Birzeit University
Faculty of Engineering & Technology
Department of Electrical & Computer Engineering
DIGITAL SIGNAL PROCESSING (DSP)
ENCS4310

Filtering A Real Electrocardiographic Signal
DSP Course Project

Prepared By:
Zeina Odeh 1190083
Raghad Afaghani 1192423

Supervised By:
Dr. Qadri Mayyaleh

Birzeit
June, 2024

Abstract

In order to improve peak identification and classification, this project applies high-pass and low-pass filters to electrocardiogram (ECG) signals using Python code. The filters are tested on two ECG datasets to enhance signal clarity and processing efficiency. To determine the filter families (FIR or IIR), the transfer functions and frequency responses were computed. Advanced methods, such as adaptive filters for noise reduction, were also used.

Contents

1 Introduction

1.1 Dataset

The dataset consists of two Electrocardiogram (ECG) files: `ECG1.csv` and `ECG2.csv`. Each file contains columns for the sample number, time in seconds (`temps(s)`), and amplitude in millivolts (`amplitude(mv)`). Additionally, the `ECG1.csv` file includes an extra column for amplitude (`amplitude(mv).1`). These data sets are essential for detailed analysis and examination of ECG signals, crucial in assessing cardiac health.

1.2 Frequency Response

Frequency response is a way to measure how a system reacts to different input frequencies - it shows how the strength and timing of the output signal changes compared to the input across a range of frequencies. This is a very useful tool for engineers to understand and design all kinds of systems, not just audio ones. By looking at the frequency response, they can simplify the math and figure out how the system will behave when presented with different frequencies [1].

1.3 High Pass Filter

A high-pass filter (HPF) is an electronic filter designed to allow signals with frequencies above a specified cutoff frequency to pass through while attenuating signals with frequencies below the cutoff. The degree of attenuation for each frequency depends on the specific design of the filter. Typically modeled as a linear time-invariant system, a high-pass filter is also referred to as a low-cut filter or bass-cut filter, particularly in audio engineering contexts [2].

1.4 Low Pass Filter

A low-pass filter allows signals with frequencies below a specified cutoff frequency to pass through while attenuating those above it, with the precise frequency response varying based on the filter's design. Often referred to as a high-cut or treble-cut filter in audio applications, this type of filter serves as the counterpart to a high-pass filter [3].

2 Data Visualization

2.1 Insert the real-time column for each signal

We write a Python code to load and clean Electrocardiogram (ECG) data from two CSV files, `'ECG1.csv'` and `'ECG2.csv'`. It utilizes key libraries such as `pandas` for data manipulation, `matplotlib.pyplot` for data visualization, and `scipy.signal` for signal processing. The script first reads the CSV files, handling any issues with the decimal separator in the 'temps (s)' column by replacing '·' with '.' and converting the column to a float data type. It then converts the 'amplitude (mv)' column to a numeric data type, handling any errors by setting invalid values to `NaN`. The resulting cleaned-up data is stored in two separate `pandas` DataFrames, one for each ECG file, which can then be used for further analysis or visualization as needed.

	sample number	temps (s)	amplitude (mv)	amplitude (mv).1
0	0.0	0.00	3.0	950
1	1.0	0.00	-13.0	934
2	2.0	0.01	-13.0	934
3	3.0	0.01	-9.0	938
4	4.0	0.01	-9.0	938

Figure 1: ECG1 File

	n	temps (s)	amplitude (mv)	amplitude (mv).1
0	0.0	0.00	-42.0	942.0
1	1.0	0.00	-43.0	941.0
2	2.0	0.01	-40.0	944.0
3	3.0	0.01	-39.0	945.0
4	4.0	0.01	-36.0	948.0

Figure 2: ECG2 File

2.2 Display each of these signals

After loading and cleaning the data, we create a code for plots each ECG signal, displaying the `'amplitude (mv)'` data against the `'temps "Time"(s)'` data. This visualization step allows to inspect of the loaded and cleaned ECG data, which can be useful for further analysis or to ensure the data has been properly prepared.

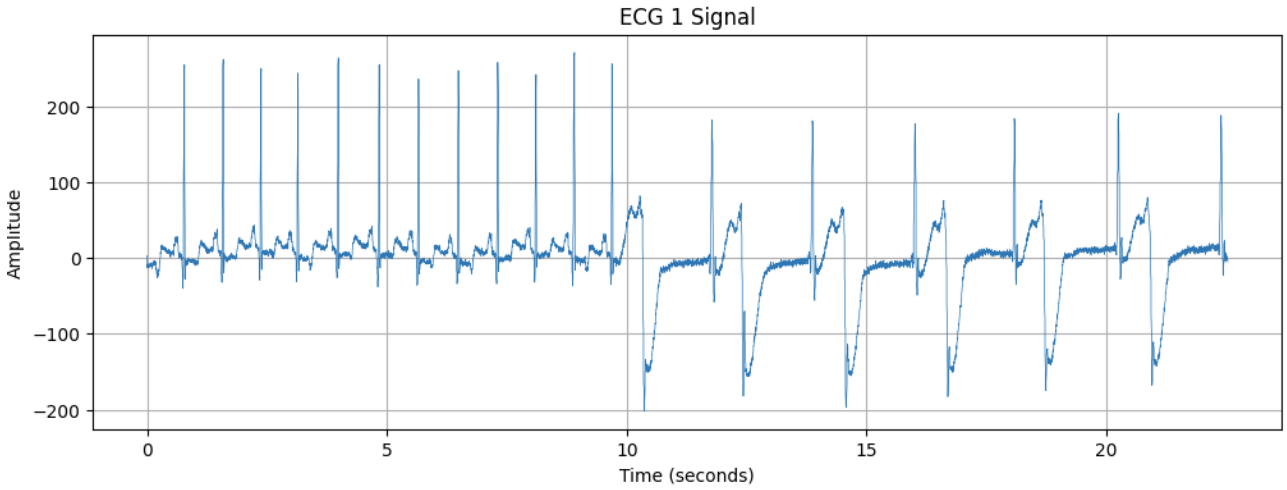

Figure 3: ECG1 Signal

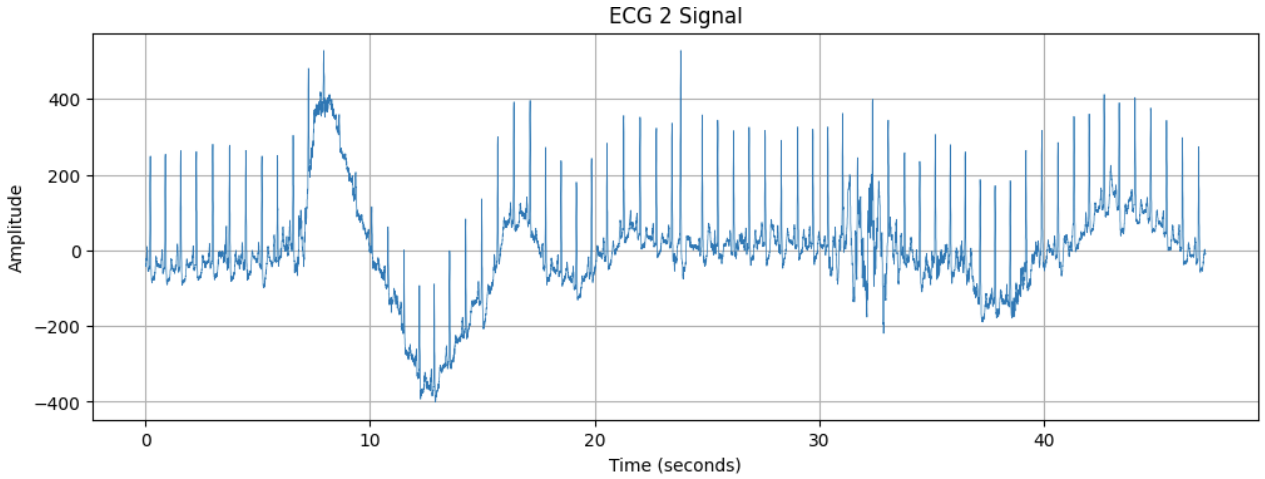

Figure 4: ECG2 Signal

2.3 Improve the readability of data

The bandpass filter is crucial for cleaning the "amplitude (mv)" data in ECG files by removing low-frequency noise and high-frequency noise. This ensures the ECG signals accurately reflect the heart's electrical activity, leading to clearer,better visualization data and more accurate analysis.

