CHAPTER

WAVEFORM QUANTIZATION AND COMPRESSION

10

CHAPTER OUTLINE

10.1 LINEAR MIDTREAD QUANTIZATION

As we discussed in Chapter 2, in a digital signal processing (DSP) system, the first step is to sample and quantize the continuous signal. Quantization is the process of rounding off the sampled signal voltage to the predetermined levels that will be encoded by analog-to-digital conversion (ADC). We have described the quantization process in Chapter 2, in which we studied unipolar and bipolar linear quantizers in detail. In this section, we focus on a linear midtread quantizer, which is used in the digital communications (Roddy and Coolen, 1997; Tomasi, 2004), and its use to quantize the speech waveform. The linear midtread quantizer is similar to the bipolar linear quantizer discussed in Chapter 2 except that the midtread quantizer offers the same decoded magnitude range for both positive and negative voltages.

Let us look at a midtread quantizer. The characteristics and binary codes for a 3-bit midtread quantizer are depicted in Fig. 10.1, where the code is in a sign magnitude format. Positive voltage is coded

Digital Signal Processing. https://doi.org/10.1016/B978-0-12-815071-9.00010-5

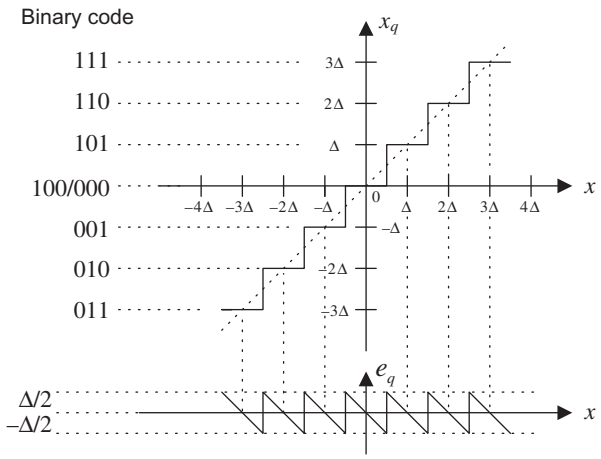

FIG. 10.1

Characteristics of a 3-bit midtread quantizer.

using a sign bit of logic 1, while negative voltage is coded by a sign bit of logic 0; the next two bits are the magnitude bits. The key feature of the linear midtread quantizer is noted as follows: when $0 \leq x < \Delta/2$, the binary code of 100 is produced; when $-\Delta/2 \leq x < 0$, the binary code of 000 is generated, where Δ is the quantization step size. However, the quantized values for both codes of 100 and 000 are the same and equal to $x_q = 0$. We can also see details in Table 10.1. For the 3-bit midtread quantizer, we expect seven quantized values instead of 8; that is; there are $2^n - 1$ quantization levels for the n-bit midtread quantizer. Note that the quantization signal range is $(2^n - 1)\Delta$ and the magnitudes of the quantized values are symmetric, as shown in Table 10.1. We apply the midtread quantizer particularly for speech waveform coding.

The following example serves to illustrate coding principles of the 3-bit midtread quantizer.

Table 10.1 Quantization Table for the 3-Bit Midtread Quantizer

Binary Code	Quantization Level x_q (V)	Input Signal Subrange (V)
0 1 1	-3Δ	$-3.5\Delta \leq x < -2.5\Delta$
0 1 0	-2Δ	$-2.5\Delta \leq x < -1.5\Delta$
0 0 1	$-\Delta$	$-1.5\Delta \leq x < -0.5\Delta$
0 0 0	0	$-0.5\Delta \leq x < 0$
1 0 0	0	$0 \leq x < 0.5\Delta$
1 0 1	Δ	$0.5\Delta \leq x < 1.5\Delta$
1 1 0	2Δ	$1.5\Delta \leq x < 2.5\Delta$
1 1 1	3Δ	$2.5\Delta \leq x < 3.5\Delta$

Note that: step size $= \Delta = (x_{max} - x_{min})/(2^3 - 1)$; x_{max} *= maximum voltage; and* $x_{min} = -x_{max}$; *and coding format: (a) sign bit: 1 = plus; 0 = minus; (b) 2 magnitude bits.*

EXAMPLE 10.1

For the 3-bit midtread quantizer described in Fig. 10.1 and the analog signal with a range from −5 to 5 V,

(a) Determine the quantization step size, and

(b) Determine the binary codes, recovered voltages, and quantization errors when the input is −3.6 and 0.5 V, respectively.

Solution:

(a) The quantization step size is calculated as

$$\Delta = \frac{5-(-5)}{2^3-1} = 1.43\,\text{V}.$$

(b) For $x = -3.6$ V, we have $x = \frac{-3.6}{1.43} = -2.52\Delta$. From quantization characteristics, it follows that the binary code = 011 and the recovered voltage is $x_q = -3\Delta = -4.29$ V. Thus the quantization error is computed as

$$e_q = x_q - x = -4.28 - (-3.6) = -0.69\,\text{V}.$$

For $x = 0.5 = \frac{0.5}{1.43}\Delta = 0.35\Delta$, we get binary code = 100. Based on Fig. 10.1, the recovered voltage and quantization error are found to be $x_q = 0$ and

$$e_q = 0 - 0.5 = -0.5\,\text{V}.$$

As discussed in Chapter 2, the linear midtread quantizer introduces quantization noise, as shown in Fig. 10.1; and the signal-to-noise power ratio (SNR) is given by

$$\text{SNR dB} = 10.79 + 20 \times \log_{10}\left(\frac{x_{rms}}{\Delta}\right), \tag{10.1}$$

where x_{rms} designates the root-mean-squared value of the speech data to be quantized. The practical equation for estimating the SNR for the speech data sequence $x(n)$ of N data points is written as

$$\text{SNR} = \left(\frac{\sum_{n=0}^{N-1} x^2(n)}{\sum_{n=0}^{N-1} \left(x_q(n) - x(n)\right)^2}\right) \tag{10.2}$$

$$\text{SNR dB} = 10 \cdot \log_{10}(\text{SNR}). \tag{10.3}$$

Note that $x(n)$ and $x_q(n)$ are the speech data to be quantized and the quantized speech data, respectively. Eq. (10.2) gives the absolute SNR, and Eq. (10.3) produces the SNR in terms of decibel (dB). Quantization error is the difference between the quantized speech data (or quantized voltage level) and speech data (or analog voltage), that is, $(x_q(n) - x(n))$. Also note that from Eq. (10.1), increasing 1 bit to the linear quantizer would improve SNR by approximately 6 dB. Let us examine performance of the 5-bit linear midtread quantizer.

In the following simulation, we use a 5-bit midtread quantizer to quantize the speech data. After quantization, the original speech, quantized speech, and quantized error are plotted in Fig. 10.2. Since the program calculates $x_{rms}/x_{\max} = 0.203$, we yield x_{rms} as $x_{rms} = 0.203 \times x_{\max} = 0.0203 \times 5 = 1.015$ and

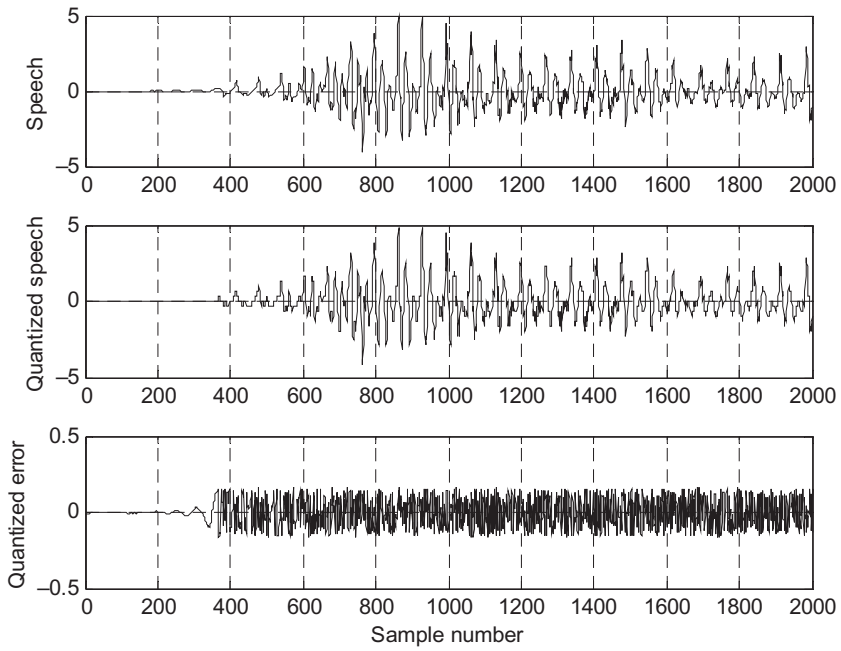

FIG. 10.2

Plots of original speech, quantized speech, and quantization error.

$\Delta = 10/(2^5 - 1) = 0.3226$. Applying Eq. (10.1) gives SNR = 21.02 dB. The SNR using Eqs. (10.2) and (10.3) is approximately 21.6 dB.

The first plot in Fig. 10.2 is the original speech, and the second plot shows the quantized speech. Quantization error is displayed in the third plot, where the error amplitude interval is uniformly distributed between -0.1613 and 0.1613, indicating the bounds of the quantized error ($\Delta/2$). The details of the MATLAB implementation is given in Programs 10.1–10.3, 10.7 in Section 10.6.

To improve the SNR, the number of bits must be increased. However, increasing the number of encoding bits will cost an expansive ADC device, larger storage media for storing the speech data, and a larger bandwidth for transmitting the digital data. To gain a more efficient quantization approach, we study the μ-law companding in the next section.

10.2 μ-LAW COMPANDING

In this section, we study the analog μ-law companding, which takes an analog input signal, and digital μ-law companding, which deals with linear pulse-code modulation (PCM) codes.

10.2.1 ANALOG μ-LAW COMPANDING

To reduce the number of bits to encode each speech datum, μ-law companding, called log-PCM coding, is applied. μ-Law companding (Roddy and Coolen, 1997; Tomasi, 2004) was first used in the United States and Japan in the telephone industry (G.711 standard). μ-Law companding is a compression process. It explores the principle that the higher amplitudes of analog signals are compressed before ADC while expanded after digital-to-analog conversion (DAC). As studied in the linear quantizer, the quantization error is uniformly distributed. This means that the maximum quantization error stays the same no matter how big or small the speech samples are. μ-Law companding can be employed to make the quantization error smaller when the sample amplitude is smaller and to make the quantization error bigger when the sample amplitude is bigger, using the same number of bits per sample. It is described in Fig. 10.3.

As shown in Fig. 10.3, x is the original speech sample, which is the input to the compressor, while y is the output from the μ-law compressor; then the output y is uniformly quantized. Assuming that the quantized sample y_q is encoded and sent to the μ-law expander, the expander will perform the reverse process to obtain the quantized speech sample x_q. The compression and decompression processes cause the maximum quantization error $|x_q - x|_{\max}$ to be small for the smaller sample amplitudes and large for the larger sample amplitudes.

The equation for the μ-law compressor is given by

$$y = \text{sign}(x)\frac{\ln\left(1+\mu\dfrac{|x|}{|x|_{\max}}\right)}{\ln(1+\mu)}, \tag{10.4}$$

where $|x|_{\max}$ is the maximum amplitude of the inputs, while μ is a positive parameter to control the degree of the compression. $\mu = 0$ corresponds to no compression, while $\mu = 255$ is adopted in the industry. The compression curve with $\mu = 255$ is plotted in Fig. 10.4. Note that the sign function $sign(x)$ shown Eq. (10.4) is defined as

$$\text{sign}(x) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases}. \tag{10.5}$$

Solving Eq. (10.4) by substituting the quantized value, that is, $y = y_q$ we achieve the expander equation as

$$x_q = |x|_{\max}\,\text{sign}(y_q)\frac{(1+\mu)^{|y_q|}-1}{\mu}. \tag{10.6}$$

For the case $\mu = 255$, the expander curve is plotted in Fig. 10.5.

Let us look at Example 10.2 for the μ-law compression.

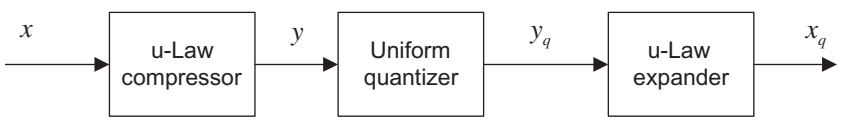

FIG. 10.3

Block diagram for μ-law compressor and μ-law expander.

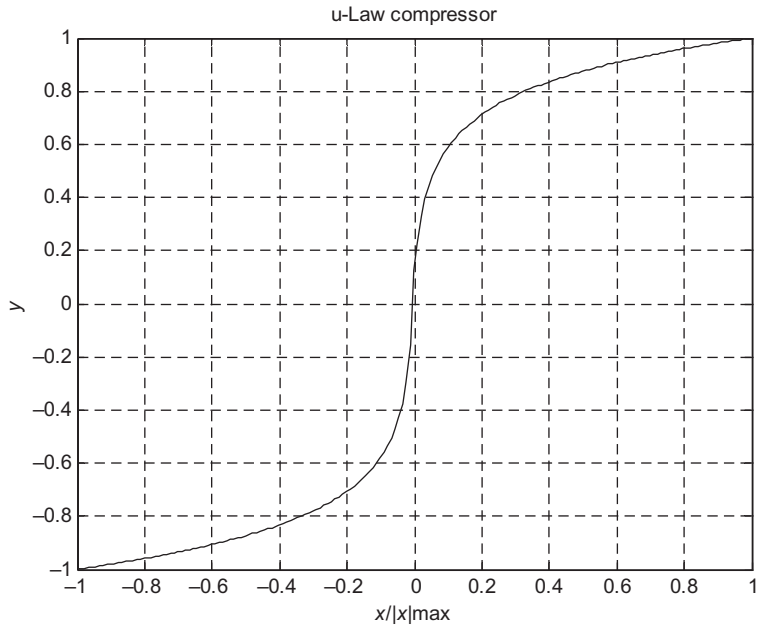

FIG. 10.4

Characteristics for the μ-law compander.

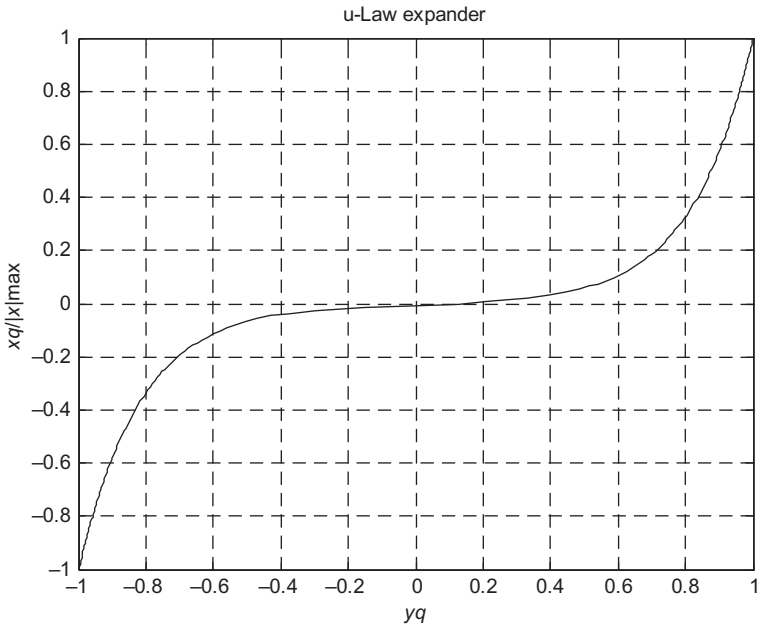

FIG. 10.5

Characteristics for the μ-law expander.

EXAMPLE 10.2

For the μ-law compression and expansion process shown in Fig. 10.3, with $\mu = 255$, the 3-bit midtread quantizer described in Fig. 10.1, and an analog signal ranging from -5 to 5 V, determine the binary codes, recovered voltages, and quantization errors when the input is

(a) -3.6 V

(b) 0.5 V.

