P-Wave Electrical Alternans as an Indicator of Paroxysmal Atrial Fibrillation

by

Nikhil Iyengar

Bachelor of Science, Electrical Engineering and Computer Science
Massachusetts Institute of Technology, 1998

Submitted to the
Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 1999

Author:_____

Department of Electrical Engineering and Computer Science
January 21, 1999

Certified by:_____

Richard J. Cohen, M.D., Ph.D.
Professor, Harvard-MIT Division of Health Sciences and Technology
Thesis Supervisor

Accepted by:_____

Arthur C. Smith
Chairman, Department Committee on Graduate Theses

P-Wave Electrical Alternans as an Indicator of Paroxysmal Atrial Fibrillation

by

Nikhil Iyengar

Submitted to the Department of Electrical Engineering and Computer Science
on January 21, 1999, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

P-wave electrical alternans is a beat to beat alternation in the morphology of the P-wave, the wave in an electrocardiogram corresponding to atrial depolarization. In this investigation, software to measure p-wave alternans was developed and then tested on simulated data and on real data previously recorded from human subjects. The software consists of C programs managed by shell scripts and contains several features to minimize user interaction in data processing. The software can also calculate a signal averaged P-wave, apply various filters, and find the P-wave duration.

Thesis Supervisor: Richard J. Cohen, M.D., Ph.D.
Title: Professor, Harvard-MIT Division of Health Sciences and Technology

Acknowledgments

I would like to thank my thesis supervisor Prof. Richard Cohen for his guidance, advice, and encouragement. I would also like to thank Antonis Armoundas for his very generous help with all aspects of my thesis. It has been a pleasure to work with him. I am grateful for the valuable assistance and advice provided by other members of the Cohen lab, including Nikolai Aljuri, Paul Belk, Yuri Chernyak, Ki Chon, Andrew Feldman, Ming Maa, Ramakrishna Mukkamala, Tom Mullen, and Derin Sherman. Additionally, Joseph Galvin provided valuable discussion about the clinical aspects of this thesis. Most of all, I would like to thank my parents for their constant encouragement and support and for always being there when I needed them.

Contents

List of Figures

Chapter 1

Introduction

Atrial fibrillation (AF) is an arrhythmia in which the atria enter into a condition of uncoordinated muscular activity and are unable to perform their function of pumping blood to the ventricles. AF is not an acute medical condition, and therefore it is possible for someone to have the condition yet not have it affect his or her daily life. Since the atria are not functioning, the ventricles do not fill as completely and cardiac output is decreased slightly. More importantly, a patient with AF has 5 to 6 times the chance of stroke as a person without AF due to pooling of blood in the atria [23]. This pooling of blood may result in the formation of clots which enter the systemic circulation and become lodged in blood vessels in the brain. However, once AF has been detected, it can be treated with antiarrhythmic drugs which can decrease the possibility of stroke [18, 23].

1.1 Objective

The primary objective of this thesis was to develop the algorithms and software necessary to measure P-wave alternans and the duration of the P-wave signal averaged ECG. This software was tested on a few samples of patient data to verify its correctness and robustness. The software can be used to perform a larger clinical study of

P-wave alternans.

Measurement of P-wave alternans is a multi step process, modeled after previously developed algorithms for the measurements of T-wave alternans. The process involves detecting the QRS peaks, aligning the beats on those peaks using an autocorrelation method, and then aligning on the P-waves using another series of autocorrelations. To aid in the evaluation of the clinical predictive value of P-wave alternans, the signal averaged P-wave was also calculated.

Chapter 2

Background

In this section, the anatomy of the human heart and its electrophysiology is discussed to allow an understanding of atrial fibrillation and P-wave alternans.

2.1 Anatomy and Electrophysiology of the Heart

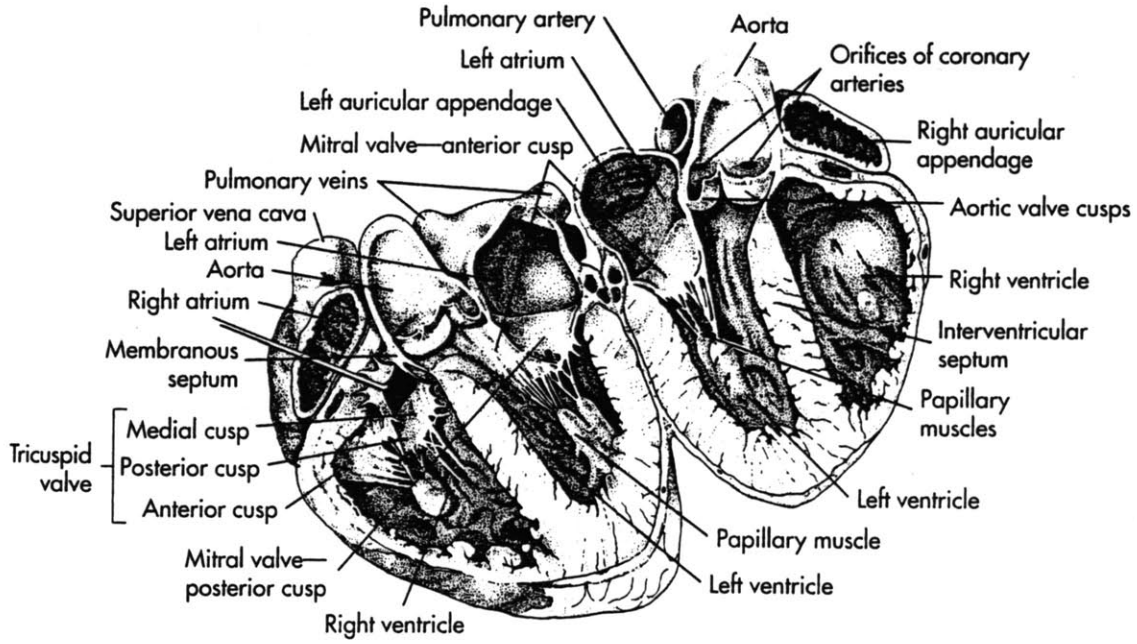
The human heart is a hollow, thickly muscular organ which facilitates blood flow through the vascular system by rhythmic contraction. The heart is responsible for supplying all organs of the body with blood, which provides them with necessary nutrition and oxygen.

The heart consists of four chambers as shown in Figure 2-1: the left and right ventricles and the left and right atria. The right side of the heart receives blood from the systemic (main body) circulation and pumps it to the lungs for oxygenation. The left side of the heart receives the blood from the lungs and pumps it back to the systemic circulation.

The ventricles are responsible for the primary pumping action as they pump blood out of the heart. The atria act as receiving chambers for blood entering the heart.

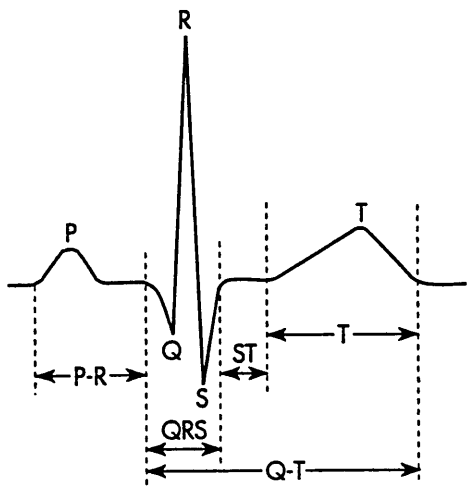
A typical cardiac cycle consists of filling of the atria, contraction of the atria, filling of the ventricles, and contraction of the ventricles, in sequence. These contrac-

Figure 2-1 Anatomy of the Heart (From [1])



tions are achieved through a wave of muscular depolarization traveling through the heart which causes contraction of myocardial muscle. The electrical impulses begin in the heart's natural pacemaker, the sinoatrial node, located in the right atrium. An impulse then travels to the atria and results in their contraction. Conduction continues through the atrioventricular node and finally to the ventricles [1]. This wave of conduction occurring on the cellular level can be summed to a single overall three dimensional heart vector which can be projected onto various lines, or leads going across the surface of the body. The projection onto a particular lead over time is called an electrocardiogram (ECG). A typical ECG is shown in Figure 2-2. This ECG is shown over one cardiac cycle, or one period of contraction and relaxation of the heart. The first wave, the P wave, shows the electrical activity associated with

Figure 2-2 A Typical ECG showing P, QRS, and T waves (From [1]).



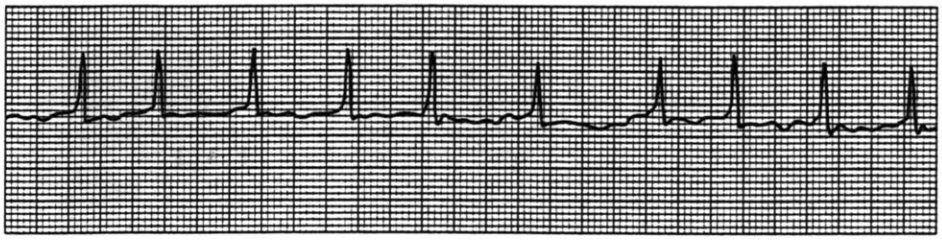
atrial depolarization, which results in contraction of the atria. After a slight pause in the A-V node, the depolarization wavefront proceeds to the ventricles through the heart's conduction system as described above. The depolarization and resulting contraction of the ventricles is visible on the ECG as the QRS complex. Afterward, the T wave results from the repolarization of the ventricles as they prepare for the next wave of depolarization. The atria also repolarize; however, this waveform is not visible on the ECG because atrial repolarization often overlaps with the QRS and is thus obscured by this much larger waveform [1].

2.1.1 Atrial fibrillation

The description given above is one of a normal heart in which the atria and the ventricles are pumping in a coordinated manner. However, various dysfunctions in the heart's rhythm or conduction may be present and are termed arrhythmias.

The main arrhythmia of interest in this study is atrial fibrillation, a very common arrhythmia which affects 0.5-1% of the general population rising to over 10 percent

Figure 2-3 ECG Showing Atrial Fibrillation (from [1]).



in people over 70 [18]. Fibrillation is a condition which can occur in either the atria or the ventricles. In fibrillation, the heart muscle does not undergo a coordinated contraction, but instead, various regions of the muscle contract in a disorganized, uncoordinated fashion [1]. While normal function may be regarded as a singular contraction of the muscle, fibrillation more closely resembles a quivering of the muscle. In this state, the muscle is completely unable to contract and is rendered useless for the purpose of pumping blood. A ECG of a patient in atrial fibrillation is shown in Figure 2-3. Note the absence of P-waves.

When such a condition occurs in the ventricles, it is life threatening. Ventricular fibrillation is almost immediately followed by unconsciousness and death will occur within minutes if the condition persists. This is because the ventricles are the primary agents which pump blood to the body, including the brain.

Atrial fibrillation, on the other hand, is less life threatening. Since the atria serve to help filling of the ventricles, loss of atrial function results in a less efficient overall pumping of the heart. This loss is minor in most people and may only have a real effect during physical exertion or when combined with heart failure [1, 4]. The main symptoms of atrial fibrillation are palpitations or a sensation of erratic or irregular heart beating. In some cases, patients may be unaware that he or she has AF [1].

The main danger of AF is the increased incidence of stroke [1, 13, 18]. Because the atria are not pumping effectively, blood may pool in the atria and form blood

clots. These blood clots may eventually enter the systemic circulation and go to the brain, where they cause a stroke. Studies have shown a fivefold to sixfold increase in the rate of stroke associated with AF [13]. Overall, AF is responsible for more hospital admissions and days spent in hospitals in the United States than any other arrhythmia [18].

Once AF has been diagnosed, it can be treated directly with antiarrhythmatic drugs which keep the atria from fibrillating. Another method of treatment is to prescribe anticoagulants to prevent blood clots from forming, therefore preventing stroke [18].

2.1.2 Paroxysmal Atrial Fibrillation and Vulnerability

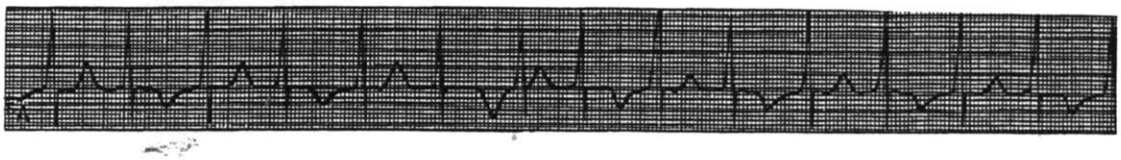
Paroxysmal Atrial Fibrillation (PAF) is often a precursor to atrial fibrillation. Patients with this arrhythmia go in to AF under certain conditions or may have seemingly random sporadic episodes of AF. At other times their atria beat normally. Currently, there is no effective treatments to give to patients with PAF to prevent the development of chronic AF [9,13].

The causes of PAF and sustained AF are not completely known. However, these conditions are often found in people with organic diseases of the heart or with a mitral valve defect [9].

Atrial fibrillation is thought to be analogous to ventricular fibrillation in its origins. Atrial fibrillation is thought to be due to a reentrant mechanism in which the atria require areas of slow conduction to initiate and maintain the reentrant circuit [13,18].

In this thesis, we postulate that the heart has a certain vulnerability, or susceptibility, to atrial fibrillation based on the character of the atrial muscle. In most adults, this susceptibility is extremely low, but in a certain elderly population the susceptibility is higher. It is especially high in patients with PAF, making them likely to become chronic AF.

Figure 2-4 ECG showing T wave alternans. Here the ABABAB... pattern is clearly visible (from [2].



If one could measure the susceptibility to AF of a certain individual, one could (1) test the efficacy of different medicines in preventing the onset of AF and (2) diagnose patients with PAF (who are unaware of their condition) or patients with a high atrial susceptibility so that may be monitored more closely in the future. There is currently no convenient method available to predict efficacy of one medication over another in a given individual [18].

2.2 P-Wave Alternans

One of the aims of this thesis is to determine whether P-wave alternans can be used as a measure of atrial susceptibility. First, some background on P-wave alternans will be given.

Alternans is a beat to beat variation in the morphology of a certain waveform, such as the P, QRS, or T. The waveforms should alternate in an ABABAB... pattern. A case of T wave alternans is shown in Figure 2-4. Alternans has long been recognized in the electrocardiogram. The first record of this was in Feigenbaum who described large beat to beat alternans in the T wave. T wave alternans have since been researched extensively . It has been found that T wave alternans correspond to a ventricular instability which may be predictive of ventricular fibrillation [5, 16, 17, 20]. Patients with a high T-wave alternans level may choose to have a defibrilator or some other

Figure 2-5 P-wave alternans example (from [7]).

intervention to prevent the onset of ventricular fibrillation.

P-wave alternans has also been documented though its clinical significance is highly unclear [3, 7, 15]. Since T wave alternans is associated with ventricular instability and fibrillation, it is possible that P wave alternans are associated with similar problems in the atria, as will be explored in this thesis. Figure 2-5 depicts very large P-wave alternans visible to the naked eye. However, examples of T-wave and especially P-wave alternans which are visible to the naked eye are very rare. Thus more sophisticated methods must be used to detect and quantify alternans which may be a result of fluctuations on the microvolt level.

2.3 T-wave Alternans measurement

A technique for measuring these microvolt potentials has previously been developed for use with T-wave alternans [17, 20]. The software described in this thesis will use a similar technique for the measurement of P-wave alternans. However, as will

be described later, several modifications were required because of the nature of the P-wave.

T-wave alternans measurement is often done by aligning a series of several successive beats on their QRS complexes to form a two dimensional array of points as shown in Figure 2-6. Next an fast Fourier transform (FFT) is taken across the beats on a point by point basis, as illustrated. Thus, alternations occurring in an ABABABAB fashion should result in a high power content at the Nyquist frequency. If we let the sampling frequency be 1 Hz, then the Nyquist frequency will be 0.5 Hz the power at that frequency will be alternans.

