Performance Analysis of Different Filtering Techniques in Biomedical Signal Processing

Batuhan Ekmekçioğlu (280206020), Harun Durmuş (270206025)

Abstract*—This paper provides a comprehensive review of several filtering strategies used in biomedical signal processing, with a focus on electrocardiogram (ECG) signals. The MATLAB-based examination comprises FIR filtering, IIR filtering (Butterworth, Chebyshev Type I and II), Wavelet Transform, Wiener filtering, and adaptive filtering with the Least Mean Square algorithm. Each filter's performance is evaluated based on its ability to reduce noise and enhance signals, providing insight into its practical applications in biomedical engineering.

I. INTRODUCTION

Biomedical signal processing is essential in modern healthcare, notably for the analysis and interpretation of ECG signals. ECG signals are frequently distorted by noise and distortions due to other muscle parts or organs, necessitating the employment of multiple filtering algorithms for reliable diagnosis. In this study, we investigate many filtering methods, including FIR, IIR (Butterworth, Chebyshev Types I and II), Wavelet Transform, Wiener Filter, and Adaptive Filtering with the LMS algorithm. Each technique has distinct advantages and challenges in processing biomedical information, which we shall go over in depth.

A. *Finite Impulse Response (FIR) Filtering)*

FIR filtering is a fundamental approach in digital signal processing that is distinguished by its finite-duration impulse responses. This means that the filter responds to an impulse input for a finite number of samples before settling to zero. FIR filters are intrinsically stable and have a linear phase response, making them ideal for a wide range of applications, including biomedical signal processing.

The output *y[n]* of an FIR filter is the weighted sum of the current and a finite number of previous input values *x[n]*. Mathematically, it can be expressed as:

$$y[n] = \sum_{i=0}^{M-1} b[i].x[n-i] \qquad (1)$$

Where:

- *y[n]* is the output signal at time *n*.
- *x[n]* is the input signal at time *n*.
- *b[i]* are the filter coefficients.
- *M* is the order of the filter, indicating the number of coefficients.

10-Point Moving Average Filter

A moving average filter is a simple FIR filter that is widely used to smoothen the data. It calculates the average of a certain number of input points and uses it as the output. A 10-point moving average filter calculates the average of ten consecutive input samples. This functions as a low-pass filter, enabling only the slower-varying components of the signal to pass through while effectively smoothing out high-frequency variations. The mathematical representation of a 10-point moving average filter is:

$$y[n] = \frac{1}{10} \sum_{i=0}^{9} x[n-i] \qquad (2)$$

In this equation, each output sample *y[n]* is the average of the current and the previous 9 input samples *x[n], x[n-1], …, x[n-9]*. The coefficients *b[i]* of this filter are all equal to $\frac{1}{10}$.

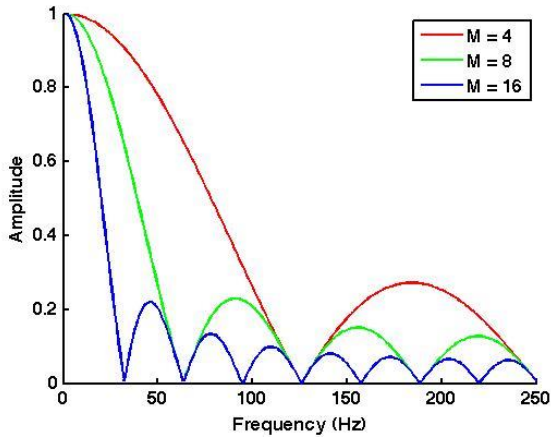

Figure 1: *Moving Average Filters with different M values[1]*

Advantages of FIR Filters

- ***Stability:*** Because FIR filters lack feedback (recursive component), they are naturally stable.
- ***Linear Phase Response:*** FIR filters can be designed with a linear phase response, which ensures that the input signal is not phase-distorted.
- ***Flexibility in Design:*** FIR filters can be tailored to suit frequency response requirements.

- ***Finite Duration:*** The impulse response has a defined period, making it easier to analyze and apply.

Applications in Biomedical Signal Processing

In biomedical signal processing, such as ECG signal analysis, FIR filters are employed to reduce noise and eliminate baseline wandering while avoiding phase distortion. This preserves waveform forms, which are essential for accurate diagnosis and analysis.

Drawbacks of FIR Filtering

While Finite Impulse Response (FIR) filters have various benefits in digital signal processing, they also have several downsides that must be considered:

- ***Computational Complexity:*** FIR filters, particularly ones with a high order (large number of coefficients), necessitate additional processing resources. This additional complexity may be a constraint in real-time applications or systems with limited computing capability.
- ***Longer Impulse Response:*** To achieve a sharp cutoff in the frequency response or to effectively mimic certain frequency responses, FIR filters frequently require a higher filter order. This produces a prolonged impulse response, which might cause substantial delay in the processed signal. In real-time applications, such as live ECG monitoring, this latency may be unacceptable.
- ***Increased Memory Requirement:*** FIR filters must store previous input values (coefficients) for processing. Higher order filters with more coefficients demand more memory, which might be a bottleneck in systems with limited memory capacity.
- ***Lower Efficiency for Certain Applications:*** When compared to Infinite Impulse Response (IIR) filters, FIR filters might be less efficient, especially in applications that need sharp frequency roll-off. IIR filters can achieve comparable or greater performance with a lower order, resulting in fewer computations.
- ***Design Complexity:*** developing FIR filters to fulfill precise frequency response criteria, particularly when a linear phase is not required, can be more difficult and computationally intensive than developing equivalent IIR filters. The design process frequently incorporates techniques like as windowing and the use of optimization tools, which increase the complexity.
- ***Energy Dissipation:*** For applications where energy efficiency is critical, such as battery-powered biomedical devices, the higher computing demand of FIR filters may result in faster battery depletion.

Despite these limitations, FIR filters are widely utilized in a variety of applications, including biomedical signal processing, due to their inherent stability and linear phase response. Their application is especially useful where the phase of the signal is critical, as in ECG signal analysis. In reality, the decision between FIR and IIR filters is often made by balancing these trade-offs against the application's specific requirements.

In conclusion, FIR filtering, particularly approaches such as the 10-point moving average filter, plays an important role in digital signal processing due to its stability, linear phase properties, and ease of implementation, making it an ideal choice for processing biomedical signals such as ECG.

B. Infinite Impulse Response (IIR) Filtering

IIR filters are a fundamental class of digital filters with an infinite impulse response. They are distinguished by recursive elements, which means that the output is dependent on both past inputs and outcomes. IIR filters are frequently favored over FIR filters because they are more efficient at achieving crisp frequency response characteristics with a lower filter order.