Solution:

(a) For the μ-law compression and $x = -3.6$ V, we can determine the quantization input as

$$y = \text{sign}(-3.6)\frac{\ln\left(1+255\frac{|-3.6|}{|5|_{\max}}\right)}{\ln(1+255)} = -0.94.$$

As shown in Fig. 10.4, the range of y is 2, thus the quantization step size is calculated as

$$\Delta = \frac{2}{2^3-1} = 0.286 \text{ and } y = \frac{-0.94}{0.286} = -3.28\Delta.$$

From quantization characteristics, it follows that the binary code = 011 and the recovered signal is $y_q = -3\Delta = -0.858$.

Applying the μ-law expander leads to

$$x_q = |5|_{\max}\,\text{sign}(-0.858)\frac{(1+255)^{|-0.858|}-1}{255} = -2.264.$$

Thus the quantization error is computed as

$$e_q = x_q - x = -2.264 - (-3.6) = 1.336 \text{ volts.}$$

(b) Similarly, for $x = 0.5$, we get

$$y = \text{sign}(0.5)\frac{\ln\left(1+255\frac{|0.5|}{|5|_{\max}}\right)}{\ln(1+255)} = 0.591$$

In terms of the quantization step, we get

$$y = \frac{0.519}{0.286}\Delta = 2.1\Delta \text{ and binary code} = 110.$$

Based on Fig. 10.1, the recovered signal is.

$y_q = 2\Delta = 0.572$

and the expander gives

$$x_q = |5|_{\max}\,\text{sign}(0.572)\frac{(1+255)^{|0.572|}-1}{255} = 0.448 \text{ volts.}$$

Finally, the quantization error is given by

$$e_q = 0.448 - 0.5 = -0.052 \text{ volts.}$$

As we can see, with 3 bits per sample, the strong signal is encoded with bigger quantization error, while the weak signal is quantized with less quantization error.

In the following simulation, we apply a 5-bit μ-law compander with $\mu = 255$ in order to quantize and encode the speech data used in the last section. Fig. 10.6 is a block diagram of compression and decompression.

Fig. 10.7 shows the original speech data, the quantized speech data using μ-law compression, and the quantization error for comparisons. The quantized speech wave is very close to the original speech wave. From the plots in Fig. 10.7, we can observe that the amplitude of the quantization error changes according to the amplitude of the speech being quantized. The bigger quantization error is introduced when the amplitude of speech data is larger; on the other hand, the smaller quantization error is produced when the amplitude of speech data is smaller.

Compared with the quantized speech using the linear quantizer shown in Fig. 10.2, the decompressed signal using the μ-law compander looks and sounds much better, since the quantized signal

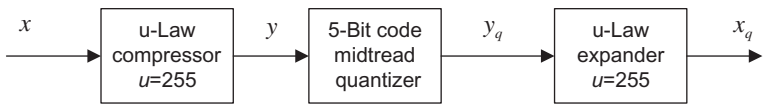

FIG. 10.6

The 5-bit midtread uniform quantizer with $\mu = 255$ used for simulation.

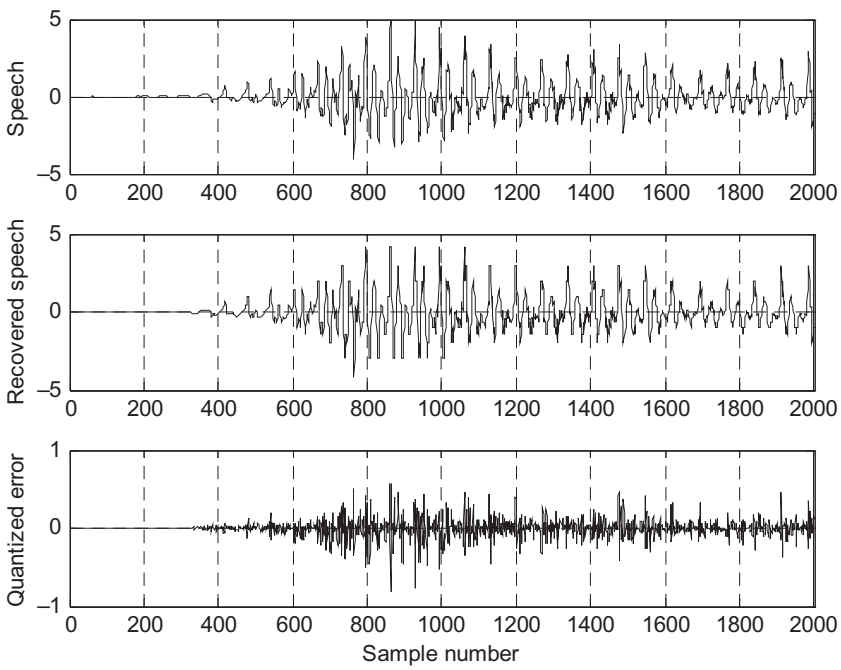

FIG. 10.7

Plots of the original speech, quantized speech, and quantization error with the μ-law compressor and expander.

can better keep tracking the original large amplitude signal and original small amplitude signal as well. MATLAB implementation is shown in Programs 10.4-7 in Section 10.6.

10.2.2 DIGITAL μ-LAW COMPANDING

In many multimedia applications, the analog signal is first sampled and then it is digitized into a linear PCM code with a larger number of bits per sample. The digital μ-law companding further compresses the linear PCM code using the compressed PCM code with a smaller number of bits per sample without losing sound quality. The block diagram of a digital μ-law compressor and expander is shown in Fig. 10.8.

The typical digital μ-law companding system compresses a 12-bit linear PCM code to an 8-bit compressed code. This companding characteristic is depicted in Fig. 10.9, where it closely resembles an analog compression curve with $\mu = 255$ by approximating the curve using a set of eight straight-line segments. The slope of each successive segment is exactly one-half of the previous segment. Fig. 10.9 shows the 12-bit to 8-bit digital companding curve for the positive portion only. There are 16 segments, accounting for both positive and negative portions.

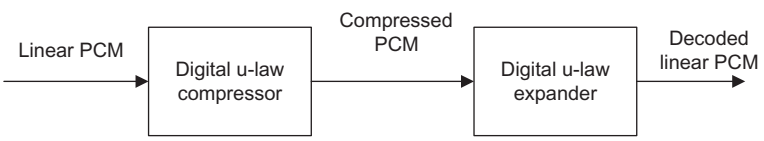

FIG. 10.8

The block diagram for the μ-law compressor and expander.

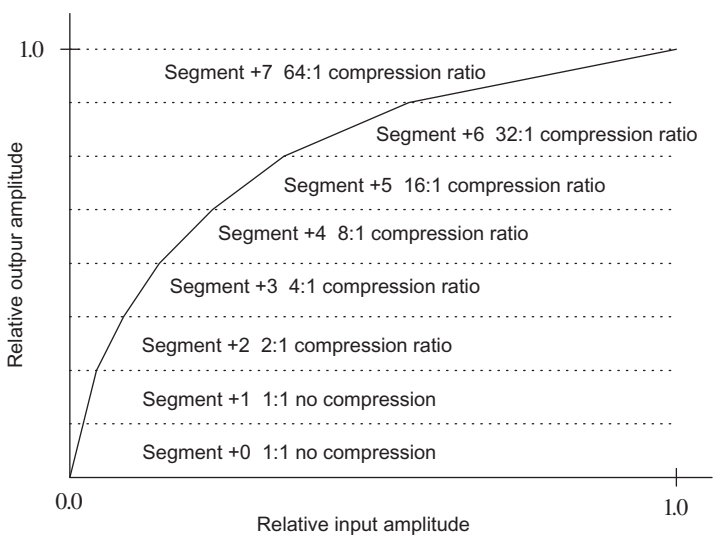

FIG. 10.9

μ-255 compression characteristics (for positive portion only).

Table 10.2 The Format of 8-Bit Compressed PCM Code

Sign Bit	3-Bit	4-Bit
1 = + 0 = −	Segment identifier 000 to 111	Quantization interval A B C D 0000 to 1111

However, like the midtread quantizer discussed in the first section, only 13-segments are used, since segments +0, −0, +1, and −1 form a straight line with a constant slope and are considered as one segment. As shown in Fig. 10.9, when the relative input is very small, such as in segment 0 or segment 1, there is no compression, while when the relative input is larger such that it is in segment 3 or segment 4, the compression occurs with the compression ratios (CRs) of 2:1 and 4:1, respectively. The format of the 12-bit linear PCM code is in the sign-magnitude form with the most significant bit (MSB) as the sign bit (1 = positive value and 0 = negative value) plus 11 magnitude bits. The compressed 8-bit code has a format shown in Table 10.2, where it consists of a sign bit, a 3-bit segment identifier, and a 4-bit quantization interval within the specified segment. Encoding and decoding procedures are very simple, as illustrated in Tables 10.3 and 10.4, respectively.

As shown in those two tables, the prefix "S" is used to indicate the sign bit, which could be either 1 or 0; A, B, C, and D, are transmitted bits; and the bit position with an "X" is the truncated bit during the compression and hence would be lost during decompression. For the 8-bit compressed PCM code in Table 10.3, the 3 bits between "S" and "ABCD" indicate the segment number that is obtained by subtracting the number of consecutive zeros (less than or equal to 7) after the "S" bit in the original 12-bit PCM code from 7. Similarly, to recover the 12-bit linear code in Table 10.4, the number of consecutive zeros after the "S" bit can be determined by subtracting the segment number in the 8-bit compressed code from 7. We will illustrate the encoding and decoding processes in Examples 10.3 and 10.4.

EXAMPLE 10.3

In a digital companding system, encode each of the following 12-bit linear PCM codes into to an 8-bit compressed PCM code.

1. 1 0 0 0 0 0 0 0 0 1 0 1
2. 0 0 0 0 1 1 1 0 1 0 1 0

Solution:

1. Based on Table 10.3, we identify the 12-Bit PCM Code as S = 1, A = 0, B = 1, C = 0, and D = 1, which is in segment 0. From the fourth column in Table 10.3, we get the 8-bit compressed code as 1 0 0 0 0 1 0 1.
2. For the second 12-bit PCM code, we note that S = 0, A = 1, B = 1, C = 0, D = 1, and XXX = 010, and the code belongs to segment 4. Thus, from the fourth column in Table 10.3, we have 0 1 0 0 1 1 0 1.

Table 10.3 $\mu - 255$ Encoding Table

Segment	12-Bit Linear Code	12-Bit Amplitude Range in Decimal	8-bit Compressed Code
0	S0000000ABCD	0 to 15	S000ABCD
1	S0000001ABCD	16 to 31	S001ABCD
2	S000001ABCDX	32 to 63	S010ABCD
3	S00001ABCDXX	64 to 127	S011ABCD
4	S0001ABCDXXX	128 to 255	S100ABCD
5	S001ABCDXXXX	256 to 511	S101ABCD
6	S01ABCDXXXXX	512 to 1023	S110ABCD
7	S1ABCDXXXXXX	1023 to 2047	S111ABCD

Table 10.4 $\mu - 255$ Decoding Table

8-Bit Compressed Code	8-Bit Amplitude Range in Decimal	Segment	12-Bit Linear Code
S000ABCD	0–15	0	S0000000ABCD
S001ABCD	16–31	1	S0000001ABCD
S010ABCD	32–47	2	S000001ABCD1
S011ABCD	48–63	3	S00001ABCD10
S100ABCD	64–79	4	S0001ABCD100
S101ABCD	80–95	5	S001ABCD1000
S110ABCD	96–111	6	S01ABCD10000
S111ABCD	112–127	7	S1ABCD100000

EXAMPLE 10.4

In a digital companding system, decode each of the following 8-bit compressed PCM codes into a 12-bit linear PCM code.

1. 1 0 0 0 0 1 0 1
2. 0 1 0 0 1 1 0 1

Solution:

1. Using the decoding Table 10.4, we note that S = 1, A = 0, B = 1, C = 0, and D = 1, and the code is in segment 0. Decoding leads to: 1 0 0 0 0 0 0 0 0 1 0 1, which is identical to the 12-bit PCM code in (1) in Example 10.3. We expect this result, since there is no compression for segment 0 and segment 1.
2. Applying Table 10.4, it follows that S = 0, A = 1, B = 1, C = 0, and D = 1, and the code resides in segment 4. Decoding achieves: 0 0 0 0 1 1 1 0 1 1 0 0.
   As expected, this code is the approximation of the code in (2) in Example 10.3. Since segment 4 has compression, the last 3 bits in the original 12-bit linear code, that is, XXX = 010 = 2 in decimal, are discarded during transmission or storage. When we recover these 3 bits, the best guess should be the middle value: XXX = 100 = 4 in decimal for the 3-bit coding range from 0 to 7.

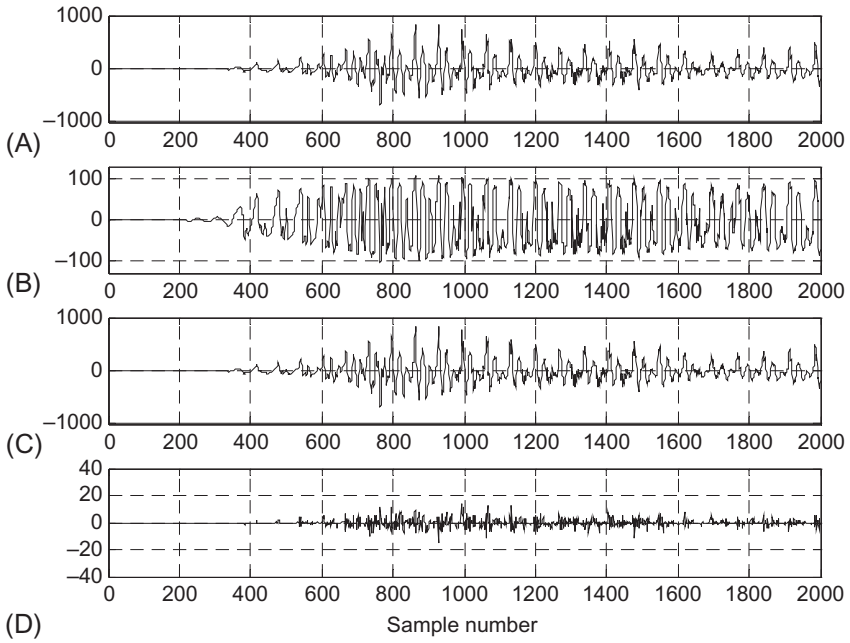

FIG. 10.10

The μ-255 compressor and expander: (A) 12-bit speech data; (B) 8-bit compressed data; (C) 12-bit decoded speech; (D) quantization error.

Now we apply the μ-255 compander to compress the 12-bit speech data as shown in Fig. 10.10A. The 8-bit compressed code is plotted in Fig. 10.10B. Plots C and D in the figure show the 12-bit speech after decoding and quantization error, respectively. We can see that the quantization error follows the amplitude of speech data relatively. The decoded speech sounds no difference when compared with the original speech. Programs 10.8–10.10 in Section 10.6 show the detail of the MATLAB implementation.

10.3 EXAMPLES OF DIFFERENTIAL PULSE CODE MODULATION (DPCM), DELTA MODULATION, AND ADAPTIVE DPCM G.721

Data compression can be further achieved using *differential pulse code modulation* (DPCM). The general idea is to use past recovered values as the basis to predict the current input data and then encode the difference between the current input and the predicted input. Since the difference has a significantly reduced signal dynamic range, it can be encoded with fewer bits per sample. Therefore, we obtain data compression. First, we study the principles of the DPCM concept that will help us understand adaptive DPCM (ADPCM) in the next subsection.