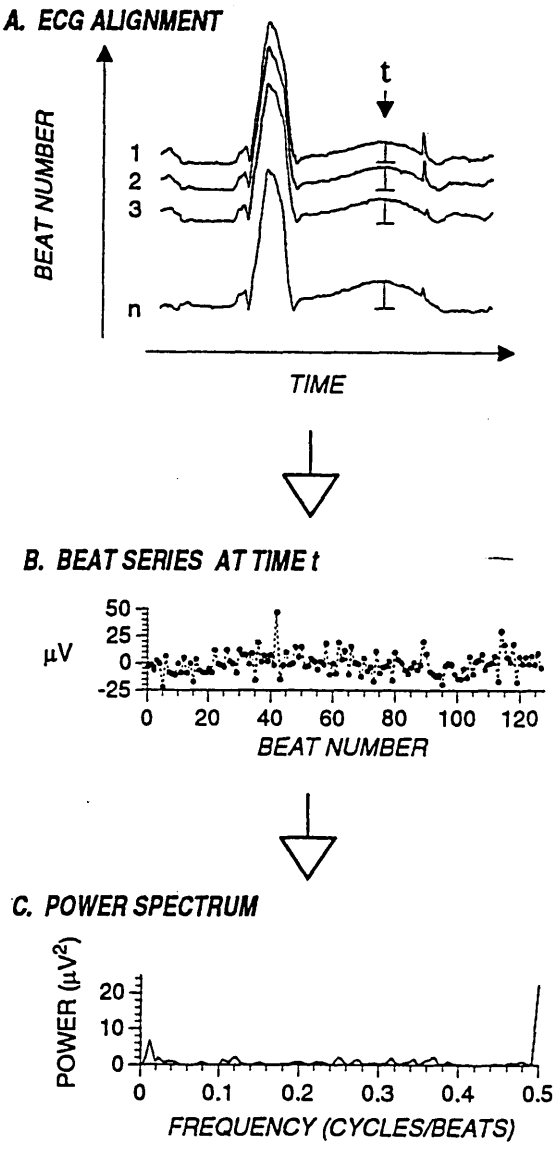
2.4 P-wave Signal Averaged ECG

A signal averaged ECG (SAECG) is a signal which is generated by averaging a sequence of beats on a given reference point (for example on the QRS peak). The purpose of the SAECG is to produce a "high resolution" beat by averaging out noise which is uncorrelated from beat to beat [10]. Thus, a 128-beat SAECG will have a noise variance 1/128th that of a single beat from an ECG.

The P-wave Signal Averaged ECG has been used in a manner similar to how this thesis proposes to use P-wave alternans. A P-wave SAECG is a SAECG where the P-wave is the focus of the signal averaging (the goal is to obtain a high resolution P-wave). A P-wave SAECG is necessary for many types of P-wave analyses since the P-wave is a small signal relative to the noise level, and is difficult to analyze unless noise is decreased significantly. To compute an SAECG, several beats are aligned on a reference and the average of all beats is taken. For the best results, alignment should be on the P-wave, since there may be a beat to beat variation in the PR interval, resulting in some jitter when averaging P-waves [4, 8, 9].

Once the SAECG is obtained, certain measurements can be made. Two methods

Figure 2-6 T-wave alternans calculation, (from [16])



relevant to assessing vulnerability to AF will be discussed here. First, as mentioned above, human studies have shown that intra-atrial conduction delays and short right atrial refractory periods characterize patients with atrial fibrillation. The result is an elongated P-wave or the presence of high frequency components at the tail end of the P-wave. An automated algorithm, as discussed later, can be used to determine the onset and offset of the P-wave in a SAECG. The SAECG can therefore be used to measure atrial instability and to predict the onset of atrial fibrillation [4, 8].

Before measuring the P-wave length, the P-wave should be band pass filtered, as this has been shown to make the difference of measurements between PAF and normal patients much more significant. Several filters have been previously tested and compared on the basis of sensitivity, specificity, predictive accuracy, and p-value when used to predict AF [8]. Filtering can serve several functions: eliminate contamination by low frequency artifact, ensure isolated detection of depolarization (without contamination by repolarization), and enhance detection of onset and offset of a low amplitude signal like the P-wave. Onset and offset are easier and more consistently detected when low frequencies are removed because the baseline on both sides of the P-wave is set to nearly zero and the P-wave stands out much more easily (see Figure 3-7). A high pass filter would also enhance the high frequency components late in the P-wave thought to be associated with atrial instability [8].

A detailed discussion of the filters which were chosen for implementation are discussed in the next chapter.

Chapter 3

Methods

This chapter will discuss the implementation of the P-wave alternans and P-wave SAECG software.

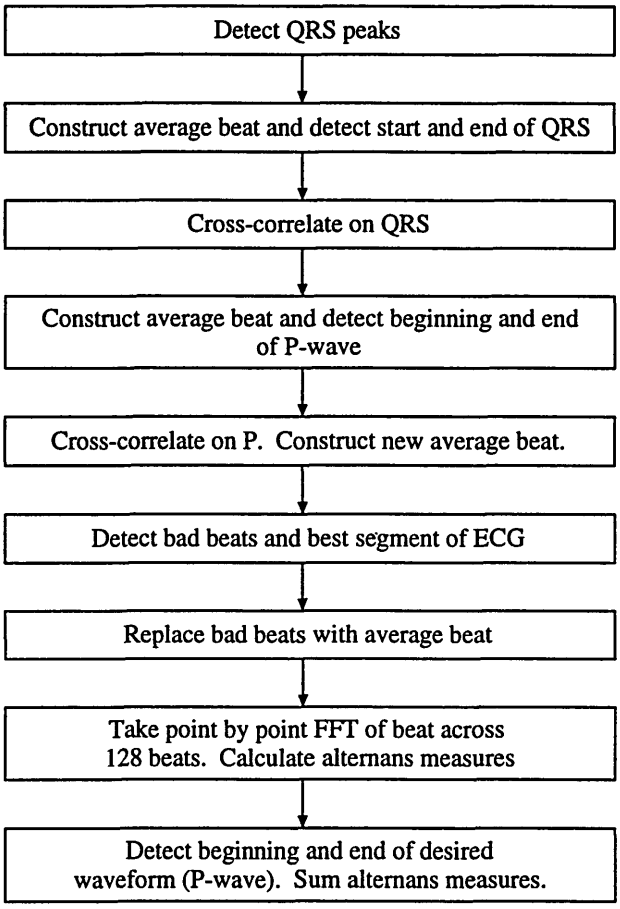
3.1 Data Input and Vector Magnitude

The input to the program is electrocardiogram (ECG) data. Currently, the supported input type are the three Frank leads, X, Y, and Z. Formulas to approximately convert 12-lead ECGs to Frank Lead ECGs have been published.

Data preparation consists of detecting the QRS peaks in the ECG and of constructing a vector magnitude ECG. In the Frank lead system, the three leads represent orthogonal projections of the heart vector so the vector magnitude can be easily computed as follows:

$$VectorMagnitude = \sqrt{X^2 + Y^2 + Z^2} \tag{3.1}$$

Figure 3-1 Overview of steps in calculating P-wave alternans



3.2 Summary of Analysis

As described in the previous section, two analyses are performed: P-wave alternans measurement and P-wave signal averaged ECG. Figure 3-1 is a summary of the steps to calculate P-wave alternans. The steps for the P-wave SAECG will be presented later.

3.3 Detect QRS peaks

The ECG peak detection was done on the individual leads rather than on the vector magnitude because the individual leads often contained sharper ECG peaks and

therefore were less prone to causing missed detection of beats. The software to detect the peaks was written previously and is based on standard methods for ECG peak detection. In short, it band pass filters the ECG and then attempts to detect a slope above a certain threshold and below another threshold (to exclude glitches, etc.). It then finds the peak value and annotates this as the peak of the ECG. There are also some more sophisticated criteria used to decrease the number of false positives such as a minimum and maximum peak length (in milliseconds) and a lockout time after peak detection when no other peaks can be detected.

Although the algorithm is successful greater than 99% of the time, it does fail on occasion. For this reason, the detected peaks should be reviewed by a human to determine that no peaks have been left undetected and to make sure that no false positives are present. This is the only point in running the software that human intervention is currently required.

3.4 Cross correlate on the QRS peaks

Since alternans detection involves aligning the beats on a fiducial point and performing a Fourier transform across the beats on a point by point basis, a very accurate alignment must be made to prevent beat to beat jitter from interfering with the analysis. Simple alignment on the maximum point of the ECG may not be the most exact alignment, since the data is sampled at only 300-500Hz, allowing for jitter of up to 3ms.

The cross correlation is performed as follows. First, the QRS peaks are roughly detected (and edited by the human operator) as described in the previous step. Once the beats are detected, the start and end of the QRS waveform is detected from an initial signal average. This region, the QRS region of the average beat, is used for cross-correlation. This initial average beat is taken by aligning all beats on the roughly

detected QRS beat. It is possible that some baseline wandering in the ECG over time may result in different beats having different DC offset values; however, this should not affect the morphology of the average beat. It should only affect the average beat's DC offset. Note that we assume that baseline wandering is not significant within a beat. If this is not the case, then some preprocessing on the ECG, such as detrending, must be performed to lower the baseline fluctuations to an acceptable level.

Once the average beat has been constructed, it can be used as a template for cross correlating the other beats. In this procedure, the average beat is time reversed and convolved with each beat separately and the point where the convolution reaches maximum after normalizing the energy in both signals is considered to be the true fiducial point. This is actually an implementation of a matched filter as we are using the average beat as a type of detector of similar waveforms, the individual QRS peaks. The output of this program is the exact time of each beat in an ASCII file. To make the fiducial point as exact as possible, an additional (sinc) interpolation is done to to determine the fiducial point to the nearest 0.2 sample points. This algorithm had been previously developed for use with T-wave alternans measurement and it was not necessary to modify it.

3.4.1 Detect Start and End of QRS

One crucial component of the cross correlation algorithm is determining which portion of the ECG will be used for cross correlation. Since the goal at this stage is to cross correlate on the QRS waveform, including other parts of the ECG in the correlation will only increase noise and result in a less accurate alignment. Thus we need to select the QRS in the average beat to do the correlation. QRS selection of each individual beat is theoretically not necessary since the average beat should act as a matched filter for the QRS. However, limiting the region to cross correlate over the ECG will decrease computation time and possibly eliminate some spurious non-QRS

spikes that may match the filter. Thus, the cross correlation is be restricted to the region near the QRS spike (100 ms on either direction). Selection of the QRS is not an exact procedure, since it is often not clear exactly when the QRS begins and ends; however, even an approximation which grabs a significant amount of the QRS would be acceptable.

The algorithm is outlined as follows:

The QRS is detected and then a window is constructed 100 points to the left of the QRS (which is presumably before the QRS begins, as the human QRS varies in length from 40-100 ms and the peak of the QRS spike is approximately in the middle of this). The average and standard deviation of the ten points in this window are calculated. Then five consecutive points which are located 10 points to the right of the window are then compared to this mean and standard deviation. If the average of the five points is greater than the mean plus three standard deviations of the previous ten points, then we assume that the signal has undergone a significant increase in a short period of time and thus the QRS portion has begun. This method was derived empirically and was tested on several samples. Although its results may not correspond to a clinician's idea of the start and end of the QRS, the algorithm is very robust and useful for correlation purposes.

The end of the QRS was chosen by assuming that the peak is at the midpoint, so its calculation is straightforward.

3.5 Detect bad beats and best segment

Alternans should only be measured on normal, sinus rhythm beats or on normal paced beats in the case of paced data. As described in the background, I am attempting to measure alternations in normal atrial beats. Thus, beats in which conduction does not originate in the atria, such as premature ventricular contractions and other

ectopic beats do not contribute valid data to the sample. However, simply omitting a bad beat is not feasible, since this will have a detrimental effect on the alternans calculation by introducing a phase shift [16].

For instance, consider the sequence ABABABXBAB... where X is a bad beat. If X is simply removed, then the sequence becomes ABABABBAB.. and if the Fourier transform is taken across beats, every beat after the bad one will be completely out of phase resulting in phase cancellation and a spreading of power originally in the 0.5 Hz band to other frequencies.

One could also simply leave the bad beats in place, which usually will not cause as much damage as phase shift cancellation. However, this will result in a significant outlier point and when an FFT is taken, this point will result in some spectral leakage [16].

The best solution is to replace the bad beats with a beat that will not significantly damage the spectra. One way to achieve this is to replace the bad beat with the average beat previously constructed; this is the method used for this program.

The program first decides which beats are bad, and outputs a complete list of beats, showing which are good and which are bad. The criteria for bad beats is either of the following:

1. Any beat which results in an interval greater than 250 ms different from the average. Such a beat is almost certainly ectopic, or is the result of a compensatory pause after an ectopic beat.

2. Any beat in which the correlation coefficient is lower than 0.95 when the beats are aligned such that the correlation is maximum. This is to reject waveforms which are highly dissimilar from the average, which would contribute to outliers in the alternans calculation.

Once the bad beats were identified, a continuous section of 128 beats was selected

which had the fewest bad beats. This segment is found by a simple linear pass through the correlation coefficients and RR intervals. The bad beats in this section are then patched with average beats in the place of bad beats. This beat section is then used for alternans analysis.

3.6 P-wave alignment

In the usual QRS or T wave alternans analysis, a single alignment on the QRS is sufficient. However, for P-wave alternans, alignment on the QRS may not be sufficient due to the fact that there is a variability from beat to beat in P-R interval. Thus, we need to align beats on P-wave to do any meaningful alternans calculation. This P-wave analysis consists of three steps: (1) construct an average beat aligned on the original (or refined) QRS peaks. (2) detect the beginning and end of the P-wave on this average beat, and (3) run the cross correlation on the P-waves rather than on the QRS to align beats by the P-wave.

The construction of an average beat has been previously described. This average beat may be constructed by aligning beats on the raw QRS or on the correlation-refined QRS. In this algorithm, the refined QRS was used. The reason for this choice is described later. Once the average beat has been constructed, the beginning and end of the P-wave must be determined. The reason for automated P-wave detection is the same as that for automated QRS detection – so that cross correlation can be performed. It is important to note that the P-wave detection is carried out on the average beat *aligned on the QRS*. It was observed that PR interval variation was small enough to allow for a reasonable detection of the P-wave for correlation purposes.

The following method was used:

1. The location of the QRS in the average beat was determined as the maximum point in the QRS complex.

2. The baseline voltage of the beat was found by (a) finding the minimum point in the 150 msec. preceding the QRS, (b) taking a window of 15 points and sliding it across 60 msec of the beat centered at this minimum point, (c) when the 15 points in the window have minimum standard deviation, the center point is chosen as the baseline point and the base amplitude is the mean of the 15 point window centered at that point.

3. A search was conducted going to the left of the baseline to find a local maximum over a 100 msec. area. This maximum should be the P-wave peak.

4. The minimum value in the 100 msec window to the left of the P-wave was found (this is referred to the left baseline in further steps).

5. The beginning of the P-wave is determined to be the point to the left of the P-peak where the signal falls 85 percent of the difference between the peak and the left baseline.

6. The end of the P-wave is determined to be the point to the right of the P-wave where the signal falls 85 percent of the difference between the peak and the right baseline.

This method was determined mainly through trial and error. At first an attempt was made to use the standard deviation of the baseline noise to determine when a waveform is present (e.g. when the waveform abruptly rises above three times the noise standard deviation). However, there was the problem that different ECG recordings had different noise levels and that determination of coefficients to satisfy all cases was not possible. If one is guaranteed that the noise level in the ECG will be in a certain range, this simpler algorithm may be used.

3.7 Baseline Measurement

This section gives an explanation for the first section of the P-wave alignment: detection of a baseline. For the cross correlation and the alternans techniques to work properly, the baseline of each beat must be known. If the baseline is not correct, then the correlation will still most likely determine the proper location of the peak, but it will report an incorrect correlation coefficient. This could be rectified by zero-meaning the data; however, baseline calculation is necessary again in the calculation of alternans, and thus it is useful to use the same algorithm in both places.

The ideal baseline is the isolectric segment closest to the waveform of interest (either the QRS or the P, in our case). The isolectric segment satisfying this criteria is the PQ segment, shown in the figure below. The PQ segment easy to identify, as it is between two prominent waves, the P and the QRS. The baseline was determined as the initial step for determining p-wave location, described above.

3.8 Second cross correlation

Once the P-wave has been detected on the average beat, it is cross correlated with the ECG analogous to the method used for QRS cross correlation to properly align the P-waves. During this cross correlation, the threshold for bad beats is lowered to allow correlation coefficients as low as 0.8 since the signal to noise ratio in a typical P-wave is lower than that for a QRS complex.

There are two possibilities for bad beat detection and replacement: one is to detect bad beats during both the QRS correlation and during the P correlation while the other is to only detect and replace bad beats during the P correlation step. In practice, the difference is minor, since a beat with an ectopic or otherwise abnormal QRS usually has an abnormal P wave also because conduction does not proceed normally through the atria. In this program, bad beats were identified at both stages.

3.9 Alternans calculation

The alternans calculation is illustrated in the figures 2-6 and 3-2. A series of beats are aligned and a two dimensional matrix is constructed as shown in Figure 2-6. 128 consecutive beats were used for the analysis for the following reasons: (i) The FFT which is taken across the beats must have an input which is a power of two in length. (ii) The segment must be long enough to allow for a decent spectral estimate yet be short enough to allow for a segment to be chosen which does not include many bad beats or any phase resetting. 128 beats has been found to be optimal in many cases for T wave alternans.

