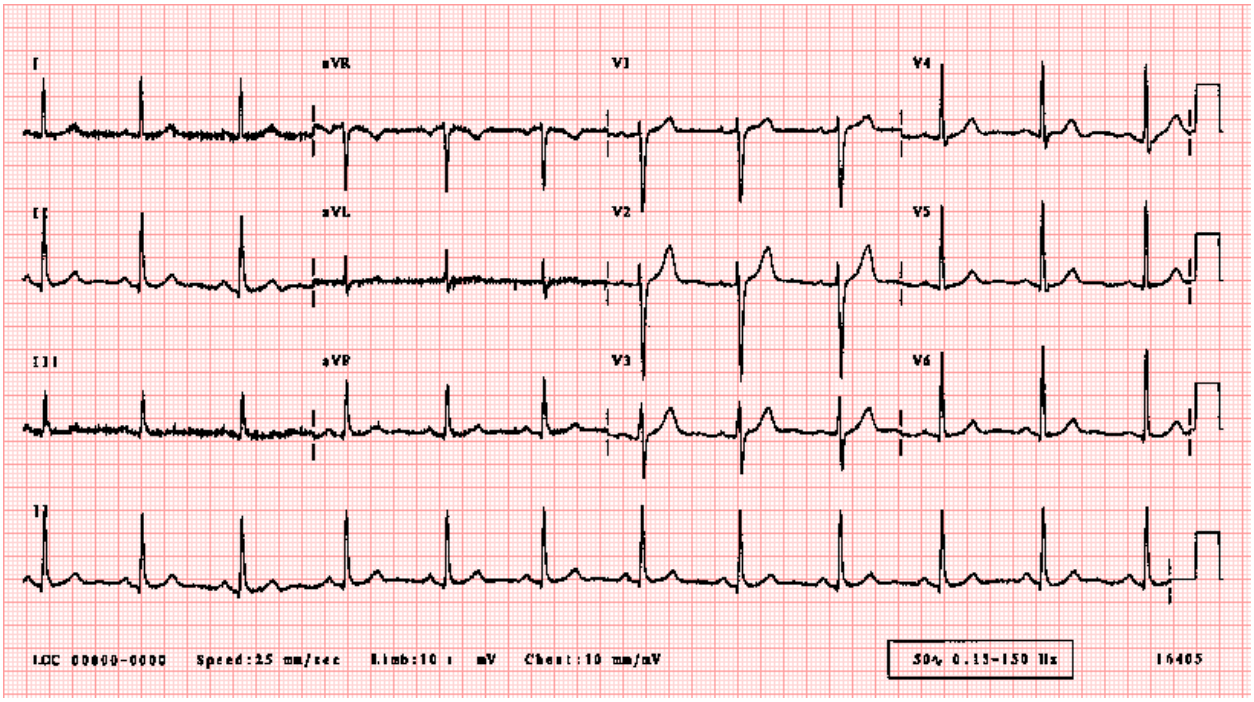
BioEngineering Case Study

Apnea Detection Algorithm using ECG



Dhiyaa Al Jorf

da2863

ENGR-UH 1000 Computer Programming for Engineers
Assignment 5

Step 1 – Problem Identification & Statement:

The objective is to develop an apnea detecting algorithm through data analysis of electrocardiography (ECG) information. The algorithm only needs to work for the file chosen in the development of this algorithm; however, the solution must be easily edited to process other ECG files.

Step 2 – Gathering Information & Input/Output Description:

Breathing monitoring is an essential diagnostic tool and a crucial safety measure when high risk apnea is present. Direct monitoring of breathing is not always possible, hence secondary sources of information, such as heart arrythmia (unexpected heart rate changes) are essential to track apnea events. The diagnosis of sleep apnea is difficult and often involves a trained medical staff to watch overnight the sleep cycle of a patient. Developing systems that can allow the diagnosis of sleep apnea in a cost effective is therefore crucial. The direct measurement of chest movement tends to carry high noise given the similar frequency of other body movements, making this technique ineffective. However, the potential to use heart arrythmia to detect apnea events is a promising approach. ECG signals are one of the most feature rich and non-intrusive ways to detect various cardiac disorders. Cost effective and powerful embedded ECG devices are now widely available. Additionally, a lot of research has been carried into determining the relationship between apnea events and heart rate [1]. Thus, Electrocardiography (ECG) signals from standardized medical datasets will be used in this program to derive relevant diagnostic information and use this information to estimate when an apnea is occurring.

DATA DESCRIPTION:

All the data that will be used in this solution comes from the Apnea-ECG-Database on the PhysioNet website (https://www.physionet.org/content/apnea-ecg/1.0.0/) [2] There are a total of 70 records on the website but this solution will only be using the data for the recording a01.

Format:

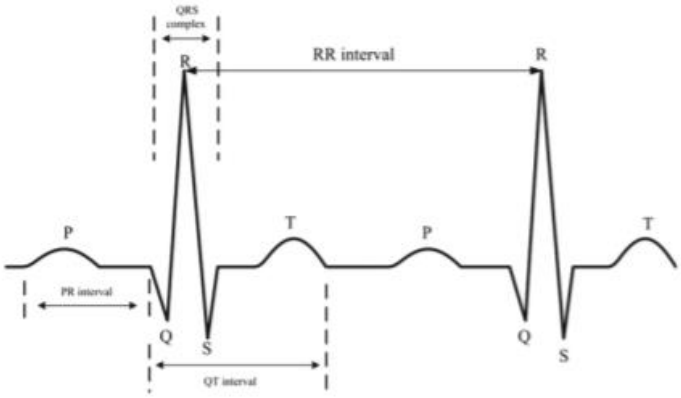
Several files are associated with each recording. The file a01.dat contains the signal information saved in int16 format (every 16 bits translate to a single signed integer), with a sampling frequency of a 100 Hz, with a conversion factor of 200 A/D units per millivolt, and 2957000 data points. The a01.hea file contains all the necessary information about the signal like the sampling frequency, conversion factor, number of data points, and format in which the signal is stored. Additionally, while the website provides a01.apn which includes binary annotations indicating whether apnea was detected by a medical expert at each minute, this solution made use of the simplified label files provided in the assignment description which contain the same information.

Dealing with the Data:

First we will need to read the data in the a01.dat in int16 format. After that, we need to truncate the data such that only the data pertaining to the first 4 hours remains, i.e. we only need the first 4*60*60 = 14400 data points. Then, we'll have to convert the signal into millivolts by dividing by the conversion factor (200 as previously mentioned).

Secondly, we'll need to resample the data to 500 Hz and apply a low-pass filter to it. The low-pass filter involves a cut-off frequency above which all frequencies must be ignored. Experimenting with the cut-off frequency will help us decide what value to use for that parameter.

After that, the data needs to undergo heart rate variation analysis (minimum, maximum, and average heart rate). We'll need to calculate the RR distances (we need a way to find the peaks) and divide them into 60 to get the heart rate for each minute, but before that, we will test our methods on a single minute.