10.3.1 EXAMPLES OF DPCM AND DELTA MODULATION

Fig. 10.11 shows a schematic diagram for the DPCM encoder and decoder. We denote the original signal $x(n)$; the predicted signal $\tilde{x}(n)$; the quantized, or recovered signal $\hat{x}(n)$; the difference signal to be quantized $d(n)$; and the quantized difference signal $d_q(n)$. The quantizer can be chosen as a uniform quantizer, a midtread quantizer (e.g., see Table 10.5), or others available. The encoding block produces binary bit stream in the DPCM encoding. The predictor uses the past predicted signal and quantized difference signal to predict the current input value $x(n)$ as close as possible. The digital filter or adaptive filter can serve as the predictor. On the other hand, the decoder recovers the quantized difference signal, which can be added to the predictor output signal to produce the quantized and recovered signal, as shown in Fig. 10.11B.

In Example 10.5, we examine a simple DPCM coding system via the process of encoding and decoding numerical actual data.

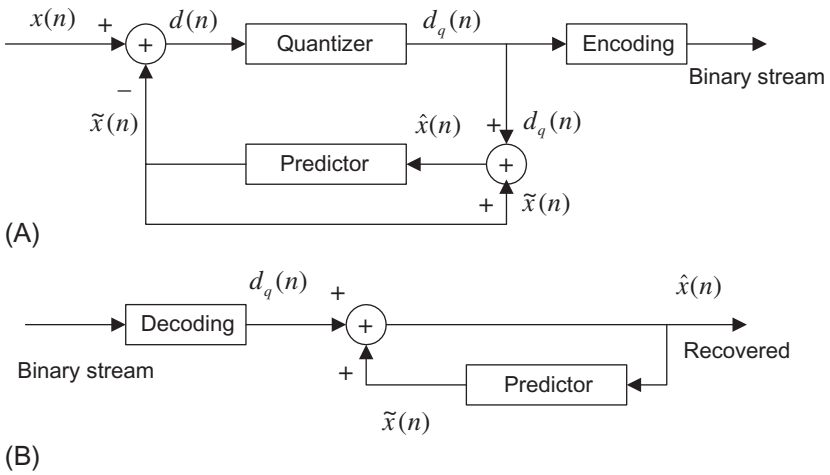

FIG. 10.11

DPCM block diagram: (A) encoder; and (B) decoder.

Table 10.5 Quantization Table for the 3-Bit Quantizer in Example 10.5

Binary Code	Quantization Value $d_q(n)$	Subrange in $d(n)$
0 1 1	−11	$-15 \le d(n) < -7$
0 1 0	−5	$-7 \le d(n) < -3$
0 0 1	−2	$-3 \le d(n) < -1$
0 0 0	0	$-1 \le d(n) < 0$
1 0 0	0	$0 \le d(n) \le 1$
1 0 1	2	$1 < d(n) \le 3$
1 1 0	5	$3 < d(n) \le 7$
1 1 1	11	$7 < d(n) \le 15$

EXAMPLE 10.5

A DPCM system has the following specifications:

Encoder scheme: $\tilde{x}(n) = \hat{x}(n-1)$; predictor
$d(n) = x(n) - \tilde{x}(n)$
$d_q(n) = Q[d(n)]$= quantizer in Table 10.5
$\hat{x}(n) = \tilde{x}(n) + d_q(n)$

Decoding scheme: $\tilde{x}(n) = \hat{x}(n-1)$; predictor
$d_q(n)$= quantizer in Table 10.5.
$\hat{x}(n) = \tilde{x}(n) + d_q(n)$

The 5-bit input data: $x(0) = 6$, $x(1) = 8$, $x(2) = 13$.

(a) Perform DPCM encoding to produce the binary code for each input datum

(b) Perform DCPM decoding to recover the data using the binary code in (1).

Solution:

(a) Let us perform encoding according to the encoding scheme.

For $n = 0$, we have.

$\tilde{x}(0) = \hat{x}(-1) = 0$
$d(0) = x(0) - \tilde{x}(0) = 6 - 0 = 6$
$d_q(0) = Q[d(0)] = 5$
$\hat{x}(0) = \tilde{x}(0) + d_q(0) = 0 + 5 = 5$
Binary code =110.

For $n = 1$, it follows that.

$\tilde{x}(1) = \hat{x}(0) = 5$
$d(1) = x(1) - \tilde{x}(1) = 8 - 5 = 3$
$d_q(1) = Q[d(1)] = 2$
$\hat{x}(1) = \tilde{x}(1) + d_q(1) = 5 + 2 = 7$
Binary code = 101.

For $n = 2$, results are.

$\tilde{x}(2) = \hat{x}(1) = 7$
$d(2) = x(2) - \tilde{x}(2) = 13 - 7 = 6$
$d_q(2) = Q[d(2)] = 5$
$\hat{x}(2) = \tilde{x}(2) + d_q(2) = 7 + 5 = 12$
Binary code = 110.

(b) We conduct the decoding scheme as follows.

For $n = 0$, we get:

Binary code = 110
$d_q(0) = 5$; from Table 10.5
$\tilde{x}(0) = \hat{x}(-1) = 0$
$\hat{x}(0) = \tilde{x}(0) + d_q(0) = 0 + 5 = 5$ (recovered).

For $n = 1$, decoding shows:

Binary code = 101
$d_q(1) = 2$; from Table 10.5
$\tilde{x}(1) = \hat{x}(0) = 5$
$\hat{x}(1) = \tilde{x}(1) + d_q(1) = 5 + 2 = 7$ (recovered).

For $n=2$, we have:

Binary code = 110

$d_q(2)=5$; from Table 10.5

$\tilde{x}(2)=\hat{x}(1)=7$

$\hat{x}(2)=\tilde{x}(2)+d_q(2)=7+5=12$ (recovered).

From this example, we could verify that the 5-bit code is compressed to the 3-bit code. However, we can see that each recovered data has a quantization error. Hence, the DPCM is a lossy data compression scheme.

DPCM for which a single bit is used in the quantization table becomes *delta modulation* (DM). The quantization table contains two quantized values, A and $-A$, where A is the quantization step size. DM quantizes the difference of the current input sample and the previous input sample using a 1-bit code word. To conclude the idea, we list the equations for encoding and decoding as follows:

Encoder scheme: $\tilde{x}(n)=\hat{x}(n-1)$; predictor

$$d(n)=x(n)-\tilde{x}(n)$$

$$d_q(n)=\begin{cases}+A, \text{ for } & d_q(n)\geq 0, \text{ output bit : } 1\\ -A, \text{ for } & d_q(n)<0, \text{ output bit : } 0\end{cases}$$

$$\hat{x}(n)=\tilde{x}(n)+d_q(n)$$

Decoding scheme: $\tilde{x}(n)=\hat{x}(n-1)$; predictor

$$d_q(n)=\begin{cases}+A, \text{ input bit : } 1\\ -A, \text{ input bit : } 0\end{cases}$$

$$\hat{x}(n)=\tilde{x}(n)+d_q(n)$$

Note that the predictor has one sample delay.

EXAMPLE 10.6

For a DM system with 5-bit input data

$$x(0)=6,\ x(1)=8,\ x(2)=13$$

and the quantized constant as $A=7$,

(a) Perform the DM encoding to produce the binary code for each input datum

(b) Perform the DM decoding to recover the data using the binary code in (a).

Solution:

(a) Applying encoding according, we have

For $n=0$,

$\tilde{x}(0)=\hat{x}(-1)=0$, $d(0)=x(0)-\tilde{x}(0)=6-0=6$.

$d_q(0)=7$, $\hat{x}(0)=\tilde{x}(0)+d_q(0)=0+7=7$.

Binary code = 1.

For $n=1$,

$\tilde{x}(1)=\hat{x}(0)=7$, $d(1)=x(1)-\tilde{x}(1)=8-7=1$.

$d_q(1)=7$, $\hat{x}(1)=\tilde{x}(1)+d_q(1)=7+7=14$.

Binary code = 1.

Continued

EXAMPLE 10.6—CONT'D

For $n=2$,

$$\tilde{x}(2)=\hat{x}(1)=14,\ d(2)=x(2)-\tilde{x}(2)=13-14=-1.$$
$$d_q(2)=-7,\ \hat{x}(2)=\tilde{x}(2)+d_q(2)=14-7=7.$$
Binary code = 0.

(b) Applying the decoding scheme leads to

For $n=0$,

Binary code = 1.
$d_q(0)=7,\ \tilde{x}(0)=\hat{x}(-1)=0.$
$\hat{x}(0)=\tilde{x}(0)+d_q(0)=0+7=7$ (recovered).

For $n=1$,

Binary code 1.
$d_q(1)=7,\ \tilde{x}(1)=\hat{x}(0)=7.$
$\hat{x}(1)=\tilde{x}(1)+d_q(1)=7+7=14$ (recovered).

For $n=2$,

Binary code 0.
$d_q(2)=-7,\ \tilde{x}(2)=\hat{x}(1)=14.$
$\hat{x}(2)=\tilde{x}(2)+d_q(2)=14-7=7$ (recovered).

We can see that the DM coding causes a larger quantization error for each recovered sample. In practice, this can be solved using a very high sampling rate (much larger than the Nyquist rate), and making the quantization step size A adaptive. The quantization step size increases by a factor when the slope magnitude of the input sample curve becomes bigger, that is, the condition in which the encoder produces continuous logic 1's or generates continuous logic 0's in the coded bit stream. Similarly, the quantization step decreases by a factor when the encoder generates logic 1 and logic 0 alternatively. Hence, the resultant DM is called *adaptive* DM. In practice, the DM chip replaces the predictor, feedback path, and adder (see Fig. 10.11) with an integrator for both the encoder and the decoder. Detailed information can be found in Li et al. (2014), Roddy and Coolen (1997), and Tomasi (2004).

10.3.2 ADAPTIVE DIFFERENTIAL PULSE CODE MODULATION G.721

In this subsection, an efficient compression technique for speech waveform is described, that is, adaptive DPCM (ADPCM), per recommendation G.721 of the CCITT (the Comité Consultatif International Téléphonique et Télégraphique). General discussion can be found in Li et al. (2014), Roddy and Coolen (1997), and Tomasi (2004). The simplified block diagrams of the ADPCM encoder and decoder are shown in Fig. 10.12A and B.

As shown in Fig. 10.12A for the ADPCM encoder, first a difference signal $d(n)$ is obtained, by subtracting an estimate of the input signal $\tilde{x}(n)$ from the input signal $x(n)$. An adaptive 16-level quantizer is used to assign four binary digits $I(n)$ to the value of the difference signal for transmission to the decoder. At the same time, an inverse quantizer produces a quantized difference signal $d_q(n)$ from the same four binary digits $I(n)$. The adaptive quantizer and inverse quantizer operate based on the quantization table and the scale factor obtained from the quantizer adaptation to keep tracking the energy change of the difference signal to be quantized. The input signal estimate from the adaptive predictor is then added to

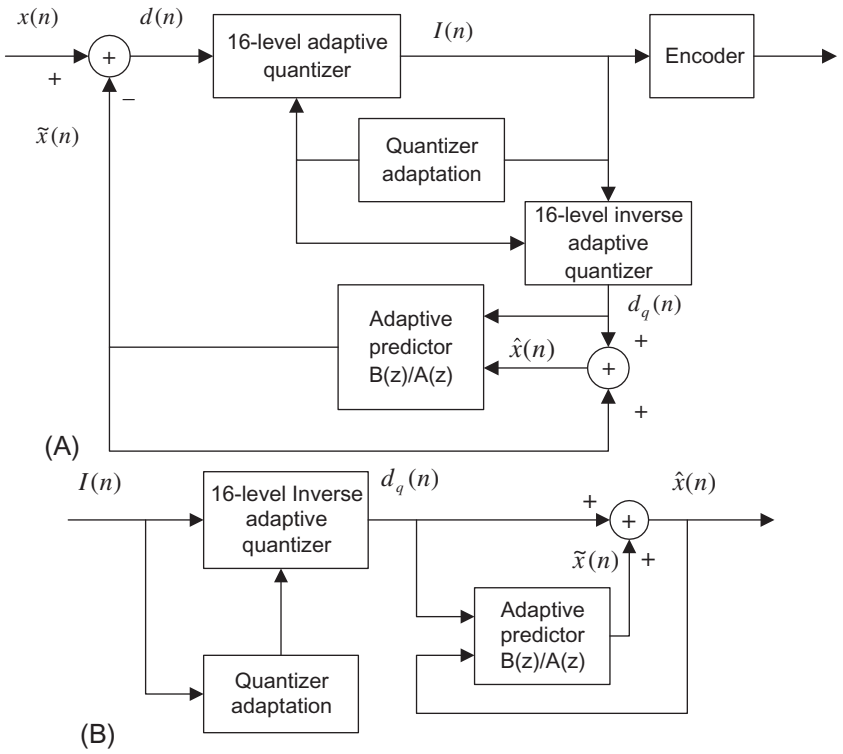

FIG. 10.12

(A) The ADPCM encoder, (B) ADPCM decoder.

this quantized difference signal to produce the reconstructed version of the input $\hat{x}(n)$. Both the reconstructed signal and the quantized difference signal are operated on by an adaptive predictor, which generates the estimate of the input signal, thereby completing the feedback loop.

The decoder shown in Fig. 10.12B includes a structure identical to the feedback part of the encoder as depicted in Fig. 10.12A. It first converts the received 4-bit data $I(n)$ to the quantized difference signal $d_q(n)$ using the adaptive quantizer. Then, at the second stage, the adaptive predictor uses the recovered quantized difference signal $d_q(n)$ and recovered current output $\tilde{x}(n)$ to generate the next output. Note that the adaptive predictors of both the encoder and the decoder change correspondingly based on the signal to be quantized. The details of the adaptive predictor will be discussed.

Now, let us examine the ADPCM encoder principles. As shown in Fig. 10.12A, the difference signal is computed as

$$d(n) = x(n) - \tilde{x}(n). \tag{10.7}$$

A 16-level nonuniform adaptive quantizer is used to quantize the difference signal $d(n)$. Before quantization, $d(n)$ is converted to a base-2 logarithmic representation and scaled by $y(n)$, which is computed by the scale-factor algorithm. Four binary codes $I(n)$ are used to specify the quantized signal level representing $d_q(n)$, and the quantized difference $d_q(n)$ is also fed to the inverse adaptive quantizer. Table 10.6 shows the quantizer normalized input and output characteristics.

Table 10.6 Quantizer Normalized Input and Output Characteristics

Normalized Quantizer Input Range: $\log_2\|d(n)\| - y(n)$	Magnitude $\|I(n)\|$	Normalized Quantizer Output: $\log_2\|d_q(n)\| - y(n)$
[3.12, +∞)	7	3.32
[2.72, 3.12)	6	2.91
[2.34, 2.72)	5	2.52
[1.91, 2.34)	4	2.13
[1.38, 1.91)	3	1.66
[0.62, 1.38)	2	1.05
[−0.98, 0.62)	1	0.031
(−∞, −0.98)	0	−∞

The scaling factor for the quantizer and the inverse quantizer $y(n)$ are computed according to the 4-bit quantizer output $I(n)$ and the adaptation speed control parameter $a_l(n)$, the fast (unlocked) scale factor $y_u(n)$, the slow (locked)-scale factor $y_l(n)$, and the discrete function $W(I)$, defined in Table 10.7:

$$y_u(n) = \left(1 - 2^{-5}\right)y(n) + 2^{-5}W(I(n)), \tag{10.8}$$

where $1.06 \le y_u(n) \le 10.00$.

The slow scale factor $y_l(n)$ is derived from the fast scale factor $y_u(n)$ using a lowpass filter as following:

$$y_l(n) = \left(1 - 2^{-6}\right)y_l(n-1) + 2^{-6}y_u(n). \tag{10.9}$$

The fast and slow scale factors are then combined to compute the scale factor

$$y(n) = a_l(n)y_u(n-1) + (1 - a_l(n))y_l(n-1). \tag{10.10}$$

Next the controlling parameter $0 \le a_l(n) \le 1$ tends toward unity for speech signals and toward zero for voice band data signals and tones. It is updated based on the following parameters: $d_{ms}(n)$, which is the relatively short term average of $F(I(n))$; $d_{ml}(n)$, which is the relatively long-term average of $F(I(n))$; and the variable $a_p(n)$, where $F(I(n))$ is defined as in Table 10.8.

