ECG Calculations

Signals & Systems Project Report

Presented to: Dr. Michael Melek

أمير أشرف لولى عباس	9230242
أحمد هشام محمد عبدالمنعم شوشه	9230103

1. Project Objective:

To explore different ideas of Heart Rate Monitoring from an ECG Signal using both Time and frequency Domains.

2. Results:

Code for all required tasks is available at the end with proper comments.

a. Plotting the first 10 seconds of the signal:

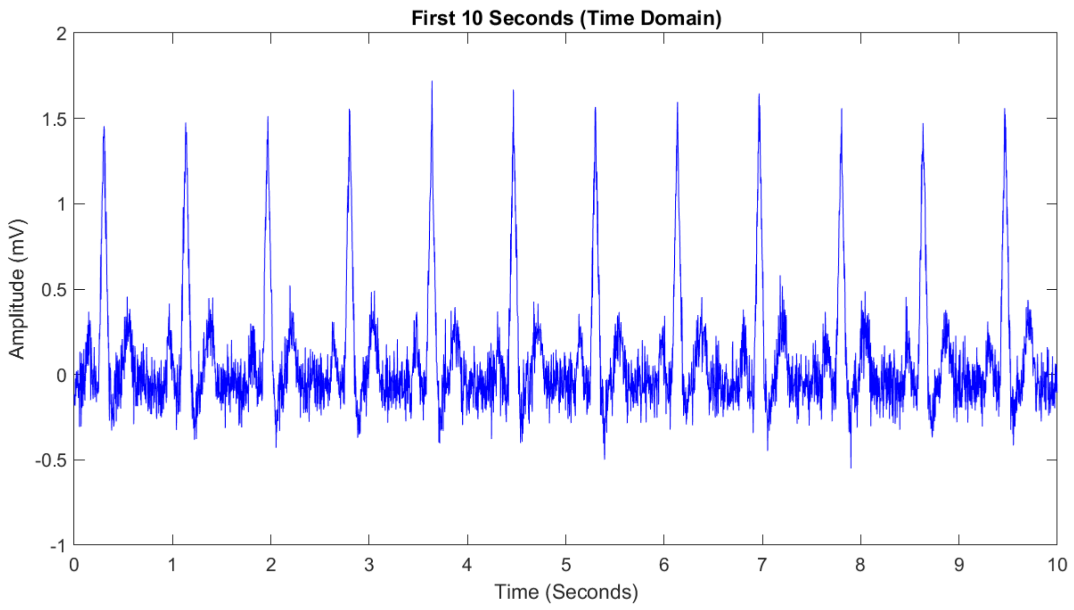


Figure 1 First 10 Seconds of ECG Signal (Time Domain)

b. Calculating average BPM from the first 10 seconds segment using time domain:

Method Used:

1. Find the R-Wave peaks in the segment and their time locations using findpeaks() function.
2. Get the difference between each peak location to get the time difference between them using diff() function.
3. Divide 60 by each Time difference to get the BPM.
4. Calculate the average using mean() function.

Notes:

A Max amplitude of 0.8 was chosen for findpeaks() as we noticed the R-waves go to an amplitude of 1.5mV, this amplitude was lowered after filtering (Task D) to around 1.2mV in most of the signal and 1mV or 0.9mV in some parts, so we found 0.8 to be the sweet spot

A minimum distance between peaks of 0.4 Seconds was chosen to realize a maximum Heart Rate of 150 BPM to give a buffer room to calculate higher heart rates even above the normal resting rate (60 to 100 BPM).

Output:

72.0458 BPM

c. Plotting the 10-Second Segment in Time-Domain:

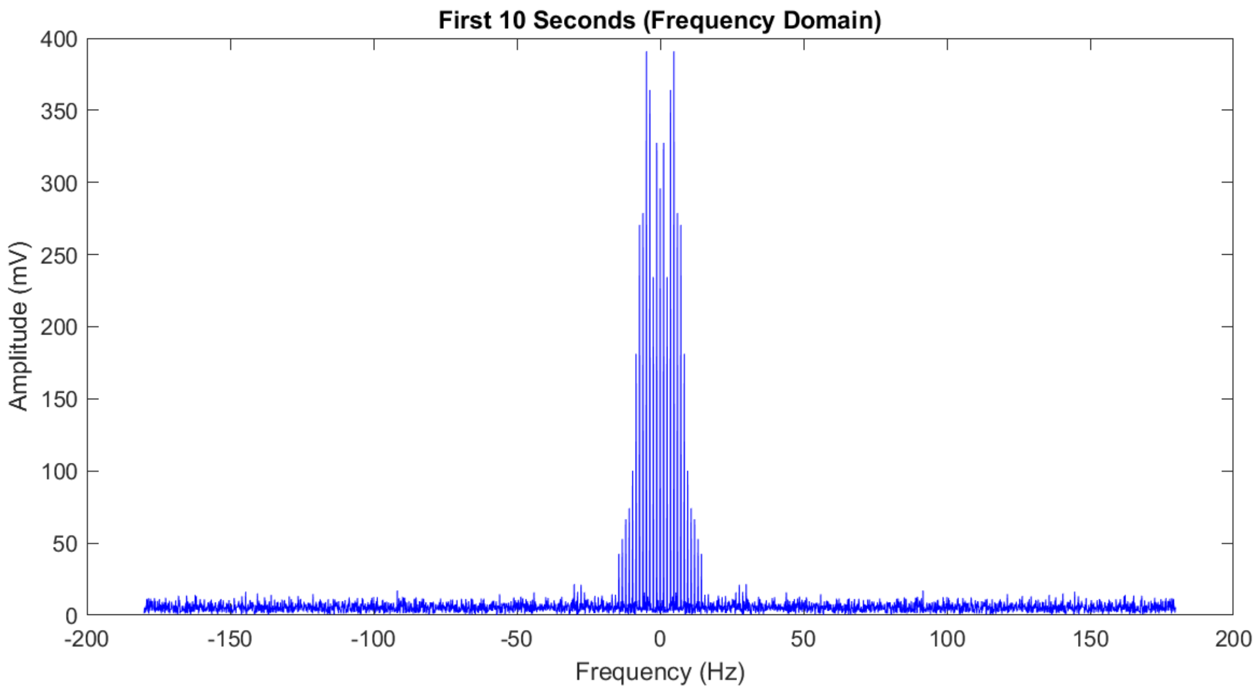


Figure 2 First 10 Seconds of ECG Signal (Frequency Domain)

d. Designing a filter to remove the Out of Band Noise

To design the filter, we plotted the whole signal in frequency domain (figure 4) to get a sense of where the out of band frequencies begin:

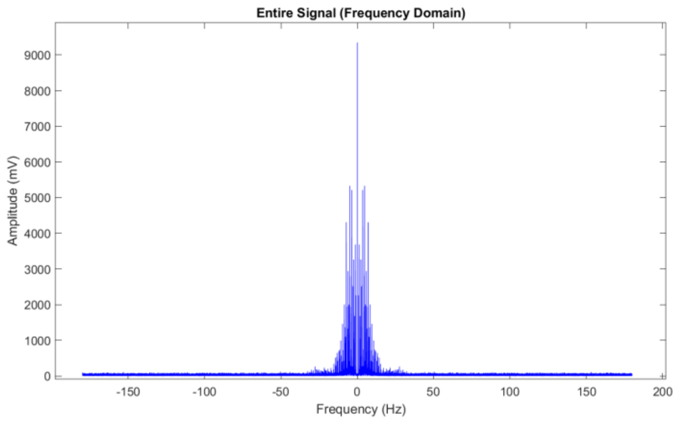


Figure 4 Entire Signal in Frequency Domain

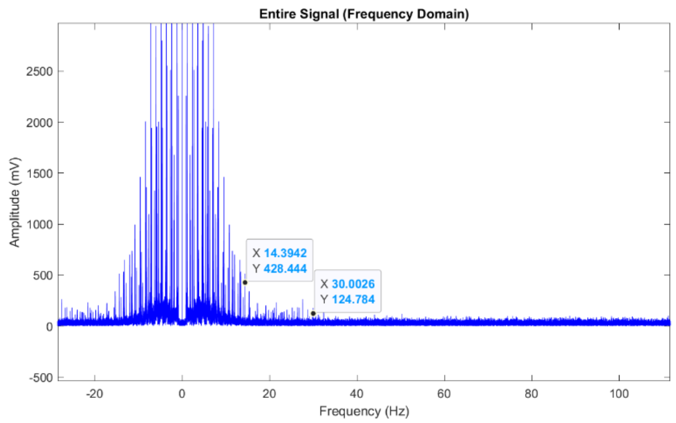
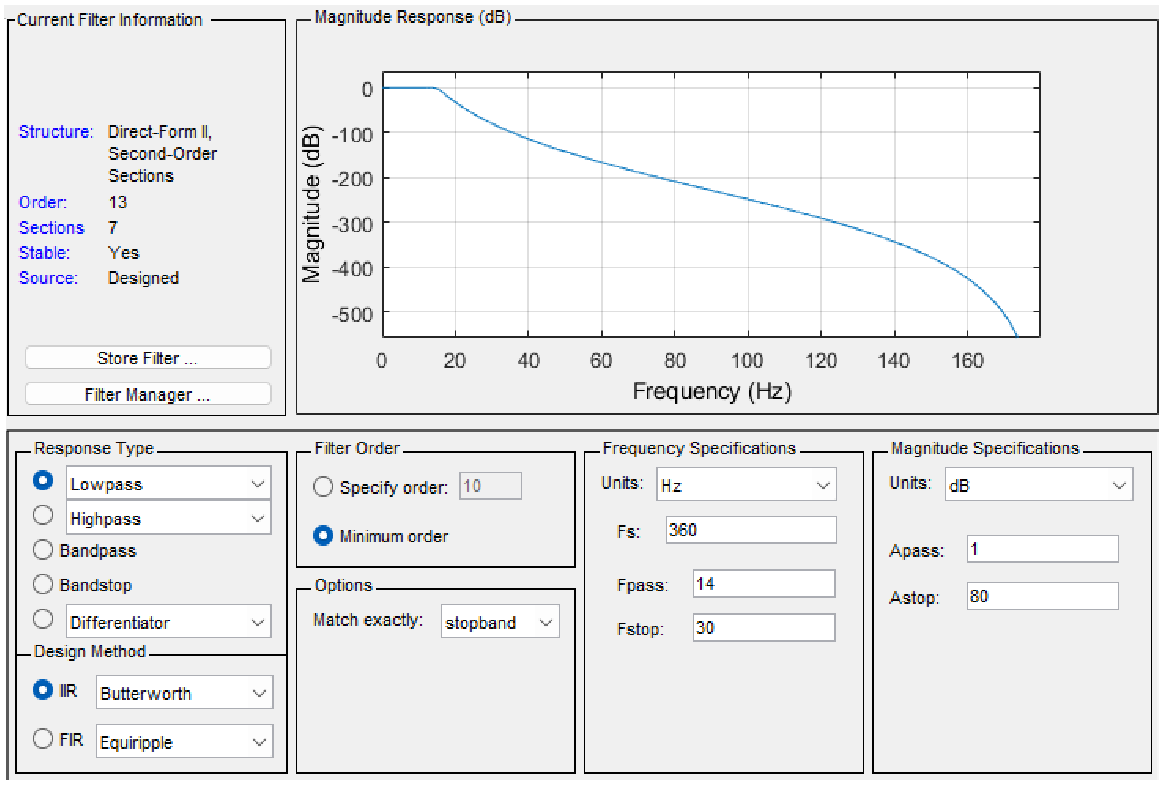


