7.21 The z transform of a filter is:

$$H(z) = 1 - z^{-360}$$

The following sine wave is applied at the input: $x(t) = 100 \sin(2\pi 10t)$. The sampling rate is 720 samples/s. (a) What is the peak-to-peak output of the filter? (b) If a *unit step* input is applied, what will the output amplitude be after 361 samples? (c) Where could poles be placed to convert this to a bandpass filter with integer coefficients?

7.22 What is the phase delay (in milliseconds) through the following filter which operates at 200 samples/sec?

$$H(z) = \frac{1 - z^{-100}}{1 - z^{-2}}$$

7.23 A filter has 8 zeros located on the unit circle starting at dc and equally spaced at 45° increments. There are two poles located at $z = \pm j$. The sampling frequency is 360 samples/s. What is the gain of the filter?

8

Adaptive Filters

Steven Tang

This chapter discusses how to build adaptive digital filters to perform noise cancellation and signal extraction. Adaptive techniques are advantageous because they do not require a priori knowledge of the signal or noise characteristics as do fixed filters. Adaptive filters employ a method of learning through an estimated synthesis of a desired signal and error feedback to modify the filter parameters. Adaptive techniques have been used in filtering of 60-Hz line frequency noise from ECG signals, extracting fetal ECG signals, and enhancing P waves, as well as for removing other artifacts from the ECG signal. This chapter provides the basic principles of adaptive digital filtering and demonstrates some direct applications.

In digital signal processing applications, frequently a desired signal is corrupted by interfering noise. In fixed filter methods, the basic premise behind optimal filtering is that we must have knowledge of both the signal and noise characteristics. It is also generally assumed that the statistics of both sources are well behaved or wide-sense stationary. An adaptive filter learns the statistics of the input sources and tracks them if they vary slowly.

8.1 PRINCIPAL NOISE CANCELER MODEL

In biomedical signal processing, adaptive techniques are valuable for eliminating noise interference. Figure 8.1 shows a general model of an adaptive filter noise canceler. In the discrete time case, we can model the primary input as $s(nT) + n_0(nT)$. The noise is additive and considered uncorrelated with the signal source. A secondary reference input to the filter feeds a noise $n_1(nT)$ into the filter to produce output $\zeta(nT)$ that is a close estimate of $n_0(nT)$. The noise $n_1(nT)$ is correlated in an unknown way to $n_0(nT)$.

The output $\zeta(nT)$ is subtracted from the primary input to produce the system output $y(nT)$. This output is also the error $\varepsilon(nT)$ that is used to adjust the taps of the adaptive filter coefficients $\{w(1,\ldots,p)\}$.

$$y(nT) = s(nT) + n_0(nT) - \zeta(nT) \tag{8.1}$$

Squaring the output and making the (nT) implicit to simplify each term

$$y^2 = s^2 + (n_0 - \zeta)^2 + 2s(n_0 - \zeta) \tag{8.2}$$

Taking the expectation of both sides,

$$\begin{aligned} E[y^2] &= E[s^2] + E[(n_0 - \zeta)^2] + 2E[s(n_0 - \zeta)] \\ &= E[s^2] + E[(n_0 - \zeta)^2] \end{aligned} \tag{8.3}$$

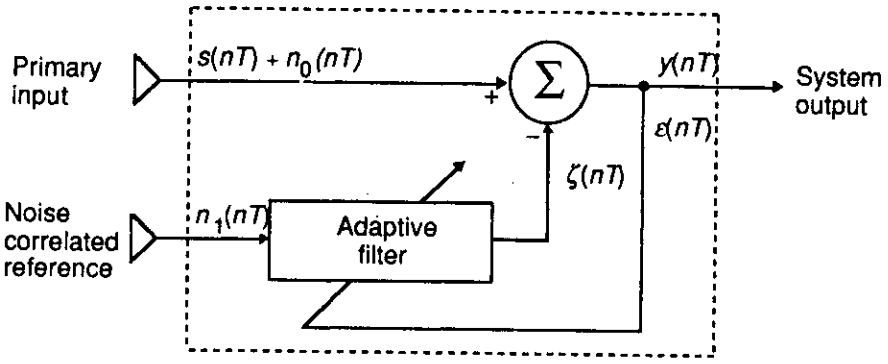


Figure 8.1 The structure of an adaptive filter noise canceler.

Since the signal power $E[s^2]$ is unaffected by adjustments to the filter

$$\min E[y^2] = E[s^2] + \min E[(n_0 - \zeta)^2] \tag{8.4}$$

When the system output power is minimized according to Eq. (8.4), the mean-squared error (MSE) of $(n_0 - \zeta)$ is minimum, and the filter has adaptively learned to synthesize the noise ($\zeta \approx n_0$). This approach of iteratively modifying the filter coefficients using the MSE is called the Least Mean Squared (LMS) algorithm.

8.2 60-HZ ADAPTIVE CANCELING USING A SINE WAVE MODEL

It is well documented that ECG amplifiers are corrupted by a sinusoidal 60-Hz line frequency noise (Huhta and Webster, 1973). As discussed in Chapter 5, a non-recursive band-reject notch filter can be implemented to reduce the power of noise at 60 Hz. The drawbacks to this design are that, while output noise power is reduced, such a filter (1) also removes the 60-Hz component of the signal, (2) has a very

slow rolloff that unnecessarily attenuates other frequency bands, and (3) becomes nonoptimal if either the amplitude or the frequency characteristics of the noise change. Adaptive transversal (tapped delay line) filters allow for elimination of noise while maintaining an optimal signal-to-noise ratio for nonstationary processes.

One simplified method for removal of 60-Hz noise is to model the reference source as a 60-Hz sine wave (Ahlstrom and Tompkins, 1985). The only adaptive parameter is the amplitude of the sine wave. Figure 8.2 shows three signals: $x(nT)$ is the input ECG signal corrupted with 60-Hz noise, $e(nT)$ is the estimation of the noise using a 60-Hz sine wave, and $y(nT)$ is the output of the filter.

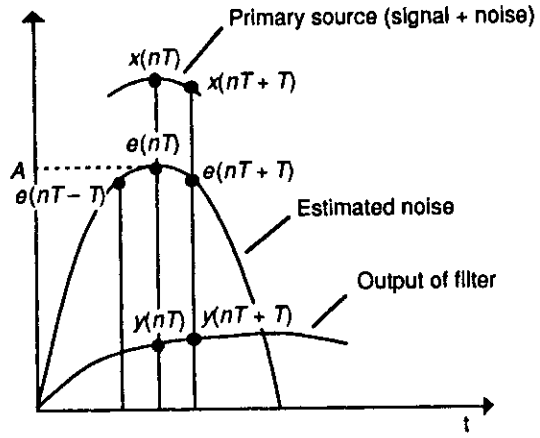


Figure 8.2 Sine wave model for 60-Hz adaptive cancellation.

The algorithm begins by estimating the noise as an assumed sinusoid with amplitude A and frequency ω

$$e(nT) = A\sin(\omega nT) \tag{8.5}$$

In this equation, we replace term (nT) by $(nT - T)$ to find an expression for the estimated signal one period in the past. This substitution gives

$$e(nT - T) = A\sin(\omega nT - \omega T) \tag{8.6}$$

Similarly, an expression that estimates the next point in the future is obtained by replacing (nT) by $(nT + T)$ in Eq. (8.5), giving

$$e(nT + T) = A\sin(\omega nT + \omega T) \tag{8.7}$$

We now recall a trigonometric identity

$$\sin(\alpha + \beta) = 2\sin(\alpha)\cos(\beta) - \sin(\alpha - \beta) \tag{8.8}$$

Now let

$$\alpha = \omega nT \quad and \quad \beta = \omega T \tag{8.9}$$

Expanding the estimate for the future estimate of Eq. (8.7) using Eqs. (8.8) and (8.9) gives

$$e(nT + T) = 2\underline{A\sin(\omega nT)}\cos(\omega T) - \underline{A\sin(\omega nT - \omega T)} \tag{8.10}$$

Note that the first underlined term is the same as the expression for $e(nT)$ in Eq. (8.5), and the second underlined term is the same as the expression for $e(nT - T)$ in Eq. (8.6). The term, $\cos(\omega T)$, is a constant determined by the frequency of the noise ω to be eliminated and by the sampling frequency, $f_s = 1/T$:

$$N = \cos(\omega T) = \cos\left(\frac{2\pi f}{f_s}\right) \tag{8.11}$$

Thus, Eq. (8.10) is rewritten, giving a relation for the future estimated point on a sampled sinusoidal noise waveform based on the values at the current and past sample times.

$$e(nT + T) = 2Ne(nT) - e(nT - T) \tag{8.12}$$

The output of the filter is the difference between the input and the estimated signals

$$y(nT + T) = x(nT + T) - e(nT + T) \tag{8.13}$$

Thus, if the input were only noise and the estimate were exactly tracking (i.e., modeling) it, the output would be zero. If an ECG were superimposed on the input noise, it would appear noise-free at the output.

The ECG signal is actually treated as a transient, while the filter iteratively attempts to change the "weight" or amplitude of the reference input to match the desired signal, the 60-Hz noise. The filter essentially learns the amount of noise that is present in the primary input and subtracts it out. In order to iteratively adjust the filter to adapt to changes in the noise signal, we need feedback to adjust the sinusoidal amplitude of the estimate signal for each sample period.

We define the difference function

$$f(nT + T) = [x(nT + T) - e(nT + T)] - [x(nT) - e(nT)] \tag{8.14}$$

In order to understand this function, consider Figure 8.3. Our original model of the

shown. Typically, however, there is a dc offset represented by V_{dc} in the input $x(nT)$ signal. From the figure

$$V_{dc}(nT+T) = x(nT+T) - e(nT+T) \tag{8.15}$$

and also

$$V_{dc}(nT) = x(nT) - e(nT) \tag{8.16}$$

Assuming that the dc level does not change significantly between samples, then

$$V_{dc}(nT+T) - V_{dc}(nT) = 0 \tag{8.17}$$

This subtraction of the terms representing the dc level in Eqs. (8.15) and (8.16) is the basis for the function in Eq. (8.14). It subtracts the dc while simultaneously comparing the input and estimated waveforms.

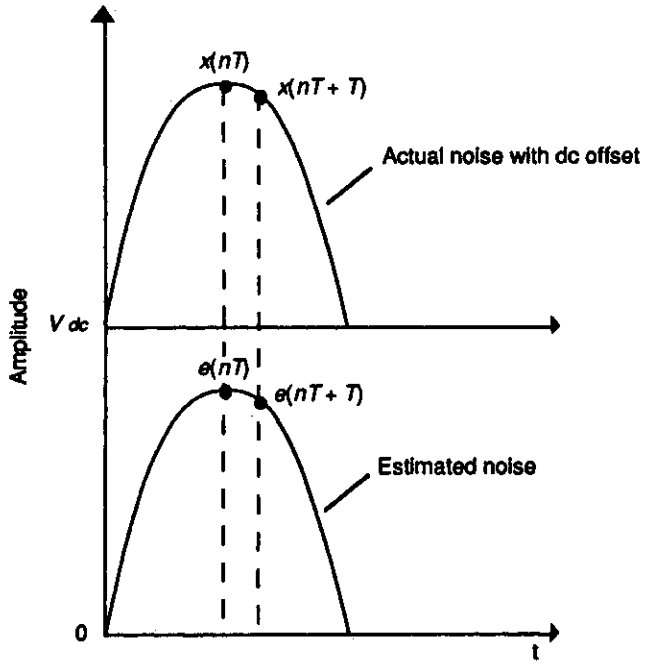


Figure 8.3 The actual noise waveform may include a dc offset that was included in the original model of the estimated signal.

We use $f(nT + T)$ to determine if the estimate $e(nT)$ was too large or too small. If $f(nT + T) = 0$, the estimate is correct and there is no need to adjust the future estimate, or

$$e(nT + T) = e(nT + T) \tag{8.18}$$

If $f(nT + T) > 0$, the estimate is low, and the estimate is adjusted upward by a small step size d

$$e(nT + T) = e(nT + T) + d \tag{8.19}$$

If $f(nT + T) < 0$, the estimate is high and the estimate is adjusted downward by a small step size d

$$e(nT + T) = e(nT + T) - d \tag{8.20}$$

The choice of d is empirically determined and depends on how quickly the filter needs to adapt to changes in the interfering noise. If d is large, then the filter quickly adjusts its coefficients after the onset of 60-Hz noise. However, if d is too large, the filter will not be able to converge exactly to the noise. This results in small oscillations in the estimated signal once the correct amplitude has been found. With a smaller d, the filter requires a longer learning period but provides more exact tracking of the noise for a smoother output. If the value of d is too large or too small, the filter will never converge to a proper noise estimate.

A typical value of d is less than the least significant bit value of the integers used to represent a signal. For example, if the full range of numbers from an 8-bit A/D converter is 0–255, then an optimal value for d might be 1/4.

Producing the estimated signal of Eq. (8.12) requires multiplication by a fraction N given in Eq. (8.11). For a sampling rate of 500 sps and 60-Hz power line noise

$$N = \cos\left(\frac{2\pi \times 60}{500}\right) = 0.7289686 \tag{8.21}$$

Such a multiplier requires floating-point arithmetic, which could considerably slow down the algorithm. In order to approximate such a multiplier, we might choose to use a summation of power-of-two fractions, which could be implemented with bit-shift operations and may be faster than floating-point multiplication in some hardware environments. In this case

$$N = \frac{1}{2} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \frac{1}{128} + \frac{1}{512} + \frac{1}{2048} = 0.72900 \tag{8.22}$$

8.3 OTHER APPLICATIONS OF ADAPTIVE FILTERING

Adaptive filtering is not only used to suppress 60-Hz interference but also for signal extraction and artifact cancellation. The adaptive technique is advantageous for generating a desired signal from one that is uncorrelated with it.

8.3.1 Maternal ECG in fetal ECG

Prenatal monitoring has made it possible to detect the heartbeat of the unborn child noninvasively. However, motion artifact and the maternal ECG make it very difficult to perceive the fetal ECG since it is a low-amplitude signal. Adaptive filtering has been used to eliminate the maternal ECG. Zhou et al. (1985) describe an algorithm that uses a windowed LMS routine to adapt the tap weights. The abdominal lead serves as the primary input and the chest lead from the mother is used as the reference noise input. Subtracting the best matched maternal ECG from the abdominal ECG which contains both the fetal and maternal ECGs produces a residual signal that is the fetal ECG.

8.3.2 Cardiogenic artifact

The area of electrical impedance pneumography has used adaptive filtering to solve the problem of cardiogenic artifact (ZCG). Such artifact can arise from electrical impedance changes due to blood flow and heart-volume changes. This can lead to a false interpretation of breathing. When monitoring for infant apnea, this might result in a failure to alarm. Sahakian and Kuo (1985) proposed using an adaptive LMS algorithm to extract the cardiogenic impedance component so as to achieve the best estimate of the respiratory impedance component. To model the cardiogenic artifact, they created a template synchronized to the QRS complex in an ECG that included sinus arrhythmia. Cardiogenic artifact is synchronous with but delayed from ventricular systole, so the ECG template can be used to derive and eliminate the ZCG.

8.3.3 Detection of ventricular fibrillation and tachycardia

Ventricular fibrillation detection has generally used frequency-domain techniques. This is computationally expensive and cannot always be implemented in real time. Hamilton and Tompkins (1987) describe a unique method of adaptive filtering to locate the poles corresponding to the frequency spectrum formants. By running a second-order IIR filter, the poles derived from the coefficients give a fairly good estimate of the first frequency peak.

The corresponding z transform of such a filter is

$$H(z) = \frac{1}{1 - b_1 z^{-1} - b_2 z^{-2}}$$

We can solve for the pole radius and angle by noting that, for all poles not on the real axis

$$b_1 = 2r\cos\theta \quad \text{and} \quad b_2 = -r^2$$

Using the fact that fibrillation produces a prominent peak in the 3–7 Hz frequency band, we can determine whether the poles fall in the "detection region" of the z plane. An LMS algorithm updates the coefficients of the filter. Figure 8.4 shows the z-plane pole-zero diagram of the adaptive filter. The shaded region indicates that the primary peak in the frequency spectrum of the ECG is in a "dangerous" area. The only weakness of the algorithm is that it creates false detections for rhythm rates greater than 100 bpm with frequent PVCs, atrial fibrillation, and severe motion artifact.

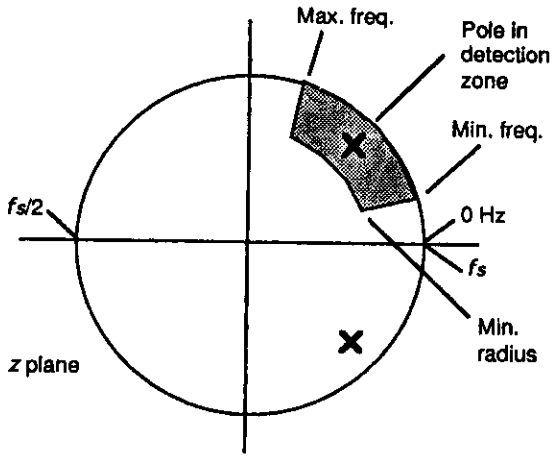


Figure 8.4 The z plane showing the complex-conjugate poles of the second-order adaptive filter.

8.4 LAB: 60-HZ ADAPTIVE FILTER

Load UW DigiScope, select `ad(V) Ops`, then `(A)daptive`. This module is a demonstration of a 60-Hz canceling adaptive filter as described in the text. You have control over the filter's step size d. This controls how quickly the filter *learns*

the amount of 60 Hz in the signal. By turning the 60-Hz noise off after the filter has adapted out the noise, you can observe that the filter must now unlearn the 60-Hz component. This routine always uses the same data file **adapting.dat** to which the 60-Hz noise is added.

8.5 REFERENCES

Ahlstrom, M. L., and Tompkins, W. J. 1985. Digital filters for real-time ECG signal processing using microprocessors. *IEEE Trans. Biomed. Eng.*, **BME-32**(9): 708–13.

Hamilton, P. S., and Tompkins, W. J. 1987. Detection of ventricular fibrillation and tachycardia by adaptive modeling. *Proc. Annu. Conf. Eng. Med. Bio. Soc.*, 1881–82.

Haykin, S. 1986. *Adaptive Filter Theory*. Englewood Cliffs, NJ: Prentice Hall.

Huhta, J. C., and Webster, J. G. 1973. 60-Hz interference in electrocardiography. *IEEE Trans. Biomed. Eng.*, **BME-20**(2): 91-101.

Mortara, D. W. 1977. Digital filters for ECG signals. *Computers in Cardiology*, 511–14.

Sahakian, A. V., and Kuo, K. H. 1985. Canceling the cardiogenic artifact in impedance pneumography. *Proc. Annu. Conf. Eng. Med. Bio. Soc.*, 855–59.

Sheng, Z. and Thakor, N. V. 1987. P-wave detection by an adaptive QRS-T cancellation technique. *Computers in Cardiology*, 249–52.

Widrow, B., Glover, J. R., John, M., Kaunitz, J., Charles, S. J., Hearn, R. H., Zeidler, J. R., Dong, E., and Goodlin, R. C. 1975. Adaptive noise canceling: principles and applications. *Proc. IEEE*, **63**(12): 1692–1716.

Zhou, L., Wei, D., and Sun, L. 1985. Fetal ECG processing by adaptive noise cancellation. *Proc. Annu. Conf. Eng. Med. Bio. Soc.*, 834–37.

8.6 STUDY QUESTIONS

8.1 What are the main advantages of adaptive filters over fixed filters?
8.2 Explain the criterion that is used to construct a Wiener filter.
8.3 Why is the error residual of a Wiener filter normal to the output?
8.4 Design an adaptive filter using the method of steepest-descent.
8.5 Design an adaptive filter using the LMS algorithm.
8.6 Why are bounds necessary on the step size of the steepest-descent and LMS algorithms?
8.7 What are the costs and benefits of using different step sizes in the 60-Hz sine wave algorithm?
8.8 Explain how the 60-Hz sine wave algorithm adapts to the phase of the noise.
8.9 The adaptive 60-Hz filter calculates a function

$$f(nT + T) = [x(nT + T) - e(nT + T)] - [x(nT) - e(nT)]$$

If this function is less than zero, how does the algorithm adjust the future estimate, $e(nT + T)$?

3.10 The adaptive 60-Hz filter uses the following equation to estimate the noise:

$$e(nT + T) = 2Ne(nT) - e(nT - T)$$

If the future estimate is found to be too high, what adjustment is made to (a) $e(nT - T)$, (b) $e(nT + T)$. (c) Write the equation for N and explain the terms of the equation.

3.11 The adaptive 60-Hz filter calculates the function

$$f(nT + T) = [x(nT + T) - e(nT + T)] - [x(nT) - e(nT)]$$

It adjusts the future estimate $e(nT + T)$ based on whether this function is greater than, less than, or equal to zero. Use a drawing and explain why the function could not be simplified to

$$f(nT + T) = x(nT + T) - e(nT + T)$$

9

Signal Averaging

Pradeep Tagare

Linear digital filters like those discussed in previous chapters perform very wel when the spectra of the signal and noise do not significantly overlap. For example a low-pass filter with a cutoff frequency of 100 Hz generally works well for attenuating noise frequencies greater than 100 Hz in ECG signals. However, if high level noise frequencies were to span the frequency range from 50–100 Hz, attempting to remove them using a 50–Hz low-pass filter would attenuate some of the components of the ECG signal as well as the noise. High-amplitude noise corruption within the frequency band of the signal may completely obscure the signal Thus, conventional filtering schemes fail when the signal and noise frequency spectra significantly overlap. Signal averaging is a digital technique for separating a repetitive signal from noise without introducing signal distortion (Tompkins and Webster, 1981). This chapter describes the technique of signal averaging for increasing the signal-to-noise ratio and discusses several applications.

9.1 BASICS OF SIGNAL AVERAGING

Figure 9.1(a) shows the spectrum of a signal that is corrupted by noise. In this case, the noise bandwidth is completely separated from the signal bandwidth, so the noise can easily be discarded by applying a linear low-pass filter. On the other hand, the noise bandwidth in Figure 9.1(b) overlaps the signal bandwidth, and the noise amplitude is larger than the signal. For this situation, a low-pass filter would need to discard some of the signal energy in order to remove the noise, thereby distorting the signal.

One predominant application area of signal averaging is in electroencephalography. The EEG recorded from scalp electrodes is difficult to interpret in part because it consists of a summation of the activity of the billions of brain cells. It is impossible to deduce much about the activity of the visual or auditory parts of the brain from the EEG. However, if we stimulate a part of the brain with a flash of light or an acoustical click, an evoked response occurs in the region of the brain

that processes information for the sensory system being stimulated. By summing the signals that are evoked immediately following many stimuli and dividing by the total number of stimuli, we obtain an averaged evoked response. This signal can reveal a great deal about the performance of a sensory system.

Signal averaging sums a set of time epochs of the signal together with the superimposed random noise. If the time epochs are properly aligned, the signal waveforms directly sum together. On the other hand, the uncorrelated noise averages out in time. Thus, the signal-to-noise ratio (SNR) is improved.

Signal averaging is based on the following characteristics of the signal and the noise:

1. The signal waveform must be repetitive (although it does not have to be periodic). This means that the signal must occur more than once but not necessarily at regular intervals.
2. The noise must be random and uncorrelated with the signal. In this application, random means that the noise is not periodic and that it can only be described statistically (e.g., by its mean and variance).
3. The temporal position of each signal waveform must be accurately known.

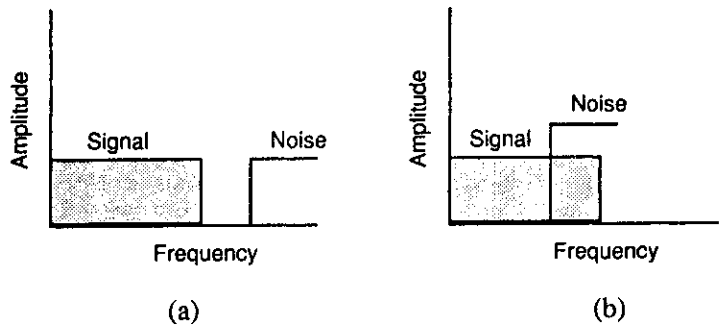


Figure 9.1 Signal and noise spectra. (a) The signal and noise bands do not overlap, so a conventional low-pass filter can be used to retain the signal and discard the noise. (b) Since the signal and noise spectra overlap, conventional filters cannot be used to discard the noise frequencies without discarding some signal energy. Signal averaging may be useful in this case.