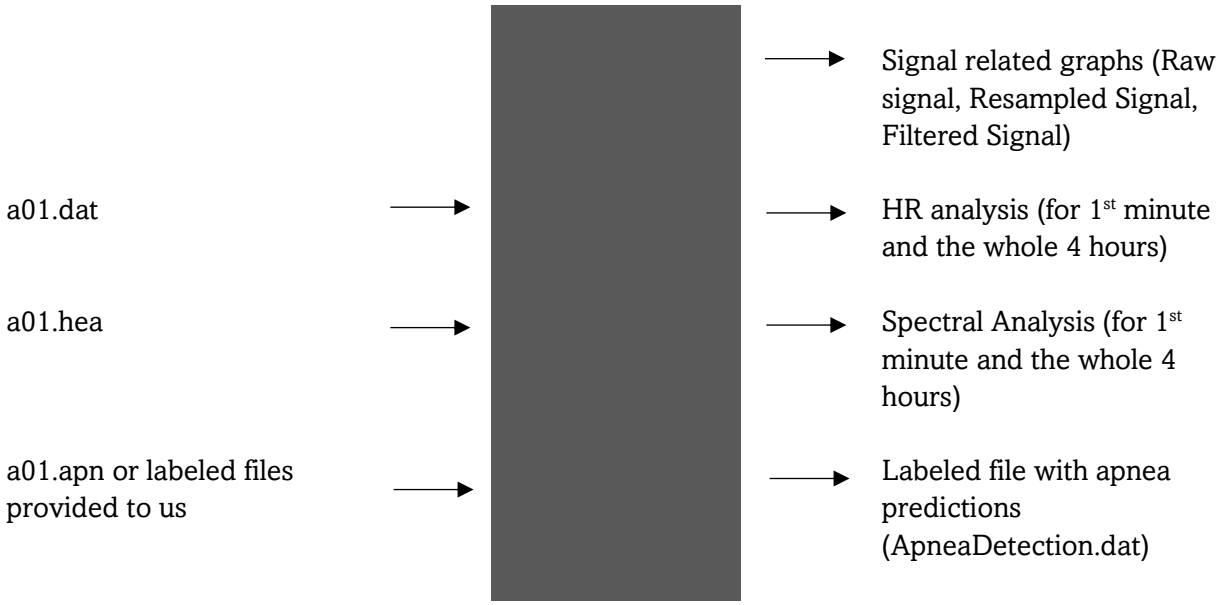


After completing all those tasks, we must perform spectral analysis of the data by performing a fast-Fourier-transform (FFT) on our data, calculating the power-spectral-density (PSD), then truncating the negative frequency related values. Before applying it on the whole 4 hours, we will test our methods on a single minute.. The PSD of every minute will indicate a peak power at a specific frequency, meaning that most of the signal's power for that minute is produced at that frequency. The PSD data will be visualized into a spectrogram, and the result of the previous step (HR analysis) will be overlapped onto the spectrogram to start analyzing any patterns.

Now that we have all the necessary information about the signal, we can analyze the patterns for which apnea is present. The resulting identified patterns will be used to design an apnea detecting algorithm on that data. After running the algorithm, the accuracy will be calculated to determine how good the algorithm is.

IO Description:

IO Diagram:



Step 3 – Test Case Description & Algorithm Design:

Test Description:

1. Raw Signal: The raw signal graph should look like a normal ECG file with some noise.
2. Resampling: The resampled signal graph should have more data points than the non-resampled signal
3. Low-pass filter:
    a. First I should perform some tests to decide on a good cut-off frequency. According to which one looks better, I will decide which to use in the final product.
    b. After deciding, I'll be plotting the filtered data for the first minute. It should look smooth yet maintain its general form
4. First Minute HR Analysis: The code should produce the minimum, maximum and average HR for the first minute.
5. Heart Rate variation: The graph should show the average HR values within normal human heart range.
6. PSD for first minute: The PSD graph should show a peak at a certain frequency. That frequency corresponds to the power-carrying frequency (the most dominant frequency).
7. Spectrogram: The data should align with the average heart rate variation to a certain extent.
8. Algorithm: The algorithm must generate a text file showing at what times the program detects apnea. The accuracy of that should then be measured (either through code or manually).

Algorithm:

```
FindRIndeces(signal)
        Assign 2 to i
        Repeat While i is less than the length of signal – 1
                If (signal(i) is greater than the previous and next values and 0.6)
                        Add I to RIndeces
        Return RIndeces

Clear all the memory
Clear the output screen
Print "We'll be dealing with the a01 information", Newline
Generate a file handle for "a01.dat" and store it into FID
If FID is equal to -1
        PrintError "Couldn't open the file, please make sure the file is in your working directory",
        Newline
Otherwise
        Read all the information stored in the file referred to by the tag FID in the int16 format into
        rawSignal
        Close the file referred to by the tag FID
        Assign 100 to Fs1
        Assign 200 to AtoVCF
        Assign the first 4*3600*Fs1 data points in rawSignal to rawSignal
        Divide each value in rawSignal by AtoVCF
        Print "a01.dat successfully read. Graph of first 2 seconds generated…", Newline
```

Set up a window with the name “Raw Signal” to display the graph into
Assign 2 to numSec
Assign the values form 1/Fs1 to 3600*4 with a step size of 1/Fs1 to t
Add the grid to the graph
Plot rawSignal from index 1 to numSec*Fs1 with t(1:numSec*Fs1) as the x-axis
Set the x label to “Time (sec)”
Set the y label to “Voltage (mV)”
Set the figure title to “Raw Signal”
Clear numSec
Save the last figure to the working directory as “RawSignal.fig”
Assign 500 to Fs2
Assign the values form 1/Fs2 to 3600*4 with a step size of 1/Fs2 to t2
Resample the data in rawSignal to Fs2 using spline interpolation and save it into resampledSignal1
Print a confirmation message that this step was successful
Set up a window with the name “Resampled Signal” to display the graph into
Assign 2 to numSec
Add the grid to the graph
Plot resampledSignal1 from index 1 to numSec*Fs2 with t2(1:numSec*Fs2) as the x-axis
Keep the graph open
Plot rawSignal from index 1 to numSec*Fs1 with t(1:numSec*Fs1) as the x-axis
Set the x label to “Time (sec)”
Set the y label to “Voltage (mV)”
Set the figure title to “Raw Signal VS Resampled Signal”
Add a legend labeling the first plot as “Resampled Signal” and the second as “Raw Signal”
Clear numSec
Stop writing on the graph
Save the last figure to the working directory as “ResampledSignal.fig”
Print a confirmation message that this step was successful
Create a low-pass filter with a filter order of 25, a cutoff frequency of 16 and a sampling rate of Fs2 and save it as lowPassFilter
Assign the data resampledSignal1 filtered through the filter lowPassFilter to filteredSignal
Set up a window with the name “Filtered Signal” to display the graph into
Assign 60 to numSec
Add the grid to the graph
Plot filteredSignal from index 1 to numSec*Fs2 with t2(1:numSec*Fs2) as the x-axis
Keep the graph open
Plot resampledSignal from index 1 to numSec*Fs2 with t2(1:numSec*Fs2) as the x-axis
Set the x label to “Time (sec)”
Set the y label to “Voltage (mV)”
Set the figure title to “Filtered Signal VS Resampled Signal”
Add a legend labeling the first plot as “Filtered Signal” and the second as “Resampled Signal”
Clear numSec
Stop writing on the graph
Save the last figure to the working directory as “FilteredSignal.fig”
Save the first 1*60*Fs2 values in filteredSignal to minute
Assign findRTimes(minute) to RIndeces
Assign RIndeces divided by Fs2 into RTimes

Assign the differences between the data points in RTimes to RR
Assign the mean of RR to RRav
Assign 60 divided by the minimum of RR to MaxHR
Assign 60 divided by the maximum of RR to MinHR
Assign 60 divided by RRav to HRav
Print a confirmation message that this step was successful
Assign 1 to i
Repeat While i is less than or equal to 240
    Save the values from indices (i-1)*60*Fs2 + 1 to i*60*Fs2 in filteredSignal to minute
    Assign findRTimes(minute) to RIndeces
    Assign RIndeces divided by Fs2 into RTimes
    Assign the differences between the data points in RTimes to RR
    Assign the mean of RR to RRav
    Add 60 divided by the minimum of RR to MaxHR
    Add 60 divided by the maximum of RR to MinHR
    Add 60 divided by RRav to HRav
    Increment i by 1
Print a confirmation message that this step has been successful
Set up a window with the name “HR Variation” to display the graph into
Assign 1 to 240 with a step size of 1 to minutes
Add the grid to the graph
Plot HRav with minutes as the x-axis
Set the x label to “Time (min)”
Set the y label to “HR (bpm)”
Set the figure title to “Filtered Signal VS Resampled Signal”
Signal”
Save the last figure to the working directory as “HRavVariation.fig”
Set up a window with the name “HR Variation” to display the graph into
Add the grid to the graph
Plot MinHR with minutes as the x-axis
Keep the graph open
Plot MaxHR with minutes as the x-axis
Set the x label to “Time (min)”
Set the y label to “HR (bpm)”
Set the figure title to “Filtered Signal VS Resampled Signal”
Add a legend labeling the first plot as “Minimum HR” and the second as “Maximum HR”
Save the last figure to the working directory as “HRVariation.fig”
Assign 4 to Fs3
Assign the values form 1/F31 to 3600*4 with a step size of 1/Fs3 to t3
Resample filteredSignal to 4 Hz and save it into resampledSignal2
Save the first 1*60*Fs3 values in resampledSignal2 to minute
Assign the length of minute to N
Perform a Fast-Fourier-Transform on minute and save it into fftOfMin
Assign (1/(Fs3*N))*|each value of fftOfMin|^2 to PSD
Assign the first N/2 values in PSD to PSD
Print a confirmation message that this step was successful
Assign the values 1/N to Fs3/2 with a step size of Fs3/N to fAxis
Set up a window with the name “Power-Spectral-Density First Minute” to display the graph into