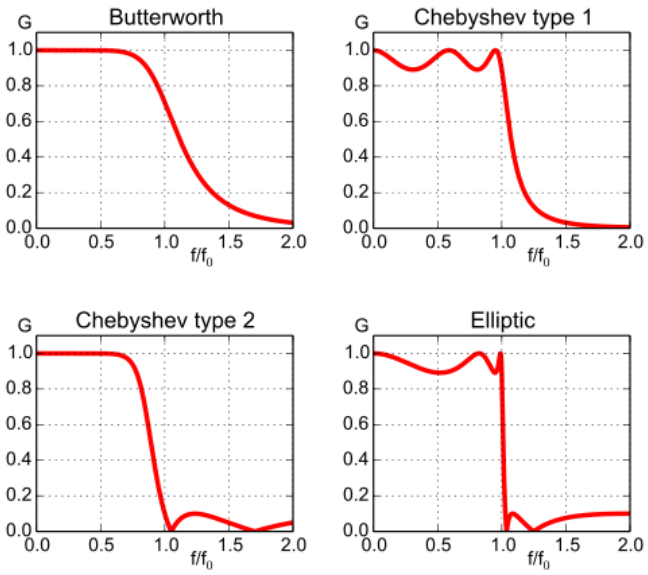

Figure 2: *Different Types of IIR Filtering Techniques*[2]

Types of IIR Filters

Butterworth Filter: The Butterworth filter has a maximum flat response in the passband, with no ripples and a smooth roll-off to the stopband. Mathematical expression of the transfer function of an *N*th order Butterworth filter is given by:

$$H(s) = \frac{1}{1+(\frac{s}{wc})^{2n}} \qquad (3)$$

where, w_C is the cutoff frequency and s is the complex frequency variable.

Chebyshev Type I Filter: This filter enables ripples in the passband, but has a stronger roll-off than the Butterworth filter. It provides a speedier transition from passband to stopband. The transfer function is given by:

$$H(s) = \frac{1}{1+\epsilon^2 Tn^2(\frac{s}{wc})} \qquad (4)$$

where, T_N is the Chebyshev polynomial of the *N*th order, $\in$ is the ripple factor and w_C is the cutoff frequency.

Chebyshev Type II Filter: Chebyshev Type II filters enable ripples in the stopband rather than the passband, resulting in an inverse Chebyshev response. Its transfer function is more complex and is defined in terms of inverse Chebyshev polynomials.

Advantages Over FIR Filters

- ***Efficiency:*** IIR filters can attain the desired frequency response characteristic at a considerably lower order than FIR filters, making them computationally more efficient.
- ***Sharper Cutoff:*** IIR filters can provide sharper cutoffs in the frequency response, which is critical in applications that need rigorous separation of frequency bands.
- ***Flexibility in design:*** The many types of IIR filters (Butterworth, Chebyshev I, and II) allow you to construct filters depending on specific needs such as ripple tolerance and roll-off sharpness.

Drawbacks of IIR Filters

- ***Stability Issues:*** Because of their recursive nature, IIR filters can be unstable if not carefully built, particularly for higher-order filters.
- ***Phase Distortion:*** Because IIR filters do not have a linear phase response, the input signal is phase distorted, which can be troublesome in applications that require phase information.
- ***Complexity in Analysis:*** IIR filters are more hard to analyze and understand due to their recursive nature, as opposed to FIR filters. This intricacy is especially noticeable in stability analysis and understanding the system's behavior at various frequencies.
- ***Sensitivity to Coefficient Quantization:*** Feedback makes IIR filters more sensitive to coefficient quantization. This can cause severe performance reduction, particularly in fixed-point arithmetic contexts like low-power microcontrollers and digital signal processors.
- ***Non-linear Phase Response:*** Unlike FIR filters, which may be easily engineered to have a linear phase response, IIR filters are intrinsically non-linear. This can cause distortion in signals with significant phase correlations, such as audio and communication signals.
- ***Difficulty in Multi-rate Applications:*** In applications where the signal's sampling rate changes (such as decimation or interpolation), IIR filters require more sophisticated structures and careful design to maintain stability and desired performance.

Despite these disadvantages, IIR filters are commonly utilized in signal processing applications due to their efficiency in producing sharp frequency responses with fewer coefficients. The decision between IIR and FIR filters is frequently based on balancing computational efficiency with phase linearity and stability requirements. In many real scenarios, especially in resource-constrained contexts, IIR filters are a better alternative due to their lower processing requirements and memory utilization.

C. Wavelet Transform

The Wavelet Transform is a sophisticated mathematical tool used in signal processing to analyse complex data. Unlike the standard Fourier Transform, which divides a signal into sinusoidal components, the Wavelet Transform divides a signal into wavelets, which are small, localized waves in both time and frequency. The Continuous Wavelet Transform (CWT) of a signal *x(t)* is defined as:

$$Wx(a,b) = \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} x(t).\Psi^*(\frac{t-b}{a})dt \qquad (5)$$

Where:

- $W_X(a,b)$ is the wavelet coefficient,
- $\Psi(t)$ is the mother wavelet,
- a is the scale parameter,
- b is the transition parameter,
- Ψ^* denotes the complex conjugate of Ψ.

The Discrete Wavelet Transform (DWT), which is more often employed in digital signal processing, analyzes distinct frequency bands by passing the signal through a succession of high-pass and low-pass filters.

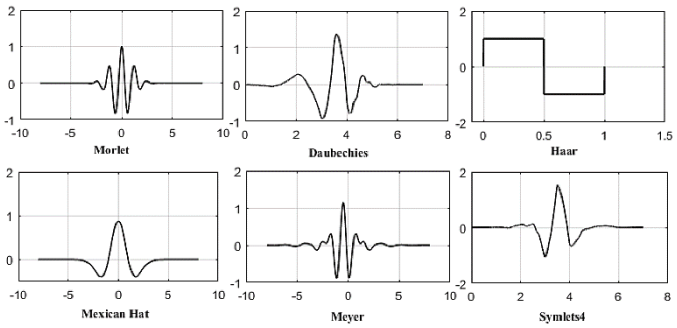

Figure 3: *Different Wavelet Transform Responses*[3]

Advantages over FIR and IIR Filters

- ***Time-Frequency Localization:*** Wavelets perform superior time-frequency localization than FIR and IIR filters. They can adjust their time and frequency resolutions.
- ***Handling Non-Stationary Signals****:* Wavelet transformations are very useful for studying non-stationary signals in which frequency components change over time, such as ECG.
- ***Multi-Resolution Analysis:*** Wavelets support multi-resolution analysis, which allows for the investigation of a signal at different degrees of detail.
- ***Sparse Representation:*** In many circumstances, wavelets can give a sparse representation of signals, resulting in effective data compression and noise reduction.

Drawbacks of Wavelet Transform

- ***Complexity:*** Wavelet transforms are more hard to build and understand than FIR or IIR filters.
- ***Wavelet Selection:*** Choosing an appropriate wavelet function is critical and can be difficult, as the wavelet transform's performance is greatly influenced by the mother wavelet.
- ***Border Effects:*** Processing finite-length signals in discrete wavelet transform can result in border effects, in which the wavelet coefficients at the beginning and end of the signal are inaccurate.
- ***Computational Load:*** Despite their computing efficiency, wavelet transforms can be more demanding than basic FIR or IIR filters, particularly in real-time applications.
- ***Lack of Standardization:*** There is a lack of uniformity in wavelet selection, which can lead to inconsistencies in analysis and interpretation.

To summarize, the wavelet transform is a versatile and powerful signal processing tool that provides advantages for examining non-stationary or time-varying frequency sources. However, its complexity and the need for precise wavelet selection make it more difficult to implement than typical FIR and IIR filters.

D. Wiener Filtering

Wiener filtering is an optimum filtering technique that minimizes the mean square error between the estimated and genuine signals. It performs particularly well in signal restoration and noise reduction scenarios where the statistical features of the signal and noise are known or can be inferred. The Wiener filter for a discrete system can be represented as:

$$H(w) = \frac{Pxd(w)}{Pxx(w)} \qquad (6)$$

Where:

- $H(w)$ is the Wiener filter in the frequency domain.
- $P_{xd}(w)$ is the cross-power spectral density of the input signal x and the desired signal d.
- $P_{xx}(w)$ is the power spectral density of the input signal.