It is the random nature of noise that makes signal averaging useful. Each time epoch (or sweep) is intentionally aligned with the previous epochs so that the digitized samples from the new epoch are added to the corresponding samples from the previous epochs. Thus the time-aligned repetitive signals S in each epoch are added directly together so that after four epochs, the signal amplitude is four times larger than for one epoch ($4S$). If the noise is random and has a mean of zero and an average rms value N, the rms value after four epochs is the square root of the sum of squares (i.e., $(4N^2)^{1/2}$ or $2N$). In general after n repetitions the signal amplitude is

mS and the noise amplitude is $(m)^{1/2}N$. Thus, the SNR improves as the ratio of m to $m^{1/2}$ (i.e., $m^{1/2}$). For example, averaging 100 repetitions of a signal improves the SNR by a factor of 10. This can be proven mathematically as follows.

The input waveform $f(t)$ has a signal portion $S(t)$ and a noise portion $N(t)$. Then

$$f(t) = S(t) + N(t) \tag{9.1}$$

Let $f(t)$ be sampled every T seconds. The value of any sample point in the time epoch ($i = 1, 2, \ldots, n$) is the sum of the noise component and the signal component.

$$f(iT) = S(iT) + N(iT) \tag{9.2}$$

Each sample point is stored in memory. The value stored in memory location i after m repetitions is

$$\sum_{k=1}^{m} f(iT) = \sum_{k=1}^{m} S(iT) + \sum_{k=1}^{m} N(iT) \qquad \text{for } i = 1, 2, \ldots, n \tag{9.3}$$

The signal component for sample point i is the same at each repetition if the signal is stable and the sweeps are aligned together perfectly. Then

$$\sum_{k=1}^{m} S(iT) = mS(iT) \tag{9.4}$$

The assumptions for this development are that the signal and noise are uncorrelated and that the noise is random with a mean of zero. After many repetitions, $N(iT)$ has an rms value of σ_n.

$$\sum_{k=1}^{m} N(iT) = \sqrt{m\sigma_n^2} = \sqrt{m}\,\sigma_n \tag{9.5}$$

Taking the ratio of Eqs. (9.4) and (9.5) gives the SNR after m repetitions as

$$\text{SNR}_m = \frac{mS(iT)}{\sqrt{m}\,\sigma_n} = \sqrt{m}\ \text{SNR} \tag{9.6}$$

Thus, signal averaging improves the SNR by a factor of $\sqrt{m}$. Figure 9.2 is a graph illustrating the results of Eq. (9.6)

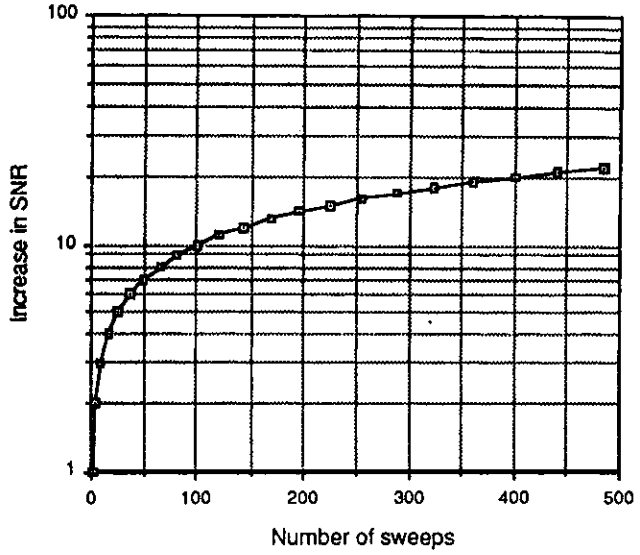


Figure 9.2 Increase in SNR as a function of the number of sweeps averaged.

Figure 9.3 illustrates the problem of signal averaging. The top trace is the ECG of the middle trace after being corrupted by random noise. Since the noise is broadband, there is no way to completely remove it with a traditional linear filter without also removing some of the ECG frequency components, thereby distorting the ECG. Signal averaging of this noisy signal requires a way to time align each of the QRS complexes with the others. By analyzing a heavily filtered version of the waveform, it is possible to locate the peaks of the QRS complexes and use them for time alignment. The lower trace shows these timing references (fiducial points) that are required for signal processing.

Figure 9.4 shows how the QRS complexes, centered on the fiducial points, are assembled and summed to produce the averaged signal. The time-aligned QRS complexes sum directly while the noise averages out to zero. The fiducial marks may also be located before or after the signal to be averaged, as long as they have accurate temporal relationships to the signals.

One research area in electrocardiography is the study of late potentials that require an ECG amplifier with a bandwidth of 500 Hz. These small, high-frequency signals of possible clinical significance occur after the QRS complex in body surface ECGs of abnormals. These signals are so small compared to the other waveforms in the ECG that they are hidden in the noise and are not observable without signal averaging. In this application, the fiducial points are derived from the QRS complexes, and the averaging region is the time following each QRS complex.

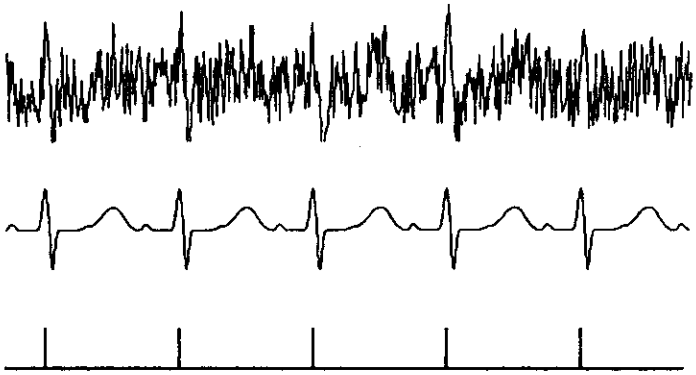


Figure 9.3 The top trace is the ECG of the center trace corrupted with random noise. The bottom trace provides fiducial marks that show the locations of the QRS peaks in the signal.

Figure 9.4 Summing the time-aligned signal epochs corrupted with random noise such as those in (a), (b), and (c), which were extracted from Figure 9.3, improves the signal-to-noise ratio. The result of averaging 100 of these ECG time epochs to improve the SNR by 10 is in (d).

9.2 SIGNAL AVERAGING AS A DIGITAL FILTER

Signal averaging is a kind of digital filtering process. The Fourier transform of the transfer function of an averager is composed of a series of discrete frequency components. Figure 9.5 shows how each of these components has the same spectral characteristics and amplitudes. Because of the appearance of its amplitude response, this type of filter is called a comb filter.

The width of each tooth decreases as the number of sweep repetitions increases. The desired signal has a frequency spectrum composed of discrete frequency components, a fundamental and harmonics. Noise, on the other hand, has a continuous distribution. As the bandwidth of each of the teeth of the comb decreases, this filter more selectively passes the fundamental and harmonics of the signal while rejecting the random noise frequencies that fall between the comb teeth. The signal averager, therefore, passes the signal while rejecting the noise.

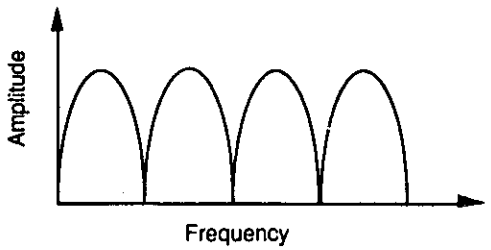


Figure 9.5 Fourier transform of a signal averager. As the number of sweeps increase, the width of each tooth of the comb decreases.

9.3 A TYPICAL AVERAGER

Figure 9.6 shows the block diagram of a typical averager. To average a signal such as the cortical response to an auditory stimulus, we stimulate the system (in this case, a human subject) with an auditory click to the stimulus input. Simultaneously, we provide a trigger derived from the stimulus that enables the summation of the sampled data (in this case, the EEG evoked by the stimulus) with the previous responses (time epochs or sweeps) stored in the buffer. When the averager receives the trigger pulse, it samples the EEG waveform at the selected rate, digitizes the signal, and sums the samples with the contents of a memory location corresponding to that sample interval (in the buffer). The process continues, stepping through the memory addresses until all addresses have been sampled. The sweep is terminated at this point. A new sweep begins with the next trigger and the cycle repeats until

the desired number of sweeps have been averaged. The result of the averaging process is stored in the buffer which can then be displayed on a CRT as the averaged evoked response.

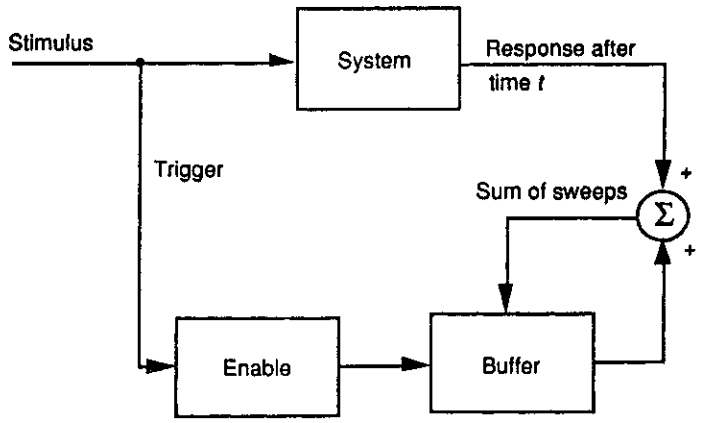


Figure 9.6 Block diagram of a typical signal averager.

9.4 SOFTWARE FOR SIGNAL AVERAGING

Figure 9.7 shows the flowchart of a program for averaging an ECG signal such as the one in Figure 9.3. The program uses a QRS detection algorithm to find a fiducial point at the peak of each QRS complex. Each time a QRS is detected, 128 new sample points are added to a buffer—64 points before and 64 points after the fiducial point.

9.5 LIMITATIONS OF SIGNAL AVERAGING

An important assumption made in signal averaging theory is that the noise is Gaussian. This assumption is not usually completely valid for biomedical signals. Also, if the noise distribution is related to the signal, misleading results can occur. If the fiducial point is derived from the signal itself, care must be taken to ensure that noise is not influencing the temporal location of the fiducial point. Otherwise, slight misalignment of each of the signal waveforms will lead to a low-pass filtering effect in the final result.

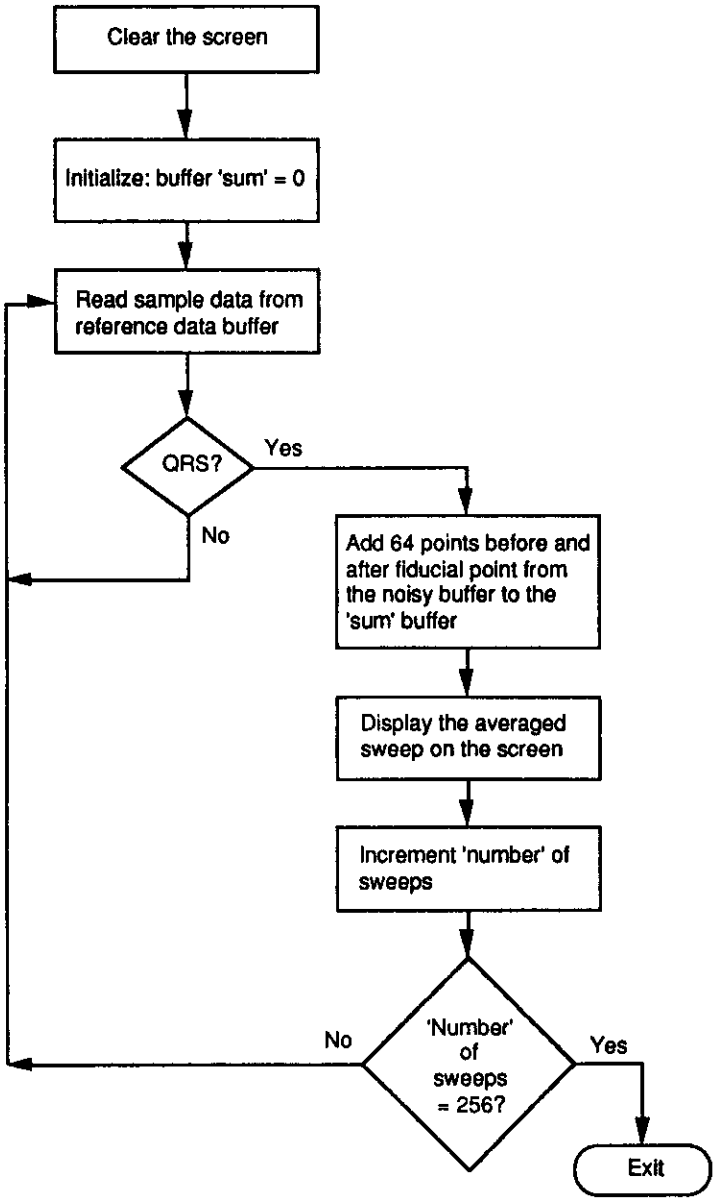


Figure 9.7 Flowchart of the signal averaging program.

9.6 LAB: ECG SIGNAL AVERAGING

Load UW DigiScope, select **ad(v) Ops**, then **A(v)erage**. This module is a demonstration of the use of signal averaging. The source file is always **average.dat** to which random, gaussian-distributed noise is added. A clean version of the data is preserved to use as a trigger signal. The averaged version of the data, displayed in the output channel, will build in size as successive waveforms are added. A summation of the accumulated signal traces is displayed, not the average. Thus you need to scale down the amplitude of the resultant signal in order to see the true average at the same amplitude scale factor as the original signal. Scaling of the output channel can be controlled as the traces are acquired. The down arrow key divides the amplitude by two each time it is struck. For example, while adding 16 heartbeats, you would need to strike the down arrow four times to divide the output by 16 so as to obtain the proper amplitude scale.

9.7 REFERENCES

Tompkins, W. J. and Webster, J. G. (eds.) 1981. *Design of Microcomputer-based Medical Instrumentation*. Englewood Cliffs, NJ: Prentice Hall.

9.8 STUDY QUESTIONS

9.1 Under what noise conditions will signal averaging fail to improve the SNR?

9.2 In a signal averaging application, the amplitude of uncorrelated noise is initially 16 times as large as the signal amplitude. How many sweeps must be averaged to give a resulting signal-to-noise ratio of 4:1?

9.3 After signal averaging 4096 EEG evoked responses, the signal-to-noise ratio is 4. Assuming that the EEG and noise sources are uncorrelated, what was the SNR before averaging?

9.4 In a signal averaging application, the noise amplitude is initially 4 times as large as the signal amplitude. How many sweeps must be averaged to give a resulting signal-to-noise ratio of 4:1?

9.5 In a signal averaging application, the signal caused by a stimulus and the noise are slightly correlated. The frequency spectra of the signal and noise overlap. Averaging 100 responses will improve the signal-to-noise ratio by what factor?

10

Data Reduction Techniques

Kok-Fung Lai

A typical computerized medical signal processing system acquires a large amount of data that is difficult to store and transmit. We need a way to reduce the data storage space while preserving the significant clinical content for signal reconstruction. In some applications, the process of reduction and reconstruction requires real-time performance (Jalaleddine et al., 1988).

A data reduction algorithm seeks to minimize the number of code bits stored by reducing the redundancy present in the original signal. We obtain the *reduction ratio* by dividing the number of bits of the original signal by the number saved in the compressed signal. We generally desire a high reduction ratio but caution against using this parameter as the sole basis of comparison among data reduction algorithms. Factors such as bandwidth, sampling frequency, and precision of the original data generally have considerable effect on the reduction ratio (Jalaleddine et al., 1990).

A data reduction algorithm must also represent the data with acceptable fidelity. In biomedical data reduction, we usually determine the clinical acceptability of the reconstructed signal through visual inspection. We may also measure the residual, that is, the difference between the reconstructed signal and the original signal. Such a numerical measure is the percent root-mean-square difference, PRD, given by

$$\mathrm{PRD} = \left\{ \frac{\sum_{i=1}^{n} [x_{org}(i) - x_{rec}(i)]^2}{\sum_{i=1}^{n} [x_{org}(i)]^2} \right\}^{\frac{1}{2}} \times 100\ \% \qquad (10.1)$$

where n is the number of samples and x_{org} and x_{rec} are samples of the original and reconstructed data sequences.

A *lossless* data reduction algorithm produces zero residual, and the reconstructed signal exactly replicates the original signal. However, clinically acceptable quality is neither guaranteed by a low nonzero residual nor ruled out by a high numerical

residual (Moody et al., 1988). For example, a data reduction algorithm for an ECG recording may eliminate small-amplitude baseline drift. In this case, the residual contains negligible clinical information. The reconstructed ECG signal can thus be quite clinically acceptable despite a high residual.

In this chapter we discuss two classes of data reduction techniques for the ECG. The first class, significant-point-extraction, includes the turning point (TP) algorithm, AZTEC (Amplitude Zone Time Epoch Coding), and the Fan algorithm. These techniques generally retain samples that contain important information about the signal and discard the rest. Since they produce nonzero residuals, they are *lossy* algorithms. In the second class of techniques based on Huffman coding, variable-length code words are assigned to a given quantized data sequence according to frequency of occurrence. A predictive algorithm is normally used together with Huffman coding to further reduce data redundancy by examining a successive number of neighboring samples.

10.1 TURNING POINT ALGORITHM

The original motivation for the turning point (TP) algorithm was to reduce the sampling frequency of an ECG signal from 200 to 100 samples/s (Mueller, 1978). The algorithm developed from the observation that, except for QRS complexes with large amplitudes and slopes, a sampling rate of 100 samples/s is adequate.

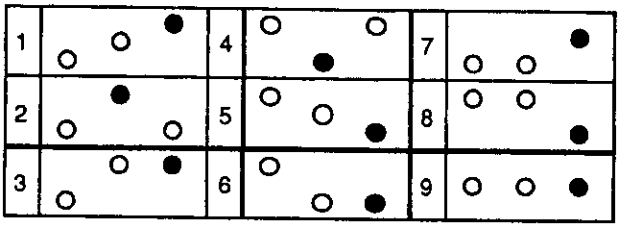
TP is based on the concept that ECG signals are normally oversampled at four or five times faster than the highest frequency present. For example, an ECG used in monitoring may have a bandwidth of 50 Hz and be sampled at 200 sps in order to easily visualize the higher-frequency attributes of the QRS complex. Sampling theory tells us that we can sample such a signal at 100 sps. TP provides a way to reduce the effective sampling rate by half to 100 sps by selectively saving important signal points (i.e., the peaks and valleys or turning points).

The algorithm processes three data points at a time. It stores the first sample point and assigns it as the reference point X_0. The next two consecutive points become X_1 and X_2. The algorithm retains either X_1 or X_2, depending on which point preserves the turning point (i.e., slope change) of the original signal.

Figure 10.1(a) shows all the possible configurations of three consecutive sample points. In each frame, the solid point preserves the slope of the original three points. The algorithm saves this point and makes it the reference point X_0 for the next iteration. It then samples the next two points, assigns them to X_1 and X_2, and repeats the process.

We use a simple mathematical criterion to determine the saved point. First consider a $sign(x)$ operation

$$sign(x) = \begin{cases} 0 & x = 0 \\ +1 & x > 0 \\ -1 & x < 0 \end{cases} \qquad (10.2)$$



(a)

Pattern	$s_1 = sign(X_1 - X_0)$	$s_2 = sign(X_2 - X_1)$	NOT(s_1) OR ($s_1 + s_2$)	Saved sample
1	+1	+1	1	X_2
2	+1	−1	0	X_1
3	+1	0	1	X_2
4	−1	+1	0	X_1
5	−1	−1	1	X_2
6	−1	0	1	X_2
7	0	+1	1	X_2
8	0	−1	1	X_2
9	0	0	1	X_2

(b)

Figure 10.1 Turning point (TP) algorithm. (a) All possible 3-point configurations. Each frame includes the sequence of three points X_0, X_1, and X_2. The solid points are saved. (b) Mathematical criterion used to determine saved point.

We then obtain $s_1 = sign(X_1 - X_0)$ and $s_2 = sign(X_2 - X_1)$, where $(X_1 - X_0)$ and $(X_2 - X_1)$ are the slopes of the two pairs of consecutive points. If a slope is zero, this operator produces a zero result. For positive or negative slopes, it yields +1 or −1 respectively. A turning point occurs only when a slope changes from positive to negative or vice versa.

We use the logical Boolean operators, NOT and OR, as implemented in the C language to make the final judgment of when a turning point occurs. In the C language, NOT(c) = 1 if $c = 0$; otherwise NOT(c) = 0. Also logical OR means that (a OR b) = 0 only if a and b are both 0. Thus, we retain X_1 only if {NOT(s_1) OR ($s_1 + s_2$)} is zero, and save X_2 otherwise. In this expression, ($s_1 + s_2$) is the arithmetic sum of the signs produced by the *sign* function. The final effect of this processing is a Boolean decision whether to save X_1 or X_2. Point X_1 is saved only when the slope changes from positive to negative or vice versa. This computation

could be easily done arithmetically, but the Boolean operation is computationally much faster.

Figure 10.2 shows the implementation of the TP algorithm in the C language. Figure 10.3 is an example of applying the TP algorithm to a synthesized ECG signal.

```
#define sign(x)  ( (x) ? ( (x > 0) ? 1 : -1 ) : 0 )
short *org, *tp ;          /* original and tp data */
short x0, x1, x2 ;         /* data points */
short s1, s2 ;                    /* signs */

x0 = *tp++ = *org++ ;      /* save the first sample */
while(there_is_sample) {

        x1 = *org++ ;
        x2 = *org++ ;
        s1 = sign(x1-x0) ;
        s2 = sign(x2-x1) ;
        *tp++ = x0 = ( !s1 || (s1+s2) ) ? x2 : x1 ;
}
```

Figure 10.2 C-language fragment showing TP algorithm implementation.

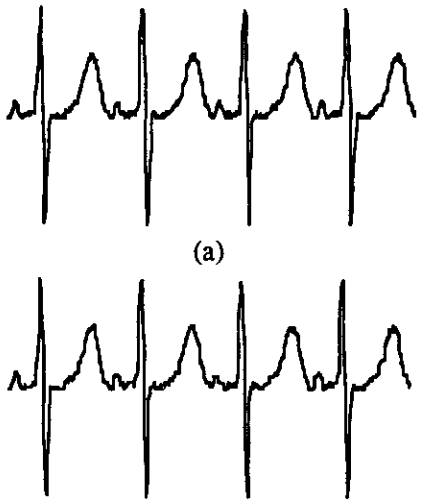


Figure 10.3 An example of the application of the TP algorithm. (a) Original waveform generated by the UW DigiScope `Genwave` function (see Appendix D). (b) Reconstructed signal after one application of the TP algorithm. Reduction ratio is 512:256, PRD = 7.78%.

The TP algorithm is simple and fast, producing a fixed reduction ratio of 2:1. After selectively discarding exactly half the sampled data, we can restore the original resolution by interpolating between pairs of saved data points.

A second application of the algorithm to the already reduced data increases the reduction ratio to 4:1. Using data acquired at a 200-sps rate, this produces compressed data with a 50-sps effective sampling rate. If the bandwidth of the acquired ECG is 50 Hz, this approach violates sampling theory since the effective sampling rate is less than twice the highest frequency present in the signal. The resulting reconstructed signal typically has a widened QRS complex and sharp edges that reduce its clinical acceptability. Another disadvantage of this algorithm is that the saved points do not represent equally spaced time intervals. This introduces short-term time distortion. However, this localized distortion is not visible when the reconstructed signal is viewed on the standard clinical monitors and paper recorders.

10.2 AZTEC ALGORITHM

Originally developed to preprocess ECGs for rhythm analysis, the AZTEC (Amplitude Zone Time Epoch Coding) data reduction algorithm decomposes raw ECG sample points into plateaus and slopes (Cox et al., 1968). It provides a sequence of line segments that form a piecewise-linear approximation to the ECG.

10.2.1 Data reduction

Figure 10.4 shows the complete flowchart for the AZTEC algorithm using C-language notation. The algorithm consists of two parts—line detection and line processing.

Figure 10.4(a) shows the line detection operation which makes use of zero-order interpolation (ZOI) to produce horizontal lines. Two variables V_{mx} and V_{mn} always reflect the highest and lowest elevations of the current line. Variable *LineLen* keeps track of the number of samples examined. We store a plateau if either the difference between V_{mxi} and V_{mni} is greater than a predetermined threshold V_{th} or if *LineLen* is greater than 50. The stored values are the length (*LineLen* – 1) and the average amplitude of the plateau $(V_{mx} + V_{mn})/2$.