Figure 3 The measured approximate end of the signal

We noticed the signal approximately starts ending at 14hz and completely ends at 30hz, so we designed a Butterworth Low Pass Filter accordingly using MATLAB's filter designer tool.



Current Filter Information

Structure:	Direct-Form II, Second-Order Sections
Order:	13
Sections	7
Stable:	Yes
Source:	Designed

Response Type: [✓ Lowpass] Highpass Bandpass Bandstop Differentiator

Design Method: [✓ IIR] Butterworth; FIR Equiripple

Filter Order: Specify order: 10 [✓ Minimum order]

Options: Match exactly: stopband

Frequency Specifications: Units: Hz; Fs: 360; Fpass: 14; Fstop: 30

Magnitude Specifications: Units: dB; Apass: 1; Astop: 80

Figure 5 Filter Designer screenshot with filter setup

We then plotted the 10-Second segment of the signal again after filtering.

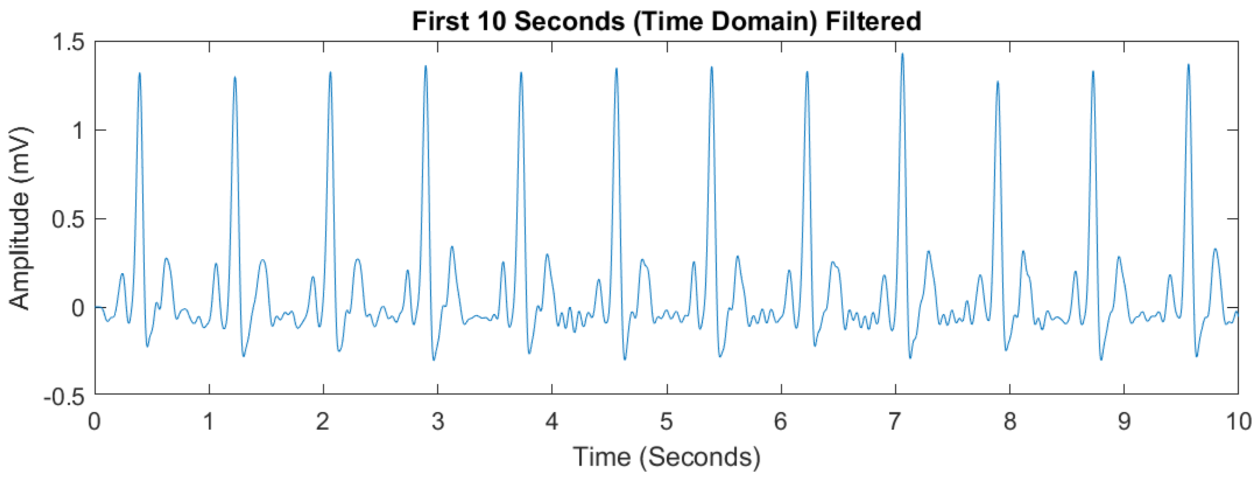


Figure 7 Filtered Time Domain Signal

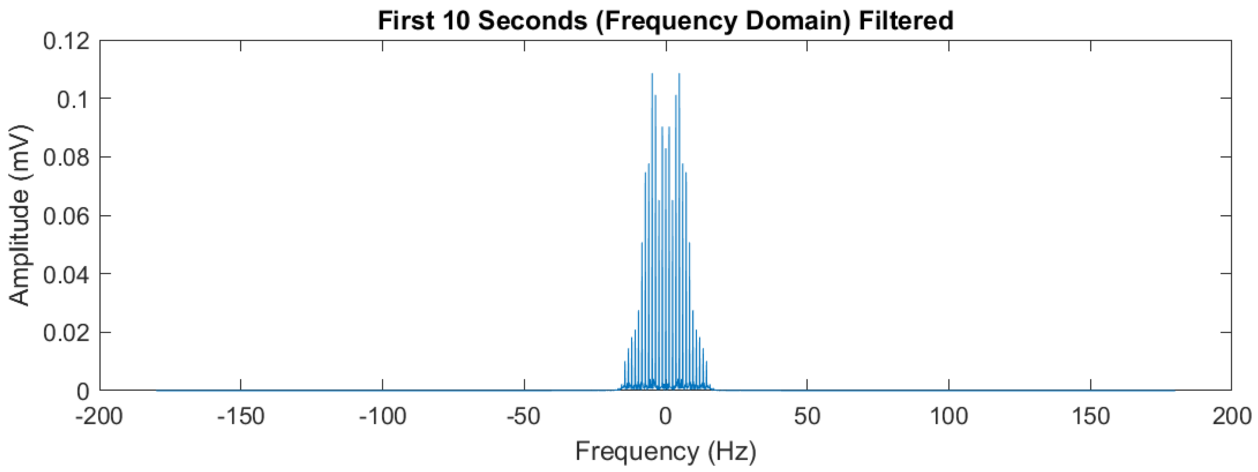


Figure 6 Filtered Frequency Domain Signal

e. Calculating Time rate from Frequency Domain:

The R-Wave has highest amplitude in the time domain signal; thus its frequency should have the highest peak in the Frequency domain representation, but due to imperfections that is not the case.

