BME 154L: Final Project

Professor Palmeri

May 1st 2012

Group 22

Max Li

Diane Shao

Hal Zhang

Peter Zhu

Students affirm their commitment to uphold the values of the Duke University community by signing a pledge that states:  
1. I will not lie, cheat, or steal in my academic endeavors, nor will I accept the actions of those who do.  
2. I will conduct myself responsibly and honorably in all my activities as a Duke student.

hz43, mzl, dys2, pyz

**Problem 1 (Code Used FinalProject1.m):**

Background:

Ultrasound transducers are all designed differently with different center frequencies. These different frequencies could potentially lead to differing spatial resolution and depth penetration and therefore is very important to determine. The transducers also contrast in their impulse responses. This could potentially lead to different image qualities despite similar center frequencies. In this problem the differences of four ultrasound transducers and how these differences affect the output will be explored.

* The power spectrum was plotted and is shown below in Figure 1.1. The spectrum was calculated by simply using the fft command in MATLAB and the fftshift command in order to center the frequency response around zero. Finally 20\*log of the frequency response was done.

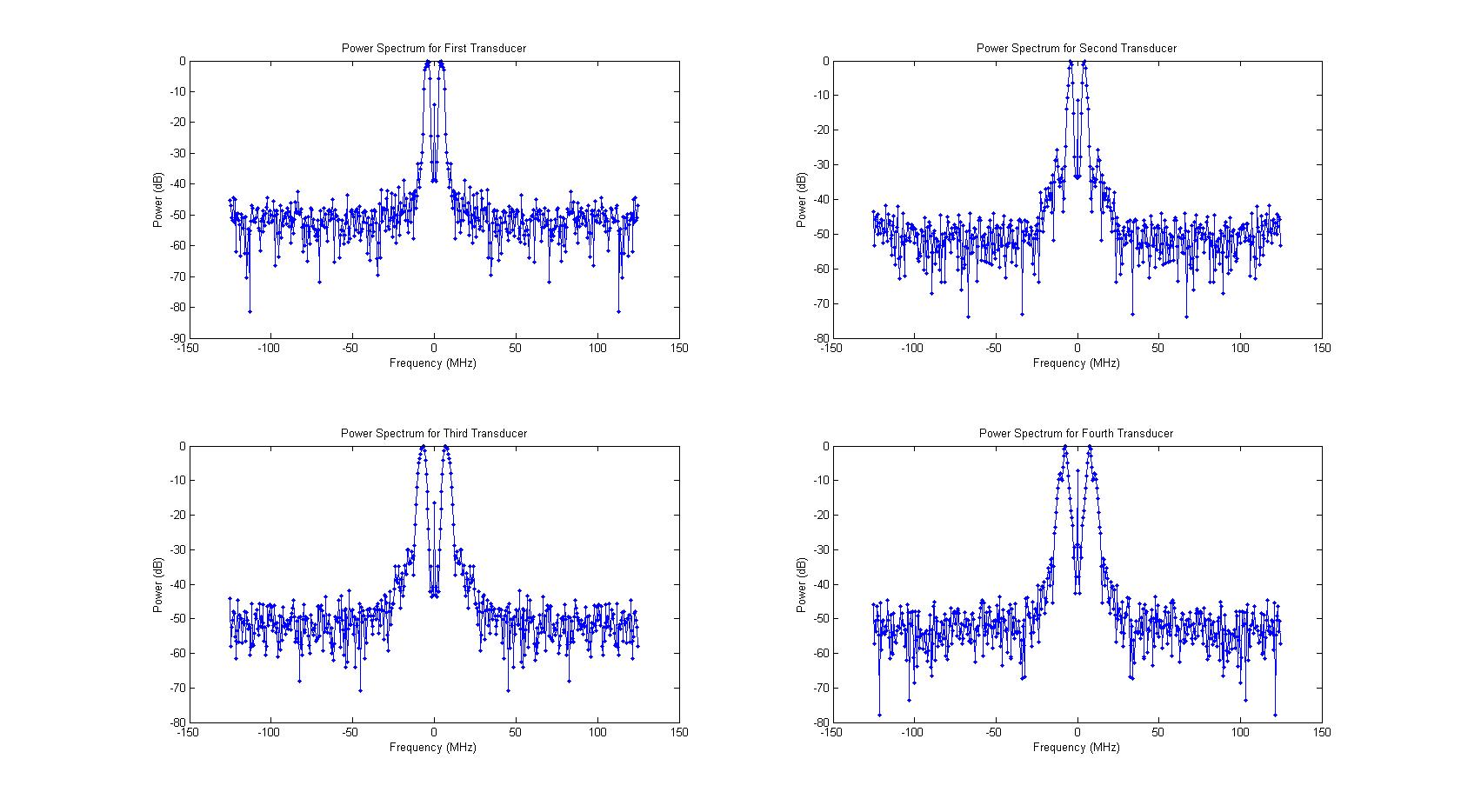


Figure 1.1: Power Spectrum of All Four Transducers

* The -3dB intersections for the graph now needs to be found. This was done by first cutting off all of the negative frequencies because that is simply a reflection of all the positive frequencies. The two -3dB points lie on different sides of the maximum frequency and therefore the index of the peak frequency was found to separate the plot into two halves with each containing one of the -3dB points. Now because the plots in MATLAB have a finite amount of points, that means that there won’t be a frequency point exactly where the -3dB power occurs therefore interpolation of the graph was used. The interpolation function in MATLAB has multiple options and the linear one was used because the graph near the -3dB points basically looks linear. The -3dB frequencies and the center frequencies found are listed below in Table 1.1.

Table 1.1: -3dB Frequencies, Center Frequencies, and Fractional Banwidths of the Four Transducers

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | First -3dB Frequency (MHz) | Second -3dB Frequency (MHz) | Center Frequency (MHz) | Fractional Banwidth |
| Transducer 1 | 3.16 | 5.97 | 4.34 | 0.65 |
| Transducer 2 | 3.33 | 5.10 | 4.12 | 0.43 |
| Transducer 3 | 5.73 | 8.76 | 7.09 | 0.43 |
| Transducer 4 | 6.35 | 8.52 | 7.36 | 0.30 |

The center frequency was calculated with the geometric means of the first and second -3dB points and the fractional bandwidth was calculated with the equation given in the handout and shown below.

* The 2 volt peak to peak, 1 cycle sinusoid at the center frequency was plotted and convolved with the impulse response of each transducer in order to obtain the outputs. When convolving in MATLAB, the number of points in both the impulse response and in the input matters a lot. Therefore in this situation a trick was used in order to get the desired output. The time step in the impulse response was maintained in the sinusoidal input and this helped make the output graph look a lot more reasonable. The graphs for one cycle excitations are shown below in Figure 1.2.

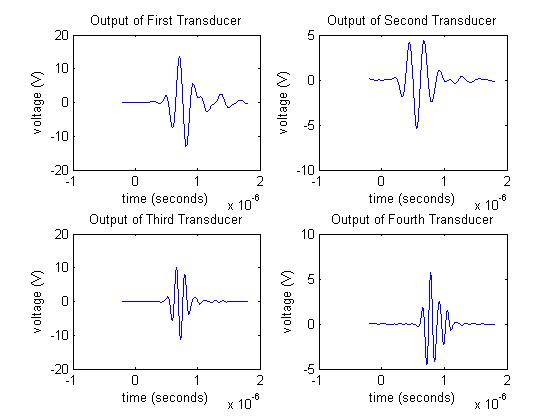


Figure 1.2: Outputs of All Four Transducers

* For this particular set of data, the number of cycles needed to reach steady state amplitude is shown below in Table 1.2.

Table 1.2: Cycles Needed to Reach Peak to Peak Steady State

|  |  |
| --- | --- |
|  | Cycles Needed |
| Transducer 1 | 13 |
| Transducer 2 | 13 |
| Transducer 3 | 22 |
| Transducer 4 | 20 |

We came to this conclusion by using the find peaks command in MATLAB in order to find the values of very peak and using the diff command to find the difference of all of these peaks. Once the difference was found, the maximum magnitude was taken and compared to 1% of the maximum peak. If the difference was less than 1% of the maximum peak then the output response was considered to be in steady state.

Once these cycles needed were found for each transducer then they had to be correlated to an intrinsic property of the transducer in order to be able to find the cycles needed for any arbitrary transducer. The first thing we thought to try was to correlate the cycles needed with the fractional bandwidth. This gave us the graph below in Figure 1.3.

Figure 1.3: Correlation between Fractional Bandwidth and Cycles Needed

As one can see, the R2 value is pretty poor so instead we decided to try correlating the cycles needed with the center frequency, another intrinsic property of the transducer. This gave us the graph in Figure 1.4.

Figure 1.4: Correlation between the Center Frequency and Number of Cycles

Now this graph shows a pretty decent correlation between the center frequency and the number of cycles needed to reach steady state. Therefore the expression to find the number of cycles needed to reach steady state for any transducer is simply

If the equation above produces a decimal then try the two integers that are closest to the answer.

**Problem 2 (Code Used: FinalProject2Quad.m, FinalProject2DepthDep.m, FinalProject2Carotid.m, and FinalProject2TGC.m):**

Background:

The ultrasound transducer can both send and receive signals. It is initially excited by a voltage from a transmitter and transduces this into a set frequency sound wave. The sound wave enters into the body and are reflected and scattered by the tissues and organs then they return to the transducer. Once they return, the transducer can now convert the sounds wave back into voltages and these measurements become the B mode data given to us. The B mode data however needs processing in order to increase the contrast and spatial resolution of the image. This processing comes from evaluating the envelope of the signal by using either quadrature demodulation or the Hilbert transform. When using the quadrature demodulation, we had to account for the change in center frequency of the received signal with respect to depth.

The second part of this problem involved producing an image of a coronary artery and using log compression in order to improve visualization. Another method used in order to help with the visualization and contrast is using the imadjust function in MATLAB. Finally the depth dependent attenuation of the signal was eliminated from an image of multiple lesions by simply normalizing the whole image to the brightest part of the background.

The first two bullets used the MATLAB code: FinalProject2Quad.m

* The image of the radio frequency data was plotted using imagesc of the RF data and is shown below in Figure 2.1:

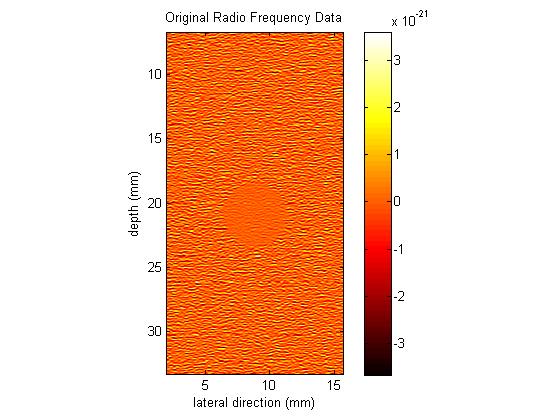


Figure 2.1: Original Radio Frequency Data

* The quadrature demodulation code works as follows. Initially the Fourier Transform of all the columns of the RF data was taken in order to find the center frequencies of each column because each column represents an individual A-line and they should all have different center frequencies. The center frequencies were found for all of these columns by looking at where the largest power occurred in the first half of the data. We looked at the maximum power for center frequencies and not the geometric mean of the -3dB frequencies because there were too many lines on the graphs that crossed the -3dB value and it would have been very hard to get a good estimate. Only the first half of the data was used because fftshift was not used in this situation meaning that the second half of the data would have been just a mirror image of the first half; the smaller center frequency made more physical sense. The average value of all the center frequencies was about 6.83 MHz well within the range of 1-15 MHz that ultrasound transducers tend to have given in Problem 1.

Once the average overall center frequency was found, the signal was multiplied separately by both sines and cosines at this center frequency. The difficulty in this part was finding the time data that actually goes with the ultrasound data. The depth data given to us with the RF data was used to calculate the amount of time elapsed in order to measure every single depth point. The equation used was just . Once that was done, the frequency response was again plotted and a low pass filter was used to get rid of all the frequencies not centered around 0 Hz. The low pass filter was simply a rect having a cutoff of around where the first peak centered at 0 Hz ended. Both the sines and cosines were filtered out like that and the inverse Fourier Transform was taken; the envelope of the original signal was finally reconstructed by multiplying 2 by the square root of both the sine and the cosine component. This gave a graph that looked much better than the graph above. This is shown in Figure 2.2.

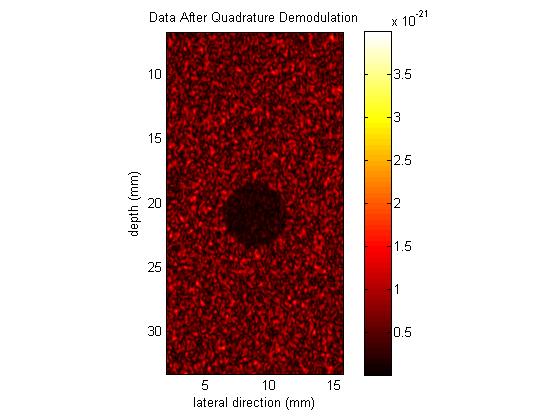


Figure 2.2: Image after Quadrature Demodulation

The log compressed data was also tested to see if it would help better visualize the lesion better. This plot is shown in Figure 2.3.

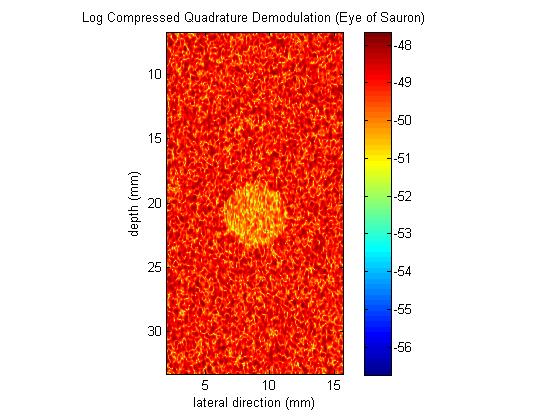


Figure 2.3: Log Compressed Image after the Quadrature Demodulation

The contrasts of all three different lesions were calculated by using the formula below:

This formula was used because we wanted a good way to normalize the difference between the intensity of the lesion and the intensity of the background. We thought that the whole intensity range would be the best way to normalize because this accounts for MATLAB plotting the both the lesion and the background along the intensity range (the scales are automatically set this way). The contrasts calculated are shown below.

Table 2.1: The Contrast of the Three Methods Mentioned Above

|  |  |  |  |
| --- | --- | --- | --- |
|  | Normal Signal | Quad Demod | Log Compressed |
| Contrast | 4.8574e-004 | -0.1894 | -0.1098 |

The signs do not matter because it simply tells us whether the intensity of the background or of the lesion is larger. It can be seen from these calculated contrasts that the quadrature demodulation clearly increased the contrast of the image while log compressing the quadrature demodulated data actually caused the contrast to decrease even though it did make the picture look like the Eye of Sauron.

In these images, the contrast looks to be directly proportional to the spatial resolution. Because qualitatively, it would be extremely hard to tell between two points in Figure 2.1 while in Figures 2.2 and 2.3, the speckle noise in the background can be clearly identified from each other. The speckles in Figure 2.2 do look a little bit clearer more distinguishable from each other and therefore it probably has better spatial resolution.

From all these contrasts calculations, it can be seen that physicians are more likely to use images that have been envelope detected because it produces an image that can be interpreted much more easily than one that has not. They will be able to tell between noise, background, and lesion and have a lesser chance of misdiagnosing or missing a tumor.

* The Hilbert transform was now used instead of the quadrature demodulation. The code was pretty simple and it only involved taking the absolute value of the Hilbert function of the original RF data. This produced a graph in Figure 2.4.

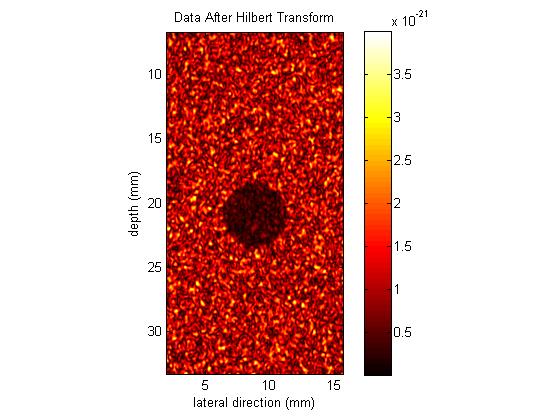


Figure 2.4: Signal after Hilbert Transform Envelope Detection

The image after the Hilbert transform actually looks pretty similar to the image after quadrature demodulation. The intensities do look different but the contrast for the Hilbert transform image was -0.1976 which is only slightly better than the quadrature demodulated image.

* The significant differences between the Hilbert Transform and the quadrature demodulation is that the quadrature demodulation requires the average center frequency of every single A-line of the RF data while the Hilbert Transform simply was a linear operator that changed the RF data into a set of real and imaginary numbers that represented the envelope of the data and could be reconstructed by simply finding the magnitude of these numbers; no center frequency or any frequency content was needed and no filtering was done.

Theoretically speaking, the quadrature demodulation should have produced “better” results because it accounts for both the signal’s center frequencies and for the time data of the signal however in our code, the Hilbert Transform worked better. This could be due to multiple things like not finding the center frequencies exactly right (we took the maximum). This could also be due to not low pass filtering at the right frequency or the added error that just comes from more calculations.

In this next optional part, the FInalProject2DepethDep.m was used.

* OPTIONAL

A MATLAB code was written to calculate frequency with respect to depth but what was seen was not a decrease in center frequency with respect to depth but a correlation that went up and down at random intervals. This can be seen in Figures 2.5 and 2.6.

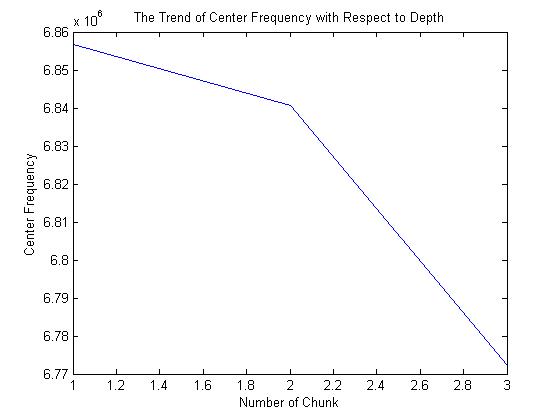


Figure 2.5: The Center Frequency of Every Third Chunk of Data

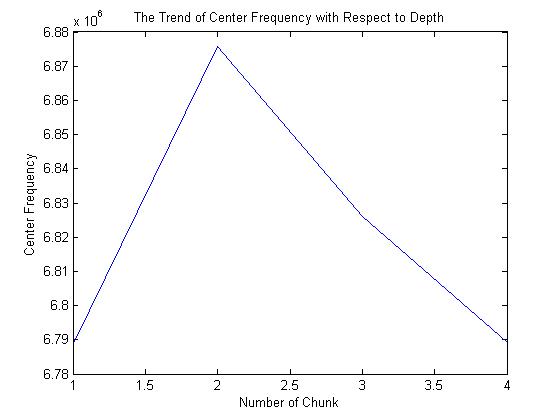


Figure 2.6: The Center Frequency of Every Fourth Chunk of Data

How this code worked is basically it split the RF data matrix into n (we used three) different matrices by rows. The center frequency of the first third chunk, second third chunk, and final third chunk of rows were measured and averaged (shown in Figure 2.7 below). This basically produced three different frequencies at the three different depths chosen and these three frequencies were then used for the sines and cosines in the quadrature demodulation method.

First Third

Second Third

Last Third

Figure 2.7: How the RF Data was Split

As can be seen from data plotted in Figures 2.5 and 2.6, the center of frequency does not really decrease with depth except for in the n=3 case and that was why we used that number of divisions.

There were multiple other ways that we tried but decided didn’t really work. We initially tried to find the center frequencies of every single column inside the row chunks instead of just simply averaging the center frequencies in the chunk of rows together, the contrast of this image was actually worse than the original image without using depth dependent frequencies. We also tried to ignore the lesion in the middle by only averaging the center frequencies of the first few columns but that did not work either. The image that had slightly better contrast is shown below in Figure 2.8.

