# *The Tell-Tale Heart*

## A Study in Electrocardiogram Abnormality Pattern Recogntion

**Group 19**
**Jeffrey Chan**
**Naomi Dambacher**
**Michael Pudup**

**18-551, Spring 2000**

## *Acknowledgements*

We would like to thank the Undergraduate Research Initiative for providing us with an emergency SURG grant of $200 for the purchase of the MIT-BIH Arrhythmia Database We would also like to thank the Vice Provost for Education, Indira Nair for providing the funding for our SURG. We would especially like to thank Steve Ives for answering our many questions via zephyr at 2 in the morning. Thanks are also extended to Professor Casasent, Ryan Kellogg and Pete Boettcher for their guidance and assistance.

## *Introduction*

**MOTIVATION**

Electrocardiograms are used by hospitals to monitor patients with known or potential heart problems. By studying the electrocardiograms of the patients, cardiologists can detect rhythmic problems, heart rotation, and some symptoms of certain diseases. While automatic pattern recognizers cannot replace cardiologists in detecting heart problems, the time required for cardiologists to detect abnormalities may be cut significantly if such a recognizer is run before the cardiologist looks at the ECG. The cardiologist's attention can be drawn quickly to unclassifiable beats and to those beats whose amplitudes or rhythms are unusual. The recognizer can give an overall summary of the beats recognized and their regularity, as well as timing of the beats so the cardiologist has some pre-compiled statistics on the patient without a time-consuming study of the full ECG.

There have been several studies of automatic recognition of ECG data. Some have approached abnormality detection by studying the first difference of the waveform.

Others utilize the DFT of the waveform to detect periodic components. Neither approach has produced highly accurate results. For the DFT, the noise at 30 and 60 Hz tends to complicate detection of waveform components at those frequencies. Our approach was to use a set of neural network nodes, due to expectations of high accuracy and the implementation seemed simpler than for a complete neural network.

## BACKGROUND

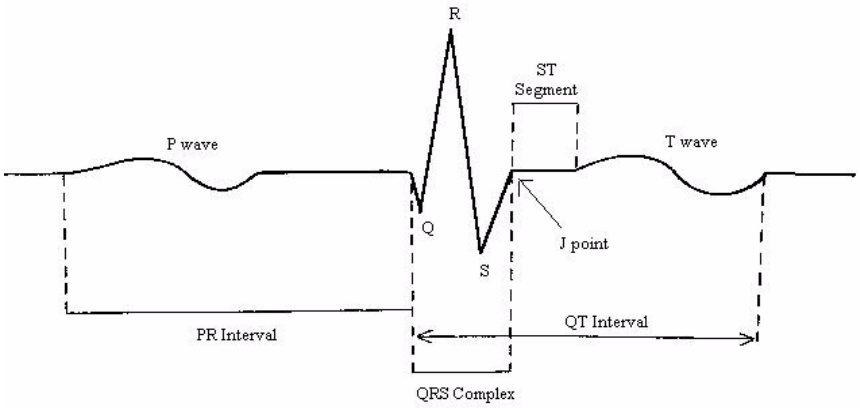
Normally an electrocardiogram is composed of 12 leads of data generated by attaching 12 separate electrodes to the patient. The electrocardiograph records the data and prints each lead out separately on a graph paper with a printed time axis for rhythm and heart rate determination. The data can be read electronically from the electrocardiograph's output, but it is in a proprietary format. For an in-depth discussion on electrocardiogram leads and use, see [2], [6]. For this project we are using 2 lead data selected by the Harvard-MIT Division of Health Sciences and Technology group, and gathered at Boston's Beth-Israel Hospital (now the Beth Israel Deaconess Medical Center). The 2 leads used in each record vary between different records. Leads used in the top signal include: V5 and MLII from a normal 12 lead configuration; leads used in the bottom signal include: V1, V2, V4, V5 and MLII. Patients are adults, both female and male, with known heart problems and symptomatic descriptions included in the record annotations.

According to the MIT website, http://ecg.mit.edu, the "analog outputs of the playback unit were filtered to limit analog-to-digital converter (ADC) saturation and for anti-aliasing, using a passband from 0.1 to 100 Hz relative to time, well beyond the lowest and highest frequencies recoverable from the recordings. The bandpass-filtered signals were digitized at 360 Hz per signal relative to real time using hardware constructed at the MIT Biomedical Engineering Center and at the BIH Biomedical Engineering Laboratory. The sampling frequency was chosen to facilitate implementations of 60 Hz (mains frequency) digital notch filters in arrhythmia detectors. Since the recorders were battery-powered, most of the 60 Hz noise present in the database arose during playback. In those records that were digitized at twice real time, this noise appears at 30 Hz (and multiples of 30 Hz) relative to real time."