Instead, we opted to analyze only the frequencies where the normal heart rate of a human being exists between 0.83Hz and 2.5Hz this allows a range between 50 BPM and 150 BPM to be measured. We then evaluated the frequency that has the maximum amplitude in this range, which is the frequency of the measured heart rate, simply multiplying it by 60 gives us the Beats Per Minute.

Calculated Heart rate: **72 BPM**

Which is near identical to the rate calculated using the time domain signal.

Notes:

In the initial Transformation to the frequency domain using FFT we didn't specify an FFT Length which prompted the function to use the same length of the given signal, however this results in a limited frequency resolution and gives us big room for error in case of changing the length of the signal ever so slightly or the window size for STFT as we'll see in Task G.

This value could be increased for better frequency resolution and less room for error, thus resulting in a closer value to the time domain value, but this comes at the cost of computational power, so we opted to leave it as is for this section.

f. Plotting BPM against Time using entire Time Domain Signal:

Using the filter designed in Task D and applying it to the entire signal for a cleaner output free of noise.

We used the same steps in Task B to calculate the BPM without calculating the mean then plotting it against the Locations of each peak to get an accurate representation of when the Heart Rate changes across time.

We additionally added a smooth() function to the BPM output to get a cleaner graph.

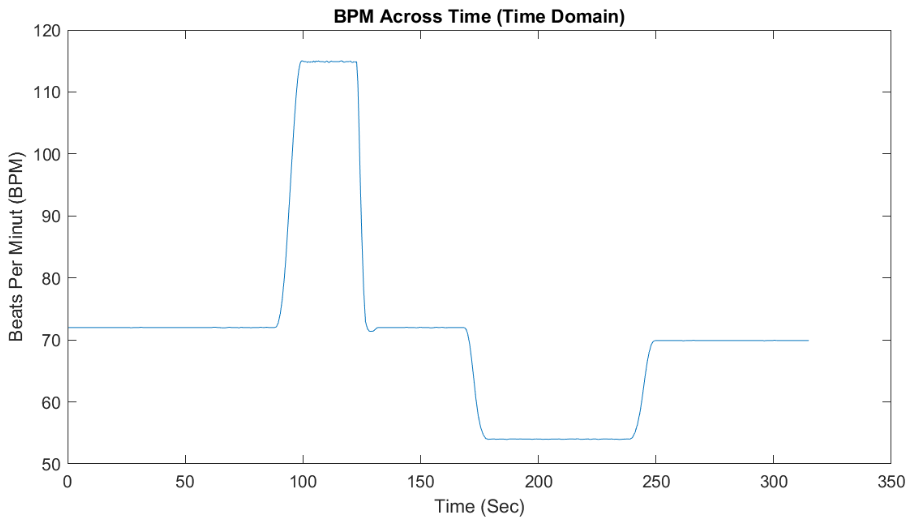


Figure 9 Heart Rate Across Time

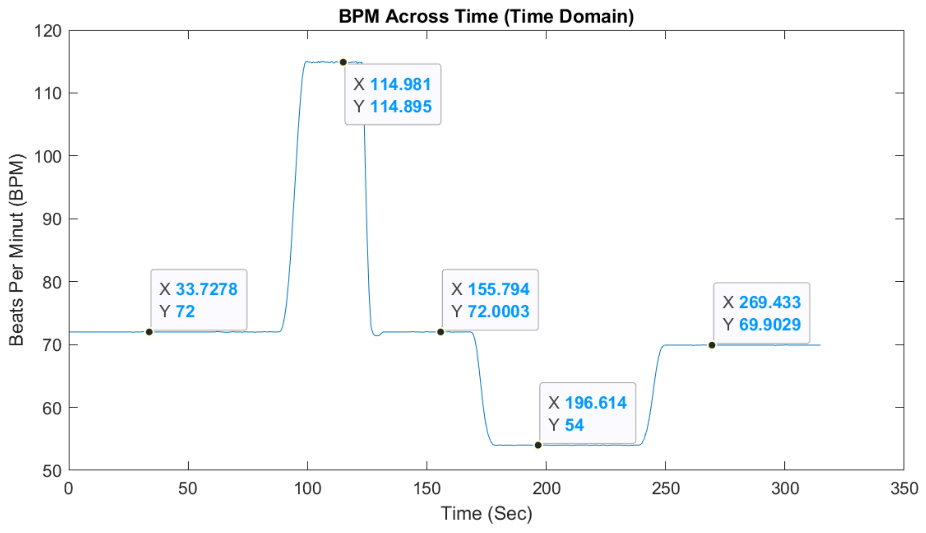


Figure 8 Showing changes of Heart Rate Across time

g. Calculating BPM across time using Frequency Domain:

Method Used:

We used the Short Time Fourier Transform to show the change of the spectrum across time. A similar method to the one in Task E was used to calculate the Heart Rate across time.

We Experimented with different: Window Types, Sizes and FFT Lengths.

As discussed earlier changing the window size greatly affected the output, too small window gives a very small frequency resolution obscuring the signal, too big may obscure the signal and make it hard to calculate the heart rate accurately

FFT Lengths as discussed could be used to increase the frequency resolution resulting in a better representation but at the cost of computational power.