Add the grid to the graph
Plot PSD with fAxis as the x-axis
Set the x label to “Frequency (Hz)”
Set the y label to “Power”
Set the figure title to “Power Spectral Density”
Save the last figure to the working directory as “PSDFirstMinute.fig”
Assign 1 to i
Repeat While i is less than or equal to 240
    Save the first the values from indices (i-1)*60*Fs3 + 1 to i*60*Fs3 values in resampledSignal2 to minute
    Assign the length of minute to N
    Perform a Fast-Fourier-Transform on minute and save it into fftOfMin
    Assign (1/(Fs3*N))*|each value of fftOfMin|^2 to PSD
    Add the first N/2 values in PSD to PSDArray
    Increment i by 1
Print a confirmation message that this step was successful
Set up a window with the name “Spectrogram” to display the graph into
Add the grid to the graph
Assign the values 1 to 240 with a step size of 1 to tAxis
Plot a the 3D surface PSDArray with fAxis as the x-axis and tAxis as the y-axis
Look at the graph from the top
Keep the graph open
Plot the rescaled HRav array on top of the current graph
Set the x label to “Time (min)”
Set the y label to “Frequency (Hz)”
Add the color bar
Set the figure title to “Spectrogram of ECG”
Save the last figure to the working directory as “Spectrogram.fig”
Assign 1 to i
Repeat While (I is less than the length of HRav)
    Find the peaks happening in PSDArray(1:65, i) above a value of 0.002 and save them into belowLinePeaks
    Suppress any error messages if there were no peaks found
    If (HRav(i) is less than or equal to 68 OR any of belowLinePeaks values are true)
        Assign 1 to predictions(i)
    Otherwise
        Assign 0 to predictions(i)
Create a file of name “ApneaPrediction.dat” in the writing format and save its handle to FID
Print “Time(min): Prediction:”, Newline into FID
Assign 1 to i
Repeat While i is less than 240
    If (predictions(i) is equal to 1)
        Assign ‘A’ to prediction
    Otherwise
        Assign ‘N’ to prediction
    Print i, prediction into FID
Close FID
Print confirmation message that the program has finished executing properly

Step 4 – MATLAB Code:

```
% This function finds the indeces at which the R peaks were found
function RIndeces = findRIndices(signal)
    RIndeces = [];
    for i = 2:length(signal)-1
        if ((signal(i) > signal(i-1)) && (signal(i) >= signal(i+1)) && (signal(i) > 0.6))
            RIndeces = [RIndeces, i];
        end
    end
end



clear;
clc;

% Step 2: Reading File and Resampling
fprintf('We''ll be dealing with the a01 information\n');
FID = fopen('a01.dat');
if (FID == -1)
    error('Couldn''t open the file, please make sure the file is in your working directory\n');
else
    rawSignal = fread(FID, 'int16');
    fclose(FID);
    Fs1 = 100; % Sampling frequency of the original data
    AtoVCF = 200; % A/D to milliVolts conversion factor
    rawSignal = rawSignal(1:4*3600*Fs1); % Truncating the values for times after 4 hours
    rawSignal = rawSignal./AtoVCF; % Converting raw signal data into Volts

    fprintf('a01.dat successfully read. Graph of first 2 seconds generated...\n');

    % Plotting the first 2 seconds of the raw signal
    figure('name','Raw Signal');
    numSec = 2;
    t = 1/Fs1:1/Fs1:3600*4; % Time axis in seconds
    grid on
    plot(t(1:numSec*Fs1),rawSignal(1:numSec*Fs1));
    xlabel('Time (sec)');
    ylabel('Voltage (mV)');
    title('Raw Signal');
    hold off

    saveas(gcf,'RawSignal.fig');

    % Resampling to 500 Hz
    Fs2 = 500;
    t2 =  1/Fs2:1/Fs2:3600*4;
    resampledSignal1 = interp1(t, rawSignal, t2, 'spline');

    fprintf('Signal successfully resampled. Graph of first 2 seconds generated...\n');

    % Plotting the first 2 seconds of the resampled signal
    figure('name','Resampled Signal');
    numSec = 2;
    plot(t2(1:numSec*Fs2),resampledSignal1(1:numSec*Fs2), '.');
    grid on
    hold on
    plot(t(1:numSec*Fs1),rawSignal(1:numSec*Fs1), '.');
    xlabel('Time (sec)');
    ylabel('Voltage (mV)');
    title('Raw Signal VS Resampled Signal');
    legend({'Resampled Signal', 'Raw Signal'})
    hold off
```

```
    saveas(gcf,'ResampledSignal.fig');

    fprintf('Signal successfully filtered. Graph of first 2 seconds generated...\n');

    % Step 3: Applying the low pass filter
    lowPassFilter = designfilt('lowpassfir', 'Filterorder', 25, 'CutoffFrequency', 16, 'SampleRate', Fs2);
    filteredSignal = filter(lowPassFilter, resampledSignal1);

    % Plotting the first 60 seconds of the filtered signal
    figure('name','Filtered Signal');
    numSec = 60;
    grid on
    plot(t2(1:numSec*Fs2),filteredSignal(1:numSec*Fs2));
    hold on
    plot(t2(1:numSec*Fs2),resampledSignal1(1:numSec*Fs2));
    title('Resampled Signal VS Filtered Signal');
    legend({'Filtered Signal', 'Resampled Signal'})
    xlabel('Time (sec)');
    ylabel('Voltage (mV)');
    hold off

    saveas(gcf,'FilteredSignal.fig');

    % Step 4: Finding the Average. Minimum, and Maximum HR for the first
    % minute

    minute = filteredSignal(1:1*60*Fs2);
    RIndeces = findRIndices(minute);
    RTimes = RIndeces./Fs2;
    RR = diff(RTimes);
    RRav = mean(RR);
    MaxHR = 60/min(RR);
    MinHR = 60/max(RR);
    HRav = 60/RRav;

    fprintf('First minute Heart-Rate information analyzed:\nMin Heart-Rate: %.0f\nMax Heart-Rate:
%.0f\nAverage Heart-Rate: %.0f\n', MinHR, MaxHR, HRav);

    % Step 5: Finding the Average, Minimum, and Maximum HR for each minute
    HRav = [];
    MaxHR = [];
    MinHR = [];

    for i = 1:240
        minute = filteredSignal((i-1)*60*Fs2 + 1:i*60*Fs2);
        RIndeces = findRIndices(minute);
        RTimes = RIndeces./Fs2;
        RR = diff(RTimes);
        RRav = mean(RR);
        MaxHR = [ MaxHR, 60/min(RR)];
        MinHR = [MinHR, 60/max(RR)];
        HRav = [HRav, 60/RRav];
    end

    fprintf('First 4-hour Heart-Rate information analyzed. Graphs generated\n');

    % Plotting the average, minimum, & maximum heart rate for each minute over time
    figure('name','HR Variation');
    minutes = 1:240;
    grid on
    plot(minutes, HRav);
    title('Average HR Over the 4 Hr Period');
    xlabel('Time (min)');
    ylabel('HR (bpm)');

    saveas(gcf,'HRavVariation.fig');

    figure('name','Max and Min HR Variation');
    grid on
```