Samples were digitized such that the intersignal sampling skew was on the order of a few microseconds. The ADCs "were unipolar, with 11-bit resolution over a 5 mV range. Sample values thus range from 0 to 2047 inclusive, with a value of 1024 corresponding to zero volts. [3]" All records are annotated for every beat. These annotations are included in a file separate from the data file. These annotation files can be read in with the data by using wave.exe which is available online and in the CD-ROM for the database. There is a program that allows users to strip out the time stamp in seconds, the absolute time stamp in samples, the annotations, and subannotations. We used this program in testing our accuracy and picking our testing samples.

In order to recognize patterns in the ECG leads, it is first necessary to locate the QRS complex, if it is present. The QRS complex is a waveform that appears in most normal and abnormal signals in an ECG. The elements of an ECG may include a preliminary P wave, a Q peak (negative amplitude), an R peak (the most prominent feature), an S peak (negative amplitude), and a T wave, as shown below in Figure 1[2]:

**FIGURE 1. ECG Waveform**




Once the QRS is determined to be present and it located (if it is present), pattern matching can be performed to determine the characteristics of the beat. Some characteristics that may be checked for include rhythm, duration, amplitude, maximum and minimum slopes. Abnormalities can be detected by comparing beats with a generalized format for normal and abnormal beats. Humans have different normal heartbeats and their abnormal beats will vary accordingly. In order to computationally characterize one person's abnormal heartbeats, the beats must be similar enough to other people's abnormal beats that a system can detect them and classify them properly.

**GOALS**

The goals are to locate QRS complexes in the data, to match the waveform corresponding to the QRS complex with a generalized waveform, and output a useful collection of statistics and labels so users can identify the waveforms on an ECG.

## *QRS Complex Detection*

**INTRODUCTION**

One of the most obvious characteristics of the QRS Complex is the large peak in R wave. A simple detection routine would just look for a periodic peak. However, because of the noise present in the aquisition of ECGs, this technique will return false waves. As a result, something more involved is required.

**ALGORITHM**

The detection algorithm that was used is based on a Pascal program written by W.A.H. Engelse and C. Zeelenberg, "A single scan algorithm for QRS-detection and feature extraction", Computers in Cardiology 6:37-42 (1979). [1]

The first sample is read and copied into variables that will be used to store the ten most recent samples. An FIR filter then variables that will be used to store the ten most recent samples. A threshold is set which is actually a slope criterion. This threshold is adjusted if more than two seconds have elapsed since a QRS was detected. If this crition is satisfied, a timer is set to 160 msec and the sign of the slope and the current time relative to the previous beat is saved.

Each time the filter output crosses the threshold, another slope is recorded and the program begins looking for a threshold crossing of the opposite sign, which must occur within a specified time. The maximum absolute value of the filter is recorded for eventual use in updating the threshold. Once a sufficient interval has elapsed following the last threshold crossing, if there were between 2 and 4 slopes, the program (apparently) found a QRS complex. If there were 5 or more slopes, the program records an artifact annotation, which designates an artificial beat. If only 1 slope was found, it is assumed to represent a baseline shift and no output is produced. [1]