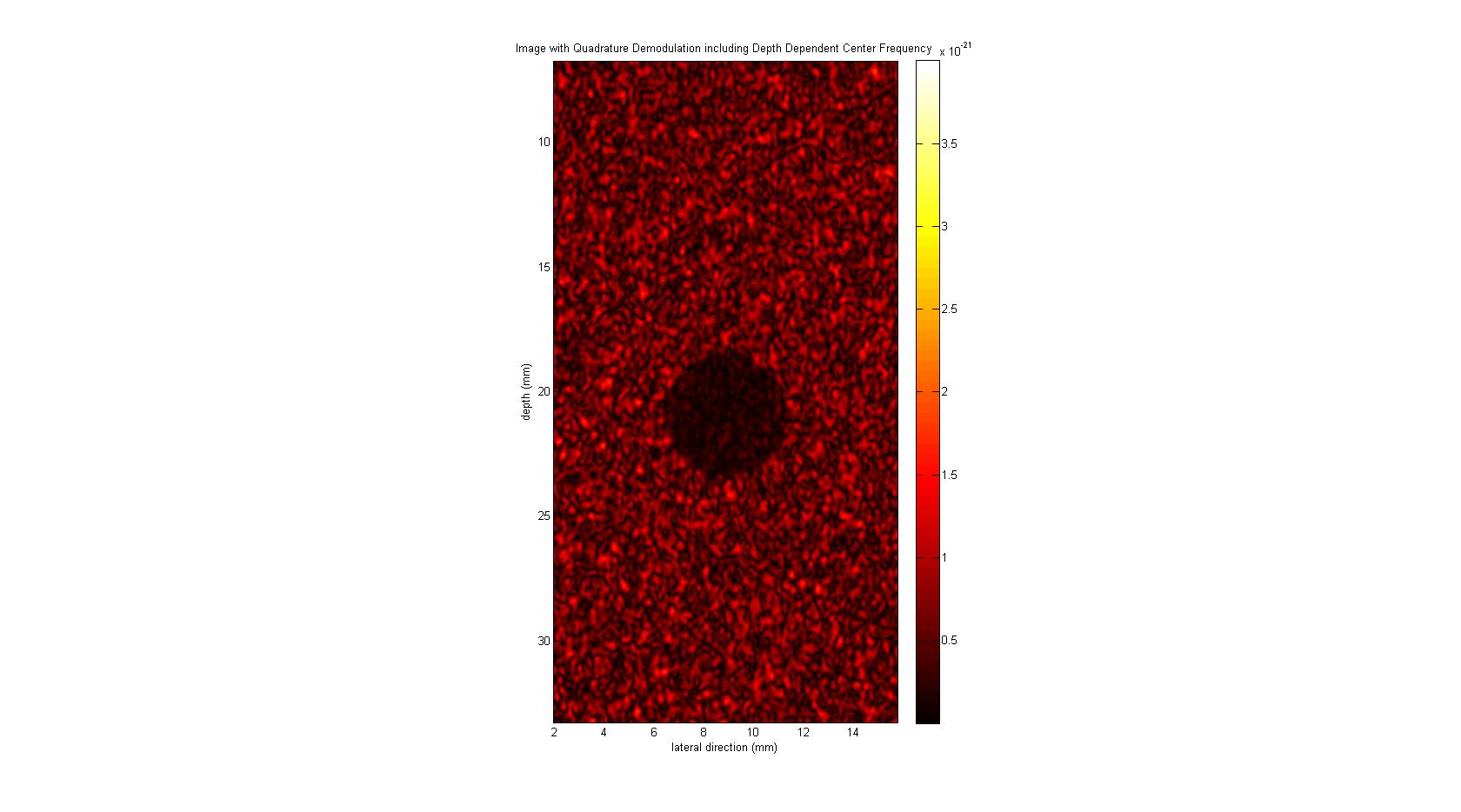


Figure 2.8: Image of Lesion with Quadrature Demodulation Done with Depth Dependence

The contrast produced from this plot was -0.1895 which is slightly better than the contrast in Table 2.1. It makes sense that the contrasts would be so close because the center frequencies actually do not vary that much with depth as can be seen in Figures 2.5 and 2.6 and therefore accounting for this issue should not change the image by too much.

The MATLAB code used for the next two bullets is FinalProject2Carotid.m

* The carotid artery data was stored in 16-bit integers and therefore had to be converted to double in order to do calculations in MATLAB. This conversion was simply done by using the double command in MATLAB. The problem with storing in 16-bit integer is that all the decimal values are lost and when converting back to double the decimals could not be recovered. The benefit of course is that int16 takes up less space than double (64-bit). After the conversion to double, the Hilbert transform method was again used in order to increase the contrast to the image. The non-log compressed image is shown below in Figure 2.9.

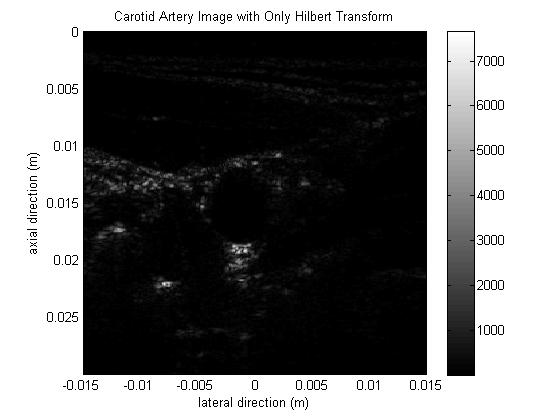


Figure 2.9: Image of a Carotid Artery with Only the Hilbert Tranform

The contrast does not look so good in the image above and the numbers in the colorbar look pretty large so log compressing the data would probably be a good idea. That image is shown below in Figure 2.10.

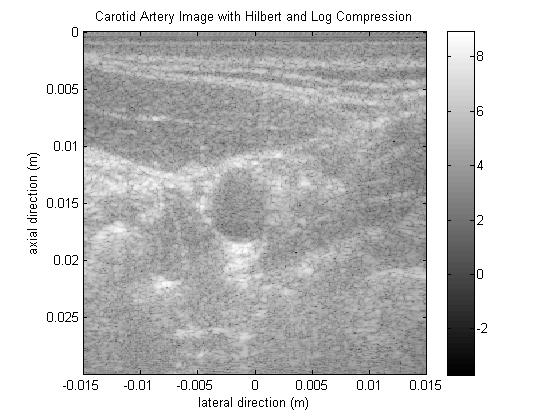


Figure 2.10: The Carotid Artery Image with Both the Hilbert Transform and Log Compression

This image above clearly looks better than the one in Figure 2.8 and the carotid artery opening can actually be seen apart from the background.

The log compression has such an effect on the image because it looks like in the first picture that the image is being dominated by very small intensities in the background and has some but very few large frequencies that represent the image. Taking the log of this will decrease the range dramatically and therefore increase the contrast. Instead of there being a dramatic difference between the low and high intensities, log compressing will pull them closer and also will help with distinguishing between the low intensity data because now they are not as different from the high intensity data. This contrast change can be seen at the end of the section in Table 2.2.

* OPTIONAL

We did some research into MATLAB commands and found that the imadjust function is actually very helpful in the situation with data that have very large ranges. What this function does is it basically takes the top and bottom 1% of the data and saturates them at the highest and lowest intensities respectively. This percentage can be changed by the user. This is really good because our data as shown by the histogram (Figure 2.11) below is highly skewed to the left.

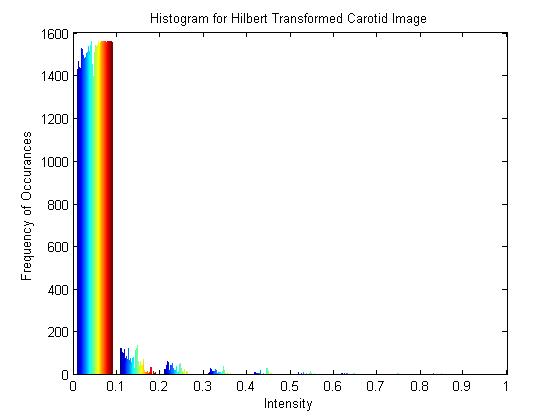


Figure 2.11: Histogram of Hilbert Transform Data

Using the fact that most of data is located under 0.1 intensity, we used the imadjust function in order to saturate all the high intensity data to 0.1 and therefore create much better contrast with the lower intensity data. However, the Hilbert Transform data needed to be processed before using imadjust. Imadjust only works on data that have intensity ranges from 0 to 1 therefore the intensity data was simply normalized by the largest intensity. The graph after the imadjust is shown in Figure 2.12.

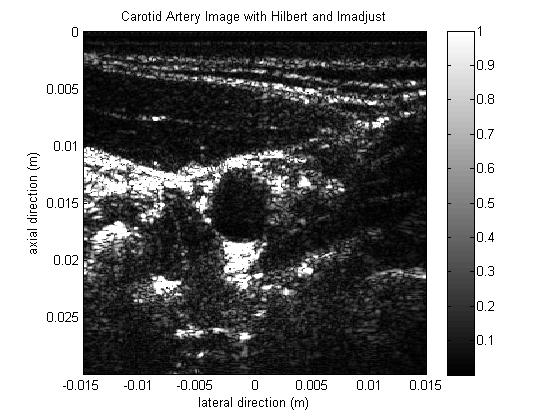


Figure 2.12: Carotid Image with Hilbert Transform and Imadjust

This looks to have even better contrast than the log compressed data because all the parts of the image that need to be bright are brighter due to the saturation and the low intensity data stayed the same but look much darker due to the comparison to much brighter data. The contrast numbers are in Table 2.2.

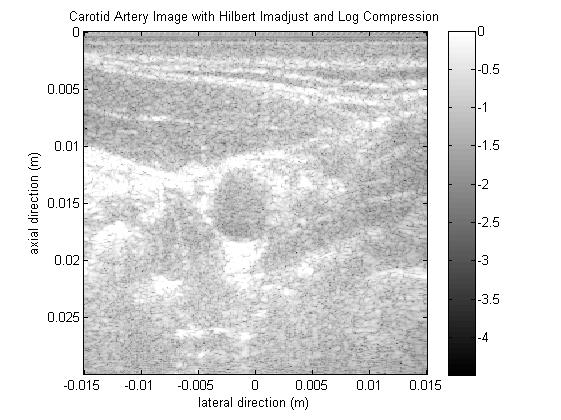


Figure 2.13: Carotid Image with Hilbert Transform Imadjust and Log Compression

The imadjust method could be done with the log compression together but the image (Figure 2.13) does not look as good as the one just with imadjust because it seems to pull the brightness of the higher intensities parts closer to the brightness of the low intensity parts thereby decreasing the contrast. The contrast of this image was almost not as good as the log compressed image of the Hilbert transformed carotid image (Table 2.2).

To actually quantify the contrast changes, the contrasts for the four different methods are shown below.

Table 2.2: Contrast for the Four Different Methods Described Above

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Hilbert | Log(Hilbert) | ImAdjust | Log(ImAdjust) |
| Contrast | -0.1326 | -0.2400 | -0.8152 | -0.2606 |

The main difference between the imadjust method and the log compression method is that imadjust physically brings all the high intensities a lot closer to each other and leaves all the low intensities at very low values. This helps the high intensity data to really stand out from the background intensities. The log compression however follows a formula that can only scale down high intensities up to a point but also pulls closer the high to low intensities. That is why log compression does not work so well with images that already have pretty low intensity ranges. Rather than improving the contrast, it will actually pull the high and low intensity data together ruining the contrast.

The MATLAB used for the section below is FinalProject2TGC.m

* OPTIONAL

The intensity of the signal can drastically decrease with respect to depth as seen in Hilbert transformed version of the original TGC image given in this problem as shown below in Figure 2.14.

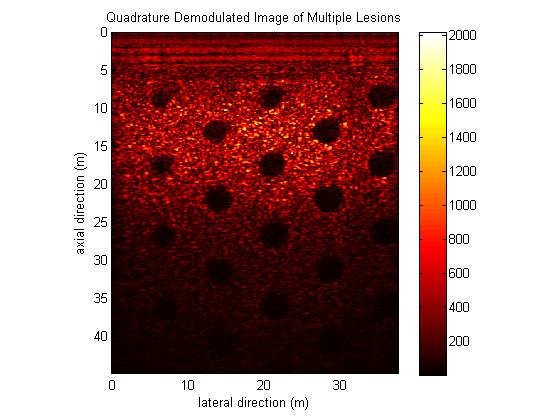


Figure 2.14: Quadrature Demodulated Image of Multiple Lesions

Because the bottom part of the matrix looks so low in intensity as compared to the top, the data was normalized with a constant in order to make the image at a more uniform intensity. The normalization constant was calculated by finding the average across all the different depths (averages across different rows) and finding the maximum average intensity. Then the normalization constant was simply the maximum average intensity divided by the average of every single row. Multiplying this by the original data, both the background and the lesion intensity were normalized and the image is shown below in Figure 2.15.

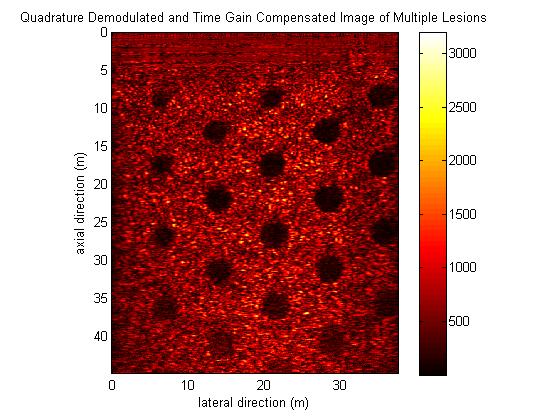


Figure 2.15: Time Gain Compensated Image of Multiple Lesions

The contrast calculation for these images were done the same way as above but the image in Figure 2.14 clearly do have regions of better and worse contrast and the worst contrast value was taken to show how big of a change the time gain compensation did.

The values are for the contrast are -0.0379 and -0.0712 for the non –time gain compensated and for the time gain compensated respectively. There is a drastic increase (2 time increase) in contrast but the absolute magnitudes are very small. This is due to the definition of contrast that we proposed earlier. In this data set, the range of data points is actually very large in comparison to both the intensity of the background and the intensity of the lesion. Another better definition could have been used but we wanted to be consistent with before and also we do see a dramatic relative improvement in contrast. This improvement in contrast can also clearly be seen in both images and this will definitely help with lesion detection in deeper areas of the body. The circles (lesions) can actually be seen in Figure 2.15 at lower depths but cannot be seen at all in Figure 2.14 as they blend in with the low intensity background.

**Problem 3 (Code Used: mmodecorrv.m and dopplershift.m) :**

**Background**

M-mode ultrasound acquires RF lines at the same spatial locations over sequential times. The pulse can be characterized by the sampling frequency defining the axial resolution, the pulse center frequency defining the range of depths of the images and the pulse repetition frequency prf, defining the temporal resolution. M-mode ultrasound images are often used to calculate the flow rate of inhomogeneous liquids such as blood. Inhomogeneity of the liquid allows for contrast in reflected echoes as particles travel through the field of view and thus yields an oscillatory signal whose frequency may be used to determine the velocity of the motion. Only the component of the direction of flow parallel to the RF beam, often represented by the sign of the velocity, can be determined from one M-mode image.

**Method**

Velocity of a motion is calculated using two distinct methods; the first method is cross correlating subsequent RF lines with that at the first time point and finding the lag time in peaks compared to the peak present in the autocorrelation of the first RF line. The slope of the linear relation between the time of each RF line and the lag time of the cross correlation peaks is the velocity of the motion. Furthermore, the direction of flow is represented by positive, traveling towards the detector, and negative, traveling away from the detector. This sign convention is derived from the fact subsequent RF lines are shifted forward or backward in phase of position which can be quantified through cross-correlation. It is important to note MATLAB’s definition of the cross-correlation function are chiral to that presented during lecture, though the consistency of signs given direction of motion is all that needs to be maintained to verify the algorithm.

In the code, the data is loaded and the axial and time resolutions are calculated by finding the slope of the linear fit of the axial data and time data. A for loop is used to cross correlate the first RF line against subsequent ones to produce a matrix of cross-correlation data points. The ‘find’ command is used to find the index of the highest peak. The indices are plotted against time and the slope is found in terms of axial and time steps. Velocity is then calculated by multiplying the slope by the ratio of the axial and time values of each step.

The second method for calculating the velocity uses principles of Doppler shift. The center frequency of each time series data is extracted and averaged together. The velocity is calculated by the following formula

Where c is the speed of sound in the material, is the center frequency of the ultrasound pulse and is the center frequency of the data. The direction of motion can be identified by the direction of shift in the center frequency of the axial spectrum with respect to . This is because when RF pulse reflects off a moving object, the echo either red or blue shifts depending on whether the object is moving away or towards the transducer. This shift is reflected in a decrease or increase of the center frequency.

The annotated codes used for this section are (mmodecorrv.m and dopplershift.m).

**Results and Discussion**

Plotting the M-mode images directly (Figure 3.1) we can see that datasets A and B shows motion in the one direction and dataset C shows motion in the other.

|  |
| --- |
|  |
| **Figure 3.1.** M-mode images of dataset A, B and C. |

Figures 3.2 – 3.4 shows a comparison of the RF lines at t = 0 and t = 0.320s. The x-axis represents depth in meters while the y-axis represents the magnitude of the echo of the sound waves. Notice the frequency and amplitude of the echoes stay relatively similar even when velocity varies.

|  |
| --- |
|  |
| **Figure 3.2.** RF lines in dataset A |

|  |
| --- |
|  |
| **Figure 3.3**. RF lines in dataset B |
|  |
| **Figure 3.4**. RF lines in dataset C |

Figures 5 – 7 shows comparison of the autocorrelation of the first RF line with the cross correlation of the first RF and that at t = 0.0320s. Direction can be differentiated by the relative location of the cross-correlated peak in reference to that of the auto-correlation.

|  |
| --- |
|  |
| **Figure 3.5**. Correlation peaks (at the red lines) for dataset A |

|  |
| --- |
|  |
| **Figure 3.6**. Correlation peaks (red line) for dataset B |

|  |
| --- |
|  |
| **Figure 3.7**. Correlation peaks (red line) for dataset C |

These phase shifts in position are plotted against their corresponding time to produce a linear relation (Figures 3.8 – 3.10) whose slope is the velocity (Table 1). Velocities from dataset A, B are of positive suggesting the motion is towards the transducer. In contrast, velocity from dataset C is negative, representing motion away from the transducer. Velocities A and C are equal in magnitude but of opposite signs.

|  |  |
| --- | --- |
| Table 1. Velocities calculated by cross correlation | |
| Dataset | Velocities |
| A | 9.62 cm/s |
| B | 3.21 cm/s |
| C | -9.62 cm/s |

|  |
| --- |
|  |
| **Figure 3.8.** Shift in depth through time for dataset A |

|  |
| --- |
|  |
| **Figure 3.9.** Shift in depth through time for dataset B |

|  |
| --- |
|  |
| **Figure 3.10.** Shift in depth through time for dataset C. |

**Calculating velocity by Doppler shift**

As mentioned in the methods section, the power spectrum of both the RF lines and the time series signal at specific depth can be used to calculate the velocity and direction. The Fourier transform of each time series signal at each depth is taken (figure 3.11 shows for dataset A) and the center frequency, identified as the frequency of the maximum signal, is averaged for each time series signal. Using the Doppler shift formula, velocity is the following

|  |
| --- |
|  |
| **Figure 3.11.** Power spectrum of each time series signal at each depth. The center frequencies are identified and averaged. |

**Calculating direction of motion**

The Fourier transform of each RF line is averaged and the center frequency is compared against the pulse center frequency . Figure 3.12 shows comparison of the center frequencies (red circle) with (red line). Dataset A and B shows a decrease in the center frequency suggesting the motion is away from the transducer while dataset C shows an increase in center frequency. This is consistent with visual inspection of the image as well as the results produced in the cross-correlation approach.

|  |
| --- |
|  |
| **Figure 3.12.** Shifts in the center frequency compared to for all datasets |

The velocities are listed below in Table 2 and their percent difference compared to values listed in Table 1.

|  |  |  |  |
| --- | --- | --- | --- |
| Table 2. Velocities calculated by Doppler shift | | | |
| Dataset A | Direction | Velocities | % difference from Table 1 |
| A | Away | 9.64 cm/s | 0.21 % |
| B | Away | 3.23 cm/s | 0.62 % |
| C | Toward | 10.1 cm/s | 4.67 % |

The table shows clearly that calculating velocities through Doppler shift is an accurate alternative. The advantage of using this method is that only the time series signal at a few or even just one depth is required while on the other hand, the cross correlation-based approach requires time series signal at several depths to detect periodic changes in axial data through time. However, cross-correlation based approach is much more robust in finding direction of motion.