Hence, we have

$$d_{ms}(n) = \left(1 - 2^{-5}\right)d_{ms}(n-1) + 2^{-5}F(I(n)) \tag{10.11}$$

and

$$d_{ml}(n) = \left(1 - 2^{-7}\right)d_{ml}(n-1) + 2^{-7}F(I(n)), \tag{10.12}$$

Table 10.7 Discrete Function $W(I)$

$\|I(n)\|$	7	6	5	4	3	2	1	0
$W(I)$	70.13	22.19	12.38	7.00	4.0	2.56	1.13	−0.75

Table 10.8 Discrete Function $F(I(n))$

$\lvert I(n)\rvert$	7	6	5	4	3	2	1	0
$F(I(n))$	7	3	1	1	1	0	0	0

while the variable $a_p(n)$ is given by

$$a_p(n)=\begin{cases}(1-2^{-4})a_p(n-1)+2^{-3} & \text{if } |d_{ms}(n)-d_{ml}(n)|\geq 2^{-3}d_{ml}(n)\\ (1-2^{-4})a_p(n-1)+2^{-3} & \text{if } y(n)<3\\ (1-2^{-4})a_p(n)+2^{-3} & \text{if } t_d(n)=1\\ 1 & \text{if } t_r(n)=1\\ (1-2^{-4})a_p(n) & \text{otherwise}\end{cases}. \tag{10.13}$$

$a_p(n)$ approaches 2 when the difference between $d_{ms}(n)$ and $d_{ml}(n)$ is large and approaches 0 when the difference is small. Also $a_p(n)$ approaches 2 for an idle channel (indicated by $y(n)<3$) or partial band signals (indicated by $t_d(n)=1$). Finally, $a_p(n)$ is set to 1 when the partial band signal transition is detected ($t_r(n)=1$).

$a_l(n)$ used in Eq. (10.10) is defined as

$$a_l(n)=\begin{cases}1 & a_p(n-1)>1\\ a_p(n-1) & a_p(n-1)\leq 1\end{cases}. \tag{10.14}$$

The partial band signal $t_d(n)$ and the partial band signal transition $t_r(n)$ that appear in Eq. (10.13) will be discussed later.

The predictor is to compute the signal estimate $\tilde{x}(n)$ from the quantized difference signal $d_q(n)$. The predictor z-transfer function, which is effectively suitable for a variety of input signals, is given by

$$\frac{B(z)}{A(z)}=\frac{b_0+b_1z^{-1}+b_2z^{-2}+b_3z^{-3}+b_4z^{-4}+b_5z^{-5}}{1-a_1z^{-1}-a_2z^{-2}}. \tag{10.15}$$

It consists of a fifth-order portion that models the zeros and a second-order portion that models poles of the input signals. The input signal estimate is expressed in terms of the processed signal $\hat{x}(n)$ and the signal $x_z(n)$ processed by the finite impulse response (FIR) filter as follows:

$$\tilde{x}(n)=a_1(n)\hat{x}(n-1)+a_2(n)\hat{x}(n-2)+x_z(n), \tag{10.16}$$

where

$$\hat{x}(n-i)=\tilde{x}(n-i)+d_q(n-i) \tag{10.17}$$

$$x_z(n)=\sum_{i=0}^{5}b_i(n)d_q(n-i). \tag{10.18}$$

Both sets of predictor coefficients are updated using a simplified gradient algorithm:

$$a_1(n)=(1-2^{-8})a_1(n-1)+3\cdot 2^{-8}\,\text{sign}(p(n))\,\text{sign}(p(n-1)) \tag{10.19}$$

$$a_2(n)=(1-2^{-7})a_2(n-1)+2^{-7}\{\text{sign}(p(n)\,\text{sign}(p(n-2)-f(a_1(n-1))\,\text{sign}(p(n)\,\text{sign}(p(n-1))\}, \tag{10.20}$$

where $p(n)=d_q(n)+x_z(n)$ and

$$f(a_1(n))=\begin{cases}4a_1(n) & |a_1(n)|\le 2^{-1}\\ 2\,\text{sign}(a_1(n)) & |a_1(n)|>2^{-1}\end{cases}. \tag{10.21}$$

Note that the function $sign(x)$ is defined in Eq. (10.5), while the function signn$(x)=1$ when $x>0$; signn$(x)=0$ when $x=0$; and signn$(x)=-1$ when $x<0$ with stability constrains as

$$|a_2(n)|\le 0.75 \text{ and } |a_1(n)|\le 1-2^{-4}-a_2(n) \tag{10.22}$$

$$a_1(n)=a_2(n)=0 \text{ if } t_r(n)=1. \tag{10.23}$$

Also, the equations for updating the coefficients for the zero-order portion are given by

$$b_i(n)=\left(1-2^{-8}\right)b_i(n-1)+2^{-7}\,\text{sign}\left(d_q(n)\right)\text{sign}\left(d_q(n-i)\right) \tag{10.24}$$

for $i=0, 1, 2, \ldots, 5$ with the following constrains:

$$b_0(n)=b_1(n)=b_2(n)=b_3(n)=b_4(n)=b_5(n)=0 \text{ if } t_r(n)=1. \tag{10.25}$$

$$t_d(n)=\begin{cases}1 & a_2(n)<-0.71875\\ 0 & \text{otherwise}\end{cases} \tag{10.26}$$

$$t_r(n)=\begin{cases}1 & a_2(n)<-0.71875 \text{ and } \left|d_q(n)\right|>24\cdot 2^{y_l}\\ 0 & \text{otherwise}\end{cases}. \tag{10.27}$$

$t_d(n)$ is the indicator of detecting a partial band signal (tone). If a tone is detected ($t_d(n)=1$); Eq. (10.13) is invoked to drive the quantizer into the fast mode of adaptation. $t_r(n)$ is the indicator for a transition from a partial band signal. If it is detected ($t_r(n)=1$), setting the predictor coefficients to be zero as shown in Eqs. (10.23) and (10.25) will force the quantizer into the fast mode of adaptation.

Simulation Example

To illustrate the performance, we apply the ADPCM encoder to the speech data used in Section 10.1 and then operate the ADPCM decoder to recover the speech signal. As described, the ADPCM uses 4 bits to encode each speech sample. The MATLAB implementations for the encoder and decoder are listed in Programs 10.11–10.13 in Section 10.6. Fig. 10.13 plots the original speech samples, decoded speech samples, and the quantization errors. From the figure, we see that the decoded speech data are very close to the original speech data; the quantization error is very small as compared with the speech sample, and its amplitude follows the change in amplitude of the speech data. In practice, we cannot tell the difference between the original speech and the decoded speech by listening to them. However, the ADPCM encodes each speech sample using 4 bit per sample, while the original data are presented using 16 bits, thus the CR of 4:1 is achieved.

In practical applications, data compression can reduce the storage media and bit rate for the efficient digital transmission. To measure the performance of data compression, we use

- the data CR, which is the ratio of original data file size to the compressed data file size, or ratio of the original code size in bits to the compressed code size in bits for the fixed length coding, and
- the bit rate, which is in terms of bits per second (bps) and can be calculated by:

$$\text{bit rate}=m\times f_s(\text{bps}), \tag{10.28}$$

where $m=$number of bits per sample (bits) and $f_s=$ sampling rate (samples per second).

Now, let us look at an application example.

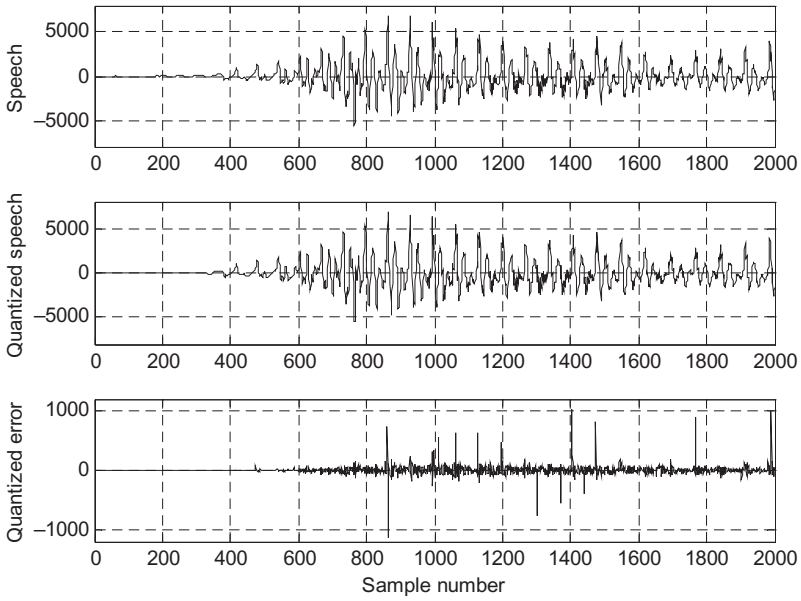

FIG. 10.13

Original speech, quantized speech, and quantization error using the ADPCM.

EXAMPLE 10.7

Speech is sampled at 8 kHz and each sample is encoded by 12 bits per sample. Using the following encoding methods:

1. Non compression
2. Standard μ-law compression
3. Standard ADPCM encoding (4 bits per sample),
   (a) Determine the CR and the bit rate for each of the encoders and decoders.
   (b) Determine the number of channels that the phone company can carry if a telephone system can transport the digital voice channel over the digital link having a capacity of 1.536 Mbps.

Solution:

a.1 For noncompression:
CR = 1:1

$$\text{Bit rate} = 12\frac{\text{bits}}{\text{sample}} \times 8000\frac{\text{sample}}{\text{second}} = 96\,(\text{kbps}).$$

b.1 $$\text{Number of channels} = \frac{1.536}{96}\frac{\text{MBPS}}{\text{KBPS}} = 16.$$

a.2 For the standard μ-law compression, each sample is encoded using 8 bits per sample. Hence, we have

Continued

EXAMPLE 10.7—CONT'D

$$\text{CR} = \frac{12\,\text{bits/sample}}{8\,\text{bits/sample}} = 1.5:1.$$

$$\text{Bit rate} = 8\frac{\text{bits}}{\text{sample}} \times 8000\frac{\text{sample}}{\text{second}} = 64\ \text{(kbps)}.$$

b.2 $$\text{Number of channels} = \frac{1.536\,\text{MBPS}}{64\,\text{KBPS}} = 24.$$

a.3 For the standard ADPCM with 4 bits per sample, it follows that

$$\text{CR} = \frac{12\,\text{bits/sample}}{4\,\text{bits/sample}} = 3:1$$

$$\text{Bit rate} = 4\frac{\text{bits}}{\text{sample}} \times 8000\frac{\text{sample}}{\text{second}} = 32\ \text{(kbps)}.$$

b.3 $$\text{Number of channels} = \frac{1.536\,\text{MBPS}}{32\,\text{KBPS}} = 48.$$

10.4 DISCRETE COSINE TRANSFORM, MODIFIED DISCRETE COSINE TRANSFORM, AND TRANSFORM CODING IN MPEG AUDIO

This section introduces *discrete cosine transform* (DCT) and explains how to apply it in transform coding. This section also shows how to remove the block effects in transform coding using a modified DCT (MDCT). Finally, we examine how MDCT coding is used in the MPEG (Motion Picture Experts Group) audio format, which is used as a part of MPEG audio, such as MP3 (MPEG-1 layer 3).

10.4.1 DISCRETE COSINE TRANSFORM

Given N data samples, we define the one-dimensional (1D) DCT pair given:

Forward transform:

$$X_{DCT}(k) = \sqrt{\frac{2}{N}}C(k)\sum_{n=0}^{N-1}x(n)\cos\left[\frac{(2n+1)k\pi}{2N}\right],\quad k=0,1,\ldots,N-1. \tag{10.29}$$

Inverse transform:

$$x(n) = \sqrt{\frac{2}{N}}\sum_{k=0}^{N-1}C(k)X_{DCT}(k)\cos\left[\frac{(2n+1)k\pi}{2N}\right],\quad n=0,1,\ldots,N-1 \tag{10.30}$$

$$C(k) = \begin{cases} \dfrac{\sqrt{2}}{2} & k=0 \\ 1 & \text{otherwise,} \end{cases} \tag{10.31}$$

where $x(n)$ is the input data sample and $X_{DCT}(k)$ is the DCT coefficient. The DCT transforms the time domain signal to frequency domain coefficients. However, unlike the discrete Fourier transform (DFT), there are no complex number operations for both forward and inverse transforms. Both forward and inverse transforms use the same scale factor:

$$\sqrt{\frac{2}{N}}C(k).$$

In terms of transform coding, the DCT decomposes a block of data into the direct-current (DC) coefficient corresponding to the average of the data samples and the alternating-current (AC) coefficients, each corresponding to the frequency component (fluctuation). The terms "DC" and "AC" come from basic electrical engineering. In transform coding, we can quantize the DCT coefficients and encode them into binary information. The inverse DCT can transform the DCT coefficients back to the input data. Let us proceed to Examples 10.8 and 10.9.

EXAMPLE 10.8

Assuming that the following input data each can be encoded by 5 bits, including a sign bit:

$$x(0)=10, x(1)=8, x(2)=10, \text{ and } x(3)=12,$$

(a) Determine the DCT coefficients.
(b) Use the MATLAB function **dct()** to verify all the DCT coefficients.

Solution:

(a) Using Eq. (10.29) leads to

$$X_{DCT}(k)=\sqrt{\frac{1}{2}}C(k)\left[x(0)\cos\left(\frac{\pi k}{8}\right)+x(1)\cos\left(\frac{3\pi k}{8}\right)+x(2)\cos\left(\frac{5\pi k}{8}\right)+x(3)\cos\left(\frac{7\pi k}{8}\right)\right].$$

When $k=0$, we see that the DC component is calculated as

$$\begin{aligned}X_{DCT}(0)&=\sqrt{\frac{1}{2}}C(0)\left[x(0)\cos\left(\frac{\pi\times 0}{8}\right)+x(1)\cos\left(\frac{3\pi\times 0}{8}\right)+x(2)\cos\left(\frac{5\pi\times 0}{8}\right)+x(3)\cos\left(\frac{7\pi\times 0}{8}\right)\right]\\&=\sqrt{\frac{1}{2}}\times\frac{\sqrt{2}}{2}[x(0)+x(1)+x(2)+x(3)]=\frac{1}{2}(10+8+10+12)=20.\end{aligned}$$

We clearly see that the first DCT coefficient is a scaled average value.
For $k=1$,

$$\begin{aligned}X_{DCT}(1)&=\sqrt{\frac{1}{2}}C(1)\left[x(0)\cos\left(\frac{\pi\times 1}{8}\right)+x(1)\cos\left(\frac{3\pi\times 1}{8}\right)+x(2)\cos\left(\frac{5\pi\times 1}{8}\right)+x(3)\cos\left(\frac{7\pi\times 1}{8}\right)\right]\\&=\sqrt{\frac{1}{2}}\times 1\times\left[10\times\cos\left(\frac{\pi\times 1}{8}\right)+8\times\cos\left(\frac{3\pi\times 1}{8}\right)+10\times\cos\left(\frac{5\pi\times 1}{8}\right)+12\times\cos\left(\frac{7\pi\times 1}{8}\right)\right]=-1.8478.\end{aligned}$$

Similarly, we have the rest as

$$X_{DCT}(2)=2 \text{ and } X_{DCT}(3)=0.7654.$$

(b) Using the MATLAB 1D-DCT function **dct(),** we can verify that

```
>>dct([10 8 10 12])
ans=20.0000 -1.8478 2.0000 0.7654.
```

EXAMPLE 10.9

Assuming the following DCT coefficients:

$$X_{DCT}(0) = 20, X_{DCT}(1) = -1.8478, X_{DCT}(2) = 2, \text{ and } X_{DCT}(3) = 0.7654,$$

(a) Determine $x(0)$.
(b) Use the MATLAB function **idct()** to verify all the recovered data samples.