Trial & Error:

Using the Rectangular Window (Simplest type of window) we tested using different window sizes and FFT Lengths to get the optimum output.

We settled on a Window Size of 5 Seconds and FFT Length of 4 times the Window size, this resulted in an output very close to the time domain output.

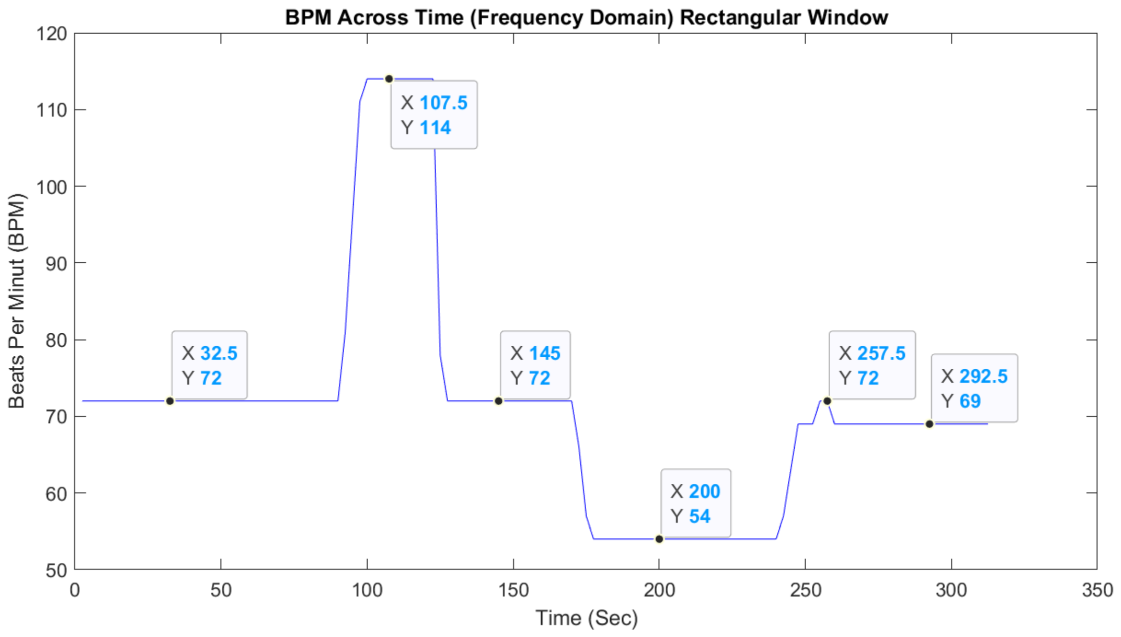


*Figure 10 BPM Across time using Windows Size = 5sec and FFT Length = 4*Window Size*

Experimenting with Window Types:

We experimented with 4 types of Window Functions: Rectangular, Triangular, Hanning and Hamming.

