
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

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

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.

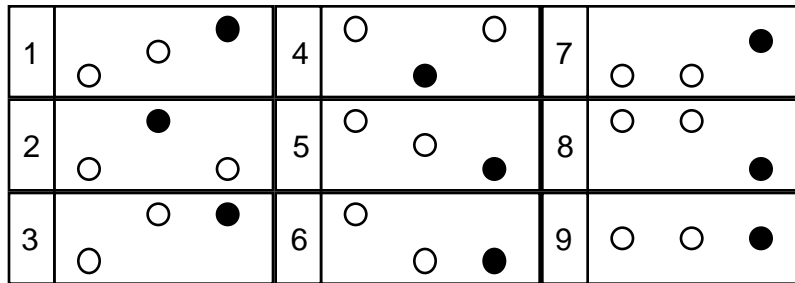
TP is based on the concept that ECG signals are normally oversampled at four or five times faster than the highest frequency present. For example, an ECG used in monitoring may have a bandwidth of 50 Hz and be sampled at 200 sps in order to easily visualize the higher-frequency attributes of the QRS 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

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



(a)

Pattern	$s_1 = \text{sign}(X_1 - X_0)$	$s_2 = \text{sign}(X_2 - X_1)$	$\text{NOT}(s_1) \text{ OR } (s_1 + s_2)$	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 $s_1 = \text{sign}(X_1 - X_0)$ and $s_2 = \text{sign}(X_2 - X_1)$, 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, $\text{NOT}(c) = 1$ if $c = 0$; otherwise $\text{NOT}(c) = 0$. Also logical OR means that $(a \text{ OR } b) = 0$ only if a and b are both 0. Thus, we retain X_1 only if $\{\text{NOT}(s_1) \text{ OR } (s_1 + s_2)\}$ 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.

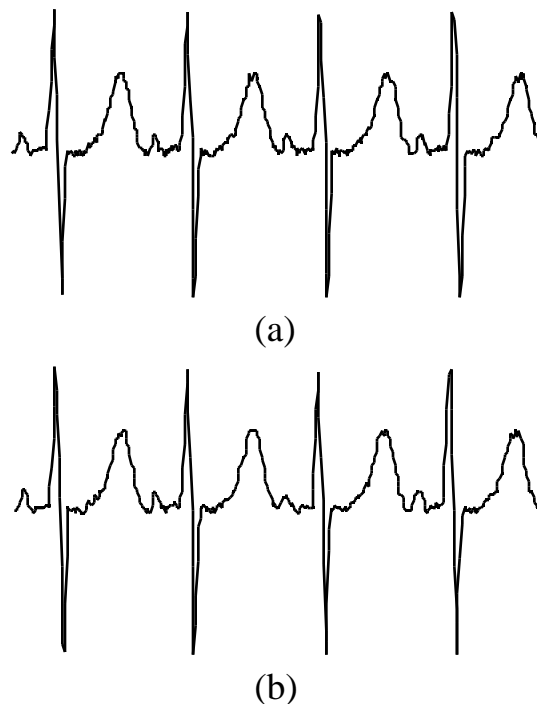


Figure 10.3 An example of the application of the TP algorithm. (a) Original waveform generated by the UW DigiScope **Genwave** function (see Appendix D). (b) Reconstructed signal after one application of the TP algorithm. Reduction ratio is 512:256, PRD = 7.78%.