Figure 10.4(b) shows the line processing algorithm which either produces a plateau or a slope depending on the value of the variable *LineMode*. We initialize *LineMode* to _PLATEAU in order to begin by producing a plateau. The production of an AZTEC slope begins when the number of samples needed to form a plateau is less than three. Setting *LineMode* to _SLOPE indicates that we have entered slope production mode. We then determine the direction or *sign* of the current slope by subtracting the previous line amplitude V_1 from the current amplitude V_{si}. We also reset the length of the slope T_{si}. The variable V_{si} records the current line amplitude so that any change in the direction of the slope can be tracked. Note that

V_{mxl} and V_{mnl} are always updated to the latest sample before line detection begins. This forces ZOI to begin from the value of the latest sample.

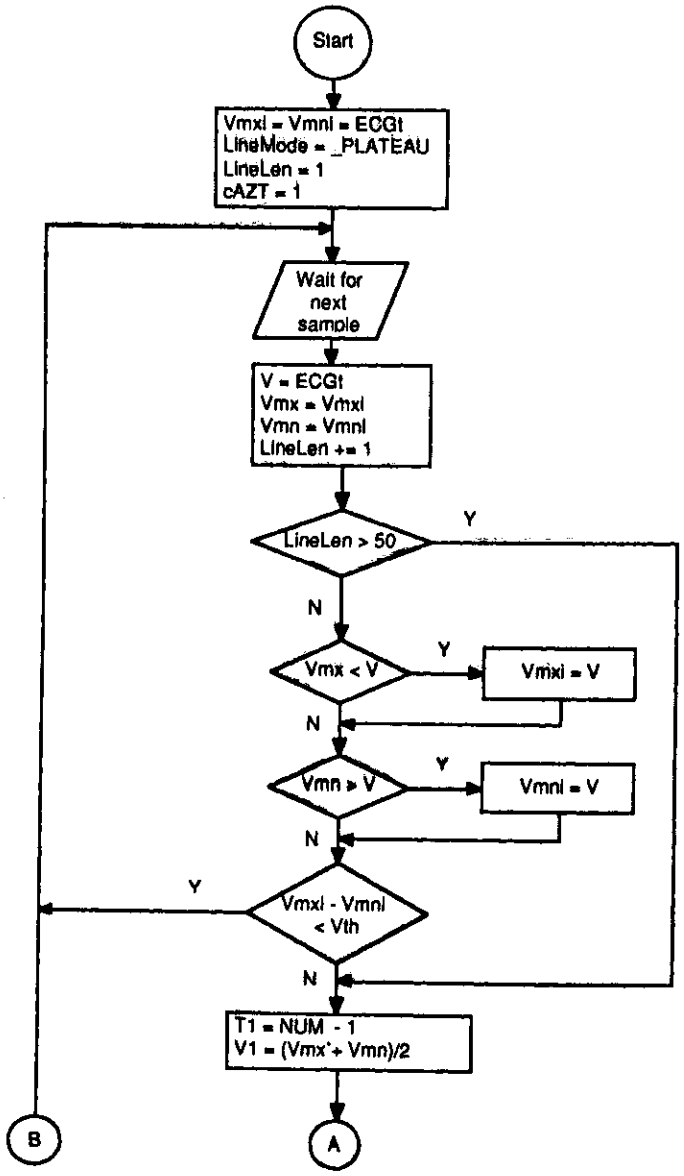


Figure 10.4(a) Flowchart for the line detection operation of the AZTEC algorithm.

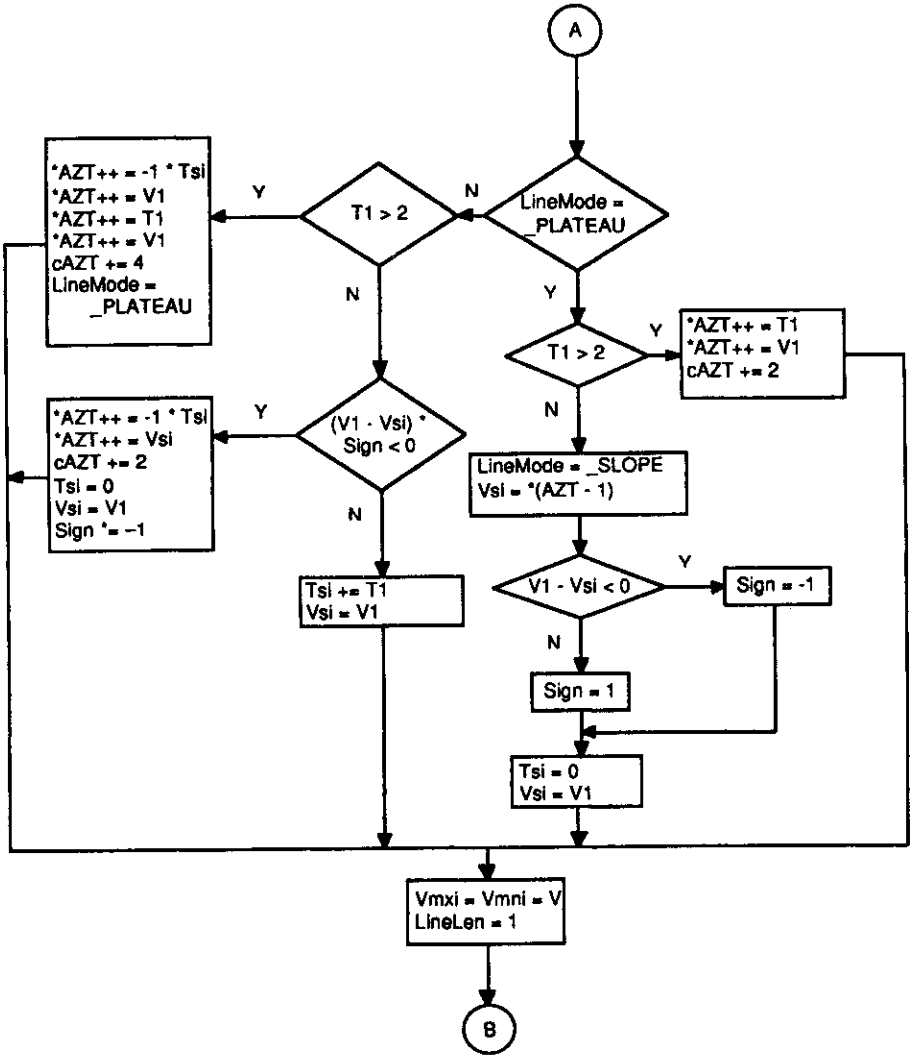


Figure 10.4(b) Flowchart of the line processing operation of the AZTEC algorithm.

When we reenter line processing with *LineMode* equal to _SLOPE, we either save or update the slope. The slope is saved either when a plateau of more than three samples can be formed or when a change in direction is detected. If we detect a new plateau of more than three samples, we store the current slope and the new plateau. For the slope, the stored values are its length T_{si} and its final elevation V_1.

Note that T_{si} is multiplied by –1 to differentiate a slope from a plateau (i.e., the minus sign serves as a flag to indicate a slope). We also store the length and the amplitude of the new plateau, then reset all parameters and return to plateau production.

If a change in direction is detected in the slope, we first save the parameters for the current slope and then reset *sign*, V_{si}, T_{si}, V_{mxi}, and V_{mni} to produce a new AZTEC slope. Now the algorithm returns to line detection but remains in slope production mode. When there is no new plateau or change of direction, we simply update the slope's parameters, T_{si} and V_{si}, and return to line detection with *LineMode* remaining set to _SLOPE.

AZTEC does not produce a constant data reduction ratio. The ratio is frequently as great as 10 or more, depending on the nature of the signal and the value of the empirically determined threshold.

10.2.2 Data reconstruction

The data array produced by the AZTEC algorithm is an alternating sequence of durations and amplitudes. A sample AZTEC-encoded data array is

$$\{18, 77, 4, 101, -5, -232, -4, 141, 21, 141\}$$

We reconstruct the AZTEC data by expanding the plateaus and slopes into discrete data points. For this particular example, the first two points represent a line 18 sample periods long at an amplitude of 77. The second set of two points represents another line segment 4 samples long at an amplitude of 101. The first value in the third set of two points is negative. Since this represents the length of a line segment, and we know that length must be positive, we recognize that this minus sign is the flag indicating that this particular set of points represents a line segment with nonzero slope. This line is five samples long beginning at the end of the previous line segment (i.e., amplitude of 101) and ending at an amplitude of –235. The next set of points is also a line with nonzero slope beginning at an amplitude of –235 and ending 4 sample periods later at an amplitude of 141.

This reconstruction process produces an ECG signal with steplike quantization, which is not clinically acceptable. The AZTEC-encoded signal needs postprocessing with a curve smoothing algorithm or a low-pass filter to remove its jagged appearance and produce more acceptable output.

The least square polynomial smoothing filter described in Chapter 5 is an easy and fast method for smoothing the signal. This family of filters fits a parabola to an odd number $(2L + 1)$ of input data points. Taking $L = 3$, we obtain

$$p_k = \frac{1}{21}(-2x_{k-3} + 3x_{k-2} + 6x_{k-1} + 7x_k + 6x_{k+1} + 3x_{k+2} - 2x_{k+3}) \quad (10.3)$$

where p_k is the new data point and x_k is the expanded AZTEC data. The smoothing function acts as a low-pass filter to reduce the discontinuities. Although this produces more acceptable output, it also introduces amplitude distortion.

Figure 10.5 shows examples of the AZTEC algorithm applied to an ECG.

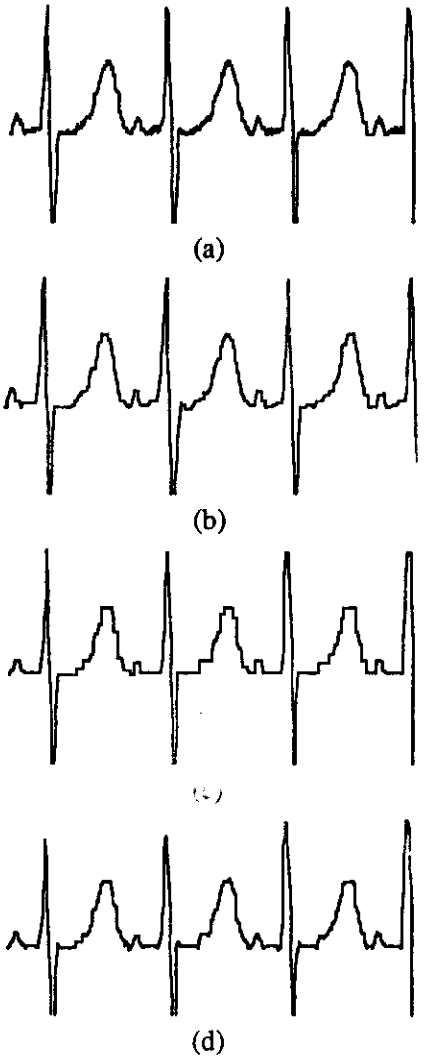


Figure 10.5 Examples of AZTEC applications. (a) Original waveform generated by the UW DigiScope **Genwave** function (see Appendix D). (b) Small threshold, reduction ratio = 512:233, PRD = 24.4%. (c) Large threshold, reduction ratio = 512:153, PRD = 28.1%. (d) Smoothed signal from (c), $L = 3$, PRD = 26.3%.

10.2.3 CORTES algorithm

The CORTES (Coordinate Reduction Time Encoding System) algorithm is a hybrid of the TP and AZTEC algorithms (Abenstein and Tompkins, 1982; Tompkins and Webster, 1981). It attempts to exploit the strengths of each while sidestepping the weaknesses. CORTES uses AZTEC to discard clinically insignificant data in the isoelectric region with a high reduction ratio and applies the TP algorithm to the clinically significant high-frequency regions (QRS complexes). It executes the AZTEC and TP algorithms in parallel on the incoming ECG data.

Whenever an AZTEC line is produced, the CORTES algorithm decides, based on the length of the line, whether the AZTEC data or the TP data are to be saved. If the line is longer than an empirically determined threshold, it saves the AZTEC line. Otherwise it saves the TP data points. Since TP is used to encode the QRS complexes, only AZTEC plateaus, not slopes, are implemented.

The CORTES algorithm reconstructs the signal by expanding the AZTEC plateaus and interpolating between each pair of the TP data points. It then applies parabolic smoothing to the AZTEC portions to reduce discontinuities.

10.3 FAN ALGORITHM

Originally used for ECG telemetry, the Fan algorithm draws lines between pairs of starting and ending points so that all intermediate samples are within some specified error tolerance, ε (Bohs and Barr, 1988). Figure 10.6 illustrates the principles of the Fan algorithm. We start by accepting the first sample X_0 as the nonredundant permanent point. It functions as the origin and is also called the originating point. We then take the second sample X_1 and draw two slopes $\{U_1, L_1\}$. U_1 passes through the point $(X_0, X_1 + \varepsilon)$, and L_1 passes through the point $(X_0, X_1 - \varepsilon)$. If the third sample X_2 falls within the area bounded by the two slopes, we generate two new slopes $\{U_2, L_2\}$ that pass through points $(X_0, X_2 + \varepsilon)$ and $(X_0, X_2 - \varepsilon)$. We compare the two pairs of slopes and retain the most converging (restrictive) slopes (i.e., $\{U_1, L_2\}$ in our example). Next we assign the value of X_2 to X_1 and read the next sample into X_2. As a result, X_2 always holds the most recent sample and X_1 holds the sample immediately preceding X_2. We repeat the process by comparing X_2 to the values of the most convergent slopes. If it falls outside this area, we save the length of the line T and its final amplitude X_1 which then becomes the new originating point X_0, and the process begins anew. The sketch of the slopes drawn from the originating sample to future samples forms a set of radial lines similar to a fan, giving this algorithm its name.

When adapting the Fan algorithm to C-language implementation, we create the variables, X_{U1}, X_{L1}, X_{U2}, and X_{L2}, to determine the bounds of X_2. From Figure 10.6(b), we can show that

$$X_{U2} = \frac{X_{U1} - X_0}{T} + X_{U1} \tag{10.4a}$$

and

$$X_{L2} = \frac{X_{L1} - X_0}{T} + X_{L1} \tag{10.4b}$$

where $T = t_T - t_0$.

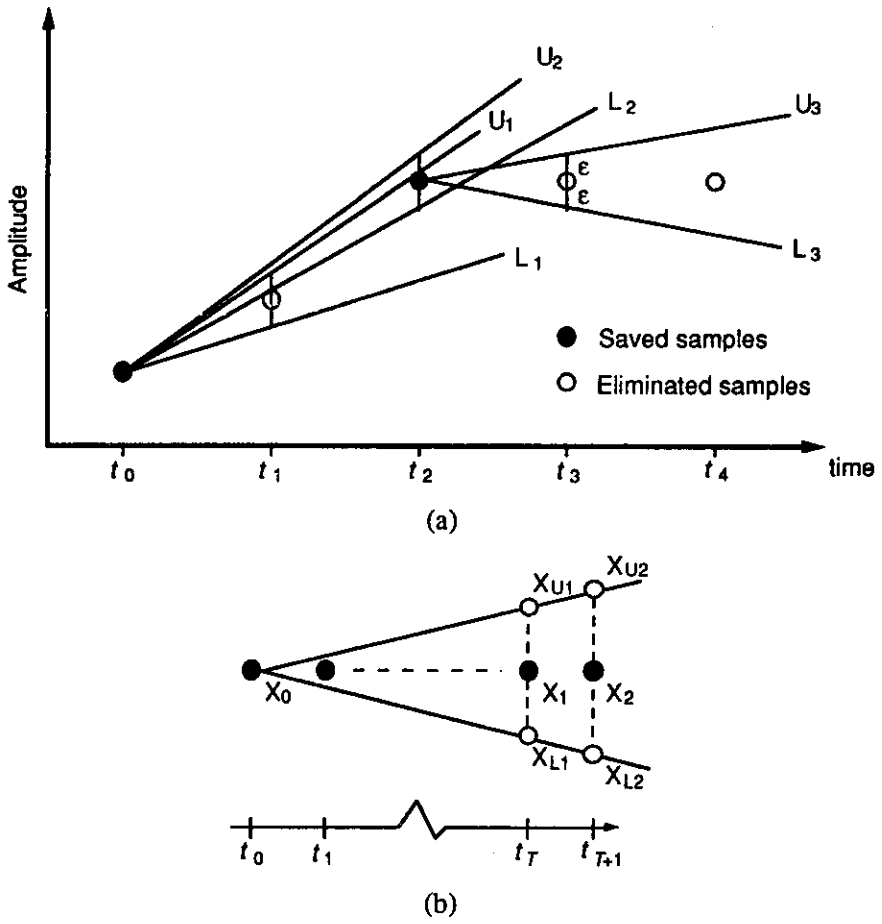


Figure 10.6 Illustration of the Fan algorithm. (a) Upper and lower slopes (U and L) are drawn within error threshold ε around sample points taken at $t_1, t_2, \ldots$ (b) Extrapolation of X_{U2} and X_{L2} from X_{U1}, X_{L1}, and X_0.

Figure 10.7 shows the C-language fragment that implements the Fan algorithm. Figure 10.8 shows an example of the Fan algorithm applied to an ECG signal.

```
short X0, X1, X2 ;              /* sample points */
short XU2, XL2, XU1, XL1 ;      /* variable to determine bounds */
short Epsilon ;                 /* threshold */
short *org, *fan ;              /* original and Fan data */
short T ;                       /* length of line */
short V2 ;                              /* sample point */

/* initialize all variables */
X0 = *org++ ;                   /* the originating point */
X1 = *org++ ;                   /* the next point */
T = 1 ;                         /* line length is initialize to 1 */
XU1 = X1 + Epsilon ;            /* upper bound of X1 */
XL1 = X1 - Epsilon ;            /* lower bound of X1 */
*fan++ = X0 ;                   /* save the first permanent point */

while( there_is_data )    {

        V2 = *org++ ;                           /* get next sample point */
        XU2 = (XU1 - X0)/T + XU1;               /* upper bound of X2 */
        XL2 = (XL2 - X0)/T + XL2; /* lower bound of X2 */

        if( X2 <= XU2 && X2 >= XL2 )    {       /* within bound */

                /* obtain the most restrictive bound */
                XU2 = (XU2 < X2 + Epsilon) ? XU2 : X2 + Epsilon ;
                XL2 = (XL2 > X2 - Epsilon) ? XL2 : X2 - Epsilon ;

                T++ ;  /* increment line length */
                X1 = X2 ;       /* X1 hold sample preceding X2 */
        }

        else    {       /* X2 out of bound, save line */

                *fan++ = T ;   /* save line length */
                *fan++ = X1 ;           /* save final amplitude */

                /* reset all variables */
                X0 = X1 ;
                X1 = X2 ;
                T = 1 ;
                XU1 = X1 + Epsilon ;
                XL1 = X1 - Epsilon ;
        }
}
```

Figure 10.7 Fragment of C-language program for implementation of the Fan algorithm.

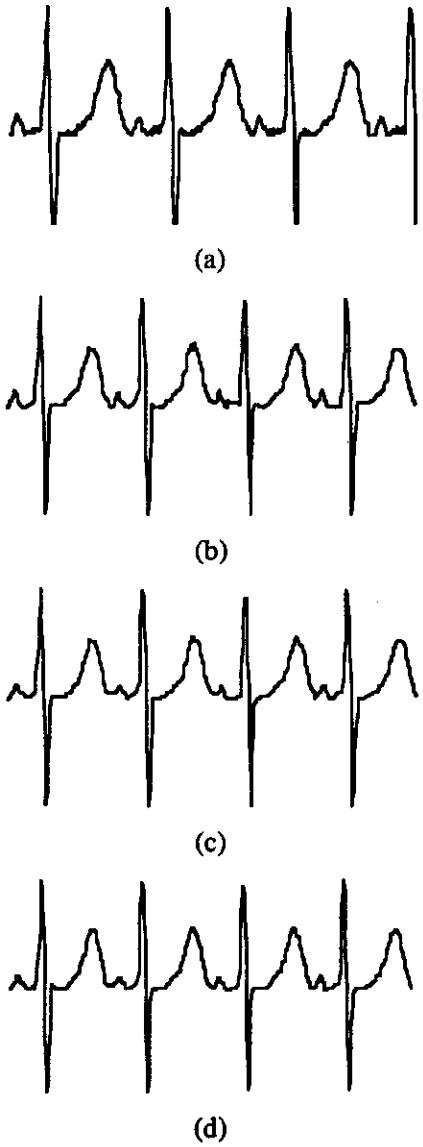


Figure 10.8 Examples of Fan algorithm applications. (a) Original waveform generated by the UW DigiScope **Genwave** function (see Appendix D). (b) Small tolerance, reduction ratio = 512:201 PRD = 5.6%. (c) Large tolerance, reduction ratio = 512:155, PRD = 7.2%. (d) Smoothed signal from (c), $L = 3$, PRD = 8.5%.

We reconstruct the compressed data by expanding the lines into discrete points. The Fan algorithm guarantees that the error between the line joining any two permanent sample points and any actual (redundant) sample along the line is less than or equal to the magnitude of the preset error tolerance. The algorithm's reduction ratio depends on the error tolerance. When compared to the TP and AZTEC algorithms, the Fan algorithm produces better signal fidelity for the same reduction ratio (Jalaleddine et al., 1990).

Three algorithms based on Scan-Along Approximation (SAPA) techniques (Ishijima et al., 1983; Tai, 1991) closely resemble the Fan algorithm. The SAPA-2 algorithm produces the best results among all three algorithms. As in the Fan algorithm, SAPA-2 guarantees that the deviation between the straight lines (reconstructed signal) and the original signal never exceeds a preset error tolerance.

In addition to the two slopes calculated in the Fan algorithm, SAPA-2 calculates a third slope called the center slope between the originating sample point and the actual future sample point. Whenever the center slope value does not fall within the boundary of the two converging slopes, the immediate preceding sample is taken as the originating point. Therefore, the only apparent difference between SAPA-2 and the Fan algorithm is that the SAPA-2 uses the center slope criterion instead of the actual sample value criterion.

10.4 HUFFMAN CODING

Huffman coding exploits the fact that discrete amplitudes of quantized signal do not occur with equal probability (Huffman, 1952). It assigns variable-length code words to a given quantized data sequence according to their frequency of occurrence. Data that occur frequently are assigned shorter code words.

10.4.1 Static Huffman coding

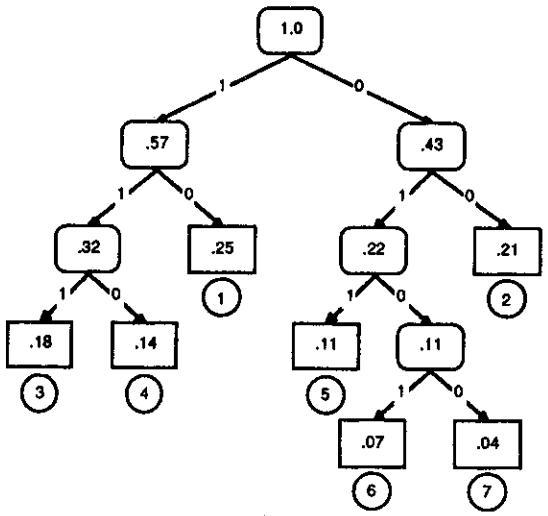
Figure 10.9 illustrates the principles of Huffman coding. As an example, assume that we wish to transmit the set of 28 data points

$$\{1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 6, 6, 7\}$$

The set consists of seven distinct quantized levels, or *symbols*. For each symbol, S_i, we calculate its probability of occurrence P_i by dividing its frequency of occurrence by 28, the total number of data points. Consequently, the construction of a Huffman code for this set begins with seven nodes, one associated with each P_i. At each step we sort the P_i list in descending order, breaking the ties arbitrarily. The two nodes with smallest probability, P_i and P_j, are merged into a new node with probability $P_i + P_j$. This process continues until the probability list contains a single value, 1.0, as shown in Figure 10.9(a).