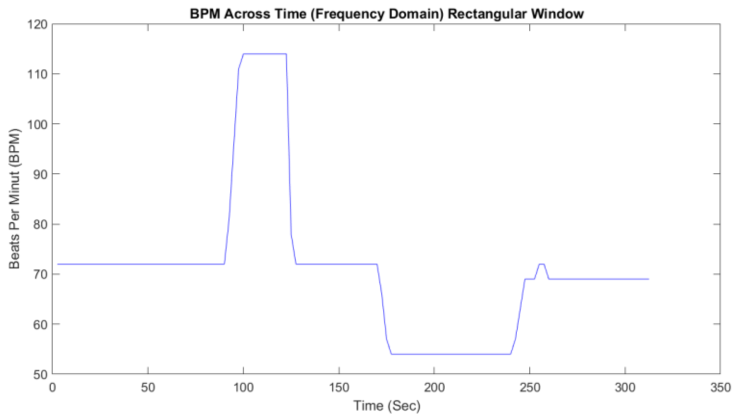


Figure 14 Rectangular Window Output

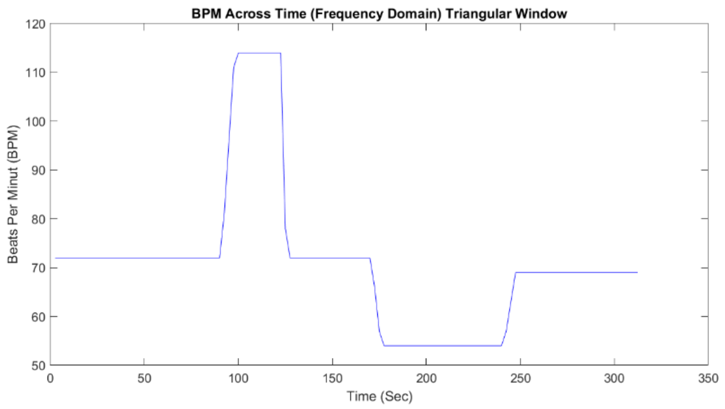


Figure 13 Triangular Window Output

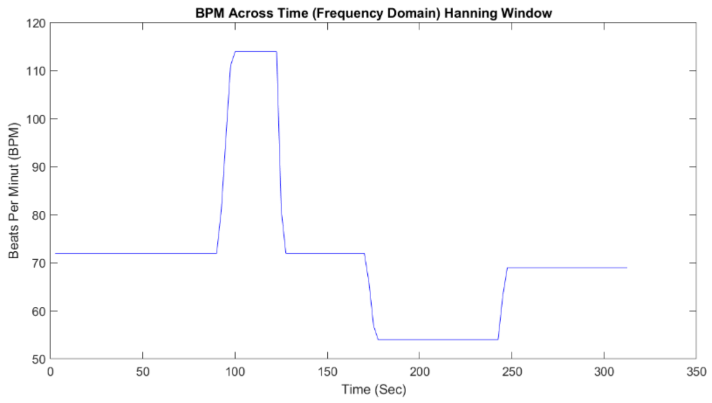


Figure 12 Hanning Window Output

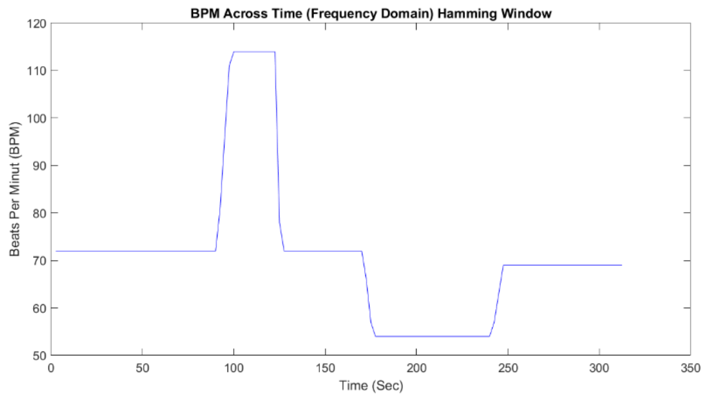


Figure 11 Hamming Window Output

As we can see, all windows give almost identical outputs with the exception to the rectangular making a small bump at the beginning of the signal.

We believe this is due to the chosen method's focus on only the frequency with the max amplitude between 0.5 to 2.5 Hz which are low frequencies close to the center making them far from the effect of the Window function given the window length.

Final Values:

Window Size = 5 Sec * Fs = 1800

FFT Length = 4 * Window Size = 7200

Window Type: Triangular

Since all window types showed similar results, we chose the Triangular window as it produced the closest values to the time domain signal and for having the lowest computational cost.

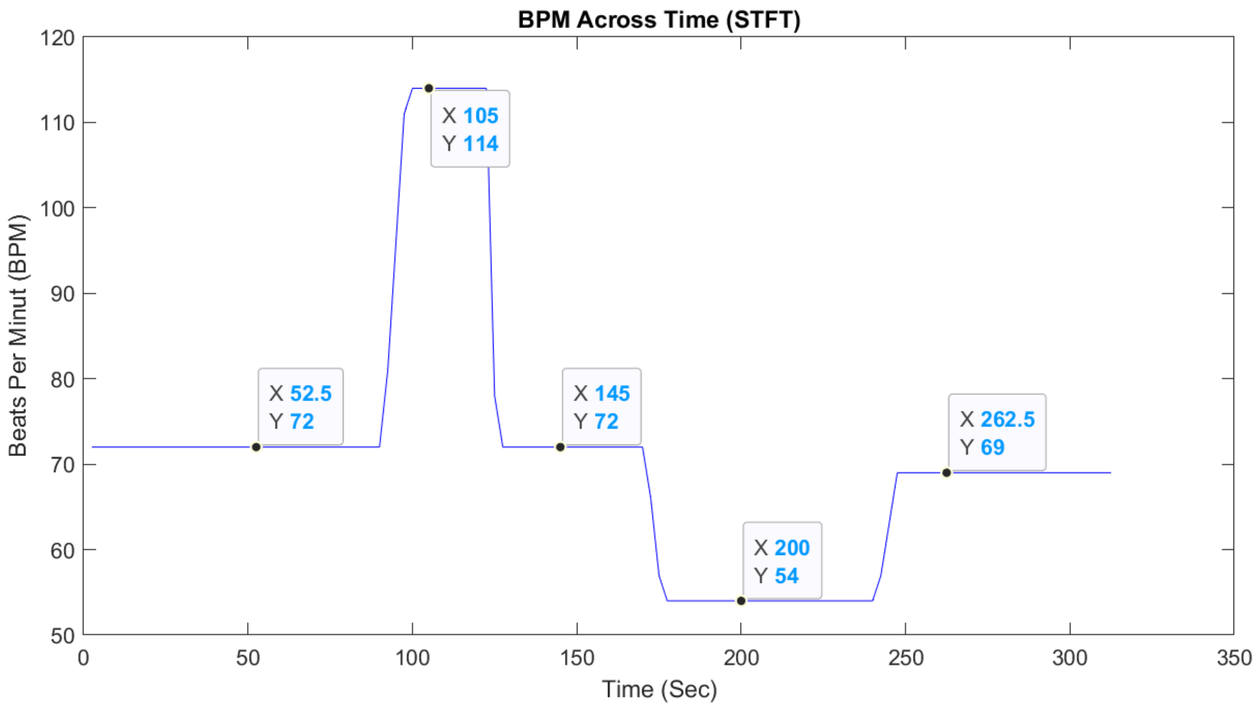


Figure 15 Final BPM Across time using Frequency Domain

h. Identification of the Abnormal Heartbeat Rate regions with their time intervals:

Plotting Abnormal Intervals:

We used the plot obtained from Task F to plot our original signal first, then we determined the ranges where the BPM went above or below the normal range and then we highlighted these ranges where the BPM were abnormal and obtained the following graph:

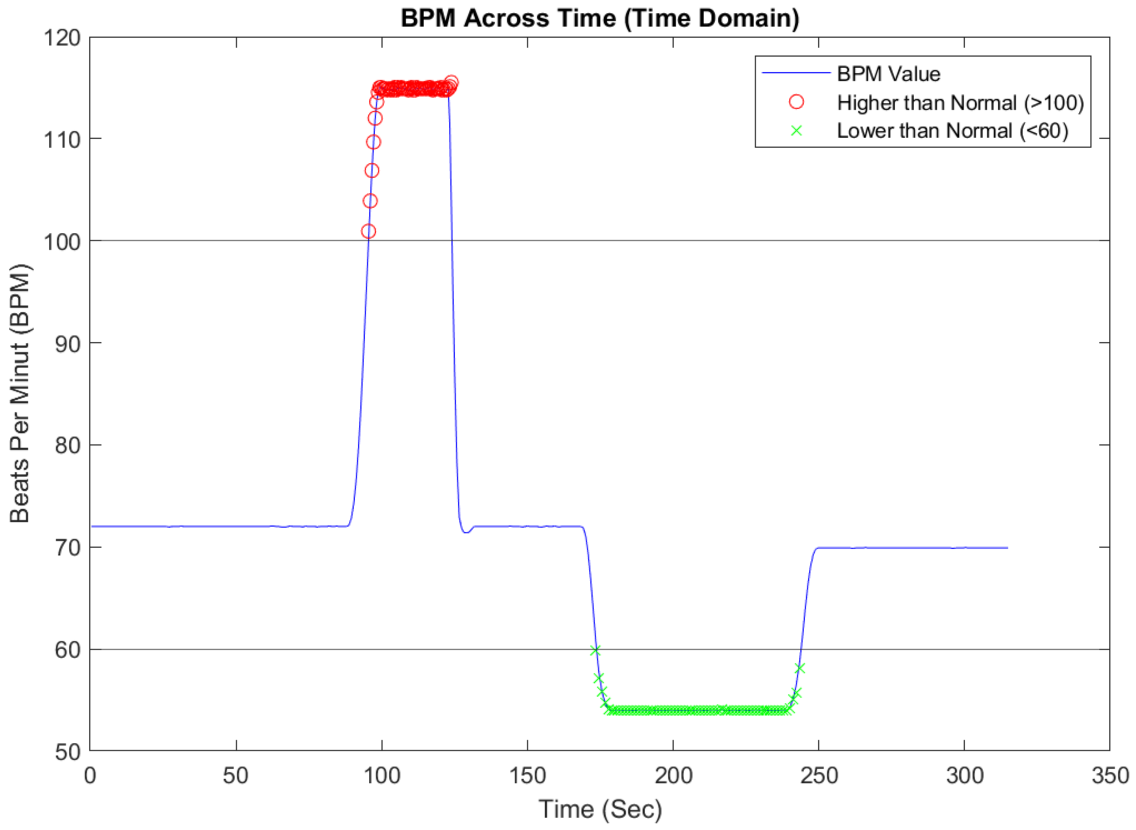


Figure 16 BPM Across Time highlighting Abnormal Intervals

Calculating Abnormal Heart Rate Intervals:

To calculate the abnormal heart rate intervals, we chose a general approach to account for discontinuities and multiple abnormal intervals in the same signal, this approach was not required for this signal, but it's a more general approach that could suit any ECG Signal in the future.

The interval where the BPM exceeded 100 is: **[95.4583, 124.3778] Sec**

The interval where the BPM was below 60 is: **[173.4583 , 244.2944] Sec**

3. Code

Task A:

```
%% Task a
% Plotting Against Time
clc;
clear;
close all;

load('ecg_data.mat');
N=length(ecg_signal); % Number of Signals
Ts=(0:N - 1)/fs; % Calculating Time Period

% Variables to determine starting and ending time
T_start=0;
T_end=10;

ecg_selected_range=ecg_signal((T_start*fs+1):(T_end*fs));

figure(1);
plot(Ts(T_start*fs+1:T_end*fs), ecg_selected_range, 'b')
xlabel('Time (Seconds)');
ylabel('Amplitude (mV)');
title('First 10 Seconds (Time Domain)');
```

Task B:

```
%% Task b
% Calculating BPM using time domain in previously selected range
Max_Amplitude=0.8;
Min_Distance=0.4;

% Finding Peaks
[peaks,
loc]=findpeaks(ecg_selected_range,fs,'MinPeakHeight',Max_Amplitude,'MinPeakDistance',
Min_Distance);

% Calculates the different periods between each Heart Beat in the signal
difference=diff(loc);
average_BPM=mean(60./difference)
```

Task C:

```
%% Task C
% Plotting ECG Signal in Frequency Domain
clc;
close all;

ECG_SIGNAL_SELECTED_RANGE=fft(ecg_selected_range); % Converting to Frequency Domain
using FFT
N_Freq=length(ECG_SIGNAL_SELECTED_RANGE);

Freq=((-N_Freq/2):((N_Freq/2)-1))*fs/N_Freq;  % Shifting X-Axis to be centered around
Zero

% Plotting ECG in Frequency domain using fftshift
plot(Freq,abs(fftshift(ECG_SIGNAL_SELECTED_RANGE)), 'b')
xlabel('Frequency (Hz)');
ylabel('Amplitude (mV)');
title('First 10 Seconds (Frequency Domain)');
```