## *Neural Network for Classification*

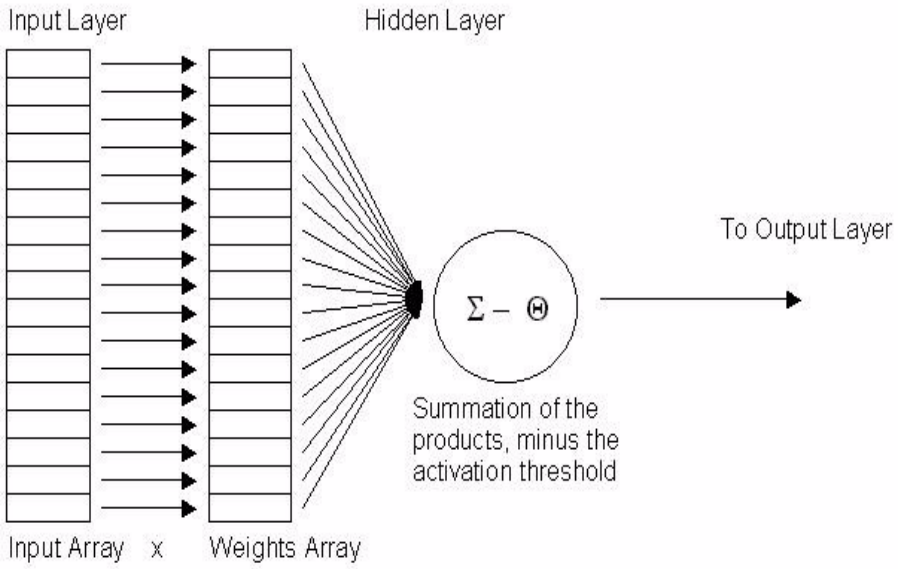
**INTRODUCTION**

The system primarily uses the neural network nodes for waveform classification. While other algorithms were considered, we decided that using a neural network would give us the best general functionality with other algorithms used secondarily for specific anomalies. When the secondary algorithms were abandoned for implementation issues the neural network became the sole classification algorithm. After reading into the capabilities and implementation of neural networks, and considering the waveforms from the database, we decided that we could achieve a simpler functionality by just using basic neural network nodes. The way we define a single neural network node is a single neuron in the hidden layer that accepts 300 inputs (300 samples centered about the QRS complex) and has one output (see figure 3.1 for clarification).

In basic operation the input array is dot multiplied with the weights of the hidden layer neuron. The activation threshold of the neuron is subtracted from the result of the dot product. If the dot product is greater than the activation threshold, theta, then a one is passed to the output layer, and if the dot product is less than the threshold a zero is passed to the output layer.

After the waveform being classified has been analyzed by all the nodes using the verify function, the program attempts to choose the best classification for the waveform. If all the nodes return zero for their activation, then the wave is determined to be unclassifiable. If only one node is active, then the wave is considered to be the type of the single active node. If multiple nodes are active, then the node one with the highest dot product result with the waveform is choosen to classify that wave.

**FIGURE 2. Node Diagram**




## NODE FUNCTIONS

All of the source code for the neural network nodes is contained in nnet.h and nnet.c. The nodes are created as structs that contain the values of theta (the activation threshold), the weights, the node's learning rate, and a text label to identify each node with a waveform. The three functions, which define the action of the nodes are initialize, learn, and verify. All three of these functions depend on the functions normalize and dotprod, which will be discussed later as support functions.

**Initialize.** The first step in training the nodes is initialization. The first training wave is used as the input to the initialization function, which sets the weights of the node to the (see below) normalized version of the input training wave. The activation threshold theta is set to 1. Because the learning rate is set to the number of waves used in the training set, setting the initial weights to the first waveform allow for equal representation by all the waves in the training set and gives faster learning than if random weights had been used.

**Learn.** After the node has been initialized, the rest of the training waves are sent as inputs to the learning function of the node. The incoming wave is normalized and dot multiplied with the weights of the node. If the product is less than the current value of theta then theta is set to the value of the product. Next, the weights are adjusted using the Widrow-Hoff algorithm. The new weight is the current weight plus the result of the adjustment value. The adjustment value is the current weight minus the corresponding input element, all divided by the learning rate. Once the weights have all been adjusted in this way, the weights vector is normalized (see below).

**Verify.** The verification function is used by the classification methods, though by itself it does not classify the waveform. Instead, the function tries to determine the similarity between the input waveform and the node's set of weight. The input waveform is nor-

malized and the dot product of the normalized wave and the weights is calculated. If the product is greater than the activation threshold of the node the node is considered active, if not it is considered inactive. Both the product and the result that indicates activity are returned by the function using the verReply struct defined in nnet.h.

**SUPPORT FUNCTIONS**

The functions normalize and dotprod are essential to the functionality of the neural network node functions. The code is contained in the files normalize.c, normalize.h, dotprod.h, and dotprod.c.

**Normalize.** The normalization function is applied to both incoming waveforms and the weights of the node. Normalization finds the mean of the array passed to it, and subtracts the mean from all values in the array, making the new mean of the array zero. Next, all the values of the array are squared and summed. The sum of the squares is then square rooted to create a normalization factor. The whole array is then divided by the normalization factor to finally create a normalized version of the input vector. This is done to bound the dot product of the normalized incoming waveforms and the normalized weights between negative one and positive one. On this scale one indicates a complete match, while negative one indicates that the two are additive inverses. The motivation to approach the input and weight values in the way comes from Albert Nigrin, "Neural Networks for Pattern Recognition" and it is also part of the Widrow-Hoff algorithm.