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

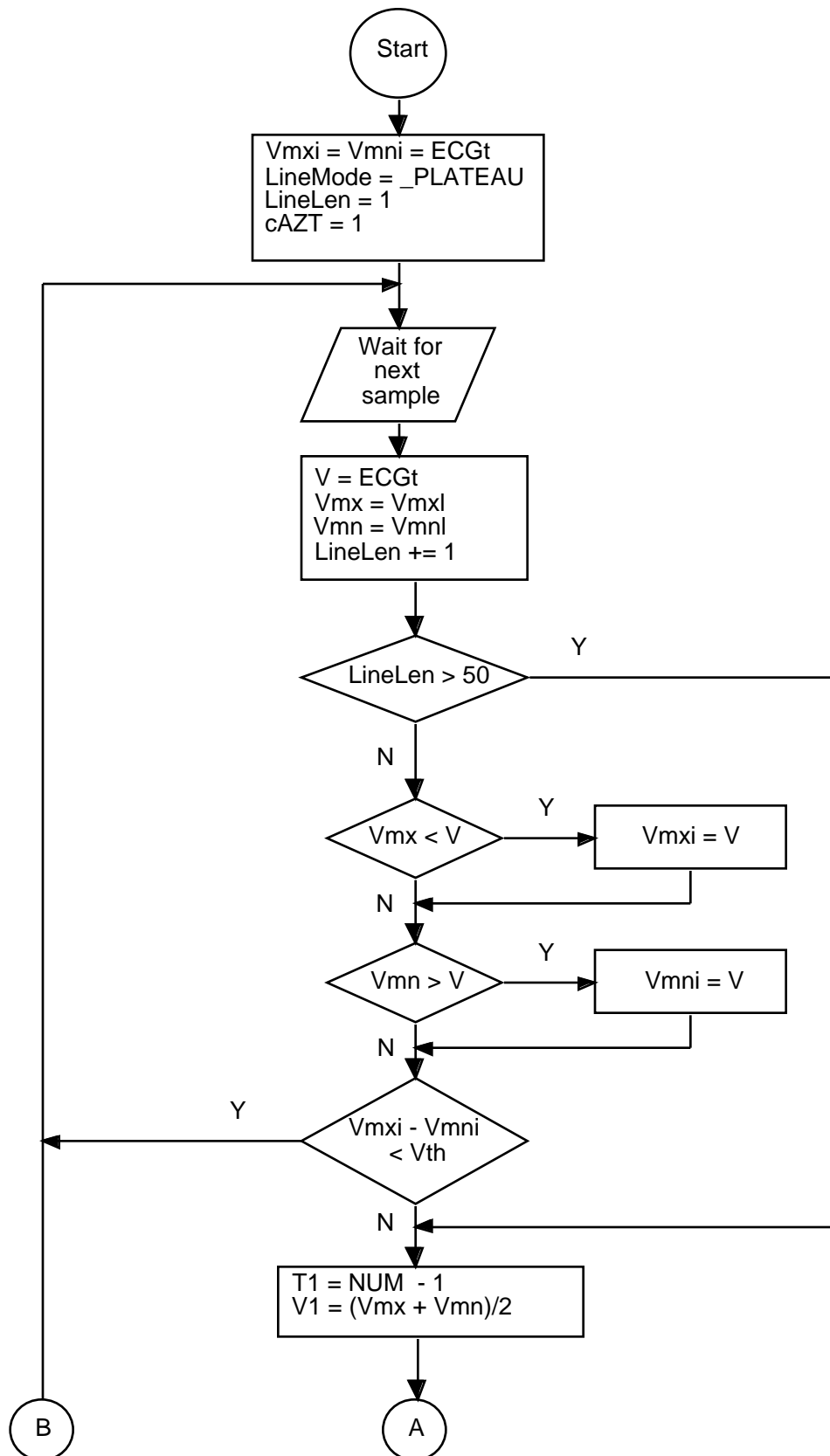


Figure 10.4(a) Flowchart for the line detection operation of the AZTEC algorithm.