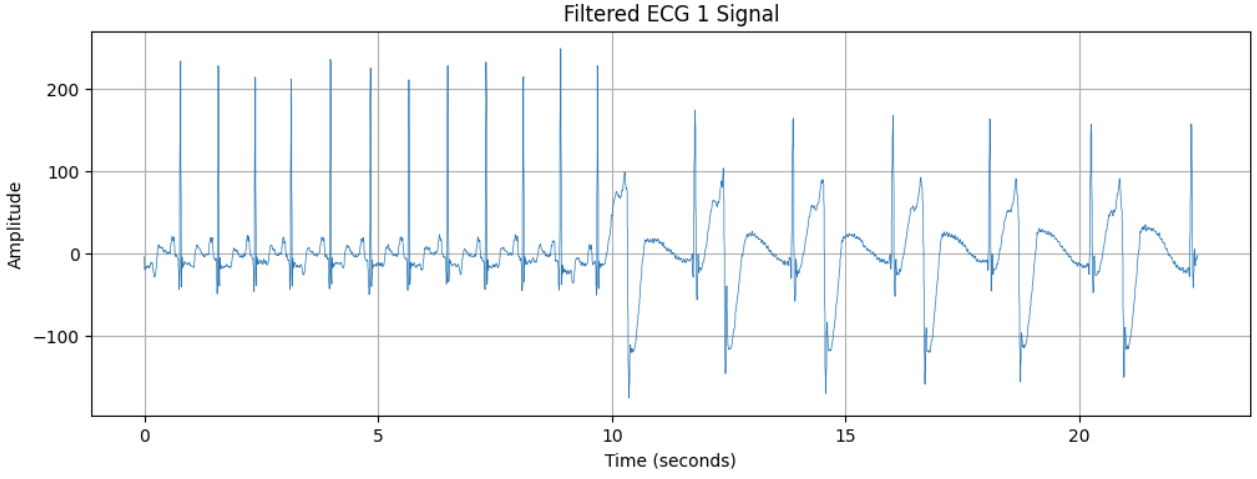

Figure 5: Bandpass ECG 1 Signal

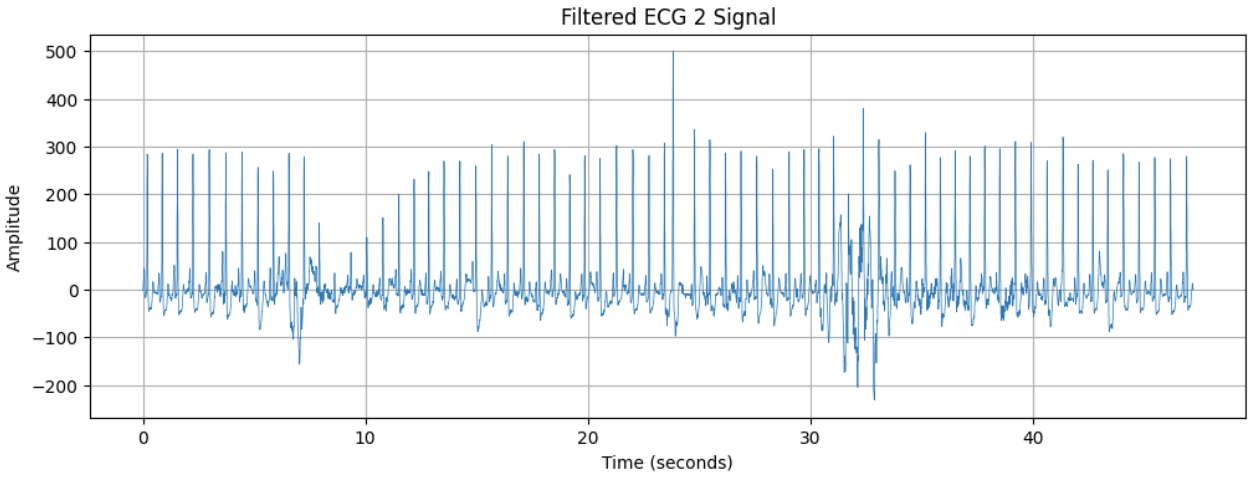

Figure 6: Bandpass ECG 2 Signal

3 Filtering the ECG

We will use two filters, the output of the high-pass filter is HP(n) and the low pass is LP(n). The input is X(n).

3.1 High-pass filter

- **Transfer Function**

  Given:

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

  Taking the Z-transform of both sides, we get:

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

  This can be written as:

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

  Solving for the transfer function $H(z)$ by isolating $HP(z)$ and $X(z)$ terms:

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

$$HP(z)(1-z^{-1}) = X(z)\left(z^{-16} - z^{-17} - \frac{1}{32} + \frac{1}{32}z^{-32}\right)$$

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

  Factoring the numerator:

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

  Simplifying:

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

$$H(z) = \frac{31 - z^{-16} + z^{-17} - z^{-32}}{32(1 - z^{-1})}$$

Which is equal

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

Therefore, the transfer function $H(z)$ is:

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

- **Frequency response**

  To calculate the frequency response of the transfer function $H(z)$, we need to substitute $z = e^{j\omega}$, where ω is the angular frequency.

  The transfer function $H(z)$ is:

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

  Substituting $z = e^{j\omega}$, we get:

$$H(e^{j\omega}) = \frac{e^{-16j\omega} - e^{-17j\omega} + e^{-32j\omega}}{1 - e^{-j\omega}}$$