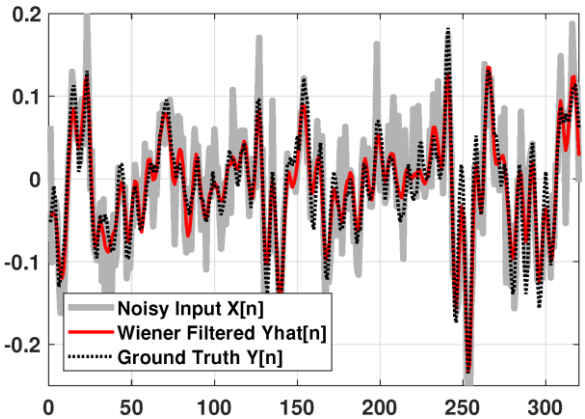

Figure 4: *Expected effect by using Wiener filter*[4]

Advantages Over Other Filtering Techniques

- ***Optimal in the Minimum Mean Square Error Sense:*** The Wiener filter is designed to reduce the total mean square error in estimating the desired signal from noisy observations, which gives it a significant advantage over other types of filters.
- ***Effective in Stationary Noise Environments:*** It performs very well in situations where noise is stationary and its statistical features are known or estimated.
- ***Signal and Noise Characteristics:*** Unlike many other filters, which solely consider signal characteristics, the Wiener filter takes into consideration noise characteristics as well.

Drawbacks

- ***Requirement for Statistical Knowledge:*** Its success is largely dependent on prior knowledge of the statistical features of the signal and noise, which may not always be available or precisely calculated.
- ***Non-Stationary Signal Limitation:*** Wiener filtering is not suitable for non-stationary signals in which the statistical features of the signal or noise fluctuate with time.
- ***Computational Complexity:*** Using a Wiener filter can be computationally costly, particularly for high-dimensional data.

E. Adaptive Filtering

Adaptive filters actively modify their coefficients to reflect the changing statistical features of the input signal. The Least Mean Squares (LMS) algorithm is a popular adaptive filtering technique. The LMS algorithm updates the filter coefficients as follows:

$$w_{n+1} = w_n + \mu \,.\, e_n + x_n \qquad (7)$$

Where:

- w_n is the coefficient vector at time n.
- μ is the step size or learning rate.
- e_n is the error signal, which is the difference between the desired signal and the filter output.
- x_n is the input signal vector.

Advantages Over Other Filtering Techniques

- ***Adaptability:*** The fundamental benefit of adaptive filters is their ability to adjust to changing signal properties, making them appropriate for non-stationary situations.
- ***No Prior Knowledge Required:*** Unlike Wiener filters, adaptive filters do not require prior knowledge of signal or noise properties.
- ***Real-time Signal Processing:*** They can process signals in real time and adjust to changes as they occur.

Drawbacks

- ***Convergence Issues:*** The choice of step size (μ) affects convergence speed and stability. Improper selection may result in sluggish convergence or instability.
- ***Computational Load:*** Although adaptive filters are not as computationally costly as Wiener filters, they nevertheless require significant computer resources, especially for high-dimensional data.
- ***Sensitivity to Parameter Settings:*** The performance of adaptive filters can be affected by parameters such as filter length and step size.

To summarize everything that have been stated so far for the Wiener and Adaptive Filtering methods, the Wiener filter performs well in contexts where the statistical features of the signal and noise are known or can be predicted. Its strength stems from its capacity to minimize the mean square error between the estimated and genuine signals. This makes it very useful for noise reduction and signal restoration in stationary noise settings. However, its reliance on prior statistical knowledge and poor performance in non-stationary signal situations are key drawbacks. Furthermore, the computational complexity of Wiener filtering might be a bottleneck in real-time or resource-constrained applications.

Adaptive filters, particularly those based on the LMS algorithm, stand out for their ability to dynamically adjust to changing signal characteristics. This versatility makes them appropriate for real-time applications and situations with non-stationary or unpredictable signal and noise characteristics. They do not require prior knowledge of signal or noise qualities, making them more flexible than Wiener filters. However, adaptive filters are sensitive to parameter settings such as step size and filter length, which can affect convergence speed and stability.

In conclusion, the choice between Wiener and adaptive filtering should be guided by the application's specific requirements, taking into account factors such as signal stationarity, prior knowledge about the signal and noise, computational constraints, and the need for real-time processing. Adaptive filters are commonly selected for their flexibility and adaptation in dynamic contexts, whereas Wiener filters are recommended for their superior performance in stationary noise scenarios.

II. MATLAB DEMONSTRATION

The demonstration of the filtering methods and the resultant figures are stated in this chapter. Firstly, 10-Point Moving Average Filter is applied to the noisy ECG signals.

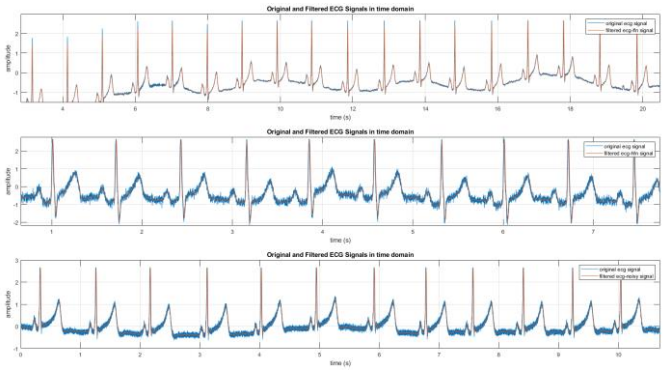

Figure 5: *10-Point Moving Average Filter*[5]

A 10-point moving average filter used to ECG readings can greatly reduce noise and smooth the signal, making it easier to understand. However, several ECG patterns might be diminished or shifted over time, so interpret the data with caution. The noise reduction can be observed from the frequency spectrum.

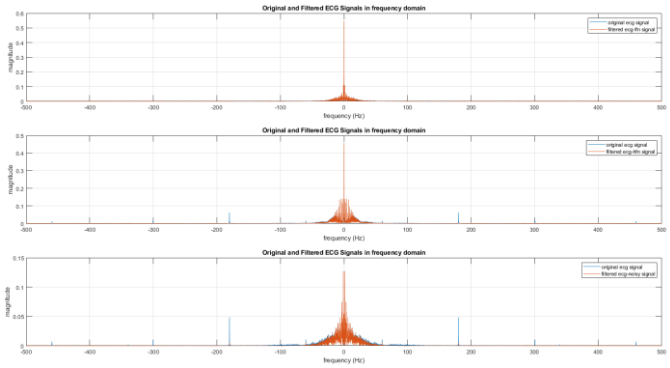

Figure 6: *Frequency Spectrum of FIR filtering*

It is observed that the unwanted components are cancelled after filtering.

Secondly, by determining the order and filter parameters which are requires for implementation, Butterworth, Chebyshev Type I and II IIR filters are demonstrated.