Once the beats have been chosen, the analysis is straightforward. FFTs with rectangular windows are taken across the beats as shown. The purpose is to calculate the alternans measure at each point in time, relative to each fiducial point. Two measures of alternans can be calculated: (i) the alternans metric, which is the voltage present in the 0.5 Hz peak, and (ii) the k-score, which is a measure of the ratio of the alternans peak to the measure of the average noise surrounding the peak in the FFT [16, 17, 20]. Thus, a k-score (also called the Alternans Ratio) of 1 shows a lack of alternans, while a k-score of 4 or more shows a high probability of alternans. This is illustrated in Figure 3-2.

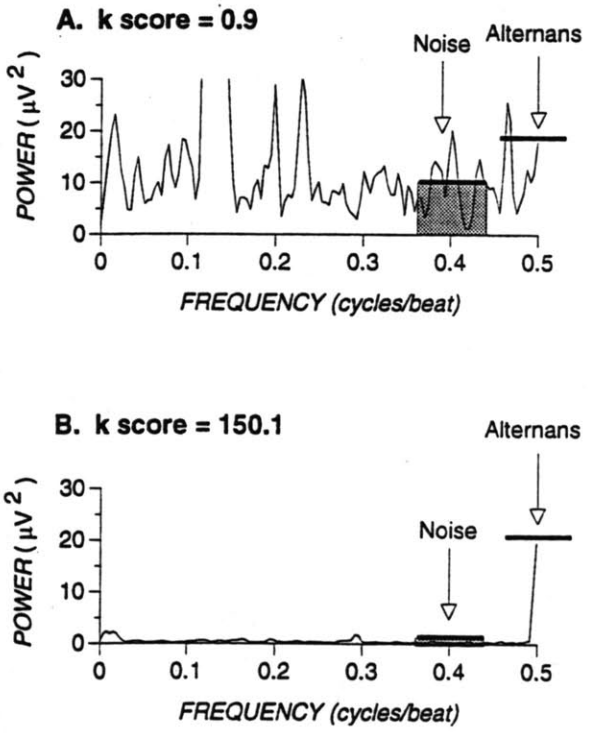
$$\text{Alternans voltage} = (\alpha - \eta)^{\frac{1}{2}} \tag{3.2}$$

$$\text{Alternans ratio} = \text{K-Score} = \frac{(\text{Alternans voltage})^2}{\sigma} \tag{3.3}$$

Where σ = standard deviation of the magnitudes of the power spectra in the noise window, η = mean of magnitudes of power spectra in the noise window, and α = magnitude of the power spectrum at 0.5 cycles/beat.

The program calculates the k-score and alternans metric at each point in the beat, and then outputs graphs showing these values. These graphs (presented in the results

Figure 3-2 Shown above are two plots of the power spectrum at a particular point in the beat. Although both spectra have the same alternans voltage, they have very different K-scores. Plot B shows a significant level of alternans while A does not. The K-score is a measure of the alternans voltage relative to the noise level (From [16]).



section) show the specific locations of alternans (beginning of P-wave versus end of P-wave, etc.). One should note that the values given outside of the P-wave are most likely invalid, or at least inaccurate, because the beats were aligned on the P-wave, and therefore variability in the P-R interval would result in jitter in other areas of the ECG. For example, beat to beat alternation in the PQ interval may manifest itself as (spurious) T-wave alternans.

A further parameterization of alternans may be useful for clinical use. The sum-new.c program sums the alternans measure and k-score over various time intervals in the beat. For the present study, four intervals were chosen: The entire P-wave as determined by the algorithm described above and the first, second, and last third of the P-wave.

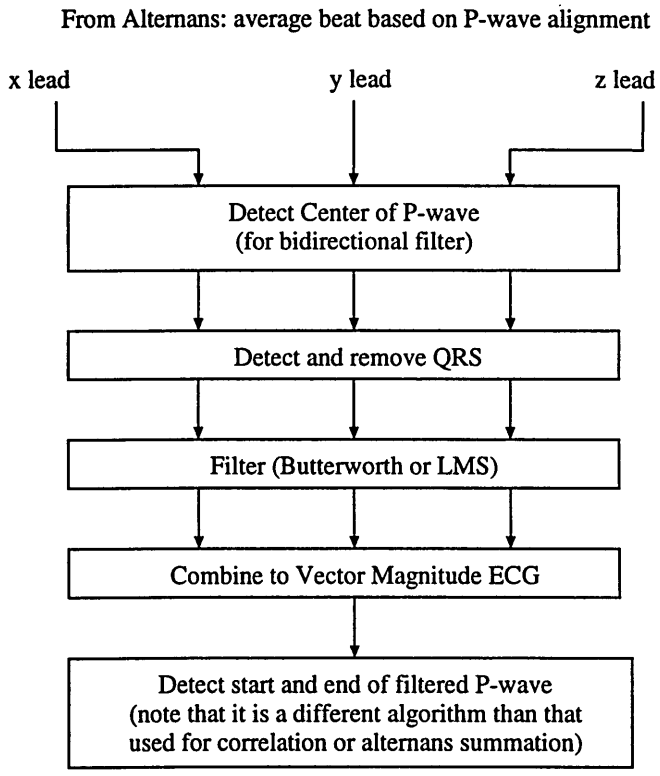
3.10 Signal Averaged ECG calculation

This measure is not directly connected with alternans, as it is a separate diagnostic procedure as described in the background section. However, calculation of the SAECG has several steps in common with alternans calculation. The steps in SAECG calculation are shown in Figure 3-3. Essentially, all steps are identical to alternans up to and including calculating the average beat aligned on the P-wave. Once the average beat has been calculated, it must be filtered and then the length of the filtered P-wave can be determined.

3.11 Filtering the SAECG

The individual X, Y, and Z leads were passed through a high pass filter to magnify the high frequency components of the P-wave and to get rid of the low frequency components in the baseline to make P-wave detection somewhat easier and more reliable. The filters were chosen based on the results of previous investigators, shown

Figure 3-3 Overview of steps in calculating P-wave alternans



Filter Type	Sensitivity (%)	Specificity (%)	Predictive Accuracy (%)
Bidirectional Butterworth	78.26	86.11	83.05
Unidirectional Butterworth	78.26	83.33	81.36
Least Mean Square	82.61	86.11	84.75

Table 3.1: Comparison of various filters for the P-wave SAECG (from [8])

in Table 3.1. A P-wave SAECG was taken on a control group and on a group with PAF. The sensitivity, specificity, and predictive accuracy for distinguishing the groups based on the filtered P-wave length for various filters is shown.

Two filters were chosen for implementation in this thesis: a four pole bidirectional butterworth filter and a least mean squares (or Savitzky-Golay) filter.

3.11.1 Butterworth filter

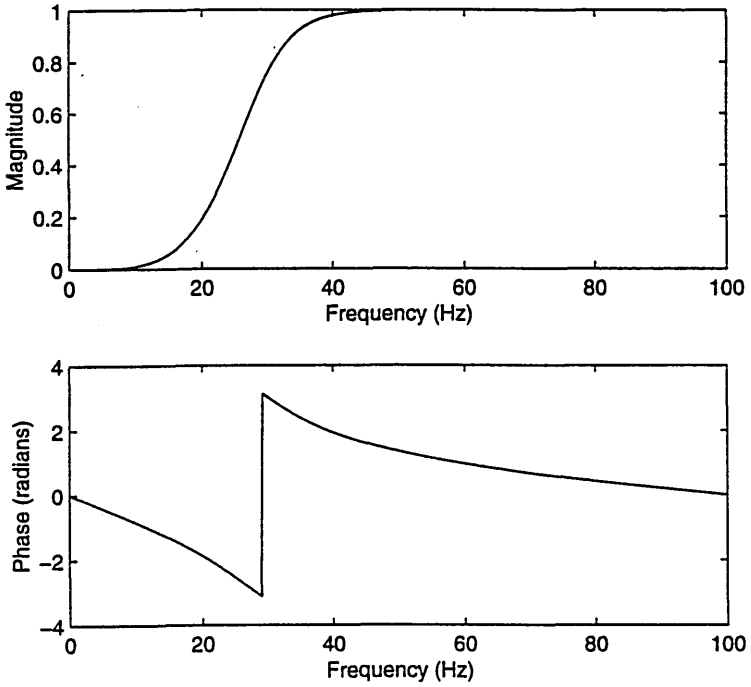
A butterworth filter has a frequency response which is "maximally flat" near $w = 0$. An example of this filter's magnitude and phase response is shown in Figure 3-4. The magnitude frequency response of a continuous time butterworth filter is given as follows [14],

$$|H_c(jw)|^2 = \frac{1}{1 + (w/w_c)^{2N}} \tag{3.4}$$

where w_c is the cutoff frequency and N is the order of the filter. Although a four pole butterworth filter has a fairly flat impulse response, ringing will still occur if there is a significant discontinuity in the signal. Since the purpose of the filtering is to highlight high frequency late potentials in the P-wave and not to extend the P-wave because of ringing, a bidirectional butterworth filter is used.

The butterworth filter is a causal filter, meaning that ringing will only occur in the direction of filtering, which is usually forward in time, as shown in part a of the figure. In a bidirectional filter, a fiducial point is chosen, and filtering to the left of that point is done forward in time while filtering to the right of the point is done

Figure 3-4 Frequency response of a discrete-time butterworth filter. This filter is a four pole high pass with a cutoff of 0.29π radians. So, if data is sampled at 200Hz, the cutoff is 29Hz.



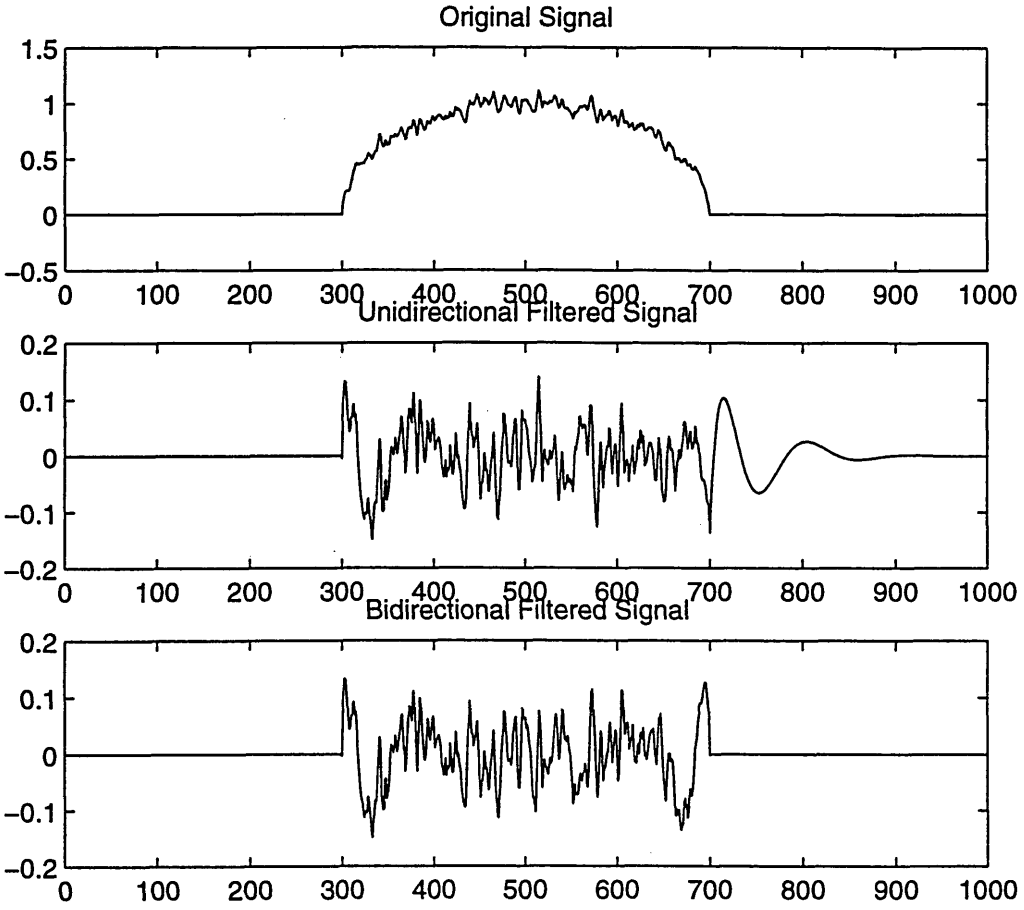
backward in time. If this fiducial point is chosen to be near the center of the P-wave, ringing from the filter will not artificially extend its length.

Based on the suggestions of previous researchers [8], a bandpass filter with pass-band 29Hz-250Hz was used.

3.11.2 Least Squares filter

A least squares filter, also called a Savitzky-Golay filter [21], is a type of finite impulse response (FIR) linear filter. Unlike the butterworth filter, this filter is only applicable to a discrete time signal as it relies on fitting a curve to discrete points. The least squares filter is defined by two parameters: window width and filter order. To filter a given point of a signal, the window is centered on the point and a polynomial curve

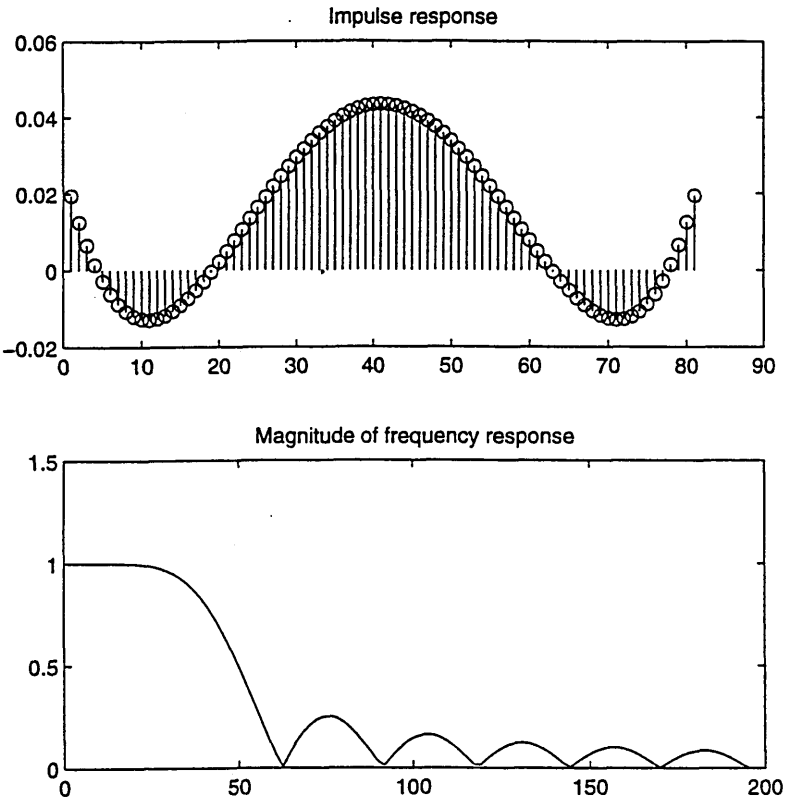
Figure 3-5 The difference between a unidirectional and a bidirectional filter. The original signal is a simulated P-wave, consisting of an ellipse with gaussian white noise added. Note that the unidirectional filter elongates the signal due to ringing.



is least squares fit to the points in the window to the specified order of the filter. The value of the polynomial curve at the center point of the window is the filtered value of the signal.

Although not immediately obvious, the least squares filter is a linear time invariant filter. Given a window width and polynomial order, an FIR filter can be constructed which performs the same filtering operation [21]. To achieve a bandpass filter with a passband of 29-250Hz, a window width of 100 ms (36 points at 360Hz) and a polynomial order of 4 was chosen. The impulse response and frequency response for

Figure 3-6 Least Squares Filter. The least squares filter is always a low pass filter. To perform a high pass filtering operation, a least squares filter is applied to the signal and then the low pass result is subtracted off the original signal.



