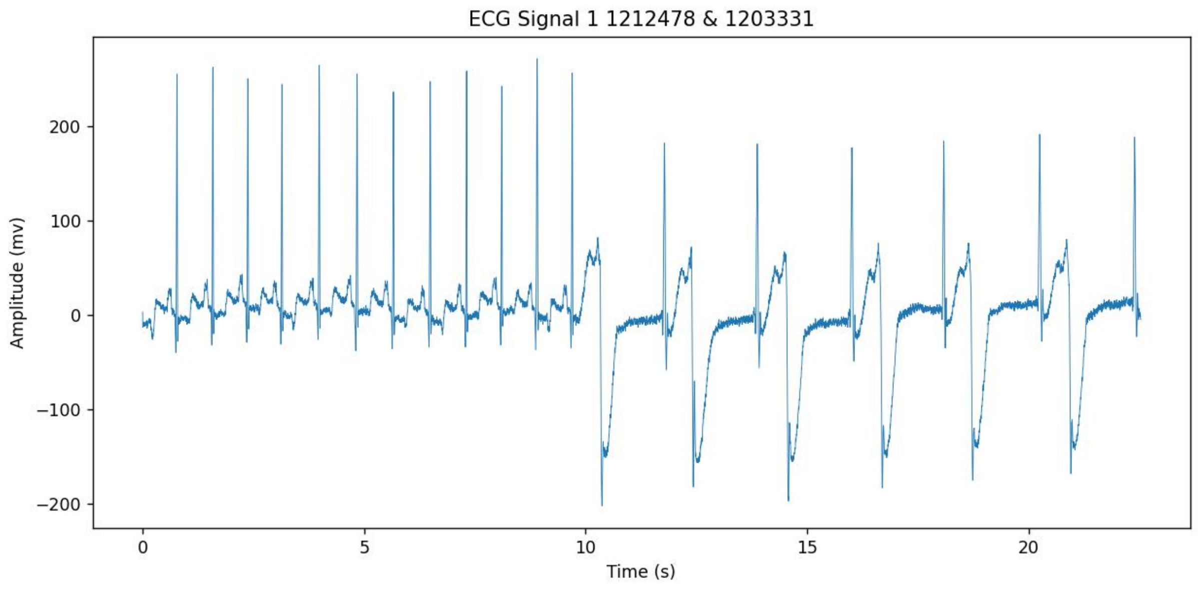


Figure 2

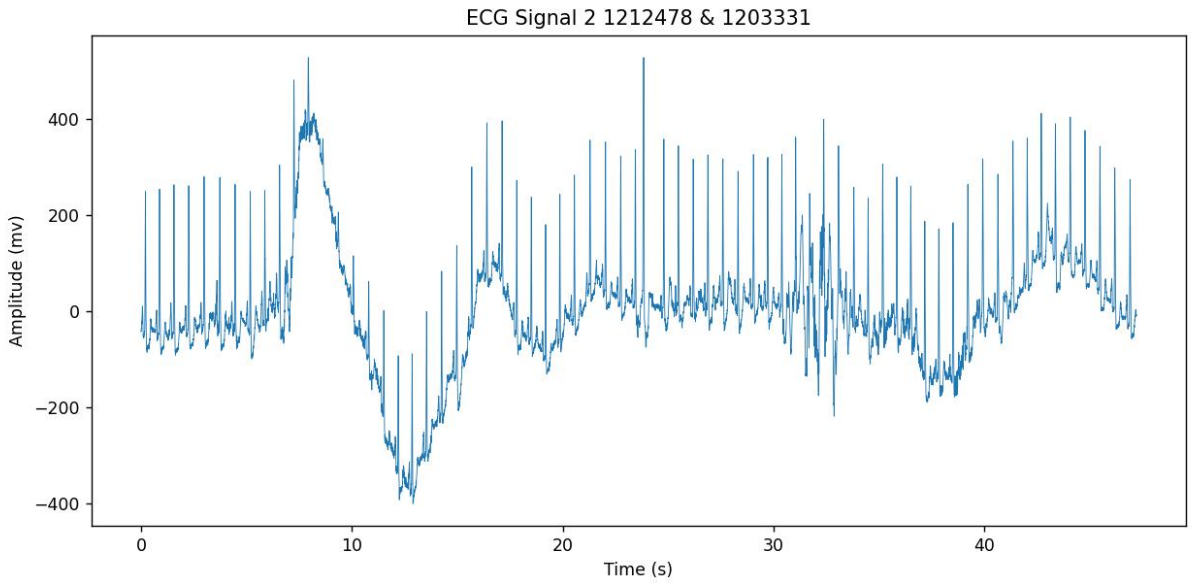


Figure 3

1.3 What filtering types are necessary to improve the readability of data and make automatic processing possible?

To improve the readability and enable automatic processing of ECG signals, various filtering techniques can be applied:

Baseline Wander Removal:

1. **Problem**: Low-frequency baseline wander caused by respiration, body movement, or electrode impedance changes.
2. **Solution**: High-pass filters to remove low-frequency noise.

Powerline Interference Removal:

1. **Problem**: Electrical interference from power lines (e.g., 50/60 Hz).
2. **Solution**: Notch filters or band-stop filters to eliminate powerline interference.

Muscle Artifact Removal:

1. **Problem**: High-frequency noise from muscle activity or movement.
2. **Solution**: High-pass filters or adaptive filters to reduce muscle artifacts.

Noise Reduction:

1. **Problem**: Various noise sources, such as electrode motion artifacts, electrode contact noise, and electronic noise.
2. **Solution**: Adaptive filters or wavelet denoising to reduce noise while preserving important ECG features.

Signal Enhancement:

1. **Problem**: Need for smoothing ECG signals and improving visibility of important features.
2. **Solution**: Suavity-Golan filters or moving average filters to enhance signal quality.

QRS Detection:

1. **Problem**: Accurate detection of QRS complexes for automated analysis.
2. **Solution**: Band-pass filters or matched filters to enhance QRS complexes.