Solution:

(a) Applying Eqs. (10.30) and (10.31), we have

$$x(0) = \sqrt{\frac{1}{2}} \times \left\{ C(0)X_{DCT}(0)\cos\left(\frac{\pi}{8}\right) + C(1)X_{DCT}(1)\cos\left(\frac{3\pi}{8}\right) + C(2)X_{DCT}(2)\cos\left(\frac{5\pi}{8}\right) + C(3)X_{DCT}(3)\cos\left(\frac{7\pi}{8}\right) \right\}$$

$$= \sqrt{\frac{1}{2}} \times \left\{ \frac{\sqrt{2}}{2} \times 20 \times \cos\left(\frac{\pi}{8}\right) + 1 \times (-1.8474) \times \cos\left(\frac{3\pi}{8}\right) + 1 \times 2 \times \cos\left(\frac{5\pi}{8}\right) + 1 \times 0.7654 \times \cos\left(\frac{7\pi}{8}\right) \right\} = 10.$$

(b) With the MATLAB 1D inverse DCT function **idct(),** we obtain

```
>>idct([20 -1.8478 2 0.7654])
ans = 10.0000 8.0000 10.0000 12.0000.
```

We verify that the input data samples are as the ones in Example 10.8.

In Example 10.9, we obtained an exact recovery of the input data from the DCT coefficients, since infinite precision of each DCT coefficient is preserved. However, in transform coding, each DCT coefficient is quantized using the number of bits per sample assigned by a bit allocation scheme. Usually the DC coefficient requires a larger number of bits to encode, since it carries more energy of the signal, while each AC coefficient requires a smaller number of bits to encode. Hence, the quantized DCT coefficients approximate the DCT coefficients in infinite precision, and the recovered input data with the quantized DCT coefficients will certainly have quantization errors.

EXAMPLE 10.10

Assuming the following DCT coefficients in infinite precision:

$$X_{DCT}(0) = 20, X_{DCT}(1) = -1.8478, X_{DCT}(2) = 2, \text{ and } X_{DCT}(3) = 0.7654,$$

we had exact recovered data as: 10, 8, 10, 12; this was verified in Example 10.9. If a bit allocation scheme quantizes the DCT coefficients using a scale factor of 4 in the following form:

$$X_{DCT}(0) = 4 \times 5 = 20, X_{DCT}(1) = 4 \times (-0) = 0, X_{DCT}(2) = 4 \times 1 = 4, \text{ and } X_{DCT}(3) = 4 \times 0 = 0.$$

We can code the scale factor of 4 by 3 bits (magnitude bits only), the scaled DC coefficient of 5 with 4 bits (including a sign bit), and the scaled AC coefficients of 0, 1, 0 using 2 bits each. 13 bits in total are required.
Use the MATLAB **idct()** to recover the input data samples.

Solution:
Using the MATLAB function **idct()** and the quantized DCT coefficients, it follows that.
>> idct([20 0 4 0])
ans = 12 8 8 12.
As we see, the original sample requires 5 bits (4 magnitude bits and 1 sign bit) to encode each of 10, 8, 10, and 12 for a total of 20 bits. Hence, 7 bit are saved for coding this data block using the DCT. We expect many more bits to be saved in practice, in which a longer frame of the correlated data samples is used. However, quantization errors are introduced.

For comprehensive coverage of the topics on DCT, see Li et al. (2014), Nelson (1992), Sayood (2012), and Stearns and Hush (2011).

10.4.2 MODIFIED DISCRETE COSINE TRANSFORM

In the previous section, we have seen how a 1D-DCT is adopted for coding a block of data. When we apply the 1D-DCT to audio coding, we first divide the audio samples into blocks and then transform each block of data with DCT. The DCT coefficients for each block are quantized according to the bit allocation scheme. However, when we decode DCT blocks back, we encounter edge artifacts at boundaries of the recovered DCT blocks, since the DCT coding is block based. This effect of edge artifacts produces periodic noise and is annoying in the decoded audio. To solve for such a problem, the windowed MDCT has been developed (described in Pan, 1995; Princen and Bradley, 1986). The principles are illustrated in Fig. 10.14. As we shall see, the windowed MDCT is used in MPEG-1 MP3 audio coding.

We describe and discuss only main steps for coding data blocks using the windowed MDCT (W-MDCT) based on Fig. 10.14.

Encoding stage:

1. Divide data samples into blocks that each have N (must be an even number) samples, and further divide each block into two subblocks, each with $N/2$ samples for the data overlap purpose.
2. Apply the window function for the overlapped blocks. As shown in Fig. 10.14, if one block contains the subblocks A and B, the next one would consist of subblocks B and C. The subblock B is the overlapped block. This procedure continues. A window function $h(n)$ is applied to each N sample block to reduce the possible edge effects. Next, the W-MDCT is applied. The W-MDCT is given by

$$X_{MDCT}(k) = 2\sum_{n=0}^{N-1} x(n)h(n)\cos\left[\frac{2\pi}{N}(n+0.5+N/4)(k+0.5)\right] \text{ for } k = 0,1,\cdots,N/2-1. \tag{10.32}$$

Note that we need to compute and encode only half of the MDCT coefficients (since the other half can be reconstructed based on the first half of the MDCT coefficients).

3. Quantize and encode the MDCT coefficients.

Decoding stage:

1. Receive the $N/2$ MDCT coefficients, and use Eq. (10.33) to recover the second half of the coefficients:

$$X_{MDCT}(k) = (-1)^{\frac{N}{2}+1} X_{MDCT}(N-1-k), \text{ for } k = N/2, N/2+1,\cdots,N-1. \tag{10.33}$$

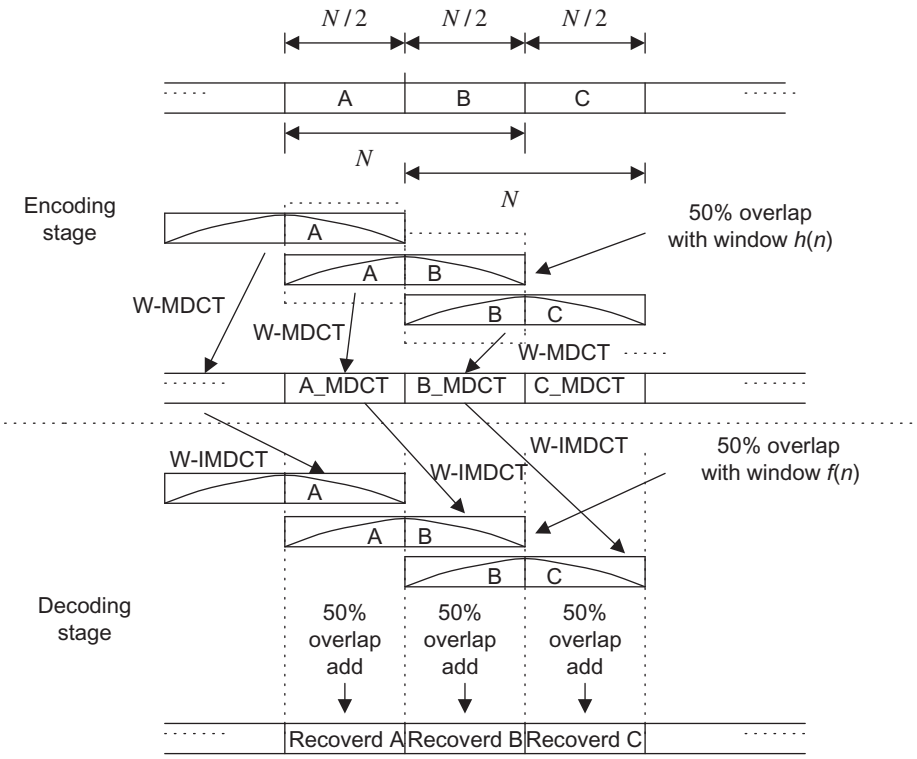

FIG. 10.14

Modified discrete cosine transform (MDCT).

2. Apply the windowed inverse MDCT (W-IMDCT) to each N MDCT coefficient block using Eq. (10.34) and then apply a decoding window function $f(n)$ to reduce the artifacts at the block edges:

$$x(n) = \frac{1}{N} f(n) \sum_{k=0}^{N-1} X_{MDCT}(k) \cos\left[\frac{2\pi}{N}(n + 0.5 + N/4)(k + 0.5)\right] \text{ for } n = 0, 1, \cdots, N-1. \tag{10.34}$$

Note that the recovered sequence contains the overlap portion. As shown in Fig. 10.14, if a decoded block has the decoded subblocks A and B, the next one would have subblocks B and C, where the subblock B is an overlapped block. The procedure continues.

3. Reconstruct the subblock B using the overlap and add operation, as shown in Fig. 10.14, where two subblocks labeled B are overlapped and added to generate the recovered subblock B. Note that the first subblock B comes from the recovered N samples containing A and B, while the second subblock B belongs to the next recovered N samples containing B and C.

In order to obtain the perfect reconstruction, that is, the full cancellation of all aliasing introduced by the MDCT, the following two conditions must be met for selecting the window functions, in which one is used for encoding while the other is used for decoding (Princen and Bradley, 1986):

$$f\left(n+\frac{N}{2}\right)h\left(n+\frac{N}{2}\right)+f(n)h(n)=1 \tag{10.35}$$

$$f\left(n+\frac{N}{2}\right)h(N-n-1)-f(n)h\left(\frac{N}{2}-n-1\right)=0. \tag{10.36}$$

Here, we choose the following simple function for the W-MDCT given by

$$f(n)=h(n)=\sin\left(\frac{\pi}{N}(n+0.5)\right). \tag{10.37}$$

Eq. (10.37) must satisfy the conditions described in Eqs. (10.35) and (10.36). This will be left for an exercise in the problem section at the end of this chapter. The MATLAB functions **wmdcth()** and **wimdctf()** relate to this topic are listed in the MATLAB program section (Section 10.6). Now, let us examine the W-MDCT in Example 10.11.

EXAMPLE 10.11

Given the data sequence: 1, 2, −3, 4, 5, −6, 4, 5 …,

(a) Determine the W-MDCT coefficients for the first three blocks using a block size of 4.

(b) Determine the first two overlapped subblocks, and compare the results with the original data sequence using the W-MDCT coefficients in (a).

Solution:

(a) We divided the first two data blocks using the overlapping of two samples:

First block data: 1 2 −3 4

Second block data: −3 4 5 −6

Third block data: 5 −6 4 5

We apply the W-MDCT to get.

>> wmdct([1 2 −3 4])

ans = 1.1716 3.6569

>> wmdct([−3 4 5 −6])

ans = −8.0000 7.1716

>> wmdct([5 −6 4 5])

ans =−4.6569 −18.0710.

(b) We show the results from W-IWDCT as:

>> x1=wimdct([1.1716 3.6569])

x1 =−0.5607 1.3536 −1.1465 −0.4749

>> x2=wimdct([−8.0000 7.1716])

x2 = −1.8536 4.4749 2.1464 0.8891

>> x3=wimdct([−4.6569 −18.0711])

x3 =2.8536 −6.8891 5.1820 2.1465

Applying the overlap and add, we have.

>> [x1 0 0 0 0]+ [0 0 x2 0 0]+ [0 0 0 0 x3]

ans = −0.5607 1.3536 −3.0000 4.0000 5.0000 −6.0000 5.1820 2.1465

The recovered first two subblocks have values −3, 4, 5, −6 which are consistent with the input data.

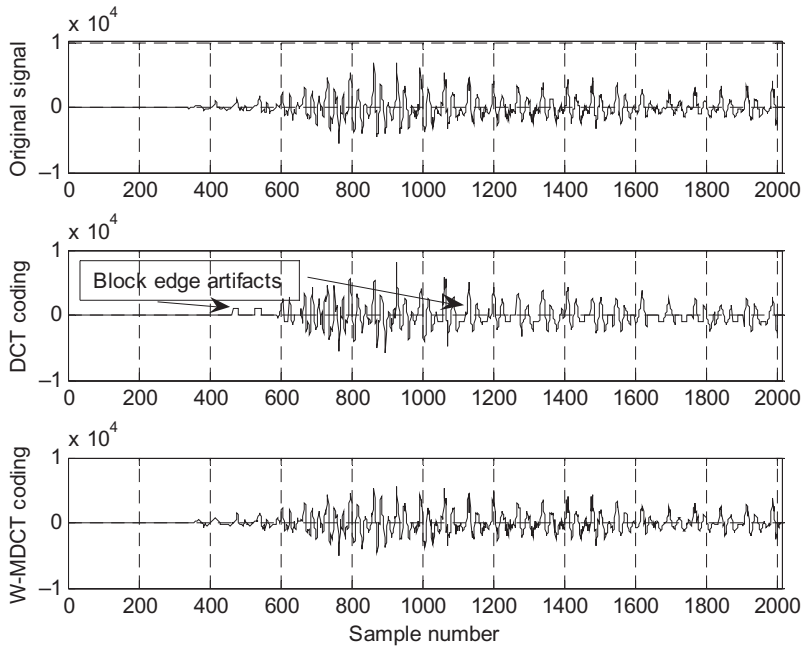

FIG. 10.15

Waveform coding using DCT and W-MDCT.

Fig. 10.15 shows coding of speech data we.dat using the DCT transform and W-MDCT transform. To be able to see the block edge artifacts, the following parameters are used for both DCT and W-MDCT transform coding:

Speech data: 16 bits per sample, 8000 samples per second

Block size: 16 samples

Scale factor: 2-bit nonlinear quantizer

Coefficients: 3-bit linear quantizer

Note that we assume a lossless scheme will further compress the quantized scale factors and coefficients. This stage does not affect the simulation results.

We use a 2-bit nonlinear quantizer with four levels to select the scale factor so that the block artifacts can be clearly displayed in Fig. 10.15. We also apply a 3-bit linear quantizer to the scaled coefficients for both DCT and W-MDCT coding. As shown in Fig. 10.15, the W-MDCT demonstrates significant improvement in smoothing out the block edge artifacts. The MATLAB simulation list is given in Programs 10.14–10.16 (Section 10.6), where Program 10.16 is the main program.

10.4.3 TRANSFORM CODING IN MPEG AUDIO

With the DCT and MDCT concepts developed, we now explore the MPEG audio data format, where the DCT plays a key role. MPEG was established in 1988 to develop a standard for delivery of digital video and audio. Since MPEG audio compression contains so many topics, we focus here

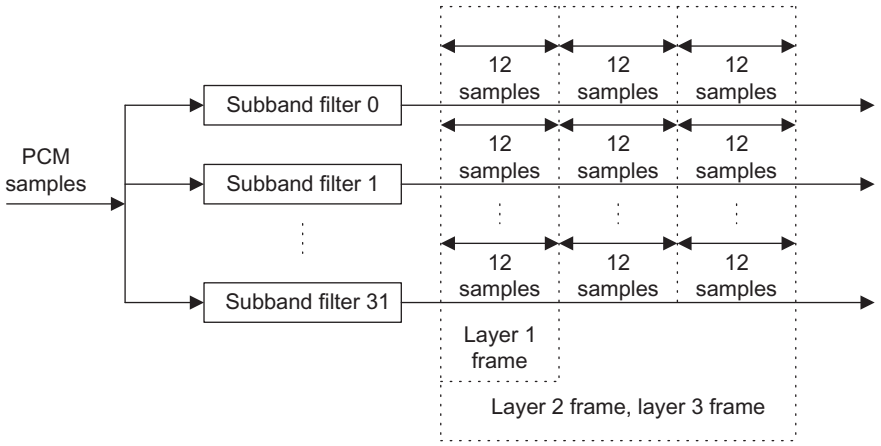

FIG. 10.16

MPEG audio frame size.

on examining its data format briefly, using the basic concepts developed in this book. Readers can further explore this subject by reading Pan's (1995) tutorial on MPEG audio compression, as well as Li et al. (2014).

Fig. 10.16 shows the MPEG audio frame. First, the input PCM samples—with a possible selection of sampling rates of 32, 44.1, and 48 kHz—are divided into 32 frequency subbands. All the subbands have equal bandwidths. The sum of their bandwidths covers up to the folding frequency, that is, $f_s/2$, which is the Nyquist limit in the DSP system. The subband filters are designed to minimized aliasing in frequency domain. Each subband filter outputs one sample for every 32 input PCM samples continuously, and forms a data segment for every 12 output samples. The purpose of the filter banks is to separate the data into different frequency bands so that the psychoacoustic model of the human auditory system (Yost, 2000) can be applied to activate the bit allocation scheme for a particular subband. The data frames are formed before quantization.