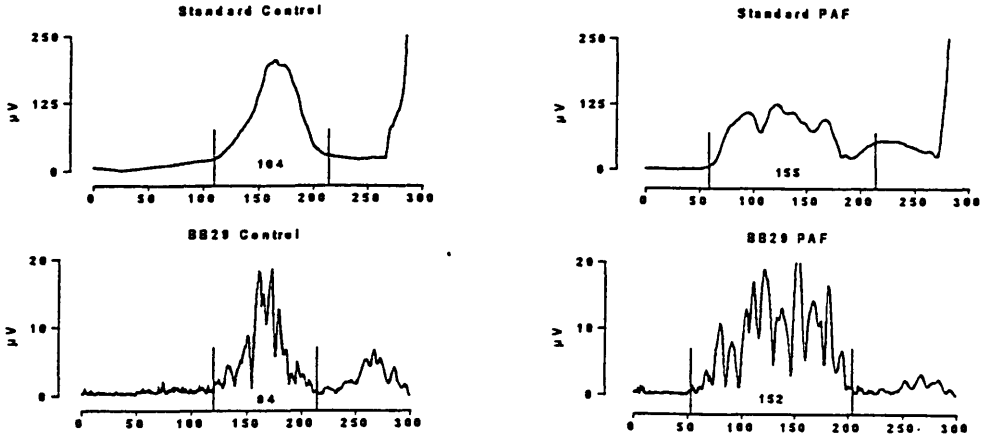
this filter is shown in Figure 3-6.

3.11.3 Detecting the P-wave

P-wave detection for the purposes of measuring alternans has been described above. A different method is used for the measurement of the length of the P-wave in the filtered SAECG. This measurement must be more exact, which is possible because the high pass filtering simplifies detection. Figure 3-7 shows the difference between a typical ECG and a high pass filtered ECG. Detection is performed as follows:

1. Calculate the average noise as the average of 10 points 50 msec. from the start of the P-wave.

Figure 3-7 The Effect of Filtering the P-Wave. The two P-waves in the first row are unfiltered, while the two in the second row are filtered. After filtering, high frequency components are highlighted and the transition from baseline to P-wave is sharpened



2. Keep checking to the right until the signal goes above 3 times the average noise for 3 points in a row. The first of the three points is the start of the p-wave.

3. Keep going to the right until a 30 msec window where no point goes above 3 times the average is found. The beginning of this window is the end of the p-wave.

4. Check to make sure that no point has been reached which is greater than 30% QRS amplitude. In this case, step 3 has failed, so redo step 3, except this time only look in a 7 msec window (this second check may catch cases where the P-wave end is very close to the QRS complex).

5. Next, refine the beginning and ending points as follows: Go left of the starting point of the p-wave until 2 successive points that are less than the average noise value are reached. Similarly, Go th the right of the ending point of the p-wave until 2 successive points that are less than the average noise value are reached.

6. If step 5 takes you more than 40 points away from the initial p-wave estimate, then revert back to the initial estimates.

The length of the p-wave is then simply calculated as the start time of the p-wave minus the end time of the p-wave.

3.12 Software Description

The following briefly summarizes the functions of the various programs and gives the procedure for calculating p-wave alternans and the p-wave SAECG.

1. Prepare the data: The data must be three channel data in woven format, where each data point is a short, two-byte integer. Thus, the first point of the first signal is stored in the first two bytes, the first point of the second signal in the next two bytes, and so on.

2. Run the script RR.ii. It prompts the user for several parameters to be used for detecting QRS complexes. Default values are provided, and it is nearly always sufficient to use these default values.

3. Run the script display. This will run an interactive signal display program, which will allow the user to look at the QRS annotations and make any changes. In general, the QRS peak detection algorithm will make a mistake 0.2% to 5% of the time, depending on the quality of the ECG recording and on the number of ectopic beats present.

4. Run the script pwa_auto. This is a mostly automated script, which first performs an autocorrelation on the QRS peaks to more closely align those peaks. To do this, it detects the QRS start and end automatically, but also allows the user to manually select it in case there is a problem. After the QRS correlation,

an average beat is constructed and the P-wave is detected. Again, the user can change the computer's choice. Next, a cross correlation is performed on the P-waves. Bad P-waves must then be found – this is again automated by rejecting P-waves which correlate poorly with other P-waves, or which come very early or late. The user may override the computer's parameter choices if necessary. Finally, the P-wave alternans is calculated in each lead and in the vector magnitude using the program spitnew3_s.c, as described above.

5. The final portion of the analysis is currently done through matlab although it would be fairly straightforward to compile the code if necessary for performance purposes. The matlab script loads the results of spitnew3_s.c and plots them, and provides an index of the overall alternans over the P-wave. It then filters the individual leads of the average beat, and creates a filtered vector magnitude to determine P-wave SAECG parameters. The start and end of the p-wave are determined as described in the previous section.

Source code and shell scripts are provided in Appendix A.

Chapter 4

Results

4.1 Programming

The programming consisted of updating previously existing software and of creating new programs. The entire software package consists of 22 shell scripts and compiled C programs. The previous set of programs was created to measure T-wave alternans, as described earlier. I modified, to a small extent, about 15 programs, and I created or rewrote 6 programs, listed in the appendix.

All programs have been tested on five datasets. In a future study, large amounts of data will be processed to determine the clinical significance of P-wave alternans and its correlation with atrial fibrillation. This thesis focuses on the technical aspects of the software rather than its clinical application.

4.2 P-wave alternans results

P-wave alternans analysis was run on several test datasets from normal subjects (those without atrial arrhythmias) and on some high resolution ECGs from goats. These were used for algorithm development and testing purposes.

4.2.1 Simulated Results

In addition, a "simulated" ECG signal was prepared as follows: (i) A single beat was extracted from the ECG of a human subject by taking the data from 300ms before the QRS to 400ms after the QRS. The P and T waves fell within this region. (ii) A copy of the beat was created and was manually edited in MATLAB by multiplying the P-wave by a cosine wave with a period equal to the P-wave (time zero at start of P-wave). The rest of the beat was unchanged. The original beat was called type A, the modified beat, type B. (iii) 25ms of data was cut out of the PQ intervals of both beats, and then 64 A beats and 64 B beats were constructed by adding a random interval of flatline data to the PQ intervals (0-50ms uniformly distributed). (iv) The beats were concatenated in ABABAB... order to form a simulated ECG with a random flatline interval added between beats (200-300ms uniformly distributed). (v) Finally, gaussian white noise with variance of 2% the height of the QRS was added to the entire ECG. Thus the simulated ECG had P-wave alternans with varying interbeat intervals, varying PR intervals, and noise.

If P-wave alternans were perfectly detected then there would be alternans in the P-wave in a cosine varying pattern: minimum alternans would be present at the start and end of the P-wave and maximum alternans would be present at the trough of the cosine, in the middle of the P-wave. When the dataset was run through the alternans detection programs, initially the cross correlation on the P-waves failed: all were considered "bad" p-waves since the correlation coefficients of all P-waves with the average beat were lower than 0.85. This is a result of the artificially high level of alternans in the P-waves; if the correlation cutoff is lowered to 0.75 then all beats are detected as good beats. After this modification, alternans is detected in the P-waves. Although alternans is not found in precisely the cosine pattern it was introduced, the alternans metric and k-score were greatest in the center of the P-wave and least at the edges. Alternans was detected in a smeared cosine pattern, possibly because

during the cross correlation, the A and B p-waves were lined up in a slightly different manner than the original alignment used to generate the waveforms. This is not a problem with the alternans detection software, but a problem inherent in trying to detect alternans in a waveform which is also being used for beat alignment, when the alternans are large enough to affect cross correlation. A further investigation may attempt to more clearly define alternans in cases such as this.

4.2.2 Clinical Testing

The primary aim of this thesis was to produce functioning software which can be later used for clinical testing. Figure 4-1 shows the results of running the data on a typical subject. This subject does not have significant P wave alternans.

Overall, five human data sets were tested:

- A patient with Wolff-Parkinson-White Syndrome, paced and unpaced.
- A patient with EP induced atrial flutter, paced and unpaced.
- A patient with EP induced atrial flutter, unpaced.

The alternans metric and k-score were calculated for the five data sets on the x, y, z, and combined vector magnitude leads. These measures were summed over the first, second, and last third of the P-waves. None of the patients showed significant P-wave alternans (none had a k-score above 4 for a significant portion of the P-wave).

The re-triggering on the P-wave proved to be important in many of the patients. Figure 4-2 shows a the same set of 128 beats triggered on the QRS and on the P waves. One can see that the P-wave is more clearly visible in the beats aligned on the P wave and the QRS is slightly blurred.

Patient	Butterworth	Least Squares
1 WPW unpaced	131.2	133.1
1 WPW paced	121.3	123.1
2 EP induced AF unpaced	147.0	148.0
2 EP induced AF paced	144.0	141.7
3 EP induced AF unpaced	154.9	158.4

Table 4.1: Lengths (in ms) of filtered SAECG P-waves (from [8].

4.2.3 P-wave SAECG

P-wave SAECGs were taken for the five datasets listed above and for 7 normal datasets. The values for the duration of the P-wave SAECG (vector magnitude) are given in Table 4.1. The results are in the range of previously published values.

Figure 4-1 Results of running the P-wave analysis software on a human subject in sinus rhythm. Atrial flutter was induced in this patient during a previous EP test. In this figure, the vector magnitude was used for alternans calculation and 12 out of 128 badbeats were replaced with the average beat. The K-Score is not consistently above 4 in the region of the P-wave so alternans is not present.

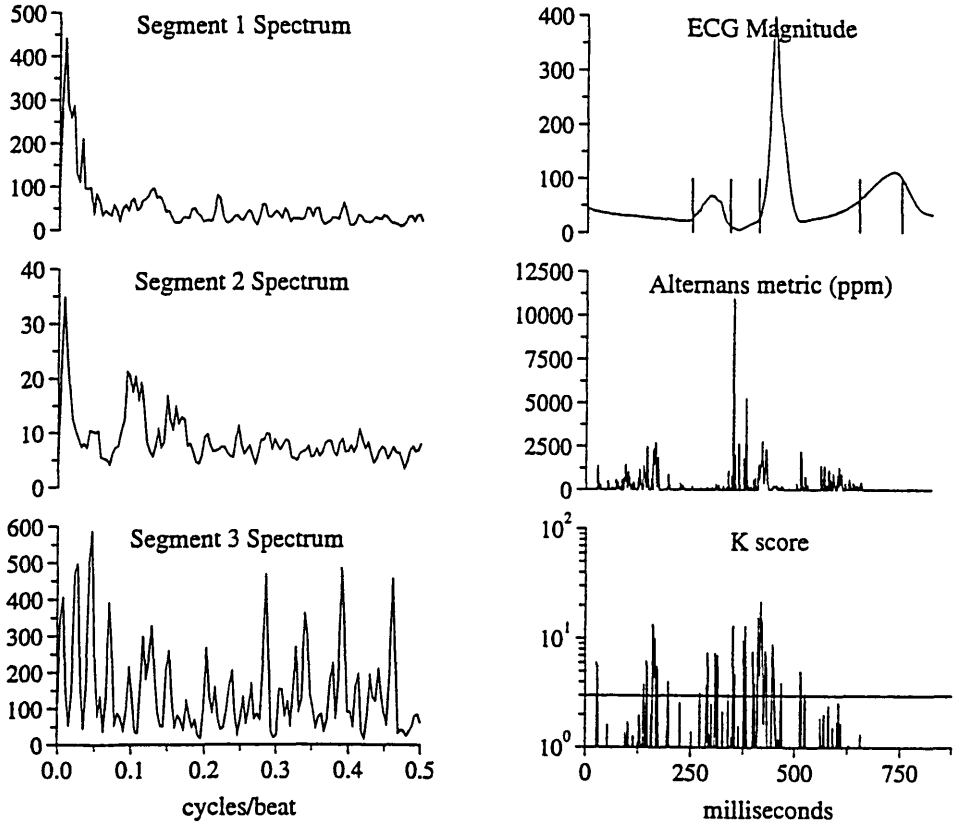
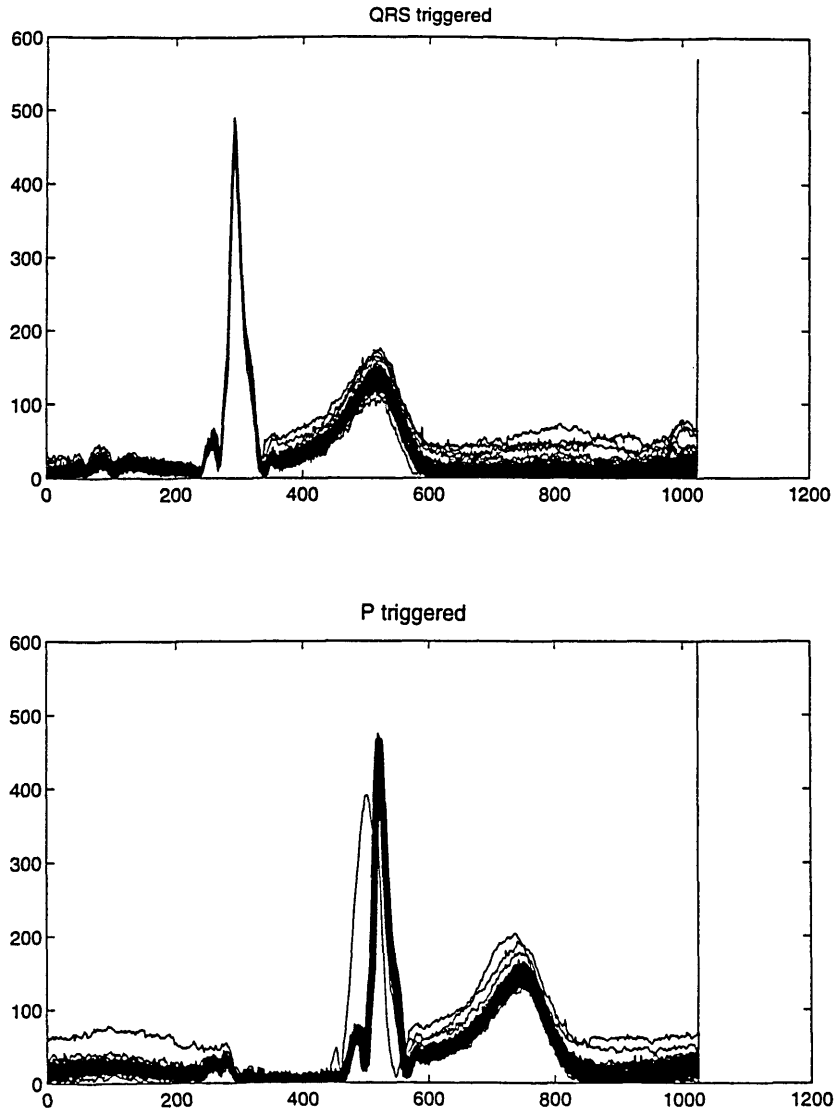


Figure 4-2 P-triggering vs. QRS-triggering.



Chapter 5

Conclusions

In this thesis, software was developed to measure P-wave alternans from an input of three digitized orthogonal ECG leads.

Previously existing code for measuring T-wave alternans was adapted to perform triggering on the P-wave and to perform a more automated and consistent alternans calculations by the use of various waveform detection algorithms, a more accurate averaging procedure, and "intelligent" algorithms to allow cross correlation, bad beat selection, and other functions to be done without user input. The user interface was constructed, however, to allow the operator to override automatically detected settings and parameters, if necessary. In addition, software to perform a P-wave signal averaged ECG and P-wave length detection with two types of commonly used filters was developed.

The alternans and P-wave software were tested on simulated and real data. A simulated ECG with a known level of P-wave alternans was run through the software. This ECG contained added signal noise with a known variance, random interbeat intervals, and random PQ intervals (with given distributions). The alternans power and K-score results were approximately equal to theoretically predicted values based on the alternans and noise present in the simulated signal (see previous chapter for

details). As long as the PR and interbeat interval variability stayed within physiologic ranges, they had no effect on alternans levels.

The P-wave SAECG software was run with data from normal patients and on data from patients with atrial arrhythmias and the results were similar to those from previous investigators (Table 4.1).

The P-wave alternans software was tested on a limited set of patients with atrial arrhythmias and controls, but no statistical alternans were observed in any of the patients. A larger data set with a variety of atrial pacing rates and a variety of atrial arrhythmias (PAF, WPW, EP induced AF) should be used to further investigate the clinical significance of P-wave alternans in assessing atrial instability. The software is suited performing this sort of batch mode automated analyses.

Appendix A

Source code

In this chapter I present source code for my program. Note that this appendix only contains code for the programs which I have created or to which I have added a major section.