S_i	Lists of P_i						
1	.25	.25	.25	.32	.43	.57	1.0
2	.21	.21	.22	.25	.32	.43	
3	.18	.18	.21	.22	.25		
4	.14	.14	.18	.21			
5	.11	.11	.14				
6	.07	.11					
7	.04						

(a)



(b)

Symbols, S_i	3-bit binary code	Probability of occurrence, P_i	Huffman code
1	001	0.25	10
2	010	0.21	00
3	011	0.18	111
4	100	0.14	110
5	101	0.11	011
6	110	0.07	0101
7	111	0.04	0100

(c)

Figure 10.9 Illustration of Huffman coding. (a) At each step, P_i are sorted in descending order and the two lowest P_i are merged. (b) Merging operation depicted in a binary tree. (c) Summary of Huffman coding for the data set.

The process of merging nodes produces a binary tree as in Figure 10.9(b). When we merge two nodes with probability $P_i + P_j$, we create a parent node with two children represented by P_i and P_j. The root of the tree has probability 1.0. We obtain the Huffman code of the symbols by traversing down the tree, assigning 1 to the left child and 0 to the right child. The resulting code words have the *prefix property* (i.e., no code word is a proper prefix of any other code word). This property ensures that a coded message is uniquely decodable without the need for lookahead. Figure 10.9(c) summarizes the results and shows the Huffman codes for the seven symbols. We enter these code word mappings into a translation table and use the table to pad the appropriate code word into the output bit stream in the reduction process.

The reduction ratio of Huffman coding depends on the distribution of the source symbols. In our example, the original data requires three bits to represent the seven quantized levels. After Huffman coding, we can calculate the expected code word length

$$E[l] = \sum_{i=1}^{7} l_i P_i \tag{10.5}$$

where l_i represents the length of Huffman code for the symbols. This value is 2.65 in our example, resulting in an expected reduction ratio of 3:2.65.

The reconstruction process begins at the root of the tree. If bit 1 is received, we traverse down the left branch, otherwise the right branch. We continue traversing until we reach a node with no child. We then output the symbol corresponding to this node and begin traversal from the root again.

The reconstruction process of Huffman coding perfectly recovers the original data. Therefore it is a lossless algorithm. However, a transmission error of a single bit may result in more than one decoding error. This propagation of transmission error is a consequence of all algorithms that produce variable-length code words.

10.4.2 Modified Huffman coding

The implementation of Huffman coding requires a translation table, where each source symbol is mapped to a unique code word. If the original data were quantized into 16-bit numbers, the table would need to contain 2^{16} records. A table of this size creates memory problems and processing inefficiency.

In order to reduce the size of the translation table, the modified Huffman coding scheme partitions the source symbols into a frequent set and an infrequent set. For all the symbols in the frequent set, we form a Huffman code as in the static scheme. We then use a special code word as a prefix to indicate any symbol from the infrequent set and attach a suffix corresponding to the ordinary binary encoding of the symbol.

Assume that we are given a data set similar to the one before. Assume also that we anticipate quantized level 0 to appear in some future transmissions. We may decide to partition the quantized levels {0, 7} into the infrequent set. We then apply Huffman coding as before and obtain the results in Figure 10.10. Note that

quantized levels in the infrequent set have codes with prefix 0100, making their code length much longer than those of the frequent set. It is therefore important to keep the probability of the infrequent set sufficiently small to achieve a reasonable reduction ratio.

Some modified Huffman coding schemes group quantized levels centered about 0 into the frequent set and derive two prefix codes for symbols in the infrequent set. One prefix code denotes large positive values and the other denotes large negative values.

10.4.3 Adaptive coding

Huffman coding requires a translation table for encoding and decoding. It is necessary to examine the entire data set or portions of it to determine the data statistics. The translation table must also be transmitted or stored for correct decoding.

An adaptive coding scheme attempts to build the translation table as data are presented. A dynamically derived translation table is sensitive to the variation in local statistical information. It can therefore alter its code words according to local statistics to maximize the reduction ratio. It also achieves extra space saving because there is no need for a static table.

An example of an adaptive scheme is the Lempel-Ziv-Welch (LZW) algorithm. The LZW algorithm uses a fixed-size table. It initializes some positions of the table for some chosen data sets. When it encounters new data, it uses the uninitialized positions so that each unique data word is assigned its own position. When the table is full, the LZW algorithm reinitializes the oldest or least-used position according to the new data. During data reconstruction, it incrementally rebuilds the translation table from the encoded data.

Symbols, S_i	3-bit binary code	Probability of occurrence, P_i	Huffman code
0	000	0.00	0100000
1	001	0.25	10
2	010	0.21	00
3	011	0.18	111
4	100	0.14	110
5	101	0.11	011
6	110	0.07	0101
7	111	0.04	0100111

Figure 10.10 Results of modified Huffman coding. Quantized levels {0, 7} are grouped into the infrequent set.

10.4.4 Residual differencing

Typically, neighboring signal amplitudes are not statistically independent. Conceptually we can decompose a sample value into a part that is correlated with past samples and a part that is uncorrelated. Since the intersample correlation corresponds to a value predicted using past samples, it is redundant and removable. We are then left with the uncorrelated part which represents the prediction error or residual signal. Since the amplitude range of the residual signal is smaller than that of the original signal, it requires less bits for representation. We can further reduce the data by applying Huffman coding to the residual signal. We briefly describe two ECG reduction algorithms that make use of residual differencing.

Ruttimann and Pipberger (1979) applied modified Huffman coding to residuals obtained from prediction and interpolation. In prediction, sample values are obtained by taking a linearly weighted sum of an appropriate number of past samples

$$x'(nT) = \sum_{k=1}^{p} a_k x(nT - kT) \tag{10.6}$$

where $x(nT)$ are the original data, $x'(nT)$ are the predicted samples, and p is the number of samples employed in prediction. The parameters a_k are chosen to minimize the expected mean squared error $E[(x - x')^2]$. When $p = 1$, we choose $a_1 = 1$ and say that we are taking the *first difference* of the signal. Preliminary investigations on test ECG data showed that there was no substantial improvement by using predictors higher than second order (Ruttimann et al., 1976). In interpolation, the estimator of the sample value consists of a linear combination of past and future samples. The results for the predictor indicated a second-order estimator to be sufficient. Therefore, the interpolator uses only one past and one future sample

$$x'(n) = ax(nT - T) + bx(nT + T) \tag{10.7}$$

where the coefficients a and b are determined by minimizing the expected mean squared error. The residuals of prediction and interpolation are encoded using a modified Huffman coding scheme, where the frequent set consists of some quantized levels centered about zero. Encoding using residuals from interpolation resulted in higher reduction ratio of approximately 7.8:1.

Hamilton and Tompkins (1991a, 1991b) exploited the fact that a typical ECG signal is composed of a repeating pattern of beats with little change from beat to beat. The algorithm calculates and updates an average beat estimate as data are presented. When it detects a beat, it aligns and subtracts the detected beat from the average beat. The residual signal is Huffman coded and stored along with a record of the beat locations. Finally, the algorithm uses the detected beat to update the average beat estimate. In this scheme, the estimation of beat location and quantizer location can significantly affect reduction performance.

0.4.5 Run-length encoding

Used extensively in the facsimile technology, run-length encoding exploits the high egree of correlation that occurs in successive bits in the facsimile bit stream. A bit n the facsimile output may either be 1 or 0, depending on whether it is a black or vhite pixel. On a typical document, there are clusters of black and white pixels that ive rise to this high correlation. Run-length encoding simply transforms the riginal bit stream into the string $\{v_1, l_1, v_2, l_2, \ldots\}$ where v_i are the values and l_i re the lengths. The observant reader will quickly recognize that both AZTEC and he Fan algorithm are special cases of run-length encoding.

Take for example the output stream {1, 1, 1, 1, 1, 3, 3, 3, 3, 0, 0, 0} with 12 lements. The output of run-length encoding {1, 5, 3, 4, 0, 3} contains only six lements. Further data reduction is possible by applying Huffman coding to the utput of run-length encoding.

10.5 LAB: ECG DATA REDUCTION ALGORITHMS

This lab explores the data reduction techniques reviewed in this chapter. Load UW DigiScope according to the directions in Appendix D.

10.5.1 Turning point algorithm

From the **ad(V) Ops** menu, select **C(O)mpress** and then **(T)urn pt**. The program compresses the waveform displayed on the top channel using the TP algorithm, then decompresses, reconstructs using interpolation, and displays the results on the bottom channel. Perform the TP algorithm on two different ECGs read from files and on a sine wave and a square wave. Observe

1. Quality of the reconstructed signal
2. Reduction ratio
3. Percent root-mean-square difference (PRD)
4. Power spectra of original and reconstructed signals.

Tabulate and summarize all your observations.

10.5.2 AZTEC algorithm

Repeat section 10.5.1 for the AZTEC algorithm by selecting **(A)ztec** from the **COMPRESS** menu. Using at least three different threshold values (try 1%, 5%, and 15% of the full-scale peak-to-peak value), observe and comment on the items in the list in section 10.5.1. In addition, summarize the quality of the reconstructed

signals both before and after applying the smoothing filter. Tabulate and summarize all your observations.

10.5.3 Fan algorithm

Repeat section 10.5.2 for the Fan algorithm by selecting **(F)an** from the **COMPRESS** menu. What can you deduce from comparing the performance of the Fan algorithm with that of the AZTEC algorithm? Tabulate and summarize all your observations.

10.5.4 Huffman coding

Select **(H)uffman** from the **COMPRESS** menu. Select **(R)un** in order to Huffman encode the signal that is displayed on the top channel. Do not use first differencing at this point in the experiment. Record the reduction ratio. Note that this reduction ratio does not include the space needed for the translation table which must be stored or transmitted. What can you deduce from the PRD? Select **(W)rite table** to write the Huffman data into a file. You may view the translation table later with the DOS **type** command after exiting from **SCOPE**.

Load a new ECG waveform and repeat the steps above. When you select **(R)un**, the program uses the translation table derived previously to code the signal. What can you deduce from the reduction ratio? After deriving a new translation table using **(M)ake** from the menu, select **(R)un** again and comment on the new reduction ratio.

Select **(M)ake** again and use first differencing to derive a new Huffman code. Is there a change in the reduction ratio using this newly derived code? Select **(W)rite table** to write the Huffman data into a file. Now reload the first ECG waveform that you used. Without deriving a new Huffman code, observe the reduction ratio obtained. Comment on your observations.

Exit from the **SCOPE** program to look at the translation tables that you generated. What comments can you make regarding the overhead involved in storing a translation table?

10.6 REFERENCES

Abenstein, J. P. and Tompkins, W. J. 1982. New data-reduction algorithm for real-time ECG analysis, *IEEE Trans. Biomed. Eng.*, **BME-29**: 43–48.

Bohs, L. N. and Barr, R. C. 1988. Prototype for real-time adaptive sampling using the Fan algorithm, *Med. & Biol. Eng. & Comput.*, **26**: 574–83.

Cox, J. R., Nolle, F. M., Fozzard, H. A., and Oliver, G. C. Jr. 1968. AZTEC: a preprocessing program for real-time ECG rhythm analysis. *IEEE Trans. Biomed. Eng.*, **BME-15**: 128–29.

Hamilton, P. S., and Tompkins, W. J. 1991a. Compression of the ambulatory ECG by average beat subtraction and residual differencing. *IEEE Trans. Biomed. Eng.*, **BME-38**(3): 253–59.

Hamilton, P. S., and Tompkins, W. J. 1991b. Theoretical and experimental rate distortion performance in compression of ambulatory ECGs. *IEEE Trans. Biomed. Eng.*, **BME-38**(3): 260–66.

Huffman, D. A. 1952. A method for construction of minimum-redundancy codes. *Proc. IRE*, **40**: 1098–1101.
Ishijima, M., Shin, S. B., Hostetter, G. H., and Skalansky, J. 1983. Scan-along polygonal approximation for data compression of electrocardiograms, *IEEE Trans. Biomed. Eng.*, **BME-30**: 723–29.
Jalaleddine, S. M. S., Hutchens, C. G., Coberly, W. A., and Strattan, R. D. 1988. Compression of Holter ECG data. *Biomedical Sciences Instrumentation*, **24**: 35–45.
Jalaleddine, S. M. S., Hutchens. C. G., and Strattan, R. D. 1990. ECG data compression techniques — A unified approach. *IEEE Trans. Biomed. Eng.*, **BME-37**: 329–43.
Moody, G. B., Soroushian., and Mark, R. G. 1988. ECG data compression for tapeless ambulatory monitors. *Computers in Cardiology*, 467–70.
Mueller, W. C. 1978. Arrhythmia detection program for an ambulatory ECG monitor. *Biomed. Sci. Instrument.*, **14**: 81–85.
Ruttimann, U. E. and Pipberger, H. V. 1979. Compression of ECG by prediction or interpolation and entropy encoding. *IEEE Trans. Biomed. Eng.*, **BME-26**: 613–23.
Ruttimann, U. E., Berson, A. S., and Pipberger, H. V. 1976. ECG data compression by linear prediction. *Proc. Comput. Cardiol.*, 313–15.
Tai, S. C. 1991. SLOPE — a real-time ECG data compressor. *Med. & Biol. Eng. & Comput.*, 175–79.
Tompkins, W. J. and Webster, J. G. (eds.) 1981. *Design of Microcomputer-based Medical Instrumentation*. Englewood Cliffs, NJ: Prentice Hall.

10.7 STUDY QUESTIONS

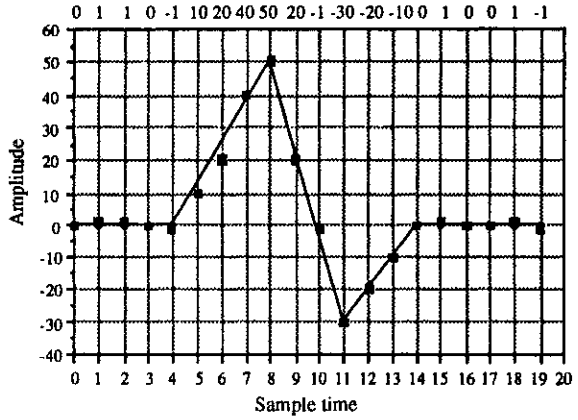
10.1 Explain the meaning of lossless and lossy data compression. Classify the four data reduction algorithms described in this chapter into these two categories.
10.2 Given the following data: {15, 10, 6, 7, 5, 3, 4, 7, 15, 3}, produce the data points that are stored using the TP algorithm.
10.3 Explain why an AZTEC reconstructed waveform is unacceptable to a cardiologist. Suggest ways to alleviate the problem.
10.4 The Fan algorithm can be applied to other types of biomedical signals. List the desirable characteristics of the biomedical signal that will produce satisfactory results using this algorithm. Give an example of such a signal.
10.5 Given the following data set: {a, a, a, a, b, b, b, b, b, c, c, c, d, d, e}, derive the code words for the data using Huffman coding. What is the average code word length?
10.6 Describe the advantages and disadvantages of modified Huffman coding.
10.7 Explain why it is desirable to apply Huffman coding to the residuals obtained by subtracting the estimated sample points from the original sample points.
10.8 Data reduction can be performed using parameter extraction techniques. A particular characteristic or parameter of the signal is extracted and transmitted in place of the original signal. Draw a block diagram showing the possible configuration for such a system. Your block diagram should include the compression and the reconstruction portions. What are the factors governing the success of these techniques?
10.9 Does the TP algorithm (a) produce significant time-base distortion over a very long time, (b) save every turning point (i.e., peak or valley) in a signal, (c) provide data reduction of 4-to-1 if applied twice to a signal without violating sampling theory, (d) provide for exactly reconstructing the original signal, (e) perform as well as AZTEC for electroencephalography (EEG)? Explain your answers.
10.10 Which of the following are characteristic of a Huffman coding algorithm? (a) Guarantees more data reduction on an ECG than AZTEC; (b) Cannot perfectly reconstruct the sampled data points (within some designated error range); (c) Is a variable-length code; (d) Is derived directly from Morse code; (e) Uses ASCII codes for the most frequent A/D

values; (f) Requires advance knowledge of the frequency of occurrence of data patterns; (g) Includes as part of the algorithm self-correcting error checks.

10.11 After application of the TP algorithm, what data sequence would be saved if the data sampled by an analog-to-digital converter were: (a) {20, 40, 20, 40, 20, 40, 20, 40}, (b) {50, 40, 50, 20, 30, 40}, (c) {50, 50, 40, 30, 40, 50, 40, 30, 40, 50, 50, 40}, (d) {50, 25, 50, 25, 50, 25, 50, 25}?

10.12 After application of the TP algorithm on a signal, the data points saved are {50, 70, 30, 40}. If you were to reconstruct the original data set, what is the data sequence that would best approximate it?

10.13 The graph below shows a set of 20 data points sampled from an analog-to-digital converter. At the top of the chart are the numerical values of the samples. The solid lines represent AZTEC encoding of this sampled signal.



(a) List the data array that represents the AZTEC encoding of this signal.
(b) How much data reduction does AZTEC achieve for this signal?
(c) Which data points in the following list of raw data that would be saved if the Turning Point algorithm were applied to this signal?

0 1 1 0 -1 10 20 40 50 20 -1 -30 -20 -10 0 1 0 0 1 -1

(d) If this signal were encoded with a Huffman-type variable-bit-length code with the following four bit patterns as part of the set of codes, indicate which amplitude value you would assign to each pattern.

Code	Amplitude value
1	
01	
001	
0001	

(e) How much data reduction does each algorithm provide (assuming that no coding table needs to be stored for Huffman coding)?

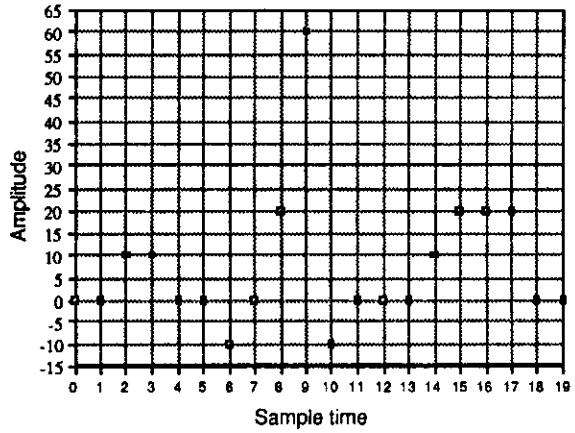
10.14 AZTEC encodes a signal as {2, 50, –4, 30, –4, 50, –4, 30, –4, 50, 2, 50}. How many data points were originally sampled?

10.15 After applying the AZTEC algorithm to a signal, the saved data array is {2, 0, –3, 80, –3, –30, –3, 0, 3, 0}. Draw the waveform that AZTEC would reconstruct from these data.

10.16 AZTEC encodes a signal from an 8-bit analog-to-digital converter as {2, 50, –4, 30, –6, 50, –6, 30, –4, 50, 2, 50}. (a) What is the amount of data reduction? (b) What is the peak-to-peak amplitude of a signal reconstructed from these data?

10.17 AZTEC encodes a signal from an 8-bit analog-to-digital converter as {3, 100, –5, 150, –5, 50, 5, 100, 2, 100}. The TP algorithm is applied to the same original signal. How much more data reduction does AZTEC achieve on the same signal compared to TP?

10.18 The graph below shows a set of 20 data points of an ECG sampled with an 8-bit analog-to-digital converter.



(a) Draw a Huffman binary tree similar to the one in Figure 10.9(b) including the probabilities of occurrence for this set of data.
(b) (5 points) From the binary tree, assign appropriate Huffman codes to the numbers in the data array:

Number	Huffman code
–10	
0	
10	
20	
60	

(c) Assuming that the Huffman table does not need to be stored, how much data reduction is achieved with Huffman coding of this sampled data set? (Note: Only an integral number of bytes may be stored.)
(d) Decode the following Huffman-coded data and list the sample points that it represents:

0 1 0 1 0 1 1 0 0 0 1

11

Other Time- and Frequency-Domain Techniques

Dorin Panescu

A biomedical signal is often corrupted by noise (e.g., powerline interference, muscle or motion artifacts, RF interference from electrosurgery or diathermy apparatus). Therefore, it is useful to know the frequency spectrum of the corrupting signal in order to be able to design a filter to eliminate it. If we want to find out, for example, how well the patient's cardiac output is correlated with the area of the QRS complex, then we need to use proper correlation techniques. This chapter presents time and frequency-domain techniques that might be useful for situations such as those exemplified above.

11.1 THE FOURIER TRANSFORM

The digital computer algorithm for Fourier analysis called the fast Fourier transform (FFT) serves as a basic tool for frequency-domain analysis of signals.

11.1.1 The Fourier transform of a discrete nonperiodic signal

Assuming that a discrete-time aperiodic signal exists as a sequence of data sampled from an analog prototype with a sampling period of T, the angular sampling frequency being $\omega_s = 2\pi/T$, we can write this signal in the time domain as a series of weighted Dirac functions. Thus

$$x(t) = \sum_{n=-\infty}^{\infty} x(n)\,\delta(t - nT) \tag{11.1}$$

The Fourier transform of this expression is

$$X(\omega) = \int_{-\infty}^{\infty} x(t)\, e^{-j\omega t} dt \tag{11.2}$$

or

$$X(\omega) = \int_{-\infty}^{\infty} \sum_{n=-\infty}^{\infty} x(n)\ \ \delta(t - nT)\, e^{-j\omega t} dt \tag{11.3a}$$

The ordering of integration and summation can be changed to give

$$X(\omega) = \sum_{n=-\infty}^{\infty} x(n) \int_{-\infty}^{\infty} \delta(t - nT) e^{-j\omega t}\, dt \tag{11.3b}$$

Thus we obtain

$$X(\omega) = \sum_{n=-\infty}^{\infty} x(n)\, e^{-j\omega nT} \tag{11.3c}$$

And similarly, we can find that the inverse Fourier transform is

$$x(n) = \frac{T}{2\pi} \int_{0}^{\omega_S} X(\omega)\, e^{j\omega nT}\, d\omega \tag{11.4}$$

One of the important properties of the Fourier transform, which is shown in Figure 11.1(b), is its repetition at intervals of the sampling frequency in both positive and negative directions. Also it is remarkable that the components in the interval $0 < \omega < \omega_S/2$ are the complex conjugates of the components in the interval $\omega_S/2 < \omega < \omega_S$. It is modern practice to use normalized frequencies, which means that the sampling period T is taken to be 1. Therefore, the Fourier transform pair for discrete signals, considering normalized frequencies, is

$$X(\omega) = \sum_{n=-\infty}^{\infty} x(n)\, e^{-j\omega n} \tag{11.5a}$$

$$x(n) = \frac{1}{2\pi} \int_{-\pi}^{\pi} X(\omega)\ e^{j\omega n}\, d\omega \tag{11.5b}$$

We observe that this kind of Fourier transform is continuous, and it repeats at intervals of the sampling frequency.

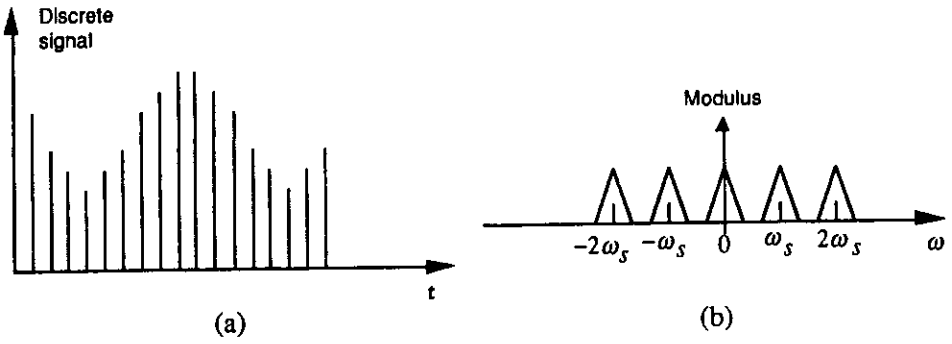


Figure 11.1 (a) A discrete-time signal, and (b) the modulus of its Fourier transform. Symmetry about $\omega_s/2$, due to the sampling process, is illustrated.