Frequency Domain Analysis:

1. **Problem**: Understanding frequency components of ECG signals.
2. **Solution**: Fourier transform or wavelet transform for detailed frequency analysis.

By employing these filtering techniques, the readability of ECG data is significantly improved, facilitating automatic processing tasks such as QRS detection, arrhythmia classification, and heart rate variability analysis.

2. Filtering the ECG:

2.1 The high-pass filter:

$$HP(n) = HP(n-1) - \frac{1}{32}X(n) + X(n-16) - X(n-17) + \frac{1}{32}X(n-32)$$

High-pass filters allow signals with frequencies higher than a certain cutoff frequency to pass through while attenuating lower-frequency components. In ECG signal processing, high-pass filters are used to eliminate baseline wander, which is a low-frequency noise caused by patient movement, respiration, or electrode impedance changes. The transfer function of a high-pass filter describes its frequency response, indicating how different frequency components of the input signal are modified. High-pass filters can be either Finite Impulse Response (FIR) or Infinite Impulse Response (IIR) filters, with FIR filters generally providing linear phase response and IIR filters offering more efficient implementations.

The given filter equation is:

$$\mathrm{HP(n) = HP(n-1) - \frac{1}{32}X(n) + X(n-16) - X(n-17) + \frac{1}{32}X(n-32)}$$

To find the transfer function:

Step 1: Taking the Z-transform

$$\mathrm{H(z) = z^{-1}H(z) - \frac{1}{32}X(z) + z^{-16}X(z) - z^{-17}X(z) + \frac{1}{32}z^{-32}X(z)}$$

Step 2: Rearrange the equation

$$\mathrm{H(z)(1-z^{-1}) = (-\frac{1}{32} + z^{-16} - z^{-17} + \frac{1}{32}z^{-32})X(z)}$$

Step 3: Find the transfer function

$$\mathrm{\frac{H(z)}{X(z)} = \frac{-\frac{1}{32} + z^{-16} - z^{-17} + \frac{1}{32}z^{-32}}{1-z^{-1}}}$$

Step 4: Poles and Zeros Analysis

Zeros: The zeros of the transfer function are the values of z that make the numerator zero.

Poles: The poles of the transfer function are the values of z that make the denominator zero.

The numerator is: $\mathrm{-\frac{1}{32} + z^{-16} - z^{-17} + \frac{1}{32}z^{-32}}$

The denominator is: $\mathrm{1 - z^{-1}}$

The denominator has a pole at z=1, indicating an IIR filter because it has a pole inside the unit circle. However, the structure of the numerator suggests it may cancel out this pole.

Checking for FIR Behavior

If the numerator and denominator have a common factor that cancels the pole at z=1, the filter might exhibit FIR-like behavior. Specifically, if:

$$-1/32 \quad + \quad 1 \quad - \quad 1 \quad + \quad 1/32 \qquad = \qquad 0$$

Since this holds true, the numerator cancels the pole at z=1. Therefore, while the filter has feedback and initially looks like an IIR filter, the cancellation of the pole at z=1 makes it behave like an FIR filter.

Conclusion

The given filter equation has characteristics of both FIR and IIR filters:

It has a pole and zero at z=1, which cancel each other.

Despite the presence of feedback in the equation, the pole-zero cancellation at z=1 makes it behave like an FIR filter.

Hence, this filter can be considered an FIR filter due to the pole-zero cancellation at z=1.