A.1 delqrs.c

```
/*  delqrs.c  -- Detects the beginning and end of QRS
    Nikhil Iyengar,  10/22/97

    Usage:  delqrs

    Takes column input file from standard input.  This column file
    should contain the vector magnitude of an average beat with the
    annotation at precisely the center of the beat (from sigavg.c, for
    example).

    Outputs three numbers on one line:  a   b   c
    a = baseline.  points to the right of b where baseline is located.
                   should always be negative
    b = qrs center.  points to the right of the annotation that qrs
                   center is located.
    c = QRS width.  width of the QRS around its center.


    Procedure:  1.  Baseline estimate
```

```
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

double mean(double [], int, int);
double std(double [], int, int);


main()
{

  double beat[3000], qrsampl, baseampl, rightmin;
  int i, width, qrsindex, baseindex, newindex, rightindex, center;
  double curstd, minstd, thresh;
  int qrsstart, qrsend, out1, out2, out3;


  i=1;
  while (scanf("%lf", &beat[i]) == 1) {
    i++;
  }
  width = i-1;
  center = width/2;

  /* Find the maximum point in beat */

  qrsampl = 0;
  for(i=1;i<=width;i++) {
    if (beat[i] > qrsampl) {
      qrsampl = beat[i];
      qrsindex = i;
    }
  }

  /* Find the baseline point */

  baseampl = qrsampl;
  for(i = qrsindex-150 ; i<qrsindex ; i++) {
    if (beat[i] < baseampl) {
      baseampl = beat[i];
      baseindex = i;
    }
  }

  minstd = qrsampl;
```

```
  for(i = baseindex - 30 ; i<baseindex + 30; i++) {
    curstd = std(beat,i-7,i+7);
    if (curstd < minstd) {
      minstd = curstd;
      newindex = i;
    }
  }
  baseindex = newindex;
  baseampl = mean(beat,baseindex-7,baseindex+7);

  /* We found the baseline to the left of the QRS.
     Now, find the minimum to its immediate right  */

  rightmin = qrsampl;
  for(i = qrsindex; i < qrsindex+100 ; i++) {
    if (beat[i] < rightmin) {
      rightmin = beat[i];
      rightindex = i;
    }
  }

  /* The beginning and end of QRS will be when the amplitude first
     falls below 5% of the difference between QRS and baseline */

  thresh = baseampl + 0.05*(qrsampl - baseampl);
  for(i = qrsindex; i >= qrsindex - 100; i--) {
    if (beat[i] < thresh) {
      qrsstart = i;
      break;
    }
  }

  thresh = rightmin + 0.05*(qrsampl - rightmin);
  for(i = qrsindex; i <= qrsindex + 100; i++) {
    if (beat[i] < thresh) {
      qrsend = i;
      break;
    }
  }

  out1 = baseindex - (qrsstart+qrsend)/2;
  out2 = (qrsstart+qrsend)/2 - center;
  out3 = qrsend - qrsstart;

  printf("%d \t %d \t %d\n", out1, out2, out3);

}
```

```
double mean(double vec[], int start, int end)
{
  int i;
  double sum=0, avg;

  for(i=start ; i<=end ; i++) {
    sum = sum + vec[i];
  }

  return sum/(end-start+1);

}

double std(double vec[], int start, int end)
{
  int i;
  double sum=0, avg;

  for(i=start ; i<=end ; i++) {
    sum = sum + vec[i];
  }
  avg = sum/(end-start+1);

  sum = 0;
  for(i=start ; i<=end ; i++) {
    sum = sum + (vec[i] - avg)*(vec[i] - avg);
  }

  return  sqrt(sum/(end-start));
}
```

A.2 spitnew3_s.c

```
/*      spitnew3_s.c       Nikhil Iyengar       July 1997

Function        Sample-point analysis and display for spectral
                decomposition of variance in waveform morphology.
                The entire waveform is analyzed, broken into
                three abutting segments, the first is QRS wide,
                the second is ST wide, and the third is T wide.
                Baseline drift is subtracted, after it is estimated
                100 milliseconds before the fiducial point.
                DOES INTERPOLATION
                A k-score is output for every point in the
```

```
                waveform.

*/

#include        <stdio.h>
#include        <math.h>
#include        <sgtty.h>
/* #include     <libmp.h> */
#include        "utils.c"

#define         MAXCHAN 3
#define         TWOPI   6.2831854
#define         MAXLEN  1700
#define         MAXOBS  258
#define         BASE    8
#define         OFFSET  300

FILE    *pksfp, *outfp, *klfp, *matrixfp, *spectfp;
FILE    *matrixfp_x, *matrixfp_y, *matrixfp_z;
short   buf[MAXLEN*16];
int     nchan=0, data_chan=3, close_pt, joe_switch=0, smooth_switch=0;
int     beat_flag=0;
int     chan[MAXCHAN]={0,1,2};
int     datafd, n_obs=256, result_fd, beat_fd, sum_window = 1;
int     result_fd_x, result_fd_y, result_fd_z;
int     beat_fd_x, beat_fd_y, beat_fd_z;
int     beat_no, begin=50; /* begin QRS 50 msecs before mark */
int     pk_offset=0, bad_counter=0, bad_beats[128], beat_length=300;
char    option(), *opt_argv(), ctmp, bmatrix[20],
        fmatrix[20], tmode = 0;
char    *mname, mname_x[20], mname_y[20], mname_z[20];
char    *Bname, *Bname_x, *Bname_y, *Bname_z;
char    *beatfile, beatfile_x[25], beatfile_y[25], beatfile_z[25];
char    cr;
char    pksname[20], dataname[20];
long    pks[MAXOBS]; /* peak locations  */
double  g[MAXCHAN] = {1.0,1.0,1.0};
float   temp_offset[MAXOBS];
double  kscore[MAXLEN], alt_metric[MAXLEN];
double  r_array[MAXOBS], rmean, rstd, rsum, rsum2, total=0.0;
float   big_array[MAXOBS][MAXLEN];
float   big_array_x[MAXOBS][MAXLEN];
float   big_array_y[MAXOBS][MAXLEN];
float   big_array_z[MAXOBS][MAXLEN];
double  inv_n_obs;
double  datasum[MAXLEN];
double  datasum_x[MAXLEN];
```

```
/*  matrix with average beat of lead X  */
double  datasum_y[MAXLEN];
/*  matrix with average beat of lead Y  */
double  datasum_z[MAXLEN];
/*  matrix with average beat of lead Z  */
double  temparray[MAXOBS];
double  tmp_array[MAXOBS];
double  mean_est, var_est;
double  energy[MAXLEN];
int     best_index = 0, base_offset=0;
int     Samprt = 1000;
int     i_switch = 0;

main(argc, argv)
int argc;
char **argv;
{
int     i=0, j=0;
double  data[MAXLEN];


for(i=0;i<MAXLEN;i++) {
  data[i]=0;
}

i=0;

        while( (ctmp = option(&argc, argv)) != 0){
                switch(ctmp){
                        case 'b': /* bad beats for exclusion */
                                bad_beats[bad_counter++] = atoi(opt_argv());
                                break;

                        case 'B': /* name of big output file */
                                Bname = opt_argv();
                                break;

                        case 'c': /* channels to be analyzed */
                                chan[nchan++]= atoi(opt_argv());
                                break;

                        case 'd': /* data file  */
                                datafd = ofile( opt_argv() );
                                break;

                        case 'g': /* relative gains of channels */
                                g[i++] = atof(opt_argv());
```

```
                break;

            case 'i': /* Switch for individual lead analysis */
                i_switch = 1;
                break;

            case 'j': /* just for joe - will throw out worst
beats */
                joe_switch = 1;
                break;

            case 'l': /* base offset */
                base_offset = atoi(opt_argv());
                break;

            case 'm': /* output filename for matrix file */
                mname = opt_argv();
                break;

            case 'N': /* # of chans in data */
                data_chan = atoi(opt_argv());
                break;

            case 'n': /* number of observations */
                n_obs = atoi(opt_argv());
                n_obs = npwr2(n_obs);
                break;

            case 'o': /* name of beat file */
                beatfile = opt_argv();
                beat_fd = cfile( beatfile );
                beat_flag=1;
                break;

            case 'p': /* peaks file */
                pksfp = fofile( opt_argv(), "r" );
                break;

            case 'r':
                pk_offset = atoi(opt_argv());
                break;

            case 'w': /* width of beat (max = 800) */
                beat_length =atoi(opt_argv());
                break;

            case 's': /* sum up last (number) points */
```

```
                        sum_window = atoi(opt_argv());
                        break;

                case 'S': /* sampling rate */
                        Samprt = atoi(opt_argv());
                        break;

                case 'h': /* help */
                        printf("\n\n");
                        printf("splat.c - (N <= 3) channel sample
point analysis \n");
                        printf("\n\n");
                        printf("command line arguments:\n");
                        printf("        -b(beat #)   - iteratively
called for each beat to be excluded\n");
                        printf("        -B(name)     - name of pt.
spectrum output file\n");
                        printf("        -c(chan #)   - iteratively
called for each channel used\n");
                        printf("        -d(filename) - name of raw
data file\n");
                        printf("        -g(gains)    - iteratively
called for each channel used\n");
                        printf("        -i           - analyze each
channel individually\n");
                        printf("        -l(number)   - base offset\n");
                        printf("        -m(name)     - name of matrix
file\n");
                        printf("        -n(number)   - number of
events (pks)\n");
                        printf("        -N(number)   - number of
channel in sampled data\n");
                        printf("        -o(filename) - name of beat
output file\n");
                        printf("        -p(filename) - name of peaks
file\n");
                        printf("        -r(number)   - shifts buffer
to right\n");
                        printf("        -s(number)   - will sum over
last (number) points\n");
                        printf("        -S(number)   - the sampling
rate\n");
                        printf("        -w(width)    - the width of a
beat (sample points)\n");

                        exit(0);
                default:
```

```
                        printf("\n ILLEGAL OPTION %c\n", ctmp);
                        exit(0);
                }
        }

        beat_length = beat_length*Samprt/1000;
        sprintf(fmatrix,"%s",mname);
        matrixfp = fofile(fmatrix, "w");
        result_fd = cfile(Bname);


        if(i_switch)   {

            Bname_x=(char *)malloc(strlen(Bname)+3);
            sprintf(Bname_x,"%s_x",Bname);
            result_fd_x = cfile(Bname_x);
            sprintf(mname_x,"%s_x",mname);
            matrixfp_x = fofile(mname_x, "w");

            Bname_y=(char *)malloc(strlen(Bname)+3);
            sprintf(Bname_y,"%s_y",Bname);
            result_fd_y = cfile(Bname_y);
            sprintf(mname_y,"%s_y",mname);
            matrixfp_y = fofile(mname_y, "w");

            Bname_z=(char *)malloc(strlen(Bname)+3);
            sprintf(Bname_z,"%s_z",Bname);
            result_fd_z = cfile(Bname_z);
            sprintf(mname_z,"%s_z",mname);
            matrixfp_z = fofile(mname_z, "w");

        }

        get_pks();      /* loads array of peak locations */

        inv_n_obs = 1.0/((double)n_obs);
        for(beat_no=0; beat_no<n_obs; beat_no++)
                get_data(data);

/*              SCALE THE AVERAGE BEAT          */

        for(i=0; i<MAXLEN; i++)
                datasum[i] *= inv_n_obs;
        if(i_switch) {
          for(i=0; i< MAXLEN; i++) {
            datasum_x[i] *= inv_n_obs;
            datasum_y[i] *= inv_n_obs;
```

```
            datasum_z[i] *= inv_n_obs;
          }
        }


/*          REPLACE BAD BEATS WITH THE AVERAGE    */

        for(i=0; i<bad_counter; i++){
                printf("Replaced beat %d  peak location %d \n", bad_beats[i],
pks[bad_beats[i]]);
                for(j=0; j<MAXLEN; j++) {
                        big_array[bad_beats[i]][j] = datasum[j];
                        if (i_switch) {
                          big_array_x[bad_beats[i]][j] = datasum_x[j];
                          big_array_y[bad_beats[i]][j] = datasum_y[j];
                          big_array_z[bad_beats[i]][j] = datasum_z[j];
                        }
                      }
        }

          if(beat_flag==1) {
          for (i=0;i<n_obs;i++) {
            write(beat_fd, &big_array[i][0], sizeof(float)*1024);
          }
        if(i_switch) {

            sprintf(beatfile_x,"%s_x",beatfile);
            beat_fd_x = cfile(beatfile_x);
            sprintf(beatfile_y,"%s_y",beatfile);
            beat_fd_y = cfile(beatfile_y);
            sprintf(beatfile_z,"%s_z",beatfile);
            beat_fd_z = cfile(beatfile_z);

            for (i=0;i<n_obs;i++) {
              write(beat_fd_x, &big_array_x[i][0], sizeof(float)*1024);
            }

            for (i=0;i<n_obs;i++) {
              write(beat_fd_y, &big_array_y[i][0], sizeof(float)*1024);
            }

            for (i=0;i<n_obs;i++) {
              write(beat_fd_z, &big_array_z[i][0], sizeof(float)*1024);
            }

          }
        }
```

```
        write(result_fd, &n_obs, sizeof(int));
        write(result_fd, &bad_counter, sizeof(int));
        write(result_fd, &beat_length, sizeof(int));
        write(result_fd, datasum, sizeof(double)*beat_length);

        make_spect(0, beat_length, big_array, result_fd, datasum);

        out_results(matrixfp, datasum);

        /*  If individual lead analysis is desired the program runs the
analysis */
        /*  for each lead that the gain is not zero.
     */
        /*  ------------------------------------------------------------*/

        if(i_switch)   {

             write(result_fd_x, &n_obs, sizeof(int));
             write(result_fd_x, &bad_counter, sizeof(int));
             write(result_fd_x, &beat_length, sizeof(int));
             write(result_fd_x, datasum_x, sizeof(double)*beat_length);

             make_spect(0, beat_length, big_array_x, result_fd_x, datasum_x);

             out_results(matrixfp_x, datasum_x);


             write(result_fd_y, &n_obs, sizeof(int));
             write(result_fd_y, &bad_counter, sizeof(int));
             write(result_fd_y, &beat_length, sizeof(int));
             write(result_fd_y, datasum_y, sizeof(double)*beat_length);

             make_spect(0, beat_length, big_array_y, result_fd_y, datasum_y);

             out_results(matrixfp_y, datasum_y);


             write(result_fd_z, &n_obs, sizeof(int));
             write(result_fd_z, &bad_counter, sizeof(int));
             write(result_fd_z, &beat_length, sizeof(int));
             write(result_fd_z, datasum_z, sizeof(double)*beat_length);

             make_spect(0, beat_length, big_array_z, result_fd_z, datasum_z);

             out_results(matrixfp_z, datasum_z);
```

```
        }

    exit(0);
}

make_spect(start, seg_end, bg_array, result, datasm)
int     start, seg_end;
float   bg_array[MAXOBS][MAXLEN];
int     result;
double  datasm[];
{
    int     i, j;
    int     wintype = 2;
    double  limit = 3.0;
    register double temp_var, mean_energy = 0.0;

    for(j=start; j<seg_end; j++){
        temp_var = datasm[j];
        temp_var *= temp_var;
        energy[j] *= inv_n_obs;
        mean_energy += temp_var;
        for(i=0; i<n_obs; i++)
            temparray[i] = bg_array[i][j];
        trim(temparray, n_obs, limit);
        pse(temparray, n_obs, wintype);
        mean_est = 0.0;
        var_est = 0.0;

        write(result,temparray,sizeof(double)*(n_obs+1));

        close_pt = 15*n_obs/16;
        for(i=close_pt-8; i<close_pt; i++){
            temp_var = temparray[i];
            mean_est += temp_var;
            temp_var *= temp_var;
            var_est += temp_var;
            }
        mean_est *= .125;
        var_est = .125 * var_est - (mean_est * mean_est);
        total = 0.0;
        for(i=0; i<sum_window; i++)
            total += temparray[n_obs-i];

        kscore[j] = (total-mean_est)/sqrt(sum_window*var_est);
        alt_metric[j] = (total-mean_est)/(datasum[j] * datasum[j]);
    }
```

```
}

out_results(dum, datasm)
FILE *dum;
double datasm[];
{
        register int i;

        for(i=0; i<beat_length; i++) {
                fprintf(dum,"%g    %g    %g\n", datasm[i], alt_metric[i],
kscore[i]);
                alt_metric[i] = 0.0;
                kscore[i] = 0.0;
              }
}


get_pks()
{
int     i;
register double r = 0.0, rmax = 0.0, test=0.0;
float   rcoef=0.0;
char    dummy1[10], dummy2[10], dummy5[10];

        for(i=0; i<n_obs; i++){
                fscanf(pksfp, "%s %ld %s %f %f %s\n", dummy1, &pks[i], dummy2,
 &temp_offset[i], &rcoef, dummy5);
                r = rcoef;
                if(r > rmax){
                        rmax = r;
                        best_index = i;
                        }
                r_array[i] = r;
                rsum += r;
                r *= r;
                rsum2 += r;
                }
        if(joe_switch){
                rmean = rsum/n_obs;
                rstd = sqrt((rsum2/n_obs)-rmean*rmean);
                test = rmean - 3.0*rstd;
                for(i=0; i<n_obs; i++){
                        if(r_array[i] < test)
                                bad_beats[bad_counter++] = i;
                }
        }
}
```

```
get_data(data)
double  data[];
{

/*      Subroutine to retrieve vector magnitude of ECG given pk location
        Data stored as interleaved samples of channels X, Y, and Z

*/
        register double *dptr;
        register short *ptr, *eptr;
        register int i;
        register double temp, gain, b_est;

        int bufsize, j;
        double base[3];
        double data_x[MAXLEN];
        double data_y[MAXLEN];
        double data_z[MAXLEN];
        double datum[MAXLEN];
        long    offset;
        bufsize = 1024 * 2 * data_chan; /* retrieve 1024 samples from each
channel */


/*      Now, we start data OFFSET=300 msec before fiducial point */

        offset = 2*data_chan * (pks[beat_no] + (pk_offset - OFFSET)*Samprt
/1000);
        lseek(datafd, offset, 0);
        read(datafd,buf,bufsize);

/*      Create a baseline estimate for each channel     */

        for(i=0; i<nchan; i++){
                base[i] = 0.0;
/*      Start baseline calculations -base_offset-8 to left of QRS */
                ptr = &buf[chan[i]+(OFFSET+base_offset-8)*data_chan*Samprt/
1000];
                eptr = &buf[chan[i]+(OFFSET+base_offset+8)*data_chan*Samprt/
1000];
                while(ptr < eptr){
                        base[i] += *ptr;
                        ptr += data_chan;
                }
                base[i] *= 0.0625*1000/Samprt;
        }
```

```
        dptr = data;
        for(i=0; i<nchan; i++){
                ptr = &buf[chan[i]];
                eptr = &buf[MAXLEN*data_chan+chan[i]];
                gain = g[i];
                b_est = base[i];
                while ( ptr < eptr ){
                        temp = (*ptr - b_est) * gain;
                        temp *= temp;
                        *dptr++ += temp;
                        ptr += data_chan;
                }
                dptr = data;
        }

        /*  If individual lead analysis is desired the program creates
big_array  */
        /*  and datasum arrays for each lead that the gain is not zero. */
        /*  ------------------------------------------------------------*/

        if(i_switch)   {

             ptr = &buf[0];
             eptr = &buf[MAXLEN*data_chan+0];
             gain = g[0];
             b_est = base[0];
             j=0;
             while ( ptr < eptr ){
                   data_x[j++] = (*ptr - b_est) * gain;
                   ptr += data_chan;
             }
             shift_data(data_x, temp_offset[beat_no]);
             for(i=0; i<MAXLEN; i++)  {
               datasum_x[i] += data_x[i];
               big_array_x[beat_no][i] = data_x[i];
               data_x[i] = 0.0;
             }


             ptr = &buf[1];
             eptr = &buf[MAXLEN*data_chan+1];
             gain = g[1];
             b_est = base[1];
             j = 0;
             while ( ptr < eptr ){
```

```
                data_y[j++] = (*ptr - b_est) * gain;
                ptr += data_chan;
            }
            shift_data(data_y, temp_offset[beat_no]);
            for(i=0; i<MAXLEN; i++)  {
              datasum_y[i] += data_y[i];
              big_array_y[beat_no][i] = data_y[i];
              data_y[i] = 0.0;
            }


            ptr = &buf[2];
            eptr = &buf[MAXLEN*data_chan+2];
            gain = g[2];
            b_est = base[2];
            j = 0;
            while ( ptr < eptr ){
                  data_z[j++] = (*ptr - b_est) * gain;
                  ptr += data_chan;
            }
            shift_data(data_z, temp_offset[beat_no]);
            for(i=0; i<MAXLEN; i++)  {
              datasum_z[i] += data_z[i];
              big_array_z[beat_no][i] = data_z[i];
              data_z[i] = 0.0;
            }
        }

        for(i=0; i<MAXLEN; i++){
                *dptr = sqrt(*dptr);
                dptr++;
        }

        shift_data(data, temp_offset[beat_no]);

        for(i=0; i<MAXLEN; i++){
                energy[i] += data[i]*data[i];
                datasum[i] += data[i];
                big_array[beat_no][i] = data[i];
                data[i] = 0.0;
        }
}

/*****************************************************/

shift_data(data, fract)
double  data[];
```

```
float   fract;
{
int     i;
double  cfract;

        if(fract > 0){
                cfract = 1 - fract;
                for(i=0; i<MAXLEN-1; i++)
                        data[i] = cfract * data[i] + fract * data[i+1];
                }
        if(fract < 0){
                fract *= -1;
                cfract = 1 - fract;
                for(i=1; i<MAXLEN-1; i++)
                        data[MAXLEN-i] = cfract * data[MAXLEN-i] + fract *
data[MAXLEN-i-1];
                }
        return(0);
}

/* trim  */
/*      NO LONGER ..trims the outliers of an array of length "length".
        NO LONGER ..replaces elements which are "limit" standard deviations
from the mean by the mean
        STILL     ..zero-means the entire trimmed array
*/

trim(array, length, limit)
double  array[];
int     length;
double  limit;
{
        register int i;

        double mean, stdev=0.0, sum_sq = 0.0, sum = 0.0;
        register double value, temp;

        for(i=0; i< length; i++){
                temp = array[i];
                sum += temp;
/*              temp *= temp;
                sum_sq += temp;         */
        }
        mean = sum/length;
/*      stdev = sqrt( (sum_sq/length) - mean*mean );
        temp = limit * stdev;           */
        for(i=0; i< length; i++){
```

```
                value = array[i];
                value -= mean;
/*              if((value > temp) || (value < (-1)*temp))
                        value = 0.0;    */
                array[i] = value;
        }
}


/*      power spectrum estimate         */

pse(array,length, wintype)
double  array[];
int     length;
int     wintype;
{
        register        int     i;
        double  dummy[2050];
        register double scale;

        scale = 1.0/((double)length);

        for(i=0; i<length; i++) dummy[i] = array[i]; /* pass values to dummy
array */

        for(i=length; i<2050; i++) dummy[i] = 0.0; /*    pad dummy array */

        length *= 2; /* twice as long due to padding */

        fft(dummy, length); /* take fft of dummy and store results in dummy */

        for(i=0; i<length+2; i++) dummy[i] *= scale;

        ccmx(dummy, length); /* complex conjugate multiply */

        ifft(dummy, length); /* inverse fft to construct autocorrelation */

        window(dummy, length, wintype); /* window the autocorrelation function
 with wintype
                                        1 = rectangular, 2 = Hanning, 3 = Gaussian,
 4 = Bartlett
                                        Takes into account the biasing of zero
padding       */

        fft(dummy, length); /* fft of autocorrelation function */

        mag(dummy, length); /* create magnitude of fft  == power spectrum
```

```
est */

        if(smooth_switch)
                sm_spect(dummy, length);        /* smooth the spectrum */

        for(i=0; i<(length/2)+1; i++) array[i] = dummy[i];


}

fft(array, length)
double  array[];
int     length;
{
float   farray[2050];
register int i;
        for(i=0; i<length+2; i++)
                farray[i] = array[i];
        ffa_(farray, &length);
        for(i=0; i<length+2; i++)
                array[i] = farray[i];


}


ccmx(array, length)
double  array[];
int     length;
{
        register        int     i;
        register        double  imag, real;

        for(i=0; i<length+2; i=i+2){
                real = array[i];
                imag = array[i+1];
                real *= real;
                imag *= imag;
                real += imag;
                array[i] = real;
                array[i+1] = 0.0;
        }
}

mag(array, length)
double  array[];
int     length;
```

```
{
        register int    i;
        register double mag2, real, imag;

        for(i=0; i<length+2; i=i+2){
                real = array[i];
                imag = array[i+1];
                real *= real;
                imag *= imag;
                mag2 = real + imag;
                array[i/2] = sqrt(mag2);
        }
        for(i=(length+2)/2; i<length+2; i++)
              array[i] = 0.0;
}


ifft(array, length)
double  array[];
int     length;
{
float   farray[2050];
int     i;
        for(i=0; i<length+2; i++)
                farray[i] = array[i];
        ffs_(farray, &length);
        for(i=0; i<length+2; i++)
                array[i] = farray[i];

}

window(array, length, wintype)
double  array[];
int     length;
int     wintype;
{
        register int    i;
        double          sigt;
        register double         wind, scale;

        sigt = ((double)(length))/(2*TWOPI);

/*      Correct for biasing encountered by zero padding */

        for(i=0; i<length/2; i++){
                scale = length/((double)(length-2*i));
                array[i] *= scale;
```

```
        }
        array[length/2] = 0.0;
        for(i=length/2+1; i<length; i++){
                scale = length/((double)(2*i-length));
                array[i] *= scale;
        }

        /*      Rectangular Window    */
/*      if(wintype == 1) printf(" Using Rectangular window for spectral
estimation\n"); */

        /*      Hanning Window  */

        if(wintype == 2){
/*              printf(" Using a Hanning window for spectral estimation\n");*/
                for(i=0; i<length; i++){
                        wind = (1+cos(TWOPI*i/length))/2;
                        array[i] *= wind;
                }
        }

        /*      Gaussian Window  (a la Ron Berger) */

        if(wintype == 3){
/*              printf(" Using a Gaussian window for spectral estimation\n");
   */
                for(i=0; i<length/2; i++){

                        wind = exp(-(double)(i*i)/(2*sigt*sigt));
                        array[i] *= wind;
                        array[length-i] *= wind;
                }
        }

         /*     Bartlett Window  */

        if(wintype == 4){
/*              printf(" Using a Bartlett window for spectral estimation\n");
 */
                for(i=0; i<length; i++){
                        wind = 1 - 2 * ((double)(i))/(length);
                        if(wind < 0.0) wind = -wind;
                        array[i] *= wind;
                }
        }
        if(wintype > 4 || wintype < 1)  printf(" Bad window type - %d\n",
wintype);
```

```
}

sm_spect(array, length)
double  array[];
int     length;
{
        int i, j;
        int smooth_window;
        int half;
        int vect_end;
        double temp_smooth = 0.0;
        float   vector[2050];

        half = length/2;
        vect_end = 2*length;
        smooth_window = length/128; /* length is twice the size of the
spectrum */
                                    /* we want to smooth over 1/64 th of the
spectrum */

        for(i=0; i<half; i++){
                vector[half+i] = array[i];
                vector[half-i] = array[i];
                vector[length+i] = array[half+i];
                vector[vect_end-i] = array[half+i];
        }
        vector[length+half] = array[length];
        for(i=half; i<half+length+1; i++){
                temp_smooth = 0.0;
                for(j=0; j<smooth_window; j++)
                        temp_smooth += vector[i-j];
                temp_smooth /= smooth_window;
                array[i-half] = temp_smooth;
                }
}

/*      npwr2   */
/*      Finds the nearest power of 2 less than the candidate number */

npwr2(number)
int     number;
{
        register int i=1;

        if(number < 2){
                printf(" Bad number of observations - %d\n", number);
                exit(0):
```

```
        }
        while(number >= 2*i ) i *= 2;
        return(i);
}
```