**Problem 4: ECG Monitoring (Code used: BME154L\_S12\_Project\_Question4.m, ECGsnr.m)**

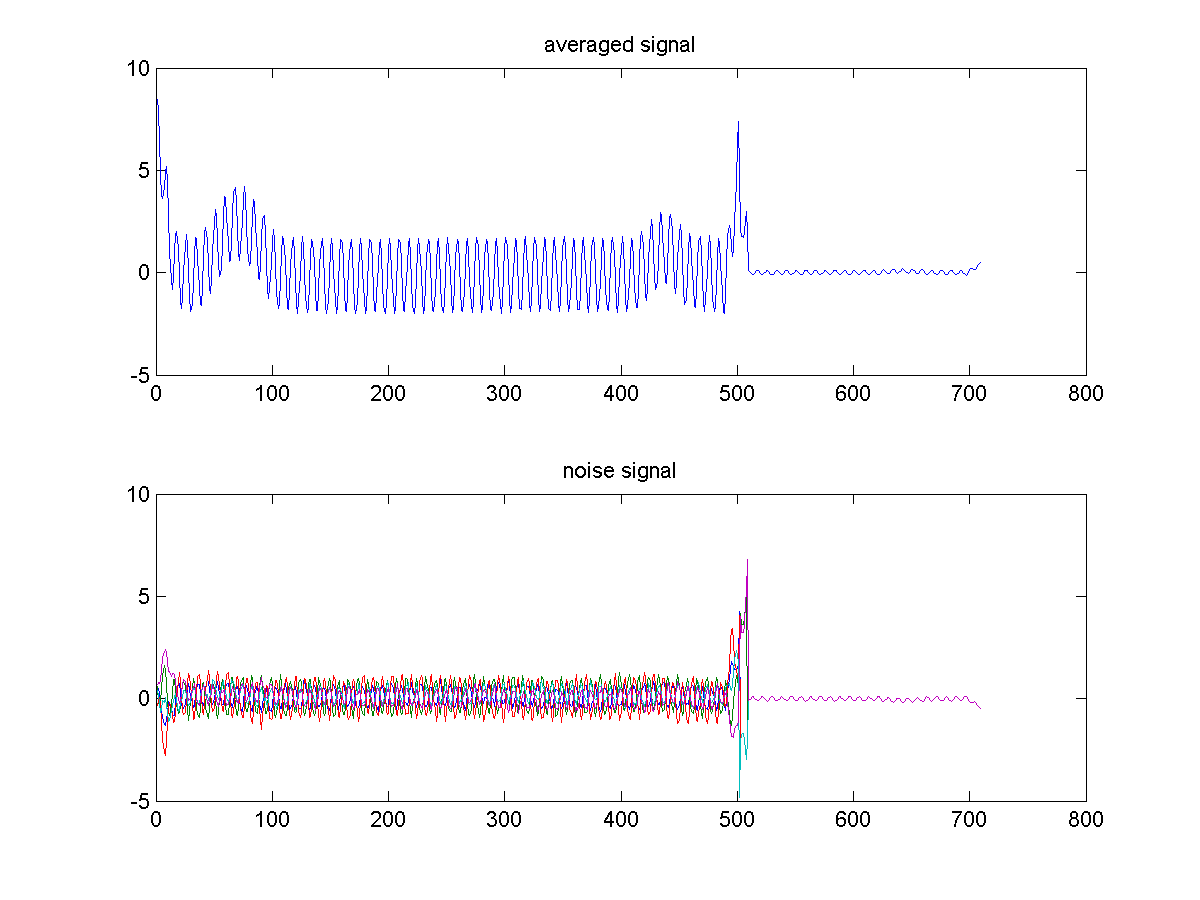
**Introduction**

ECGs are of extreme interest to cardiologists and others specializing heart-related fields because the ECG provides a very detailed visualization of the heart cycle. A heartbeat in an ECG is composed of a P wave, a QRS complex, and a T wave. The P wave represents atrial depolarization, which involves the contraction of the muscles of the atrium to start blood flow throughout the heart. The QRS peak is mainly defined by the muscles involved in ventricular depolarization. Atrial repolarization is also involved during the QRS peak, but atrial muscles are so much smaller than ventricular muscles that atrial repolarization is buried by ventricular depolarization. The T wave represents ventricular repolarization, which consists of the ventricles returning to their original state. In this problem, ECG data is analyzed in great detail. Signal-to-noise ratio analysis is included, which is particularly important with the signal used because there is a significant amount of noise generated by cheap ECG leads. The abnormalities of bradycardia and premature ventricular contractions (PVCs) are explored as well: bradycardia is a slow heart rate (generally symptomatic under 50 beats/min), and PVCs are a result of the heart’s cycle being activated by the ventricles rather than the sinoatrial node. This results in large voltage variations very unlike the standard PQRST waveforms. It is important to be able to monitor the presence of abnormalities such as PVCs and bradycardia as detection of these problems can indicate impaired function of the patient’s heart. A temporal/ frequency domain filtering method of choice is selected to further improve the signal to noise ratio, and other data formats besides the .bin binary file format given were considered in order to maximize efficiency and save space. Also, a cross correlation-based method can be used to find an optimal PR interval through phase alignment. Ensuring consistent PR intervals can be important, because significant deviations in the PR interval could indicate issues such as first degree block. Overall, this portion of the project introduces many aspects of the ECG and how abnormalities can be detected and/or removed to generate an optimal signal as well as locate problems in the heart’s activity for diagnostic purposes.

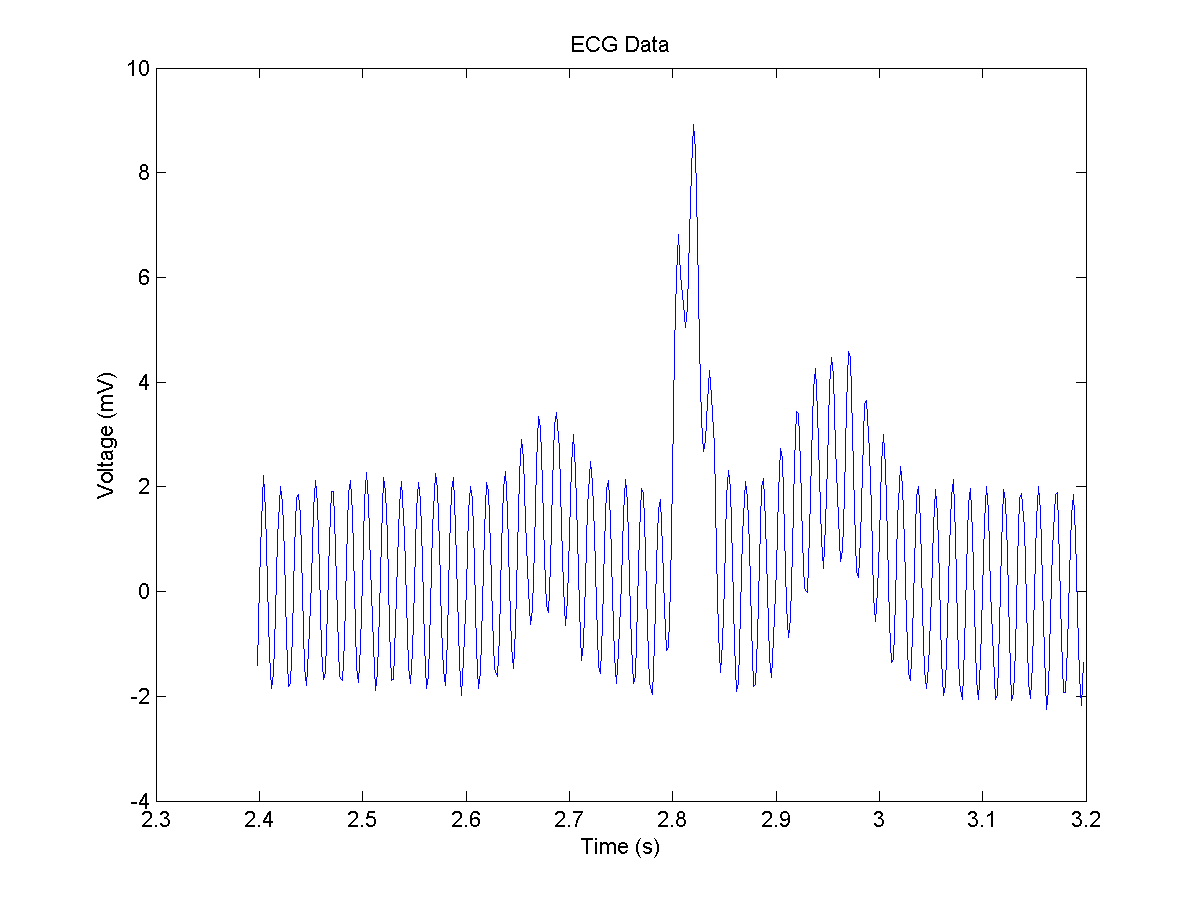
**Methods**

SNR calculation is performed in function file ECGsnr.m and called upon in BME154L\_FinalProject\_Question4.m.

* SNR can be calculated by taking the ratio of the root mean square (RMS) of the signal and the RMS of the noise over areas assumed to be zero. The ECGsnr.m function file parses the entire ECG signal into separate waveforms based on the QRS peak and zeropads to ensure each waveform is the same length. We can assume that the QRS peaks are the highest peaks present in the data and thus can be identified with the findpeaks command. These waveforms have now been phase-aligned at the QRS peak and the mean of these waveforms is representative of the entire waveform, and thus can be used to calculate the representative RMS noise. A portion of the waveform presumed to be zero (between the T-wave and the P-wave) is excised and detrended with the assumption that our noise will be centered at zero. The SNR (dB) is calculated as 20\*log(signalrms/noiserms) and comes out to be 11.461 dB for the raw ECG data. Figure 4.1 below plots examples of the average signal and the noise signals we generated.

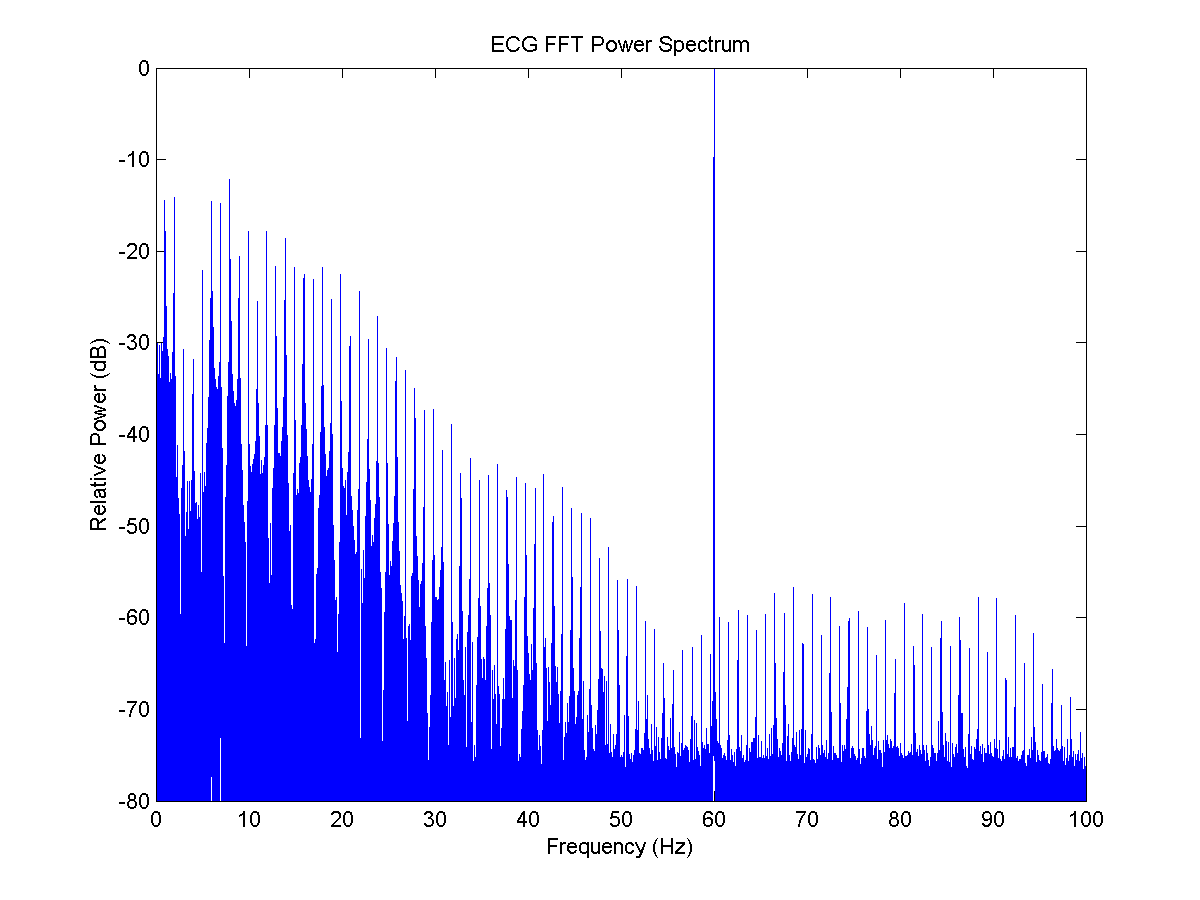


**Figure 4.1:** Phase-aligned average signal and noise signals for raw data.



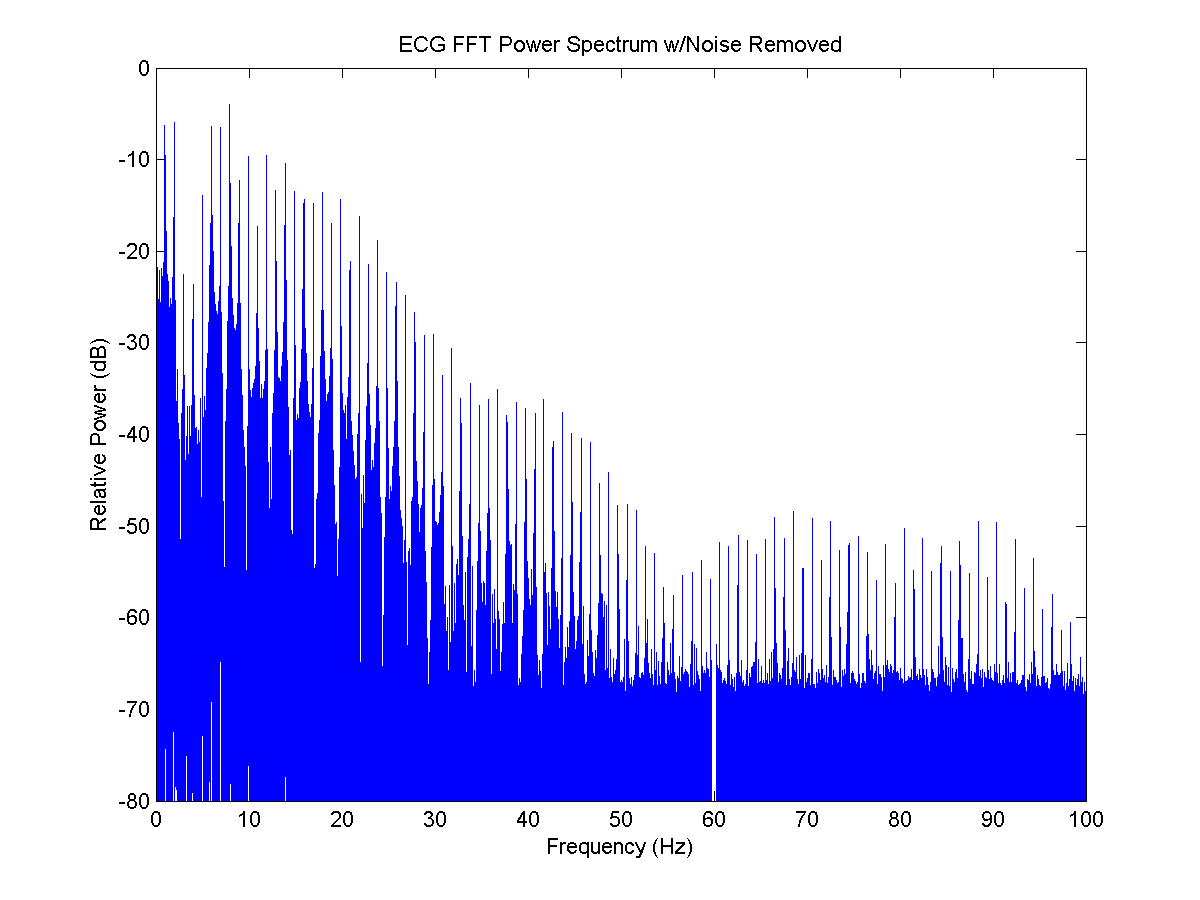
**Figure 4.2:** Sample raw ECG data with strong power noise present.

* Due to the extremely poor and noisy ECG leads used to generate the data, there is a significant amount of noise that needs to be removed through a filtering process. Plotting the frequency power spectrum makes it very apparent below in Figure 4.3 that the noise is concentrated at 60 Hz (and -60 Hz).



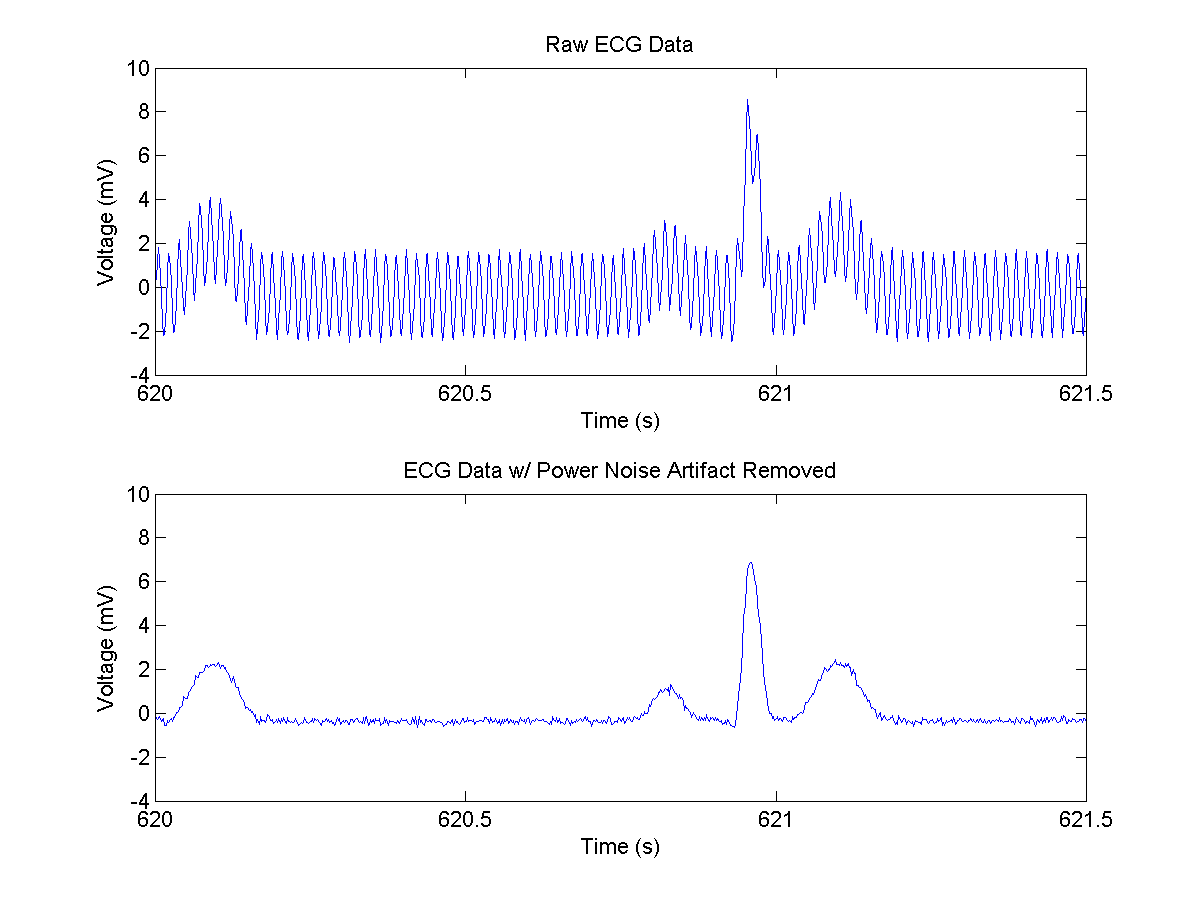
**Figure 4.3:** Relative power spectrum of original data. 60 Hz noise dominates.