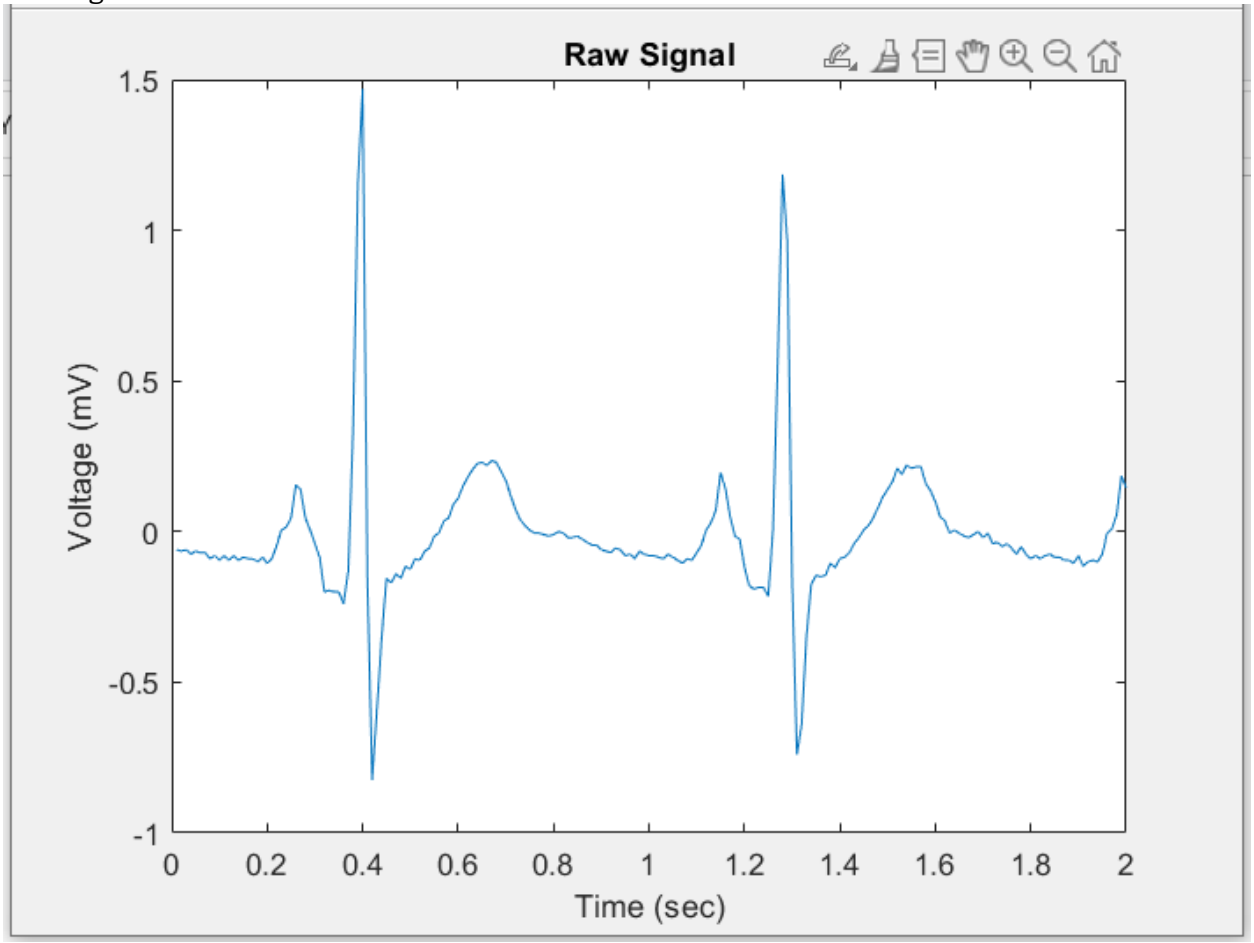
```
plot(minutes, MaxHR);
hold on
plot(minutes, MinHR);
legend({'Maximum HR', 'Minimum HR'});
title('Min and Max HR Over the 4 Hr Period');
xlabel('Time (min)');
ylabel('HR (bpm)');
hold off

saveas(gcf,'HRVariation.fig');

% Step 6: Performing a Fast-Fourier-Transform on the First Minute of Data

Fs3 = 4;
t3 = 1/Fs3: 1/Fs3:4*3600;
resampledSignal2 = resample(filteredSignal, Fs3, Fs2);

minute = resampledSignal2(1:1*60*Fs3);
N = length(minute);
fftOfMin = fft(minute);
PSD = (1/(Fs3*N))* abs(fftOfMin).^2;
PSD = PSD(1:N/2); % Removing the negative side

fprintf('Spectral analysis of first minute completed. Graph generated\n');

% Plotting the Power-Spectral-Density of the first minute
fAxis = 1/N:Fs3/N:Fs3/2;
figure('name','Power-Spectral-Density First Minute');
plot(fAxis, PSD);
grid on
title ('Power Spectral Density');
xlabel('Frequency (Hz)');
ylabel('Power');
hold off

saveas(gcf,'PSDFirtsMinute.fig');


% Step 7: Performing FFt on every minute and plotting the spectrogram

% Performing FFT on every minute and saving PSD into PSDArray
PSDArray = [];
hold on
for i = 1:240
    minute = resampledSignal2((i-1)*60*Fs3 + 1:i*60*Fs3);
    %t = i/Fs + 1:1/Fs:i*60;
    N = length(minute);
    fftOfMin = fft(minute);
    PSD = (1/(Fs3*N))* abs(fftOfMin).^2;
    PSD = PSD(1:N/2);
    PSDArray(:,i) = PSD;
end

fprintf('Spectral analysis of the 4 hours completed. Spectrogram generated\n');

figure('name','Spectrogram');
fAxis = 1/N:Fs3/N:Fs3/2;
tAxis = 1:240;
surf(tAxis, fAxis, PSDArray); % A spectrogram is a 3D plot of the PSD looked at from above
hold on
view(2); % To look at the data from above
plot3(1:length(HRav), HRav./(HRav(1)/1.05), ones(length(HRav)), 'y', 'LineWidth',2)
title('Spectrogram of ECG');
xlabel('Time (min)');
ylabel('Frequency (Hz)');
colorbar % Shows the z values for every color, useful to determine the minimum
% Peak threshold for the algorithm
hold off
```

```
    saveas(gcf,'Spectrogram.fig');


    % Step 8: Apnea-Detecting Algorithm


    % Analyizng the data:
    % Uncomment this section if you'd like to try it but make sure to
    % include the text file a01Labled.txt that I attached in my submission
%     fprintf('Analyzing the data...\n');
%     figure('name','Spectrogram Analysis');
%     fAxis = 1/N:Fs3/N:Fs3/2;
%     tAxis = 1:240;
%     surf(tAxis, fAxis, PSDArray); % A spectrogram is a 3D plot of the PSD looked at from above
%     hold on
%     view(2); % To look at the data from above
%     plot3(1:length(HRav), HRav./(HRav(1)/1.05), ones(length(HRav)), 'y', 'LineWidth',2)
%     title('Analyzing the Spectrogram of ECG');
%     %legend({'', 'Average HR'});
%     xlabel('Time (min)');
%     ylabel('Frequency (Hz)');
%     colorbar
%
%     FID = fopen("a01Labeled.txt");
%     raw = fread(FID);
%     labeled = [];
%     j = 1;
%     for i = 1:length(raw)
%         if (raw(i) == 65)
%             labeled(j) = 1;
%             j = j + 1;
%         end
%         if (raw(i) == 78)
%             labeled(j) = 0;
%             j = j + 1;
%         end
%     end
%     labeled = labeled(1:240);
%     plot3(1:length(labeled), labeled.*1.08, ones(length(labeled)), '_w')
%     clear raw;
%     hold off
%
%     saveas(gcf,'SpectrogramAnalysis.fig');
%
%     figure('name','Analyzing HR Variation');
%     plot(tAxis, HRav);
%     hold on
%     title('Analyzing Average HR')
%     xlabel('Time (min)');
%     ylabel('Heart Rate (bpm)');
%     plot(labeled.*68, '.');
%     hold off
%
%     saveas(gcf,'HRAnalysis.fig');


    % After analyzing the data for a01.dat, a minimum frequency threshold is
    % about 1.1 Hz and minimum HR threshold is 68

    fprintf('Apnea detection in progress...\n');

    % Algorithm:

    predictions = [];
    for i = 1:length(HRav)
        % Seeing if there are any peaks in the PSD below the threshold
        % requency
        belowLinePeaks = findpeaks(PSDArray(1:65, i), 'MinPeakHeight', 0.002); % Index 65 corresponds to
the frequency 1.1 Hz which was decided on after analyzing the data
```

```
        [msg id] = lastwarn;
        warning('off',id) % Suppressing the warning message when the findpeaks finds no peaks
        if (HRav(i) <= 68 || any(belowLinePeaks)) % This threshold for HRav was decided upon after
visually analyzing the data
            predictions(i) = 1;
        else
            predictions(i) = 0;
        end
    end
    [msg id] = lastwarn;
    warning('off',id) % Suppressing the warning message when the findpeaks finds no peaks


    % Measuring Prediction Accuracy:
    % Uncomment this section and the other commented section above to try it out
%     figure
%     plot(predictions, '.');
%     hold on
%     plot(labeled.*2, '.');
%     hold off
%     numSame = 0;
%     for i = 1:240
%         if (labeled(i) == predictions(i))
%             numSame = numSame  + 1;
%         end
%     end
%     accuracy = (numSame / 240)  * 100;
%     fprintf('The accuracy of generated labels is %.2f%%\n', accuracy);

    % Printing into File:

    FID = fopen("ApneaPredictions.dat", 'w');
    fprintf(FID, 'Time(min): Prediction:\n');
    for i = 1:240
        if (predictions(i) == 1)
            prediction = 'A';
        else
            prediction = 'N';
        end
        fprintf(FID, '%d %c\n', i, prediction);
    end
    fclose(FID);
    fprintf('Successfully completed execution of program. All Graphs saved into your system...\nHave a
great day!\n');
end
```