There are three types of data frames, as shown in Fig. 10.16. Layer 1 contains 32 data segments, each coming from one subband with 12 samples, so the total frame has 384 data samples. As we see, layers 2 and 3 have the same size data frame, consisting of 96 data segments, where each filter outputs 3 data segments of 12 samples. Hence, layer 2 or layer 3 each have 1152 data samples.

Next, let us examine briefly the content of each data frame, as shown in Fig. 10.17. Layer 1 contains 384 audio samples from 32 subbands, each having 12 samples. It begins with a header followed by a cyclic redundancy check (CRC) code. The numbers within parentheses indicate the possible number of bits to encode each field. The bit allocation informs the decoder of the number of bits used for each encoded sample in the specific band. Bit allocation can also be set to zero number of bits for a particular subband if analysis of the psychoacoustic model finds that the data in the band can be discarded without affecting the audio quality. In this way, the encoder can achieve more data compression. Each scale factor is encoded with 6 bits. The decoder will multiply the scale factor by the decoded quantizer output to get the quantized subband value. Use of the scale factor allows for utilization of the full range of the quantizer. The field "ancillary data" is reserved for "extra" information.

Layer 2 encoder takes 1152 samples per frame, with each subband channel having 3 data segments of 12 samples. These 3 data segments may have a bit allocation and up to three scale factors. Using one

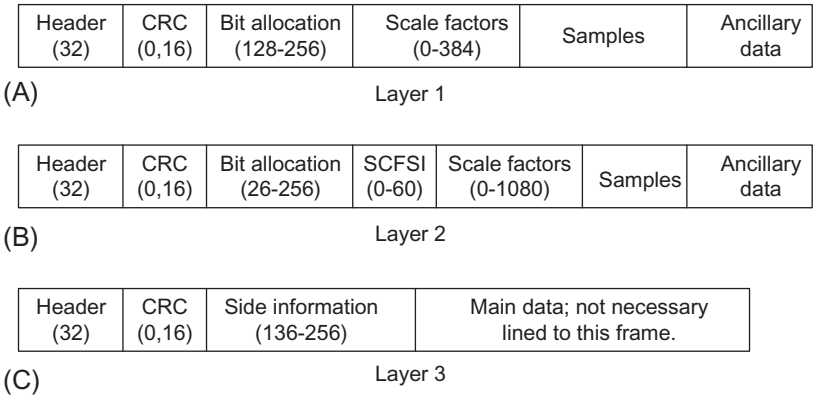

FIG. 10.17

MPEG audio frame formats.

scale factor for 3 data segments would be called when values of the scale factors per subband are sufficiently close and the encoder applies temporal noise masking (a type noise masking by human auditory system) to hide any distortion. In Fig. 10.17, the field "SCFSI" (scale-factor selection information) contains the information to inform the decoder. A different scale factor is used for each suabband channel when avoidance of audible distortion is required. The bit allocation can also provide a possible single compact code word to represent three consecutive quantized values.

The layer 3 frame contains side information and main data that come from Huffman encoding (lossless coding having an exact recovery) of the W-MDCT coefficients to gain improvement over the layers 1 and 2.

Fig. 10.18 shows the MPEG-1 layers 1 and 2 encoder, layer 3 encoder. For the MPEG-1 layers 1 and 2, the encoder examines the audio input samples using a 1024-point fast Fourier transform (FFT). The psychoacoustic model is analyzed based on the FFT coefficients. This includes the possible frequency masking (hiding noise in frequency domain) and the noise temporal masking (hiding noise in time domain). The result of the analysis of the psychoacoustic model instructs the bit allocation scheme.

The major difference in layer 3, called MP3 (the most popular format in the multimedia industry), is that it adopts the MDCT. First, the encoder can gain further data compression by transforming the data segments using DCT from each subband channel and then quantize the DCT coefficients, which, again, are losslessly compressed using Huffman encoding. As shown in Examples 10.8–10.11, since the DCT uses block-based processing, it produces block edge effects, where the beginning samples and ending samples show discontinuity and causes audible periodic noise. This periodic edge noise can be alleviated, as discussed in the previous section, by using the W-MDCT, in which there are 50% overlap between successive transform windows.

There are two sizes of windows. One has 36 samples and another has 12 samples used in MPEG-1 layer 3 (MP3) audio. The larger block length offers the better frequency resolution for low-frequency tonelike signals, hence it is used for the lowest two subbands. For the rest of the subbands, the shorter block is used, since it allows better time resolution for noise-like transient signals. Other improvements of MP3 over layers 1 and 2 include use of the scale-factor band, where the W-MDCT coefficients are regrouped from the original 32 uniformly divided subbands into 25 actual critical bands based on the human auditory system. Then the corresponding scale factors are assigned, and a nonlinear quantizer is used.

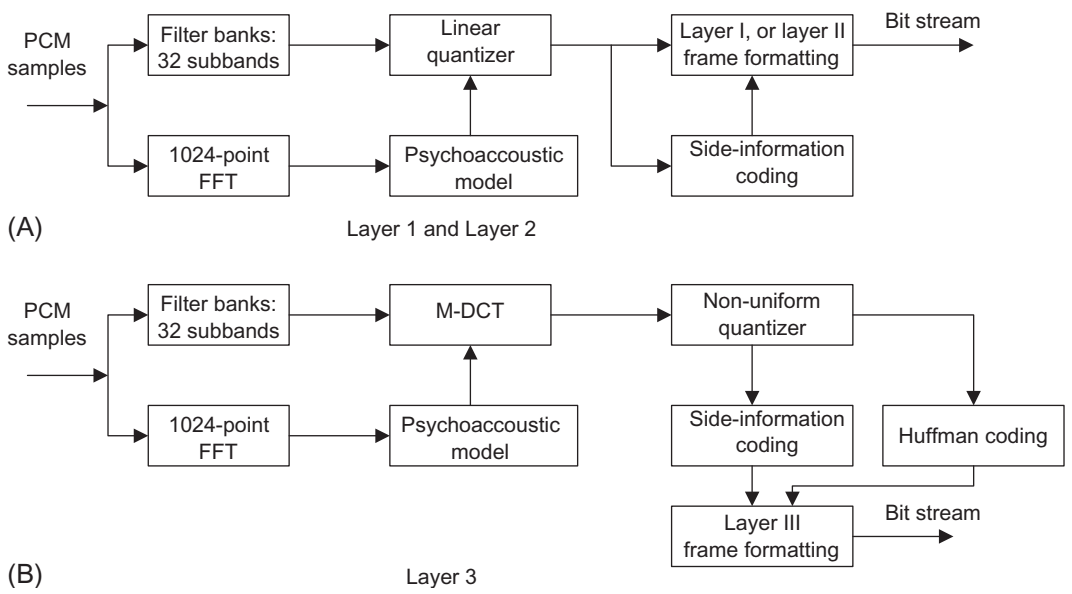

FIG. 10.18

Encoder block diagrams for layers 1 and 2 and for layer 3.

Finally, Huffman coding is applied to the quantizer outputs to obtain more compression. Particularly in CD-quality audio, MP3 (MPEG-1 layer 3) can achieve CRs varying from 12:1 to 8:1, corresponding to the bit rates from 128 to 192 kbps. Besides the use of DCT in MP3, MPEG-2 audio coding methods such as AC-2, AC-3, ATRAC, and PAC/MPAC also use W-MDCT coding. Readers can further explore these subjects in Brandenburg (1997) and Li et al. (2014).

10.5 SUMMARY

1. The linear midtread quantizer used in the PCM coding has an odd number of quantization levels, that is, $2^n - 1$. It accommodates the same decoded magnitude range for quantizing the positive and negative voltages.
2. Analog or digital μ-law compression improves coding efficiency. 8-bit μ-law compression of speech is equivalent to 12-bit linear PCM coding, with no difference in the sound quality. These methods are widely used in the telecommunications industry and multimedia system applications.
3. DPCM encodes the difference between the input sample and predicted sample using a predictor to achieve the coding efficiency.
4. DM coding is essentially a 1-bit DPCM.
5. ADPCM is similar to DPCM except that the predictor transfer function has six zeros and two poles and is an adaptive filter. ADPCM is superior to 8-bit μ-law compression, since it provides the same sound quality with only 4 bits per code.
6. Data compression performance is measured in terms of the data CR and the bit rate.

7. The DCT decomposes a block of data to the DC coefficient (average) and AC coefficients (fluctuation) so that different numbers of bits are assigned to encode DC coefficients and AC coefficients to achieve data compression.
8. W-MDCT alleviates the block effects introduced by the DCT.
9. The MPEG-1 audio formats such as MP3 (MPEG-1, layer 3) include W-MDCT, filter banks, a psychoacoustic model, bit allocation, a nonlinear quantizer, and Huffman lossless coding.

10.6 MATLAB PROGRAMS

Program 10.1 MATLAB program for the linear midtread quantizer.

```
clear all;close all
disp('load speech: We');
load we.dat;                    % Provided by your instructor
sig=we;
lg=length(sig);                 % Length of the speech data
t=[0:1:lg-1];                   % Time index
sig=5*sig/max(abs(sig));        % Normalize signal to the range between -5 to 5
Emax=max(abs(sig));
Erms=sqrt(sum(sig. * sig) / length(sig))
k=Erms/Emax
disp('20*log10(k)≥');
k=20*log10(k)
bits=input('input number of bits≥');
lg=length(sig);
% Encoding
for x=1:lg
  [indx(x) qy]=mtrdenc(bits, 5, sig(x));
end
disp('Finished and transmitted');
% Decoding
for x=1:lg
  qsig(x)=mtrddec(bits, 5, indx(x));
end
disp('decoding finished');
  qerr=sig-qsig;          % Calculate quantization errors
subplot(3,1,1);plot(t, sig);grid
ylabel('Speech');axis([0 length(we) -5 5]);
subplot(3,1,2);plot(t, qsig);grid
ylabel('Quantized speech');axis([0 length(we) -5 5]);
subplot(3,1,3);plot(qerr);grid
axis([0 length(we) -0.5 0.5]);
ylabel('Qunatized error');xlabel('Sample number');
disp('signal to noise ratio due to quantization noise')
snr(sig,qsig);            % Calculate signal to noise ratio due to quantization
```

Program 10.2. MATLAB function for midread quantizer encoding.

```
function [indx, pq] =mtrdenc(NoBits,Xmax,value)
% function pq=mtrdenc(NoBits, Xmax, value)
% This routine is created for simulation of midtread uniform quatizer.
%
% NoBits: number of bits used in quantization.
% Xmax: overload value.
% value: input to be quantized.
% pq: output of quantized value
% indx: integer index
%
% Note: the midtread method is used in this quantizer.
%
 if NoBits == 0
  pq=0;
  indx=0;
 else
  delta=2*abs(Xmax)/(2^NoBits-1);
  Xrmax=delta*(2^NoBits/2-1);
  if abs(value)>= Xrmax
    tmp=Xrmax;
  else
    tmp=abs(value);
  end
  indx=round(tmp/delta);
  pq=indx*delta;
  if value <0
    pq=-pq;
    indx=-indx;
  end
end
```

Program 10.3. MATLAB function for midtread quantizer decoding.

```
function pq=mtrddec(NoBits,Xmax,indx)
% function pq=mtrddec(NoBits, Xmax, value)
% This routine is dequantizer.
%
% NoBits: number of bits used in quantization.
% Xmax: overload value.
% pq: output of quantized value
% indx: integer index
%
% Note: the midtread method is used in this quantizer.
%
 delta=2*abs(Xmax)/(2^NoBits-1);
 pq=indx*delta;
```

Program 10.4. MATLAB program for μ-law encoding and decoding.

```
close all; clear all
disp('load speech file');
load we.dat;                  % Provided by your instructor
lg=length(we);                % Length of the speech data
we=5*we/max(abs(we));         % Normalize the speech data
we_nor=we/max(abs(we));       % Normalization
t=[0:1:lg-1];                 % Time index
disp('mulaw companding')
mu=input('input mu≥');
for x=1:lg
  ymu(x)=mulaw(we_nor(x),1,mu);
end
disp('finished mu-law companding');
disp('start to quantization')
bits=input('input bits≥');
% Midtread quantization and encoding
for x=1:lg
  [indx(x) qy]=mtrdenc(bits, 1, ymu(x));
end
disp('finished and transmitted');
%
% Midtread decoding
for x=1:lg
  qymu(x)=mtrddec(bits, 1, indx(x));
end
disp('expander');
for x=1:lg
  dymu(x)=muexpand(qymu(x),1,mu)*5;
end
disp('finished')
qerr=dymu-we; % Quantization error
subplot(3,1,1);plot(we);grid
ylabel('Speech');axis([0 length(we) -5 5]);
subplot(3,1,2);plot(dymu);grid
ylabel('recovered speech');axis([0 length(we) -5 5]);
subplot(3,1,3);plot(qerr);grid
ylabel('Quantized error');xlabel('Sample number');
axis([0 length(we) -1 1]);
snr(we,dymu);            % Calculate signal to noise ratio due to quantization
```

Program 10.5. MATLAB function for μ-law companding.

```
Function qvalue=mulaw(vin, vmax, mu)
% This function performs mu-law companding
% Usage:
% function qvalue=mulaw(vin, vmax, mu)
% vin=input value
```

```
% vmax=maximum input amplitude
% mu=parameter for controlling the degree of compression which must be the same as
mu-law expander
% qvalue=output value from the mu-law compander
%
       vin=vin/vmax;       % Normalization
% mu-law companding
       qvalue=vmax*sign(vin)*log(1+mu*abs(vin))/log(1+mu);
```

Program 10.6. MATLAB program for μ-law expanding.

```
function rvalue=muexpand(y,vmax, mu)
% This function performs mu-law expanding
% Usage:
% function rvalue=muexpand(y,vmax, mu)
% y=input signal
% vmax=maximum input amplitude
% mu=parameter for controlling the degree of compression, which must be the same
% as the mu-law compander
% rvalue=output value from the mu-law expander
%
       y=y/vmax;                % Normalization
% mu-law expanding
       rvalue=sign(y)*(vmax/mu)*((1+mu)âbs(y) -1);
```

Program 10.7. MATLAB function for calculation of signal to quantization noise ratio (SNR).

```
function snr=calcsnr(speech, qspeech)
% function snr=calcsnr(speech, qspeech)
% This routine is created for calculation of SNR
%
% speech: original speech waveform.
% qspeech: quantized speech.
% snr: output SNR in dB.
%
% Note: midrise method is used in this quantizer.
%
 qerr=speech-qspeech;
 snr=10*log10(sum(speech.*speech)/sum(qerr.*qerr))
```

Program 10.8. Main program for the digital μ-law encoding and decoding.

```
load we12b.dat
for i=1:1:length(we12b)
    code8b(i)=dmuenc(12, we12b(i)); % Encoding
    qwe12b(i)=dmudec(code8b(i));    % Decoding
end
subplot(4,1,1),plot(we12b);grid
ylabel('a');axis([0 length(we12b) -1024 1024]);
subplot(4,1,2),plot(code8b);grid
ylabel('b');axis([0 length(we12b) -128,128]);
subplot(4,1,3),plot(qwe12b);grid
ylabel('c');axis([0 length(we12b) -1024 1024]);
subplot(4,1,4),plot(qwe12b-we12b);grid
ylabel('d');xlabel('Sample number');axis([0 length(we12b) -40 40]);
```