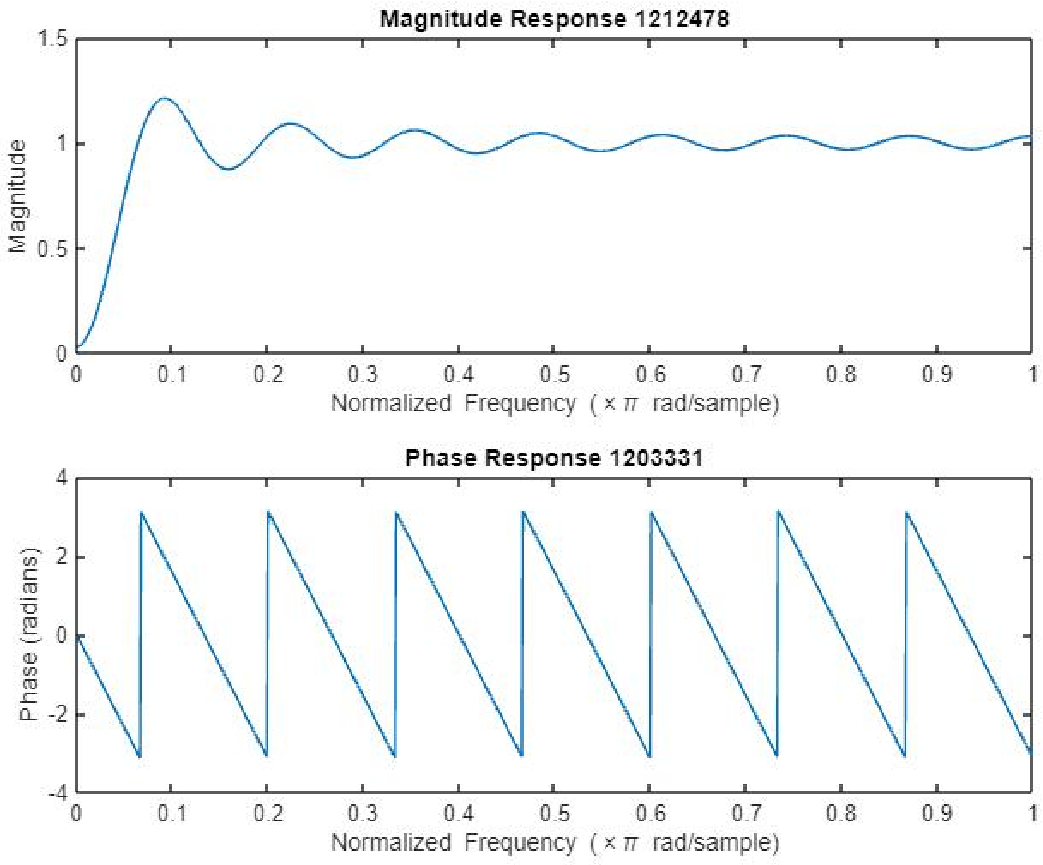


Figure 4

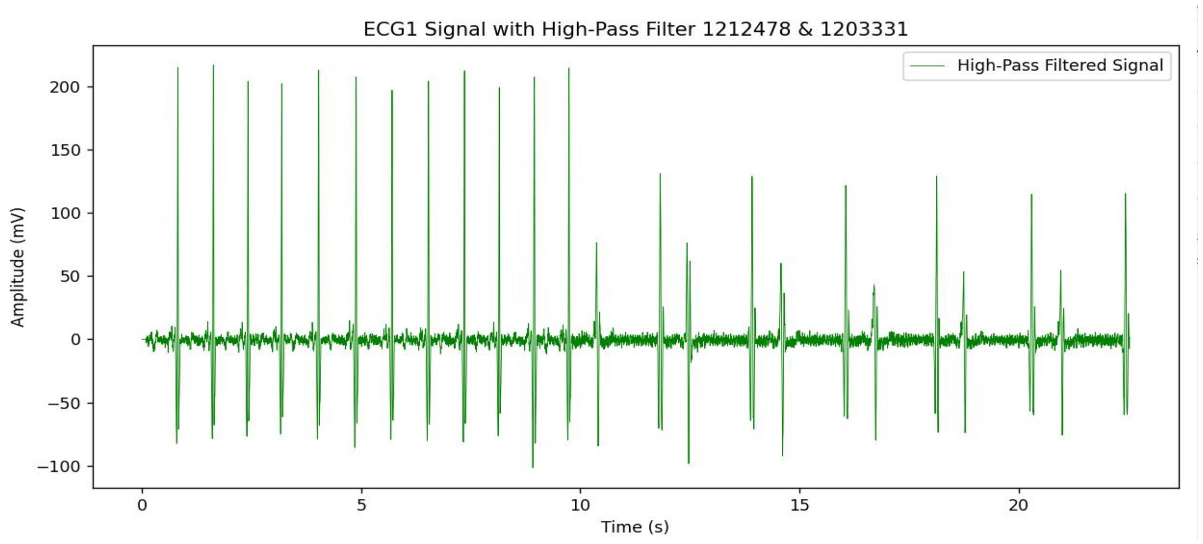


Figure 5

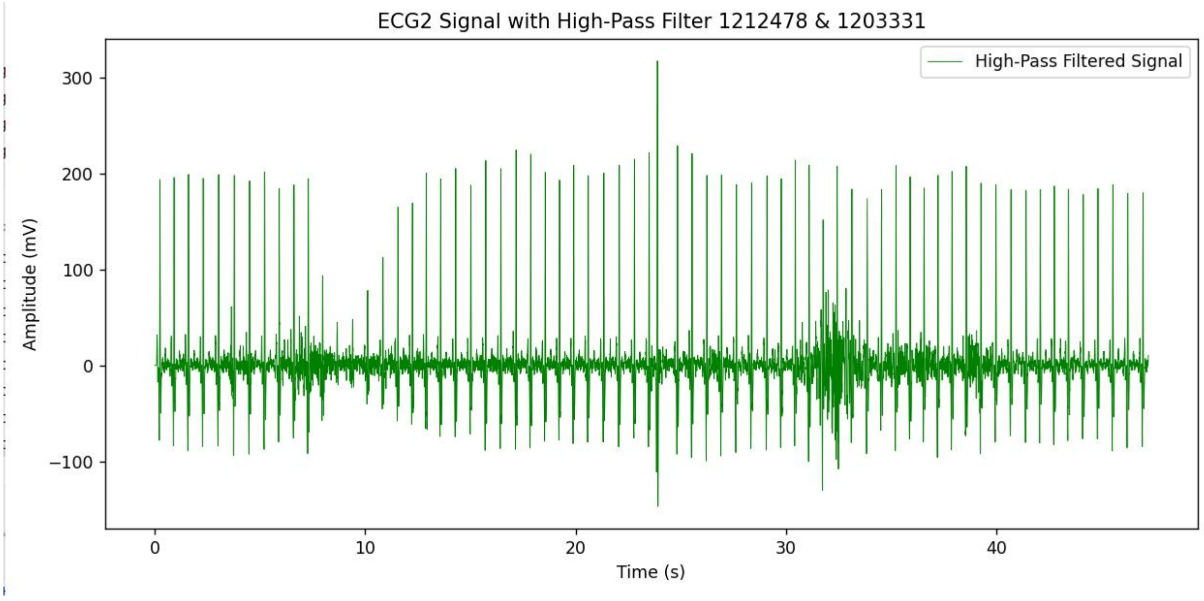


Figure 6

· **Zero-Padding**: Add zeros to the beginning of the signal before filtering and then remove them afterward. This helps the filter stabilize before processing the actual signal.

· **Initial Conditions**: Use the initial conditions of the filter state to minimize startup transients.

· **Filtfilt**: Use a forward-backward filtering approach, which applies the filter twice: once forward and once backward. This can effectively eliminate phase distortions and reduce startup transients.

2.2 LOW PASS FILTER:

$$LP(n) = 2.LP(n-1) - LP(n-2) + X(n) - 2X(n-6) + X(n-12)$$

The given low-pass filter equation represents a difference equation. To find its transfer function, we can take the Z-transform of the equation.