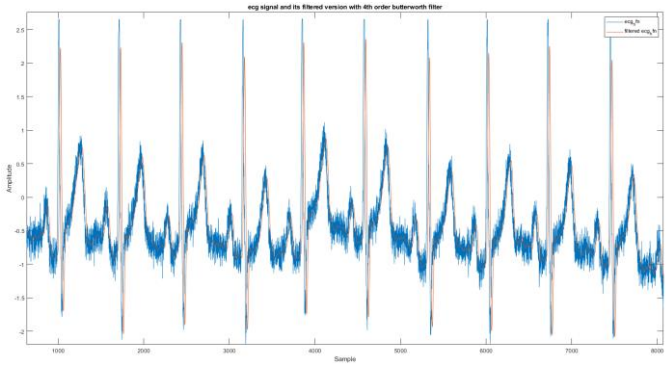

Figure 7: *IIR (Butterworth) Filter Result*

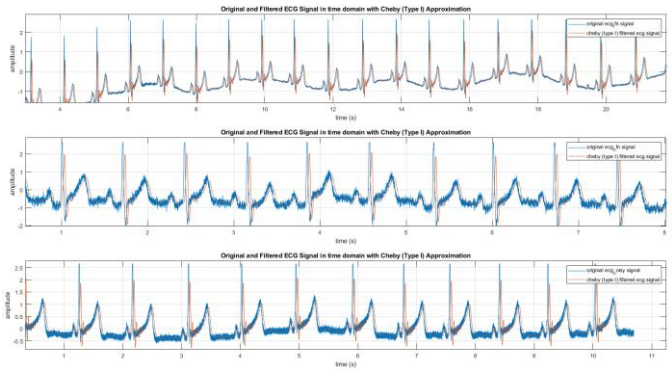

Figure 8: *IIR (Chebyshev Type I) Filter Result*

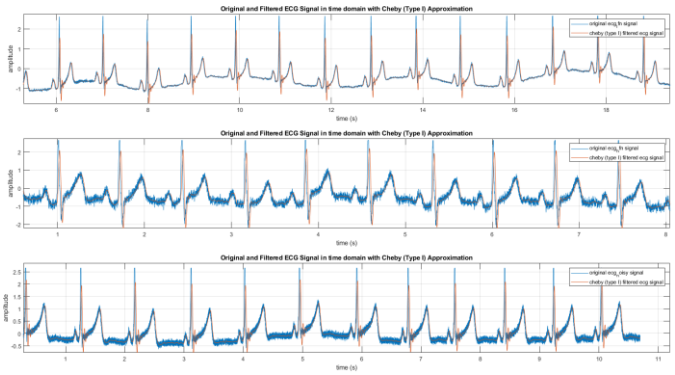

Figure 9: *IIR (Chebyshev Type II) Filter Result*

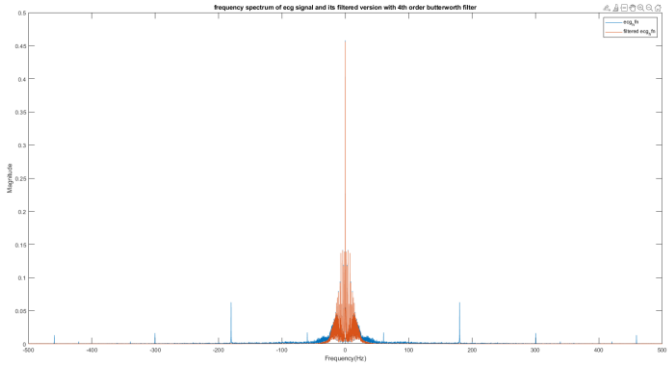

Figure 10: *Frequency Spectrum of Butterworth Result*

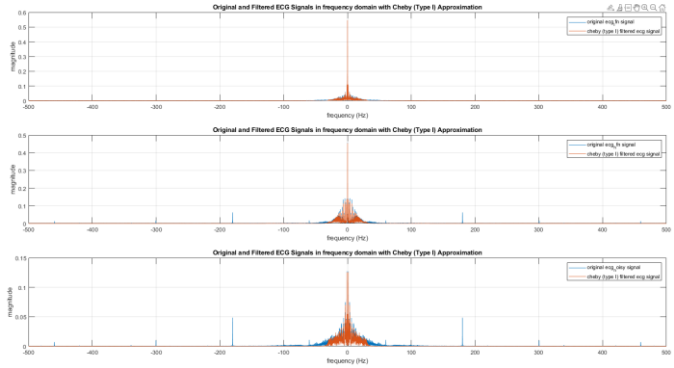

Figure 11: *Frequency Spectrum of Chebyshev Type I Result*

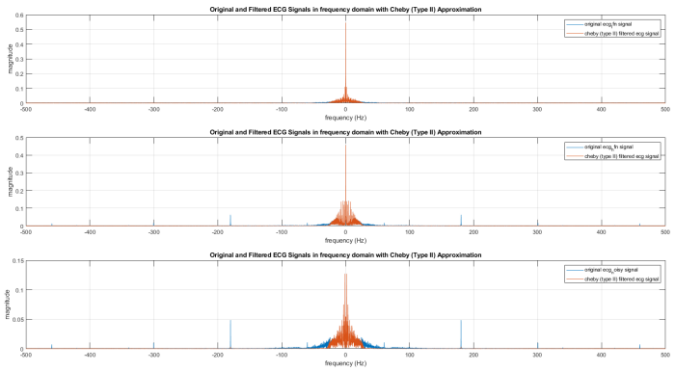

Figure 12: *Frequency Spectrum of Chebyshev Type II Result*

It is observed that, IIR filters which are Butterworth, Chebyshev Type I and II have better performance compared to M-Point Moving Average Filtering. But there is still problem about saving the information in QRS intervals.

Applying Wavelet Transformation methods that are Daubechies (db4) and Symbolic (sym4) into the ECG signals and their results are visualized below.