A.3 pwa_auto

```
#!/bin/sh

# pwa_auto
# Nikhil Iyengar, October 1997

# Run this script after running RR.ii and display to revise the
# annotations

# This script will then detect p-waves and calculate p-wave
# alternans

DB="/home/nik/header";export DB
datapath="/ldata2/nik";export datapath

BEATS="128"
rate="s"
#Number of iterations for cross-correlation
ITERS="3"
rev=`expr $ITERS - 1 `

if [ -z "$case" ]
then
   echo -n "Case number does not exist. "
   echo -n "Enter Case number "
   read case
else
   case="$case"
fi

if [ -z "$int" ]
then
   echo -n "Enter intervention code (e.g. nsr,a600,...) "
   read int
else
   int="$int"
fi

#*****************************************************************
# Assign ID code
```

```
#*********************************************************************
ID="$case.$int"
echo "ID=$ID"

if [ -z "$1" ]
then
   file="$ID"
else
   file="$1"
fi

dpath="$datapath/$case/"$case"_$int";export dpath
echo "dpath=$dpath"

if [ -z "$dpath" ]
then
  echo "ERROR  dpath does not exist."
  exit 1
fi

wpath=`pwd`

if [ !  -f "$dpath/$file" ]
then
  echo "Can't find $file in $dpath "
  echo "Your choices of data files are "
  ls $dpath
  exit 1
fi

export wpath file

#*********************************************************************
#Sampling rate
#*********************************************************************
SAMP=`awk 'NR==5 {print $0}' $DB/header.$file`

#*********************************************************************
#Number of channels to analyze
#*********************************************************************
channels=`awk '/channels/ {print $2}' $DB/header.$file`

if [ -z "$channels" ]
then
  echo "Can't find channel # in header file "
  echo -n "Enter number of data channels [3] "
```

```
  read channels
  if [ -z "$channels" ]
  then
    channels="3"
  fi
fi

#*********************************************************************
#set channel calls to variable C
#*********************************************************************
a=1
C="-c0"
while [ "$a" -lt "$channels" ]
do
        C="$C -c$a"
        a=`calc "$a + 1" `
done

#*********************************************************************
#SET GAINS FOR STUDY
#*********************************************************************
OLDGAIN=""
G=""
G1=`awk '/gains/ {print $2}' $DB/header.$file`
G2=`awk '/gains/ {print $3}' $DB/header.$file`
G3=`awk '/gains/ {print $4}' $DB/header.$file`
for GAIN in $G1 $G2 $G3
do
        G="$OLDGAIN -g $GAIN"
        OLDGAIN="$G"
done

# Check to see if a pks file exists.

skipqrs="no"
if [ -f "pks.$ID" ]
then
echo -n "pks file detected.  Use it and skip QRS correlation? "
read ans
if [ "$ans" = "y" ]
then
skipqrs="yes"
fi
fi

# BEGIN CONDITIONAL A
if [ "$skipqrs" = "no" ]
```

```
then
# Create a crude sig. avg. based on RR file

WIDTH=`RRstats RR.$ID | awk '{print \$2}'`
WIDTH=`calc "int( ($WIDTH/$SAMP) * 1000 )"`
sigavg -aint.$ID -d$dpath/$file $G  -n 100 -pRR.$ID -w$WIDTH $C \
 -N $channels -S $SAMP -b -50

# Detect baseline and beginning and end of QRS.

cat int.$ID | awk '{print $7}' | delqrs > temp.1
baseshift=`awk '{print $1}' temp.1`
peakshift=`awk '{print $2}' temp.1`
WINDOW=`awk '{print $3}' temp.1`
rm temp.1

#Allow the user to make changes if program messed up

echo "baseshift="$baseshift
echo "peakshift="$peakshift
echo "WINDOW="$WINDOW
plot.windownik RR.$ID $file $peakshift $WINDOW $baseshift
echo -n "Do you want to make any changes in the template window ? "
read decision
until [ "$decision" = "n" ]
do

echo -n "Shift for baseline estimate (- = left , + = right) [-35 ms] "
read baseshift
[ -n "$baseshift" ] || baseshift="-35";
echo -n "Shift raw peak for centering template ( - = left , + =   right)
[0 ms] "
read peakshift
[ -n "$peakshift" ] || peakshift="0";
echo -n 'Window size[70 ms]? '
read WINDOW
[ -n "$WINDOW" ] || WINDOW="70";
plot.windownik RR.$ID $file $peakshift $WINDOW $baseshift
echo -n "Do you want to make more !!! changes ? "
read foo
if [ "$foo" = "n" ]
then
   break 1
fi

done
```

```
#*********************************************************************
#Maximum allowable shift for template to cross-correlate
#*********************************************************************
SHIFT=`calc "int( $WINDOW/2 )" `

#*********************************************************************
#note -x switch in xcnew will prevent a reloction of raw peak
#estimate to the highest point within the template window.
#*********************************************************************

#*********************************************************************
#If you want Alternans Metric and K Score to be calculated
#from last 4 points of spectrum (rather than last point e.g.
#Nyquest frequency) insert -s4 as switch to spitnew program."
#THIS OPTION IS USEFULL WHEN THE ALTERNANS ENERGY IS SHIFTED OVER
#TO ADJACENT FREQUENCY BANDS SECONDARY TO PHASE RESETTING OF BEATS
#*********************************************************************

echo "baseshift=$baseshift" >> var.$file
echo "peakshift=$peakshift" >> var.$file
echo "WINDOW=$WINDOW" >> var.$file
echo $baseshift > base.$ID

xcnew_$rate -aavg.$ID  -d$dpath/$file $G  -n$ITERS -pRR.$ID \
 -s$SHIFT  -w$WINDOW -l $peakshift -o $baseshift -x $C \
 -N $channels -S $SAMP
get :1, pks.rev$rev % -P > pks.$ID
rm pks.rev*
WIDTH=`RRstats pks.$ID | awk '{print $2}'`
badbeat -w $WIDTH -c 0.95 -i 1 -m 1 -p pks.$ID -t 150  -b bad.$ID

export start end
echo "start: \$start"  >> "var.$file"
echo "end: \$end"   >>  "var.$file"

echo
echo "QRS cross-correlation completed.  Press return ..."
read anykey

else
WIDTH=`RRstats pks.$ID | awk '{print $2}'`
baseshift=`cat base.$ID`
fi
#END CONDITIONAL A


# Check to see if a p_pks file exists.
```

```
skipp="no"
if [ -f "p_pks.$ID" ]
then
echo -n "p_pks file detected.  Use it and skip P correlation? "
read ans
if [ "$ans" = "y" ]
then
skipp="yes"
fi
fi

#BEGIN CONDITIONAL B
if [ "$skipp" = "no" ]
then

#Now do p-detection.  Start by averaging on the QRS, skipping badbeats

OLDGAIN=""
GX=""
echo -n "Enter p-detection gain for channel 0 [$G1]: "
read GX1
echo -n "Enter p-detection gain for channel 1 [$G2]: "
read GX2
echo -n "Enter p-detection gain for channel 2 [$G3]: "
read GX3
[ -n "$GX1" ] || GX1="$G1";
[ -n "$GX2" ] || GX2="$G2";
[ -n "$GX3" ] || GX3="$G3";
for GAIN in $GX1 $GX2 $GX3
do
GX="$OLDGAIN -g $GAIN"
OLDGAIN="$GX"
done

peaks="pks.$ID"
WIDTH=`calc "int( ($WIDTH/$SAMP) * 1000 )"`
sigavg -aint.$ID -d$dpath/$file $GX  -n 100 -p pks.$ID -w$WIDTH $C \
 -N $channels -S $SAMP -b $baseshift -x bad.$ID

# Detect baseline and beginning and end of P.

cat int.$ID | awk '{print $7}' | delp > temp.1
baseshift=`awk '{print $1}' temp.1`
peakshift=`awk '{print $2}' temp.1`
WINDOW=`awk '{print $3}' temp.1`
rm temp.1
```

```
#Allow the user to make changes if program messed up

#BEGIN LOOP A
until [ "black" = "white" ]
do

echo "baseshift="$baseshift
echo "peakshift="$peakshift
echo "WINDOW="$WINDOW

plot.windownik $peaks $file $peakshift $WINDOW $baseshift

echo -n "Do you want to make any changes in the template window ? "
read decision

#BEGIN LOOP B
until [ "$decision" = "n" ]
do

echo -n "Shift for baseline estimate (- = left , + = right) [-35 ms] "
read baseshift
[ -n "$baseshift" ] || baseshift="-35";
echo -n "Shift raw peak for centering template ( - = left , + =   right)
[0 ms] "
read peakshift
[ -n "$peakshift" ] || peakshift="0";
echo -n 'Window size[70 ms]? '
read WINDOW
[ -n "$WINDOW" ] || WINDOW="70";
plot.windownik $peaks $file $peakshift $WINDOW $baseshift
echo -n "Do you want to make more !!! changes ? "
read foo
if [ "$foo" = "n" ]
then
   break 1
fi

done
#END LOOP B


SHIFT=`calc "int( $WINDOW/2 )" `
echo "baseshift=$baseshift" >> var.$file
echo "peakshift=$peakshift" >> var.$file
echo "WINDOW=$WINDOW" >> var.$file
```

```
echo $baseshift > p_base.$ID

rate=s

xcnew_$rate -aavg.$ID  -d$dpath/$file $GX  -n$ITERS -p$peaks -s$SHIFT
-w$WINDOW -l $peakshift -o $baseshift -x $C -N $channels -S $SAMP
get :1, pks.rev$rev % -P > p_pks.$ID
rm pks.rev*

#display $file p_pks.$ID
echo "Finished p-wave cross-correlation.  Press return ..."
read anykey


#BEGIN LOOP C
until [ "good" = "bad" ]
do

corcof="0.95"
echo -n "Correlation coef. threshold to use [0.95]? "
read corcof
[ -n "$corcof" ] || corcof="0.95";

threshr="100"
echo -n "Maximum allowable RR interval deviation (msec) [100]? "
read threshr
[ -n "$threshr" ] || threshr="100";

WIDTH=`RRstats p_pks.$ID | awk '{print $2}'`
badbeat -w $WIDTH -c $corcof -i 1 -m 1 -p p_pks.$ID -t $threshr -b p_bad.$ID
numbad=`cat p_bad.$ID | awk '{print $4}' | grep 1 | wc -l`
totalnum=`cat p_bad.$ID | wc -l`
echo "There are $numbad bad beats out of $totalnum beats."

echo -n "Try with a different threshold (y/n)? "
read barfu

if [ "$barfu" = "n" ]
then
   break 1
fi
done
#END LOOP C

echo "Running display.  Be sure to redo the cross-correlation if"
echo "you make any changes"
```

```
display $file "p_pks.$ID"

echo -n "Do you want to redo the cross-correlation? "
read fubar

if [ "$fubar" = "n" ]
then
   break 1
fi
peaks="p_pks.$ID"
peakshift=0
done
#END LOOP A

else
WIDTH=`RRstats p_pks.$ID | awk '{print $2}'`
baseshift=`cat p_base.$ID`
fi
# END CONDITIONAL B


start=`segment  -b p_bad.$ID -c 3 -l $BEATS`
end=`calc "$start + $BEATS - 1"`
awk '(NR - 1) >= '$start' { print $0 }' p_pks.$ID > bestpks.$$
beatout=""
for F in `awk '$1 >= '$start' && $1 <= '$end' && $4 == 1 \
  {print ( $1 - '$start' ) }'  p_bad.$ID  `
do
beatout="$beatout -b $F "
done
WIDTH=`calc "int( ($WIDTH/$SAMP) * 1000 )"`
mkavnew_$rate -aint.$ID -d$dpath/$file $G  -n 10 -pbestpks.$$ \
 -w$WIDTH $C -N $channels -S $SAMP -b $baseshift
PTERM=xw
export PTERM
segs  $ID $start $end p_pks > seg.$ID
spitnew3_$rate -n$BEATS -l $baseshift $G   -d$dpath/$file -pbestpks.$$ \
  -mmat.$ID $C  -N $channels -Bbig.$ID -w $WIDTH $beatout -S $SAMP -i
rm *.$$


# Now do the stuff in FINAL,,

#############################################################################

PTERM=xw; export PTERM
BEAT="128"
```

```
# First step:  use vector magnitude from spitnew to determine
# pstart and pend.

# IMPORTANT:  The number of points in mat.*.* before the fiducial
# point (now the P-wave) is hard-coded into spitnew_3.  Currently
# this value, spitoff, is:
spitoff=300

cat mat.$ID | awk '{print $1}' | delp > temp.1
pcent=`awk '{print $2}' temp.1`
pwin=`awk '{print $3}' temp.1`

# So, the center of the p-wave is at spitoff and the window width
# is pwin.

pstart=`calc "int( $spitoff - (pwin/2) )"`
pend=`calc "int( $spitoff + (pwin/2) )"`
math mat.$ID 0 -o"row*1000/$SAMP,c0" |plt 0 1 -st   -F"
hl .5 1.1 c 1
Vector Magnitude $ID
xa 0 - 10 -  5 -"

echo -n "Enter start of p-wave (ms) [$pstart] "
read ans
if [ "$ans" != "" ]
then
  pstart="$ans"
fi

echo -n "Enter end of p-wave (ms) [$pstart] "
read ans
if [ "$ans" != "" ]
then
  pend="$ans"
fi

echo -n "Enter start of BLANKING interval for P-wave (ms) "
read bstr_p
echo -n "Enter end of BLANKING interval for P-wave (ms) "
read bend_p

blank_p=""
if [ "$bstr_p" != "" ]
then
  Bstr_p=`calc "$bstr_p*$SAMP/1000"`
  if [ "$bend_p" != "" ]
```

```
  then
    Bend_p='calc "$bend_p*$SAMP/1000"'
    blank_p="-B $Bstr_p $Bend_p"
  fi
fi

# Now, write out the start and end of P-wave to be used for matlab

echo "$pstart   $pend" > ptimes.$ID
```