Program 10.9. The digital μ-law compressor.

```
function [cmp_code]=dmuenc(NoBits, value)
% This routine is created for simulation of 12-bit mu law compression.
% function [cmp_code]=dmuenc(NoBits, value)
% NoBits=number of bits for the data
% value=input value
% cmp_code=output code
%
 scale=NoBits-12;
 value=value*2^(-scale); % Scale to 12 bit
 if (abs(value)>=0) & (abs(value)<16)
     cmp_code=value;
 end
 if (abs(value)>=16) & (abs(value)<32)
     cmp_code=sgn(value)*(16+fix(abs(value)-16));
 end
 if (abs(value)>=32) & (abs(value)<64)
     cmp_code=sgn(value)*(32+fix((abs(value) -32)/2));
 end
 if (abs(value)>=64) & (abs(value)<128)
     cmp_code=sgn(value)*(48+fix((abs(value) -64)/4));
 end
 if (abs(value)>=128) & (abs(value)<256)
     cmp_code=sgn(value)*(64+fix((abs(value) -128)/8));
 end
 if (abs(value)>=256) & (abs(value)<512)
     cmp_code=sgn(value)*(80+fix((abs(value) -256)/16));
end
if (abs(value)>=512) & (abs(value)<1024)
```

```
    cmp_code=sgn(value)*(96+fix((abs(value) -512)/32));
end
if (abs(value)>=1024) & (abs(value)<2048)
    cmp_code=sgn(value)*(112+fix((abs(value) -1024)/64));
end
```

Program 10.10. The digital μ-law expander.

```
function [value]=dmudec(cmp_code)
% This routine is created for simulation of 12-bit mu law decoding.
% Usage:
% unction [value]=dmudec(cmp_code)
% cmp_code=input mu-law encoded code
% value=recovered output value
%
 if (abs(cmp_code)>=0) & (abs(cmp_code)<16)
    value=cmp_code;
 end
 if (abs(cmp_code)>=16) & (abs(cmp_code)<32)
    value=sgn(cmp_code)*(16+(abs(cmp_code)-16));
 end
 if (abs(cmp_code)>=32) & (abs(cmp_code)<48)
    value=sgn(cmp_code)*(32+(abs(cmp_code)-32)*2+1);
 end
 if (abs(cmp_code)>=48) & (abs(cmp_code)<64)
    value=sgn(cmp_code)*(64+(abs(cmp_code)-48)*4+2);
 end
 if (abs(cmp_code)>=64) & (abs(cmp_code)<80)
    value=sgn(cmp_code)*(128+(abs(cmp_code)-64)*8+4);
 end
 if (abs(cmp_code)>=80) & (abs(cmp_code)<96)
    value=sgn(cmp_code)*(256+(abs(cmp_code)-80)*16+8);
 end
 if (abs(cmp_code)>=96) & (abs(cmp_code)⟨112)
    value=sgn(cmp_code)*(512+(abs(cmp_code)-96)*32+16);
 end
 if (abs(cmp_code)>=112) & (abs(cmp_code)<128)
    value=sgn(cmp_code)*(1024+(abs(cmp_code)-112)*64+32);
 end
```

Program 10.11. Main program for ADPCM coding.

```
% This program is written for off-line simulation
% file: adpcm.m
clear all; close all
load we.dat                % Provided by the instructor
```

```
speech=we;
desig=speech;
lg=length(desig);        % Length of speech data
enc=adpcmenc(desig);     % ADPCM encoding
%ADPCM finished
dec=adpcmdec(enc);       % ADPCM decoding
snrvalue=snr(desig,dec) % Calculate signal to noise ratio due to quantization
subplot(3,1,1);plot(desig);grid;
ylabel('Speech');axis([0 length(we) -8000 8000]);
subplot(3,1,2);plot(dec);grid;
ylabel('Quantized speech');axis([0 length(we) -8000 8000]);
subplot(3,1,3);plot(desig-dec);grid
ylabel('Quantized error');xlabel('Sample number');
axis([0 length(we) -1200 1200]);
```

Program 10.12 MATLAB function for ADPCM encoding.

```
function iiout = adpcmenc(input)
% This function performs ADPCM encoding
% function iiout = adpcmenc(input)
% Usage:
% input = input value
% iiout = output index
%
% Quantization tables
fitable = [0 0 0 1 1 1 1 3 7];
witable = [-0.75 1.13 2.56 4.00 7.00 12.38 22.19 70.13 ];
qtable = [ -0.98 0.62 1.38 1.91 2.34 2.72 3.12 ];
invqtable = [0.031 1.05 1.66 2.13 2.52 2.91 3.32 ];
lgth = length(input);
sr = zeros(1,2); pk = zeros(1,2);
a = zeros(1,2); b = zeros(1,6);
dq = zeros(1,6); ii= zeros(1,lgth);
y=0; ap = 0; al = 0; yu=0; yl = 0; dms = 0; dml = 0; tr = 0; td = 0;
for k = 1:lgth
sl = input(k);
%
% predict zeros
%
sez = b(1)*dq(1);
for i=2:6
sez = sez + b(i)*dq(i);
end
se = a(1)*sr(1)+a(2)*sr(2)+ sez;
d = sl - se;
%
% Perform quantization
%
dqq = log10(abs(d))/log10(2.0)-y;
```

```
 ik= 0;
 for i=1:7
 if dqq > qtable(i)
       ik = i;
 end
end
if d < 0
 ik = -ik;
end
ii(k) = ik;
yu = (31.0/32.0)*y + witable(abs(ik)+1)/32.0;
if yu > 10.0
 yu = 10.0;
end
if yu < 1.06
 yu = 1.06;
end
yl = (63.0/64.0)*yl+yu/64.0;
%
%Inverse quantization
%
if ik == 0
 dqq = 2^(-y);
else
 dqq = 2^(invqtable(abs(ik))+y);
end
if ik < 0
 dqq = -dqq;
end
srr = se + dqq;
dqsez = srr+sez-se;
%
% Update state
%
pk1 = dqsez;
%
% Obtain adaptive predictor coefficients
%
 if tr == 1
  a = zeros(1,2); b = zeros(1,6);
  tr = 0;
 td = 0; %Set for the time being
 else
% Update predictor poles
% Update a2 first
 a2p = (127.0/128.0)*a(2);
 if abs(a(1)) <= 0.5
       fa1 = 4.0*a(1);
else
       fa1 = 2.0*sgn(a(1));
end
 a2p=a2p+(sign(pk1)*sgn(pk(1))-fa1*sign(pk1)*sgn(pk(2)))/128.0;
```

```
 if abs(a2p) > 0.75
     a2p = 0.75*sgn(a2p);
end
     a(2) = a2p;
%
% Update a1
 a1p = (255.0/256.0)*a(1);
      a1p = a1p + 3.0*sign(pk1)*sgn(pk(2))/256.0;
 if abs(a1p) > 15.0/16.0-a2p
      a1p = 15.0/16.0 -a2p;
end
  a(1) = a1p;
%
% Update b coefficients
%
for i=1:6
 b(i) = (255.0/256.0)*b(i)+sign(dqq)*sgn(dq(i))/128.0;
end
if a2p < -0.7185
 td = 1;
else
  td = 0;
end
if a2p < -0.7185 & abs(dq(6)) > 24.0*2^(yl)
 tr = 1;
else
  tr = 0;
end
for i=6:-1:2
 dq(i) = dq(i-1);
end
dq(1) = dqq; pk(2) = pk(1); pk(1) = pk1; sr(2) = sr(1); sr(1) = srr;
%
% Adaptive speed control
%
dms = (31.0/32.0)*dms; dms = dms + fitable(abs(ik)+1)/32.0;
dml = (127.0/128.0)*dml; dml = dml + fitable(abs(ik)+1)/128.0;
if ap > 1.0
 al = 1.0;
else
 al = ap;
end
ap = (15.0/16.0)*ap;
if abs(dms-dml) >= dml/8.0
  ap = ap + 1/8.0;
end
if y < 3
  ap = ap +1/8.0;
end
if td == 1
  ap = ap + 1/8.0;
end
```

```
if tr == 1
   ap = 1.0;
end
y = al*yu + (1.0-al)*yl;
end
end
iiout = ii;
```

Program 10.13. MATLAB function for ADPCM decoding.

```
function iiout = adpcmdec(ii)
% This function performs ADPCM decoding
% function iiout = adpcmdec(ii)
% Usage:
% ii = input ADPCM index
% iiout = decoded output value
%
% Quantization tables:
fitable = [0 0 0 1 1 1 1 3 7];
witable = [-0.75 1.13 2.56 4.00 7.00 12.38 22.19 70.13 ];
qtable = [ -0.98 0.62 1.38 1.91 2.34 2.72 3.12 ];
invqtable = [0.031 1.05 1.66 2.13 2.52 2.91 3.32 ];

lgth = length(ii);
sr = zeros(1,2); pk = zeros(1,2);
a = zeros(1,2); b = zeros(1,6);
dq = zeros(1,6); out= zeros(1,lgth);
y=0; ap = 0; al = 0; yu=0; yl = 0; dms = 0; dml = 0; tr = 0; td = 0;
for k = 1:lgth
%
sez = b(1)*dq(1);
for i=2:6
sez = sez + b(i)*dq(i);
end
se = a(1)*sr(1)+a(2)*sr(2)+ sez;
%
%Inverse quantization
%
ik = ii(k);
yu = (31.0/32.0)*y + witable(abs(ik)+1)/32.0;
if yu > 10.0
   yu = 10.0;
end
if yu < 1.06
   yu = 1.06;
end
yl = (63.0/64.0)*yl+yu/64.0;
if ik == 0
   dqq = 2^(-y);
```

```
else
   dqq = 2^(invqtable(abs(ik))+y);
end
if ik < 0
   dqq = -dqq;
end
srr = se + dqq;
dqsez = srr+sez-se;
out(k) =srr;
%
% Update state
%
pk1 = dqsez;
%
% Obtain adaptive predictor coefficients
%
  if tr == 1
         a = zeros(1,2);
         b = zeros(1,6);
         tr = 0;
         td = 0; %Set for the time being
         else
% Update predictor poles
% Update a2 first;
   a2p = (127.0/128.0)*a(2);
   if abs(a(1)) <= 0.5
      fa1 = 4.0*a(1);
   else
      fa1 = 2.0*sgn(a(1));
   end
   a2p=a2p+(sign(pk1)*sgn(pk(1))-fa1*sign(pk1)*sgn(pk(2)))/128.0;
   if abs(a2p) > 0.75
      a2p = 0.75*sgn(a2p);
   end
      a(2) = a2p;
%
% Update a1
  a1p = (255.0/256.0)*a(1);
      a1p = a1p + 3.0*sign(pk1)*sgn(pk(2))/256.0;
  if abs(a1p) > 15.0/16.0-a2p
       a1p = 15.0/16.0-a2p;
  end
a(1) = a1p;
%
% Update b coefficients
%
for i= 1: 6
  b(i) = (255.0/256.0)*b(i)+sign(dqq)*sgn(dq(i))/128.0;
end
if a2p < -0.7185
  td = 1;
else
```

```
  td = 0;
end
if a2p < -0.7185 & abs(dq(6)) > 24.0*2^(yl)
  tr = 1;
else
  tr = 0;
end
for i=6:-1:2
  dq(i) = dq(i-1);
end
dq(1) = dqq; pk(2) = pk(1); pk(1) = pk1; sr(2) = sr(1); sr(1) = srr;
%
% Adaptive speed control
%
dms = (31.0/32.0)*dms;
dms = dms + fitable(abs(ik)+1)/32.0;
dml = (127.0/128.0)*dml;
dml = dml + fitable(abs(ik)+1)/128.0;
if ap > 1.0
  al = 1.0;
else
  al = ap;
end
ap = (15.0/16.0)*ap;
if abs(dms-dml) >= dml/8.0
   ap = ap + 1/8.0;
end
if y < 3
   ap = ap +1/8.0;
end
if td == 1
  ap = ap + 1/8.0;
end
if tr == 1
  ap = 1.0;
end
y = al*yu + (1.0-al)*yl;
end
end
iiout = out;
```

Program 10.14. W-MDCT function.

```
function [ tdac_coef ] = wmdct(ipsig)
%
% This function transforms the signal vector using the W-MDCT
% Usage:
% ipsig: inpput signal block of N samples (N=even number)
% tdac_coe: W-MDCT coefficients (N/2 coefficients)
```

```
%
N = length(ipsig);
NN =N;
for i=1:NN
  h(i) = sin((pi/NN)*(i-1+0.5));
end
for k=1:N/2
  tdac_coef(k) = 0.0;
  for n=1:N
    tdac_coef(k) = tdac_coef(k) + ...
    h(n)*ipsig(n)*cos((2*pi/N)*(k-1+0.5)*(n-1+0.5+N/4));
    end
end
tdac_coef=2*tdac_coef;
```

Program 10.15. Inverse W-IMDCT function.

```
function [ opsig ] = wimdct(tdac_coef )
%
% This function transform the W-MDCT coefficients back to the signal
% Usage:
% tdac_coeff: N/2 W-MDCT coefficients
% opsig: output signal block with N samples
%
N = length(tdac_coef );
tmp_coef = ((-1)^(N+1))*tdac_coef(N:-1:1);
tdac_coef = [ tdac_coef tmp_coef];
N = length(tdac_coef );
 NN =N;
for i=1:NN
  f(i) = sin((pi/NN)*(i-1+0.5));
end
for n=1:N
  opsig(n) = 0.0;
  for k=1:N
   opsig(n) = opsig(n) + ...
   tdac_coef(k)*cos((2*pi/N)*(k-1+0.5)*(n-1+0.5+N/4));
  end
  opsig(n) = opsig(n)*f(n)/N;
end
```

Program 10.16. Waveform coding using DCT and W-MDCT.

```
% Waveform coding using DCT and MDCT for a block size of 16 samples
% Main program
close all; clear all
```

```
load we.dat              % Provided by the instructor
% Create a simple 3 bit scale fcator
scalef4bits=[1 2 4 8 16 32 64 128 256 512 1024 2048 4096 8192 16384 32768];
scalef3bits=[256 512 1024 2048 4096 8192 16384 32768];
scalef2bits=[4096 8192 16384 32768];
scalef1bit=[16384 32768];
scalef=scalef1bit;
nbits =3;
% Ensure the block size to be 16 samples.
x=[we zeros(1,16-mod(length(we),16))];
Nblock=length(x)/16;
DCT_code=[]; scale_code=[];
% DCT transform coding
% Encoder
 for i=1:Nblock
  xblock_DCT=dct(x((i-1)*16+1:i*16));
  diff=abs(scalef-(max(abs(xblock_DCT))));
  iscale(i)=min(find(diff<=min(diff ))); %find a scale factor
  xblock_DCT=xblock_DCT/scalef(iscale(i)); % scale the input vetor
   for j=1:16
   [DCT_coeff(j) pp]=biquant(nbits,-1,1,xblock_DCT(j));
   end
   DCT_code=[DCT_code DCT_coeff ];
 end
%Decoder
Nblock=length(DCT_code)/16;
xx=[];
for i=1:Nblock
    DCT_coefR=DCT_code((i-1)*16+1:i*16);
   for j=1:16
      xrblock_DCT(j)=biqtdec(nbits,-1,1,DCT_coefR(j));
   end
     xrblock=idct(xrblock_DCT.*scalef(iscale(i)));
   xx=[xx xrblock];
end
% Transform coding using MDCT
xm=[zeros(1,8) we zeros(1,8-mod(length(we),8)), zeros(1,8)];
Nsubblock=length(x)/8;
MDCT_code=[];
% Encoder
 for i=1:Nsubblock
  xsubblock_DCT=wmdct(xm((i-1)*8+1:(i+1)*8));
  diff=abs(scalef-max(abs(xsubblock_DCT)));
  iscale(i)=min(find(diff<=min(diff ))); %find a scale factor
  xsubblock_DCT=xsubblock_DCT/scalef(iscale(i)); % scale the input vetor
  for j=1:8
   [MDCT_coeff(j) pp]=biquant(nbits,-1,1,xsubblock_DCT(j));
   end
   MDCT_code=[MDCT_code MDCT_coeff];
end
%Decoder
% Recover thr first subblock
```

```
Nsubblock=length(MDCT_code)/8;
xxm=[];
MDCT_coeffR=MDCT_code(1:8);
for j=1:8
      xmrblock_DCT(j)=biqtdec(nbits,-1,1,MDCT_coeffR(j));
end
xmrblock=wimdct(xmrblock_DCT*scalef(iscale(1)));
xxr_pre=xmrblock(9:16) % recovered first block for overlap and add
for i=2:Nsubblock
   MDCT_coeffR=MDCT_code((i-1)*8+1:i*8);
   for j=1:8
      xmrblock_DCT(j)=biqtdec(nbits,-1,1,MDCT_coeffR(j));
   end
     xmrblock=wimdct(xmrblock_DCT*scalef(iscale(i)));
   xxr_cur=xxr_pre+xmrblock(1:8); % overlap and add
   xxm=[xxm xxr_cur];
   xxr_pre=xmrblock(9:16);         % set for the next overlap
end

subplot(3,1,1);plot(x,'k');grid; axis([0 length(x) -10000 10000])
ylabel('Original signal');
subplot(3,1,2);plot(xx,'k');grid;axis([0 length(xx) -10000 10000]);
ylabel('DCT coding')
subplot(3,1,3);plot(xxm,'k');grid;axis([0 length(xxm) -10000 10000]);
ylabel('W-MDCT coding');
xlabel('Sample number');
```

Program 10.17. Sign function.

```
function sgn = sgn(sgninp)
%
% Sign function
% if signp >=0 then sign=1
% else sign =-1
%
 if sgninp >= 0
    opt = 1;
 else
   opt = -1;
 end
 sgn = opt;
```

10.7 PROBLEMS

10.1 For the 3-bit midtread quantizer described in Fig. 10.1, and the analog signal range from −2.5 to 2.5 V, determine
- **(a)** the quantization step size
- **(b)** the binary codes, recovered voltages, and quantization errors when each input is 1.6 and −0.2 V.