11.1.2 The discrete Fourier transform for a periodic signal

The discrete Fourier transform (DFT) is the name given to the calculation of the Fourier series coefficients for a discrete periodic signal. The operations are similar to the calculation of Fourier coefficients for a periodic signal, but there are also certain marked differences. The first is that the integrals become summations in the discrete time domain. The second difference is that the transform evaluates only a finite number of complex coefficients, the total being equal to the original number of data points in one period of original signal. Because of this, each spectral line is regarded as the k-*th* harmonic of the basic period in the data rather than identifying with a particular frequency expressed in Hz or radian/s. Algebraically, the forward and reverse transforms are expressed as

$$X(k) = \sum_{n=0}^{N-1} x(n)\, e^{\frac{-jkn\,2\pi}{N}} \tag{11.6a}$$

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k)\, e^{\frac{jkn\,2\pi}{N}} \tag{11.6b}$$

Figure 11.2 shows a discrete periodic signal and the real and imaginary parts of its DFT. The first spectral line ($k = 0$) gives the amplitude of the dc component in the signal, and the second line corresponds to that frequency which represents one

cycle in N data points. This frequency is $2\pi/N$. The N-*th* line corresponds to the sampling frequency of the discrete N-sample data sequence per period and the $k = N/2$-*th* line corresponds to the Nyquist frequency. Using the symmetry of the DFT, algorithms for fast computation have been developed. Also, the symmetry has two important implications. The first is that the transformation will yield N unique complex spectral lines. The second is that half of these are effectively redundant because all of the information contained in a real time domain signal is contained within the first $N/2$ complex spectral lines. These facts permitted the development of the Fast Fourier Transform (FFT), which is presented in the next section.

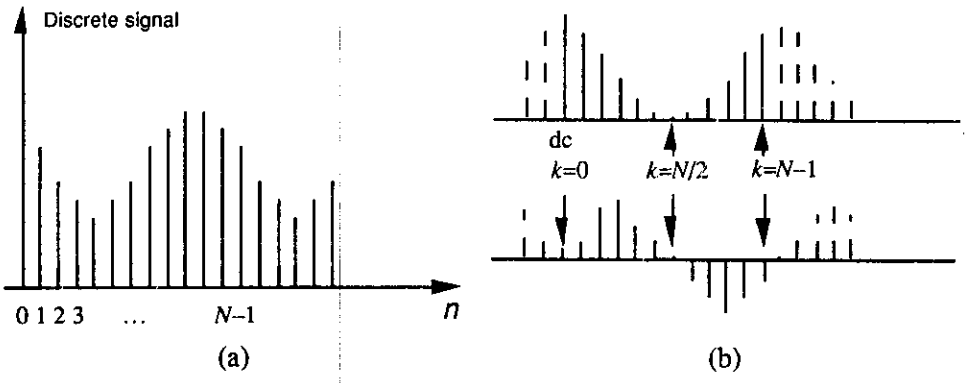


Figure 11.2 (a) A discrete periodic signal and (b) the real and imaginary parts of its DFT.

11.1.3 The fast Fourier transform

For a complete discussion of this subject, see Oppenheim and Schafer (1975). The term FFT applies to any computational algorithm by which the discrete Fourier transform can be evaluated for a signal consisting of N equally spaced samples, with N usually being a power of two. To increase the computation efficiency, we must divide the DFT into successively smaller DFTs. In this process we will use the symmetry and the periodicity properties of the complex exponential

$$W_N^{kn} = e^{-j(2\pi/N)kn}$$

where W_N substitutes for $e^{-j(2\pi/N)}$. Algorithms in which the decomposition is based on splitting the sequence $x(n)$ into smaller sequences are called decimation in time algorithms. The principle of decimation in time is presented below for N equal to an integer power of 2. We can consider, in this case, $X(k)$ to be formed by two

$N/2$-point sequences consisting of the even-numbered points in $x(n)$ and odd-numbered points in $x(n)$, respectively. Thus we obtain

$$X(k) = \sum_{n=2p+1} x(n) W_N^{nk} + \sum_{n=2p} x(n) W_N^{nk} \tag{11.7}$$

which can also be written as

$$X(k) = \sum_{p=0}^{N/2-1} x(2p) W_N^{2pk} + \sum_{p=0}^{N/2-1} x(2p+1) W_N^{(2p+1)k} \tag{11.8}$$

But $W_N^2 = W_{N/2}$ and consequently Eq. (11.8) can be written as

$$X(k) = \sum_{p=0}^{N/2-1} x(2p) W_{N/2}^{pk} + W_N^k \sum_{p=0}^{N/2-1} x(2p+1) W_{N/2}^{pk} = X_e(k) + W_N^k X_o(k) \tag{11.9}$$

Each of the sums in Eq. (11.9) is an $N/2$-point DFT of the even- and odd-numbered points of the original sequence, respectively.

After the two DFTs are computed, they are combined to give the DFT for the original N-point sequence. We can proceed further by decomposing each of the two $N/2$-point DFTs into two $N/4$-point DFTs and each of the four $N/4$-point DFTs into two $N/8$-point DFTs, and so forth. Finally we reduce the computation of the N-point DFT to the computation of the 2-point DFTs and the necessary additions and multiplications.

Figure 11.3 shows the computations involved in computing $X(k)$ for an 8-point original sequence. Oppenheim and Schafer (1975) show that the total number of complex additions and multiplications involved is $N\log_2 N$. The original N-point DFT requires N^2 complex multiplications and additions; thus it turns out that the FFT algorithm saves us considerable computing time.

Figure 11.4 shows the computation time for the FFT and the original DFT versus N. The FFT requires at least an order of magnitude fewer computations than the DFT. As an example, some modern microcomputers equipped with a math coprocessor are able to perform an FFT for a 1024-point sequence in much less than 1 s. In the case when N is not an integer power of 2, the common procedure is to augment the finite-length sequence with zeros until the total number of points reaches the closest power of 2, or the power for which the FFT algorithm is written. This technique is called zero padding. In order to make the error as low as possible, sometimes the signal is multiplied with a finite-length window function. Windowing is also applied when N is an integer power of 2 but the FFT-analyzed signal does not contain an integer number of periods within the N points. In such cases, the error introduced by the unfinished period of the signal may be reduced by a proper choice of the window type.

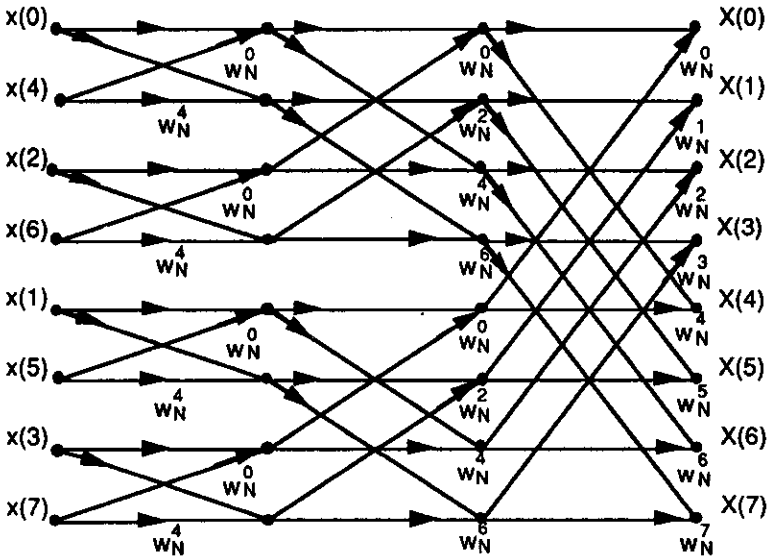


Figure 11.3 The flow graph of the decimation-in-time of an 8-point DFT.

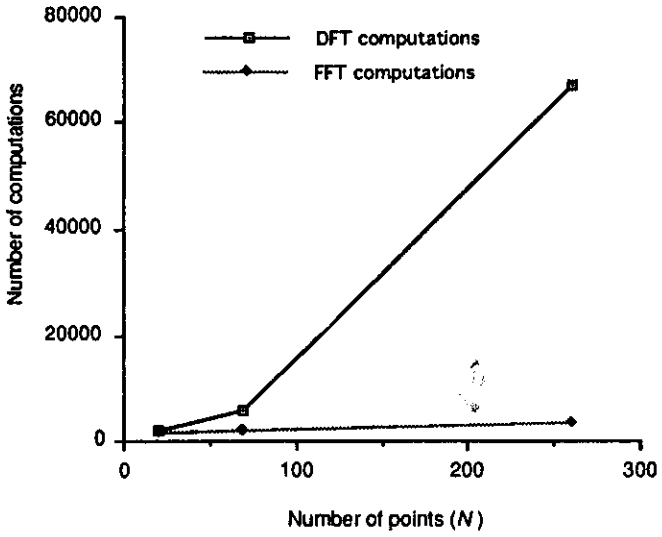


Figure 11.4 Computational savings with the FFT.

11.1.4 C-language FFT function

The computational flow graph presented in Figure 11.3 describes an algorithm for the computation of the FFT of finite-length signal applicable when the number of points is an integer power of 2. When implementing the computations depicted in this figure, we can imagine the use of two arrays of storage registers, one for the array being computed and one for the data being used in computation. For example, in computing the first array, one set of storage registers would contain the input data and the second set of storage registers would contain the computed results for the first stage.

In order to perform the computation based on the "butterfly" graph, the input data must be stored in a nonsequential order. In fact, the order in which the input data must be stored is the bit-reversed order.

To show what is meant by this, we write the index of the output data and the index of the corresponding input data using three binary digits.

X(000) — x(000)
X(001) — x(100)
X(010) — x(010)
X(011) — x(110)
X(100) — x(001)
X(101) — x(101)
X(110) — x(011)
X(111) — x(111)

If $(n_2\ n_1\ n_0)$ is the binary representation of the index of the sequence $x(n)$, which is the input data, then $\{x(n)\}$ must be rearranged such that the new position of $x(n_2\ n_1\ n_0)$ is $x(n_0\ n_1\ n_2)$.

Figure 11.5 shows a C-language fragment for the FFT computation, which reorders the input array, $x[nn]$. The FFT function in UW DigiScope allows the user to zero-pad the signal or to window it with the different windows.

11.2 CORRELATION

We now investigate the concept of correlation between groups of data or between signals. Correlation between groups of data implies that they move or change with respect to each other in a structured way. To study the correlation between signals, we will consider signals that have been digitized and that therefore form groups of data.

```
#define RORD(a,b) tempr=(a);(a)=(b);(b)=tempr
...
...
float tempr,x[512];
int i,j,m,n,nn;
...
...
nn=512;
n=nn<<1;
j=1;
for (i=1;i<n;i++){
     if (j>i){
        RORD(x[j],x[i]);/*this is the bit-reversal section of*/
        RORD(x[j+1],x[i+1]);/*a FFT computation routine*/
     }
     m=n >>1;
     while (m>=2 && j>m){
          j-=m;
          m >>1;
     }
     j+=m;
}
```

Figure 11.5 The C-language program for bit-reversal computations.

11.2.1 Correlation in the time domain

For N pairs of data $\{x(n),y(n)\}$, the correlation coefficient is defined as

$$rxy = \frac{\sum_{n=1}^{N} \{x(n)-\bar{x}\}\ \{y(n)-\bar{y}\}}{\sqrt{\sum_{n=1}^{N} \{x(n)-\bar{x}\}^2 \sum_{n=1}^{N} \{y(n)-\bar{y}\}^2}} \tag{11.10}$$

If finite-length signals are to be analyzed, then we must define the crosscorrelation function of the two signals.

$$r_{xy}(k) = \frac{\sum_{n=1}^{N} \{x(n)-\bar{x}\}\ \{y(n+k)-\bar{y}\}}{\sqrt{\sum_{n=1}^{N} \{x(n)-\bar{x}\}^2 \sum_{n=1}^{N} \{y(n)-\bar{y}\}^2}} \tag{11.11}$$

In the case when the two input signals are the same, the crosscorrelation function becomes the autocorrelation function of that signal. Thus, the autocorrelation function is defined as

$$r_{xx}(k) = \frac{\sum_{n=1}^{N} \{x(n) - \bar{x}\}\ \{x(n+k) - \bar{x}\}}{\sum_{n=1}^{N} \{x(n) - \bar{x}\}^2} \tag{11.12}$$

Figure 11.6 presents the crosscorrelation of respiratory signals recorded simultaneously from a human subject using impedance pneumography. In Figure 11.6(a), the signals were acquired at different points along the midaxillary line of the subject. The subject was breathing regularly at the beginning of the recording, moving without breathing in the middle of the recording, and breathing regularly again at the end. In Figure 11.6(b), each combination of two recorded channels were crosscorrelated in order to try to differentiate between movement and regular breathing.

11.2.2 Correlation in the frequency domain

The original definition for the crosscorrelation was for continuous signals. Thus if $h(t)$ and $g(t)$ are two continuous signals, then their crosscorrelation function is defined as

$$c_{gh}(t) = \int_{-\infty}^{\infty} g(\tau)\, h(t+\tau) d\tau \tag{11.13}$$

The Fourier transform of the crosscorrelation function satisfies

$$Corr\ (\omega) = G(\omega)^*\, H(\omega) \tag{11.14}$$

where $G(\omega)^*$ is the complex conjugate of $G(\omega)$.

Thus if we consider h and g to be digitized, we may approximate the crosscorrelation function as

$$c_{gh}(m) = \frac{1}{N} \sum_{n=0}^{N-1} g(n)\, h(m+n) \tag{11.15}$$

This equation is also known as the biased estimator of the crosscorrelation function. Between the DFTs of the two input discrete signals and the DFT of the biased estimator, we have the relationship

$$Corr\ (k) = G(k)^*\, H(k) \tag{11.16}$$

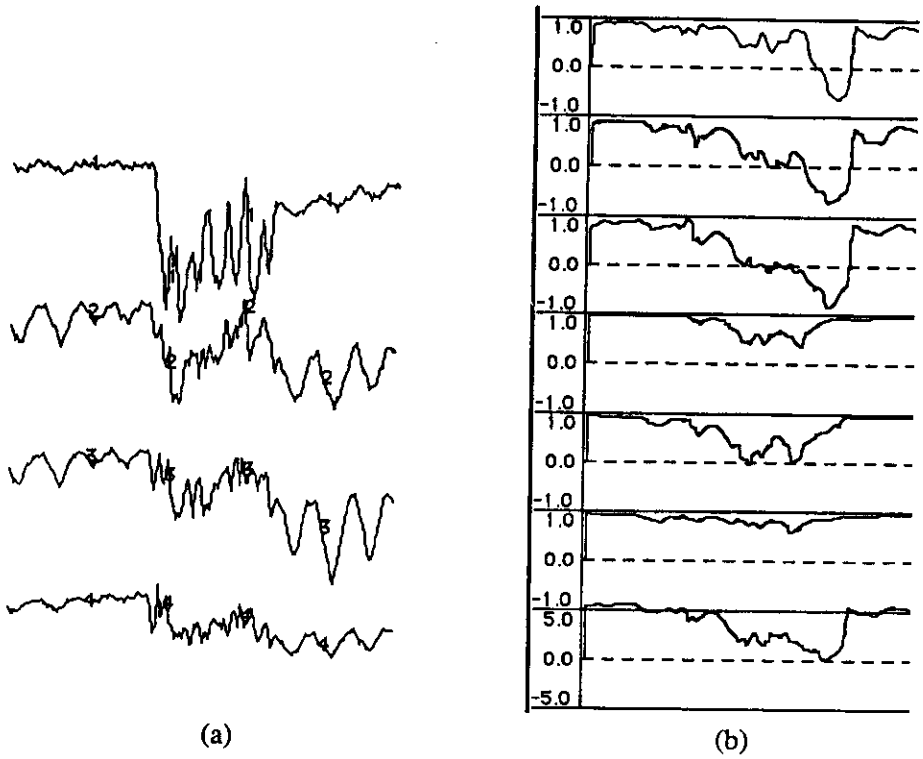


Figure 11.6 Respiratory signals recorded using impedance pneumography. (a) From top to bottom, signals 1, 2, 3, and 4 were simultaneously recorded along the midaxillary line of a human subject using a sampling rate of 5 sps. (b) From top to bottom, the traces represent the crosscorrelation of the channels 1–2, 1–3, 1–4, 2–3, 2–4, 3–4, and the averaged correlation coefficient. The results show, in part, that channels 2 and 3 are highly correlated during normal breathing without motion, but not well correlated during motion without breathing.

Thus, the crosscorrelation of the two discrete signals can also be computed as the inverse DFT of the product of their DFTs. This can be implemented using the FFT and inverse FFT (IFFT) algorithms for increasing the computational speed in the way given by

$$c_{gh}(n) = \text{IFFT}\,(\,\text{FFT}^*(g) * \text{FFT}(h)) \qquad (11.17)$$

11.2.3 Correlation function

The C-language program in Figure 11.7 computes the crosscorrelation function of two 512-point input sequences $x[512]$ and $y[512]$ and stores the output data into $rxy[512]$. The idea of this program was used to implement the C-language function

to compute the crosscorrelation between an ECG signal and a template. In such a case, the array $y[\,]$ must have the same dimension as the template.

```
/* The crosscorrelation function of x[ ] and y[ ] is */
/* output into rxy[ ] */

void corr(float *x,float *y)
{
  int i,m,n;
  float s,s1,s2,xm,ym,t;
  float rxy[512];
  n=512;
  s=s1=s2=xm=ym=0.0;
  for (i=0;i<n;i++){
        xm=xm+x[i];
        ym=ym+y[i];  /* the arithmetic mean of x[ ] and y[ ] */
  }                  /* computed */
  xm=xm/(float)n;
  ym=ym/(float)n;
    for   ( i=0 ; i<n ; i++){
                s1=s1+pow((x[i]-xm),2.0);
                s2=s2+pow((y[i]-ym),2.0);
    }
    s=sqrt(s1*s2);
    for ( m=0 ; m<n ; m++) {
                t=0.0;
                for ( i=0 ;i<n ;i++) {
                       t=t+(x[i]-xm)*(y[(i+m)%n]-ym);
                }
                 rxy[m]=t/s;
    }
}
```

Figure 11.7 C-language function for computing crosscorrelation.

11.3 CONVOLUTION

11.3.1 Convolution in the time domain

It is well known that the passage of a signal through a linear system can be described in the frequency domain by the frequency response of the system. In the time domain, the response of the system to a specific input signal can be described using convolution. Thus, the convolution is the time-domain operation, which is the equivalent of the process of multiplication of the frequency spectra, of the input signal, and of the pulse response of the analyzed system, in order to find the frequency-domain description of the output signal.

If a continuous signal $x(t)$ is applied at the input of an analogous system which has the impulse response $h(t)$, then the equation for the output signal $y(t)$ is known as the convolution equation.

$$y(t) = \int_{-\infty}^{\infty} x(t-\tau)\, h(\tau)\, d\tau \tag{11.18}$$

For discrete signals, the convolution equation becomes

$$y(m) = \sum_{n=0}^{N-1} x(m-n)\, h(n) \tag{11.19}$$

where $x(n)$ is the input signal, $h(n)$ is the sampled impulse response of the system, and $y(n)$ is the output signal. Equation (11.19) can be used for implementing finite impulse response digital filters. In this case, $h(n)$ would be a finite-length sequence which represents, in fact, the coefficients of the FIR filter.

Figure 11.8 shows the input and output signals for a low-pass FIR filter with nine coefficients used for filtering the cardiogenic artifact from respiratory signals recorded using impedance pneumography. The corner frequency of this filter is 0.7 Hz, and the attenuation in the stopband is about 20 dB.

11.3.2 Convolution in the frequency domain

Time-domain convolution is often expressed in a shorthand notation using a '$*$' operator, thus

$$y(t) = x(t) * h(t) = h(t) * x(t)$$

or for discrete signals

$$y(n) = x(n) * h(n) = h(n) * x(n)$$

The following relates the DFTs:

$$Y(k) = X(k)\, H(k)$$

If the FFT is used for computing time-domain convolution, this method is called "fast convolution." We obtain

$$x(n) * h(n) = \text{IFFT}(X(k)\, H(k)) \tag{11.20}$$

where $X(k)$ and $H(k)$ are computed using the FFT algorithm.

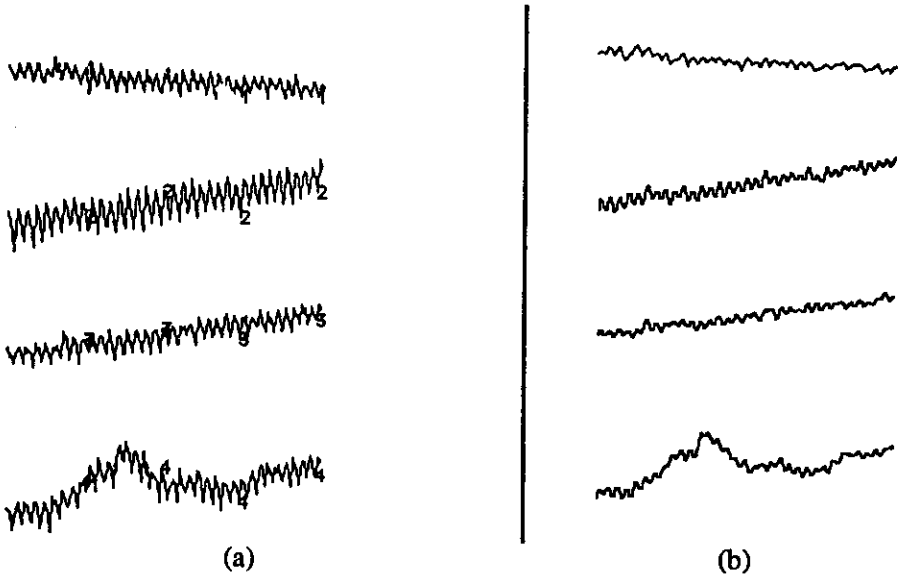


Figure 11.8 Respiratory signals recorded using impedance pneumography. (a) The signals from four different recording channels sampled at a rate of 5 sps. (b) The corresponding signals after being filtered with a 9-coefficient low-pass filter. The cardiogenic artifact, represented by the additive noise seen in (a), is much attenuated by the filtering process.

Convolution in the frequency domain has a similar definition to that for convolution in the time domain. For continuous spectra it is expressed by the integral

$$Y(\omega) = \frac{1}{2\pi} \int_{-\infty}^{\infty} X(\omega - \Omega)\, S(\Omega)\, d\Omega \tag{11.21}$$

The time-domain equivalent for $Y(\omega)$ is

$$y(t) = x(t)\, s(t)$$

Thus, the multiplication of two signals in the time domain is equivalent to the convolution of their Fourier transforms in the frequency domain.

The same principles apply to discrete signals. We know that the discrete signal spectrum has a basic structure defined in the interval $-f_S/2 \leq f \leq +f_S/2$, where f_S is the sampling frequency. Outside this interval, the spectrum of the sampled signal repeats identically, in positive and negative frequencies. If two discrete signals are multiplied together in the time domain the resulting frequency spectrum would also

repeat identically at intervals of sampling frequency. The repeating function would of course be the convolution of the Fourier transforms of the two sampled signals. It is important to note that its form would not be identical with the form of the spectrum of the convolution of two continuous time signals that had the shapes of the envelopes of the sampled signals under consideration.

The concept of convolution in the frequency domain is fundamental to the signal windowing approach. As an example, if $h(n)$ is the impulse response of an ideal low-pass filter with frequency characteristics $H(\omega)$, $h(n)$ will be an infinite length sequence. For example, in order to implement an FIR filter which approximates $H(\omega)$, we must window $h(n)$. Thus, we obtain $h'(n)$ given by

$$h'(n) = w(n)\, h(n) \tag{11.22}$$

where $w(n)$ is the finite-length windowing sequence. We can obtain the Fourier transform of the implemented FIR filter, $H'(\omega)$, using convolution in the frequency domain. Thus, we get

$$H'(\omega) = H(\omega) * W(\omega) \tag{11.23}$$

where $W(\omega)$ is the Fourier transform of the windowing sequence. How well $h(n)$ is approximated by $h'(n)$ depends on the windowing sequence properties.

One way to analyze the performance of the window is to study its Fourier transform. In this approach, one may be interested in analyzing the attenuation in the stopband and the transition width. Figure 11.9 presents the most important parameters, which are mostly used in low-pass filter design, for several types of windows. The designer should make a trade-off between the transition width, the number of coefficients, and the minimum attenuation in the stopband. As shown in Figure 11.9, for middle values of the transition width, the best results are obtained using a Hamming window. If the designer is not interested in the transition width performance, then the best results are obtained using a Blackman window. For a detailed approach of the windowing theory, see Oppenheim and Schafer (1975).

Window	Transition width of the main lobe	Minimum stopband attenuation
Rectangular	2fs/N	−21 dB
Hanning	4fs/N	−44 dB
Hamming	4fs/N	−53 dB
Blackman	6fs/N	−74 dB

Figure 11.9 The performance of different window functions. N represents the number of coefficients used in the function which describes the window, and fs is the sampling frequency.