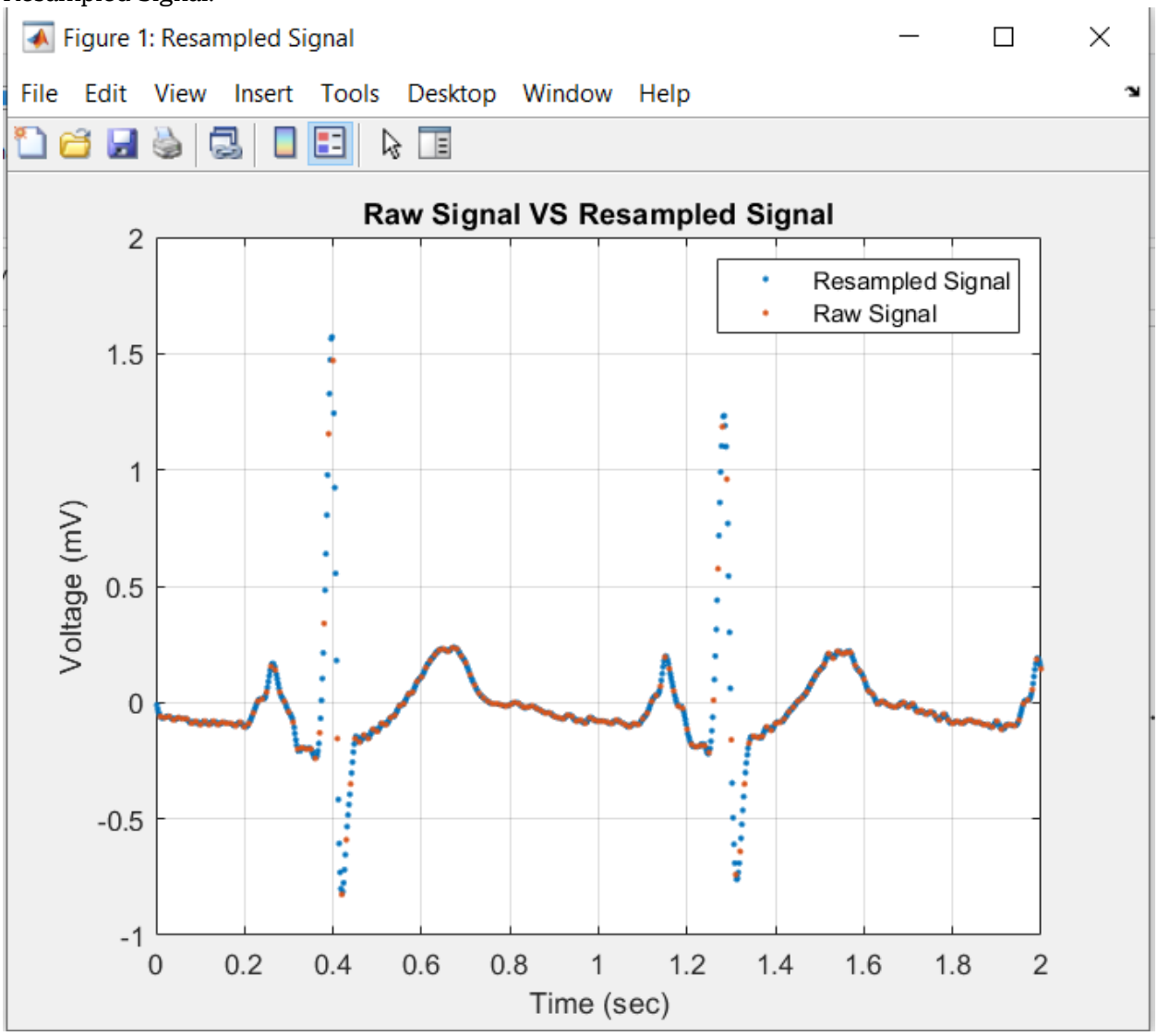
Step 5 – Tests & Verification:

Raw Signal:



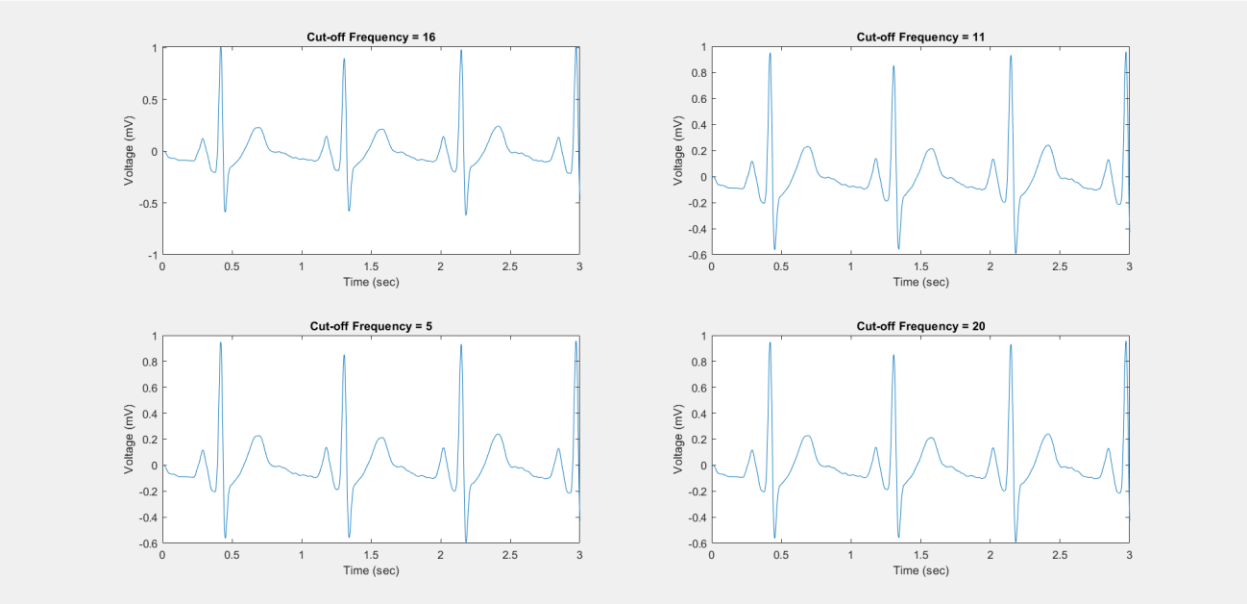
Data looks as expected with the PQRST complex clearly shown => This figure passes

Resampled Signal:



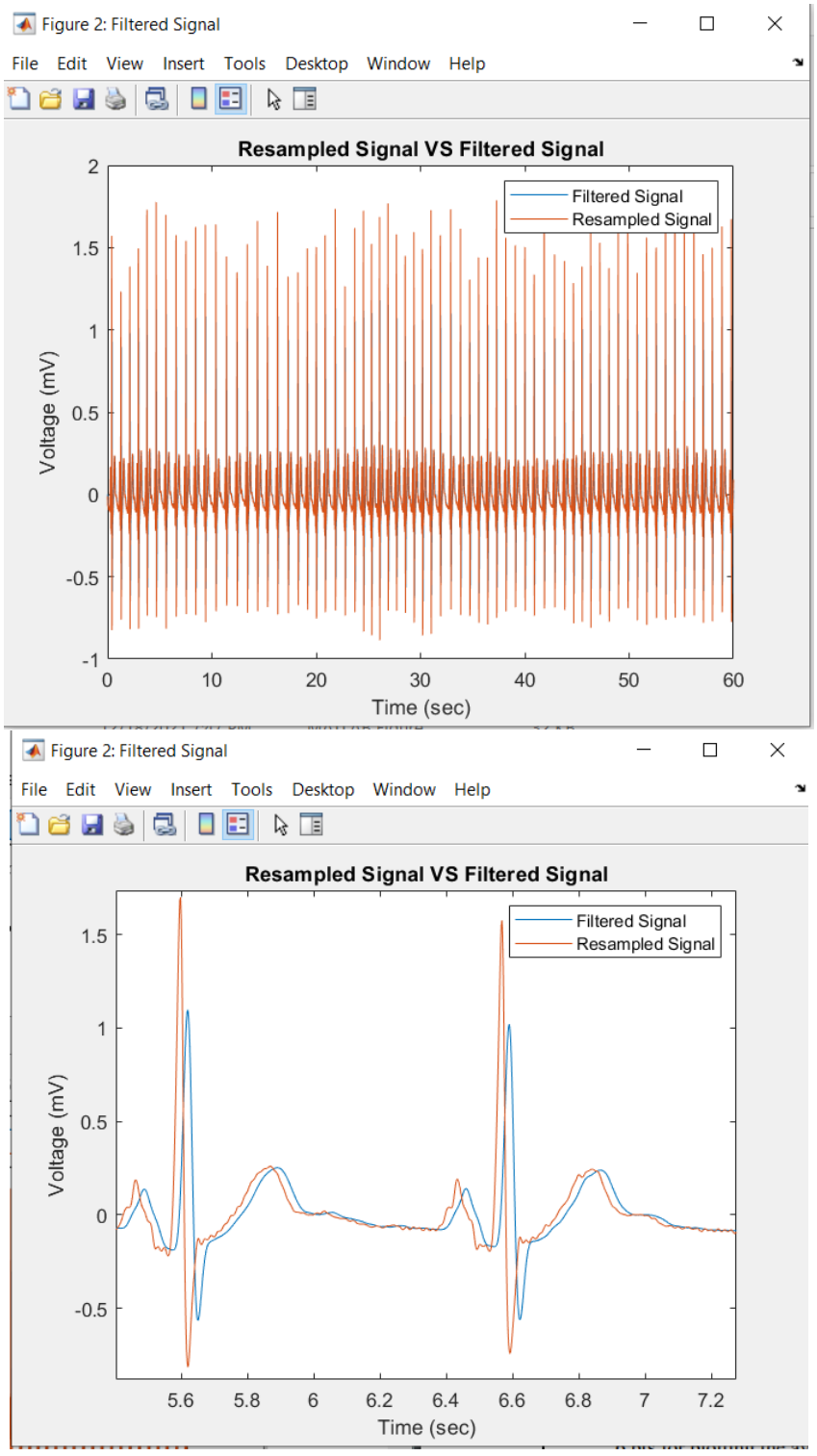
Data looks as expected with the 5 times as many datapoints for the resampled signal as the number of data points in the raw signal => This figure passes

Deciding on the Cut-off Frequency:



All 4 cut-off frequencies worked well so I just settled on 16 Hz in the final implementation of the program.

Filtered Signal with Cut-off frequency = 16(Second image is zoomed in):



The filtered signal looks much smoother yet maintains the PQRST structure.

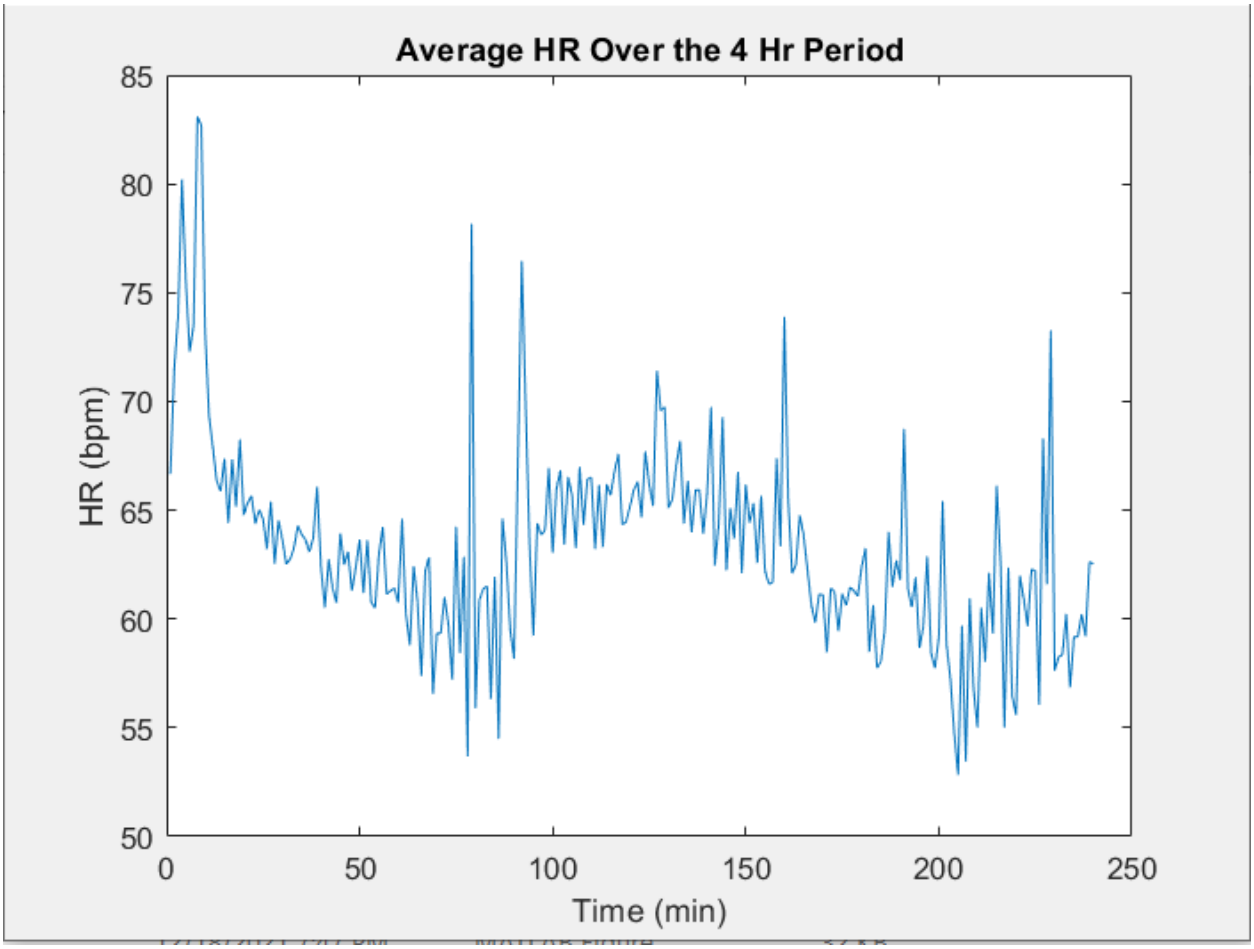
First Minute Heart Rate Analysis:

```
First minute Heart-Rate information analyzed:
Min Heart-Rate: 59
Max Heart-Rate: 78
Average Heart-Rate: 67
```