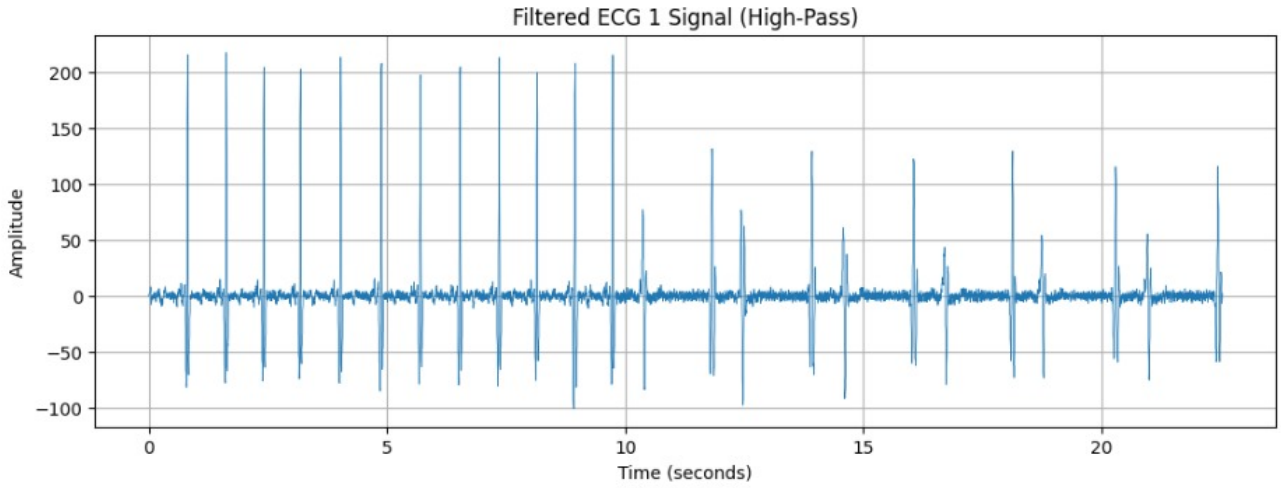

Figure 7: High Pass Filtered ECG 1 Signal

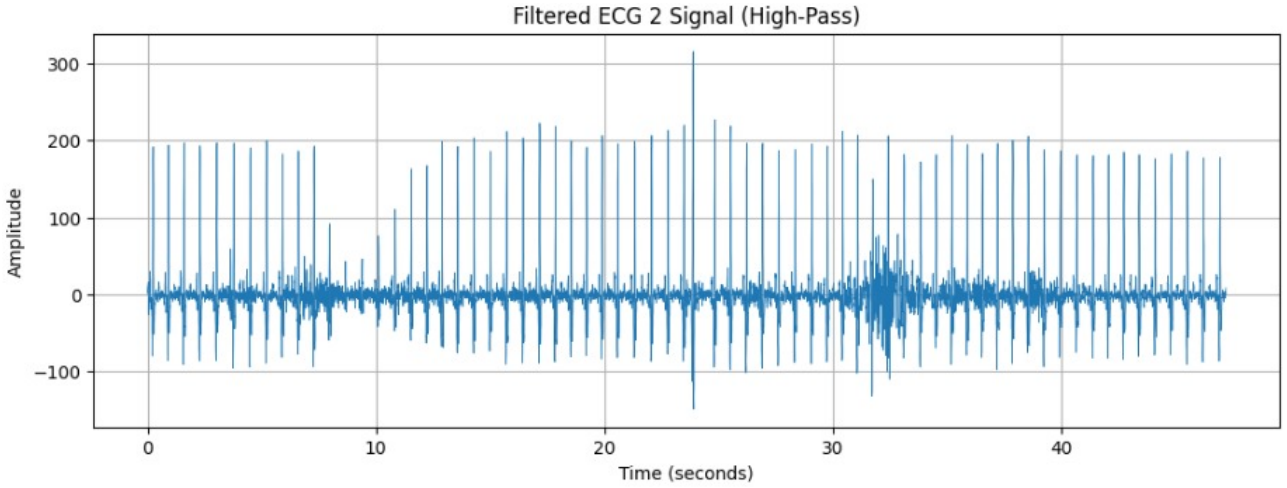

Figure 8: High Pass Filtered ECG 2 Signal

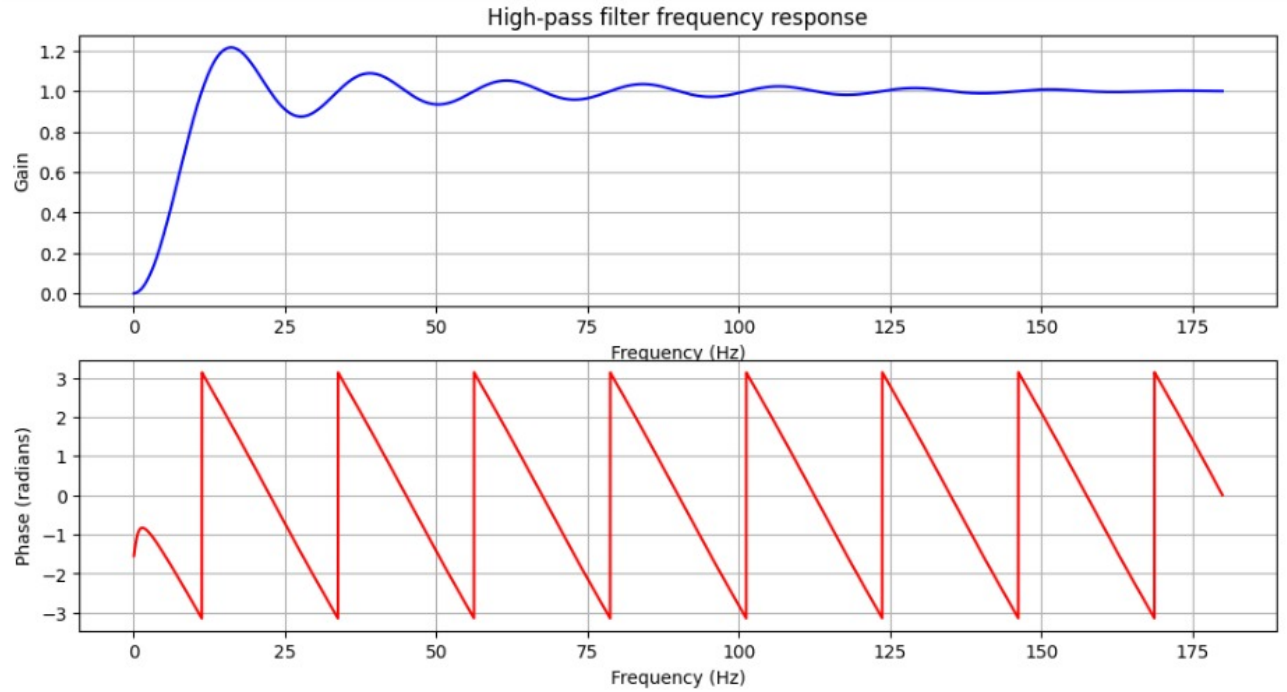

Figure 9: High Pass Frequency and Magnitude Response

- **Which family does it belong to (FIR, IIR)**

  This transfer function has a denominator term $32(1-z^{-1})$, which indicates that it has an infinite impulse response (IIR). In contrast, a Finite Impulse Response (FIR) filter would have a transfer function with only a numerator term, without any denominator.

- **Applying high-pass filter to the ECG1 and then ECG2 signal**

  High-pass filters attenuate low-frequency frequencies while permitting high-frequency ones to flow through. This is essential for eliminating low-frequency noise in ECG signal processing, such as baseline wander.

  The filter coefficients b_{hp} (numerator) and a_{hp} (denominator) define the behavior of the filter. The

general form of a digital filter can be described by the difference equation:

$$y[n] = \sum_{k=0}^{M} b[k] \cdot x[n-k] - \sum_{k=1}^{N} a[k] \cdot y[n-k]$$

Explanation of the Coefficients:

$$a_{\text{hp}} = [1, -1]:$$

This corresponds to a simple first-order difference equation, which can be written as:

$$y[n] = x[n] - x[n-1]$$

We use this difference equation since we indicate that the filter is an IIR (Infinite Impulse Response) filter, which uses both the current and previous input samples.

$$b_{\text{hp}} \text{ Coefficients}:$$

The coefficients of b_{hp} are designed to create a specific high-pass filter characteristic. The sequence $[-1/32, 0, \ldots, 0, 1, -1, 0, \ldots, 0, 1/32]$ can be seen as implementing a combination of two delayed signals and scaled versions.

Specifically:

$-\frac{1}{32}$ at the beginning and $+\frac{1}{32}$ at the end introduce a very low-frequency attenuation.

The center values (16th and 17th positions, $1, -1$) differentiate the input signal over a range, resulting in a high-pass filter. The coefficient placement makes sure that the high-pass filter affects frequencies around a particular cut-off frequency, which makes it useful for eliminating slow variations (baseline wander) without substantially altering the primary components of the ECG signal.

The recursive difference equation is used to carry out the filtering procedure. For every signal sample n:

- The equation takes into account the contributions from the 16th, 17th, and 32nd prior samples if n is 32 or more, according to the particular high-pass filter design.
- Since not all of the prior samples are accessible, the equation becomes simpler for samples where n is between 16 and 31.
- Because there are even fewer prior samples available, the equation becomes even simpler for samples when n is less than 16.

This guarantees that the filter is applied uniformly across the signal, even at the start where there aren't as many samples before it.

- **How to partially resolve the problems occurring at the start of the signal?**
  We used the lfilter function from the SciPy library which is a crucial tool for applying linear filters to input signals, such as the ECG data in our code . It takes two sets of coefficients, b and a, which define the characteristics of the filter in the z-domain. The b coefficients represent the numerator of the transfer function, while the a coefficients represent the denominator. By processing the input signal

x through these filter coefficients using the lfilter function, unwanted frequency components can be effectively removed or emphasized, allowing for targeted signal processing.

```
Number of samples in original ECG1: 8113
Number of samples in original ECG2: 17003
Number of samples in high-pass filtered ECG1: 8113
Number of samples in high-pass filtered ECG2: 17003
```

Figure 10: Resolving the problems occurring at the start of the signal

The results indicate that the number of samples remains the same before and after applying the filters. This suggests that the problem caused at the start of the signal by convolving one of the samples before the other samples have been processed has been resolved.