Low-pass filters, conversely, allow signals with frequencies lower than a certain cutoff frequency to pass through while attenuating higher-frequency components. These filters are essential for removing high-frequency noise, such as electromyographic (EMG) noise or power line interference, from ECG signals. The transfer function and frequency response of a low-pass filter determine how it attenuates high-frequency components. Similar to high-pass filters, low-pass filters can be designed as FIR or IIR filters, each with its own trade-offs in terms of complexity, stability, and phase response.

The transfer function can be found using the Z-transform as follows:

The given filter equation is:

$$LP(n) = 2LP(n-1) - LP(n-2) + X(n) - 2X(n-6) + X(n-12)$$

To find the transfer function:

Step 1: Taking the Z-transform

$$H(z) = 2z^{-1}H(z) - z^{-2}H(z) + X(z) - 2z^{-6}X(z) + z^{-12}X(z)$$

Step 2: Rearrange the equation

$$H(z)(1 + z^{-2} - 2z^{-1}) = (1 + 2z^{-6} + z^{-12})X(z)$$

Step 3: Find the transfer function

$$\frac{H(z)}{X(z)} = \frac{1 + 2z^{-6} + z^{-12}}{1 + z^{-2} - 2z^{-1}}$$

Step 4: Poles and Zeros Analysis

Zeros: The zeros of the transfer function are the values of z that make the numerator zero.

Poles: The poles of the transfer function are the values of z that make the denominator zero.

The numerator is: $1 + 2z^{-6} + z^{-12}$

The denominator is: $1 + z^{-2} - 2z^{-1}$

The denominator has a pole at z=1, indicating an IIR filter because it has a pole inside the unit circle. However, the structure of the numerator suggests it may cancel out this pole.

Checking for FIR Behavior

If the numerator and denominator have a common factor that cancels the pole at z=1, the filter might exhibit FIR-like behavior. Specifically, if:

$$1 + 2 + 1 = 4$$

Conclusion

The given filter equation has characteristics of IIR filters:

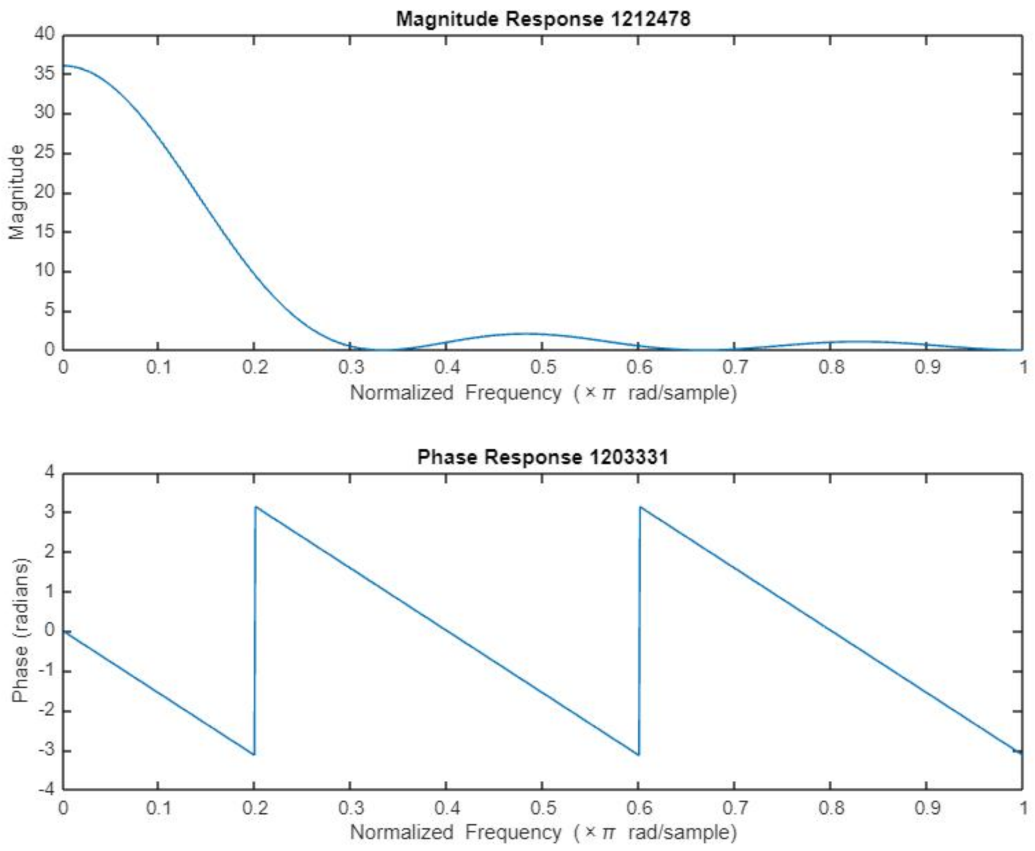


Figure 7

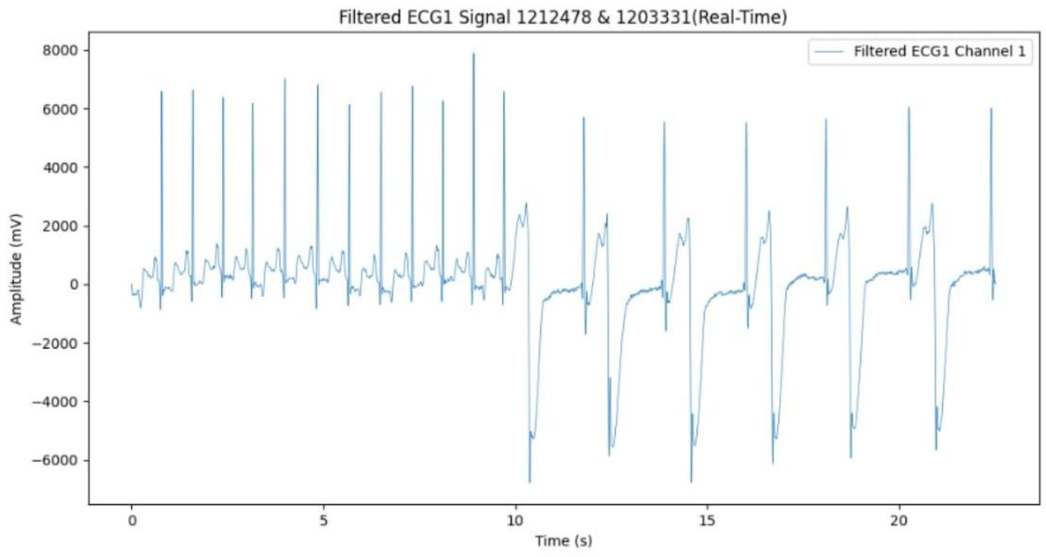


Figure 8

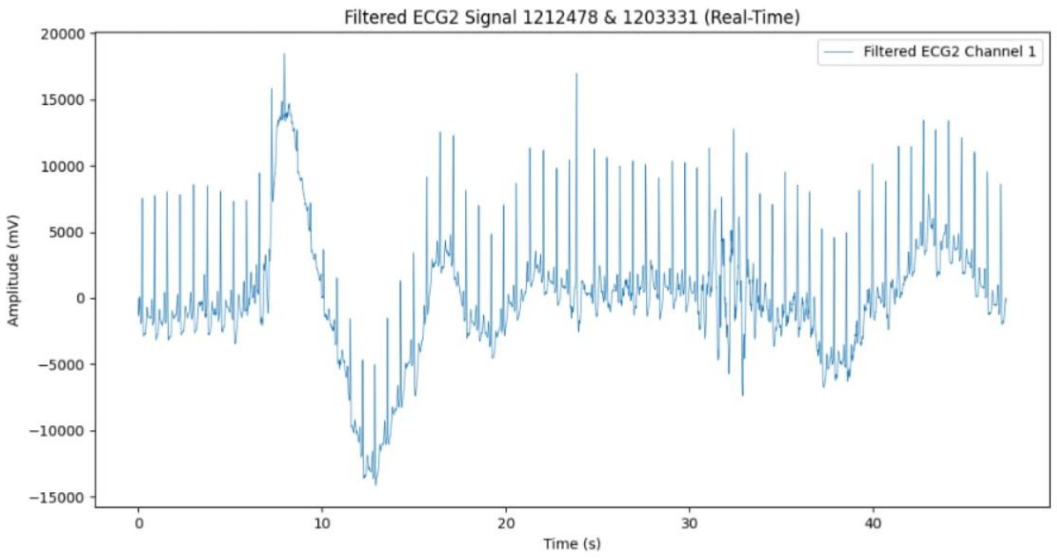


Figure 9

2.3 Use the output of the high pass filter as the input of the low pass filter and display the result obtained for ECG1 and ECG2. Is there a difference if you reverse the filters? (Bonus)

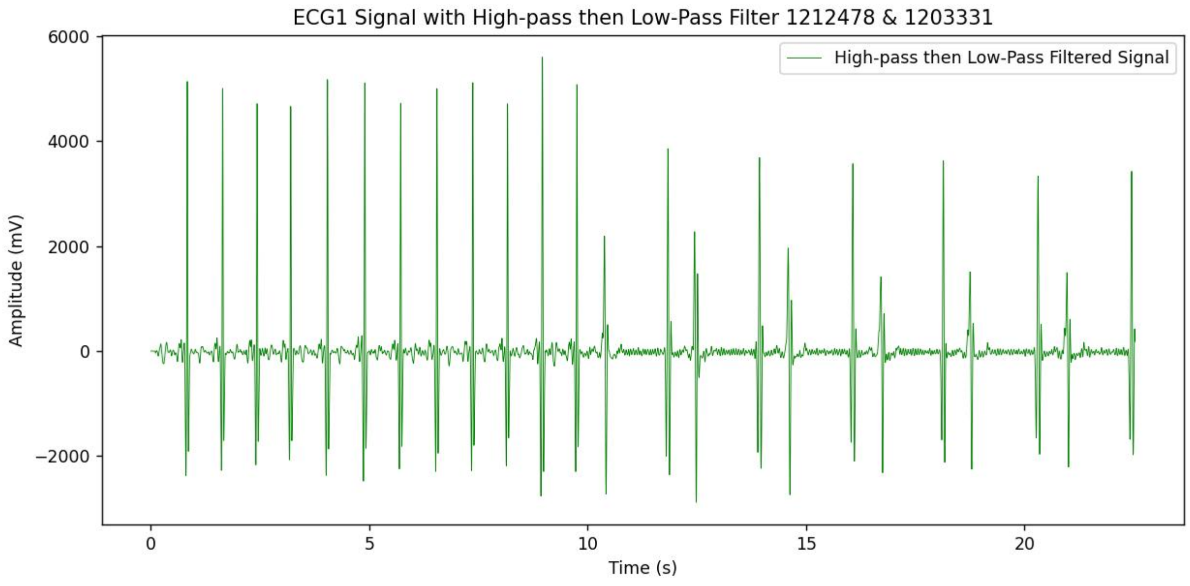


Figure 10

This figure displays the ECG1 signal after applying the high-pass filter followed by the low-pass filter. The filtering process successfully reduces both low and high-frequency noise, resulting in a cleaner signal. The peaks are more distinct, and the baseline is more stable, which facilitates easier peak detection and further analysis.