**Dotprod.** The dot product of two arrays is taken multiple times between learn and verify, but the function itself is very simple. A single loop adds the product of corresponding array values and keeps tract of the sum of the products. The function then returns the final sum of all the products.

**DESIGN DECISIONS**

While doing reading on neural network training and weight adjustment we did not come upon as many refrences to gradient search as opposed to Widrow-Hof. So, we implemented the simpler, thought less efecient, Widrow-Hoff algorithm. Also, since the presentation and demonstration we have been considering the suggestion of using batch processingof training inputs. However, we are unsure if this would effect the end result of the Widrow-Hoff algorithm or if it would only aid if using a gradient search.

## *System Level Implementation*

In terms of computation, the system did equal amounts of work on both the PC and the EVM. Since libraries existed for the PC from the MIT ECG website [1], the PC did the ECG specific functions while the EVM implemented the neural network.

## SIGNAL FLOW

**FIGURE 3. Signal Flow (Training)**

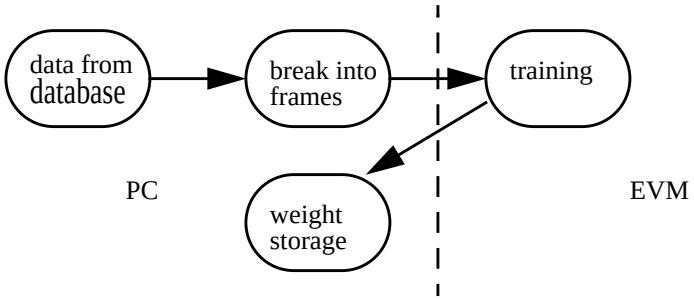
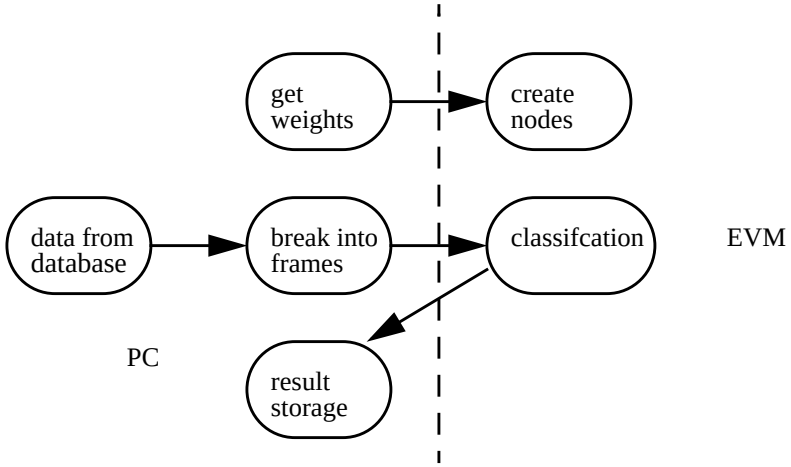



**FIGURE 4. Signal Flow (Classification)**




## WAVEFORM EXTRACTION

Before any processing could be done on the ECG records, a useable version of the data must be extracted. Since the data is stored in the database as 11-bit integers, simply reading in the file and using the results to generate the waveform was impossible. The DB library however, provided a host of functions which could not only extract the waveform from each record, but could also extract annotation information and signal information (such as sampling rate). The waveforms were extracted and stored as floats since the EVM had problems loading doubles.

## QRS EXTRACTION

Once each waveform was stored, the PC chunked the waveform into a set of QRS complexes and stored them in a seperate data structure which also contained a text string denoting the type of lead the signal was extracted. Modifications had to be made to the existing library functions provided by the MIT DB library. Since the algorithm was optimized for signals sampled at 250 Hz, changes needed to be made to center the QRS peak at the center of the waveform. Since the algorithm returned an approximate location we looked for the biggest value in a 10 sample range around the returned location. Each complex was then sent one-by-one over to the EVM for processing.