A.4 sigavg.c

```
/*  sigavg.c          October, 1997
    By Nikhil Iyengar
    based on mkavgm.c by Joe Smith,  December,  1987

Function: constructs a signal average of each of N
          ECGs together with the vector magnitude.

Note:  only n GOOD beats are included in the average.

Output:  eight ASCII columns
  column 1:  baseline subtracted from channel 1 beats, beats averaged.
  column 2:  baseline subtracted from channel 1 beats, beats squared
             and the average of the squares taken
  columns 3-6:  same for channel 2,3
  column 7:  baseline subtracted from each channel of each beat,
             vector mag. of each beat computed, vector mags. averaged
  column 8:  baseline subtracted from each channel of each beat, vector
             mag. of each beat computed, squares of vector mags averaged.
*/


#include        <stdio.h>
#include        <math.h>
#include        <sgtty.h>
/* #include     <libmp.h> */
#include        "utils.c"

#define         OFFSET  200
#define         MAXCHAN 3
#define         MAXSCHAN 8
#define         MAXLEN  1600
#define         MAXOBS  1024

FILE    *pksfp, *outfp, *badfp;
```

```
int     chan[MAXCHAN];
int     datafd, n_obs=256, width = MAXLEN, s=NO, data_chan, nchan;
int     beat_no;
int     base_offset=0;

char    option(), *opt_argv(), ctmp, tmode = 0;
char    cr;
char    pksname[20], dataname[20];
long    pks[MAXOBS]; /* peak locations  */
long    pk;
double  base, g[MAXCHAN];
double  data[MAXCHAN][MAXLEN];
double  data2[MAXCHAN][MAXLEN];
double  mag[MAXLEN], mag2[MAXLEN];
double  temp_array[MAXCHAN][MAXLEN];
double  temp_offset[MAXOBS];
double  inv_n_obs;
short   Samprt=1000, gflag=0, cflag=0;
int     f_beat=0, bufoff;
int     badbeat[1000], badcount=0, curbad=0;

main(argc, argv)
int argc;
char **argv;
{

        char    xtitle[20], ytitle[20];

        int     i, j, k;

        while( (ctmp = option(&argc, argv)) != 0){
                switch(ctmp){
                        case 'a': /* output file name */
                                outfp = fofile( opt_argv(), "w" );
                                break;

                        case 'b': /* baseshift */
                                base_offset = atoi(opt_argv());
                                break;

                        case 'c': /* channels to be analyzed */
                                chan[cflag] = atoi(opt_argv());
                                cflag += 1;
                                break;

                        case 'd': /* data file  */
                                datafd = ofile( opt_argv(), 0 );
```

```
                break;

            case 'g': /* relative gains of channels */
                g[gflag] = atof(opt_argv());
                gflag += 1;
                break;

            case 'N': /* number of channels in data */
                data_chan = atoi(opt_argv());
                nchan = data_chan;
                break;

            case 'n': /* number of beats to average */
                n_obs = atoi(opt_argv());
                /* n_obs = npwr2(n_obs); */
                break;

            case 'p': /* peaks file */
                pksfp = fofile( opt_argv(), "r" );
                break;

            case 's': /* Number of first beat */
                f_beat = atoi(opt_argv());
                break;

            case 'w':
                width =  atoi(opt_argv());
                break;

            case 'x':  /* badbeats file */
                badfp = fofile(opt_argv(), "r");
                break;

            case 'S':
                Samprt =  atoi(opt_argv());
                break;

            case 'h': /* help */
                printf("\n\n");
                printf("mkavgm.c - N-channel ecg average\n");
                printf("        expects a pks file in the csp
format\n");
                printf("\n\n");
                printf("command line arguments:\n");
                printf("        -a(filename) - name of ouput
(average) file\n");
                printf("        -c(chan #)   - iteratively
```

```
called for each channel\n");
                                printf("        -d(filename) - name of raw
data file\n");
                                printf("        -g(gain)     - iteratively
called for each channel\n");
                                printf("        -n(number)   - number of beats
 to average\n");
                                printf("        -N(# of data chans)\n");
                                printf("        -p(filename) - name of peaks
file\n");
                                printf("        -w(number)   - width of
averaging buffer\n");
                                printf("        -s(number)   - number of first
 beat to start averaging\n");
                                printf("        -S(number)   - the sampling
rate\n");
                                printf("        -x(filename) - name of badbeat
 file\n");
                                printf("\n");
                                exit(0);                        default:
                                printf("\n ILLEGAL OPTION %c\n", ctmp);
                                exit(0);
                        }
                }

        bufoff = width/2;

        /* Init arrays to zero */

        for(i=0;i<MAXLEN;i++) {
          for(j=0;j<MAXCHAN;j++) {
            data[j][i] = 0;
            data2[j][i] = 0;
          }
          mag[i] = 0;
          mag2[i] = 0;
        }

        if(outfp == NULL) outfp = fofile("average", "w" );
        get_pks();      /* loads array of peak locations */
        inv_n_obs = 1.0/(double)n_obs;

        for(beat_no=0; beat_no<n_obs; beat_no++)
                get_data();

        for(j=0; j<(width*Samprt/1000); j++){
                for(k=0; k<nchan; k++){
```

```
                        data[k][j] *=  inv_n_obs;
                        data2[k][j] *= inv_n_obs;
                }
                mag[j] *= inv_n_obs;
                mag2[j] *= inv_n_obs;
        }
        for(i=0; i<(width*Samprt/1000); i++)
        {
                for(k=0; k<nchan; k++)
                {
                        fprintf(outfp, "  %g        %g        ", data[k][i],
data2[k][i]);
                }
                fprintf(outfp,"  %g       %g  \n", mag[i], mag2[i]);
        }
        exit(0);
}

get_pks()
{
  int   chk, i, row, val, done;
  char  dummy1[10], dummy2[10], dummy4[10], dummy5[10];
  char  buf[100];
  float tmpoff;

  /* Open the badbeats file.  Note that if this file does not exist,
     the program will say so, and will simply assume all beats are
     good */

  if (badfp == NULL) {
    fprintf(stderr, "sigavg:  No badbeat file specified.  Assuming all beats
are good.\n");
    badbeat[0] = 10000;
  }
  else {
    /* Read in the badfile */

    row = 0;
    while (fgets(buf, sizeof(buf), badfp) != NULL) {

      /* Get the value of the 4th column of badfile */
      if (sscanf(buf, "%*f %*f %*f %d", &val) != 1) {
        fprintf(stderr, "sigavg: error encountered in row %d\n", row);
        return(-2);
      }

      if (row >= f_beat) {
```

```
        if (val == 1) {
          badbeat[badcount++] = row;
        }
        if ((row-f_beat-badcount+1) == n_obs) {
          done=1;
        }
      }

      if (done) break;
      row++;
    }
  }

  curbad=0;

  for(i=0; i<f_beat+n_obs+badcount; i++){
    chk = fscanf(pksfp, "%s %ld %s %f %s %s\n", dummy1, &pk, dummy2, &tmpoff,
dummy4, dummy5);
    if ((i>=f_beat) && (i != badbeat[curbad])) {
      pks[i] = pk;
      temp_offset[i] = tmpoff;
    }
    else if (i == badbeat[curbad]) {
      curbad++;
    }
  }
}

get_data()
{

/*      Subroutine to retrieve vector of data given pk location
*/

        int i,j,k;
        short   buf[2*MAXLEN*MAXSCHAN]; /* twice MAXLEN available */
        int     m, bufsize;
        double temp2, temp, gain;
        long    offset;

        bufsize = 4 * MAXLEN * MAXSCHAN; /* twice the needed data */
/* begin data buffer bufoff msecs to left of QRS marker */
        offset = 2 * data_chan * (pks[beat_no] - bufoff*Samprt/1000 );
        lseek(datafd, offset, 0);
        read(datafd,buf,bufsize);

/*      DO IT CHANNEL BY CHANNEL        */
```

```
        for(k=0; k<nchan; k++){
                base = 0.0;
                for(i=bufoff+base_offset-8;
                    i<bufoff+base_offset+8;
                    i++)
                        base += (double)(buf[data_chan*i+chan[k]]);
                base *= .0625*1000/Samprt;
                gain = g[k];

                for(i=0; i<MAXLEN; i++){
                        m = data_chan * i;
                        temp = gain * (buf[m + chan[k]] - base);
                        temp_array[k][i] = temp;
                }
        }
        lin_shift();

        /* At this point, temp_array[k][i] contains the current beat
           for each channel k. */

        for(i=0; i<MAXLEN; i++){
                temp2 = 0.0;
                for(k=0; k<nchan; k++){
                        temp = temp_array[k][i];
                        temp2 += (temp * temp);
                        data[k][i] += temp;
                        data2[k][i] += temp * temp;
                }
                mag2[i] += temp2;
                mag[i] += sqrt(temp2);
        }
}



/*      npwr2   */
/*      Finds the nearest power of 2 less than the candidate number */

npwr2(number)
int     number;
{
        int i=1;

        if(number < 2){
                printf(" Bad number of observations - %d\n", number);
                exit(0);
        }
```

```
        while(number >= 2*i ) i *= 2;
        return(i);
}

lin_shift()
{
int     k, p;
double  fract, cfract;

        fract = temp_offset[beat_no];
        if(fract > 0){
                cfract = 1 - fract;
                for(k=0; k<nchan; k++){
                        for(p=0; p<MAXLEN-1; p++){
                                temp_array[k][p] = cfract * temp_array[k][p] +
 fract * temp_array[k][p+1];
                        }
                }
        }
        if(fract < 0){
                fract *= -1;
                cfract = 1 - fract;
                for(k=0; k<nchan; k++){
                        for(p=1; p<MAXLEN; p++){
                                temp_array[k][MAXLEN-p] = cfract *
temp_array[k][MAXLEN-p] + fract * temp_array[k][MAXLEN-p-1];
                        }
                }
        }
        return(0);
}
```