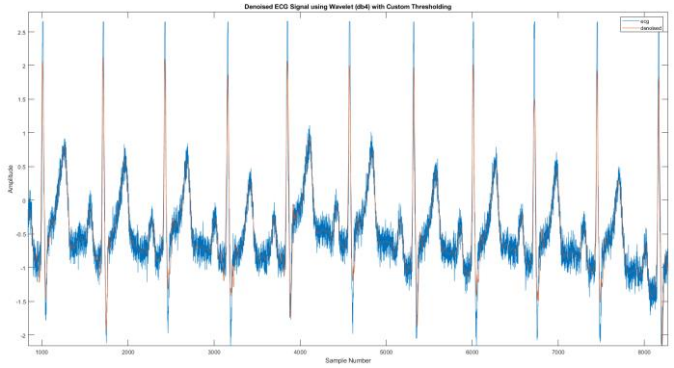

Figure 13: *Wavelet Transformation (db4)*

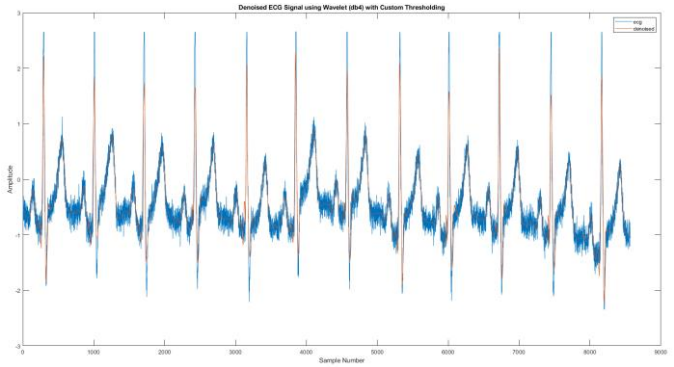

Figure 14: *Wavelet Transformation (sym4)*

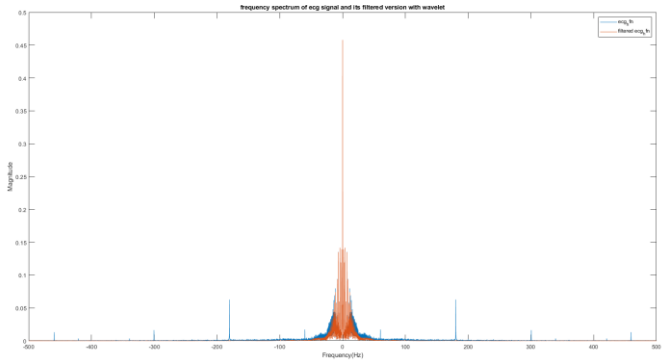

Figure 15: *Wavelet Daubechies Frequency Spectrum*

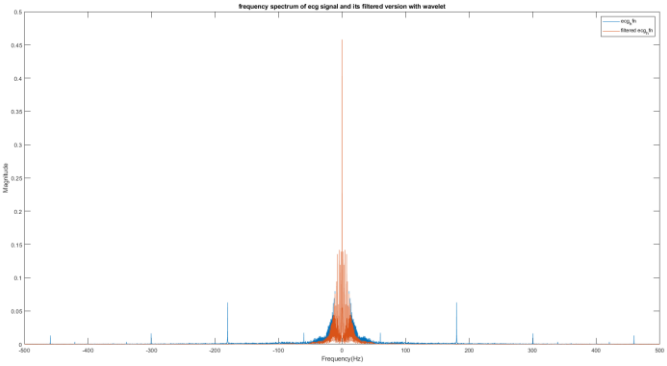

Figure 16: *Wavelet Symbolic Frequency Spectrum*

In the provided code for Wavelet Transform, the operation starts by choosing which wavelet type will be used firstly. In this case, we used "db4" or "sym4" as we visualize above. Then, it is determined that with how many coefficient we are going to represent our filter by choosing the level number. The built-in function *wavedec(.)* creates the coefficients for corresponding level and their length.

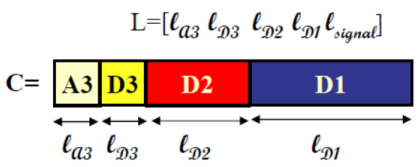

Figure 17: *Construction of coefficient vector with respect to length informations*

To obtain coefficient vector, this algorithm which is visualized in *Figure 17* is applied. By increasing the level number, coefficient level lengths becomes shorter. The highest level coefficient is placed left-hand side of the coefficient vector and the hierarchy continues similarly. The threshold value fort his coefficient placement process is determined as the formula below:

$$\delta_k = \sigma_k\sqrt{2\log(Nk)} \qquad (8)$$

Where:

- δ_k is threshold for subband *k*,
- σ_k is noise std. deviation for subband *k*,
- N_k is length of the DWT coefficients at level k.

Thresholding is applied for each level. The noise content in the figure is determined as the original signal minus the output of the Butterworth filtered signal (this reference is chosen as it satisfies the conditions). With these threshold values, the parameters that are required for the filtering process such as coefficient vector are created. The built-in function *waverec(.)*, inverse wavelet transform is applied and we are able to obtain the filtered signal.

Wiener and Adaptive Filtering methods are finally demonstrated over the ECG signal examples that we use during the process.

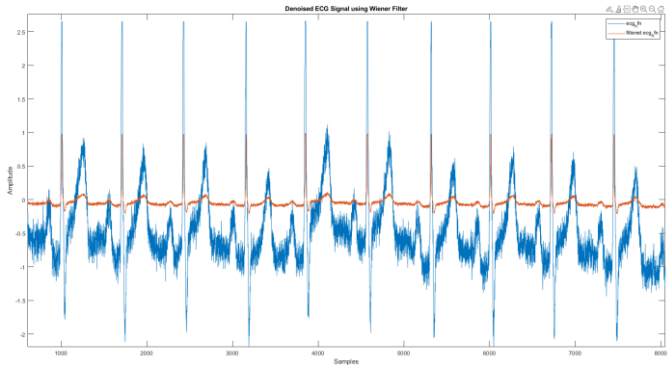

Figure 18: *Wiener Filtering Results*

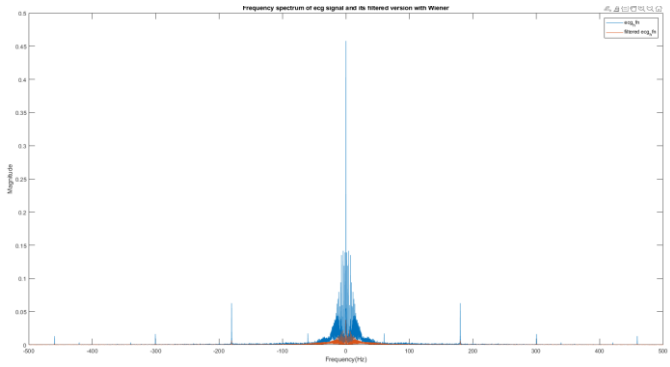

Figure 19: *Wiener Filtering Frequency Spectrum*

It is observed that, power of the filtered signal is reduced greatly by using Wiener filter even the QRS informations are saved. This power loss may cause some errors during QRS detection methods.

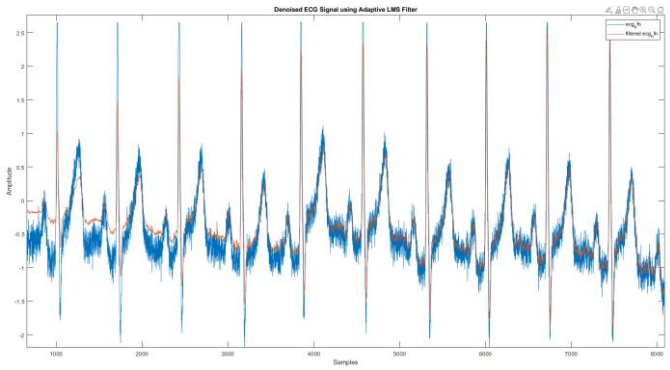

Figure 20*: Adaptive Filtering Results (LMS Algorithm)*

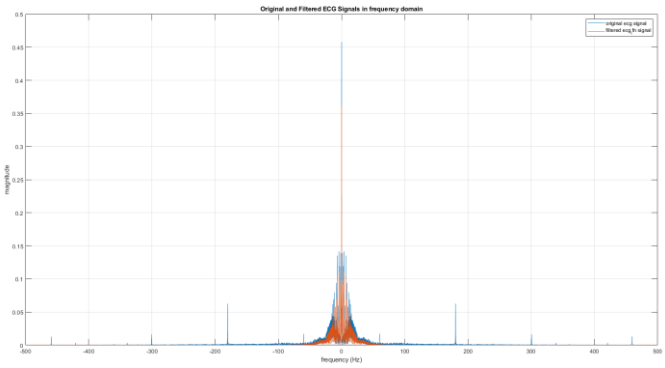

Figure 21: *Adaptive Filtering Frequency Spectrum*

III. RESULTS AND CONCLUSION

REFERENCES

[1] The Scientist and Engineer's Guide to Digital Signal Processing, Chapter 15, Steven W. Smith, Ph.D., 1997.

[2] "How Does An IIR Filter Change Depending On The Approximation Used?", Harry Svenson, Electrical Engineering Stack Exchange, 2018.

[3] "Wavelet Transform Processor Based Surface Acoustic Wave Devices", Hagar A. Ali, 2022.

[4] “Introduction to Probability for Data Science”, Chapter 10, Stanley H. Chan, 2022.