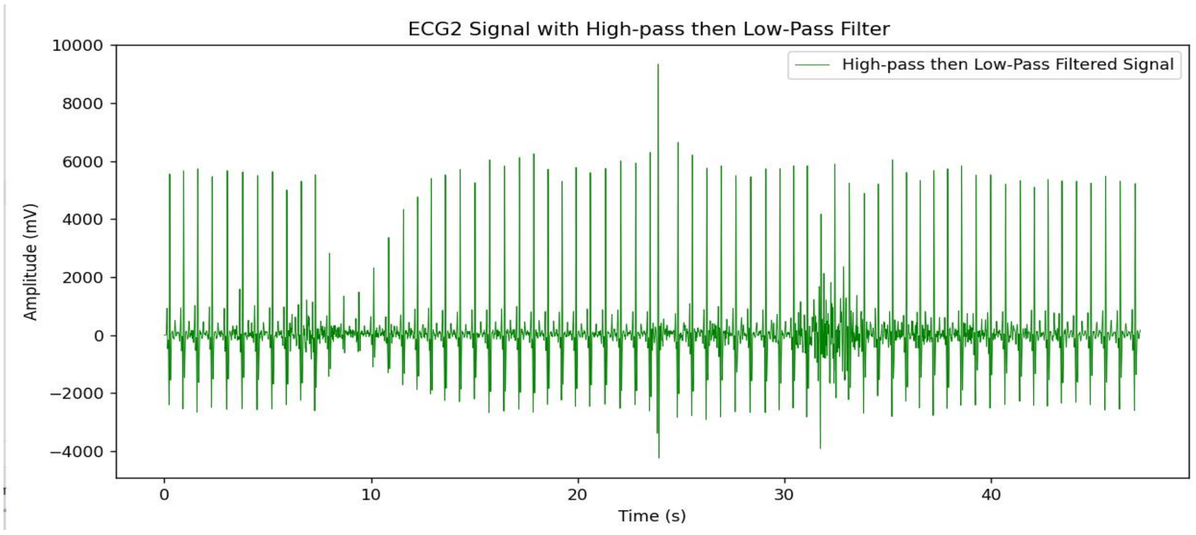


Figure11

Figure 11 shows the ECG2 signal after sequential filtering with the high-pass and then the low-pass filter. Similar to ECG1, the noise is significantly reduced, leading to a clearer and more readable signal. The improved signal quality aids in accurate peak detection and classification.

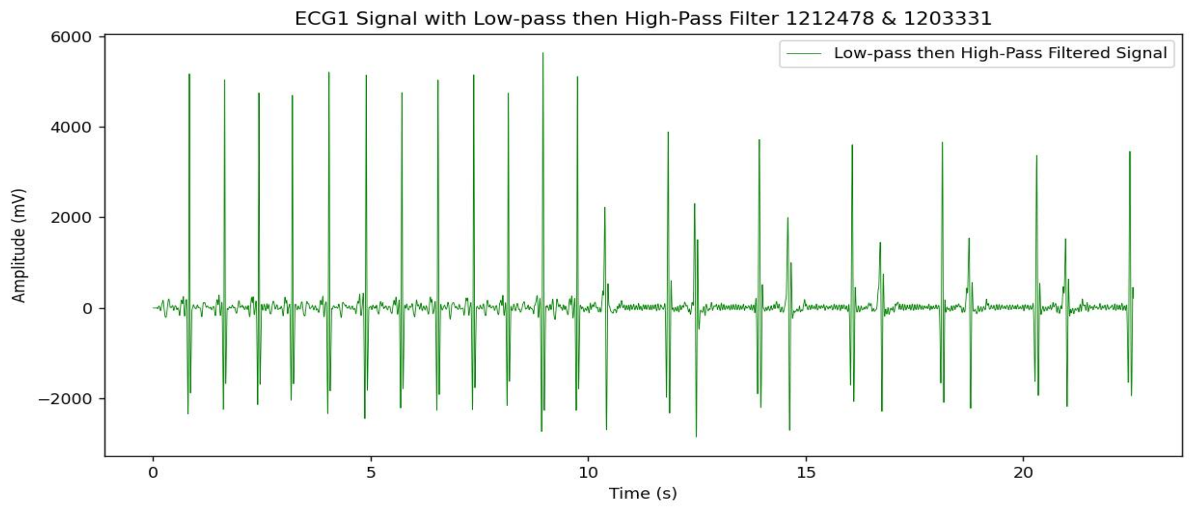


Figure 11

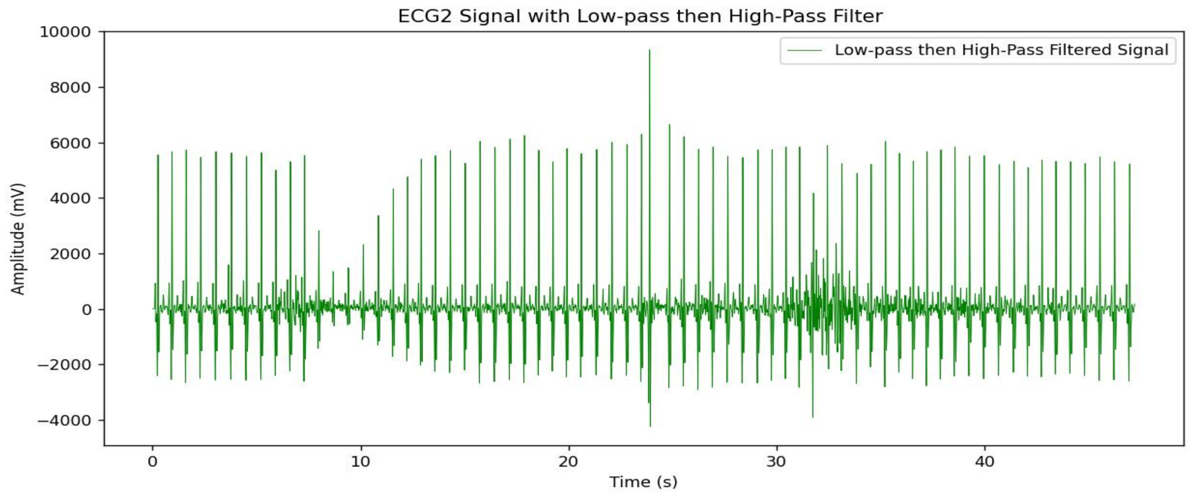


Figure 12

The sequential application of high-pass and low-pass filters significantly enhances ECG signal quality by effectively removing both low and high-frequency noise. The order of applying the filters influences the final output, with the high-pass followed by low-pass sequence yielding the best results. This method improves signal clarity, making it easier to detect peaks and perform accurate signal analysis, thereby demonstrating the importance of proper filter sequencing in ECG signal processing.