3.2 Low-pass filter

- **Transfer Function**

Given:

$$Z\{LP(n)\} = Z\{2 \cdot LP(n-1)\} - (Z\{LP(n-2)\} + X(n) - 2X(n-6) + X(n-12))$$

Rearranging, we get:

$$H(z) = \frac{Z\{LP(n)\}}{Z\{x(n)\}}$$

Substituting, we have:

$$H(z) = \frac{Z\{2 \cdot LP(n-1)\} - (Z\{LP(n-2)\} + X(n) - 2X(n-6) + X(n-12))}{Z\{x(n)\}}$$

Factoring out $H(z)$ from the left side:

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

Therefore, the transfer function $H(z)$ is:

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

- **Frequency response**

$H(e^{j\omega})$ for the given transfer function $H(z) = 2z^{-1} - z^{-2} - 1 + 2z^{-6} - z^{-12}$ is obtained by substituting $z = e^{j\omega}$:

$$H(e^{j\omega}) = 2e^{-j\omega} - e^{-j2\omega} - 1 + 2e^{-j6\omega} - e^{-j12\omega}$$

Simplifying the expression, we get:

$$H(e^{j\omega}) = 2e^{-j\omega} - e^{-j2\omega} - 1 + 2e^{-j6\omega} - e^{-j12\omega}$$

This represents the frequency response of the filter. To further analyze it, The frequency response $H(e^{j\omega})$ can be expressed in terms of cosine and sine functions as follows:

The magnitude response is given by:

$$|H(e^{j\omega})| = \sqrt{[\text{Re}(H(e^{j\omega}))]^2 + [\text{Im}(H(e^{j\omega}))]^2}$$

where

$$\text{Re}(H(e^{j\omega})) = -1 + 2\cos(\omega) - \cos(2\omega) + 2\cos(6\omega) - \cos(12\omega)$$
$$\text{Im}(H(e^{j\omega})) = -2\sin(\omega) + \sin(2\omega) - 2\sin(6\omega) + \sin(12\omega)$$

The phase response is given by:

$$\arg(H(e^{j\omega})) = \tan^{-1}\left(\frac{\text{Im}(H(e^{j\omega}))}{\text{Re}(H(e^{j\omega}))}\right)$$

where

$$\text{Re}(H(e^{j\omega})) = -1 + 2\cos(\omega) - \cos(2\omega) + 2\cos(6\omega) - \cos(12\omega)$$
$$\text{Im}(H(e^{j\omega})) = -2\sin(\omega) + \sin(2\omega) - 2\sin(6\omega) + \sin(12\omega)$$

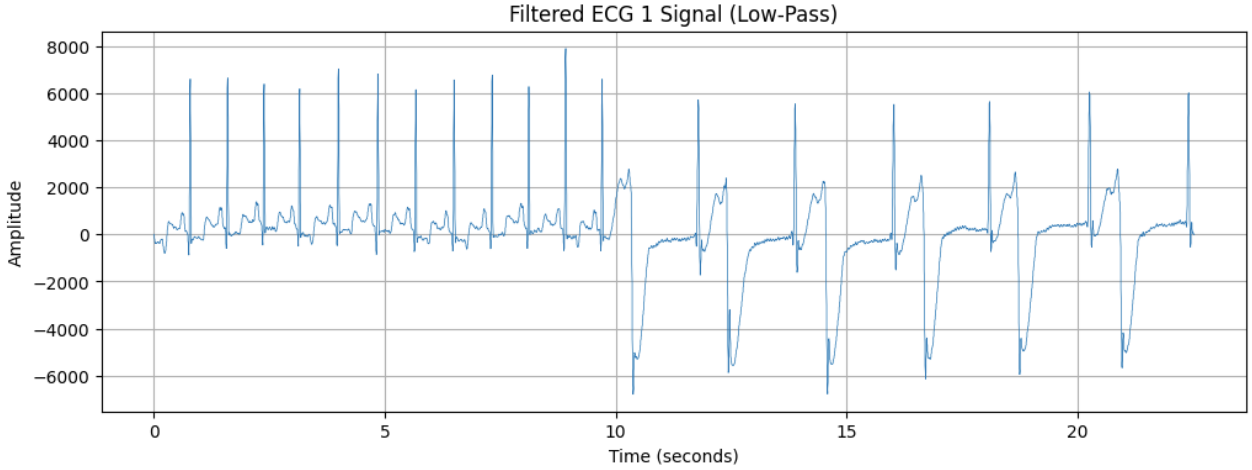

Figure 11: Low-Pass Filtered ECG 1 Signal

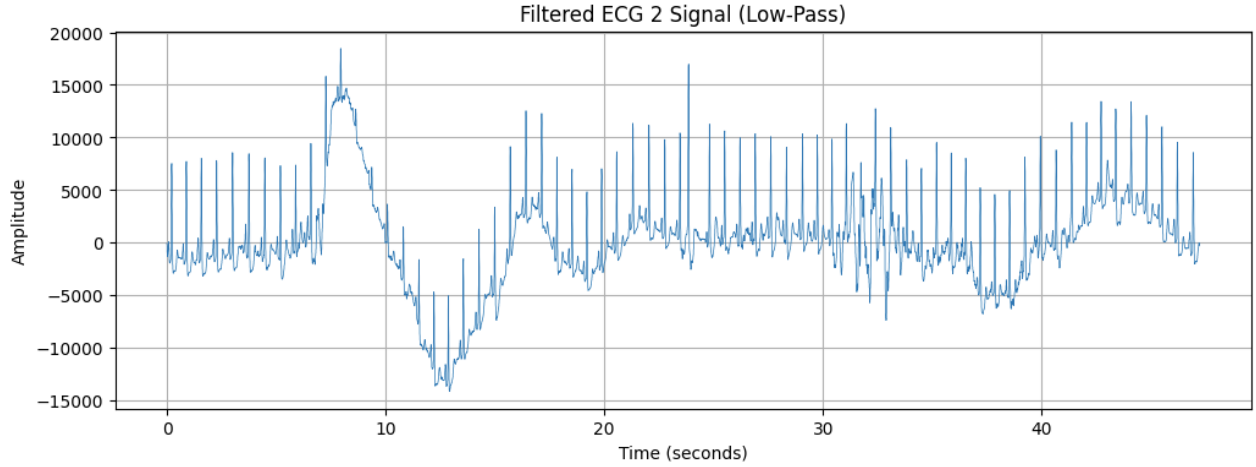

Figure 12: Low-Pass Filtered ECG 2 Signal

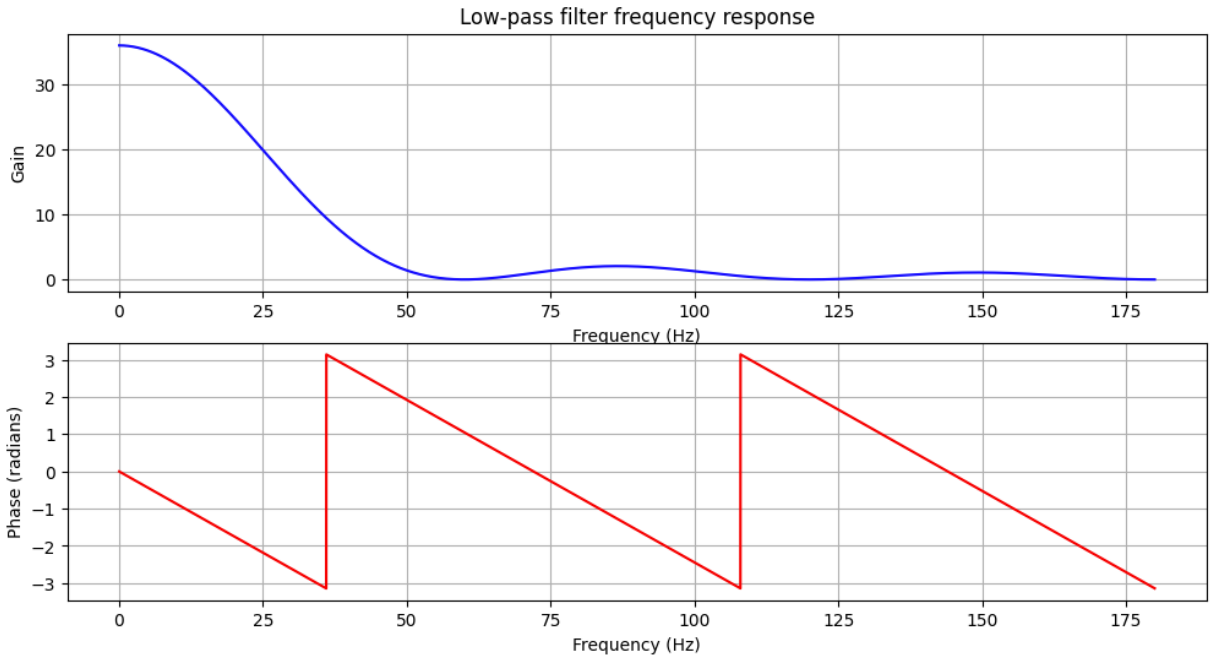

Figure 13: Low Pass Frequency and Magnitude Response

- **Which family does it belong to?**

  The difference equation includes terms $2 \cdot LP(n-1)$ and $-LP(n-2)$, which involve past output values, indicating feedback, which means that it has an infinite impulse response (IIR).