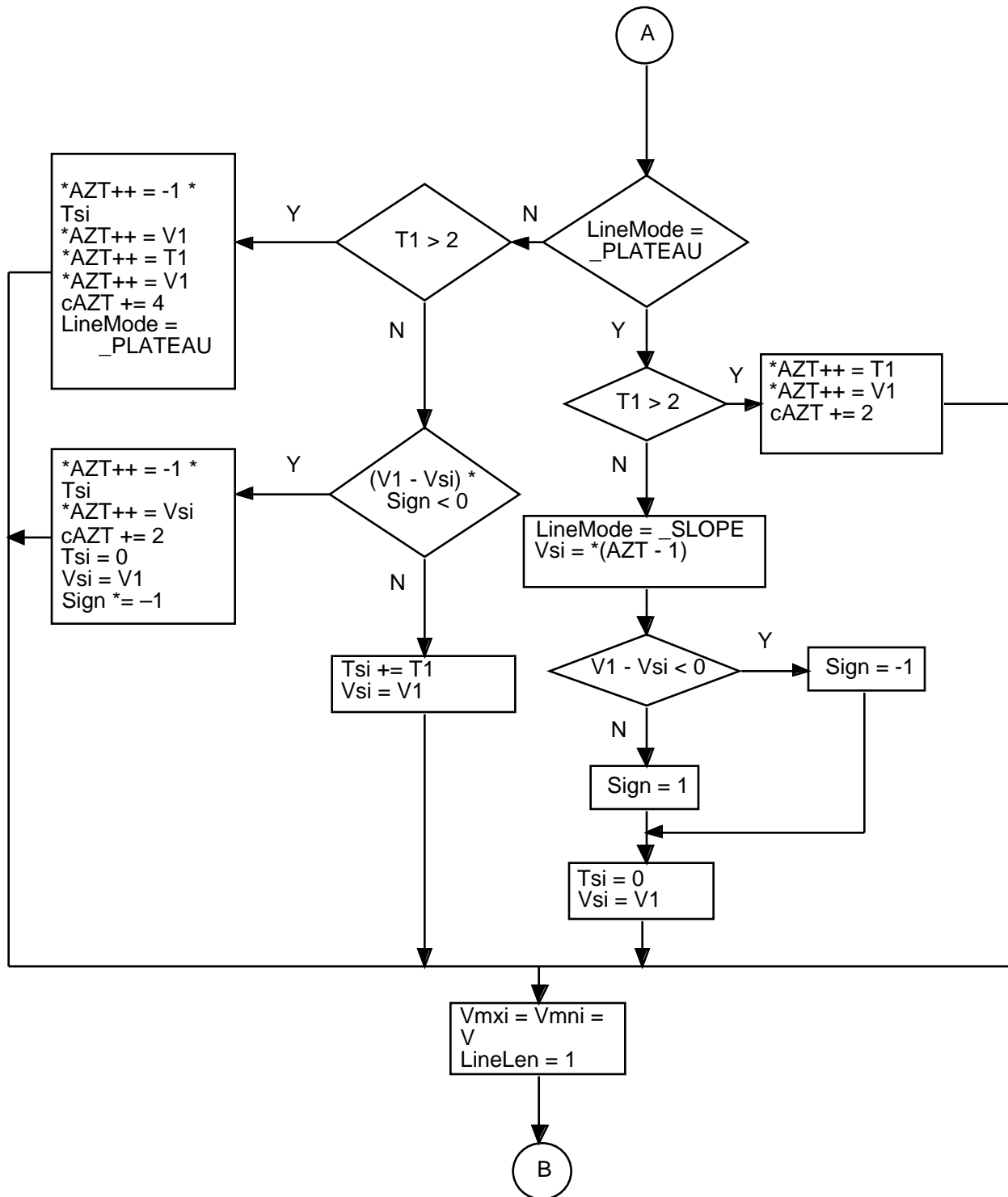


Figure 10.4(b) Flowchart of the line processing operation of the AZTEC algorithm.

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.