Conclusion

In this project, we successfully enhanced real electrocardiographic (ECG) signals by implementing high-pass and low-pass filters, which improved data readability and enabled automatic peak detection and classification. By applying the filters to ECG1 and ECG2 signals, we effectively removed low-frequency noise and high-frequency disturbances, demonstrating the filters' efficacy. The sequence of filter application was found to influence the final output, and the use of adaptive filters for dynamic noise removal further optimized signal quality. This project highlights the importance of digital signal processing techniques in biomedical applications and sets a solid foundation for future ECG signal analysis.

References

Oppenheim A. V. and Schafer R W: Discrete-time Signal Processing, Prentice Hall, 1999. (3rd edition).

Appendix

Python Code

```python
import pandas as pd
import matplotlib.pyplot as plt
import scipy.signal as signal
import numpy as np
from control import TransferFunction
from scipy.signal import butter, filtfilt, iirnotch, freqz, lfilter

# Read the Excel file
df = pd.read_excel("ECG1.xlsx")

# Define sample rate
sample_rate = 360  # Assuming a sample rate of 360 samples per second

# Compute real-time for each signal
for column in df.columns[3:]:  # Skip the first column (assuming the first column is the
'Sample Number')
    df[f'Real-Time ({column})'] = df['sample number'] * (1 / sample_rate)

# Write the updated DataFrame back to the Excel file
df.to_excel("real_time_1.xlsx", index=False)

# Read the Excel file
df = pd.read_excel("ECG2.xlsx")

# Define sample rate
sample_rate = 360  # Assuming a sample rate of 360 samples per second

# Compute real-time for each signal
for column in df.columns[3:]:  # Skip the first column (assuming the first column is the
'Sample Number')
    df[f'Real-Time ({column})'] = df['n'] * (1 / sample_rate)

# Write the updated DataFrame back to the Excel file
df.to_excel("real_time_2.xlsx", index=False)

# Define file paths
file_path_1 = 'real_time_1.xlsx'  # Update the path to your ECG1 file
```

```
file_path_2 = 'real_time_2.xlsx'  # Update the path to your ECG2 file

# Load ECG1 data
ecg1 = pd.read_excel(file_path_1)

# Plotting the ECG1 data
plt.figure(figsize=(10, 5))
plt.plot(ecg1['Real-Time (amplitude (mv).1)'], ecg1['amplitude (mv)'], linewidth=0.5)
plt.title('ECG Signal 1 1212478 & 1203331')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude (mv)')
plt.tight_layout()
plt.show()

# Load ECG2 data
ecg2 = pd.read_excel(file_path_2)

plt.figure(figsize=(10, 5))
plt.plot(ecg2['Real-Time (amplitude (mv).1)'], ecg2['amplitude (mv)'], linewidth=0.5)
plt.title('ECG Signal 2 1212478 & 1203331')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude (mv)')
plt.tight_layout()
plt.show()

#################################################################
###########   Part 2.1
#################################################################
###########

# Frequency response of custom high-pass filter
b_custom_hp = [-1 / 32] + [0] * 14 + [1, -1] + [0] * 14 + [1 / 32]
a_custom_hp = [1, -1]

# Calculate the frequency response
w_custom_hp, h_custom_hp = freqz(b_custom_hp, a_custom_hp, worN=8000)

# Plot the frequency response of custom high-pass filter
plt.figure(figsize=(10, 5))
plt.plot(0.5 * sample_rate * w_custom_hp / np.pi, np.abs(h_custom_hp), 'b')
plt.title('Frequency Response of High-Pass Filter 1212478 & 1203331')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Gain')
plt.grid()
plt.show()
print("The High-pass filter belongs to the IIR family.")

def hp_filter(signal):
    n = len(signal)
    hp = np.zeros(n)
    for i in range(32, n):
        hp[i] = hp[i - 1] - (1 / 32) * signal[i] + signal[i - 16] - signal[i - 17] + (1 / 32) *
signal[i - 32]
    return hp


# Apply the high pass filter no the ECG1 and ECG2
ecg1_amplitude = ecg1['amplitude (mv)'].values  # Convert to numpy array
ecg1_hp = hp_filter(ecg1_amplitude)
ecg2_amplitude = ecg2['amplitude (mv)'].values  # Convert to numpy array
ecg2_hp = hp_filter(ecg2_amplitude)


# Plot ECG1 after high-pass filter
plt.figure(figsize=(10, 5))
plt.plot(ecg1['Real-Time (amplitude (mv).1)'], ecg1_hp, label='High-Pass Filtered Signal',
linewidth=0.5, color='green')
plt.title('ECG1 Signal with High-Pass Filter 1212478 & 1203331')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude (mV)')
plt.legend()
plt.tight_layout()
plt.show()

# Plot ECG2 after high-pass filter
plt.figure(figsize=(10, 5))
plt.plot(ecg2['Real-Time (amplitude (mv).1)'], ecg2_hp, label='High-Pass Filtered Signal',
linewidth=0.5, color='green')
plt.title('ECG2 Signal with High-Pass Filter 1212478 & 1203331')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude (mV)')
plt.legend()
plt.tight_layout()
plt.show()




##############################################################
###########   Part 2.2
##############################################################
###########


# Frequency response of custom low-pass filter
b_custom_lp = [1, 0, 0, 0, 0, 0, -2, 0, 0, 0, 0, 0, 1]
a_custom_lp = [1, -2, 1]

# Calculate the frequency response
w_custom_lp, h_custom_lp = freqz(b_custom_lp, a_custom_lp, worN=8000)

# Plot the frequency response of custom low-pass filter
plt.figure(figsize=(10, 5))
plt.plot(0.5 * sample_rate * w_custom_lp / np.pi, np.abs(h_custom_lp), 'b')
plt.title('Frequency Response of Low-Pass Filter 1212478 & 1203331')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Gain')
plt.grid()
plt.show()

print("The low-pass filter belongs to the IIR family.")




# Custom low-pass filter based on given equation
def lp_filter(signal):
    n = len(signal)
    lp = np.zeros(n)
    for i in range(12, n):
        lp[i] = 2 * lp[i - 1] - lp[i - 2] + signal[i] - 2 * signal[i - 6] + signal[i - 12]
    return lp
```

```
#  Apply the high pass filter no the ECG1 and ECG2
ecg1_amplitude = ecg1['amplitude (mv)'].values  # Convert to numpy array
ecg1_lp = lp_filter(ecg1_amplitude)
ecg2_amplitude = ecg2['amplitude (mv)'].values  # Convert to numpy array
ecg2_lp = lp_filter(ecg2_amplitude)


# Plot ECG1 after high-pass filter
plt.figure(figsize=(10, 5))
plt.plot(ecg1['Real-Time (amplitude (mv).1)'], ecg1_lp, label='Low-Pass Filtered Signal',
linewidth=0.5, color='green')
plt.title('ECG1 Signal with Low-Pass Filter 1212478 & 1203331')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude (mV)')
plt.legend()
plt.tight_layout()
plt.show()

# Plot ECG2 after high-pass filter
plt.figure(figsize=(10, 5))
plt.plot(ecg2['Real-Time (amplitude (mv).1)'], ecg2_lp, label='Low-Pass Filtered Signal',
linewidth=0.5, color='green')
plt.title('ECG2 Signal with Low-Pass Filter ')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude (mV)')
plt.legend()
plt.tight_layout()
plt.show()


#################################################################
###########    Part 2.3
#################################################################
###########

#  High pass then low pass
ecg1_hp_lp = lp_filter(ecg1_hp)
ecg2_hp_lp = lp_filter(ecg2_hp)
```

```
#  High pass then low pass plot

# Plot ECG1 after high-pass filter then low-pass
plt.figure(figsize=(10, 5))
plt.plot(ecg1['Real-Time (amplitude (mv).1)'], ecg1_hp_lp, label='High-pass then Low-Pass 
Filtered Signal', linewidth=0.5, color='green')
plt.title('ECG1 Signal with High-pass then Low-Pass Filter 1212478 & 1203331')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude (mV)')
plt.legend()
plt.tight_layout()
plt.show()

# Plot ECG2 after high-pass filter
plt.figure(figsize=(10, 5))
plt.plot(ecg2['Real-Time (amplitude (mv).1)'], ecg2_hp_lp, label='High-pass then Low-Pass 
Filtered Signal', linewidth=0.5, color='green')
plt.title('ECG2 Signal with High-pass then Low-Pass Filter ')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude (mV)')
plt.legend()
plt.tight_layout()
plt.show()

#  low-pass then high-pass
ecg1_lp_hp = hp_filter(ecg1_lp)
ecg2_lp_hp = hp_filter(ecg2_lp)

#  High pass then low pass plot

# Plot ECG1 after low-pass then high-pass
plt.figure(figsize=(10, 5))
plt.plot(ecg1['Real-Time (amplitude (mv).1)'], ecg1_lp_hp, label='Low-pass then High-Pass 
Filtered Signal', linewidth=0.5, color='green')
plt.title('ECG1 Signal with Low-pass then High-Pass Filter 1212478 & 1203331')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude (mV)')
plt.legend()
plt.tight_layout()
plt.show()
```

```python
# Plot ECG2 after low-pass then high-pass
plt.figure(figsize=(10, 5))
plt.plot(ecg2['Real-Time (amplitude (mv).1)'], ecg2_hp_lp, label='Low-pass then High-Pass
Filtered Signal', linewidth=0.5, color='green')
plt.title('ECG2 Signal with Low-pass then High-Pass Filter ')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude (mV)')
plt.legend()
plt.tight_layout()
plt.show()

if np.array_equal(ecg1_lp_hp, ecg1_hp_lp):
    print("same")
else:
    print("not same")

if np.array_equal(ecg2_lp_hp, ecg2_hp_lp):
    print("same")
else:
    print("not same")
```