Task D:

```
%% Task D
%   a filter to remove Out-Of-Band Noise

clc;
close all;

load("Filter_object.mat")   % Loading the Filter
ecg_filtered=Hd.filter(ecg_selected_range); % Applying the filter

% Setting up a tiled layout to view multiple graphs
tiledlayout(2,1);
nexttile;

% Plotting the filtered time domain signal
plot(Ts(T_start*fs+1:T_end*fs), ecg_filtered)
xlabel('Time (Seconds)');
ylabel('Amplitude (mV)');
title('First 10 Seconds (Time Domain) Filtered');

% Calculating the filtered signal in Frequency Domain
ECG_SIGNAL_SELECTED_RANGE=fft(ecg_filtered);
N_Freq=length(ECG_SIGNAL_SELECTED_RANGE);
Freq=(-N_Freq/2:N_Freq/2-1)*fs/N_Freq;

% Plotting the filtered frequency domain signal
nexttile;
plot(Freq,abs(fftshift(ECG_SIGNAL_SELECTED_RANGE))/N_Freq)
xlabel('Frequency (Hz)');
ylabel('Amplitude (mV)');
title('First 10 Seconds (Frequency Domain) Filtered');
```

Task E:

```
%% Task E
% Finding BPM using Frequency Domain

clc;

mag_spectrum=abs(fftshift(ECG_SIGNAL_SELECTED_RANGE));  % Calculating the Magnitude
Spectrum
freq_range=find(Freq>=0.83 & Freq<=2.5);     % Selecting the location of range of
frequencies of Human Heart Rate
[~,Max_Loc]=max(mag_spectrum(freq_range));   % Finding the locations of where the max
amplitude is in the selected frequency range
BPM_Freq=60*abs(Freq(freq_range(Max_Loc)))   % calculating heart beat in the segment
```

Task F:

```
%% Task F
% Plot the Heart Rate across time using Time Domain Signal

clc;
close all;
load("Filter_object.mat")
ecg_filtered_all=Hd.filter(ecg_signal); % Applying Filter

% Calcualting Heart Rate Across Time
[all_peaks,all_loc]=findpeaks(ecg_filtered_all,fs,'MinPeakHeight',Max_Amplitude,'MinP
eakDistance',Min_Distance);
dECG=diff(all_loc);
BPM_Time=60./dECG;

% Plotting Heart Rate Across Time
plot(all_loc(1:length(BPM_Time)),smooth(BPM_Time))
title("BPM Across Time (Time Domain)");
xlabel('Time (Sec)');
ylabel('Beats Per Minut (BPM)');
```

Task G:

```
%% Task G
% Plot the Heart Rate across time using Frequency Domain Signal
clc;
close all;

% Window Setup
window_length=5*fs;             % Specifying Window Size
FTT_Length=window_length*4;     % For FFT Length, Higher for better frequency
resolution
window=triang(window_length);   % Type of Window, Hanning for best solution

% Converting to Spectrogram using Short Time Fourier Transform
[ECG_stft, Freq_stft, Time_stft]=stft(ecg_filtered_all,
fs,"Window",window,"OverlapLength",window_length/2, "FFTLength",FTT_Length);
ecg_spectrogram=abs(ECG_stft); % Calculating Magnitude Spectrogram

% Selecting the range of frequencies of normal heart rate
Freq_range_stft=find(Freq_stft>=0.83 & Freq_stft<=2.5);
Freq_stft_selected=Freq_stft(Freq_range_stft);
ecg_spectrogram_selected_range = ecg_spectrogram(Freq_range_stft,:);

[~,freq_locs]=max(ecg_spectrogram_selected_range); % Finding the Frequencies with the
highest amplitude across time
BPM_Freq_full=60*Freq_stft_selected(freq_locs); % Calculating the BPM across Time

plot(Time_stft, BPM_Freq_full, 'B');
title("BPM Across Time (STFT)");
xlabel('Time (Sec)');
ylabel('Beats Per Minut (BPM)');
```

Task H:

```
%% Task H
% Finding Abnormal Heart Rate intervals
clc;

% Selecting the Interval of Lower than normal Heart Rate
Low_range=find(BPM_Time<60);
Low_interval=all_loc(Low_range);

% Selecting the Interval of Higher than normal Heart Rate
High_range=find(BPM_Time>100);
High_interval=all_loc(High_range);

% Plotting BPM Across time with highlighting the abnormal parts
plot(all_loc(1:length(BPM_Time)),smooth(BPM_Time),'b')
hold on
plot(High_interval,smooth(BPM_Time(High_range)),'ro')
plot(Low_interval,smooth(BPM_Time(Low_range)),'gx')
yline(100)
yline(60)
legend("BPM Value","Higher than Normal (>100)", "Lower than Normal (<60)")
title("BPM Across Time (Time Domain)");
xlabel('Time (Sec)');
ylabel('Beats Per Minut (BPM)');

%state driven loop where I iterate inside loops depending on the region the
%loop is on
%initializing the counter to an appropriate number for array indexing
i=1;
while (i<=length(BPM_Time))
if(BPM_Time(i)>100);        %an abnormal heartbeat state has been detected
    first_num=all_loc(i); %keeping my first value at which the abnormality has been
detected
    while((BPM_Time(i)>100) & (i<length(BPM_Time)) ) %looping until heartbeat rate
stabilizes or until I get to the end of the recorded BPM
        i=i+1;
    end
    disp("critical region, BPM above 100: ["+first_num+","+all_loc(i)+"]")

elseif(BPM_Time(i)<60);  %an abnormal heart state has been detected
    first_num=all_loc(i); %keeping my first value at which the abnormality has been
detected
    while((BPM_Time(i)<60) & (i<length(BPM_Time))) %looping until heartbeat rate
stabilizes or until I get to the end of the recorded BPM
        i=i+1;
    end
      disp("critical region, BPM below 60: ["+first_num+","+all_loc(i)+"]")
end
i=i+1; %updating the counter at the end of the while loop
end
```