11.3.3 Convolution function

Figure 11.10 gives a C-language function that computes convolution in the time domain between an input 512-point data sequence $x[512]$ and $h[ncoef]$, which might be the coefficient array of an FIR filter, and stores the output data into $cxy[512]$.

```
/* The convolution between x[ ] and h[ ] is saved into cxy[ ] */

conv(float *x,float *h,int ncoef)
{
  int i,m,n;
  float cxy[512];
  n=512;
    for ( m=0 ; m<n ; m++) {
              cxy[m]=0.0;
              for ( i=0 ;i<ncoef && i<=m ;i++) {
                        cxy[m]+= h[i]*x[m-i];
              }
    }
}
```

Figure 11.10 C-language function for computing convolution.

11.4 POWER SPECTRUM ESTIMATION

11.4.1 Parseval's theorem

Parseval's theorem expresses the conservation of energy principle between the time and frequency domains. For a periodic signal $f(t)$ with the period T, Parseval's theorem tells us how to compute the average power contained in this signal knowing the Fourier series coefficients a_k and b_k, $k = 0, \ldots, \infty$

$$\frac{1}{T}\int_{-T/2}^{T/2} f^2(t)\,dt = a_0^2 + \sum_{k=1}^{\infty}\left(\frac{a_k^2}{2} + \frac{b_k^2}{2}\right) \tag{11.24}$$

For a continuous aperiodic signal, we have a similar relationship between $f(t)$ and its Fourier pair

$$\int_{-\infty}^{\infty} |f(t)|^2\,dt = \frac{1}{2\pi}\int_{-\infty}^{\infty} |F(\omega)|^2\,d\omega \tag{11.25}$$

Similarly, for the Fourier transform of real discrete signals

$$\sum_{-\infty}^{\infty} f^2(n) = \frac{1}{2\pi} \int_{2\pi} |F(\Omega)|^2 \, d\Omega \tag{11.26}$$

In the case of the DFT we assume that the time-domain signal repeats identically with a period of N points, thus the DFT will repeat at intervals of sampling frequency. Parseval's theorem is expressed under these conditions as

$$\sum_{n=0}^{N-1} x^2(n) = \frac{1}{N} \sum_{k=0}^{N-1} |X(k)|^2 \tag{11.27}$$

To estimate the average power of the signal, we compute the mean squared amplitude and make the approximation

$$\frac{1}{T} \int_{-T/2}^{T/2} f^2(t) \, dt \approx \frac{1}{N} \sum_{n=0}^{N-1} f^2(n) \tag{11.28}$$

The method for power spectrum estimation (PSE) used in this section is based on the periodogram concept. Thus, if we sample a function $c(t)$ and use the FFT to compute its DFT, we get

$$C_k = \sum_{n=0}^{N-1} c(n) \, e^{\frac{-jkn2\pi}{N}}$$

Then the periodogram estimate for the power spectrum is defined for $N/2 + 1$ frequencies as

$$P(0) = \frac{1}{N^2} |C_0|^2$$

$$P(k) = \frac{1}{N^2} (|C_k|^2 + |C_{N-k}|^2) \quad \text{for } k = 1, \ldots, N/2 - 1 \tag{11.29}$$

$$P(N/2) = \frac{1}{N^2} |C_{N/2}|^2$$

By Parseval's theorem, we see that the mean squared amplitude is equal to the sum of the $N/2 + 1$ values of P. We must ask ourselves how accurate this estimator is and how it can be improved. The following sections provide two methods for improving the performance of the estimator.

11.4.2 Welch's method of averaging modified periodograms

The periodogram is not a consistent spectral estimator. The variance of the estimate does not tend to go to zero as the record length approaches infinity. One method for improving the estimator proposed by Welch is based on breaking up the N-point data record $x(n)$ into M-point segments $x_k(n)$ that overlap with each other by L samples. If $L = M$ then $N = (K + 1)\,M$ where K is the total number of segments. Subsequently a window function is applied to each segment. Then a periodogram is computed for each windowed segment. Finally, these periodograms are averaged, and the result is scaled to obtain the Welch estimate.

11.4.3 Blackman-Tukey spectral estimate

The Blackman-Tukey estimation method can be implemented in three steps. In the first step, the middle $2M + 1$ samples of the autocorrelation sequence, $\emptyset_{xx}(m)$, where $-M \leq m \leq M$, are estimated from the available N-point data record. The second step is to apply a window to the estimated autocorrelation lags. Finally, the FFT is computed for the windowed autocorrelation estimate to yield the Blackman-Tukey estimate. The parameter M and the window type must be selected in accordance with the specific application.

11.4.4 Compressed spectral array and gray-scale plots

In the compressed spectral array (CSA) method, the resulting spectra are plotted in time sequence (each power spectrum is plotted slightly above the previous spectrum) in order to produce a three-dimensional effect, so that the resultant plots can be easily interpreted. To show this effect on a two-dimensional graphics printout, each subsequent power spectrum, representing a successive time period, is plotted with its origin shifted in both the x and y directions. The more the origins are shifted in the y direction relative to the x direction, the sharper the viewing angle, which allows for better separation of the individual spectra but makes for a greater difficulty in following a frequency component through several time periods. Figure 11.11(a) shows the CSA as a result of spectral analysis of the EGG (electrogastrogram) of a diabetic patient whose time-domain record shows a tachyarrhythmia (Pfister et al., 1988).

Figure 11.11(b) shows the corresponding gray-scale plot. This is a two-dimensional plot with the x axis representing frequency, the y axis representing time, and the intensity of the points representing the spectral power. The darker a point, the greater the spectral power at that point. Each data point in the gray-scale plot is represented by a 5 X 5 matrix of pixels. Each matrix data point can have one of 26 intensity levels, from all pixels off to all pixels on. All other values are scaled proportionally to the maximal level and rounded to an integer value that represents the intensity level. The gray-scale plot does not provide as great a degree of resolution of amplitude as does the CSA method, but it does facilitate observation of frequency shifts.

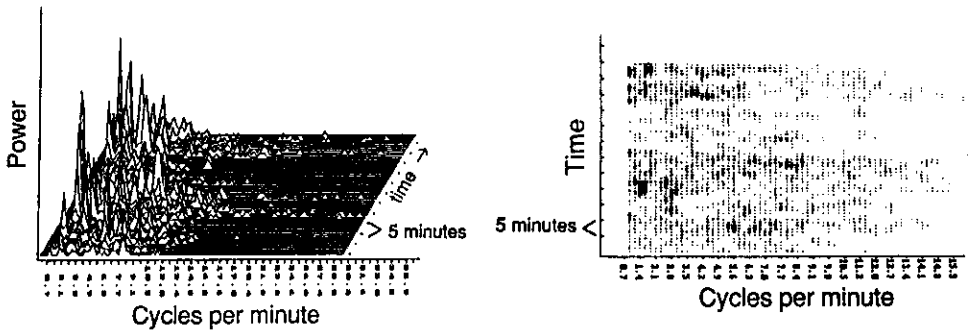


Figure 11.11 Electrogastrogram (EGG) of a diabetic patient. (a) Compressed spectral array (CSA). (b) Gray-scale plot.

11.4.5 Power spectrum function

Figure 11.12 is a C-language function that computes the power spectrum for an N-point data sequence (N = 512). The input data are taken from an input buffer as integers, converted to floating-point format, and then used to compute the power spectrum. The output data are scaled and saved in an output file as integers after conversion from the floating-point format. The program presented is based on Welch's idea of periodogram averaging. Thus, this function divides the input sequence of data into two 256-point segments, windows each segment, and performs the power spectrum estimation for each segment. Finally, the results obtained for each segment are averaged and scaled according to the window effects.

11.5 LAB: FREQUENCY-DOMAIN ANALYSIS OF THE ECG

This lab provides experience in studying the frequency characteristics of signals. Another resource for practice with these techniques is Alkin (1991).

11.5.1 Power spectral analysis of periodic signals

Use `(G)enwave` to create sine wave and square wave signals at several different fundamental frequencies using several different sampling rates. Use the `(P)wr Spect` command to compute and observe the frequency spectrum of each

signal and comment on how sampling rate affects the results. How could you use a power spectrum routine to obtain the actual Fourier coefficients?

11.5.2 Power spectral analysis of an ECG

1. Use the **(P)wr Spect** command to find the frequency corresponding to the main peak in the frequency spectrum of an ECG. Is the result what you expected?
2. Select **ad(V) Ops** from the main menu, and choose a QRS complex using **(T)emplate**. Use the **(P)wr Spect** command to find the spectrum of the zero-padded QRS complex. Is the frequency corresponding to the main peak in the frequency spectrum of the QRS complex what you expected?
3. After selecting a QRS complex template, use the **(W)indow** command to window the selected template, and then run the **(P)wr Spect** command. Document the effects of the various windows.
4. Select a P-QRS-T segment as a template. Using the **(P)wr Spect** command, find the power spectrum estimate for an ECG. Find the frequency corresponding to the main peak in the frequency spectrum of an ECG. How does it differ from the results of part 1?

```
#define WINDOW(j,a,b)(1.0-fabs((((j)-1)-(a))*(b)))/*parzen */
#define SQR(a) (sqrarg=(a),sqrarg*sqrarg)/*modulus of a,squared*/
 ...
 for(j=1;j<=mm;j++) {
          w=WINDOW(j,facm,facp);
            w1[2*j-2] *=w; /* the real part of then data segment
                           is windowed */
            w1[2*j-1]  =0.0; /*the imaginary part is
                                set to zero */
 }
 ...
 fft(w1-1,mm,1);  /*the fft of the windowed signal is performed*/
 /* the power spectrum estimate for the windowed data segment
       is computed */
 p[0] +=(SQR(w1[0]+SQR(w1[1]));
 for (j=1;j<m;j++) {
             j2=2*j;
             p[j] +=(SQR(w1[j2])+SQR(w1[j2+1])+SQR(w1[-j2+4*m])+
                           SQR(w1[-j2+4*m+1]));
 }
 p[m] +=(SQR(w1[mm]+sqr(w1[mm+1]));
 ...
 for (j=0;j<=m;j++)  p[j] /= scale;        /*the PSE is scaled */
                             /* considering the window effects */
```

Figure 11.12 C-language fragment for computing a power spectrum estimation.

1.5.3 Crosscorrelation of the ECG

Select a QRS complex using the `(T)emplate` command. Use the `(C)orrelation` command to correlate this QRS with the ECG. Explain the relationship between the output of the crosscorrelation function and the size of the selected template. What is the time delay between the peak of the crosscorrelation function and the selected QRS? Read a different ECG from a disk file, and crosscorrelate the template with the new ECG. Explain all your observations.

1.6 REFERENCES

Ilkin, O. 1991. *PC-DSP*, Englewood Cliffs, NJ: Prentice Hall.
Oppenheim, A. V., and Schafer, R. W. 1975. *Digital Signal Processing*. Englewood Cliffs, NJ: Prentice Hall.
Tfister, C. J., Hamilton, J. W., Bass, P., Webster, J. G., and Tompkins, W. J. 1988. Use of spectral analysis in detection of frequency differences in electrogastrograms of normal and diabetic subjects. *IEEE Trans. Biomed. Eng.* 935–41.

1.7 STUDY QUESTIONS

1.1 Derive Eqs. (11.24) and (11.27) considering $x(n)$ to be a digital periodic signal containing N samples in a period.
1.2 If $x(n) = 1.0 + \cos(2\pi n/4.0)$, find the DFT of this signal for $n = 0, \ldots, 7$. Write a C-language program that creates a data file with the values of this signal. Read this data file with the UW DigiScope program, and pad it with the corresponding number of zeros. Take the power spectrum of the zero-padded signal using the `(P)wr Spect` command without windowing it, and compare the result with your hand analysis. Explain the differences.
1.3 Create a file that has the values of $x(n) = \sin(2\pi n/512.0)$, $n = 0, \ldots, 511$. Take the power spectrum of this signal without windowing it. Replace the last 10 samples of this signal with zeros. Take the power spectrum of the padded signal with and without a window. Explain the differences.
1.4 Select one period of the signal created in question 11.2 as a template. Crosscorrelate this template with the created signal. Explain the shape of the crosscorrelation function. Compute the amplitude of the peak and compare it with the result of your experiment.
1.5 A 100-Hz-bandwidth ECG signal is sampled at a rate of 500 samples/s. (a) Draw the approximate frequency spectrum of the new digital signal obtained after sampling, and label important points on the axes. (b) On the same graph, draw the approximate spectrum that would be averaged from a set of normal QRS complexes.

12

ECG QRS Detection

Valtino X. Afonso

Over the past few years, there has been an increased trend toward processing of th electrocardiogram (ECG) using microcomputers. A survey of literature in this re search area indicates that systems based on microcomputers can perform neede medical services in an extremely efficient manner. In fact, many systems have a ready been designed and implemented to perform signal processing tasks such a 12-lead off-line ECG analysis, Holter tape analysis, and real-time patient monito ing. All these applications require an accurate detection of the QRS complex of th ECG. For example, arrhythmia monitors for ambulatory patients analyze the EC in real time (Pan and Tompkins, 1985), and when an arrhythmia occurs, the mon tor stores a time segment of the abnormal ECG. This kind of monitor requires a accurate QRS recognition capability. Thus, QRS detection is an important part o many ECG signal processing systems.

This chapter discusses a few of the many techniques that have been developed t detect the QRS complex of the ECG. It begins with a discussion of the powe spectrum of the ECG and goes on to review a variety of QRS detection algorithms

12.1 POWER SPECTRUM OF THE ECG

The power spectrum of the ECG signal can provide useful information about th QRS complex. This section reiterates the notion of the power spectrum presente earlier, but also gives an interpretation of the power spectrum of the QRS comple The power spectrum (based on the FFT) of a set of 512 sample points that contai approximately two heartbeats results in a series of coefficients with a maxim value near a frequency corresponding to the heart rate.

The heart rate can be determined by multiplying together the normalized fre quency and the sampling frequency. We can also get useful information about th frequency spectrum of the QRS complex. In order to obtain this information, th QRS complex of the ECG signal must be selected as a template and zero-padde

prior to the power spectrum analysis. The peak of the frequency spectrum obtained corresponds to the peak energy of the QRS complex.

The ECG waveform contains, in addition to the QRS complex, P and T waves, 60-Hz noise from powerline interference, EMG from muscles, motion artifact from the electrode and skin interface, and possibly other interference from electrosurgery equipment in the operating room. Many clinical instruments such as a cardiotachometer and an arrhythmia monitor require accurate real-time QRS detection. It is necessary to extract the signal of interest, the QRS complex, from the other noise sources such as the P and T waves. Figure 12.1 summarizes the relative power spectra of the ECG, QRS complexes, P and T waves, motion artifact, and muscle noise based on our previous research (Thakor et al., 1983).

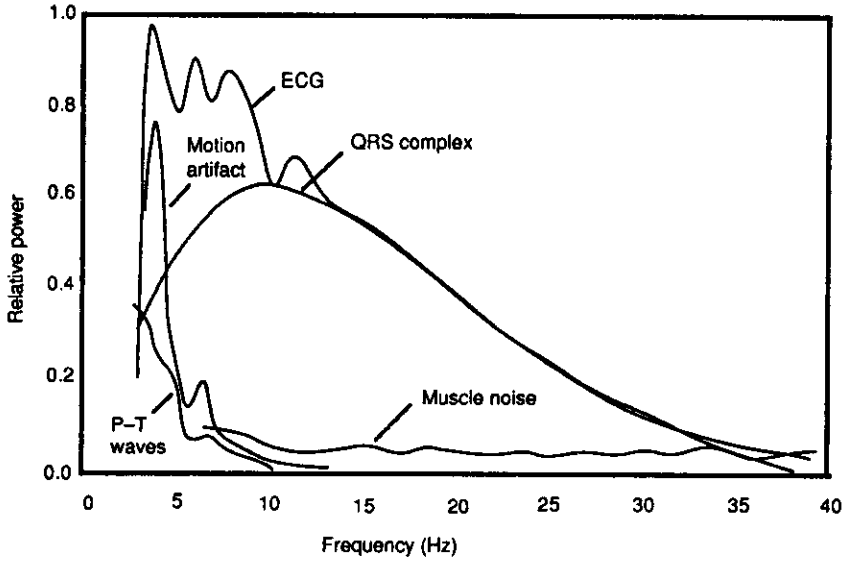


Figure 12.1 Relative power spectra of QRS complex, P and T waves, muscle noise and motion artifacts based on an average of 150 beats.

12.2 BANDPASS FILTERING TECHNIQUES

From the power spectral analysis of the various signal components in the ECG signal, a filter can be designed which effectively selects the QRS complex from the ECG. Another study that we performed examined the spectral plots of the ECG and the QRS complex from 3875 beats (Thakor et al., 1984). Figure 12.2 shows a plot of the signal-to-noise ratio (SNR) as a function of frequency. The study of the

power spectra of the ECG signal, QRS complex, and other noises also revealed that a maximum SNR value is obtained for a bandpass filter with a center frequency of 17 Hz and a Q of 3. Section 12.3 and a laboratory experiment examine the effects of different values of the Q of such a filter.

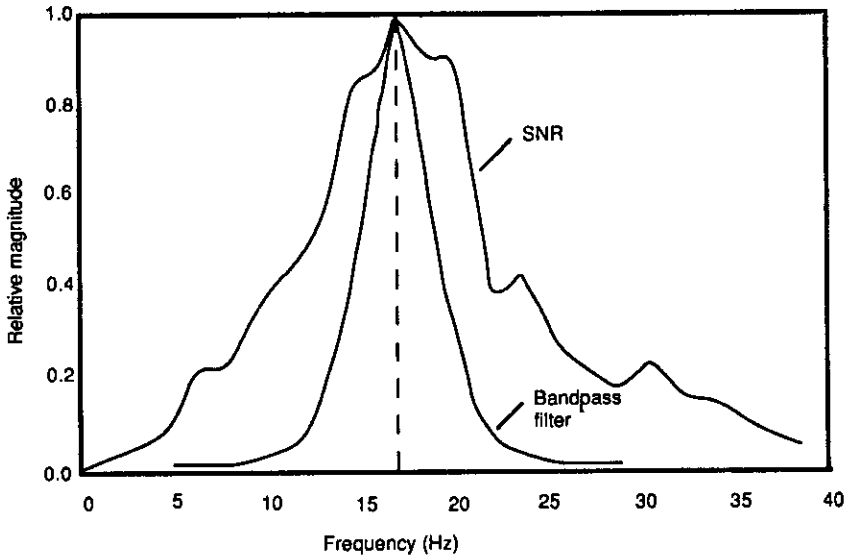


Figure 12.2 Plots of the signal-to-noise ratio (SNR) of the QRS complex referenced to all other signal noise based on 3875 heart beats. The optimal bandpass filter for a cardiotachometer maximizes the SNR.

12.2.1 Two-pole recursive filter

A simple two-pole recursive filter can be implemented in the C language to bandpass the ECG signal. The difference equation for the filter is

$$y(nT) = 1.875y(nT - T) - 0.9219y(nT - 2T) + x(nT) - x(nT - 2T) \qquad (12.1)$$

This filter design assumes that the ECG signal is sampled at 500 samples/s. The values of 1.875 and 0.9219 are approximations of the actual design values of 1.87635 and 0.9216 respectively. Since the coefficients are represented as powers of two, the multiplication operations can be implemented relatively fast using the shift operators in the C language. Figure 12.3 displays the code fragment that implements Eq. (12.1).

```
twoPoleRecursive(int data)
{
      static int xnt, xm1, xm2, ynt, ym1, ym2 = 0;
      xnt = data;

      ynt = (ym1 + ym1 >> 1 + ym1 >> 2 + ym1 >> 3) +
            (ym2 >> 1 + ym2 >> 2 + ym2 >> 3 +
             ym2 >> 5 + ym2 >> 6) + xnt - xm2;

      xm2 = xm1;
      xm1 = xnt;
      xm2 = ym1;
      ym2 = ym1;
      ym1 = ynt;
      return(ynt);
}
```

Figure 12.3 C-language code to implement a simple two-pole recursive filter.

Note that in this code, the coefficients 1.87635 and 0.9216 are approximated by

$$1.875 = 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8}$$

and

$$0.9219 = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{32} + \frac{1}{64}$$

12.2.2 Integer filter

An approximate integer filter can be realized using the general form of the transfer function given in Chapter 7. QRS detectors for cardiotachometer applications frequently bandpass the ECG signal using a center frequency of 17 Hz. The denominator of the general form of the transfer function allows for poles at 60°, 90°, and 120°, and these correspond to center frequencies of a bandpass filter of $T/6$, $T/4$, and $T/3$ Hz, respectively. The desired center frequency can thus be obtained by choosing an appropriate sampling frequency.

Ahlstrom and Tompkins (1985) describe a useful filter for QRS detection that is based on the following transfer function:

$$H(z) = \frac{(1 - z^{-12})2}{(1 - z^{-1} + z^{-2})2} \tag{12.2}$$

This filter has 24 zeros at 12 different frequencies on the unit circle with poles at ±60°. The ECG signal is sampled at 200 sps, and then the turning point algorithm is used to reduce the sampling rate to 100 sps. The center frequency is at 16.67 Hz and the nominal bandwidth is ±8.3 Hz. The duration of the ringing is approxi-

mately 240 ms (the next section explains the effects of different filter Qs). The difference equation to implement this transfer function is

$$y(nT) = 2y(nT-T) - 3y(nT-2T) + 2y(nT-3T) - y(nT-4T) + x(nT) - 2x(nT-12T) + x(nT-24T) \quad (12.3)$$

12.2.3 Filter responses for different values of Q

The value of Q of the bandpass filter centered at $f_c = 17$ Hz determines how well the signal of interest is passed without being attenuated. It is also necessary to increase the SNR of the signal of interest; that is, the QRS complex. The Q of the filter is calculated as

$$Q = \frac{f_c}{BW} \quad (12.4)$$

A value of Q that is too high will result in a very oscillatory response (Thakor et al., 1984). The ripples must die down within 200 ms. This is necessary so that the ripples from one QRS complex do not interfere with the ripples from the next one. With a center frequency of 17 Hz, the maximal permissible Q was found to be 5. Figure 12.4 shows the effect of different values of Q. For a bandpass filter with $f_c = 17$ Hz, a Q value of 5 was found to maximize the SNR (Thakor et al., 1984).

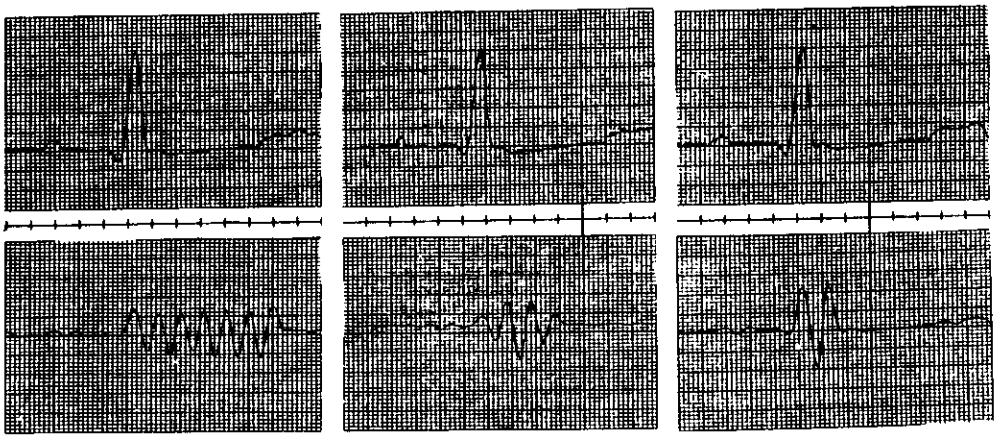


Figure 12.4 Effects of different values of Q. A higher Q results in a oscillatory transient response. (a) $Q = 8$. (b) $Q = 3$. (c) $Q = 1$.

12.3 DIFFERENTIATION TECHNIQUES

Differentiation forms the basis of many QRS detection algorithms. Since it is basically a high-pass filter, the derivative amplifies the higher frequencies characteristic of the QRS complex while attenuating the lower frequencies of the P and T waves.

An algorithm based on first and second derivatives originally developed by Balda et al. (1977) was modified for use in high-speed analysis of recorded ECGs by Ahlstrom and Tompkins (1983). Friesen et al. (1990) subsequently implemented the algorithm as part of a study to compare noise sensitivity among certain types of QRS detection algorithms. Figure 12.5 shows the signal processing steps of this algorithm.