- **Apply this filter to the ECG1 then the ECG2 signal.**

  Low-pass filters are used in signal processing to attenuate high-frequency components while allowing low-frequency components to pass through. This is crucial for smoothing out high-frequency noise in ECG signal processing, such as muscle artifacts or high-frequency interference.

  The filter coefficients b_{lp} (numerator) and a_{lp} (denominator) define the behavior of the filter. The general form of a digital filter can be described by the following recursive difference equation:

$$y[n] = \sum_{k=0}^{M} b[k] \cdot x[n-k] - \sum_{k=1}^{N} a[k] \cdot y[n-k]$$

  Explanation of the Coefficients:

  - $a_{lp} = [1, -2, 1]$: This corresponds to a second-order difference equation, which can be written as:

$$y[n] = 2 \cdot y[n-1] - y[n-2] + x[n] - 2 \cdot x[n-6] + x[n-12]$$

    We use this difference equation since we indicate that the filter is an IIR (Infinite Impulse Response) filter, which uses both the current and previous input samples.

  - b_{lp} Coefficients: The coefficients of b_{lp} are designed to create a specific low-pass filter characteristic. The sequence $[1, 0, 0, 0, 0, 0, -2, 0, 0, 0, 0, 0, 1]$ implements a combination of delayed and scaled versions of the input signal. Specifically:
    * The coefficients in positions 0, 6, and 12 affect the current and delayed samples to smooth out high-frequency noise.
    * The coefficient -2 at position 6 introduces a stronger attenuation of high-frequency components.

The recursive nature of the difference equation ensures that the filter is applied uniformly across the signal.

The recursive difference equation is used to carry out the filtering procedure. For every signal sample n:

- The equation takes into account the contributions from the 12th, 6th, and 0th prior samples if n is 12 or more, according to the particular low-pass filter design.
- Since not all of the prior samples are accessible, the equation becomes simpler for samples where n is between 6 and 11.
- Because there are even fewer prior samples available, the equation becomes even simpler for samples when n is less than 6.

This guarantees that the filter is applied uniformly across the signal, even at the start where there aren't as many samples before it.

3.3 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?

In this part, we are applying filters to the ECG1 and ECG2 signals to clean them. First, we use a high-pass filter to remove low-frequency noise, and then a low-pass filter to remove high-frequency noise. We also reverse the order by applying the low-pass filter first and then the high-pass filter.