A notch filter can thus be applied to selectively eliminate the frequencies near 60 Hz power noise and thus eliminate the noise artifact from our data.



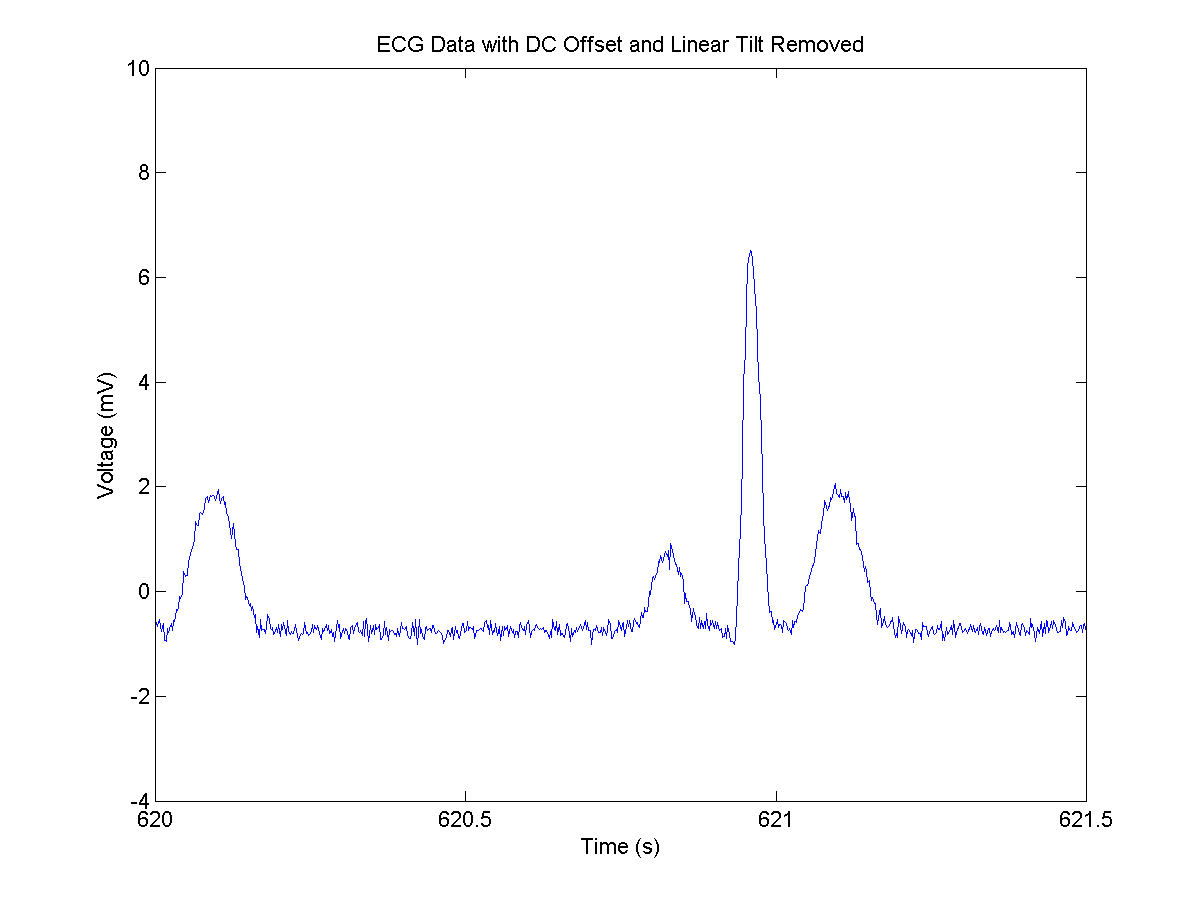
**Figure 4.4:** Relative power spectrum of artifact-free data. 60 Hz noise removed.

Once the data is plotted yet again, it is clear that the data is free of huge voltage oscillations.



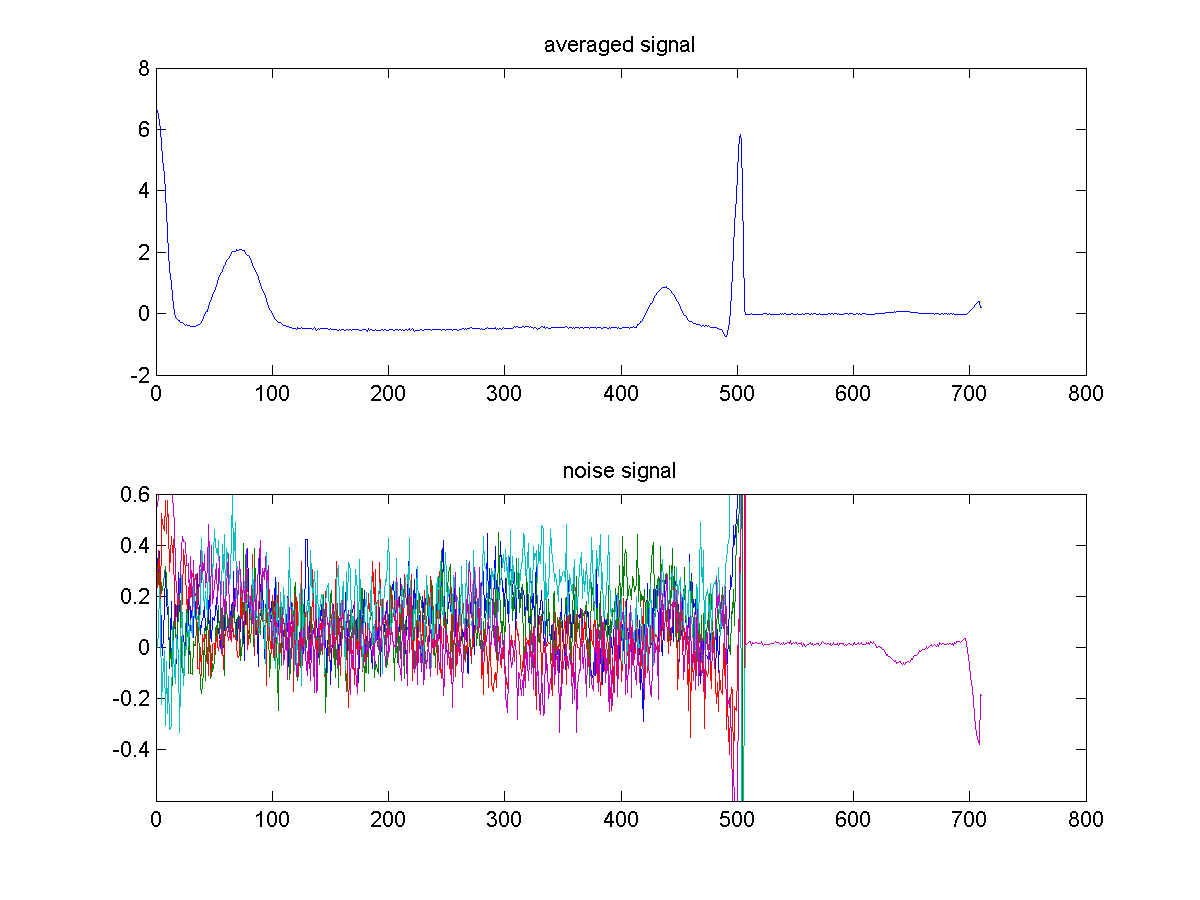
**Figure 4.5:** Raw ECG compared with artifact-free ECG.

* Although our ECG data had no apparent DC offset, the data was adjusted via polyfit to remove any lingering DC offset and set the signal mean to zero. Although this does not ‘set to zero’ the regions where we would expect a zero signal (such as between the T, P waves), this does not affect later calculations of the SNR because the noise signal is detrended as mentioned earlier regarding the raw ECG SNR calculation.

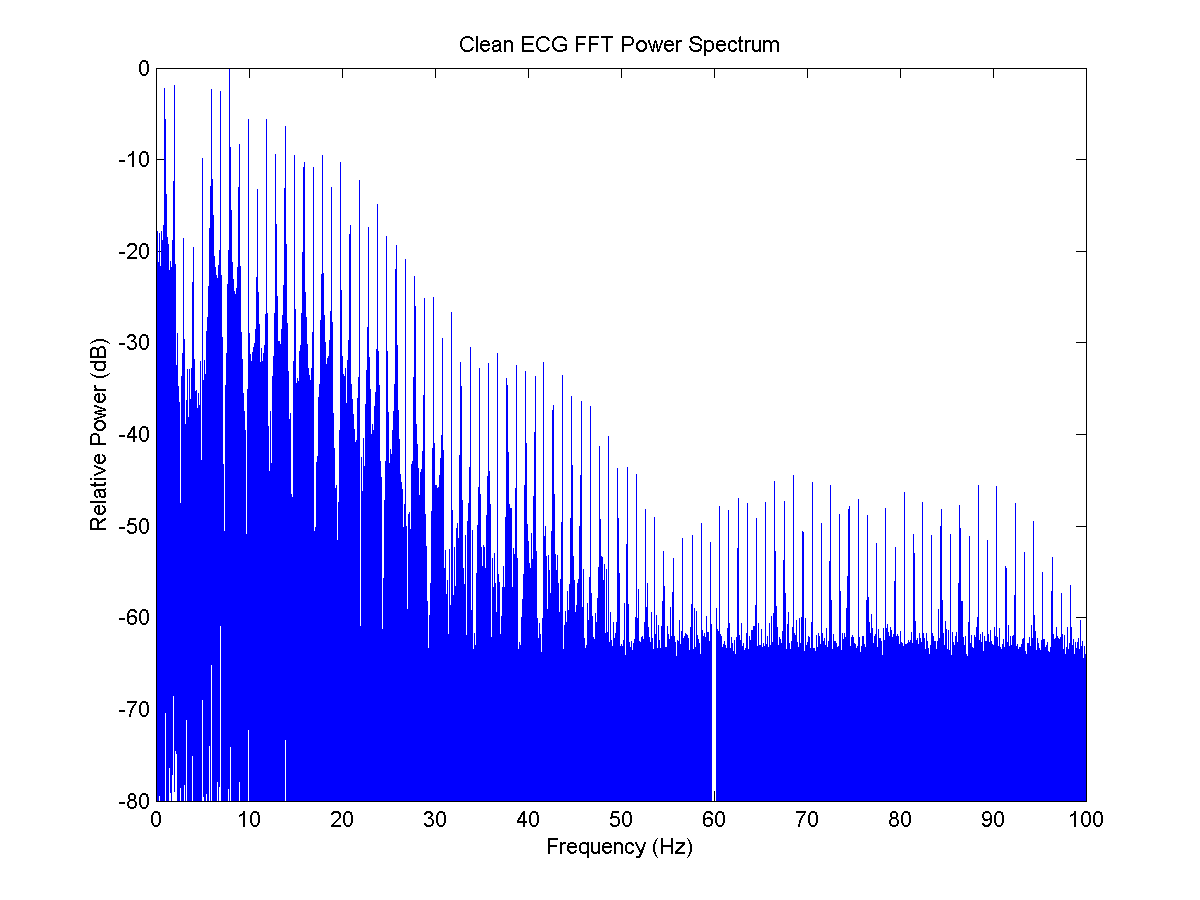


**Figure 4.6:** ECG with DC offset and/or tilt removed.

Once the data is processed and the artifact is removed, the new SNR is 20.912 dB which is significantly higher and better than the raw signal SNR. Removal of the artifact significantly and instantly improves the SNR.

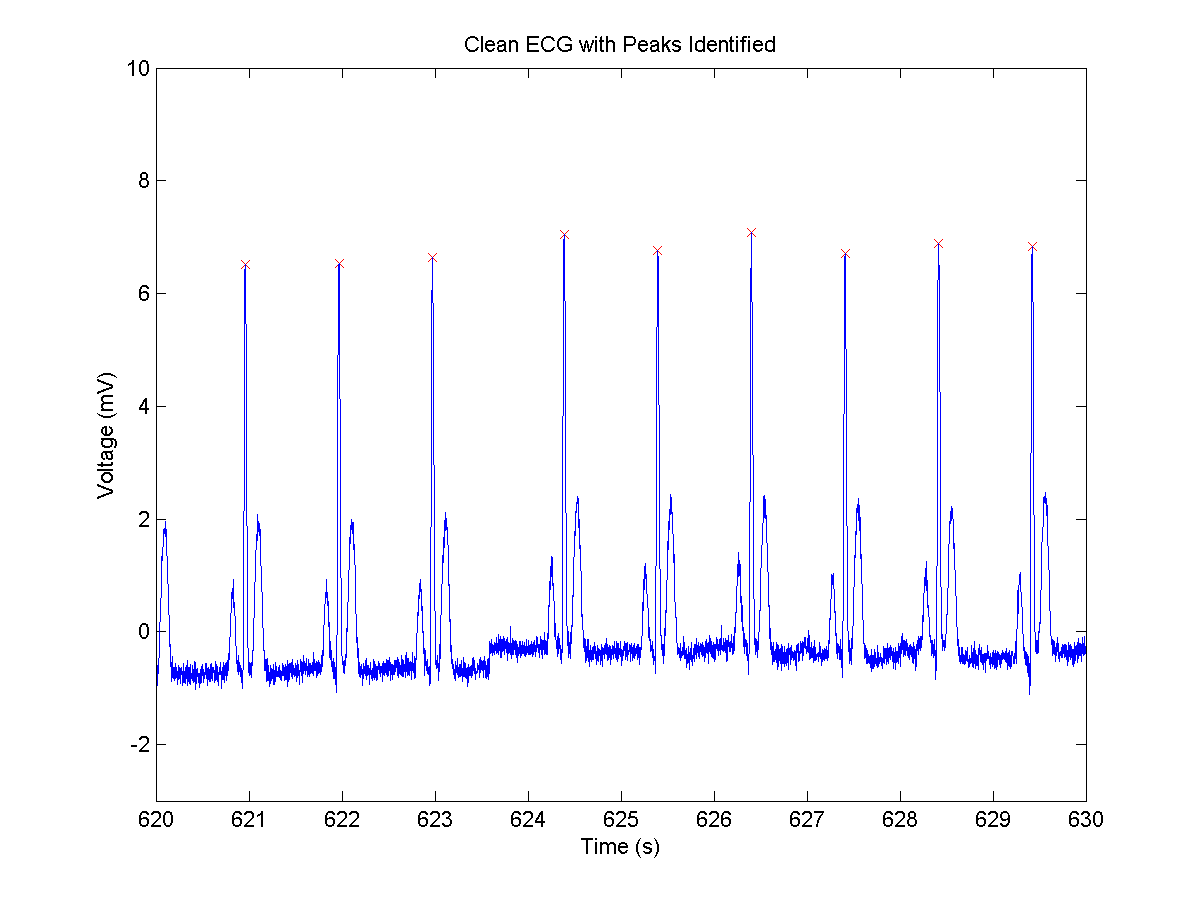


**Figure 4.7:** Phase-aligned average signal and noise signals for artifact-free data.



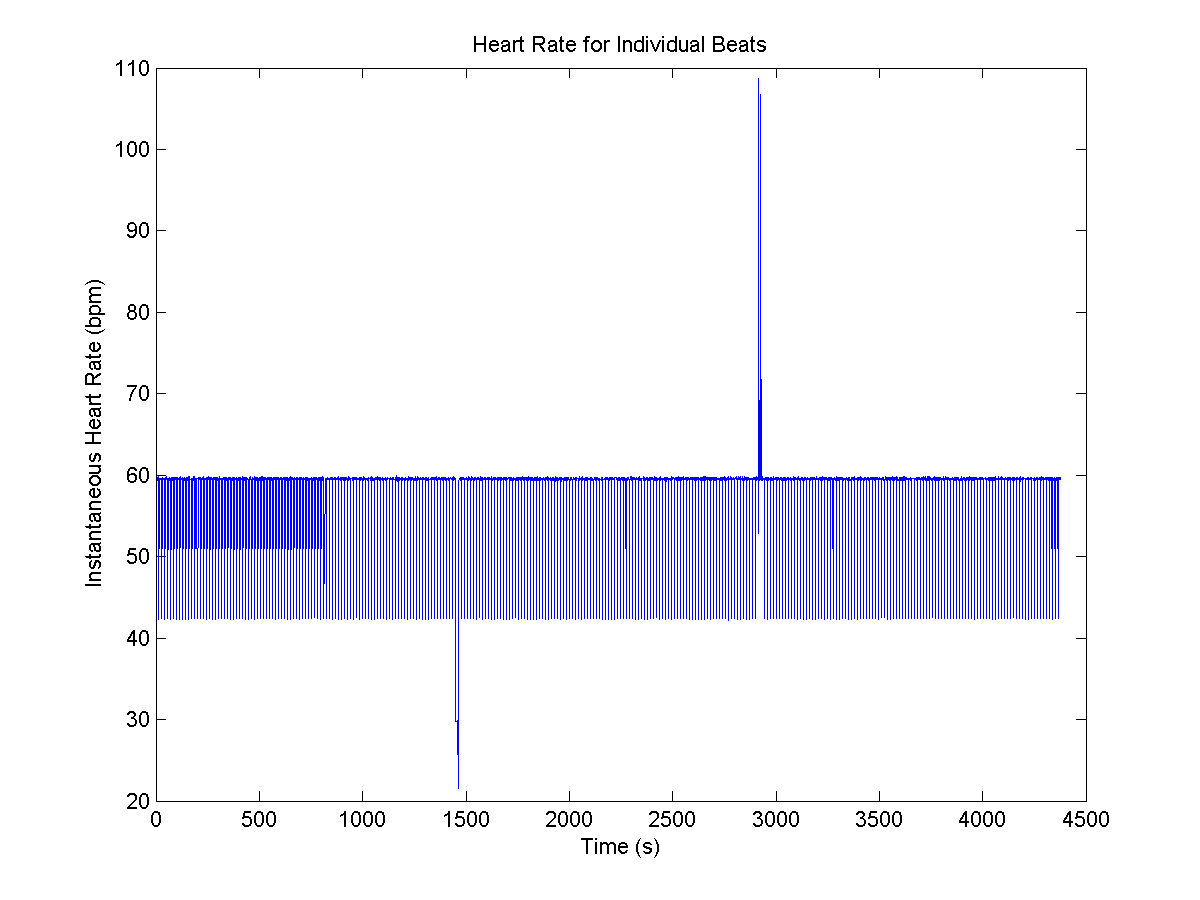
**Figure 4.8:** Relative power spectrum after polyfit removal applied. The power spectrum is the same besides the attenuation of 0 frequency signals (DC).

* Regions of bradycardia were detected automatically as well. QRS peak times were used to measure the instantaneous heart rate between QRS peaks. Any region with two heart beats in a row with a heart rate below 50 bpm was counted as an instance of bradycardia, since bradycardia becomes symptomatic at rates below 50 bpm.



**Figure 4.9:** Clean ECG with QRS peaks identified, extracted of 620-630 seconds.

* Figure 4.10 below showing the instantaneous heart rate over time clearly shows the anomaly that is the bradycardia, the region near 1500 seconds that shows heart rate significantly lower than other regions.



**Figure 4.10:** Heart rate for individual beats. The instantaneous heart rate varies between ~40 and 60 due to the presence of an extended PQRST waveform approximately every 10 beats.

Regions of bradycardia were automatically found in the script and were saved as bcardia\_locs. The times at which bradycardia occur are at t(bcardia\_locs) given in the following table:

**Table 4.1:** Time (in seconds) corresponding to regions of bradycardia.

|  |
| --- |
| Bradycardia Times (seconds) |
| 1448.6 |
| 1449.6 |
| 1451.1 |
| 1453.2 |
| 1455.2 |
| 1457.2 |
| 1459.2 |

Our QRS peak detection method means that these times correspond to the times of QRS peaks present in regions of bradycardia.