## NEURAL NET TRAINING

Before any type of classification could be done, training of the neural net was necessary. The training process is as follows. First 5 exemplar waveforms are extracted from the

database, for each type of anomaly that was to be classified. The PC sends over information conveying to the EVM that it was about to be trained. The waveforms are sent over and the weights for a node is generated, as specified in the previous section. Then the weights are sent back over to the PC and labeled with the type of waveform the weights were trained on.

This was done for each type of anomaly that was to be detected.

**NEURAL NET CLASSIFICATION**

Once all the weights for the neural network were gathered, the system was then ready for verification. The first 500 waveforms from each database entry were gathered and used for our test set. First, the PC sends the weights of each of the nodes over to the EVM which creates the appropriate nodes. Then the PC sends over a waveform which is sent into the neural network. Once the network makes a decision on the type of waveform it was presented, the result is sent over to the PC and saved to disk. This process is repeated for any number of waveforms in the record (this is a command line option).

## *Performance*

**MEMORY**

The largest source of memory useage lies in the storage of the waveforms and of the neural network node weights. Each waveform is stored at 300 floats and the weights of each node are also stored as 300 floats. Because of the unreliability of the external memory on the EVM, it was decided that we would store all the nodal weights and signals on-chip. The only limitations that resulted were the number of anomalies that could be detected. Since only one waveform was stored on-chip at any one time, the approximate memory usage was around 24kB. In the end, a total of 19 different nodes were used. Paging was not used since all variables were stored on-chip.

**SPEED**

The overall speed of our system was very good. Generation of the weights was only limited by how fast the PC could display the communcation signals (transfer requests, etc) to the screen. It took approximately 3 seconds to generate the weights for each node.

For verification, once again the limitation on speed was the display of communication signals to the screen. The QRS detection and segmenting routine took a few seconds to complete, but after that, the verification sequence was very quick.

**Profiling.** Computationally, the biggest use the DSP is the verification routine. This routine is called for every node for each waveform. As a result, the addition of more nodes causes the computation time to increase as a function of MN (M is the number of waveforms, N is the number of nodes). Therefore, this was the main focus of optimization.

**Optimization.** The verification function makes use of a normalization function which involves a series of multiplies and additions. In order to take advantage of the multiple

arithmetic units, this multiples (which were called in a for loop) were rolled out. This made a significate impact in the cycle numbers. Before optimization, the verification routine took approximately 85173.4 cycles to complete. The complete verification (from getting the wave from the CPU to the return of the result) took approximately $1.882x10^6$ cycles to run.

After the above optimizations the verification routine took approximately 75822 cycles to complete. While this may not seem like much, in the long run each complete verification now only takes 713405.6 cycles to complete.

## *Results*

**RECORD 113**

For record 113, which we trained for normal and atrial beats, we get this accuracy:

8 false negatives, all of normal beats id'd as unclassifiable
4 positive id of atrial premature beats
488 positive id of normal beats
98.4% accuracy

**RECORD 124**

For record 124, which we trained for: fusion, nodal premature, nodal escape, right bundle and premature ventricular beats, we get this accuracy:

1 false negative, atrial prem. Beat id as uncl.\
1 false negative, fusion beat id as unc
1 false negative, fusion beat id as nodal escape
7 false negatives, nodal prem. Id as nodal escape
266 false negatives, right bundle id as unc or other
202 positive id of right bundle branch block beat
22 positive id of nodal premature beats

44.8% accuracy

**RECORD 109**

For record 109, which we trained for left bundle branch block and premature ventricular beats, we get this accuracy:

2 false negative id of fusion beats as left bundle
17 false negative ids of left bundle as various
6 false negative ids of premature ventricular as left bundle
475 positive ids of left bundle branch block beats

95% accuracy

**RECORD 105**

For record 105, which we trained for: normal and premature ventricular beats, we get this accuracy

2 false negatives for Isolated QRS artifacts as normal
8 false negatives for normal as unclassifiable
13 false negatives for premature ventricular as normal
2 false negatives for premature ventricular as unclass..
1 positive id of Isolated QRS artifact as unclassifiable
474 positive id of normal beats

95% accuracy

**RECORD 221**

For record 221 which we did not train for, but which contains normal and premature ventricular signals, we get this accuracy:

165 false negatives on normals as various
80 false negatives for premature ventricular as various
240 positive id of normal as various normal types
15 positive ids of premature ventricular beats

51% accuracy

**RECORD 219**

For record 219 which we did not train for, but which contains normal and premature ventricular signals, we get this accuracy:

4 false negatives on atrial types
1 false negative on fusion beat
2 false negatives on missed beats
476 false negatives on normal beats
13 false negatives on premature ventricular beats
4 positive id of premature ventricular beats

.8% accuracy

**DISCUSSION**

The results for the records which we trained nodes on are, in general, far better than the results for the records which we did not train nodes on. This result is not surprising. The results for record 219 seem to be extremely bad but on further inspection, it became apparent that the normal waves include an inverted T-wave following the QRS complex and the Q and S peaks are extremely small in amplitude. The normal waveform thus does not correspond to (even remotely) any of the other normal waveforms in the database.