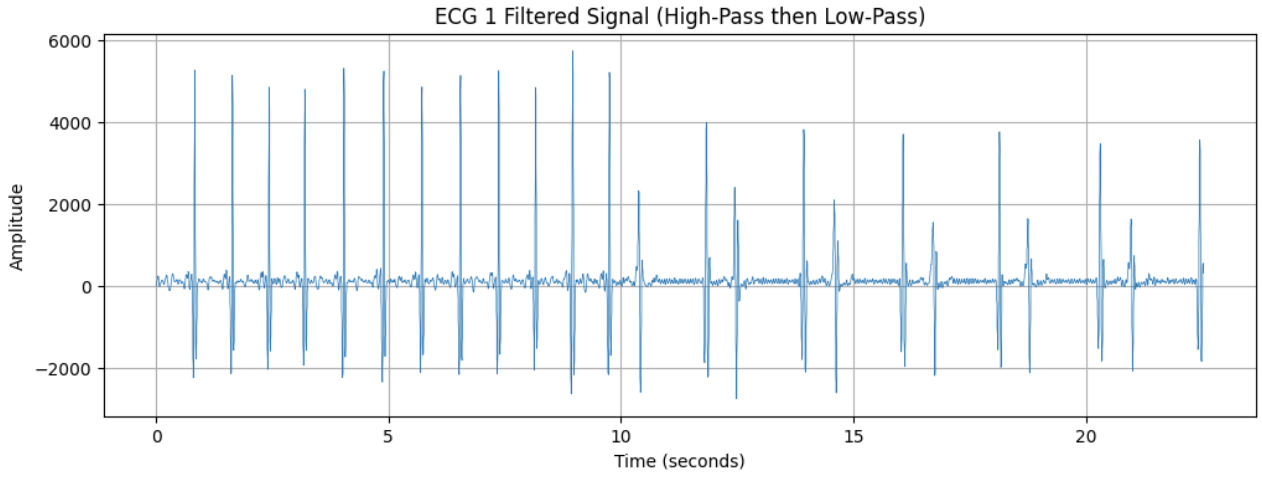

Figure 14: High-Pass then Low-Pass ECG1

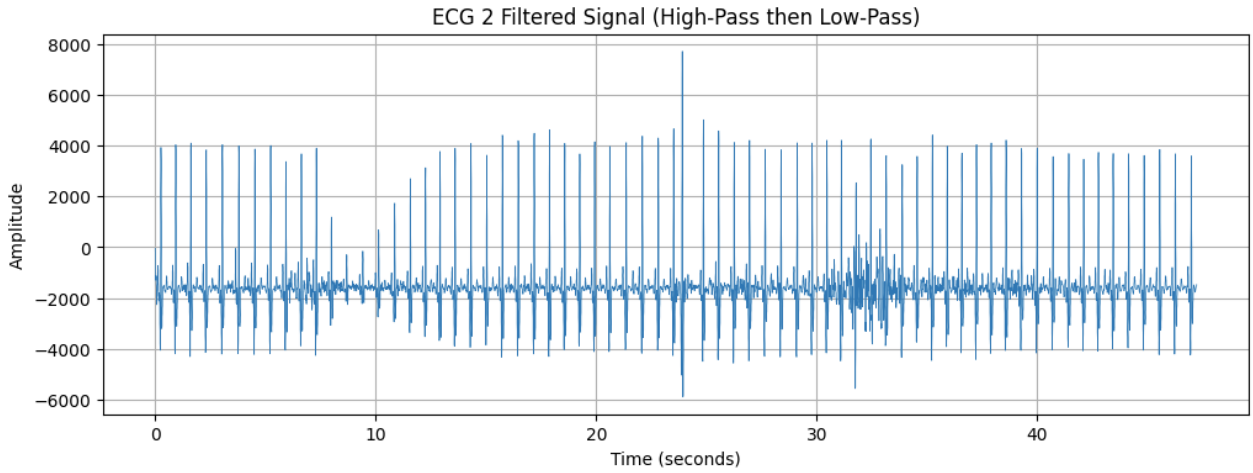

Figure 15: High-Pass then Low-Pass ECG2

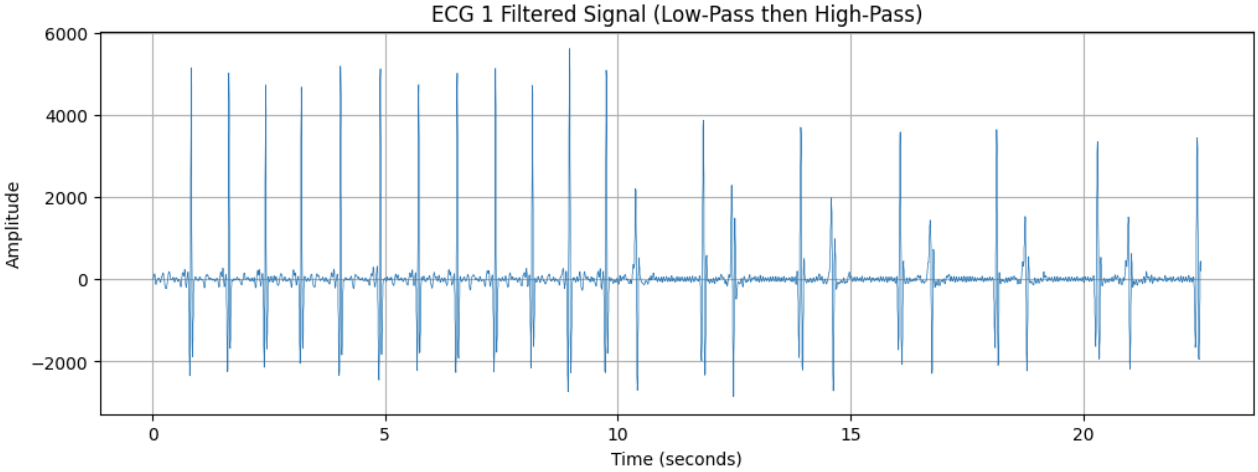

Figure 16: Low-Pass then High-Pass ECG1

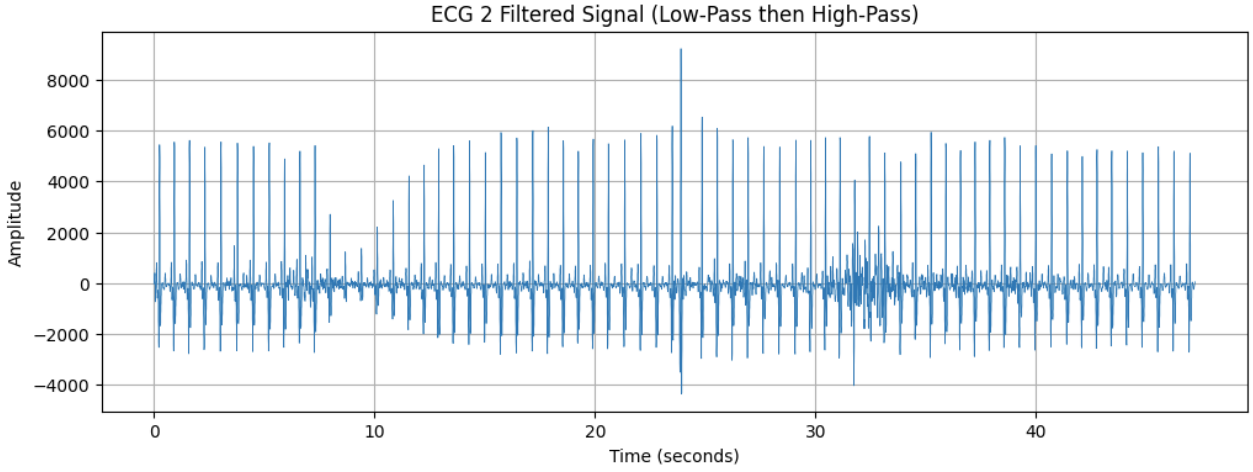

Figure 17: Low-Pass then High-Pass ECG2

4 Bonus : Adaptive filters for noise removal

In this part, We use an adaptive filter to remove noise from ECG signals.The key component is the LMS (Least Mean Squares) adaptive filter, implemented in the `lms_adaptive_filter` function. This function takes four parameters: the desired signal, the input signal, the step size (μ), and the filter order. The filter initializes arrays for the output signal (y), the error signal (e), and the filter weights (w). For each sample, it computes the output by taking the dot product of the current filter weights and a segment of the input signal. It calculates the error as the difference between the desired signal and the output signal. The filter then updates the filter weights to minimize this error using the step size and the error value. By dynamically adjusting the filter weights to reduce the error, the LMS adaptive filter effectively removes noise from the ECG signals.

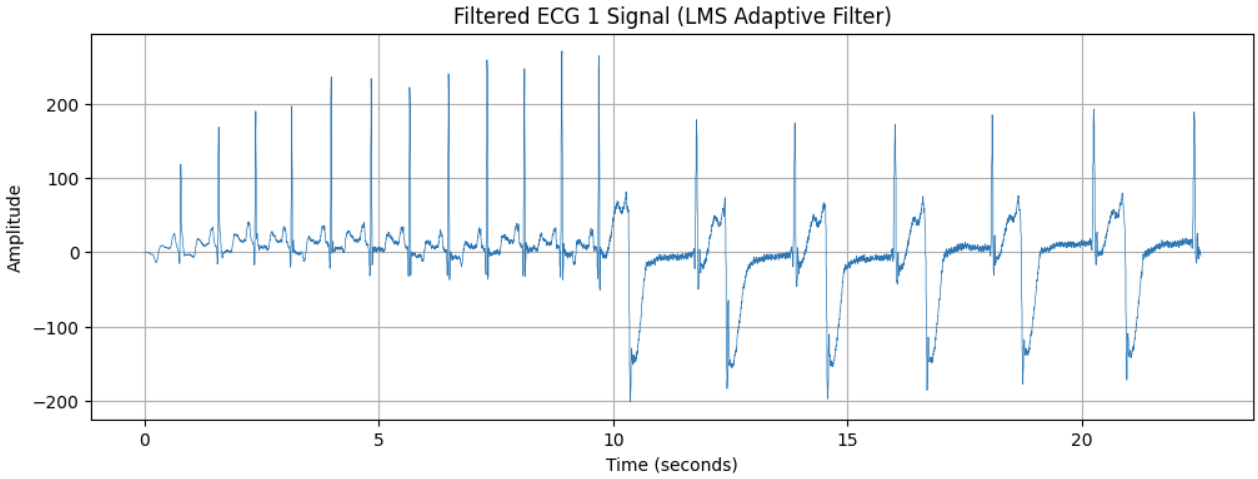

Figure 18: Filtered ECG 1 Signal (LMS Adaptive Filter)

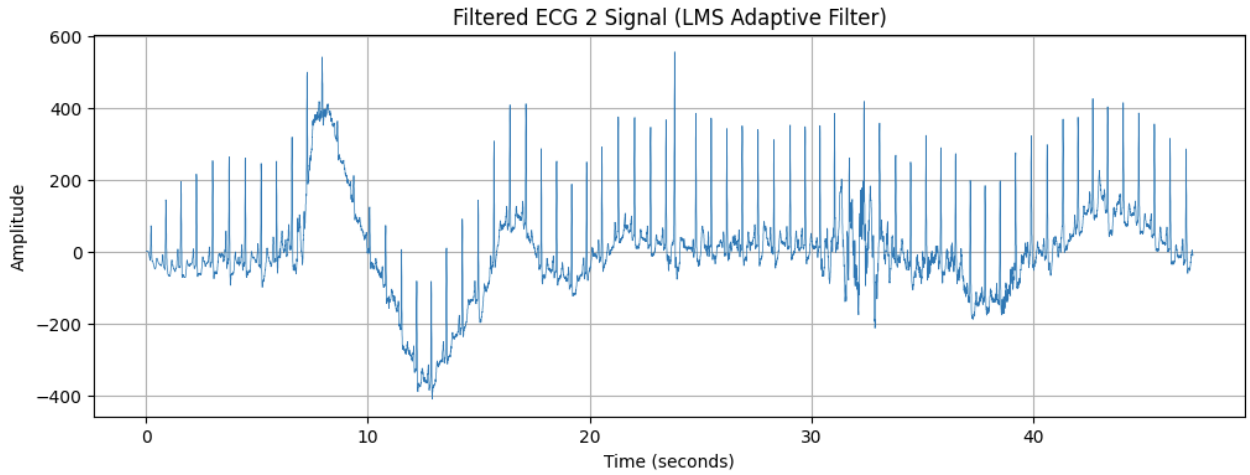

Figure 19: Filtered ECG 2 Signal (LMS Adaptive Filter)

5 Conclusion

In conclusion, our project used a structured approach to use signal processing techniques effectively to improve peak detection and classification in electrocardiogram (ECG) signals. We were able to increase signal clarity and processing efficiency across two ECG datasets by applying high-pass and low-pass filters using Python code. The best filter designs were identified through the assessment of various filter families, which included the calculation of transfer functions and frequency responses. Moreover, the integration of advanced adaptive filtering techniques enabled efficient noise reduction, hence enhancing the quality of the ECG data for analysis.