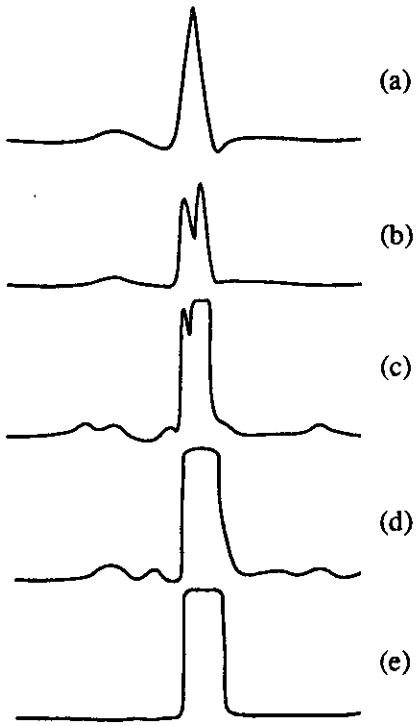


Figure 12.5 Various signal stages in the QRS detection algorithm based on differentiation. (a) Original ECG. (b) Smoothed and rectified first derivative. (c) Smoothed and rectified second derivative. (d) Smoothed sum of (b) and (c). (e) Square pulse output for each QRS complex.

The absolute values of the first and second derivative are calculated from the ECG signal

$$y0(nT) = |x(nT) - x(nT - 2T)| \tag{12.5}$$

$$y_1(nT) = |x(nT) - 2x(nT - 2T) + x(nT - 4T)| \tag{12.6}$$

These two data buffers, $y0(nT)$ and $y_1(nT)$, are scaled and then summed

$$y_2(nT) = 1.3y0(nT) + 1.1y_1(nT) \tag{12.7}$$

The data buffer $y_2(nT)$ is now scanned until a certain threshold is met or exceeded

$$y_2(iT) \geq 1.0 \tag{12.8}$$

Once this condition is met for a data point in $y_2(iT)$, the next eight points are compared to the threshold. If six or more of these eight points meet or exceed the threshold, then the segment might be part of the QRS complex. In addition to detecting the QRS complex, this algorithm has the advantage that it produces a pulse which is proportional in width to the complex. However, a disadvantage is that it is particularly sensitive to higher-frequency noise.

12.4 TEMPLATE MATCHING TECHNIQUES

In this section we discuss techniques for classifying patterns in the ECG signal that are quite related to the human recognition process.

12.4.1 Template crosscorrelation

Signals are said to be correlated if the shapes of the waveforms of two signals match one another. The correlation coefficient is a value that determines the degree of match between the shapes of two or more signals. A QRS detection technique designed by Dobbs et al. (1984) uses crosscorrelation.

This technique of correlating one signal with another requires that the two signals be aligned with one another. In this QRS detection technique, the template of the signal that we are trying to match stores a digitized form of the signal shape that we wish to detect. Since the template has to be correlated with the incoming signal, the signal should be aligned with the template. Dobbs et al. describe two ways of implementing this.

The first way of aligning the template and the incoming signal is by using the fiducial points on each signal. These fiducial points have to be assigned to the

signal by some external process. If the fiducial points on the template and the signal are aligned, then the correlation can be performed.

Another implementation involves continuous correlation between a segment of the incoming signal and the template. Whenever a new signal data point arrives, the oldest data point in time is discarded from the segment (a first-in-first-out data structure). A correlation is performed between this signal segment and the template segment that has the same number of signal points. This technique does not require processing time to assign fiducial points to the signal. The template can be thought of as a window that moves over the incoming signal one data point at a time. Thus, alignment of the segment of the signal of interest must occur at least once as the window moves through the signal.

The value of the crosscorrelation coefficient always falls between +1 and –1. A value of +1 indicates that the signal and the template match exactly. As mentioned earlier, the value of this coefficient determines how well the *shapes* of the two waveforms under consideration match. The magnitude of the actual signal samples does not matter. This shape matching, or recognizing process of QRS complexes, conforms with our natural approach to recognizing signals.

12.4.2 Template subtraction

Figure 12.6 illustrates a template subtraction technique. This is a relatively simple QRS detection technique as compared to the other ones described in this chapter.

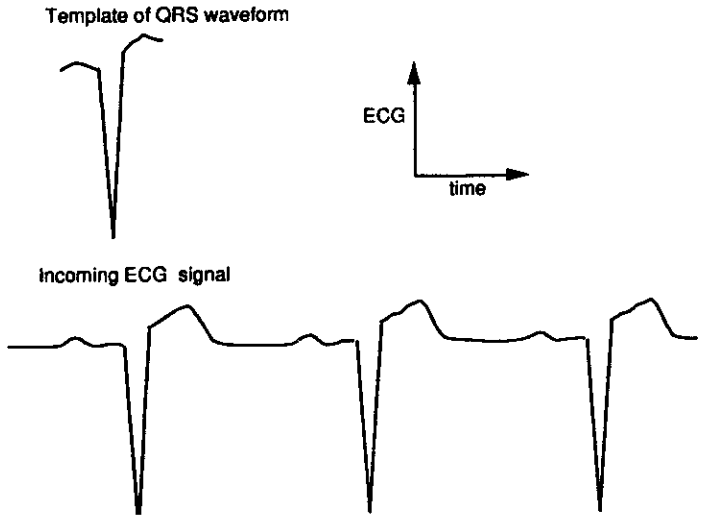


Figure 12.6 In simple template matching, the incoming signal is subtracted, point by point, from the QRS template. If the two waveforms are perfectly aligned, the subtraction results in a zero value.

The algorithm begins by saving a segment of the incoming ECG signal that corresponds to the QRS waveform. This segment or template is then compared with the incoming ECG signal. Each point in the incoming signal is subtracted from the corresponding point in the template. When the template is aligned with a QRS waveform in the signal, the subtraction results in a value very close to zero. This algorithm uses only as many subtraction operations as there are points in the template.

12.4.3 Automata-based template matching

Furno and Tompkins (1982) developed a QRS detector that is based on concepts from automata theory. The algorithm uses some of the basic techniques that are common in many pattern recognition systems. The ECG signal is first reduced into a set of predefined tokens, which represent certain shapes of the ECG waveform.

Figure 12.7 shows the set of tokens that would represent a normal ECG. Then this set of tokens is input to the finite state automaton defined in Figure 12.8. The finite state automaton is essentially a state-transition diagram that can be implemented with IF ... THEN control statements available in most programming languages. The sequence of tokens is fed into the automaton. For example, a sequence of tokens such as *zero*, *normup*, *normdown*, and *normup* would result in the automaton signaling a normal classification for the ECG.

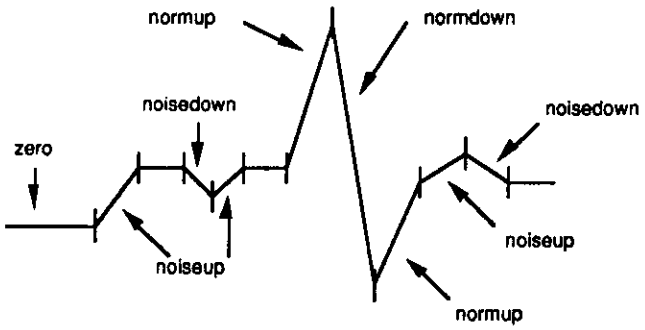


Figure 12.7 Reduction of an ECG signal to tokens.

The sequence of tokens must be derived from the ECG signal data. This is done by forming a sequence of the differences of the input data. Then the algorithm groups together those differences that have the same sign and also exceed a certain predetermined threshold level. The algorithm then sums the differences in each of the groups and associates with each group this sum and the number of differences that are in it.

This QRS detector has an initial learning phase where the program approximately determines the peak magnitude of a normal QRS complex. Then the algorithm detects a normal QRS complex each time there is a deflection in the waveform with a magnitude greater than half of the previously determined peak. The algorithm now teaches the finite state automaton the sequence of tokens that make up a normal QRS complex. The number and sum values (discussed in the preceding paragraph) for a normal QRS complex are now set to a certain range of their respective values in the QRS complex detected.

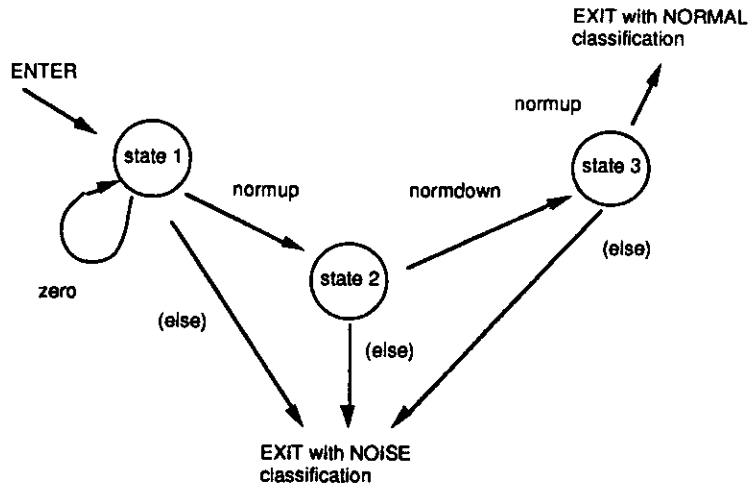


Figure 12.8 State-transition diagram for a simple automaton detecting only normal QRS complexes and noise. The state transition (else) refers to any other token not labeled on a state transition leaving a particular state.

The algorithm can now assign a waveform token to each of the groups formed previously based on the values of the number and the sum in each group of differences. For example, if a particular group of differences has a sum and number value in the ranges (determined in the learning phase) of a QRS upward or downward deflection, then a *normup* or *normdown* token is generated for that group of differences. If the number and sum values do not fall in this range, then a *noiseup* or *noisedown* token is generated. A *zero* token is generated if the sum for a group of differences is zero. Thus, the algorithm reduces the ECG signal data into a sequence of tokens, which can be fed to the finite state automata for QRS detection.

12.5 A QRS DETECTION ALGORITHM

A real-time QRS detection algorithm developed by Pan and Tompkins (1985) was further described by Hamilton and Tompkins (1986). It recognizes QRS complexes based on analyses of the slope, amplitude, and width.

Figure 12.9 shows the various filters involved in the analysis of the ECG signal. In order to attenuate noise, the signal is passed through a bandpass filter composed of cascaded high-pass and low-pass integer filters. Subsequent processes are differentiation, squaring, and time averaging of the signal.

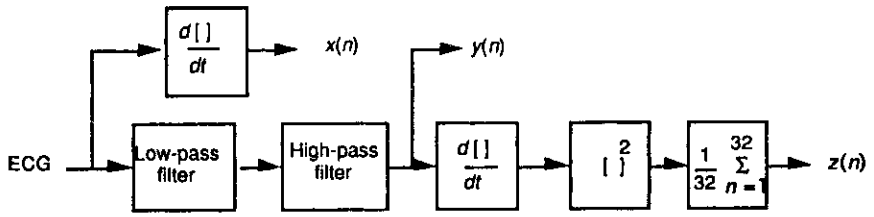


Figure 12.9 Filter stages of the QRS detector. $z(n)$ is the time-averaged signal. $y(n)$ is the bandpassed ECG, and $x(n)$ is the differentiated ECG.

We designed a bandpass filter from a special class of digital filters that require only integer coefficients. This permits the microprocessor to do the signal processing using only integer arithmetic, thereby permitting real-time processing speeds that would be difficult to achieve with floating-point processing. Since it was not possible to directly design the desired bandpass filter with this special approach, the design actually consists of cascaded low-pass and high-pass filter sections. This filter isolates the predominant QRS energy centered at 10 Hz, attenuates the low frequencies characteristic of P and T waves and baseline drift, and also attenuates the higher frequencies associated with electromyographic noise and power line interference.

The next processing step is differentiation, a standard technique for finding the high slopes that normally distinguish the QRS complexes from other ECG waves. To this point in the algorithm, all the processes are accomplished by linear digital filters.

Next is a nonlinear transformation that consists of point-by-point squaring of the signal samples. This transformation serves to make all the data positive prior to subsequent integration, and also accentuates the higher frequencies in the signal obtained from the differentiation process. These higher frequencies are normally characteristic of the QRS complex.

The squared waveform passes through a moving window integrator. This integrator sums the area under the squared waveform over a 150-ms interval, advances

one sample interval, and integrates the new 150-ms window. We chose the window's width to be long enough to include the time duration of extended abnormal QRS complexes, but short enough so that it does not overlap both a QRS complex and a T wave.

Adaptive amplitude thresholds applied to the bandpass-filtered waveform and to the moving integration waveform are based on continuously updated estimates of the peak signal level and the peak noise. After preliminary detection by the adaptive thresholds, decision processes make the final determination as to whether or not a detected event was a QRS complex.

A measurement algorithm calculates the QRS duration as each QRS complex is detected. Thus, two waveform features are available for subsequent arrhythmia analysis—RR interval and QRS duration.

Each of the stages in this QRS detection technique are explained in the following sections. Figure 12.10 is a sampled ECG that will serve as an example input signal for the processing steps to follow.

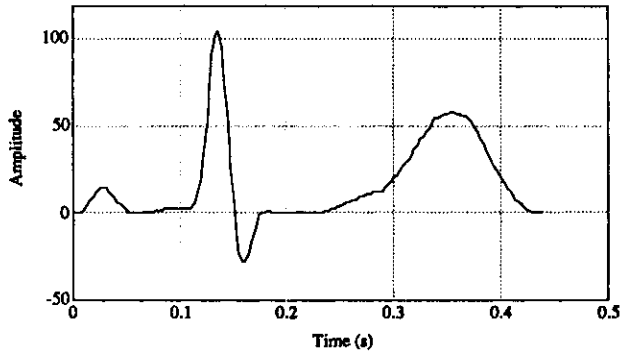


Figure 12.10 Electrocardiogram sampled at 200 samples per second.

12.5.1 Bandpass integer filter

The bandpass filter for the QRS detection algorithm reduces noise in the ECG signal by matching the spectrum of the average QRS complex. Thus, it attenuates noise due to muscle noise, 60-Hz interference, baseline wander, and T-wave interference. The passband that maximizes the QRS energy is approximately in the 5–15 Hz range, as discussed in section 12.1. The filter implemented in this algorithm is a recursive integer filter in which poles are located to cancel the zeros on the unit circle of the z plane. A low-pass and a high-pass filter are cascaded to form the bandpass filter.

Low-pass filter

The transfer function of the second-order low-pass filter is

$$H(z) = \frac{(1 - z^{-6})2}{(1 - z^{-1})2} \tag{12.9}$$

The difference equation of this filter is

$$y(nT) = 2y(nT - T) - y(nT - 2T) + x(nT) - 2x(nT - 6T) + x(nT - 12T) \tag{12.10}$$

The cutoff frequency is about 11 Hz, the delay is five samples (or 25 ms for a sampling rate of 200 sps), and the gain is 36. Figure 12.11 displays the C-language program that implements this low-pass filter. In order to avoid saturation, the output is divided by 32, the closest integer value to the gain of 36 that can be implemented with binary shift arithmetic.

```
int LowPassFilter(int data)
{
      static int y1 = 0, y2 = 0, x[26], n = 12;
      int y0;

      x[n] = x[n + 13] = data;
      y0 = (y1 << 1) - y2 + x[n] - (x[n + 6] << 1) + x[n + 12];
      y2 = y1;
      y1 = y0;
      y0 >>= 5;
      if(--n < 0)
            n = 12;

      return(y0);
}
```

Figure 12.11 C-language program to implement the low-pass filter.

Figure 12.12 shows the performance details of the low-pass filter. This filter has purely linear phase response. Note that there is more than 35-dB attenuation of the frequency corresponding to $0.3\,f/f_S$. Since the sample rate is 200 sps for these filters, this represents a frequency of 60 Hz. Therefore, power line noise is significantly attenuated by this filter. Also all higher frequencies are attenuated by more than 25 dB.

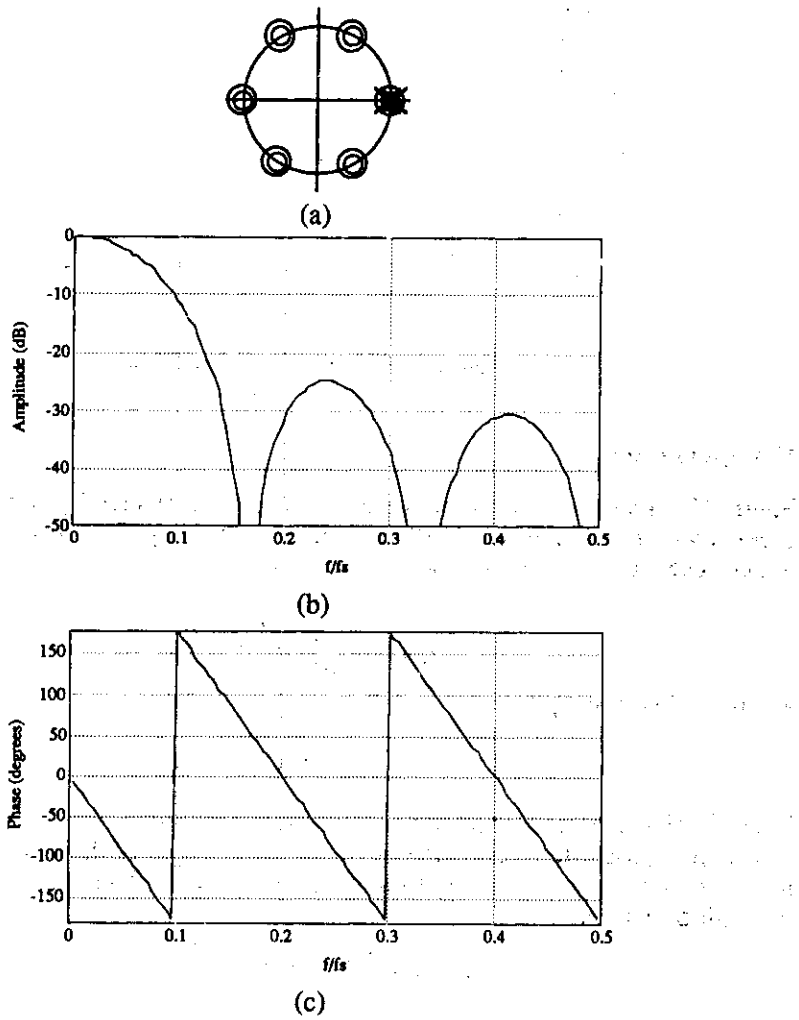


Figure 12.12 Low-pass filter. a) Pole-zero plot. b) Amplitude response. c) Phase response.

Figure 12.13 shows the ECG of Figure 12.10 after processing with the low-pass filter. The most noticeable result is the attenuation of the higher frequency QRS complex. Any 60-Hz noise or muscle noise present would have also been significantly attenuated.

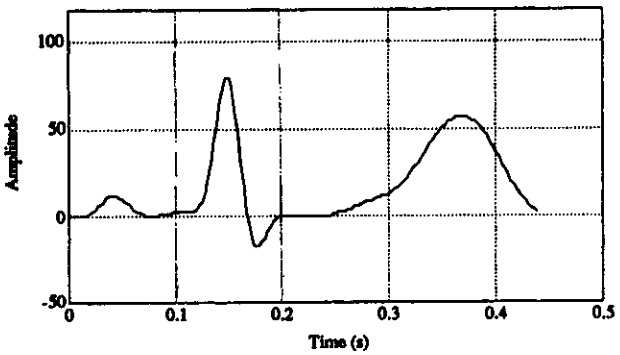


Figure 12.13 Low-pass filtered ECG.

High-pass filter

Figure 12.14 shows how the high-pass filter is implemented by subtracting a first-order low-pass filter from an all-pass filter with delay. The low-pass filter is an integer-coefficient filter with the transfer function

$$H_{lp}(z) = \frac{Y(z)}{X(z)} = \frac{1 - z^{-32}}{1 - z^{-1}} \tag{12.11}$$

and the difference equation

$$y(nT) = y(nT - T) + x(nT) - x(nT - 32T) \tag{12.12}$$

This filter has a dc gain of 32 and a delay of 15.5 samples.

The high-pass filter is obtained by dividing the output of the low-pass filter by its dc gain and then subtracting from the original signal. The transfer function of the high-pass filter is

$$H_{hp}(z) = \frac{P(z)}{X(z)} = z^{-16} - \frac{H_{lp}(z)}{32} \tag{12.13}$$

The difference equation for this filter is

$$p(nT) = x(nT - 16T) - \frac{1}{32}[y(nT - T) + x(nT) - x(nT - 32T)] \tag{12.14}$$

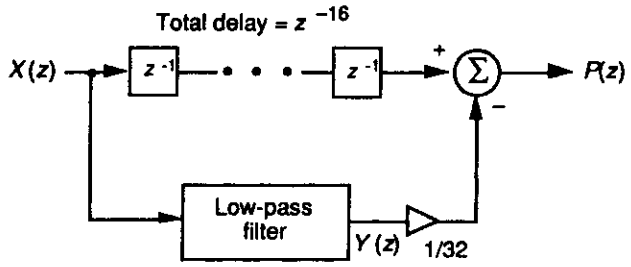


Figure 12.14 The high-pass filter is implemented by subtracting a low-pass filter from an all-pass filter with delay.

The low cutoff frequency of this filter is about 5 Hz, the delay is about 16*T* (or 80 ms), and the gain is 1. Figure 12.15 shows the C-language program that implements this high-pass filter.

```
int HighPassFilter(int data)
{
      static int y1 = 0, x[66], n = 32;
      int y0;

      x[n] = x[n + 33] = data;
      y0 = y1 + x[n] - x[n + 32];
      y1 = y0;
      if(--n < 0)
            n = 32;

      return(x[n + 16] - (y0 >> 5));
}
```

Figure 12.15 C-language program to implement the high-pass filter.

Figure 12.16 shows the performance characteristics of the high-pass filter. Note that this filter also has purely linear phase response.

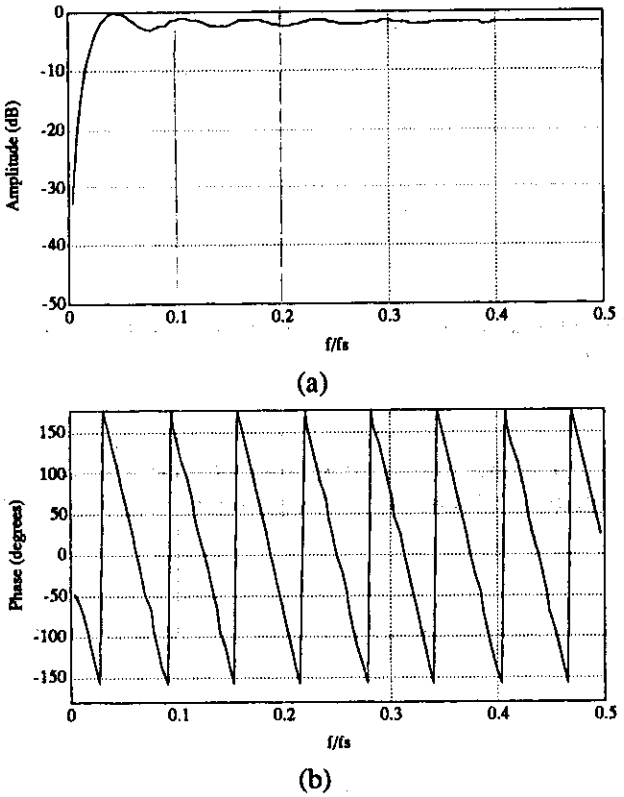


Figure 12.16 High-pass filter. a) Amplitude response. b) Phase response.

Figure 12.17 shows the amplitude response of the bandpass filter which is com posed of the cascade of the low-pass and high-pass filters. The center frequency o the passband is at 10 Hz. The amplitude response of this filter is designed to ap proximate the spectrum of the average QRS complex as illustrated in Figure 12.1 Thus this filter optimally passes the frequencies characteristic of a QRS comple while attenuating lower and higher frequency signals.

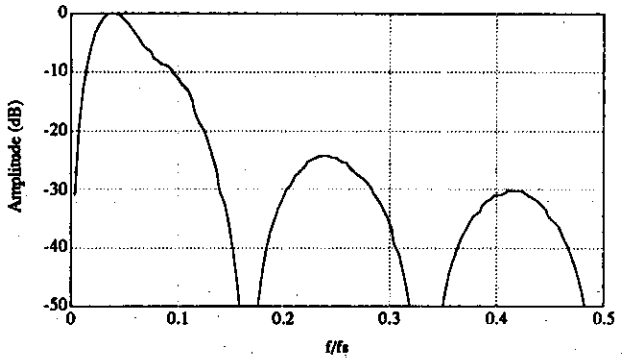


Figure 12.17 Amplitude response of bandpass filter composed of low-pass and high-pass filters.