Reasonable HR values with MinHR < HRav < MaxHR as expected

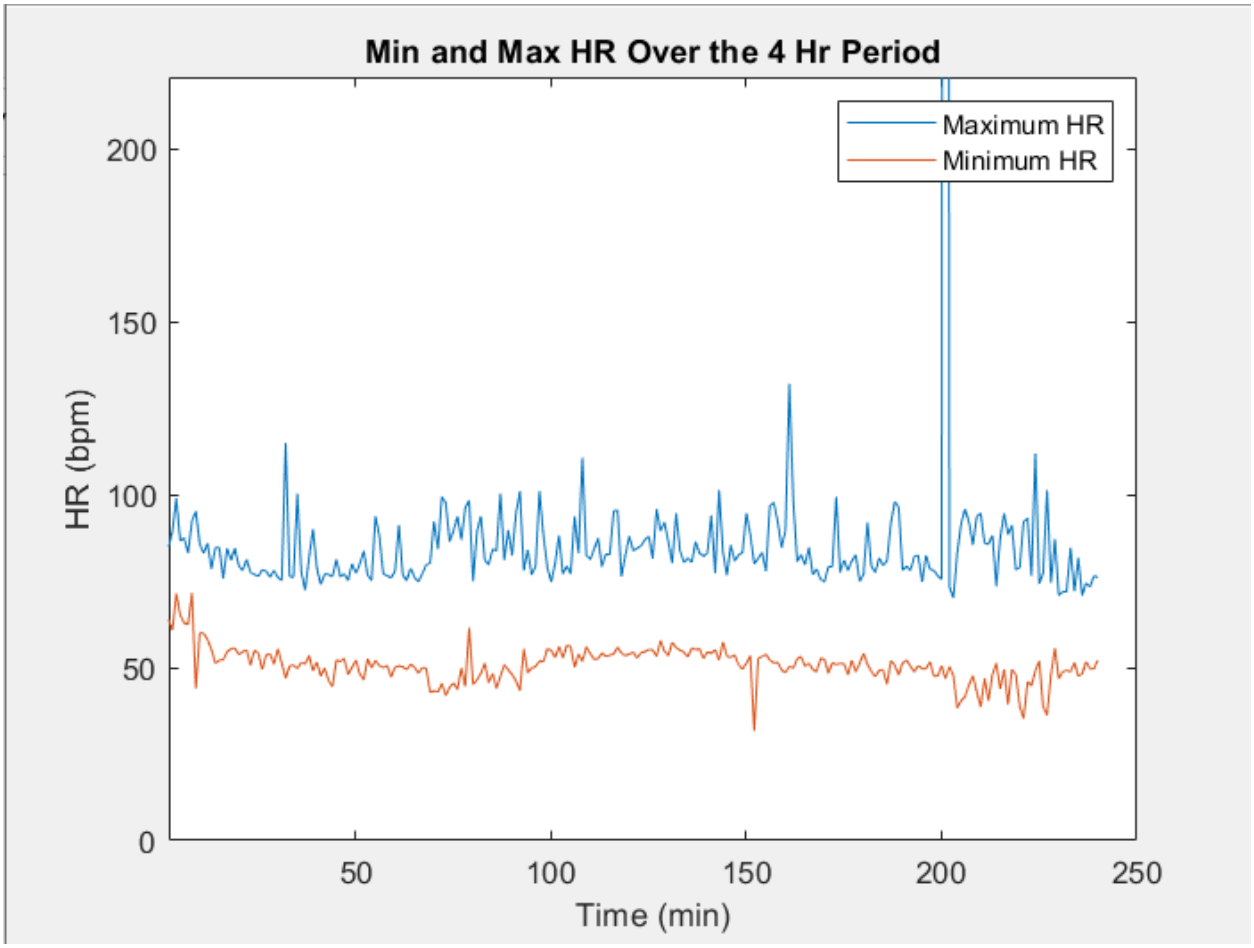
Heart Rate Analyis Over 4 Hrs:

HRav:

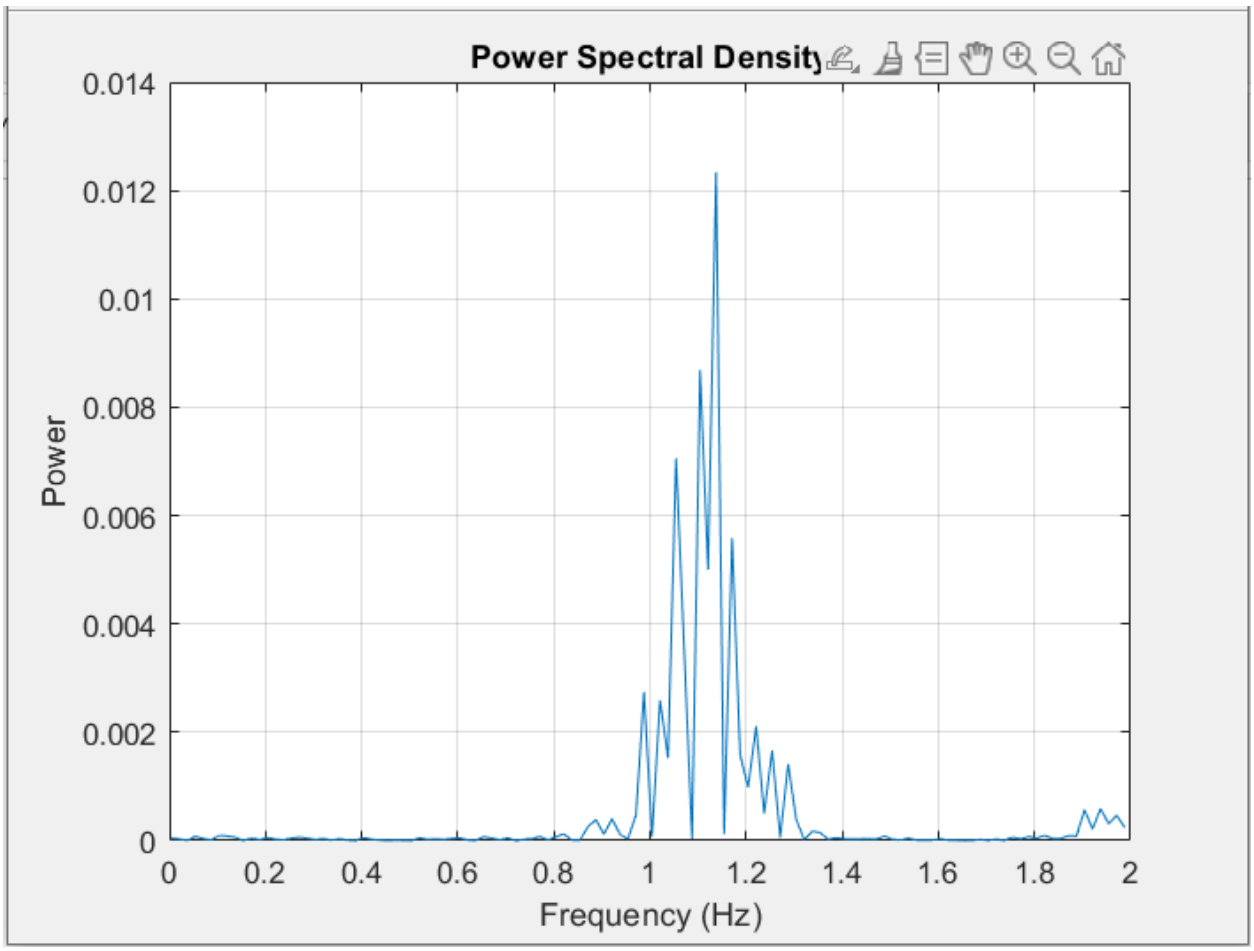


The values of HR make sense (within normal human HR ranges) => This functionality passes

MinHR and MaxHR (Extra graph):