6 Appendix

6.1 Insert the real-time column for each signal

```
import pandas as pd
import matplotlib.pyplot as plt
from scipy.signal import butter, filtfilt

# Load  and clean data
def load_and_clean_data(filepath):
    data = pd.read_csv(filepath, skiprows=1)
    data['temps (s)'] = data['temps (s)'].str.replace('  ', '.').astype(float)
    data['amplitude (mv)'] = pd.to_numeric(data['amplitude (mv)'], errors='coerce')
    return data

# ECG data
ecg1 = load_and_clean_data('/content/ECG1.csv')
ecg2 = load_and_clean_data('/content/ECG2.csv')
```

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

```
# Plotting
def plot_ecg(data, title):
    plt.figure(figsize=(12, 4))
    plt.plot(data['temps (s)'], data['amplitude (mv)'], linewidth=0.5)
    plt.title(title)
    plt.xlabel('Time (seconds)')
    plt.ylabel('Amplitude')
    plt.grid(True)
    plt.show()

plot_ecg(ecg1, 'ECG 1 Signal')
plot_ecg(ecg2, 'ECG 2 Signal')
```

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

```
   import pandas as pd
import matplotlib.pyplot as plt
from scipy.signal import butter, filtfilt

# Function to load data and clean it
def load_and_clean_data(filepath):
    data = pd.read_csv(filepath, skiprows=1)
    data['temps (s)'] = data['temps (s)'].str.replace('  ', '.').astype(float)
    data['amplitude (mv)'] = pd.to_numeric(data['amplitude (mv)'], errors='coerce')
    data['amplitude (mv)'].fillna(method='ffill', inplace=True)  # Handle NaNs
    return data

# Load ECG data
ecg1 = load_and_clean_data('/content/ECG1.csv')
ecg2 = load_and_clean_data('/content/ECG2.csv')
```

```
# Plotting function
def plot_ecg(time, amplitude, title):
    plt.figure(figsize=(12, 4))
    plt.plot(time, amplitude, linewidth=0.5)
    plt.title(title)
    plt.xlabel('Time (seconds)')
    plt.ylabel('Amplitude')
    plt.grid(True)
    plt.show()

# Filter design
def butter_bandpass(lowcut, highcut, fs, order=5):
    nyq = 0.5 * fs
    low = lowcut / nyq
    high = highcut / nyq
    b, a = butter(order, [low, high], btype='band')
    return b, a

def butter_bandpass_filter(data, lowcut, highcut, fs, order=5):
    b, a = butter_bandpass(lowcut, highcut, fs, order=order)
    y = filtfilt(b, a, data)
    return y

# Filter parameters
lowcut = 0.5
highcut = 40.0
fs = 360  # Sampling frequency

# Apply bandpass filter
ecg1_filtered = butter_bandpass_filter(ecg1['amplitude (mv)'], lowcut, highcut, fs)
ecg2_filtered = butter_bandpass_filter(ecg2['amplitude (mv)'], lowcut, highcut, fs)

# Plot the filtered signals
plot_ecg(ecg1['temps (s)'], ecg1_filtered, 'Filtered ECG 1 Signal')
plot_ecg(ecg2['temps (s)'], ecg2_filtered, 'Filtered ECG 2 Signal')
```

6.4 The high-pass and low-pass filter

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.signal import freqz

def load_and_clean_data(filepath):
    data = pd.read_csv(filepath, skiprows=1)
    data['temps (s)'] = data['temps (s)'].str.replace('  ', '.').astype(float)
    data['amplitude (mv)'] = pd.to_numeric(data['amplitude (mv)'], errors='coerce')
    data['amplitude (mv)'].fillna(method='ffill', inplace=True)  # Handle NaNs
    return data

ecg1 = load_and_clean_data('/content/ECG1.csv')
ecg2 = load_and_clean_data('/content/ECG2.csv')

# Coefficients for High-Pass Filter
```

```
b_hp = [-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, 1/32]  # Numerator coefficients
a_hp = [1, -1]               # Denominator coefficients

#Coefficients for Low-Pass Filter
b_lp = [1, 0, 0, 0, 0, 0, -2, 0, 0, 0, 0, 0, 1]   # Numerator
a_lp = [1, -2, 1]                                 # Denominator

def plot_frequency_response(b, a, title):
    w, h = freqz(b, a, worN=8000)
    plt.figure(figsize=(12, 6))

    # Magnitude response
    plt.subplot(2, 1, 1)
    plt.plot(0.5 * 360 * w / np.pi, np.abs(h), 'b')
    plt.title(f'{title} frequency response')
    plt.xlabel('Frequency (Hz)')
    plt.ylabel('Gain')
    plt.grid()

    # Phase response
    plt.subplot(2, 1, 2)
    plt.plot(0.5 * 360 * w / np.pi, np.angle(h), 'r')
    plt.xlabel('Frequency (Hz)')
    plt.ylabel('Phase (radians)')
    plt.grid()
    plt.show()

plot_frequency_response(b_hp, a_hp, 'High-pass filter')
plot_frequency_response(b_lp, a_lp, 'Low-pass filter')

# FIR or IIR
filter_type_hp = "IIR" if len(a_hp) > 1 else "FIR"
filter_type_lp = "IIR" if len(a_lp) > 1 else "FIR"
print(f"The high-pass filter is {filter_type_hp}.")
print(f"The low-pass filter is {filter_type_lp}.")

#  High-Pass Filter for ECG Signal
def apply_high_pass_filter(data, b, a):
    filtered_data = np.zeros_like(data)
    for n in range(len(data)):
        if n >= 32:
            filtered_data[n] = (filtered_data[n-1] - (1/32)*data[n] + data[n-16] - data[n
                -17] + (1/32)*data[n-32])
        elif n >= 17:
            filtered_data[n] = (filtered_data[n-1] - (1/32)*data[n] + data[n-16] - data[n
                -17])
        elif n >= 16:
            filtered_data[n] = (filtered_data[n-1] - (1/32)*data[n] + data[n-16])
        else:
            filtered_data[n] = filtered_data[n-1] - (1/32)*data[n] if n > 0 else data[n]
    return filtered_data

#  Low-Pass Filter for ECG Signal
def apply_low_pass_filter(data, b, a):
    filtered_data = np.zeros_like(data)
```

```
    for n in range(len(data)):
        if n >= 12:
            filtered_data[n] = 2*filtered_data[n-1] - filtered_data[n-2] + data[n] - 2*
                data[n-6] + data[n-12]
        elif n >= 6:
            filtered_data[n] = 2*filtered_data[n-1] - filtered_data[n-2] + data[n] - 2*
                data[n-6]
        elif n >= 2:
            filtered_data[n] = 2*filtered_data[n-1] - filtered_data[n-2] + data[n]
        elif n >= 1:
            filtered_data[n] = 2*filtered_data[n-1] + data[n]
        else:
            filtered_data[n] = data[n]
    return filtered_data

# Apply the filters
ecg1_hp_filtered = apply_high_pass_filter(ecg1['amplitude (mv)'].values, b_hp, a_hp)
ecg2_hp_filtered = apply_high_pass_filter(ecg2['amplitude (mv)'].values, b_hp, a_hp)

ecg1_lp_filtered = apply_low_pass_filter(ecg1['amplitude (mv)'].values, b_lp, a_lp)
ecg2_lp_filtered = apply_low_pass_filter(ecg2['amplitude (mv)'].values, b_lp, a_lp)

def plot_ecg(time, amplitude, title):
    plt.figure(figsize=(12, 4))
    plt.plot(time, amplitude, linewidth=0.5)
    plt.title(title)
    plt.xlabel('Time (seconds)')
    plt.ylabel('Amplitude')
    plt.grid(True)
    plt.show()

plot_ecg(ecg1['temps (s)'], ecg1_hp_filtered, 'Filtered ECG 1 Signal (High-Pass)')
plot_ecg(ecg2['temps (s)'], ecg2_hp_filtered, 'Filtered ECG 2 Signal (High-Pass)')

plot_ecg(ecg1['temps (s)'], ecg1_lp_filtered, 'Filtered ECG 1 Signal (Low-Pass)')
plot_ecg(ecg2['temps (s)'], ecg2_lp_filtered, 'Filtered ECG 2 Signal (Low-Pass)')
```

6.5 How to partially resolve the problems occurring at the start of the signal?