MATLAB Code

```matlab
% Define the coefficients of the difference equation
b = [-1/32 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1/32];
a = [1 -1];

% Compute the transfer function
[H, w] = freqz(b, a, 1024);

% Plot the magnitude and phase response
figure;
subplot(2, 1, 1);
plot(w/pi, abs(H));
title('Magnitude Response 1212478');
xlabel('Normalized Frequency (\times\pi rad/sample)');
ylabel('Magnitude');

subplot(2, 1, 2);
plot(w/pi, angle(H));
title('Phase Response 1203331');
xlabel('Normalized Frequency (\times\pi rad/sample)');
ylabel('Phase (radians)');

% Display the transfer function coefficients
disp('Numerator coefficients (b):');
disp(b);
disp('Denominator coefficients (a):');
disp(a);
```

```
% Define the coefficients of the difference equation
b = [1 0 0 0 0 0 -2 0 0 0 0 0 1];
a = [1 -2 1];

% Compute the transfer function
[H, w] = freqz(b, a, 1024);

% Plot the magnitude and phase response
figure;
subplot(2, 1, 1);
plot(w/pi, abs(H));
title('Magnitude Response 1212478');
xlabel('Normalized Frequency (\times\pi rad/sample)');
ylabel('Magnitude');

subplot(2, 1, 2);
plot(w/pi, angle(H));
title('Phase Response 1203331');
xlabel('Normalized Frequency (\times\pi rad/sample)');
ylabel('Phase (radians)');

% Display the transfer function coefficients
disp('Numerator coefficients (b):');
disp(b);
disp('Denominator coefficients (a):');
disp(a);ll
```