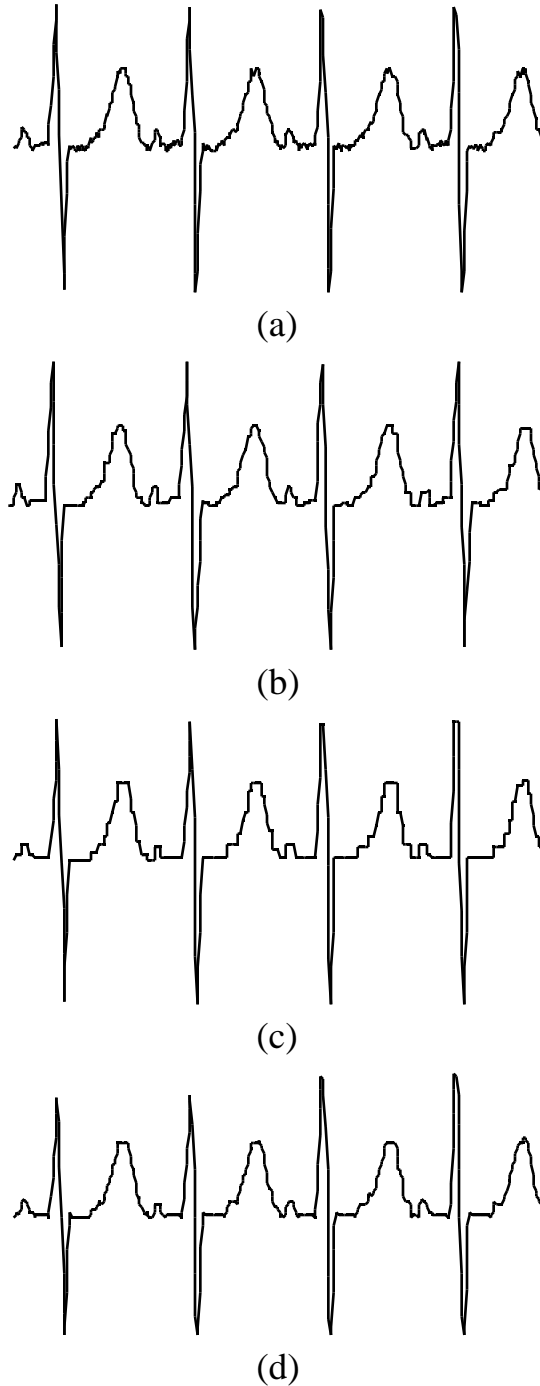


Figure 10.5 Examples of AZTEC applications. (a) Original waveform generated by the UW DigiScope **Genwave** function (see Appendix D). (b) Small threshold, reduction ratio = 512:233, PRD = 24.4%. (c) Large threshold, reduction ratio = 512:153, PRD = 28.1%. (d) Smoothed signal from (c), $L = 3$, PRD = 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, (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 $(X_0, X_1 + \epsilon)$, and L_1 passes through the point $(X_0, X_1 - \epsilon)$. 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 $(X_0, X_2 + \epsilon)$ and $(X_0, X_2 - \epsilon)$. 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

$$X_{U2} = \frac{X_{U1} - X_0}{T} + X_{U1} \quad (10.4a)$$

and

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

where $T = t_T - t_0$.