APPENDIX

[5] MATLAB code “main.m”

```
%% **IT IS SUGGESTED THAT RUNNING THE CODE SECTION BY SECTION.**
%% LINE 9: FIR Filtering
%% LINE 115: IIR Filtering (Butterworth)
%% LINE 155: IIR Filtering (Chebyshev Type I)
%% LINE 257: IIR Filtering (Chebyshev Type  II)
%% LINE 363: Wavelet Transform
%% LINE 432: Wiener Filtering
%% LINE 486: Adaptive Filtering
%% FIR Filter
clc;
clear all;
close all;
load('ecg_1.mat');
load('ecg_2.mat');
load('ecg_3.mat');

% Applying a 10-point moving average filter

% time domain

size_window = 10; % Defined window size of the filter
nom = ones(1,size_window); % nominator coefficient of the filter
denom = size_window; % denominator coefficient of the filter
ecg_lfn_filtered = filter(nom, denom, ecg_lfn);
ecg_hfn_filtered = filter(nom, denom, ecg_hfn);
ecg_noisy_filtered = filter(nom, denom, ecg_noisy);
N_lfn = length(ecg_lfn);
N_hfn = length(ecg_hfn);
N_noisy = length(ecg_noisy);
Fs = 1000; % sample frequency
Ts = 1/Fs;
t_lfn = 0:Ts:(N_lfn-1)*Ts;
t_hfn = 0:Ts:(N_hfn-1)*Ts;
t_noisy = 0:Ts:(N_noisy-1)*Ts;

figure;
set(gcf,'color','w');
subplot(311)
plot(t_lfn, ecg_lfn);
hold on;
plot(t_lfn, ecg_lfn_filtered);
grid on;
xlabel("time (s)");
ylabel("amplitude");
title('Original and Filtered ECG Signals in time domain');
legend("original ecg signal","filtered ecg-lfn signal");
xlim([0 N_lfn*Ts]);

subplot(312)
plot(t_hfn, ecg_hfn);
hold on;
plot(t_hfn, ecg_hfn_filtered);
grid on;
xlabel("time (s)");
ylabel("amplitude");
title('Original and Filtered ECG Signals in time domain');
legend("original ecg signal","filtered ecg-hfn signal");
xlim([0 N_hfn*Ts]);

subplot(313)
plot(t_noisy, ecg_noisy);
hold on;
plot(t_noisy, ecg_noisy_filtered);
grid on;
xlabel("time (s)");
ylabel("amplitude");
title('Original and Filtered ECG Signals in time domain');
legend("original ecg signal","filtered ecg-noisy signal");
xlim([0 N_noisy*Ts]);

% frequency domain

ecg_lfn_freq = abs(fftshift(fft(ecg_lfn,N_lfn)))/N_lfn;
ecg_lfn_filtered_freq = abs(fftshift(fft(ecg_lfn_filtered,N_lfn)))/N_lfn;
f_lfn = linspace(-Fs/2,Fs/2,N_lfn);

ecg_hfn_freq = abs(fftshift(fft(ecg_hfn,N_hfn)))/N_hfn;
ecg_hfn_filtered_freq = abs(fftshift(fft(ecg_hfn_filtered,N_hfn)))/N_hfn;
f_hfn = linspace(-Fs/2,Fs/2,N_hfn);

ecg_noisy_freq = abs(fftshift(fft(ecg_noisy,N_noisy)))/N_noisy;
ecg_noisy_filtered_freq =
abs(fftshift(fft(ecg_noisy_filtered,N_noisy)))/N_noisy;
f_noisy = linspace(-Fs/2,Fs/2,N_noisy);

figure;
set(gcf,'color','w');
subplot(311)
plot(f_lfn, ecg_lfn_freq);
hold on;
plot(f_lfn, ecg_lfn_filtered_freq);
grid on;
xlabel("frequency (Hz)");
ylabel("magnitude");
title('Original and Filtered ECG Signals in frequency domain');
legend("original ecg signal","filtered ecg-lfn signal");

subplot(312)
plot(f_hfn, ecg_hfn_freq);
hold on;
plot(f_hfn, ecg_hfn_filtered_freq);
grid on;
xlabel("frequency (Hz)");
ylabel("magnitude");
title('Original and Filtered ECG Signals in frequency domain');
legend("original ecg signal","filtered ecg-hfn signal");

subplot(313)
plot(f_noisy, ecg_noisy_freq);
hold on;
plot(f_noisy, ecg_noisy_filtered_freq);
grid on;
xlabel("frequency (Hz)");
ylabel("magnitude");
title('Original and Filtered ECG Signals in frequency domain');
legend("original ecg signal","filtered ecg-noisy signal");

%% IIR Filter (Butterworth)
clc;
clear all;
close all;
load('ecg_1.mat');
load('ecg_2.mat');
load('ecg_3.mat');

Fs = 1000;
Ts = 1/Fs;
Fc = 50/Fs; %cutoff freq/Fs for normalization. cutoff freq determined by
observing freq spectrum of the ecg signal
N = 4; %filter order
% by increasing order we get a sharper filter freq response supresses freqs
above Fc better.
% but increases phase shift and decreases amplitude more. it also requires
more computational power so order of 4 is good enough
[b,a] = butter(N,Fc,"low");
ecg_butter_filtered = filter(b,a,ecg_hfn);

figure;
set(gcf,'color','w');
plot(ecg_hfn)
hold on
```

```
plot(ecg_butter_filtered)
title("ecg signal and its filtered version with 4th order butterworth
filter")
xlabel("Sample");
ylabel("Amplitude");
legend("ecg_hfn","filtered ecg_hfn")

ecg_hfns=fftshift(abs(fft(ecg_hfn,length(ecg_hfn))/length(ecg_hfn)));
ecg_butter_filtereds=fftshift(abs(fft(ecg_butter_filtered,length(ecg_butter_
filtered))/length(ecg_butter_filtered)));
f=linspace(-Fs/2,Fs/2,length(ecg_butter_filtered));
f=transpose(f);

figure;
set(gcf,'color','w');
plot(f,ecg_hfns)
hold on
plot(f,ecg_butter_filtereds)
title("frequency spectrum of ecg signal and its filtered version with 4th
order butterworth filter")
xlabel("Frequency(Hz)");
ylabel("Magnitude");
legend("ecg_hfn","filtered ecg_hfn")

%% IIR Filter (Chebyshev Type I)
clc;
clear all;
close all;
load('ecg_1.mat');
load('ecg_2.mat');
load('ecg_3.mat');

Fs = 1000; % sample frequency
fc = 30; % cutoff frequency
filter_order = 5; % choosing filter order
fc_normalized = fc / (Fs/2); % normalized cutoff frequency
pb_ripple = 2; % ripple in passband
[nom_cheby1, denom_cheby1] = cheby1(filter_order, pb_ripple, fc_normalized,
'low'); % designing chebyshev type I filter
ecg_filtered_cheby1 = filter(nom_cheby1, denom_cheby1, ecg_lfn);
ecg_filtered_cheby2 = filter(nom_cheby1, denom_cheby1, ecg_hfn);
ecg_filtered_cheby3 = filter(nom_cheby1, denom_cheby1, ecg_noisy);

% time domain

N_lfn = length(ecg_lfn);
N_hfn = length(ecg_hfn);
N_noisy = length(ecg_noisy);
Ts = 1/Fs;
t_lfn = 0:Ts:(N_lfn-1)*Ts;
t_hfn = 0:Ts:(N_hfn-1)*Ts;
t_noisy = 0:Ts:(N_noisy-1)*Ts;

figure;
set(gcf,'color','w');
subplot(311)
plot(t_lfn, ecg_lfn);
hold on;
plot(t_lfn, ecg_filtered_cheby1);