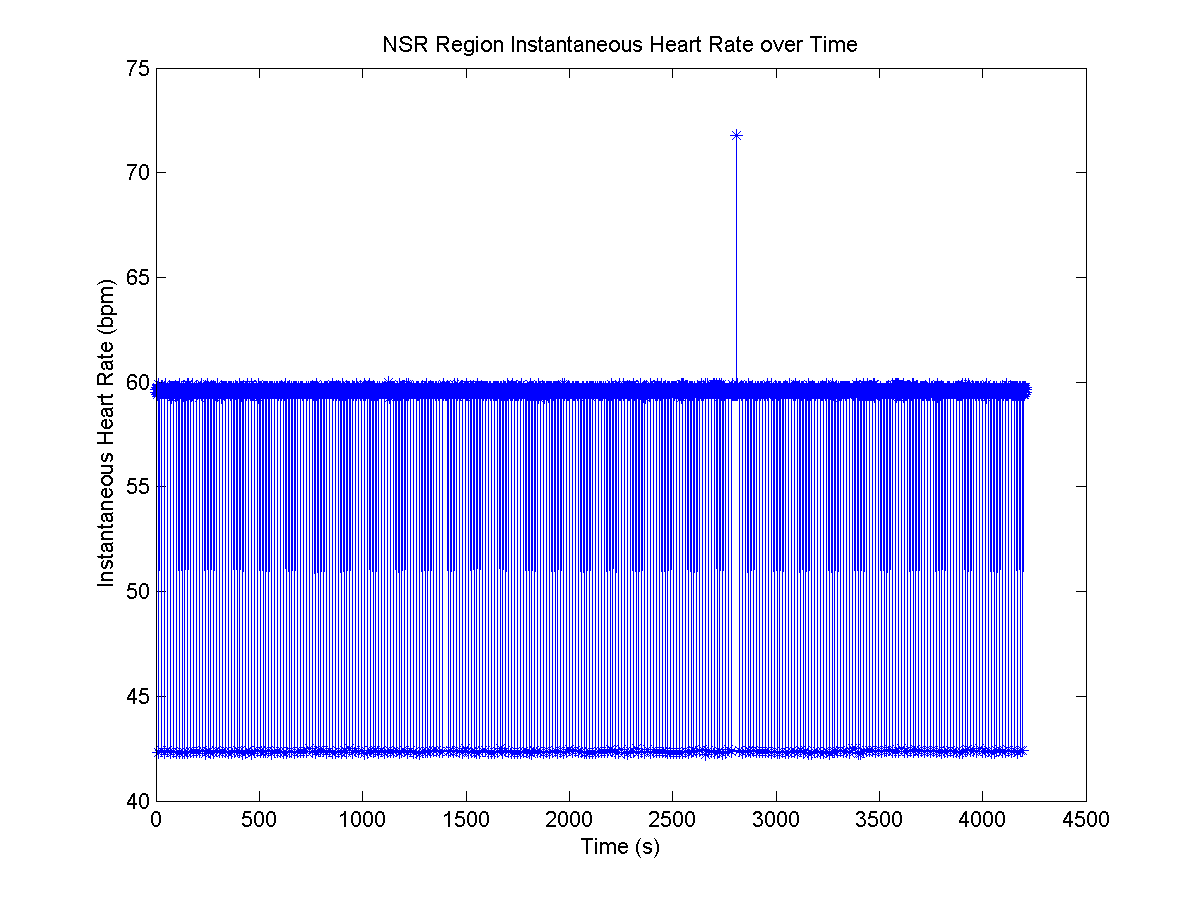
* We approached detection of premature ventricular contractions (PVCs) through a variety of methods. Initially, we attempted cross-correlation with a reference ECG in an attempt to identify higher correlation peaks as the non-PVC regions. However, due to the variable nature of correlation and area-summing, it proved rather difficult to generate consistent correlation peaks and NSR correlation thresholds. Even when the data was cleared up with a boxcar average, the correlation method still proved inconsistent and imprecise. In addition, there was the presence of an aberrant beat that had a normal QRS peak but abnormal P and T waves. This aberrant beat thus had a lower correlation peak as well leading to its improper identification as a PVC.
* Instead, we decided to approach this problem by setting a simple threshold detection based system for PVC detection. Since PVCs involve a strong negative inflection, we can make a fair assumption that no PVC will be present in the first few cycles of the data, and use the first few cycles as a reference to set a negative threshold for PVC detection.

**Table 4.2:** Time (in seconds) corresponding to the PVC peaks

|  |
| --- |
| PVC Peak Times |
| 2915.8 |
| 2916.3 |
| 2916.9 |
| 2924.3 |
| 2924.9 |
| 2925.4 |

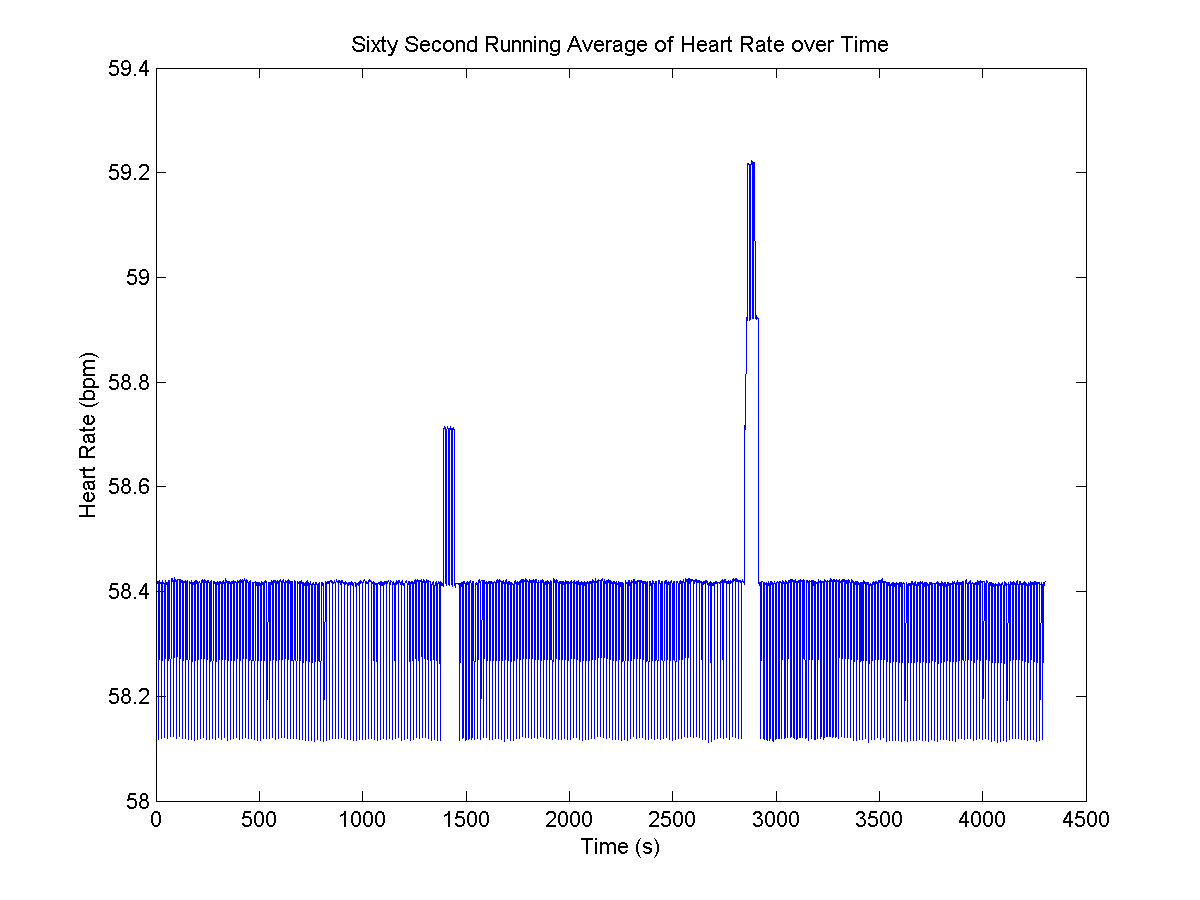
Note: since this method of PVC detection finds the local minima, the times correspond to the local valleys present in each PVC waveform and not the local peaks.

* Our next objective is to identify the heart rate but disregarding regions that are not Normal Sinus Rhythm (NSR). For the purposes of this section we considered neither PVCs nor bradycardia to be NSR. Since the PVC locations determined earlier are actually of the valleys and not the peaks, a for loop and if tree can be set up to remove from consideration those heart rates involving PVC peaks. In addition heart rates involving bradycardia were removed as well. The mean heart rate can then be easily calculated, and is determined to be about 58.39 beats per minute.



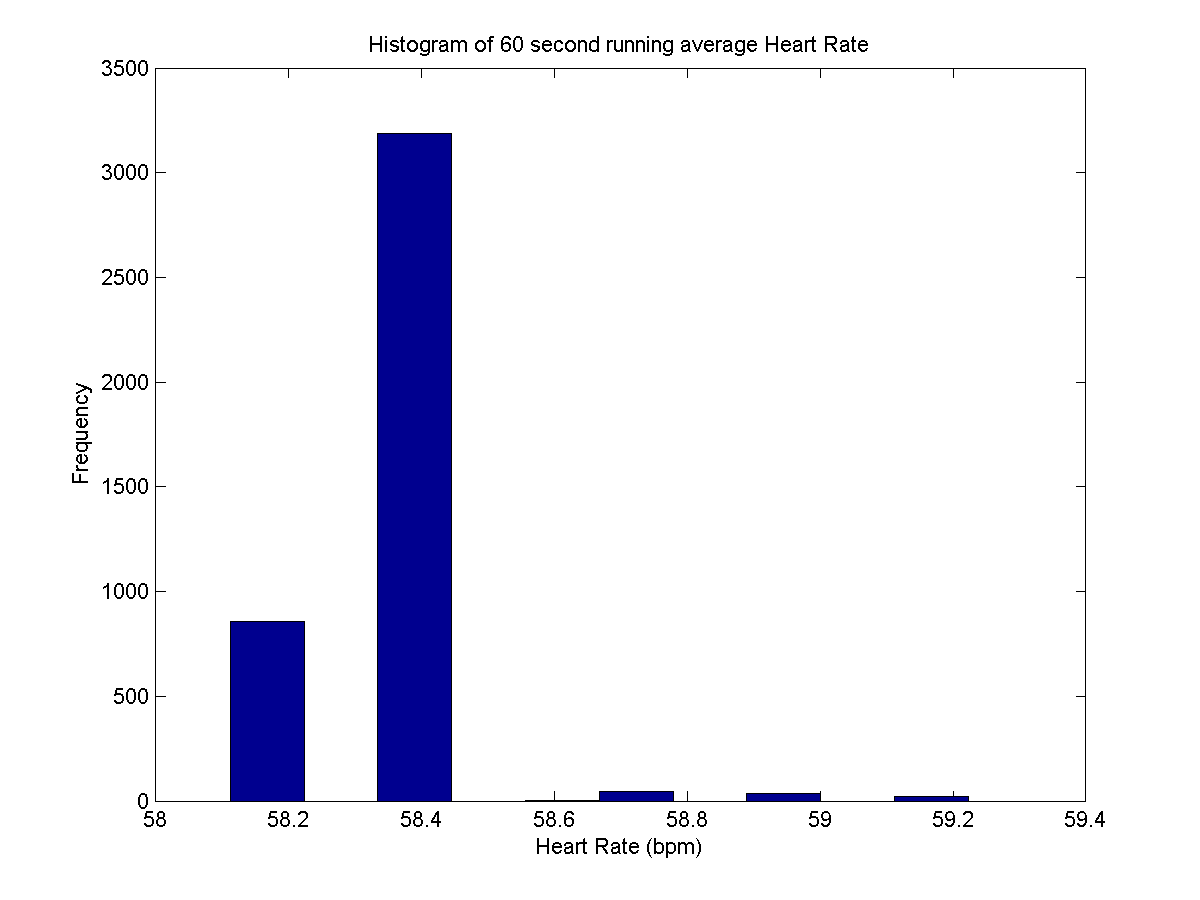
**Figure 4.10:** Only NSR region instantaneous heart rates shown. Single outlier high rate NSR waveform is the aforementioned waveform with abnormal P and T waves.

* For the 1 minute running average, the number of time points spanning 60 seconds can be automatically determined and then used to generate a running average of 60 seconds of heart rate data. Although an average and convolution could have been used here, problems may have arisen when removing those running average heart rates that included extraneous zeros at the beginning and the end of the data. Instead, it seemed more straightforward to code a simple for loop. The register variable acts to constantly replace the next element and continuously updates the running average.



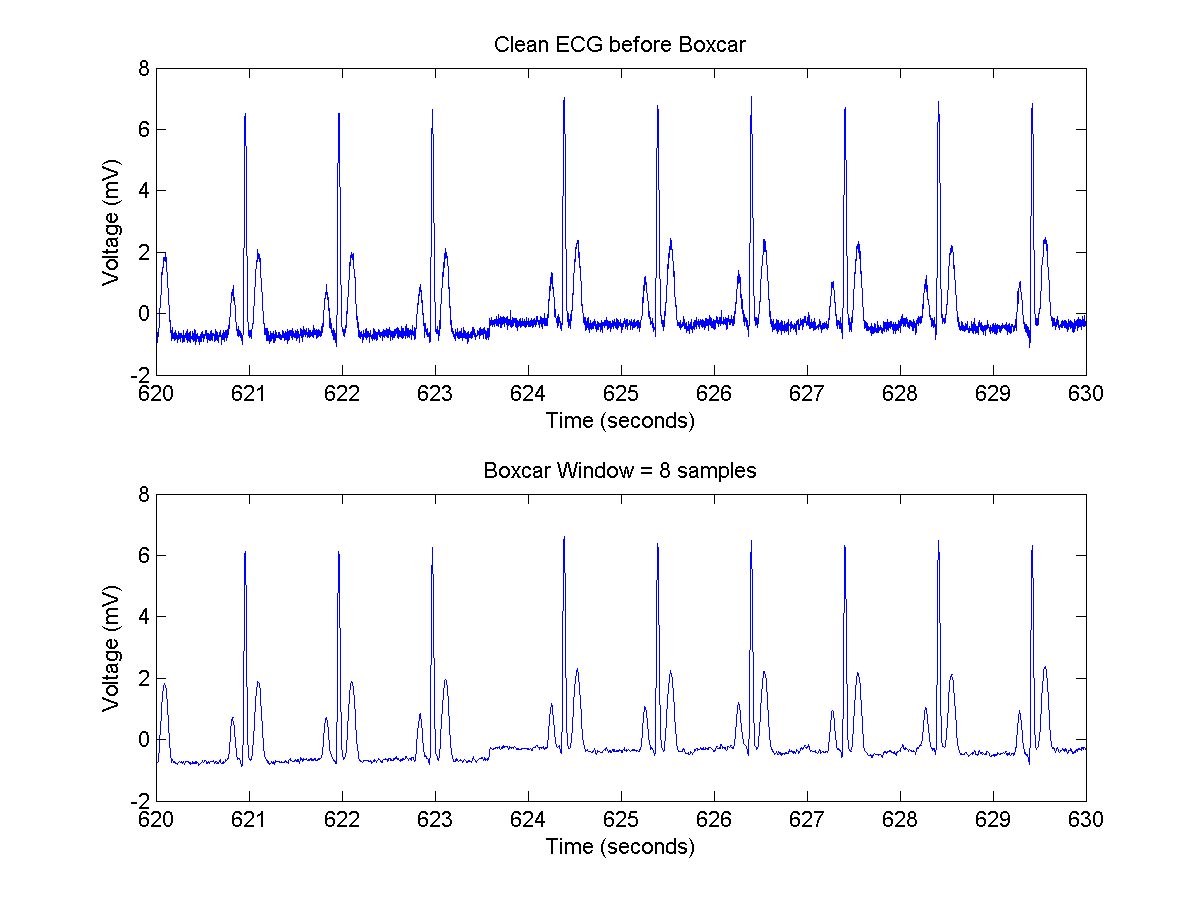
**Figure 4.11:** 60 second running average of NSR heart rates over time. The spikes near 1500 and 3000 seconds are due to the removal of bradycardia and PVC times (and lack of the regularly spaced extended PQRST waveforms in those areas)

The deviation in the heart rate is due to regularly spaced extended PQRST waves that provide a slower heart rate than average. In addition, the running average heart rate has slight spikes in the regions of bradycardia and PVC because there is a lack of the regularly spaced extended waveforms in those areas. Regardless, it can be observed that there is extreme regularity in heart rate stability.



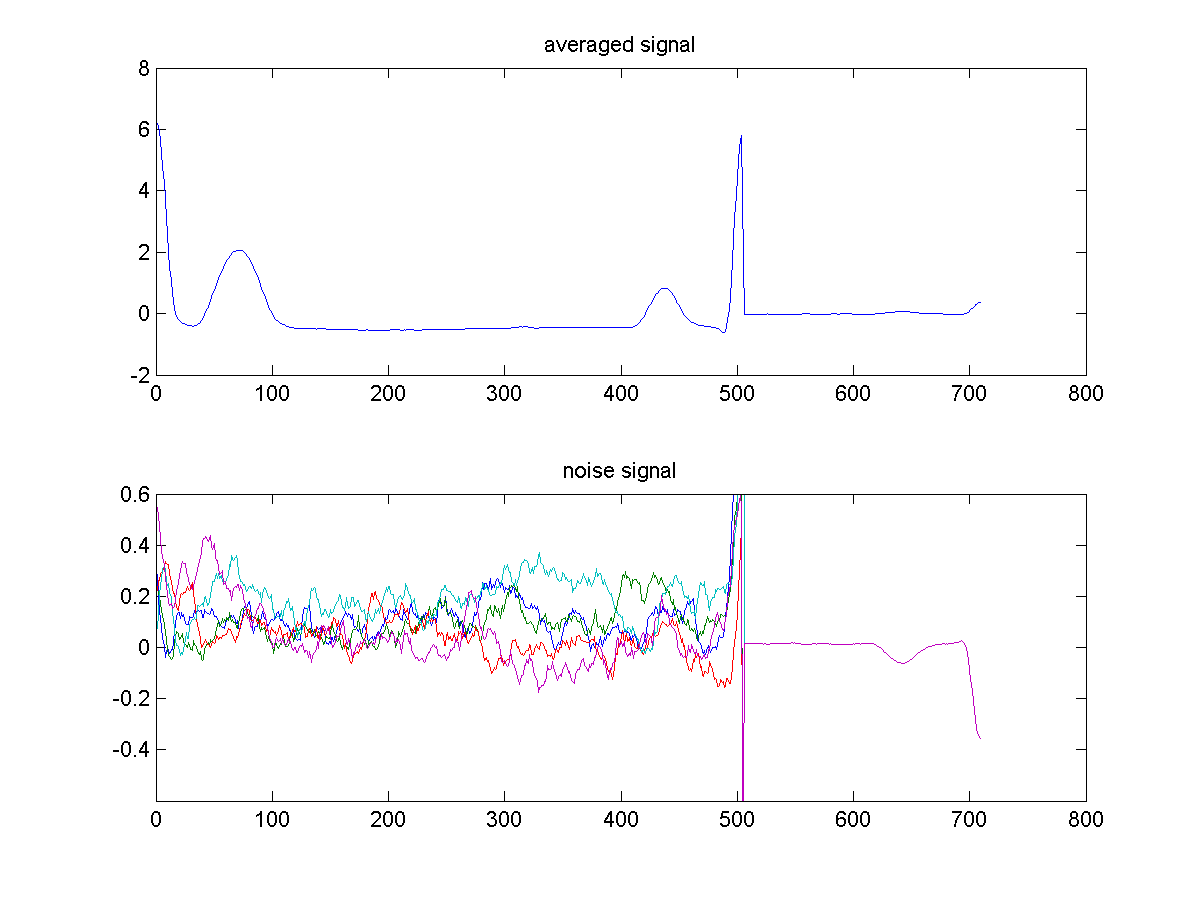
**Figure 4.12:** Histogram of 60 second running average of NSR heart rates over time. It can be seen the running average heart rate is very consistent and regular. The standard deviation of the heart rate is 0.149 bpm, indicating very low variation in the heart rate over NSR given the magnitude of the heart rate around 60 bpm.