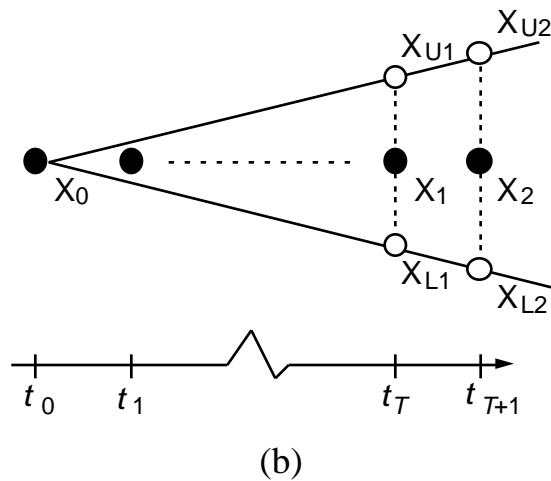
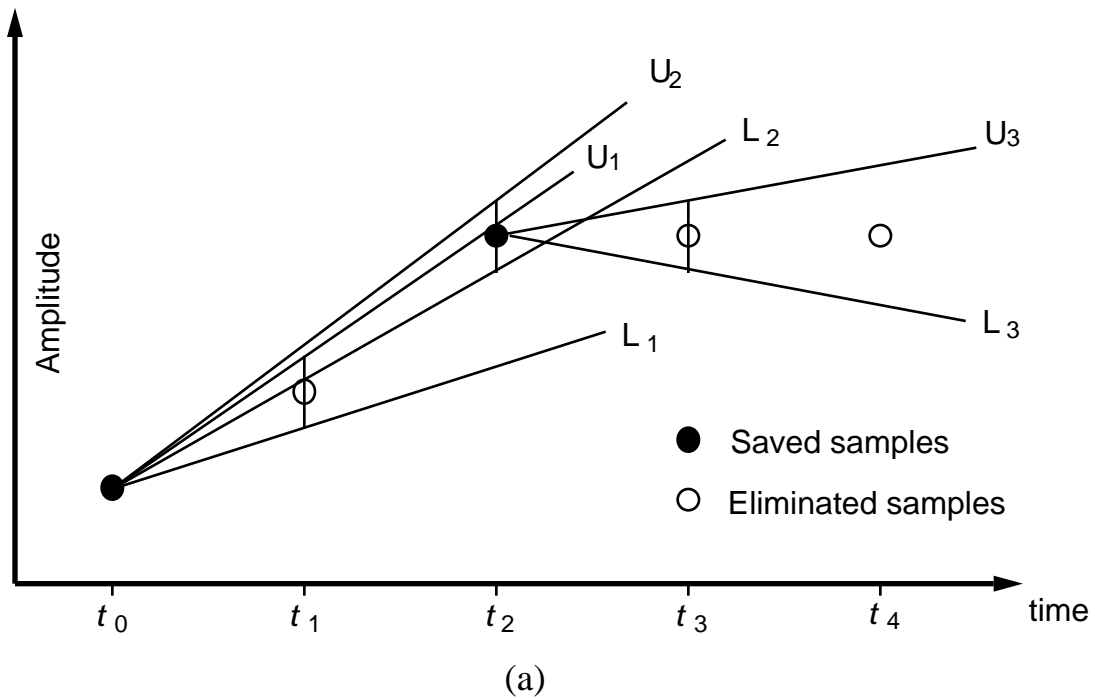


Figure 10.6 Illustration of the Fan algorithm. (a) Upper and lower slopes (U and L) are drawn within error threshold around sample points taken at t_1, t_2, \dots (b) Extrapolation of X_{U2} and X_{L2} from X_{U1}, X_{L1} , and X_0 .

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 = (XL1 - X0)/T + XL1; /* 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 */
    }

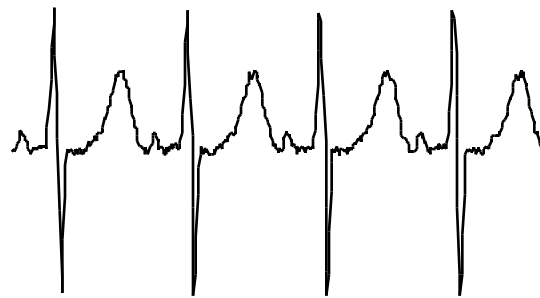
    else    { /* X2 out of bound, save line */

        *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.



(a)



(b)



(c)



(d)

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

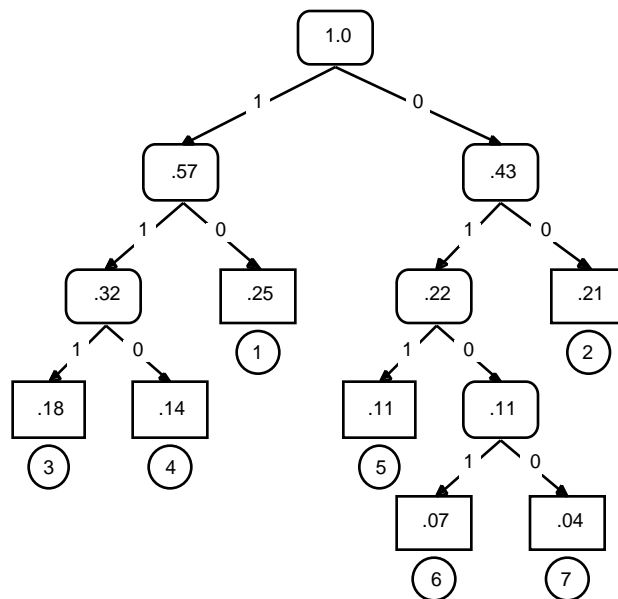
Figure 10.9 illustrates the principles of Huffman coding. As an example, assume that we wish to transmit the set of 28 data points

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

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

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

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

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

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

$$x'(n) = ax(nT - T) + bx(nT + T) \quad (10.7)$$

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.