grid on;
xlabel("time (s)");
ylabel("amplitude");
title('Original and Filtered ECG Signal in time domain with Cheby (Type I)
Approximation');
legend("original ecg_lfn signal","cheby (type I) filtered ecg signal");

subplot(312)
plot(t_hfn, ecg_hfn);
hold on;
plot(t_hfn, ecg_filtered_cheby2);
grid on;
xlabel("time (s)");
ylabel("amplitude");
title('Original and Filtered ECG Signal in time domain with Cheby (Type I)
Approximation');
legend("original ecg_hfn signal","cheby (type I) filtered ecg signal");

subplot(313)
plot(t_noisy, ecg_noisy);
hold on;
plot(t_noisy, ecg_filtered_cheby3);
grid on;
xlabel("time (s)");
ylabel("amplitude");
title('Original and Filtered ECG Signal in time domain with Cheby (Type I)
Approximation');
legend("original ecg_noisy signal","cheby (type I) filtered ecg signal");

% frequency domain

ecg_lfn_freq = abs(fftshift(fft(ecg_lfn,N_lfn)))/N_lfn;
ecg_hfn_freq = abs(fftshift(fft(ecg_hfn,N_hfn)))/N_hfn;
ecg_noisy_freq = abs(fftshift(fft(ecg_noisy,N_noisy)))/N_noisy;

ecg_cheby1_filtered_freq1 =
abs(fftshift(fft(ecg_filtered_cheby1,N_lfn)))/N_lfn;
f_lfn = linspace(-Fs/2,Fs/2,N_lfn);
ecg_cheby1_filtered_freq2 =
abs(fftshift(fft(ecg_filtered_cheby2,N_hfn)))/N_hfn;
f_hfn = linspace(-Fs/2,Fs/2,N_hfn);
ecg_cheby1_filtered_freq3 =
abs(fftshift(fft(ecg_filtered_cheby3,N_noisy)))/N_noisy;
f_noisy = linspace(-Fs/2,Fs/2,N_noisy);

figure;
set(gcf,'color','w');
subplot(311)
plot(f_lfn, ecg_lfn_freq);
hold on;
plot(f_lfn, ecg_cheby1_filtered_freq1);
grid on;
xlabel("frequency (Hz)");
ylabel("magnitude");
title('Original and Filtered ECG Signals in frequency domain with Cheby
(Type I) Approximation');
legend("original ecg_lfn signal","cheby (type I) filtered ecg signal");

subplot(312)
plot(f_hfn, ecg_hfn_freq);
hold on;
plot(f_hfn, ecg_cheby1_filtered_freq2);
grid on;
xlabel("frequency (Hz)");
ylabel("magnitude");
title('Original and Filtered ECG Signals in frequency domain with Cheby
(Type I) Approximation');
legend("original ecg_hfn signal","cheby (type I) filtered ecg signal");

subplot(313)
plot(f_noisy, ecg_noisy_freq);
hold on;
plot(f_noisy, ecg_cheby1_filtered_freq3);
grid on;
xlabel("frequency (Hz)");
ylabel("magnitude");
title('Original and Filtered ECG Signals in frequency domain with Cheby
(Type I) Approximation');
legend("original ecg_noisy signal","cheby (type I) filtered ecg signal");

%% IIR Filter (Chebyshev Type II)
clc;
clear all;
close all;
load('ecg_1.mat');
load('ecg_2.mat');
load('ecg_3.mat');

Fs = 1000; % sample frequency
fc = 30; % cutoff frequency
filter_order = 5; % choosing filter order
fc_normalized = fc / (Fs/2); % normalized cutoff frequency
sb_ripple = 20; % ripple in stopband (dB), adjust as needed
% designing chebyshev type II filter
[nom_cheby2, denom_cheby2] = cheby2(filter_order, sb_ripple, fc_normalized,
'low');

% filtering the signals
ecg_filtered_cheby2_1 = filter(nom_cheby2, denom_cheby2, ecg_lfn);
ecg_filtered_cheby2_2 = filter(nom_cheby2, denom_cheby2, ecg_hfn);
ecg_filtered_cheby2_3 = filter(nom_cheby2, denom_cheby2, ecg_noisy);

% time domain

N_lfn = length(ecg_lfn);
N_hfn = length(ecg_hfn);
N_noisy = length(ecg_noisy);
Ts = 1/Fs;
t_lfn = 0:Ts:(N_lfn-1)*Ts;
t_hfn = 0:Ts:(N_hfn-1)*Ts;
t_noisy = 0:Ts:(N_noisy-1)*Ts;

figure;
set(gcf,'color','w');
subplot(311)
plot(t_lfn, ecg_lfn);
hold on;
plot(t_lfn, ecg_filtered_cheby2_1);
grid on;
xlabel("time (s)");
ylabel("amplitude");
title('Original and Filtered ECG Signal in time domain with Cheby (Type I)
Approximation');
legend("original ecg_lfn signal","cheby (type I) filtered ecg signal");

subplot(312)
plot(t_hfn, ecg_hfn);
hold on;
plot(t_hfn, ecg_filtered_cheby2_2);
```

```
grid on;
xlabel("time (s)");
ylabel("amplitude");
title('Original and Filtered ECG Signal in time domain with Cheby (Type I)
Approximation');
legend("original ecg_hfn signal","cheby (type I) filtered ecg signal");

subplot(313)
plot(t_noisy, ecg_noisy);
hold on;
plot(t_noisy, ecg_filtered_cheby2_3);
grid on;
xlabel("time (s)");
ylabel("amplitude");
title('Original and Filtered ECG Signal in time domain with Cheby (Type I)
Approximation');
legend("original ecg_noisy signal","cheby (type I) filtered ecg signal");

% frequency domain

ecg_lfn_freq = abs(fftshift(fft(ecg_lfn,N_lfn)))/N_lfn;
ecg_hfn_freq = abs(fftshift(fft(ecg_hfn,N_hfn)))/N_hfn;
ecg_noisy_freq = abs(fftshift(fft(ecg_noisy,N_noisy)))/N_noisy;

ecg_cheby2_filtered_freq1 =
abs(fftshift(fft(ecg_filtered_cheby2_1,N_lfn)))/N_lfn;
f_lfn = linspace(-Fs/2,Fs/2,N_lfn);
ecg_cheby2_filtered_freq2 =
abs(fftshift(fft(ecg_filtered_cheby2_2,N_hfn)))/N_hfn;
f_hfn = linspace(-Fs/2,Fs/2,N_hfn);
ecg_cheby2_filtered_freq3 =
abs(fftshift(fft(ecg_filtered_cheby2_3,N_noisy)))/N_noisy;
f_noisy = linspace(-Fs/2,Fs/2,N_noisy);

figure;
set(gcf,'color','w');
subplot(311)
plot(f_lfn, ecg_lfn_freq);
hold on;
plot(f_lfn, ecg_cheby2_filtered_freq1);
grid on;
xlabel("frequency (Hz)");
ylabel("magnitude");
title('Original and Filtered ECG Signals in frequency domain with Cheby
(Type II) Approximation');
legend("original ecg_lfn signal","cheby (type II) filtered ecg signal");

subplot(312)
plot(f_hfn, ecg_hfn_freq);
hold on;
plot(f_hfn, ecg_cheby2_filtered_freq2);
grid on;
xlabel("frequency (Hz)");
ylabel("magnitude");
title('Original and Filtered ECG Signals in frequency domain with Cheby
(Type II) Approximation');
legend("original ecg_hfn signal","cheby (type II) filtered ecg signal");

subplot(313)
plot(f_noisy, ecg_noisy_freq);
hold on;
plot(f_noisy, ecg_cheby2_filtered_freq3);
grid on;