```
    import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.signal import freqz, lfilter

# Function to load data and clean it
def load_and_clean_data(filepath):
    data = pd.read_csv(filepath, skiprows=1)
    data['temps (s)'] = data['temps (s)'].str.replace('  ', '.').astype(float)
    data['amplitude (mv)'] = pd.to_numeric(data['amplitude (mv)'], errors='coerce')
    data['amplitude (mv)'].fillna(method='ffill', inplace=True)  # Handle NaNs
    return data

# Load ECG data
ecg1 = load_and_clean_data('/content/ECG1.csv')
ecg2 = load_and_clean_data('/content/ECG2.csv')
```

```
# High-Pass Filter Coefficients
b_hp = [-1/32, 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]  # Numerator coefficients
a_hp = [1, -1]               # Denominator coefficients

# Low-Pass Filter Coefficients
b_lp = [1, 0, 0, 0, 0, 0, -2, 0, 0, 0, 0, 0, 1]   # Numerator coefficients
a_lp = [1, -2, 1]                                 # Denominator coefficients

# Function to plot frequency response
def plot_frequency_response(b, a, title):
    w, h = freqz(b, a, worN=8000)
    plt.figure(figsize=(12, 6))

    # Magnitude response
    plt.subplot(2, 1, 1)
    plt.plot(0.5 * 360 * w / np.pi, np.abs(h), 'b')
    plt.title(f'{title} frequency response')
    plt.xlabel('Frequency (Hz)')
    plt.ylabel('Gain')
    plt.grid()

    # Phase response
    plt.subplot(2, 1, 2)
    plt.plot(0.5 * 360 * w / np.pi, np.angle(h), 'r')
    plt.xlabel('Frequency (Hz)')
    plt.ylabel('Phase (radians)')
    plt.grid()
    plt.show()

# Plot the frequency response for both filters
plot_frequency_response(b_hp, a_hp, 'High-pass filter')
plot_frequency_response(b_lp, a_lp, 'Low-pass filter')

# Determine if it is FIR or IIR
filter_type_hp = "IIR" if len(a_hp) > 1 else "FIR"
filter_type_lp = "IIR" if len(a_lp) > 1 else "FIR"
print(f"The high-pass filter is {filter_type_hp}.")
print(f"The low-pass filter is {filter_type_lp}.")

# Apply High-Pass Filter using lfilter
def apply_high_pass_filter(data, b, a):
    return lfilter(b, a, data)


# Apply Low-Pass Filter using lfilter
def apply_low_pass_filter(data, b, a):
    return lfilter(b, a, data)

# Apply the filters to the ECG data using lfilter
ecg1_hp_filtered = apply_high_pass_filter(ecg1['amplitude (mv)'].values, b_hp, a_hp)
ecg2_hp_filtered = apply_high_pass_filter(ecg2['amplitude (mv)'].values, b_hp, a_hp)

print(f"Number of samples in original ECG1: {len(ecg1)}")
print(f"Number of samples in original ECG2: {len(ecg2)}")
```

```
print(f"Number of samples in high-pass filtered ECG1: {len(ecg1_hp_filtered)}")
print(f"Number of samples in high-pass filtered ECG2: {len(ecg2_hp_filtered)}")

ecg1_lp_filtered = apply_low_pass_filter(ecg1['amplitude (mv)'].values, b_lp, a_lp)
ecg2_lp_filtered = apply_low_pass_filter(ecg2['amplitude (mv)'].values, b_lp, a_lp)

# Plot the filtered signals
def plot_ecg(time, amplitude, title):
    plt.figure(figsize=(12, 4))
    plt.plot(time, amplitude, linewidth=0.5)
    plt.title(title)
    plt.xlabel('Time (seconds)')
    plt.ylabel('Amplitude')
    plt.grid(True)
    plt.show()

# Plotting the results
plot_ecg(ecg1['temps (s)'], ecg1_hp_filtered, 'Filtered ECG 1 Signal (High-Pass)')
plot_ecg(ecg2['temps (s)'], ecg2_hp_filtered, 'Filtered ECG 2 Signal (High-Pass)')

plot_ecg(ecg1['temps (s)'], ecg1_lp_filtered, 'Filtered ECG 1 Signal (Low-Pass)')
plot_ecg(ecg2['temps (s)'], ecg2_lp_filtered, 'Filtered ECG 2 Signal (Low-Pass)')
```

6.6 Using the output of the high pass filter as the input of the low pass filter and display the result

```
#  High-Pass Filter first, then Low-Pass Filter
ecg1_hp_lp_filtered = apply_low_pass_filter(ecg1_hp_filtered, b_lp, a_lp)
ecg2_hp_lp_filtered = apply_low_pass_filter(ecg2_hp_filtered, b_lp, a_lp)

#  Low-Pass Filter first, then High-Pass Filter
ecg1_lp_hp_filtered = apply_high_pass_filter(ecg1_lp_filtered, b_hp, a_hp)
ecg2_lp_hp_filtered = apply_high_pass_filter(ecg2_lp_filtered, b_hp, a_hp)

plot_ecg(ecg1['temps (s)'], ecg1_hp_lp_filtered, 'ECG 1 Filtered Signal (High-Pass then
    Low-Pass)')
plot_ecg(ecg2['temps (s)'], ecg2_hp_lp_filtered, 'ECG 2 Filtered Signal (High-Pass then
    Low-Pass)')

plot_ecg(ecg1['temps (s)'], ecg1_lp_hp_filtered, 'ECG 1 Filtered Signal (Low-Pass then
    High-Pass)')
plot_ecg(ecg2['temps (s)'], ecg2_lp_hp_filtered, 'ECG 2 Filtered Signal (Low-Pass then
    High-Pass)')
```

6.7 Bonus

```
    import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

def load_and_clean_data(filepath):
    data = pd.read_csv(filepath, skiprows=1)
    data['temps (s)'] = data['temps (s)'].str.replace('  ', '.').astype(float)
```

```
    data['amplitude (mv)'] = pd.to_numeric(data['amplitude (mv)'], errors='coerce')
    data['amplitude (mv)'].fillna(method='ffill', inplace=True)  # Handle NaN (undefined
        values)
    return data

ecg1 = load_and_clean_data('/content/ECG1.csv')
ecg2 = load_and_clean_data('/content/ECG2.csv')

# Construct LMS Adaptive Filter Function to the ECG
def lms_adaptive_filter(desired, input_signal, mu, filter_order):
    n = len(input_signal)
    y = np.zeros(n)
    e = np.zeros(n)
    w = np.zeros(filter_order)
    epsilon = 1e-6  # Small constant to avoid division by zero

    for i in range(filter_order, n):
        x = input_signal[i:i-filter_order:-1]
        y[i] = np.dot(w, x)
        e[i] = desired[i] - y[i]
        w = w + 2 * mu * e[i] * x / (np.dot(x, x) + epsilon)

    return y, e

# Parameters used
mu = 0.01  # Step size
filter_order = 32

# Apply LMS Adaptive Filter
ecg1_filtered, ecg1_error = lms_adaptive_filter(ecg1['amplitude (mv)'].values, ecg1['
    amplitude (mv)'].values, mu, filter_order)
ecg2_filtered, ecg2_error = lms_adaptive_filter(ecg2['amplitude (mv)'].values, ecg2['
    amplitude (mv)'].values, mu, filter_order)

def plot_ecg(time, amplitude, title):
    plt.figure(figsize=(12, 4))
    plt.plot(time, amplitude, linewidth=0.5)
    plt.title(title)
    plt.xlabel('Time (seconds)')
    plt.ylabel('Amplitude')
    plt.grid(True)
    plt.show()

plot_ecg(ecg1['temps (s)'], ecg1_filtered, 'Filtered ECG 1 Signal (LMS Adaptive Filter)')
plot_ecg(ecg2['temps (s)'], ecg2_filtered, 'Filtered ECG 2 Signal (LMS Adaptive Filter)')
```

References

[1] Wikipedia contributors. *Frequency response*. Accessed: 2024-06-01. 2023. URL: https://en.wikipedia.org/wiki/Frequency_response.

[2] Wikipedia contributors. *High-pass filter*. Accessed: 2024-06-01. 2023. URL: https://en.wikipedia.org/wiki/High-pass_filter.

[3] Wikipedia contributors. *Low-pass filter*. Accessed: 2024-06-01. 2023. URL: https://en.wikipedia.org/wiki/Low-pass_filter.