Figure 12.18 is the resultant signal after the ECG of Figure 12.10 passes through the bandpass filter. Note the attenuation of the T wave due to the high-pass filter.

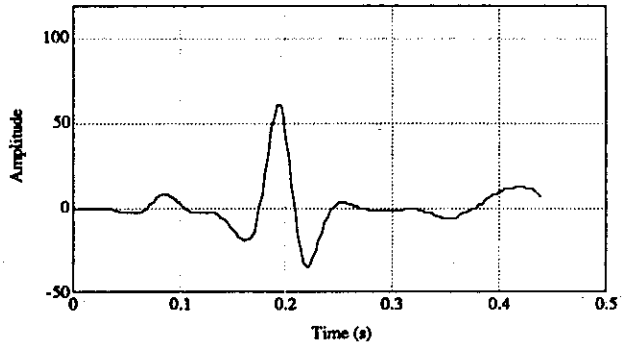


Figure 12.18 Bandpass-filtered ECG.

12.5.2 Derivative

After the signal has been filtered, it is then differentiated to provide information about the slope of the QRS complex. A five-point derivative has the transfer function

$$H(z) = 0.1\,(2 + z^{-1} - z^{-3} - 2z^{-4}) \qquad (12.15)$$

This derivative is implemented with the difference equation

$$y(nT) = \frac{2x(nT) + x(nT - T) - x(nT - 3T) - 2x(nT - 4T)}{8} \qquad (12.16)$$

The fraction 1/8 is an approximation of the actual gain of 0.1. Throughout thes filter designs, we approximate parameters with power-of-two values to facilitat real-time operation. These power-of-two calculations are implemented in the C language by shift-left or shift-right operations.

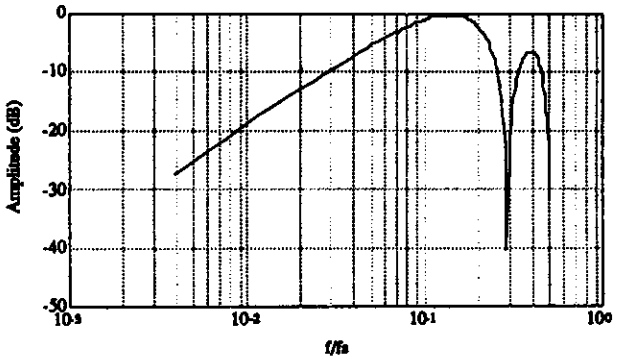
This derivative approximates the ideal derivative in the dc through 30-H frequency range. The derivative has a filter delay of $2T$ (or 10 ms). Figure 12.1 shows the C-language program for implementing this derivative.

```
int Derivative(int data)
{
      int y, i;
      static int x_derv[4];

      /*y = 1/8 (2x(nT) + x(nT - T) - x(nT - 3T) - 2x(nT - 4T))*

      y = (data << 1) + x_derv[3] - x_derv[1] - (x_derv[0] << 1)

      y >>= 3;
      for (i = 0; i < 3; i++)
            x_derv[i] = x_derv[i + 1];
      x_derv[3] = data;

      return(y);
}
```

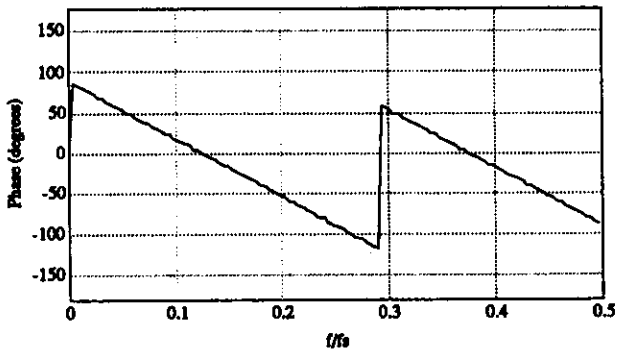
Figure 12.19 C-language program to implement the derivative.

Figure 12.20 shows the performance characteristics of this derivative implemer tation. Note that the amplitude response approximates that of a true derivative up t about 20 Hz. This is the important frequency range since all higher frequencies ar significantly attenuated by the bandpass filter.

Figure 12.21 is the resultant signal after passing through the cascade of filter including the differentiator. Note that P and T waves are further attenuated whil the peak-to-peak signal corresponding to the QRS complex is further enhanced.



(a)



(b)

Figure 12.20 Derivative. a) Amplitude response. b) Phase response.

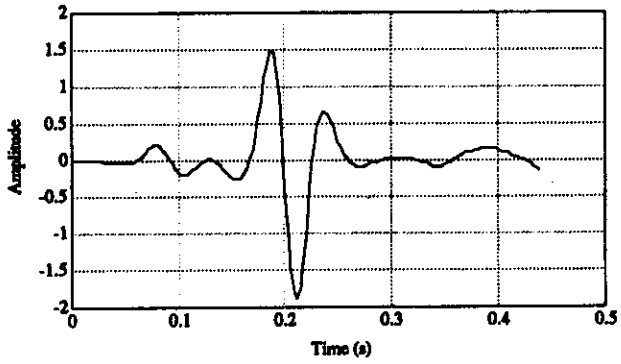


Figure 12.21 ECG after bandpass filtering and differentiation.

12.5.3 Squaring function

The previous processes and the moving-window integration, which is explained in the next section, are linear processing parts of the QRS detector. The squaring function that the signal now passes through is a nonlinear operation. The equation that implements this operation is

$$y(nT) = [x(nT)]^2 \tag{12.17}$$

This operation makes all data points in the processed signal positive, and it amplifies the output of the derivative process nonlinearly. It emphasizes the higher frequencies in the signal, which are mainly due to the QRS complex. A fact to note in this operation is that the output of this stage should be hardlimited to a certain maximum level corresponding to the number of bits used to represent the data type of the signal. Figure 12.22 shows the results of this processing for our sample ECG.

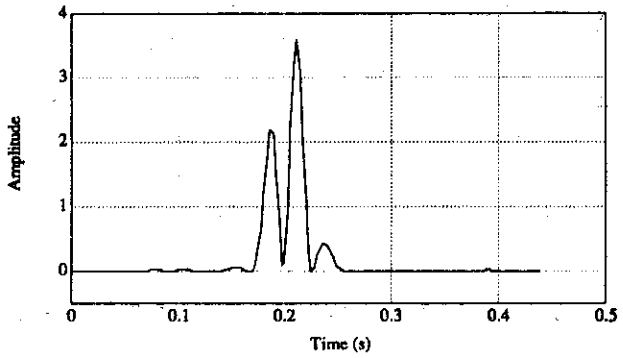


Figure 12.22 ECG signal after squaring function.

12.5.4 Moving window integral

The slope of the R wave alone is not a guaranteed way to detect a QRS event. Many abnormal QRS complexes that have large amplitudes and long durations (not very steep slopes) might not be detected using information about slope of the R wave only. Thus, we need to extract more information from the signal to detect a QRS event.

Moving window integration extracts features in addition to the slope of the R wave. It is implemented with the following difference equation:

$$y(nT) = \frac{1}{N}[x(nT - (N-1)T) + x(nT - (N-2)T) + \ldots + x(nT)] \tag{12.18}$$

where N is the number of samples in the width of the moving window. The value of this parameter should be chosen carefully.

Figure 12.23 shows the output of the moving window integral for the sample ECG of Figure 12.10. Figure 12.24 illustrates the relationship between the QRS complex and the window width. The width of the window should be approximately the same as the widest possible QRS complex. If the size of the window is too large, the integration waveform will merge the QRS and T complexes together. On the other hand, if the size of the window is too small, a QRS complex could produce several peaks at the output of the stage. The width of the window should be chosen experimentally. For a sample rate of 200 sps, the window chosen for this algorithm was 30 samples wide (which corresponds to 150 ms).

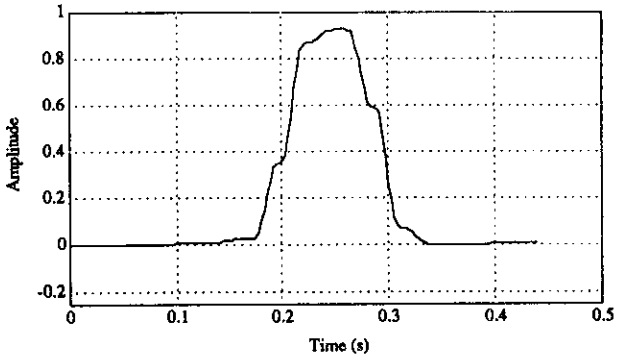


Figure 12.23 Signal after moving window integration.

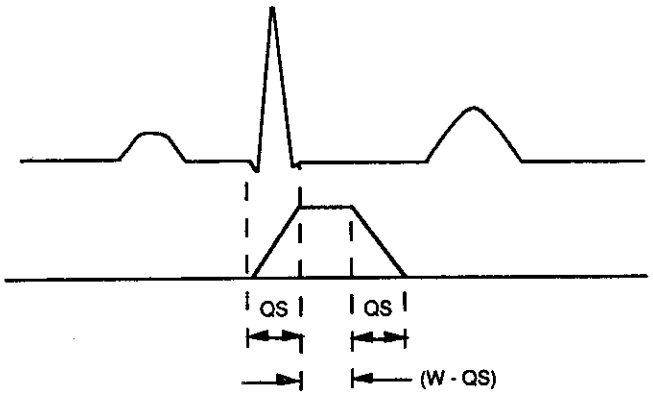


Figure 12.24 The relationship of a QRS complex to the moving integration waveform. (a) ECG signal. (b) Output of moving window integrator. QS: QRS width. W: width of the integrator window.

Figure 12.25 shows the C-language program that implements the moving window integration.

```
int MovingWindowIntegral(int data)
{
      static int x[32], ptr = 0;
      static long sum = 0;
      long ly;
      int y;

      if(++ptr == 32)
            ptr = 0;
      sum -= x[ptr];
      sum += data;
      x[ptr] = data;
      ly = sum >> 5;
      if(ly > 32400)       /*check for register overflow*/
            y = 32400;
      else
            y = (int) ly;

      return(y);
}
```

Figure 12.25 C-language program to implement the moving window integration.

Figure 12.26 shows the appearance of some of the filter outputs of this algorithm. Note the processing delay between the original ECG complexes and corresponding waves in the moving window integral signal.

12.5.5 Thresholding

The set of thresholds that Pan and Tompkins (1985) used for this stage of the QRS detection algorithm were set such that signal peaks (i.e., valid QRS complexes) were detected. Signal peaks are defined as those of the QRS complex, while noise peaks are those of the T waves, muscle noise, etc. After the ECG signal has passed through the bandpass filter stages, its signal-to-noise ratio increases. This permits the use of thresholds that are just above the noise peak levels. Thus, the overall sensitivity of the detector improves.

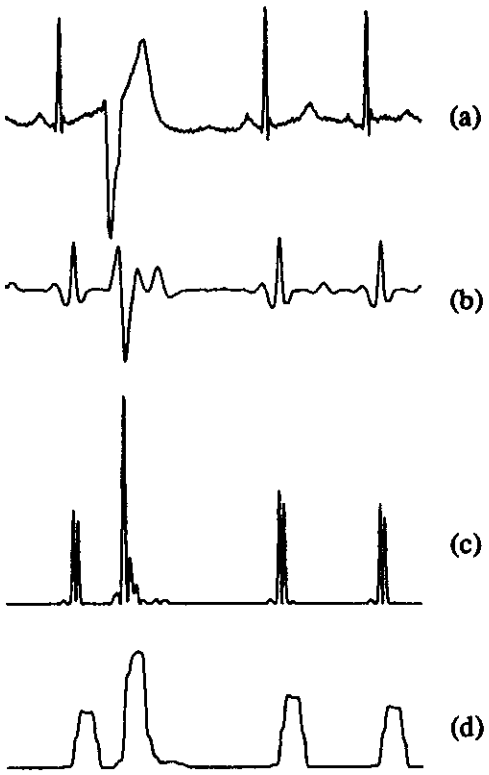


Figure 12.26 QRS detector signals. (a) Unfiltered ECG. (b) Output of bandpass filter. (c) Output after bandpass, differentiation, and squaring processes. (d) Final moving-window integral.

Two sets of thresholds are used, each of which has two threshold levels. The set of thresholds that is applied to the waveform from the moving window integrator is

$$SPKI = 0.125\ PEAKI + 0.875\ SPKI \qquad \text{if } PEAKI \text{ is the signal peak}$$

$$NPKI = 0.125\ PEAKI + 0.875\ NPKI \qquad \text{if } PEAKI \text{ is the noise peak}$$

$$THRESHOLD\ I1 = NPKI + 0.25\ (SPKI - NPKI)$$

$$THRESHOLD\ I2 = 0.5\ THRESHOLD\ I1$$

All the variables in these equations refer to the signal of the integration waveform and are described below:

PEAKI is the overall peak.
SPKI is the running estimate of the signal peak.
NPKI is the running estimate of the noise peak.
THRESHOLD I1 is the first threshold applied.
THRESHOLD I2 is the second threshold applied.

A peak is determined when the signal changes direction within a certain time interval. Thus, *SPKI* is the peak that the algorithm has learned to be that of the QRS complex, while *NPKI* peak is any peak that is not related to the signal of interest. As can be seen from the equations, new values of thresholds are calculated from previous ones, and thus the algorithm adapts to changes in the ECG signal from a particular person.

Whenever a new peak is detected, it must be categorized as a noise peak or a signal peak. If the peak level exceeds *THRESHOLD I1* during the first analysis of the signal, then it is a QRS peak. If searchback technique (explained in the next section) is used, then the signal peak should exceed *THRESHOLD I2* to be classified as a QRS peak. If the QRS complex is found using this second threshold level, then the peak value adjustment is twice as fast as usual:

$$SPKI = 0.25\,PEAKI + 0.75\,SPKI$$

The output of the final filtering stages, after the moving window integrator, must be detected for peaks. A peak detector algorithm finds peaks and a detection algorithm stores the maximum levels at this stage of the filtered signal since the last peak detection. A new peak is defined only when a level that is less than half the height of the peak level is reached. Figure 12.27 illustrates that this occurs halfway down the falling edge of the peak (Hamilton and Tompkins, 1986).

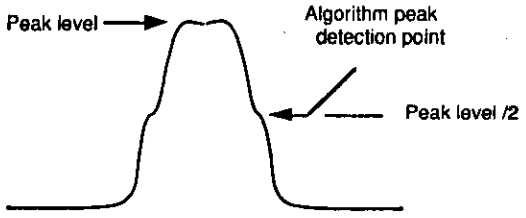


Figure 12.27 Output after the moving window integrator, with peak detection point.

12.5.6 Searchback technique

To implement the searchback technique, this algorithm maintains two RR-interval averages. One average, *RR AVERAGE1*, is that of the eight most recent heartbeats. The other average, *RR AVERAGE2*, is the average of the eight most recent beats which had *RR* intervals that fell within a certain range.

$$RR\ AVERAGE1 = 0.125\ (RR_{n-7} + RR_{n-6} + \ldots + RR_n)$$
$$RR\ AVERAGE2 = 0.125\ (RR'_{n-7} + RR'_{n-6} + \ldots + RR'_n)$$

The RR'_n values are the *RR* intervals that fell within the following limits:

$$RR\ LOW\ LIMIT = 92\% \times RR\ AVERAGE2$$
$$RR\ HIGH\ LIMIT = 116\% \times RR\ AVERAGE2$$

Whenever the QRS waveform is not detected for a certain interval, *RR MISSED LIMIT*, then the QRS is the peak between the established thresholds mentioned in the previous section that are applied during searchback.

$$RR\ MISSED\ LIMIT = 166\ \% \times RR\ AVERAGE2$$

The heart rate is said to be normal if each of the eight most recent RR intervals are between the limits established by *RR LOW LIMIT* and *RR HIGH LIMIT*.

12.5.7 Performance measurement

We tested the performance of the algorithm on the 24-hour annotated MIT/BIH database, which is composed of half-hour recordings of ECGs of 48 ambulatory patients (see references, MIT/BIH ECG database). This database, available on CD ROM, was developed by Massachusettsa Institute of Technology and Beth Israel Hospital. The total error in analyzing about 116,000 beats is 0.68 percent, corresponding to an average error rate of 33 beats per hour. In fact, much of the error comes from four particular half-hour tape segments (i.e., two hours of data from the total database).

Figure 12.28 shows the effect of excluding the four most problematic half-hour tapes from the overall results. Notice that the false-positive errors decrease much more than do the false negatives. This difference indicates that this algorithm is more likely to misclassify noise as a QRS complex than it is to miss a real event. Elimination of these four half-hour tape segments reduces the error rate below 10 beats per hour. Another available ECG database was developed by the American Heart Association (see references, AHA ECG database).

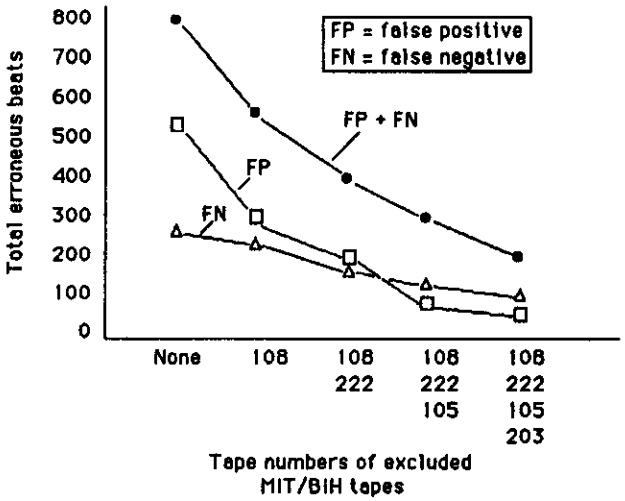


Figure 12.28 Performance of QRS detection software. The total error of the QRS detection algorithm can be substantially reduced by selectively eliminating problem tapes in the database.

12.6 LAB: REAL-TIME ECG PROCESSING ALGORITHM

This lab lets you "look inside" the inner workings of the algorithm QRS detection algorithm developed by Pan and Tompkins (1985) that is described in section 12.5. Load UW DigiScope, select **ad(V) Ops**, then **(Q)RS detect**.

12.6.1 QRS detector algorithm processing steps

Observe the output of each of the stages in the QRS detector. Sketch or print one cycle of the original ECG signal and the outputs of the low-pass, bandpass, derivative, squaring, and moving window integrator stages. Note the filter delay at each of these stages.

12.6.2 Effect of the value of the Q of a filter on QRS detection

Implement several two-pole recursive filters with 17-Hz center frequencies to observe the effects of different values of Q on the ECG, as in section 12.2.1. What value of r produces the most desirable response for detecting the QRS complex?

12.6.3 Integer filter processing of the ECG

Use (G) **enwave** to generate an ECG signal sampled at 100 Hz. Process this signal with a filter having the following difference equation.

$$y(nT) = 2y(nT - T) - 3y(nT - 2T) + 2y(nT - 3T) - y(nT - 4T) + x(nT) - 2x(nT - 12T) + x(nT - 24T)$$

Observe the output and note the duration of the ringing.

12.7 REFERENCES

AHA ECG database. Available from Emergency Care Research Institute, 5200 Butler Pike, Plymouth Meeting, PA 19462.

Ahlstrom, M. L. and Tompkins, W. J. 1983. Automated high-speed analysis of Holter tapes with microcomputers. *IEEE Trans. Biomed. Eng.*, **BME-30**: 651–57.

Ahlstrom, M. L. and Tompkins, W. J. 1985. Digital filters for real-time ECG signal processing using microprocessors. *IEEE Trans. Biomed. Eng.*, **BME-32**: 708–13.

Balda R. A., Diller, G., Deardorff, E., Doue, J., and Hsieh, P. 1977. The HP ECG analysis program. *Trends in Computer-Processed Electrocardiograms*. J. H. vanBemnel and J. L. Willems, (eds.) Amsterdam, The Netherlands: North Holland, 197–205.

Dobbs, S. E., Schmitt, N. M., Ozemek, H. S. 1984. QRS detection by template matching using real-time correlation on a microcomputer. *Journal of Clinical Engineering*, **9**: 197–212.

Friesen, G. M., Jannett, T. C., Jadallah, M. A., Yates, S. L., Quint, S. R., Nagle, H. T. 1990. A comparison of the noise sensitivity of nine QRS detection algorithms. *IEEE Trans. Biomed. Eng.*, **BME-37**: 85–97.

Furno, G. S. and Tompkins, W. J. 1982. QRS detection using automata theory in a battery-powered microprocessor system. *IEEE Frontiers of Engineering in Health Care*, **4**: 155–58.

Hamilton, P. S. and Tompkins, W. J. 1986. Quantitative investigation of QRS detection rules using the MIT/BIH arrhythmia database. *IEEE Trans. Biomed. Eng.* **BME-33**: 1157–65.

MIT/BIH ECG database. Available from: MIT-BIH Database Distribution, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Room 20A-113, Cambridge, MA 02139.

Pan, J. and Tompkins, W. J. 1985. A real-time QRS detection algorithm. *IEEE Trans. Biomed. Eng.* **BME-32**: 230–36,.

Thakor, N. V., Webster, J. G., and Tompkins, W. J. 1983. Optimal QRS detector. *Medical and Biological Engineering*, 343–50.

Thakor, N. V., Webster, J. G., and Tompkins, W. J. 1984. Estimation of QRS complex power spectra for design of a QRS filter. *IEEE Trans. Biomed. Eng.*, **BME-31**: 702–05.

12.8 STUDY QUESTIONS

12.1 How can ectopic beats be detected using the automata approach to QRS detection?

12.2 How can QRS complexes in abnormal waveforms be detected using the crosscorrelation method?

12.3 In the moving window integrator of the algorithm in section 12.5, how should the width of the window be chosen? What are the effects of choosing a window width that is too large or too small?

12.4 In the QRS detection algorithm explained in section 12.5, how should the first threshold in each set of thresholds be changed so as to increase the detection sensitivity of irregular heart rates?

12.5 What are the effects of bandpass filter Q on the QRS-to-noise ratio in the ECG?

12.6 Design an algorithm that obtains the fiducial point on the ECG.

12.7 As an implementation exercise write a program using the C language, to detect QRS complexes in the ECG signal using any of the techniques described in this chapter.

12.8 Suggest a QRS detection algorithm, based on some of the techniques explained in this chapter or in other related literature, that can detect QRS complexes from the ECG in real time.

12.9 Experiments to determine the frequency characteristics of the average QRS complex have shown that the largest spectral energy of the QRS complex occurs at approximately what frequency?

12.10 A filter with the difference equation, $y(nT) = (y(nT - T))^2 + x(nT)$, is *best* described as what traditional filter type?

12.11 The center frequency of the optimal QRS bandpass filter is not at the location of the maximal spectral energy of the QRS complex. (a) What function is maximized for the optimal filter? (b) What is the center frequency of the optimal QRS filter for cardiotachometers? (c) If this filter has the proper center frequency and a $Q = 20$, will it work properly? If not, why not?

12.12 In addition to heart rate information, what QRS parameter is provided by the QRS detection algorithm that is based on the first and second derivatives?

12.13 The derivative algorithm used in a real-time QRS detector has the difference equation: $y(nT) = 2x(nT) + x(nT - T) - x(nT - 3T) - 2x(nT - 4T)$. (a) Draw its block diagram. (b) What is its output sequence in response to a *unit step* input? Draw the output waveform.

12.14 Write the equations for the amplitude and phase responses of the derivative algorithm used in a real-time QRS detector that has the transfer function

$$H(z) = \frac{-2z^{-2} - z^{-1} + z^{1} + 2z^{2}}{8}$$

12.15 A moving window integrator integrates over a window that is 30 samples wide and has an overall amplitude scale factor of 1/30. If a unit impulse (i.e., 1, 0, 0, 0, ...) is applied to the input of this integrator, what is the output sequence?

12.16 A moving window integrator is five samples wide and has a unity amplitude scale factor. A pacemaker pulse is described by the sequence: (1, 1, 1, 1, 0, 0, 0, 0, ...). Application of this pulse to the input of the moving window integrator will produce what output sequence?

12.17 The transfer function of a filter used in a real-time QRS detection algorithm is

$$H(z) = \frac{(1 - z^{-6})^2}{(1 - z^{-1})^2}$$

For a sample rate of 200 sps, this filter *eliminates* input signals of what frequencies?