xlabel("frequency (Hz)");
ylabel("magnitude");
title('Original and Filtered ECG Signals in frequency domain with Cheby
(Type II) Approximation');
legend("original ecg_noisy signal","cheby (type II) filtered ecg signal");

%% Denoising With Wavelet Transform (db4 or sym4)
clc;
clear all;
close all;
load('ecg_1.mat');
load('ecg_2.mat');
load('ecg_3.mat');

Fs = 1000;
Ts = 1/Fs;
Fc = 50/Fs; %cutoff freq/Fs for normalization. cutoff freq determined by
observing freq spectrum of the ecg signal
N = 4; %filter order
% by increasing order we get a sharper filter freq response supresses freqs
above Fc better.
% but increases phase shift and decreases amplitude more. it also requires
more computational power so order of 4 is good enough
[b,a] = butter(N,Fc,"low");
ecg_butter_filtered = filter(b,a,ecg_hfn);

waveletType = 'db4'; %'sym4'
noise_content = ecg_hfn - ecg_butter_filtered;
variance=var(noise_content);

%multilevel

level=5;
[c,l]=wavedec(ecg_hfn,level,waveletType);

j=1;
len=0;
for i=level:-1:1
    cD = detcoef(c,l, i); %detail coef in each level
    th=sqrt(2*variance*log(length(cD)));
    cD = wthresh(cD, 's', th);
    len=len+l(j);
    c(len+1:len+length(cD))=cD;
    j=j+1;
end

denoised_ecg_wavelet=waverec(c,l,waveletType);
denoised_ecg_wavelet_s=fftshift(abs(fft(denoised_ecg_wavelet,length(denoised
_ecg_wavelet))/length(denoised_ecg_wavelet)));


% Plot original and denoised ECG signal

figure;
set(gcf,'color','w');
plot(ecg_hfn);
title('Original ECG Signal');
xlabel('Sample Number');
ylabel('Amplitude');
hold on
plot(denoised_ecg_wavelet);
title('Denoised ECG Signal using Wavelet (db4) with Custom Thresholding');
xlabel('Sample Number');
ylabel('Amplitude');
legend("ecg","denoised")

%freq domain plots
ecg_hfns=fftshift(abs(fft(ecg_hfn,length(ecg_hfn))/length(ecg_hfn)));
f=linspace(-Fs/2,Fs/2,length(ecg_butter_filtered));
f=transpose(f);

figure;
set(gcf,'color','w');
plot(f,ecg_hfns)
hold on
plot(f,denoised_ecg_wavelet_s)
title("frequency spectrum of ecg signal and its filtered version with
wavelet")
xlabel("Frequency(Hz)");
ylabel("Magnitude");
legend("ecg_hfn","filtered ecg_hfn")

%% Wiener Filter
clc;
clear all;
close all;
load('ecg_1.mat');
load('ecg_2.mat');
load('ecg_3.mat');

Fs = 1000;
Ts = 1/Fs;
Fc = 50/Fs; %cutoff freq/Fs for normalization. cutoff freq determined by
observing freq spectrum of the ecg signal
N = 4; %filter order
% by increasing order we get a sharper filter freq response supresses freqs
above Fc better.
% but increases phase shift and decreases amplitude more. it also requires
more computational power so order of 4 is good enough
[b,a] = butter(N,Fc,"low");
ecg_butter_filtered = filter(b,a,ecg_hfn);

noise_content = ecg_hfn - ecg_butter_filtered;
n_l = length(noise_content);
noise_power = sum(abs(noise_content).^2) /n_l;
N = 10;

ecg_hfns=fftshift(abs(fft(ecg_hfn,length(ecg_hfn))/length(ecg_hfn)));
f=linspace(-Fs/2,Fs/2,length(ecg_butter_filtered));
f=transpose(f);

ecg_hfnd = zeros(length(ecg_hfns),length(ecg_hfns));
ecg_hfnd(1:length(ecg_hfn)) = ecg_hfn;
ecg_denoised_wiener = wiener2(ecg_hfnd, [1, N],noise_power);
ecg_denoised_wiener = ecg_denoised_wiener(1:length(ecg_hfn),1);
ecg_denoised_wiener_s =
fftshift(abs(fft(ecg_denoised_wiener,length(ecg_denoised_wiener))/length(ecg
_denoised_wiener)));

% Plot the original and denoised signals for comparison
figure;
set(gcf,'color','w');
plot(ecg_hfn);
title('Original Noisy ECG Signal');
xlabel('Samples');
```

```
ylabel('Amplitude');
hold on
plot(ecg_denoised_wiener);
title('Denoised ECG Signal using Wiener Filter');
xlabel('Samples');
ylabel('Amplitude');
legend("ecg_hfn","filtered ecg_hfn")

figure;
set(gcf,'color','w');
plot(f,ecg_hfns)
hold on
plot(f,ecg_denoised_wiener_s)
title("Frequency spectrum of ecg signal and its filtered version with
Wiener")
xlabel("Frequency(Hz)");
ylabel("Magnitude");
legend("ecg_hfn","filtered ecg_hfn")

%% Adaptive Filter With LMS (Least Mean Square) Algorithm
clc;
clear all;
close all;
load('ecg_1.mat');
load('ecg_2.mat');
load('ecg_3.mat');

N = length(ecg_hfn);
Fs = 1000; % sample frequency

% Parameters for LMS algorithm
mu = 0.0001; % Step size (learning rate)
M = 10;    % Order of the filter
w = zeros(M, 1); % Initial filter weights

% Initialize variables for storing the filtered signal
ecg_filtered = zeros(N, 1);

% Adaptive LMS filtering
for n = M:N
    % Input vector from the signal
    x = ecg_hfn(n:-1:n-M+1);

    % Output of the filter
    y = w' * x;

    % Error calculation (if the reference signal is available)
    % e = reference_signal(n) - y;
    % For noise cancellation, this is usually the noisy signal itself
    e = ecg_hfn(n) - y;

    % Update filter weights
    w = w + mu * e * x;

    % Store the output
    ecg_filtered(n) = y;
end

% Plot the original and filtered signals for comparison
figure;
set(gcf,'color','w');
plot(ecg_hfn);
title('Original Noisy ECG Signal');
xlabel('Samples');
ylabel('Amplitude');
hold on
plot(ecg_filtered);
title('Denoised ECG Signal using Adaptive LMS Filter');
xlabel('Samples');
ylabel('Amplitude');
legend("ecg_hfn","filtered ecg_hfn")

ecg_hfn_freq = abs(fftshift(fft(ecg_hfn,N)))/N;
ecg_hfn_filtered_freq = abs(fftshift(fft(ecg_filtered,N)))/N;
f = linspace(-Fs/2,Fs/2,N);

figure;
set(gcf,'color','w');
plot(f, ecg_hfn_freq);
hold on;
plot(f, ecg_hfn_filtered_freq);
grid on;
xlabel("frequency (Hz)");
ylabel("magnitude");
title('Original and Filtered ECG Signals in frequency domain');
legend("original ecg signal","filtered ecg_lfn signal");
```