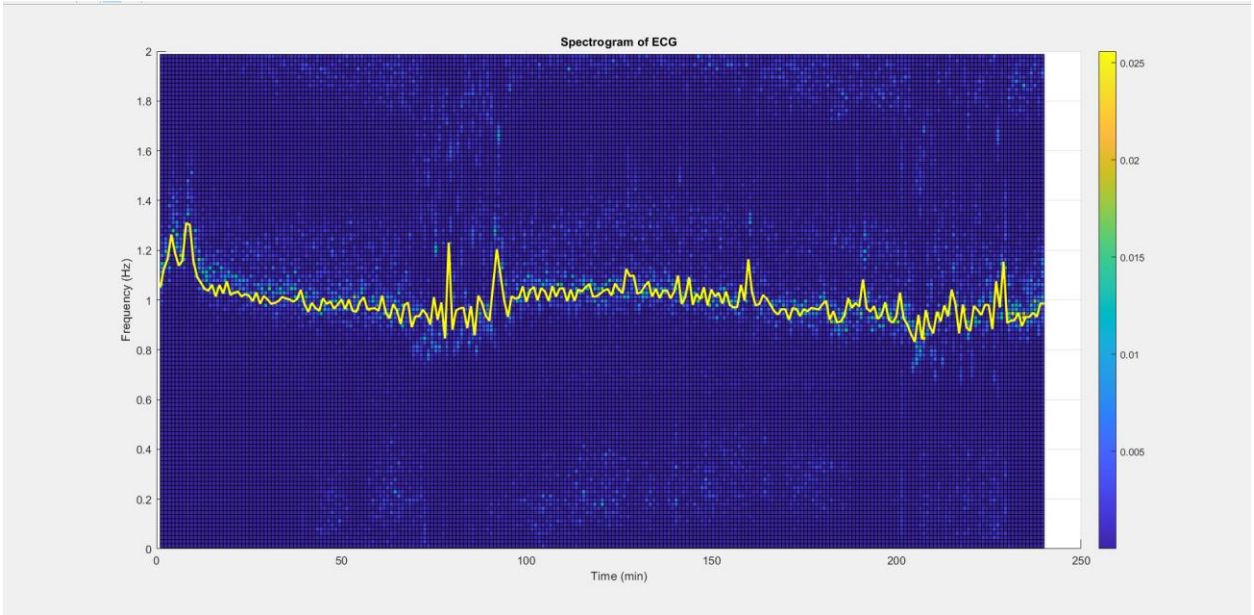


Power-Spectral-Density for First Minute:



There is 1 major peak at frequency ~1.13Hz and it only shows the positive frequencies from 0 to Fs/2 = 2 => This functionality passes

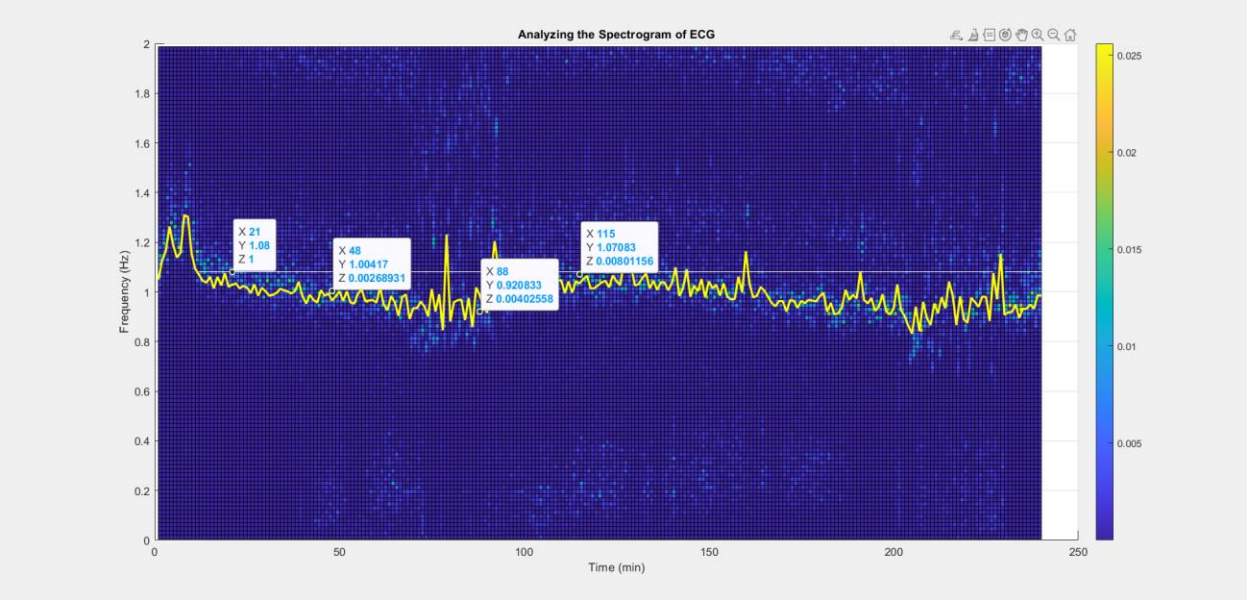
Spectrogram:



The HR variation matches almost perfectly with the spectrogram & the spectrogram looks as it should => This functionality passes

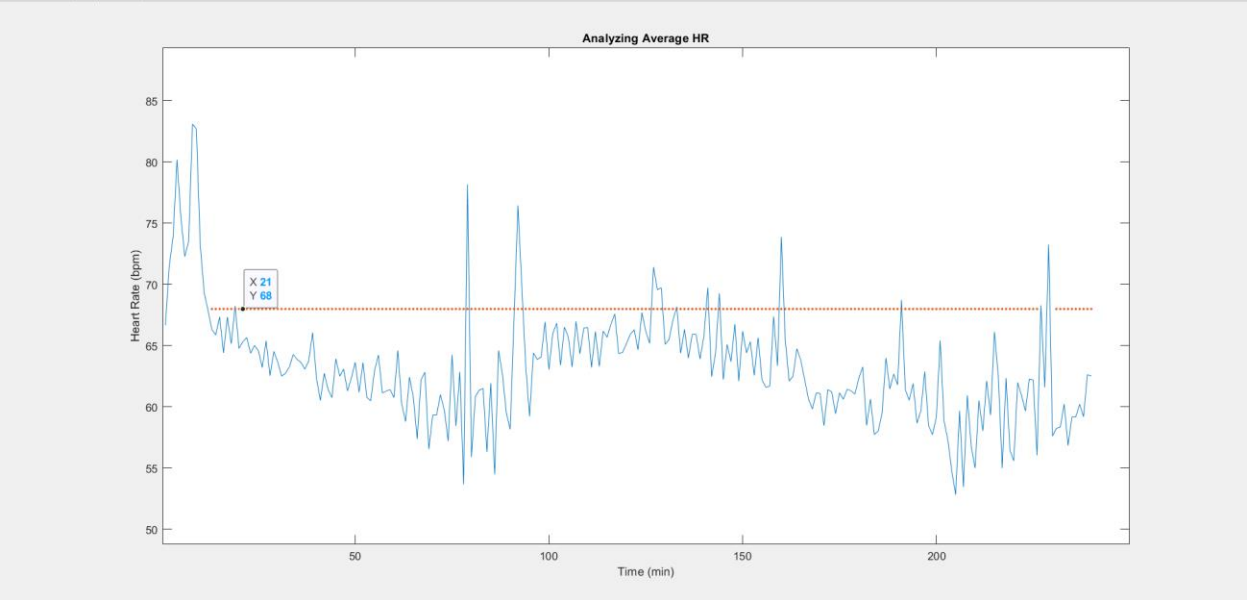
Deciding on the Values to Use (Analyzing the Data):

Spectrogram Analysis:



Apnea was happening whenever the frequency was less than around 1.1Hz (with a few exceptions). Thus, we need to check if there were any peak power values (above z=0.002 as shown by the data points in the figure) for frequencies below 1.1 Hz.

Heart Rate Variation Analysis:



Apnea was happening whenever the heart rate was less than around 68 (with a few exceptions). Thus, we will be setting that as the minimum threshold for non-apnea values

Apnea Detection Algorithm:

	Number	Categorical
1	Time(min):	Prediction:
2	1	A
3	2	N
4	3	N
5	4	N
6	5	N
7	6	N
8	7	N
9	8	N
10	9	N
11	10	N
12	11	N
13	12	A
14	13	A
15	14	A
16	15	A
17	16	A
18	17	A
19	18	A
20	19	N
21	20	A
22	21	A
23	22	A
24	23	A
25	24	A
26	25	A
27	26	A

…

Accuracy of Apnea Detection:

```
The accuracy of generated labels is 95.42%
```

User Guide:

- Download all the file in the submission (a01.dat, a01Labeled.txt, findRIndices.m, Assignment5.m) into the same folder
- Download the Signal Processing Toolbox for MATLAB (necessary for using designfilt)
- Run the Assignment5.m file in MATLAB (make sure your directory is the one you saved the files in)
- Uncomment the indicated sections if you'd like to see more of the data analysis sections. The uncommented code does all the required sections, but the commented sections give more data analysis if you'd like to check that out.
- All the graphs will be generated and saved to the folder you are currently working in.