The SNR of the ECG data can be maximized through the use of a boxcar averager. A vector containing several window sizes can be created for comparison of the effects of boxcar averaging. The ideal window size would remove as much noise as possible without distorting the true height of the QRS peak and losing the power of our signal. We found the ideal window size to be around 8 samples. Upon plotting the data, we can see that a window size of 8 reduces the noise present in the data while maintaining the amplitude of the QRS signal.



**Figure 4.13:** Comparison of the ECG signal before the boxcar average and after the 8-sample boxcar average. It can be seen a great deal of the noise is cleared up while the overall signal strength does not seem to significantly decrease.

Finally, the SNR of the ECG after the boxcar average is calculated using the same ECGsnr.m function to be 25.659 dB. This is a significant SNR improvement over even the artifact-free signal.



**Figure 4.14:** Phase-aligned average signal and noise for boxcar averaged data.

* The original ECG data is stored as 32-bit floats. However, we have calculated that the actual SNR of the original raw data is 11.461 dB, which corresponds to a signalrms/noiserms of about 3.74. Thus, regardless of how ridiculous it sounds, it only requires 2 bits to accurate represent the raw data. If we were to store the artifact-free data, its SNR of 20.912 dB indicates a signalrms/noiserms of about 11.11. This indicates it would be necessary to store about 4 bits of information to efficiently save this data. Thus, regardless of which data we are converting, an 8-bit data format would be a much more efficient data format to use to save this ECG data. In MATLAB, int8 can be used to convert our stored data to up to 255 discrete states, which is much more than necessary according to efficiency but will still save a great deal of space. This usage of 8-bits as opposed to 32-bits means that we will use 4 times less data. Thus, the 17,112 kilobyte binary data file would become a data file of size 4,278 kilobytes.

**OPTIONAL:**

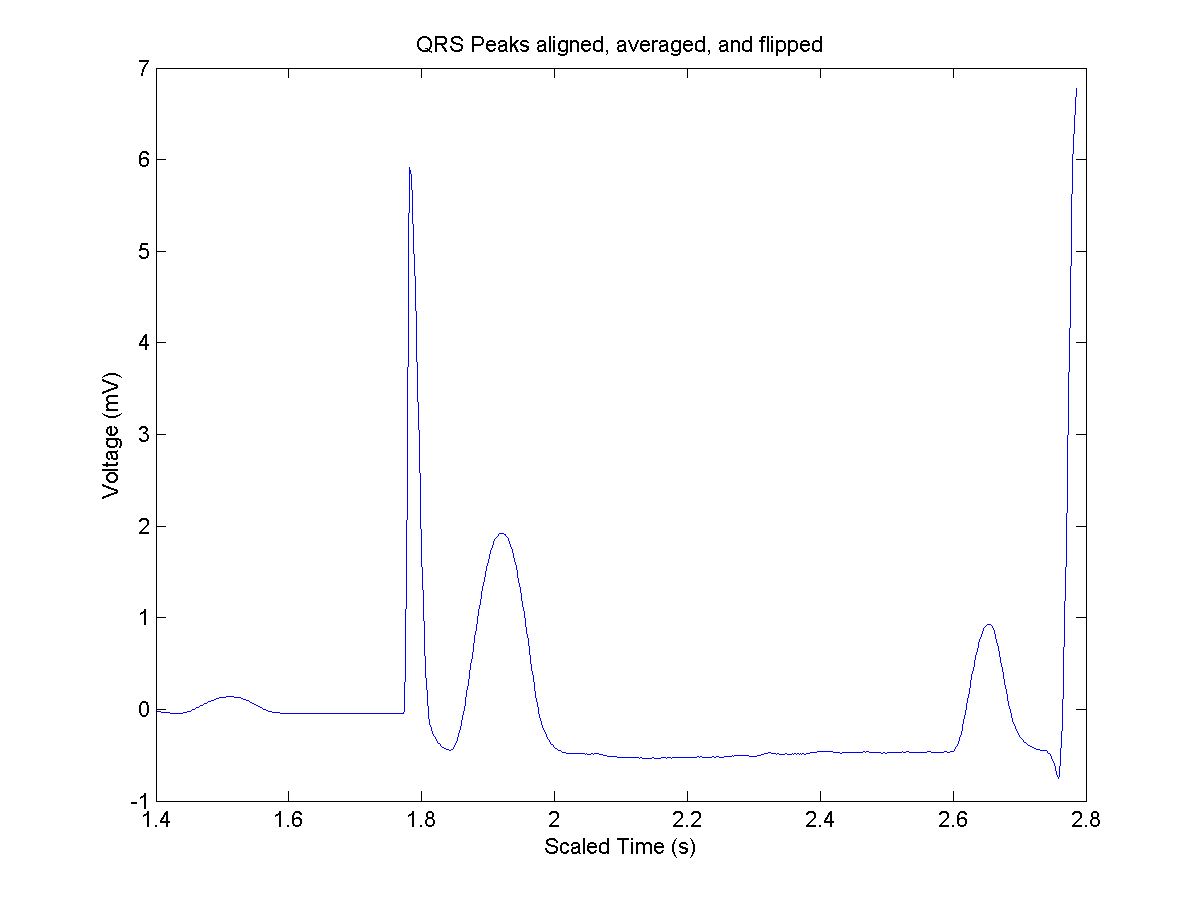
In approaching this optional problem, we believed the correlation-based approach suggested by the problem is highly inefficient and inconsistent. For this approach to be accurate, an NSR as free of noise as possible must be manually excised from the data itself, which is both labor intensive and subject to variability. Furthermore, while the PR interval itself remains the same over varying heart rates, using an NSR that has a period significantly different from the cycle it is cross correlated with will lead to inconsistent correlation peaks. Our approach parses the cycles at the QRS peaks and directly aligns all data with the second peak. Our assumptions in this approach are that the R peak is much higher than P or T wave peaks and PR intervals are consistent over varying heart rates. The advantages of this approach are that it mitigates any visual bias, is automated and is consistent for all human ECG signals including those that exhibit abnormalities which do not modify the relative height of the R peak.

The entire dataset is parsed so that each cycle is flipped and phase aligned at each R peak. Since the data has been flipped, aligning the QRS peaks and averaging means that the resultant waveform will begin at the R peak, transition down to the Q dip, follow into the P peak. The averaged signal is then flipped again to maintain the order of PQR peaks. It is important to note the RP interval is equal to the PR interval and flipping does not modify our measurements. Table below shows the averaged PR interval in comparison to the PR interval of several types of heart beats.

**Table 4.3:** Average PR interval in comparison to the PR interval of several types of heart beats

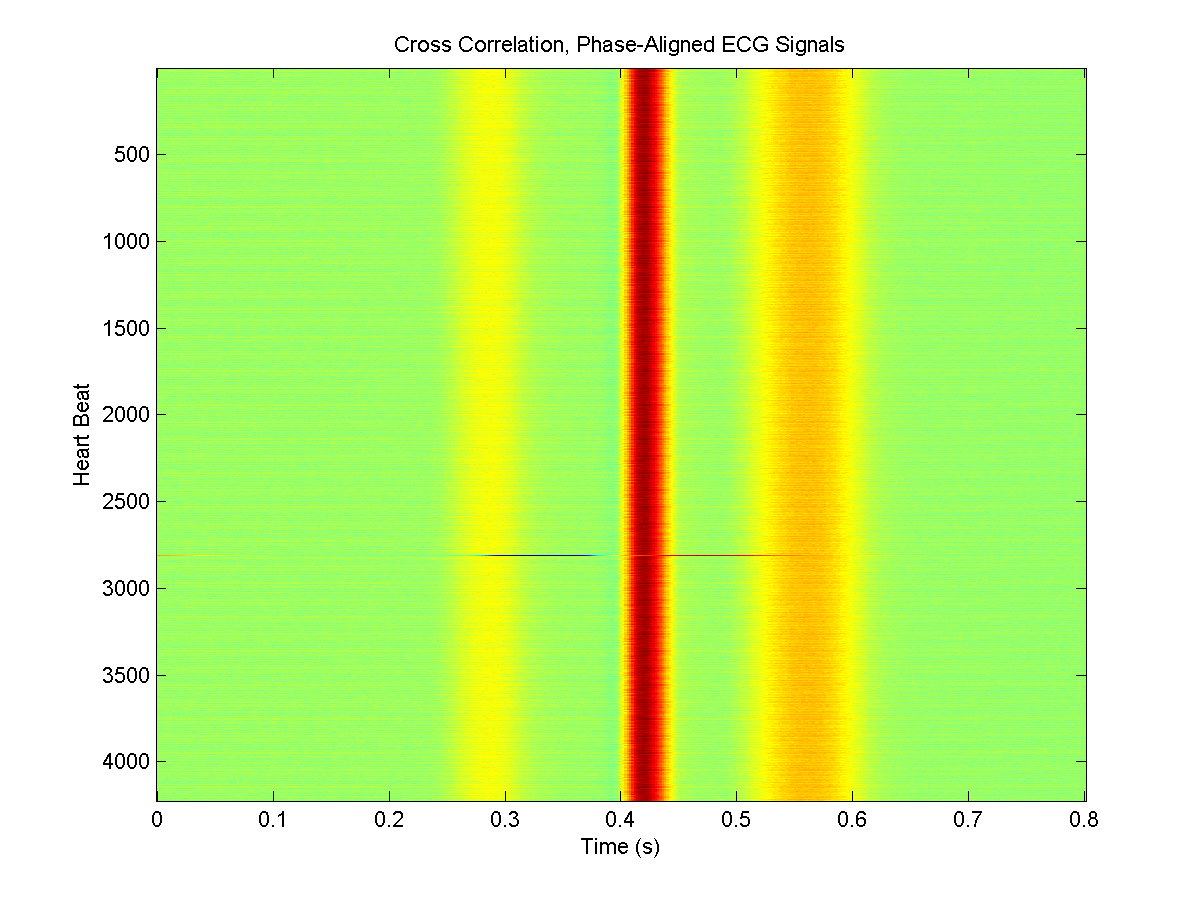
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Average PR* | *Sample 1 NSR* | *Sample 2 NSR* | *Sample 3*  *Extended NSR (occur every 10 or so beats)* | *Sample 4 Bradycardia* |
| 0.1320 seconds | 0.1380 seconds | 0.1360 seconds | 0.1380 seconds | 0.1400 seconds |

It can be seen from the estimated PR intervals above that the sampled PR intervals are extremely close to although ever so slightly longer than the average PR interval as determined by alignment of QRS peaks.

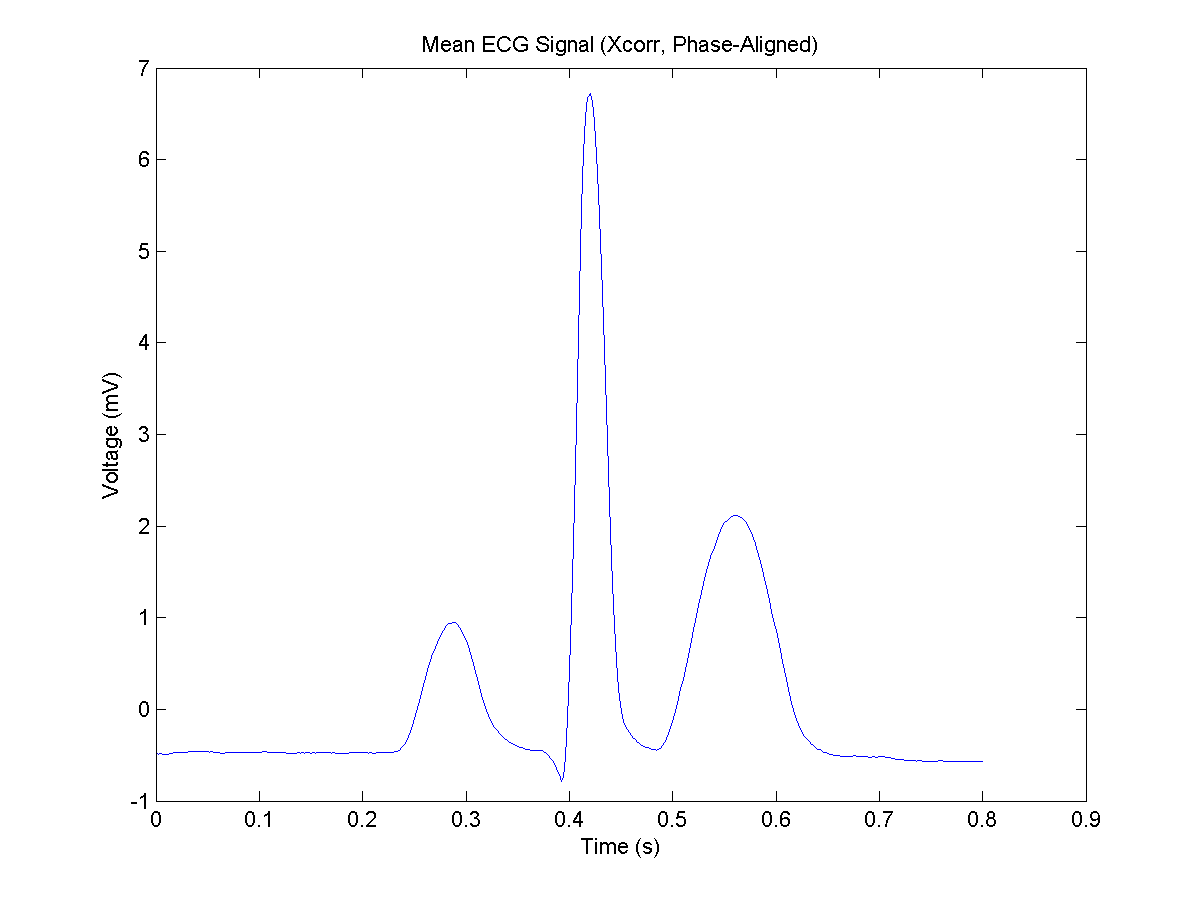


**Figure 4.15:** QRS-aligned average signal for determination of PR interval. Note that half of the QRS peak is on the far right and half is located near 1.8 seconds. The half before 1.8 seconds is just the correlation tailing down to the zero padding.

In addition, we decided to use code based on the PS 7 solutions to confirm our mean PR interval. Using the cross correlation method of heart beat detection and using correlation-based phase alignment to match up different heart beats, we returned the following two figures:



**Figure 4.16:** Phase-aligned ECG signals. Note the distortions around 1500 and 3000 seconds corresponding to bradycardia and PVCs.



**Figure 4.17:** Mean ECG signal averaged over waveforms. From this average the mean PR interval can be determined to be 0.1320 seconds. This is exactly the same as what we calculated by lining up QRS peaks.

**Appendix:**

**Problem 1**

% FinalProject1.m

clf; clear all; close all;

%% Plotting Impulse Responses

figure(1)

order = {'First'; 'Second'; 'Third'; 'Fourth'};

for index = 1:4

F(index,:,:) = load(sprintf('TransImpResp%d.asc',index));

time(index,:) = F(index,:,1);

voltage(index,:) = F(index,:,2);

subplot(2,2,index)

plot(time(index,:),voltage(index,:))

title([order{index} ' Impulse Response'])

xlabel('time (s)')

ylabel('voltage (V)')

end

%% Plotting and Calculating Power Spectrums

figure(2)

for index = 1:4

T0(index) = time(index,end)-time(index,1);

N = length(time(index,:));

ts = T0(index)/N;

fs = 1/ts;

freq = fs\*(-0.5:1/N:0.5-(1/N));

freqplus = freq(length(freq)/2+1:end);

FT\_Voltage(index,:) = fft(voltage(index,:));

FT\_Shift\_Voltage(index,:) = fftshift(FT\_Voltage(index,:));

Mag\_FT\_Voltage(index,:) = abs(FT\_Shift\_Voltage(index,:)/max(abs(FT\_Shift\_Voltage(index,:))));

Power\_Voltage(index,:) = 20\*log10(Mag\_FT\_Voltage(index,:));

Power\_Voltage\_Plus(index,:) = Power\_Voltage(index,length(freq)/2+1:end); % Splits matrix in half

subplot(2,2,index)

plot(freq,Power\_Voltage(index,:),'.-')

xlabel('Frequency (Hz)'); ylabel('Power (dB)'); title(['Power Spectrum for ' order{index} ' Transducer'])

end

%% Finding -3dB

for index = 1:4

maxindex(index)=find(max(Power\_Voltage\_Plus(index,:))==Power\_Voltage\_Plus(index,:)); % Finds where the maximum is

FirstTrans3dB1(index) = interp1(Power\_Voltage\_Plus(index,2:maxindex(index)),freqplus(2:maxindex(index)),-3,'linear'); % interpolates to find one -3dB intercept

FirstTrans3dB2(index) = interp1(Power\_Voltage\_Plus(index, maxindex(index):end),freqplus(maxindex(index):end),-3,'linear'); % interpolates to find one -3dB intercept

centerfreq(index) = geomean([FirstTrans3dB1(index),FirstTrans3dB2(index)]); % Uses the geometric mean of the -3dB points to calculate center freqquency

FractionalBandwidth(index) = abs((FirstTrans3dB1(index)-FirstTrans3dB2(index)))/centerfreq(index);

end

%% Plotting Input

NumberofPeriods=22;

figure(3)

for index = 1:4

timestep(index) = mean(diff(time(index,:)));

period(index) = 1/centerfreq(index);

end

% Each time and input vector is of a different length; hence hardcoding is

% done

% This code makes sure to maintain the same time step in the transducer

t1 = linspace(0, NumberofPeriods\*period(1), floor(NumberofPeriods\*period(1)/timestep(1)));

t2 = linspace(0, NumberofPeriods\*period(2), floor(NumberofPeriods\*period(2)/timestep(2)));

t3 = linspace(0, NumberofPeriods\*period(3), floor(NumberofPeriods\*period(3)/timestep(3)));

t4 = linspace(0, NumberofPeriods\*period(4), floor(NumberofPeriods\*period(4)/timestep(4)));

Input1=2\*sin(2\*pi\*centerfreq(1)\*t1);

Input2=2\*sin(2\*pi\*centerfreq(2)\*t2);

Input3=2\*sin(2\*pi\*centerfreq(3)\*t3);

Input4=2\*sin(2\*pi\*centerfreq(4)\*t4);

for index = 1:4

subplot(2,2,index)

t = eval(sprintf('t%d',index));

Input = eval(sprintf('Input%d',index));

plot(t,Input)

xlabel('time (s)')

ylabel('Voltage (V)')

title([order{index} ' Sinusoidal Input'])

end

%% Plotting the Output

figure(4)

for index = 1:4

Input = eval(sprintf('Input%d',index));

Output(index,:) = conv(voltage(index,:),Input,'same');

subplot(2,2,index)

plot(time(index,:),Output(index,:))

xlabel('time (seconds)')

ylabel('voltage (V)')

title(['Output of ' order{index} ' Transducer'])

end

%% Determine When Steady State Occurs

% Hardcoding was used here because the findpeaks command would produce a

% different amount of peaks

%This produces the maximum different between the peaks

MaxSinePeaksDiff1=max(abs(diff(findpeaks(Output(1,:)))));

MaxSinePeaksDiff2=max(abs(diff(findpeaks(Output(2,:)))));

MaxSinePeaksDiff3=max(abs(diff(findpeaks(Output(3,:)))));

MaxSinePeaksDiff4=max(abs(diff(findpeaks(Output(4,:)))));

%This simply returns a 1 or 0 for if the output is at steady state or not

Steady1=MaxSinePeaksDiff1<0.01\*max(Output(1,:))

Steady2=MaxSinePeaksDiff2<0.01\*max(Output(2,:))

Steady3=MaxSinePeaksDiff3<0.01\*max(Output(3,:))

Steady4=MaxSinePeaksDiff4<0.01\*max(Output(4,:))

% This was hardcoded in to get a fit between cycles needed and center

% frequency

CyclesNeeded=[13 13 22 20];

figure (5)

plot (centerfreq,CyclesNeeded)

title('Plot of Cycles Needed vs, Cneter Frequency')

xlabel('Center Frequency (Hz)')

ylabel('Cycles Needed')

**Problem 2.1**

%FinalProject2Quad

%% Plotting the RF Data

clear all; close all; clf

load('rf.mat')

figure(1)

imagesc(lat,depth,rfdata)

xlabel('lateral direction (mm)')

ylabel('depth (mm)')

title('Original Radio Frequency Data')

axis image

colormap hot

colorbar

%Calculate Contrast

SignalIntensity=mean(mean(rfdata(854:975,360:521)));