Similarly, for record 221 the ventricular waves would have been more easily detected if we had run the classification on the bottom lead, which we had not run it on.

In all cases, we only classified the first 500 beats of each record, so our results might have varied if we had performed the classification on the entire record. We chose 500 beats because this was 1/4 of the length of the record and it seemed like a good number at the time.

Given more time, we would have performed parallel analyses on the top and bottom leads. We would have then perfomed some comparisons to determine if the leads return the same classifications or different ones. In some records, the bottom leads shows more prominent characteristics for certain abnormalities than the top lead does. This would require more nodes for classification though, because the bottom lead waveforms do not look like the top lead waveforms. In some cases, the bottom lead is much more noisy than the top, and rotation affects the bottom leads far more often than the top leads. This is presumably due to the use of V leads for the bottom whereas the top lead is almost always MLII.

Additionally, we might have performed some simple threshold analyses to pick off maxima and minima in the data in order to locate the Q, R, and S peaks. Given these locations, we might have compared rhythms between different waveforms in the same record. Rhythm changes are important to note for ECG analysis because they are used to detect escape beats, prematurity in beats, general rhythm changes, drastic rhythm changes (which may be indicative of massive heart failure) and atrial premature beats of types which we cannot at present locate or classify.xs

## *Future Extensions/Current Problems*

Because of the bug with LDDW in the version of the DSP chip that we have, there are many things that we would like to have done. First, memory managment on the EVM was more than painful. There is a section of our code where the existence of a printf statement prevented any strange behavior by the EVM. Errors that were generated in the creation process include communication ghosts (the EVM mistakenly believed it was receiving signals from the PC) and resetting of memory. We would like to have been able to use more nodes and a more complicated neural net. Unfortunately, all the problems we had with the EVM used up the time that could have been used developing and researching a better net. Blame TI :)

## *References*

1. ECG Database Programmer’s Guide, http://ecg.mit.edu/dbpg/dbu_toc.htm

2. Elementary ECG (aka Electrocardiogram or EKG), http://circpc.epfl.ch/personal/schimmin/uni/ecglex/ekg.htm

3. Harvard-MIT Division of Health Sciences and Technology web site, http://ecg.mit.edu

4. MIT-BIH Arrhthmia Database Directory, http://www.physionet.org/physiobank/database/mitdbdir/mitdbdir.htm

5. Nigrin, Albert. *Neural Networks for Pattern Recognition*, MIT Press: Cambridge, MA, 1993.

6. D. J. Weatherall, J. G. G. Ledingham, and D. A. Warrell, eds., *Oxford textbook of Medicine*, Oxford University Press: Oxford, 1987, pp.13.21-34.