A.5 delp.c

```
/*  delp.c  -- Detects the beginning and end of P
    Nikhil Iyengar,  10/24/97

    Usage:  delp

    Takes column input file from standard input.  This column file
    should contain the vector magnitude of an average beat with the
    annotation at precisely the center of the beat (from sigavg.c, for
    example).

    Outputs three numbers on one line:  a   b   c
    a = baseline.  points to the right of b where baseline is located.
```

```
                  should always be positive
    b = P center.  points to the right of the annotation that p
                  center is located.
    c = P width.   width of the P around its center.
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

double mean(double [], int, int);
double std(double [], int, int);


main()
{

  double beat[3000], qrsampl, baseampl;
  int i, width, qrsindex, baseindex, newindex, center;
  double curstd, minstd, thresh;
  double curmax, prevmax, leftmin, pampl;
  int curcount, pindex, pstart, pend, out1, out2, out3;


  i=1;
  while (scanf("%lf", &beat[i]) == 1) {
    i++;
  }
  width = i-1;
  center = width/2;

  /* Find the maximum point in beat */

  qrsampl = 0;
  for(i=1;i<=width;i++) {
    if (beat[i] > qrsampl) {
      qrsampl = beat[i];
      qrsindex = i;
    }
  }

  /* Find the baseline point */

  baseampl = qrsampl;
  for(i = qrsindex-150 ; i<qrsindex ; i++) {
    if (beat[i] < baseampl) {
      baseampl = beat[i];
```

```
    baseindex = i;
  }
}

minstd = qrsampl;
for(i = baseindex - 30 ; i<baseindex + 30; i++) {
  curstd = std(beat,i-7,i+7);
  if (curstd < minstd) {
    minstd = curstd;
    newindex = i;
  }
}
baseindex = newindex;
baseampl = mean(beat,baseindex-7,baseindex+7);

/* Go left from the baseline looking for a local maximum over
   a hundred point area.  This should be the P-peak */

curmax = 0;
prevmax = 0;
curcount = 0;
for(i = baseindex; i>0 ; i--) {
  if (beat[i] > curmax) {
    curmax = beat[i];
    curcount = 1;
  }
  else
    curcount++;
  if (curcount == 100) {
    if (curmax > prevmax) {
      prevmax = curmax;
      pindex = i+99;
    }
    curmax = 0;
  }
}
pampl = beat[pindex];

/* Find the minimum to the left of the p-wave */

leftmin = qrsampl;
for(i=pindex-100 ; i<pindex ; i++) {
  if(beat[i] < leftmin) {
    leftmin = beat[i];
  }
}
```

```
  /* The beginning and end of the P will be when the amplitude first
     falls below 15% of the difference between QRS and baseline */

  thresh = leftmin + 0.15 * (pampl - leftmin);
  for(i = pindex; i > pindex-100; i--) {
    if (beat[i] < thresh) {
      pstart = i;
      break;
    }
  }

  thresh = baseampl + 0.15 * (pampl - baseampl);
  for(i=pindex; i < pindex+100; i++) {
    if (beat[i] < thresh) {
      pend = i;
      break;
    }
  }

  out1 = baseindex - (pstart+pend)/2;
  out2 = (pstart+pend)/2 - center;
  out3 = pend - pstart;


  printf("%d \t %d \t %d\n", out1, out2, out3);

}


double mean(double vec[], int start, int end)
{
  int i;
  double sum=0, avg;

  for(i=start ; i<=end ; i++) {
    sum = sum + vec[i];
  }

  return sum/(end-start+1);

}

double std(double vec[], int start, int end)
{
  int i;
  double sum=0, avg;
```

```
  for(i=start ; i<=end ; i++) {
    sum = sum + vec[i];
  }
  avg = sum/(end-start+1);

  sum = 0;
  for(i=start ; i<=end ; i++) {
    sum = sum + (vec[i] - avg)*(vec[i] - avg);
  }

  return  sqrt(sum/(end-start));
}
```

A.6 dofinal.m

```
function out = dofinal()


olddir = pwd;
subplot(1,1,1);
number = input('What number? ','s');
exper = input('What experiment (nsr, etc.) ? ','s');
newdir = ['/ldata2/nik/' number '/' number '_' exper];
cd(newdir)

fid1 = fopen(['mat.' number '.' exper], 'rt');
a = fscanf(fid1, '%f %f %f', [3,inf]);
avgbeat = a(1,:)';
alt_met = a(2,:)';
kscore = a(3,:)';
clear a;
fclose(fid1);

%%%%%%%%%%%%%%%% size of plot
pc = 3;  %columns in plot
pr = 4;  %rows in plot


subplot(pr,pc,1)
plot(avgbeat)
title('Nonfiltered vec. mag.');
%subplot(3,1,2)
%plot(alt_met)
%subplot(3,1,3)
%plot(kscore)
```

```
%startp = input('What is the start of the p-wave? ');
%endp = input('What is the end of the p-wave? ');
%startp = 250;
%endp = 320;

fid1 = fopen(['ptimes.' number '.' exper], 'rt');
a = fscanf(fid1, '%f %f');
startp = a(1);
endp = a(2);
fclose(fid1);

kscoretot = mean(kscore(startp:endp));
alt_mettot = mean(alt_met(startp:endp));

fprintf(1, 'The k-score over that region is %f.\n', kscoretot);
fprintf(1, 'The alt-metric over that region is %f.\n', alt_mettot);

subplot(pr,pc,2);
plot(avgbeat(startp-50:endp+50));
title('magnified unfilt. p-wave');

% Now, load up the individual leads and plot them

fid1 = fopen(['mat.' number '.' exper '_x'], 'rt');
fid2 = fopen(['mat.' number '.' exper '_y'], 'rt');
fid3 = fopen(['mat.' number '.' exper '_z'], 'rt');

[a] = fscanf(fid1, '%f %f %f', [3,inf]);
avgbeat_x = a(1,:)';
if length(avgbeat_x) ~= length(avgbeat)
     avgbeat_x = zeros(length(avgbeat),1);
end
clear a;

[a] = fscanf(fid2, '%f %f %f', [3,inf]);
avgbeat_y = a(1,:)';
if length(avgbeat_y) ~= length(avgbeat)
     avgbeat_y = zeros(length(avgbeat),1);
end
clear a;

[a] = fscanf(fid3, '%f %f %f', [3,inf]);
avgbeat_z = a(1,:)';
if length(avgbeat_z) ~= length(avgbeat)
     avgbeat_z = zeros(length(avgbeat),1);
end
clear a;
```

```
subplot(pr,pc,3)
plot(avgbeat_x(startp-50:endp+50));
subplot(pr,pc,4)
plot(avgbeat_y(startp-50:endp+50));
subplot(pr,pc,5)
plot(avgbeat_z(startp-50:endp+50));

% now pass each lead through a filter to get the filtered output.

sampfreq = 1000;
[b,a] = butter(4, 40/(sampfreq/2), 'high');
y1 = filtfilt(b,a,avgbeat_x);
y2 = filtfilt(b,a,avgbeat_y);
y3 = filtfilt(b,a,avgbeat_z);

subplot(pr,pc,6)
plot(y1(startp-50:endp+50));
subplot(pr,pc,7)
plot(y2(startp-50:endp+50));
subplot(pr,pc,8)
plot(y3(startp-50:endp+50));

% combine the individual filtered leads into a vector magnitude

filtavg = sqrt(y1.^2 + y2.^2 + y3.^2);
subplot(pr,pc,9)
plot(filtavg)
subplot(pr,pc,10)
plot(filtavg(startp-50:endp+50));

% now, find the start and end of the p-wave according to valverde.
% calculate the average noise 100 points from the start

avgnoise = mean(filtavg(95:105))

% The algorithm:
% 1.  find where the signal goes above 3 times the average noise for
%      five points in a row.  This is the start of the p-wave. (note:
%     skip the first 50 points because there may be a filter transient
%     there).
% 2.  keep going until you find a 30 msec window where nothing goes above
%     3 times the average.  The beginning of this window is the end
%     of the p-wave.
% 3.  Check to see if you have reached 30% QRS amplitude.  In this case,
%     step 2 has failed.  Go to alternate algorithm in step 4.
% 4.  Redo step 2 except now only a 7 msec window is used.
```

```
% 5.  Refine the beginning and ending points as follows:  Go left or
%     right for startp and endp, respectively, until you get down
%     to a point that is less than or equal to 1 * average noise.
% 6.  If the refining process takes you more than 20 points away, scrap
%     it and use the initial point +/- 10ms.


for curpoint = 50:length(filtavg)
    if sum(filtavg(curpoint:curpoint+2) > 3*avgnoise*ones(3,1)) == 3
       startp = curpoint;
       break;
    end
end
startp

successful = 0;
maxpeak = max(filtavg);
for curpoint = startp:length(filtavg)
    if filtavg(curpoint) > 0.3*maxpeak
       break;
    end
    if sum(filtavg(curpoint:curpoint+29) < 3*avgnoise*ones(30,1)) ==30
       endp = curpoint;
       successful = 1;
       break;
    end
end

if successful == 0
   for curpoint = startp:length(filtavg)
      if filtavg(curpoint) > 0.3*maxpeak
         break;
      end
      if sum(filtavg(curpoint:curpoint+6) < 3*avgnoise*ones(7,1)) ==7
         endp = curpoint;
         successful = 1;
         break;
      end
   end
end

success2 = 0
for curpoint = startp:-1:startp-40
    if sum(filtavg(curpoint:-1:curpoint-1) < 1*avgnoise*ones(2,1)) == 2
       startp = curpoint;
       success2 = 1;
       break;
```

```
    end
end
if success2 == 0
   startp = startp - 10;
end

success2 = 0;
for curpoint = endp:endp+20
    if filtavg(curpoint) <= avgnoise
       endp = curpoint;
       success2 = 1;
       break;
    end
end
if success2 == 0
   endp = endp + 10;
end


if successful == 1
   fprintf(1, 'startp: %f    endp: %f    length: %f.\n', startp, endp,
endp-startp);
end


out = avgbeat;


fclose('all');
cd(olddir);


%fprintf(1,'Where are the files [%s] ? ', pwd)
%newdir = input('','s');
%cd(newdir);
%ls
%cd(olddir);
```

Bibliography

[1] Berne, R. and Levy, M. Cardiovascular Physiology, 7th Edition. Mosby-Year Book, Inc., St. Louis. 1997.

[2] Braunwald, Eugene. Heart Disease, A Textbook in Cardiovascular Medicine.

[3] Bricker JT, Murphy DJ Jr. Atrial Alternans. American Heart Journal, Vol. 119, No. 2, part 1, pp. 412-414, 1989.

[4] Chan EKY, Steinberg JS, Santoni-Rugiu F, Gomes A. P Wave Signal-Averaged Electrocardiography Techniques. A.N.E., Vol. 3, No. 2, pp. 147-152, 1998.

[5] Clancy EA, Smith JM, Cohen RJ. A Simple Electrical-Mechanical Model of the Heart Applied to the Study of Electrical-Mechanical Alternans. IEEE Transactions on Biomedical Engineering, Vol. 38, No. 6, pp. 551-560. 1991.

[6] David, Daniel, et al. Atrial Alternans: Experimental Echocardiographic and Hemodynamic Demonstration During Programmed Pacing. American Journal of Cardiology, Vol. 48, pp. 468-472, 1981.

[7] Donato A, Oreto G, Schamroth L. P wave alternans. American Heart Journal, Vol. 116, No. 3, pp. 875-877, 1988.

[8] Ehlert FA, Steinberg JS. The P Wave Signal-averaged ECG. Journal of Electrocardiology, vol. 28 supplement, pp. 33-39, 1995.

[9] Fukunami M, et al. Detection of Patients at Risk for Paroxysmal Atrial Fibrillation During Sinus Rhythm by P Wave-Triggered Signal-Averaged Electrocardiogram. Circulation, Vol. 83, pp. 162-169. 1991.

[10] Hnatkova K, Obel O, Camm AJ, Malik M. Technical Advances in Signal-Averaged Electrocardiography. Cardiac Electrophysiology Review, Vol. 3, pp. 317-320, 1997.

[11] Hofmann M, Goedel-Meinen L, Beckhoff A, Rehbach K, Schomig A. Analysis of the P Wave in the Signal Averaged Electrocardiogram: Normal Values and Reproducibility. PACE, Vol. 19 part II, pp. 1928-1932, 1996.

[12] Lander P, et. al. The analysis of ventricular late potentials using orthogonal recording. IEEE Trans. Biomed. Eng., Vol. 35, No. 8, pp. 629-642, 1988.

[13] Michelucci A, Padeletti L, Porciani MC, Gensini GF. P-wave Signal Averaging. Cardiac Electrophysiology Review, Vol. 3, pp. 325-328, 1997.

[14] Oppenheim AV, and Schafer R. Discrete-Time Signal Processing. Prentice-Hall, Inc., Englewood Cliffs, NJ. 1989.

[15] Pop T, and Fleischmann D. Alternans in human atrial monophasic action potential. British Heart Journal, Vol. 39, pp. 1273-1275. 1977.

[16] Rosenbaum, D. S., Albrecht, P., Cohen, R. J. Predicting Sudden Cardiac Death From T Wave Alternans of the Surface Electrocardiogram: Promise and Pitfalls. Journal of Cardiovascular Electrophysiology 1996;11:1095-1111.

[17] Rosenbaum, David S., et al. Electrical Alternans and Vulnerability to Ventricular Arrhythmias. N Engl J Med, Vol. 330, pp. 235-241, 1994.

[18] Ruskin JN, Galvin JM, Bharucha DB. P wave electrical alternans analysis as a predictor of atrial fibrillation following cardiac surgery. Research protocol, unpublished.

[19] Scott WA, Donnerstein RL. Alignment of P Waves for Signal Averaging. PACE, Vol. 13 part I, pp. 1559-1562, 1990.

[20] Smith, Joseph M., et al. Electrical alternans and cardiac electrical instability. Circulation, Vol. 77, No. 1, pp. 110-121, 1988.

[21] Tompkins WJ, Webster JG, eds. Design of Microcomputer-Based Medical Instrumentation. Englewood Cliffs, NJ: Prentice-Hall, 1981:114-9.

[22] Valverde, E. et al. Influence of Filtering Techniques on the Time-Domain Analysis of Signal-Averaged P Wave Electrocardiogram. J Cardiovasc Electrophysiol, Vol. 9, pp. 253-260, March 1998.

[23] Zaman AG, Alamgir F, Richens T, Williams R, Rothman MT, Mills PG. The role of signal averaged P wave duration and serum magnesium as a combined predictor of atrial fibrillation after elective coronary artery bypass surgery. Heart, vol. 77, pp. 527-531, 1997.