BackGroundIntensity=mean(mean(rfdata(1:300,1:300)));

Contrast=(SignalIntensity-BackGroundIntensity)/(max(max(rfdata))-min(min(rfdata)));

%% Quadrature Demodulation

[row, cols]= size(rfdata);

% Find Power Spectrum to see what frequency to use

T0=(depth(length(depth))-depth(1))\*2/1540/1000;

N=row;

ts=T0/N;

fs=1/ts;

freq= fs\* (0 : 1/N: 1-(1/N));

DataFT=abs(fft(rfdata));

DataFTNorm=abs(DataFT)/max(max(DataFT));

PowerNorm=20\*log10(DataFTNorm);

figure(2)

plot(freq,PowerNorm)

title('Power Spectrum of RF Data Columns (A-Lines)')

xlabel('frequency (Hz)')

ylabel('power (dB')

% Finding the Center Frequency

for k=1:cols

Index(k)=find(max(PowerNorm(1:round(length(freq)/2),k))==PowerNorm(1:round(length(freq)/2),k));

CenterFreq(k)=freq(Index(k));

end

CenterFreqAvg=mean(CenterFreq);

t=(depth\*2/1000/1540)'; % Finding the right time points to use

SignalCos=zeros(row,cols);

SignalSin=zeros(row,cols);

%Multiply by Sine and Cosine

for i=1:cols

SignalCos(:,i)=cos(2\*pi\*CenterFreqAvg\*t).\*rfdata(:,i);

SignalSin(:,i)=sin(2\*pi\*CenterFreqAvg\*t).\*rfdata(:,i);

end

%FT

SignalCosFT=fft((SignalCos));

SignalSinFT=fft(SignalSin);

figure(4)

plot(freq,abs(SignalCosFT))

title('Amplitude Response of RF Data Multiplied by Cosine')

xlabel('frequency (Hz)')

ylabel('Amplitide')

figure(5)

plot(freq,abs(SignalSinFT))

title('Amplitude Response of RF Data Multiplied by Sine')

xlabel('frequency (Hz)')

ylabel('Amplitide')

% Multiply by Rect/Low Pass Filter

CutoffFreq=0.5e7;

SignalCosFTFiltered=zeros(row,cols);

SignalSinFTFiltered=zeros(row,cols);

for i=1:cols

SignalCosFTFiltered(:,i)=SignalCosFT(:,i).\*(abs(freq)<CutoffFreq)';

SignalSinFTFiltered(:,i)=SignalSinFT(:,i).\*(abs(freq)<CutoffFreq)';

end

figure(6)

plot(freq,abs(SignalCosFTFiltered))

title('Amplitude Response of Filtered RF Data Multiplied by Sine')

xlabel('frequency (Hz)')

ylabel('Amplitide')

figure(7)

plot(freq,abs(SignalSinFTFiltered))

title('Amplitude Response of Filtered RF Data Multiplied by Sine')

xlabel('frequency (Hz)')

ylabel('Amplitide')

% Reconstruct/InverseFT

RealSignal=real(ifft(SignalCosFTFiltered));

ImaginarySignal=real(ifft(SignalSinFTFiltered));

figure(8)

SignalRecon=2\*sqrt(RealSignal.^2+ImaginarySignal.^2);

imagesc(lat,depth,SignalRecon)

axis image

colormap hot

colorbar

xlabel('lateral direction (mm)')

ylabel('depth (mm)')

title('Data After Quadrature Demodulation')

caxis([0 4e-21])

% Calculate Contrast

SignalIntensity=mean(mean(SignalRecon(854:975,360:521)));

BackGroundIntensity=mean(mean(SignalRecon(1:300,1:300)));

ContrastDemod=(SignalIntensity-BackGroundIntensity)/(max(max(SignalRecon))-min(min(SignalRecon)));

%% Hilbert Transform

Transform=hilbert(rfdata);

DemodTransform=abs(Transform);

figure(9)

imagesc(lat,depth,DemodTransform)

axis image

colormap hot

colorbar

xlabel('lateral direction (mm)')

ylabel('depth (mm)')

title('Data After Hilbert Transform')

caxis([0 4e-21])

SignalIntensity=mean(mean(DemodTransform(854:975,360:521)));

BackGroundIntensity=mean(mean(DemodTransform(1:300,1:300)));

ContrastHilbert=(SignalIntensity-BackGroundIntensity)/(max(max(DemodTransform))-min(min(DemodTransform)));

LogCompressed=log10(SignalRecon);

figure (10)

imagesc(lat,depth,LogCompressed)

axis image

colorbar

title('Log Compressed Quadrature Demodulation (Eye of Sauron)')

xlabel('lateral direction (mm)')

ylabel('depth (mm)')

SignalIntensity=mean(mean(LogCompressed(854:975,360:521)));

BackGroundIntensity=mean(mean(LogCompressed(1:300,1:300)));

ContrastLog=(SignalIntensity-BackGroundIntensity)/(max(max(LogCompressed))-min(min(LogCompressed)));

**Problem 2.2**

% FinalProject2DepthDep.m

%% Plotting the RF Data

clear all; clf; close all

load('rf.mat')

figure(1)

imagesc(rfdata)

axis image

colormap gray

xlabel('lateral direction (mm)')

ylabel('depth (mm)')

title('Original Radio Frequency Data')

% The first one was used to split rfdata into smaller pieces to try to

% eliminate the lesion but this didn't really work

rfdata1=rfdata;

% Finding the dimensions of the matrix

[NumRows1,NumCols1]=size(rfdata1);

[NumRows,NumCols]=size(rfdata);

% Calculating time

t=2\*depth/1540/1000;

%Spltting the whole matrix into columns into to find depth dependent center

%frequency

cols=3;

rows=floor(NumRows/cols);

rfdatashort=zeros(rows,3\*NumCols);

tshort=zeros(rows,cols);

% Goes through and splits up the rfdata into the different number of

% columns (chunks)

for Index=0:NumCols1-1

for i=0:cols-1

rfdatashort(:,(i+1)+cols\*Index)=rfdata1(rows\*i+1:rows\*(i+1),Index+1);

tshort(:,i+1)=t(rows\*i+1:rows\*(i+1));

end

end

%Fourier Transform of the split rfdata

T0=tshort(rows,1)-tshort(1,1);

N=rows;

ts=T0/N;

fs=1/ts;

freq= fs\* (0 : 1/N: 1-(1/N));

rfdatashortFT=abs(fft(rfdatashort));

rfdatashortPower=20\*log(rfdatashortFT);

figure(2)

plot (freq,rfdatashortFT)

title('Amplitude Response of Split RF Data')

xlabel('frequency (Hz)')

ylabel('amplitude')

%Finding the different center frequencies

for k=1:NumCols1\*cols

Index(k)=find(max(rfdatashortPower(1:round(length(freq)/2),k))==...

rfdatashortPower(1:round(length(freq)/2),k));

CenterFreq(k)=freq(Index(k));

end

%Averaging the center frequencies for each column but across all the rows

%in the column (chunks)

for Index2=1:cols

AverageFreq(Index2)=mean(CenterFreq(Index2:cols:NumCols1\*cols));

end

figure(3)

plot(AverageFreq)

xlabel('Number of Chunk')

ylabel('Center Frequency')

title('The Trend of Center Frequency with Respect to Depth')

%% Quadrature Demodulation

n=cols;

[row, cols]= size(rfdata);

% Find Power Spectrum to see what frequency to use

T0=(depth(length(depth))-depth(1))\*2/1540/1000;

N=row;

ts=T0/N;

fs=1/ts;

freq= fs\* (0 : 1/N: 1-(1/N));

DataFT=abs(fft(rfdata));

DataFTNorm=abs(DataFT)/max(max(DataFT));

PowerNorm=20\*log10(DataFTNorm);

figure(4)

plot(freq,PowerNorm)

title('Power Spectrum of RF Data Columns (A-Lines)')

xlabel('frequency (Hz)')

ylabel('power (dB')

%Choose Frequency

t=(depth\*2/1000/1540)';

SignalCos=zeros(NumRows,NumCols);

SignalSin=zeros(NumRows,NumCols);

%Multiply by Sine and Cosine

cols=3;

for i=0:cols-1

for k=1:NumCols

SignalCos(floor(i\*NumRows/3)+1:floor((i+1)\*NumRows/3),k)=...

cos(2\*pi\*AverageFreq(i+1)\*t(floor(i\*NumRows/3)+1:floor((i+1)\*NumRows/3)))...

.\*rfdata(floor(i\*NumRows/3)+1:floor((i+1)\*NumRows/3),k);

SignalSin(floor(i\*NumRows/3)+1:floor((i+1)\*NumRows/3),k)=...

sin(2\*pi\*AverageFreq(i+1)\*t(floor(i\*NumRows/3)+1:floor((i+1)\*NumRows/3)))...

.\*rfdata(floor(i\*NumRows/3)+1:floor((i+1)\*NumRows/3),k);

end

end

[row, cols]= size(rfdata);

%FT

SignalCosFT=fft(SignalCos);

SignalSinFT=fft(SignalSin);

figure(5)

plot(freq,abs(SignalCosFT))

title('Amplitude Response of RF Data Multiplied by Cosine')

xlabel('frequency (Hz)')

ylabel('Amplitide')

figure(6)

plot(freq,abs(SignalSinFT))

plot(freq,abs(SignalSinFT))

title('Amplitude Response of RF Data Multiplied by Sine')

xlabel('frequency (Hz)')

ylabel('Amplitide')

% Multiply by Rect/Filter

SignalCosFTFiltered=zeros(row,cols);

SignalSinFTFiltered=zeros(row,cols);

for i=1:cols

SignalCosFTFiltered(:,i)=SignalCosFT(:,i).\*(abs(freq)<1e7)';

SignalSinFTFiltered(:,i)=SignalSinFT(:,i).\*(abs(freq)<1e7)';

end

figure(7)

plot(freq,abs(SignalCosFTFiltered))

title('Amplitude Response of Filtered RF Data Multiplied by Sine')

xlabel('frequency (Hz)')

ylabel('Amplitide')

figure(8)

plot(freq,abs(SignalSinFTFiltered))

plot(freq,abs(SignalSinFTFiltered))

title('Amplitude Response of Filtered RF Data Multiplied by Sine')

xlabel('frequency (Hz)')

ylabel('Amplitide')

% Reconstruct/InverseFT

RealSignal=real(ifft(SignalCosFTFiltered));

ImaginarySignal=real(ifft(SignalSinFTFiltered));

figure(9)

SignalRecon=2\*sqrt(RealSignal.^2+ImaginarySignal.^2);

imagesc(lat, depth, SignalRecon)

xlabel('lateral direction (mm)')

ylabel('depth (mm)')

title('Image with Quadrature Demodulation including Depth Dependent Center Frequency')

axis image

colormap hot

colorbar

caxis([0 4e-21])

%% Contrast

SignalIntensity=mean(mean(SignalRecon(854:975,360:521)))

BackGroundIntensity=mean(mean(SignalRecon(1:300,1:300)))

ContrastDepth=(SignalIntensity-BackGroundIntensity)/(max(max(SignalRecon))-min(min(SignalRecon)))

**Problem 2.3**

%FinalProject2Carotid.m

clear;clf;close all

% Producing the original image

load('rf\_carotid.mat')

figure(1)

imagesc(lat,axial,double(rfdata))

xlabel('lateral direction (m)')

ylabel('axial direction (m)')

title('Original Carotid Artery Image')

colormap hot

axis image

% Hilbert Tranform

Transform=hilbert(double(rfdata));

DemodTransform=abs(Transform);

% Plotting the Hilbert Transform of the Image

figure(2)

imagesc(lat,axial,DemodTransform)

axis image

colormap gray

colorbar

xlabel('lateral direction (m)')

ylabel('axial direction (m)')

title('Carotid Artery Image with Only Hilbert Transform')

SignalIntensity=mean(mean(DemodTransform(708:817,116)));

BackGroundIntensity=mean(mean(DemodTransform(661:682,52:70)));

ContrastHilbert=(SignalIntensity-BackGroundIntensity)/(max(max(DemodTransform))-min(min(DemodTransform)))

% Plotting the log compressed data of Hilbert Transform

figure(3)

DemodTransformLog=log10(DemodTransform);

imagesc(lat,axial,DemodTransformLog)

axis image

colormap gray

colorbar

xlabel('lateral direction (m)')

ylabel('axial direction (m)')

title('Carotid Artery Image with Hilbert and Log Compression')

SignalIntensity=mean(mean(DemodTransformLog(708:817,116)));

BackGroundIntensity=mean(mean(DemodTransformLog(661:682,52:70)));

ContrastHilbertLog=(SignalIntensity-BackGroundIntensity)/(max(max(DemodTransformLog))-min(min(DemodTransformLog)))

% Normalizing the Hilbert Transformed Data

DemodTransformNorm = DemodTransform./max(max(DemodTransform));

%Plotting the histogram of the normalized data

figure(4)

hist (DemodTransformNorm)

xlabel('Intensity')

ylabel('Frequency of Occurances')

title('Histogram for Hilbert Transformed Carotid Image')

% Plotting Image Adjusted Verstion of the Normalized Data

figure (5)

ImAdjust=imadjust(DemodTransformNorm,[0, 0.1]);

imagesc(lat,axial,ImAdjust)

axis image

colorbar

colormap gray

xlabel('lateral direction (m)')

ylabel('axial direction (m)')

title('Carotid Artery Image with Hilbert and Imadjust')

SignalIntensity=mean(mean(ImAdjust(708:817,116)));

BackGroundIntensity=mean(mean(ImAdjust(661:682,52:70)));

ContrastImadjust=(SignalIntensity-BackGroundIntensity)/(max(max(ImAdjust))-min(min(ImAdjust)))

% Taking the log compressed of above

figure (6)

ImAdjustLog=log10(ImAdjust);

imagesc(lat, axial, ImAdjustLog)

axis image

colorbar

colormap gray

xlabel('lateral direction (m)')

ylabel('axial direction (m)')

title('Carotid Artery Image with Hilbert Imadjust and Log Compression')

SignalIntensity=mean(mean(ImAdjustLog(708:817,116)));

BackGroundIntensity=mean(mean(ImAdjustLog(661:682,52:70)));

ContrastImadjustLog=(SignalIntensity-BackGroundIntensity)/(max(max(ImAdjustLog))-min(min(ImAdjustLog)))

**Problem 2.4**

% FinalProject2TGC.m

%% Plotting the RF Data

clear;clf; close all

load ('rf\_to\_tgc.mat')

figure(1)

imagesc(lat, axial, RfData)

axis image

colormap gray

xlabel('lateral direction (m)')

ylabel('axial direction (m)')

title('Original Data')

[rows,cols]=size(RfData);

%% Hilbert Transform

Transform=hilbert(RfData);

DemodTransform=abs(Transform);

% Plotting the HilbertTransform

figure(2)

imagesc(lat, axial, DemodTransform)

axis image

colormap hot

colorbar

xlabel('lateral direction (m)')

ylabel('axial direction (m)')

title('Hilbert Transformed Image of Multiple Lesions')

SignalIntensity=mean(mean(DemodTransform(2250:2336,1:80)));

BackGroundIntensity=mean(mean(DemodTransform(631:693,161)));

ContrastDemod=(SignalIntensity-BackGroundIntensity)/(max(max(DemodTransform))-min(min(DemodTransform)))

% Finding the mean of every single row in the DemodTransform matrix

for k=1:rows

Average(k)=mean(DemodTransform(k,:));

end

% Normalization Factor

NormalFactor=max(Average)./Average;

% Multiplying everything by the normalziation Factor

for i=1:rows

DemodTransformTGC(i,:)=NormalFactor(i)\*DemodTransform(i,:);

end

% Plotting the time gain compensated (normalized) data

figure(3)

imagesc(lat, axial, DemodTransformTGC)

axis image

colormap hot

colorbar

xlabel('lateral direction (m)')

ylabel('axial direction (m)')

title('Hilbert Transformed and Time Gain Compensated Image of Multiple Lesions')

SignalIntensity=mean(mean(DemodTransformTGC(637:701,161)));

BackGroundIntensity=mean(mean(DemodTransformTGC(2190:2336,1:41)));

ContrastTGC=(SignalIntensity-BackGroundIntensity)/(max(max(DemodTransformTGC))-min(min(DemodTransformTGC)))

**Problem 3.1**

% dopplershift.m

% BME 154 Final Project

% Problem 3 Optional

% Use Doppler shift to calculate velocity of

%% load data

clear all;

close all;

file = 'A';

a = load(['mmode', lower(file), '.mat']); % load filename

b = load('mmodeb.mat');

c = load('mmodec.mat');

rows = dot(size(a.mmode), [1, 0]); % number of rows

cols = dot(size(a.mmode), [0, 1]); % number of columns

a.c = 1540; % set the speed of sound in tissue

% fourier transform of the time series data at a specific depth

fftdataA = fftshift(fft(a.mmode'));

cf = max(abs(fftdataA)); % maximum values of each power spectrum

cfind = zeros(size(cf)); % vector of indices of center frequency

% identify the center frequency for each row

for m = 1:length(cf)

cfind(m) = dot(find(abs(fftdataA(:,m)) == cf(m)), [0, 1]);

end

% finding the frequency range. Sampling frequency of the time

% series data is the pulse repeititon frequency

freq = linspace(-a.prf/2, a.prf/2, cols);

% % % % plot all power spectrums

% % % figure

% % % plot(freq, 20\*log10(abs(fftdataA)/max(max(abs(fftdataA)))))

% % % title(['(Dataset ', file, ') Power spectrum of all time series signal'])

% % % xlabel('Frequency (Hz)')

% % % ylabel('Magnitude (dB)')

% % %

% % % print -dpng powerspectrumtimeA

avgcf = mean(freq(cfind)); % finding the average center frequency

avgcf - freq(cfind(1));

v = a.c/(2\*a.f0)\*avgcf % calculate the velocities

%% Calculating direction of motion

fftdataRF = mean(fftshift(fft(a.mmode))');

fftdataRFb = mean(fftshift(fft(b.mmode))');

fftdataRFc = mean(fftshift(fft(c.mmode))');

cfRF = max(abs(fftdataRF));

cfRFb = max(abs(fftdataRFb));

cfRFc = max(abs(fftdataRFc));

cfindRF = dot(find(abs(fftdataRF) == cfRF), [0, 1]);

cfindRFb = dot(find(abs(fftdataRFb) == cfRFb), [0, 1]);

cfindRFc = dot(find(abs(fftdataRFc) == cfRFc), [0, 1]);

freqRF = linspace(-a.fs/2, a.fs/2, rows);

dir = freqRF(cfindRF) > a.f0 % test whether center frequency is greater than f0

% % % % plot all power spectrums of each RF line

% % % figure

% % % subplot(3,1,1)

% % % plot(freqRF, abs(fftdataRF), 'k-', [a.f0, a.f0], [0, 1.1\*cfRF], 'r-', ...

% % % freqRF(cfindRF), cfRF, 'ro')

% % % title('(Dataset A) Avg Fourier transform')

% % % xlabel('Frequency (Hz)')

% % % ylabel('Magnitude')

% % % axis([0, 0.5e7, 0, 1.2\*cfRF])

% % %

% % % subplot(3,1,2)

% % % plot(freqRF, abs(fftdataRFb), 'k-', [a.f0, a.f0], [0, 1.1\*cfRFb], 'r-', ...

% % % freqRF(cfindRFb), cfRFb, 'ro')

% % % title('(Dataset B) Avg Fourier transform')

% % % xlabel('Frequency (Hz)')

% % % ylabel('Magnitude')

% % % axis([0, 0.5e7, 0, 1.2\*cfRFb])

% % %

% % % subplot(3,1,3)

% % % plot(freqRF, abs(fftdataRFc), 'k-', [a.f0, a.f0], [0, 1.1\*cfRFc], 'r-', ...