## *Appendix A: Training Waves*

**FIGURE 5. A Normal Waveform**

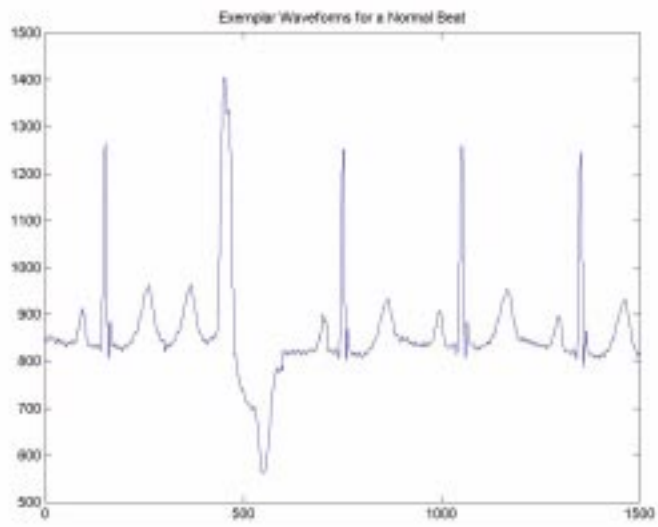


**FIGURE 6. A Fusion Beat**

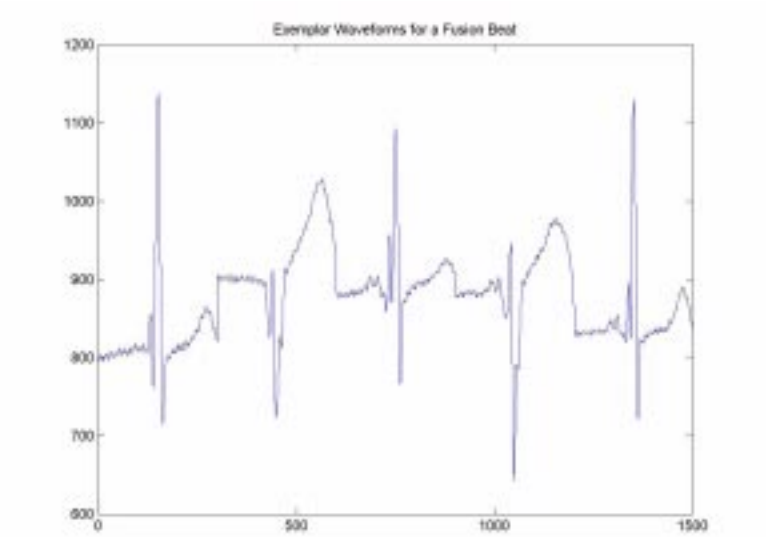



**FIGURE 7. A Nodal Escape Beat**

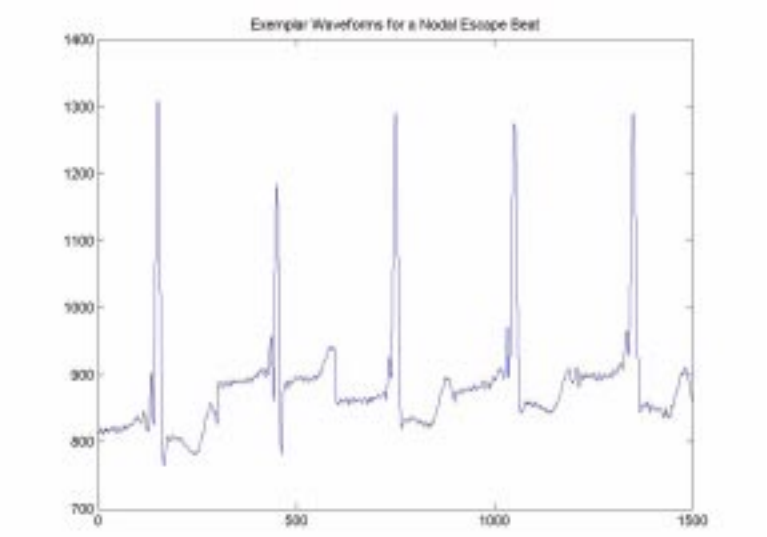


**FIGURE 8. A Nodal Premature Beat**

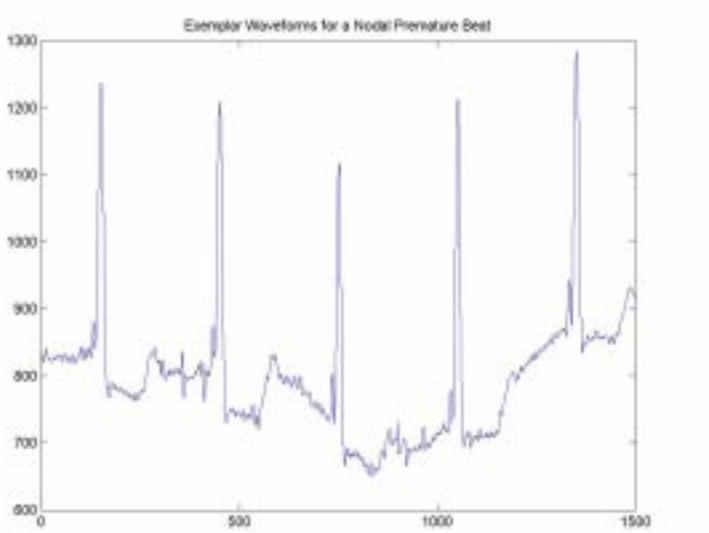



**FIGURE 9. A Paced Beat**