10.4.5 Run-length encoding

Used extensively in the facsimile technology, run-length encoding exploits the high degree of correlation that occurs in successive bits in the facsimile bit stream. A bit in the facsimile output may either be 1 or 0, depending on whether it is a black or white pixel. On a typical document, there are clusters of black and white pixels that give rise to this high correlation. Run-length encoding simply transforms the original bit stream into the string $\{v_1, l_1, v_2, l_2, \dots\}$ where v_i are the values and l_i are the lengths. The observant reader will quickly recognize that both AZTEC and the 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 elements. The output of run-length encoding $\{1, 5, 3, 4, 0, 3\}$ contains only six elements. Further data reduction is possible by applying Huffman coding to the output 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

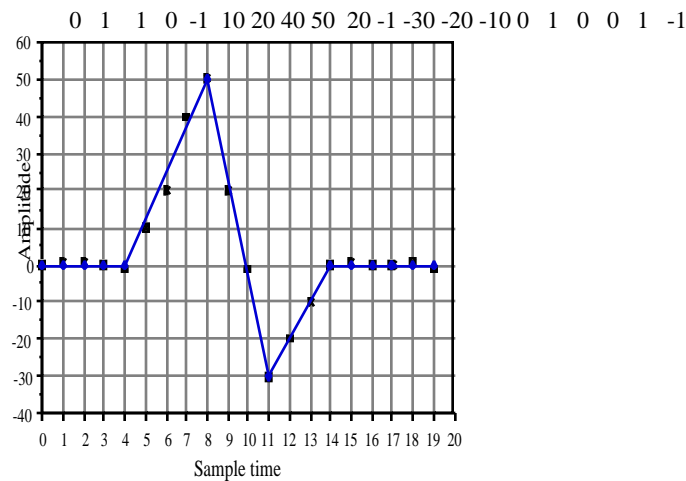
- Abenstein, J. P. and Tompkins, W. J. 1982. New data-reduction algorithm for real-time ECG analysis, *IEEE Trans. Biomed. Eng.*, **BME-29**: 43–48.
- Bohs, L. N. and Barr, R. C. 1988. Prototype for real-time adaptive sampling using the Fan algorithm, *Med. & Biol. Eng. & Comput.*, **26**: 574–83.
- 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?

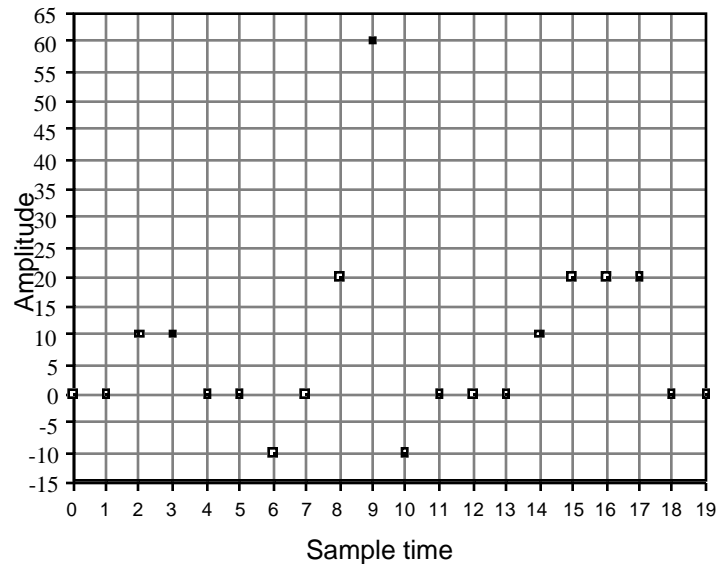
0 1 1 0 -1 10 20 40 50 20 -1 -30 -20 -10 0 1 0 0 1 -1

- (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:

01010110001