% % % freqRF(cfindRFc), cfRFc, 'ro')

% % % title('(Dataset C) Avg Fourier transform')

% % % xlabel('Frequency (Hz)')

% % % ylabel('Magnitude')

% % % axis([0, 0.5e7, 0, 1.2\*cfRFc])

% % %

% % % print -dpng dopplershiftdirection

**Problem 3.2**

%mmodecorrv.m

% BME 154 Final Project

% Question 3

%% Initialization

clear all;

close all;

%% Load data

file = 'C';

a = load(['mmode', lower(file), '.mat']); % load filename

% % % figure

% % % subplot(1,3,1)

% % % imagesc(a.mmode)

% % % colormap gray

% % % title('Dataset A')

% % %

% % % subplot(1,3,2)

% % % imagesc(b.mmode)

% % % colormap gray

% % % title('Dataset B')

% % %

% % % subplot(1,3,3)

% % % imagesc(c.mmode)

% % % colormap gray

% % % title('Dataset C')

% % %

% % % print -dpdf 3velocityimages

% find the axial and time step lengths using linear regression

dx = dot(polyfit(1:length(a.axial), a.axial, 1), [1, 0]);

dt = dot(polyfit(1:length(a.T), a.T, 1), [1, 0]);

%% Calculating velocities

corrdata = xcorr(a.mmode(:,1), a.mmode(:,1));

Axcorrmat = zeros(size(corrdata)); % cross-correlation matrix

Acorrpeakloc = zeros(size(a.T)); % vector of peaks

for k=1:length(a.T)

Axcorrmat(:, k) = xcorr(a.mmode(:,1), a.mmode(:, k));

Axcorrpeakloc(k) = find(Axcorrmat(:, k) == max(Axcorrmat(:, k))) - length(a.mmode);

end

v = dot(polyfit(1:length(Axcorrpeakloc), Axcorrpeakloc, 1), [1, 0])\*dx/dt

%% Produce figures

x = (1:length(a.axial))\*dx;

% % comparison of two RF lines

% figure

% subplot(2,1,1)

% plot(x, a.mmode(:, 1), 'k-');

% title(['(Dataset ', file, ') Single RF line at t = 0s'])

% ylabel('Magnitude')

%

% subplot(2,1,2)

% plot(x, a.mmode(:,32), 'k-');

% title(['(Dataset ', file, ') Single RF line at t = 0.032s'])

% xlabel('Depth (m)')

% ylabel('Magnitude')

%

% print -dpng 3RFlinecompB

%

% % comparison of autocorrelation and xcorrlation

% x2 = (1:length(Axcorrmat(:,1)))\*dx;

% shift0 = Axcorrmat(:, 1);

% shift0max = find(shift0 == max(shift0));

% shift32 = Axcorrmat(:, 32);

% shift32max = find(shift32 == max(shift32));

%

% figure

% subplot(2,1,1)

% plot(x2, shift0,'k-', [1,1]\*shift0max\*dx, [-2e-36, 2e-36], 'r-')

% title(['(Dataset ', file, ') Autocorrelation of the RF line at t = 0s'])

% xlabel('Depth (m)')

% ylabel('Magnitude')

%

% subplot(2,1,2)

% plot(x2, shift32, 'k-', [1,1]\*shift32max\*dx, [-2e-36, 2e-36], 'r-')

% title(['(Dataset ', file, ') Cross-correlation of RF lines at t = 0 and 0.032s'])

% xlabel('Depth (m)')

% ylabel('Magnitude')

%

% print -dpng 3corrcompB

%

% % finding velocity

% figure

% plot((1:length(Axcorrpeakloc))\*dt, (Axcorrpeakloc)\*dx, 'k-')

% title(['(Dataset ', file, ') Correlation peak shifts through time'])

% xlabel('Time (s)')

% ylabel('Depth(m)')

% annotation('textbox',...

% [0.516071428571428 0.75852380952381 0.0767857142857143 0.0666666666666667],...

% 'String',{['v = ', num2str(v), ' m/s']}, 'LineStyle', 'none');

%

% print -dpng velocityplotB

**Problem 4.1**

% BME154L\_FinalProject\_Question4.m

% Matlab Sample Code

% BME 154 Final Project

tic

%% Initialization

clear all;

close all;

%% Load data

fileid=fopen('BME154L\_S12\_PROJECT\_ECG.bin'); % code for opening .bin files

data = fread(fileid,inf,'float32');

fclose('all');

%% Define variables

t=data(1:2:end); % (seconds)

ecg= data(2:2:end); %(Voltage [mV])

ecg\_raw = ecg;

% find number of points for 0.4 seconds. used later for 'minpeakdistance'

num\_points = round(0.4/(t(end)/length(t)));

%% Plot of Original ECG

figure(1)

plot(t(1200:1600),ecg(1200:1600)) % one ecg cycle isolated

xlabel('Time (s)')

ylabel('Voltage (mV)')

title('ECG Data')

print -dpng part4fig1

%% COMPUTING SNR?

% uses function ECGsnr.m

raw\_snr = ECGsnr(ecg\_raw, 1000:20000); % indices 1000:20000 chosen as a random representation of normal signal

%% Removing Power Noise

% Compute FFT %used the fourier transform to get the power spectrum

fs=1/mean(diff(t));

ft=fft(ecg);

f = linspace(-fs/2,fs/2,length(ft));

shiftFT = fftshift(ft);

figure

plot(f, 20\*log10(abs(shiftFT)./max(abs(shiftFT))))

axis([0 100 -80 0]);

xlabel('Frequency (Hz)')

ylabel('Relative Power (dB)')

title('ECG FFT Power Spectrum');

print -dpng part4fig2

% Removing Artifact with notch filter

for index = 1:length(f)

if abs(f(index))>59.75 && abs(f(index))<60.25

shiftFT(index) = 0;

end

end

figure

plot(f, (20\*log10(abs(shiftFT)./max(abs(shiftFT)))))

axis([0 100 -80 0])

xlabel('Frequency (Hz)')

ylabel('Relative Power (dB)')

title('ECG FFT Power Spectrum w/Noise Removed');

print -dpng part4fig3

ecg = real(ifft(fftshift(shiftFT)));

% Demonstration of Signal Improvement

figure

subplot(2,1,1)

plot(t, ecg\_raw)

axis([620 621.5 -4 10])

xlabel('Time (s)'); ylabel('Voltage (mV)');

title('Raw ECG Data')

subplot(2,1,2)

plot(t, ecg)

axis([620 621.5 -4 10])

xlabel('Time (s)'); ylabel('Voltage (mV)');

title('ECG Data w/ Power Noise Artifact Removed')

print -dpng part4fig4

%% Remove DC Offset and Linear Tilt From Data

% Removing any Offset

p = polyfit (t, ecg, 1); %fit a line to the data

tilt = p(1)\*t+p(2);

clean\_ecg = ecg-tilt;

figure % this graph shows the signal with tilt removed

plot(t,clean\_ecg);

xlabel('Time (s)');

ylabel('Voltage (mV)');

title ('ECG Data with DC Offset and Linear Tilt Removed')

axis([620 621.5 -4 10])

print -dpng part4fig5

% Compute Clean FFT

ft\_clean = fft(clean\_ecg);

figure

plot(f, fftshift(20\*log10(abs(ft\_clean)./max(abs(ft\_clean)))));

axis([0 100 -80 0]);

xlabel('Frequency (Hz)');

ylabel('Relative Power (dB)')

title('Clean ECG FFT Power Spectrum')

print -dpng part4fig6

noartifact\_snr = ECGsnr(clean\_ecg,1000:20000);

%% Bradycardia Identification

[~, locs] = findpeaks(clean\_ecg, 'MINPEAKHEIGHT', 0.7\*max(clean\_ecg), 'MINPEAKDISTANCE', num\_points);

beattime = zeros(1, length(locs)-1);

for index = 1:length(locs)-1

beattime(index) = t(locs(index+1))-t(locs(index));

end

figure

plot(t(locs), clean\_ecg(locs), 'rx')

hold on

plot(t, clean\_ecg)

xlabel('Time (s)'); ylabel('Voltage (mV)');

title('Clean ECG with Peaks Identified')

axis([620 630 -3 10])

print -dpng part4fig7

HR = 60./beattime;

figure

plot(t(locs(1:end-1)), HR, '-')

xlabel('Time (s)'); ylabel('Instantaneous Heart Rate (bpm)')

title('Heart Rate for Individual Beats')

print -dpng part4fig8

counter = 0;

for index = 1:length(HR)-1

if HR(index) < 50 && HR(index+1) < 50

counter = counter+1;

bcardia\_locs(counter) = locs(index-1);

end

end

% t(bcardia\_locs) gives the times when bradycardia occurs.

%% Maximizing SNR of ECG data using Boxcar Averager

boxcar\_windowsizes= [2 3 5 6 7 8 9 10 15 20]; % number of samples

for i = 1:length(boxcar\_windowsizes)

ecg\_boxcar(i,:) = conv2(clean\_ecg,ones(boxcar\_windowsizes(i),1),'same')./boxcar\_windowsizes(i);

end;

%

% for i=1:length(boxcar\_windowsizes)

% figure(20+i)

% plot(t,ecg\_boxcar(i,:))

% axis([620 621.5 -2 8])

% title(sprintf('Boxcar Window = %i samples', boxcar\_windowsizes(i)));

% xlabel('Time (seconds)'); ylabel('Voltage (mV)');

% end

% % From observation it appears a boxcar window size of 8 is ideal for

% % smoothing out noise but maintaining signal power

boxcarECG = ecg\_boxcar(6,:);

figure

subplot(2,1,1)

plot(t, clean\_ecg)

axis([620 630 -2 8])

xlabel('Time (seconds)'); ylabel('Voltage (mV)');

title(sprintf('Clean ECG before Boxcar'));

subplot(2,1,2)

plot(t,boxcarECG)

axis([620 630 -2 8])

xlabel('Time (seconds)'); ylabel('Voltage (mV)');

title(sprintf('Boxcar Window = 8 samples'));

print -dpng part4fig9

boxcar\_snr = ECGsnr(boxcarECG,1000:20000);

%% Cross Correlation Heartbeat Detection (Didn't end up using this method)

% % % % ecg\_reference = boxcarECG(1200:1600); % based on observation

% % % % [xco, lags] = xcorr(boxcarECG,ecg\_reference);

% % % %

% % % % % % % % % % figure(10)

% % % % % % % % % %

% % % % % % % % % % hold on;

% % % % % % % % % % plot(t, clean\_ecg);

% % % % % % % % % % xlabel('Time (s)');

% % % % % % % % % % ylabel('Voltage (mV)')

% % % % % % % % % % title('ECG Signal with Cross Correlation Heartbeat Detection')

% % % %

% % % % % Find Local Maxima in Cross Correlation

% % % % % get rid of noise

% % % %

% % % % % % % w=1500;

% % % % % % % xco\_bca = conv2(xco,ones(w,1),'same')./w;

% % % % % find peaks

% % % % figure(10)

% % % % plot((t)+t(201), (xco(length(t):end)))

% % % % hold on

% % % % [pks,pk\_locs] = findpeaks(xco,'minpeakheight',631, 'minpeakdistance', 200); % observation threshold

% % % % [pks2,pk\_locs2] = findpeaks(xco,'minpeakheight',520,'minpeakdistance', 200); % observation threshold

% % % %

% % % % pk\_locs = pk\_locs+200-length(t);

% % % % pk\_locs2 = pk\_locs2+200-length(t);

% % % %

% % % % PVClocs = setdiff(pk\_locs2, pk\_locs);

% % % %

% % % % % heartbeats = t(pk\_locs);

% % % % % plot(t(pk\_locs), xco(pk\_locs), 'rx')

% % % %

% % % % % unsure how to perfectly lineup correlation, actual data

% % % % plot((t(pk\_locs)), (pks), 'rx')

% % % %

% % % % xlabel('Time (s)'); ylabel('Correlation Area');

% % % % title('ECG data Correlated w/ Reference ECG, Peaks Identified')

% % % %

% % % % % plot(heartbeats, ones(length(heartbeats),1),'rx','MarkerSize', 4,'LineWidth',1);

%% Negative Threshold PVC Detection

% Make a fair assumptiont that no PVC will occur in the first few cycles of the data

% can be assumed min(boxcarECG) will be negative as the ECG has been centered at zero

threshold = min(boxcarECG(1:2000))-3;

[dips,dip\_locs] = findpeaks(-boxcarECG, 'minpeakheight', -threshold, 'minpeakdistance',num\_points);

dips = -dips;

% PVC times are given by t(dip\_locs)

%% AVERAGE HEART RATE OF NSR

A = [];

B = [];

for index = 1:length(HR)

% selecting PVC times to later be removed

if sum(abs(t(locs(index+1))-t(dip\_locs))<0.3)>0 || sum(abs(t(locs(index))-t(dip\_locs))<0.3)>0

A = [A index];

end

end

for index = 3:length(HR)+2

% selecting bradycardia times to later be removed

if sum(t(locs(index-1))==t(bcardia\_locs))>0||sum(t(locs(index-2))==t(bcardia\_locs))>0

B = [B index];

end

end

NSRHR = HR;

NSRHR([A B]) = [];

tNSR = t(locs(1:end-1));

tNSR([A B]) = [];

figure

plot(NSRHR,'-\*')

xlabel('Time (s)'); ylabel('Instantaneous Heart Rate (bpm)')

title('NSR Region Instantaneous Heart Rate over Time')

print -dpng part4fig10

% overall mean Heart Rate

meanHR = mean(NSRHR);

%% 60 second running average HR

% find 60 seconds (roughly)

register = ceil(60\*(length(locs))/t(end));

% could have used convolution as follows:

% run\_avgHR = conv(NSRHR,ones(register,1)/register,'same');

run\_avgHR = zeros(1,length(NSRHR)-register);

run\_avgHR(1) = mean(NSRHR(1:register));

% register acts

for index3 = 2:length(NSRHR)-register

run\_avgHR(index3) = run\_avgHR(index3-1)+(-NSRHR(index3-1)+NSRHR(index3+register))/register;

end

figure

plot(t(locs(1:length(run\_avgHR))),run\_avgHR)

xlabel('Time (s)'); ylabel('Heart Rate (bpm)')

title('Sixty Second Running Average of Heart Rate over Time')

print -dpng part4fig11

%% Average Over Different Heartbeat Using Phase Alignment

% Estimated the PR interval using the method of QRS peak alignment and

% flipping

maxqrs = max(clean\_ecg); % find maximum qrs value

[peaks, qrsloc] = findpeaks(clean\_ecg, 'minpeakheight', 0.6\*maxqrs, 'minpeakdistance', num\_points);

[minpeaks, ~] = findpeaks(-clean\_ecg, 'minpeakheight', 2, 'minpeakdistance', num\_points);

% parse signals and zero-pad

meshint = max(diff(qrsloc)) + 1; % finding the largest interval

ker = zeros(1, meshint); ker(end) = 1; % create a delta function to correlate

parsearray = zeros([length(qrsloc), length(xcorr(ker, ker))]); % array containing all parsed cycles

for k = 1:length(qrsloc)-1

% zero-padding using xcorr. parsearray rows are each qrs to qrs cycle

parsearray(k, :) = xcorr(flipud(clean\_ecg(qrsloc(k):qrsloc(k+1))), ker);

end

% cut off extraneous zero-padding

parsearray = parsearray(:, 1:meshint);

% create an array where each row is the mean cycle

avgarray = ones([length(qrsloc), 1])\*mean(parsearray);

% parsed cycle adjusted by mean

adjarray = parsearray - avgarray;

figure

plot(mean(diff(t))\*(1:length(mean(parsearray))),fliplr(mean(parsearray)))

xlabel('Scaled Time (s)'); ylabel('Voltage (mV)');

title('QRS Peaks aligned, averaged, and flipped')

axis([1.6 2.8 -1 7])

print -dpng fig20

%% The following is based on PS 7 solutions and is used to check the mean PR interval

% cross corrlation heart beat detection

ecg\_reference = clean\_ecg(1200:1600); % based on observation

[xco,lags] = xcorr(clean\_ecg,ecg\_reference);

figure;

hold on;

plot((t)+t(201), (xco(length(t):end)))

xlabel('Time (s)');

ylabel('Voltage (mV');

title('ECG Signal with Cross Correlation Heartbeat Detection')

% find peaks

[pks,pk\_locs]=findpeaks(xco,'minpeakheight',500,'minpeakdistance',num\_points);

pk\_locs = pk\_locs+200-length(t);

plot((t(pk\_locs)), pks, 'rx')

print -dpng part4fig12

% average over different heart beat using correlation-based phase alignment

ecg\_width = length(ecg\_reference)-1; % based on our earlier reference signal

for i=1:length(pk\_locs),

if((pk\_locs(i)- ecg\_width/2) < 1)

zero\_pad = zeros(pk\_locs(i),1);

ecg\_corr\_ave(i,:) = [zero\_pad; clean\_ecg(1:pk\_locs(i)+ecg\_width/2)];

elseif((pk\_locs(i) + ecg\_width/2) > length(clean\_ecg)),

zero\_pad = zeros(ecg\_width/2-(length(clean\_ecg)-pk\_locs(i)),1);

ecg\_corr\_ave(i,:) = [clean\_ecg(pk\_locs(i)-ecg\_width/2:end); zero\_pad];

else

ecg\_corr\_ave(i,:) = clean\_ecg((pk\_locs(i)-ecg\_width/2):(pk\_locs(i)+ecg\_width/2));

end

end

figure

imagesc(t(1:size(ecg\_corr\_ave,2)),1:length(locs), ecg\_corr\_ave);

xlabel('Time (s)');

ylabel('Heart Beat');

title('Cross Correlation, Phase-Aligned ECG Signals');

print -dpng part4fig13

figure

plot(t(1:size(ecg\_corr\_ave,2)),mean(ecg\_corr\_ave,1)');

xlabel('Time (s)');

ylabel('Voltage (mV)');

title('Mean ECG Signal (Xcorr, Phase-Aligned)');

print -dpng part4fig14

toc

**Problem 4.2**

%ECGsnr.m

% BME 154 final project

% Problem 4

% Finding ECG SNR

function snr = ECGsnr(signal, ind)

%% load data

n = ind;

ecg = signal(ind);

% % % fileID = fopen('ecgdata.bin');

% % % data = fread(fileID, inf, 'float32');

% % % fclose('all');

% % %

% % % t = data(1:2:3400\*2);

% % % ecg = data(2:2:3400\*2);

% % % ecg = detrend(ecg, 'linear');

% % % cleanecg = load('cleanECG.mat')

% % % t = cleanecg.t(n);

% % % ecg = cleanecg.clean\_ecg(n);

%% COMPUTING SNR

maxqrs = max(ecg); % find maximum qrs value

[~, qrsloc] = findpeaks(ecg, 'minpeakheight', 0.8\*maxqrs, 'minpeakdistance', 100);

% parse signals and zero-pad

meshint = max(diff(qrsloc)) + 1; % finding the largest interval

ker = zeros(1, meshint); ker(end) = 1; % create a delta function to correlate

parsearray = zeros([length(qrsloc), length(xcorr(ker, ker))]); % array containing all parsed cycles

for k = 1:length(qrsloc)-1

% zero-padding using xcorr. parsearray rows are each qrs to qrs cycle

parsearray(k, :) = xcorr(ecg(qrsloc(k):qrsloc(k+1)), ker);

end

% cut off extraneous zero-padding

parsearray = parsearray(:, 1:meshint);

avgsig = mean(parsearray);

% create an array where each row is the mean cycle

avgarray = ones([length(qrsloc), 1])\*avgsig;

% parsed cycle adjusted by mean

noisearray = parsearray - avgarray;

% % % figure

% % % plot(mean(parsearray))

% % % title('averaged signal')

% % %

% % % figure

% % % plot(parsearray')

% % % title('signals')

% % %

% % % figure

% % % plot(noisearray(1:5, :)')

% % % title('noise signal')

figure

subplot(2,1,1)

plot(mean(parsearray))

title('averaged signal')

subplot(2,1,2)

plot(noisearray(1:5,:)')

title('noise signal')

axis([0 800 -.6 0.6])

print -dtiff fig70

% fair to assume noise can be measured within an area we assume to be

% centered at zero

noise\_adj = detrend(noisearray(:,100:400)');

% % % figure

% % % plot(noise\_adj)

% % % title('noise centered at zero')

% find the standard deviation of the adjusted cycles

noise = sqrt(mean2(noise\_adj.^2)); %2\*mean(std(noisearray(:,200:400)'))

% find the average maximum signal range

signal = sqrt(sum(sum(parsearray.^2))/length(ecg)'); %max(avgsig) - min(avgsig) %sqrt(mean(ecg.^2))

snr = 20\*log10(signal./noise);

end