10.2 For the 3-bit midtread quantizer described in Fig. 10.1, and the analog signal range from −4 to 4 V, determine
- **(a)** the quantization step size
- **(b)** the binary codes, recovered voltages, and quantization errors when each input is −2.6 and 0.1 V.

10.3 For the 3-bit midtread quantizer described in Fig. 10.1, and the analog signal range from −5 to 5 V, determine
- **(a)** the quantization step size
- **(b)** the binary codes, recovered voltages, and quantization errors when each input is −2.6 and 3.5 V.

10.4 For the 3-bit midtread quantizer described in Fig. 10.1, and the analog signal range from −10 to 10 V, determine
- **(a)** the quantization step size
- **(b)** the binary codes, recovered voltages, and quantization errors when each input is −5, 0, and 7.2 V.

10.5 For the μ-law compression and expanding process shown in Fig. 10.3 with $u = 255$ and the 3-bit midtread quantizer described in Fig. 10.1, with the analog signal range from −2.5 to 2.5 V, determine the binary codes, recovered voltages, and quantization errors when each input is 1.6 and −0.2 V.

10.6 For the μ-law compression and expanding process shown in Fig. 10.3 with $u = 255$ and the 3-bit midtread quantizer described in Fig. 10.1, with the analog signal range from −4 to 4 V, determine the binary codes, recovered voltages, and quantization errors when each input is −2.6 and 0.1 V.

10.7 For the μ-law compression and expanding process shown in Fig. 10.3 with $u = 255$ and the 3-bit midtread quantizer described in Fig. 10.1, with the analog signal range from −5 to 5 V, determine the binary codes, recovered voltages, and quantization errors when each input is −2.6 and 3.5 V.

10.8 For the μ-law compression and expanding process shown in Fig. 10.3 with $u = 255$ and the 3-bit midtread quantizer described in Fig. 10.1, with the analog signal range from −10 to 10 V, determine the binary codes, recovered voltages, and quantization errors when each input is −5, 0, and 7.2 V.

10.9 In a digital companding system, encode each of the following 12-linear PCM codes into the 8-bit compressed PCM code.
- **(a)** 0 0 0 0 0 0 0 1 0 1 0 1
- **(b)** 1 0 1 0 1 1 1 0 1 0 1 0

10.10 In a digital companding system, decode each of the following 8-bit compressed PCM codes into the 12-bit linear PCM code.
(a) 0 0 0 0 0 1 1 1
(b) 1 1 1 0 1 0 0 1

10.11 In a digital companding system, encode each of the following 12-linear PCM codes into the 8-bit compressed PCM code.
(a) 0 0 1 0 1 0 1 0 1 0 1 0
(b) 1 0 0 0 0 0 0 0 1 1 0 1

10.12 In a digital companding system, decode each of the following 8-bit compressed PCM codes into the 12-bit linear PCM code.
(a) 0 0 1 0 1 1 0 1
(b) 1 0 0 0 0 1 0 1

10.13 For a 3-bit DPCM encoding system with the following specifications (Fig. 10.19):

Encoder scheme: $\tilde{x}(n) = \hat{x}(n-1)$ (predictor)
$d(n) = x(n) - \tilde{x}(n)$
$d_q(n) = Q[d(n)] =$ Quantizer in Table 10.9
$\hat{x}(n) = \tilde{x}(n) + d_q(n)$

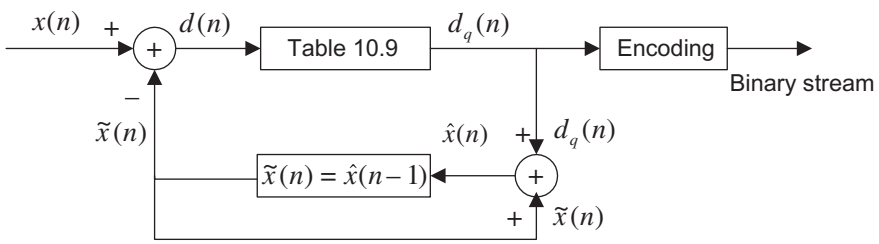

FIG. 10.19

DPCM encoding in Problem 10.13.

Table 10.9 Quantization Table for the 3-Bit Quantizer in Problem 10.13

Binary Code	Quantization Value $d_q(n)$	Subrange in $d(n)$
0 1 1	−11	$-15 \le d(n) < -7$
0 1 0	−5	$-7 \le d(n) < -3$
0 0 1	−2	$-3 \le d(n) < -1$
0 0 0	0	$-1 \le d(n) < 0$
1 0 0	0	$0 \le d(n) \le 1$
1 0 1	2	$1 < d(n) \le 3$
1 1 0	5	$3 < d(n) \le 7$
1 1 1	11	$7 < d(n) \le 15$

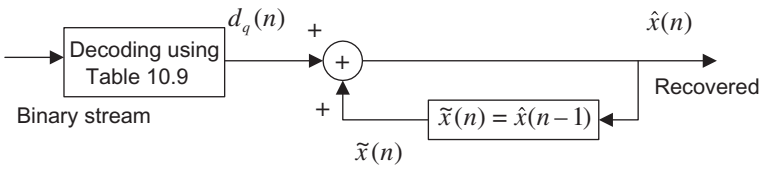

FIG. 10.20

DPCM decoding in Problem 10.14.

The 5-bit input data $x(0) = -6$, $x(1) = -8$, and $x(2) = -13$, perform DPCM encoding to produce the binary code for each input data.

10.14 For a 3-bit DPCM decoding system as shown in Fig. 10.20 with the following specifications:

Decoding scheme: $\tilde{x}(n) = \hat{x}(n-1)$ (predictor)
$d_q(n) =$ Quantizer in Table 10.9.
$\hat{x}(n) = \tilde{x}(n) + d_q(n)$

Received 3 binary codes: 110, 100, 101, perform DCPM decoding to recover each digital value using its binary code.

10.15 For a 3-bit DPCM encoding system shown in Problem 10.13 and the given data: the 5-bit input data: $x(0) = 6$, $x(1) = 8$, and $x(2) = 13$, perform DPCM encoding to produce the binary code for each input data.

10.16 For a 3-bit DPCM decoding system shown in Problem 10.14 and the received data: 010, 000, 001, perform DCPM decoding to recover each digital value using its binary code.

10.17 Assume that a speech waveform is sampled at 8 kHz and each sample is encoded by 16 bits, determine the CR for each of the encoding methods.

(a) Noncompression;
(b) Standard μ-law compression (8 bits per sample);
(c) Standard ADPCM encoding (4 bits per sample).

10.18 Suppose that a speech waveform is sampled at 8 kHz and each sample is encoded by 16 bits, determine the bit rate for each of the encoding methods.

(a) Noncompression;
(b) Standard μ-law companding (8 bits per sample);
(c) Standard ADPCM encoding (4 bits per sample).

10.19 Assume that an audio waveform is sampled at 44.1 kHz and each sample is encoded by 16 bits, determine the CR for each of the encoding methods.

(a) noncompression;
(b) standard μ-law compression (8 bits per sample);
(c) standard ADPCM encoding (4 bits per sample).

10.20 Suppose that an audio waveform is sampled at 44.1 kHz and each sample is encoded by 12 bits, determine the bit rate for each of the encoding methods.

(a) noncompression;
(b) standard μ-law companding (8 bits per sample);
(c) standard ADPCM encoding (4 bits per sample).

10.21 Speech is sampled at 8 kHz and each sample is encoded by 16 bits. The telephone system can transport the digital voice channel over a digital link having a capacity of 1.536 MBPS. Determine the number of channels that the phone company can carry for each of the following encoding methods:

(a) noncompression;

(b) standard 8-bit μ-law companding (8 bits per sample);

(c) standard ADPCM encoding (4 bits per sample).

10.22 Given the following input data:

$$x(0) = 25,\ x(1) = 30,\ x(2) = 28,\ \text{and}\ x(3) = 25,$$

determine the DCT coefficients.

10.23 Given the following input data:

$$x(0) = 25,\ \text{and}\ x(1) = 30,$$

determine the DCT coefficients.

10.24 Given the following input data:

$$x(0) = 25, x(1) = 30,\ x(2) = 28,\ x(3) = 25,$$
$$x(4) = 10,\ x(5) = 0,\ x(6) = 0,\ \text{and}\ x(7) = 0,$$

determine the DCT coefficient $X_{DCT}(0)$, $X_{DCT}(2)$, $X_{DCT}(4)$, and $X_{DCT}(6)$.

10.25 Given the following input data:

$$x(0) = 25,\ x(1) = 30,\ x(2) = 28,\ x(3) = 25,$$
$$x(4) = 10,\ x(5) = 0,\ x(6) = 0,\ \text{and}\ x(7) = 0,$$

determine the DCT coefficient $X_{DCT}(1)$, $X_{DCT}(3)$, $X_{DCT}(5)$, and $X_{DCT}(7)$.

10.26 Assuming the following DCT coefficients with infinite precision:

$$X_{DCT}(0) = 14,\ X_{DCT}(1) = 6,\ X_{DCT}(2) = -6,\ \text{and}\ X_{DCT}(3) = 8,$$

(a) determine the input data using the MATLAB function **idct();**

(b) recover the input data samples using the MATLAB function **idct()** if a bit allocation scheme quantizes the DCT coefficients as follows: 2 magnitude bits plus 1 sign bit (3 bits) for the DC coefficient, 1 magnitude bit plus 1 sign bit (2 bits) for each AC coefficient and a scale factor of 8, that is,

$$X_{DCT}(0) = 8 \times 2 = 16,\ X_{DCT}(1) = 8 \times 1 = 8,\ X_{DCT}(2) = 8 \times (-1) = -8,\ \text{and}\ X_{DCT}(3) = 8 \times 1 = 8;$$

(c) compute the quantized error in part b in this problem.

10.27 Assuming the following DCT coefficients with infinite precision:

$$X_{DCT}(0) = 11, X_{DCT}(1) = 5, X_{DCT}(2) = 7, \text{and}\, X_{DCT}(3) = -3,$$

(a) determine the input data using the MATLAB function **idct().**

(b) recover the input data samples using the MATLAB function **idct()** if a bit allocation scheme quantizes the DCT coefficients as follows: 2 magnitude bits plus 1 sign bit (3 bits) for the DC coefficient, 1 magnitude bit plus 1 sign bit (2 bits) for each AC coefficient and a scale factor of 8, that is,

$$X_{DCT}(0) = 8 \times 1 = 8, X_{DCT}(1) = 8 \times 1 = 8, X_{DCT}(2) = 8 \times 1 = -8, \text{and } X_{DCT}(3) = 8 \times 0 = 0;$$

(c) compute the quantized error in part b in this problem.

10.28 **(a)** Verify the window function

$$f(n) = h(n) = \sin\left(\frac{\pi}{N}(n+0.5)\right)$$

used in MDCT is satisfied with Eqs. (10.35) and (10.36).

(b) Verify W-MDCT coefficients

$$X_{MDCT}(k) = (-1)^{\frac{N}{2}+1} X_{MDCT}(N-1-k) \text{ for } k = N/2, N/2+1, \cdots, N-1.$$

10.29 Given a data sequence: 1, 2, 3, 4, 5, 4, 3, 2, …,

(a) Determine the W-MDCT coefficients for the first three blocks using a block size of 4;

(b) Determine the first two overlapped subblocks and compare the results with the original data sequence using the W-MDCT coefficients in part (a).

10.30 Given a data sequence: 1, 2, 3, 4, 5, 4, 3, 2, 1, 2, 3, 4, 5, …,

(a) Determine the W-MDCT coefficients for the first three blocks using a block size of 6;

(b) Determine the first two overlapped subblocks and compare the results with the original data sequence using the W-MDCT coefficients in part (a).

Computer Problems with MATLAB

Use the MATLAB programs in the program section for Problems 10.31–10.33.

10.31 Given the data file "speech.dat" with 16 bits per sample and a sampling rate of 8 kHz,

(a) Use the PCM coding (midtread quantizer) to perform compression and decompression and apply the MATLAB function **sound()** to evaluate the sound quality in terms of "excellent," "good," "intelligent," and "unacceptable" for the following bit rates

1. 4 bits/sample (32 kbits per second)
2. 6 bits/sample (48 kbits per second)
3. 8 bits/sample (64 kbits per second)

(b) Use the μ-law PCM coding to perform compression and decompression and apply the MATLAB function **sound()** to evaluate the sound quality.

1. 4 bits/sample (32 kbits per second)
2. 6 bits/sample (48 kbits per second)
3. 8 bits/sample (64 kbits per second)

10.32 Given the data file "speech.dat" with 16 bits per sample, a sampling rate of 8 kHz, and ADCPM coding, perform compression, and decompression and apply the MATLAB function **sound()** to evaluate the sound quality.

10.33 Given the data file "speech.dat" with 16 bits per sample, a sampling rate of 8 kHz, and DCT and M-DCT coding in the program section, perform compression, and decompression using the following specified parameters in Program 10.14 to compare the sound quality.

(a) nbits = 3, scalef = scalef2bits
(b) nbits = 3, scalef = scalef3bits
(c) nbits = 4, scalef = scalef2bits
(d) nbits = 4, scalef = scalef3bits

Advanced Problems

10.34 Given the companding function $y = \text{sign}(x)\dfrac{\ln\left(1+\mu\dfrac{|x|}{|x|_{\max}}\right)}{\ln(1+\mu)}$, show that

(a) $y = x/|x|_{\max}$, when $\mu \to 0$,

(b) $y = \text{sign}(x)$, when $\mu \to \infty$.

10.35 Given the companding function as defined in Eq. (10.4), prove Eq. (10.6).

10.36 For ADPCM encoding, the predictor has the following IIR transfer function:

$$\frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3} + b_4 z^{-4} + b_5 z^{-5}}{1 - a_1 z^{-1} - a_2 z^{-2}}.$$

Show that the stability of the predictor is guaranteed if the following constraint is satisfied:

$$|a_2| < 0.75 \text{ and } |a_1| < 1 - a_2.$$

That is, the poles of the predictor are inside the unit circle in the z-plane.

10.37 Given the DCT transform as defined in Eq. (10.29), prove the inverse DCT transform, that is, Eq. (10.30).

10.38 Given an N-point sequence of $x(n)$, define a $2N$-point even symmetric sequence of $s(n)$ as

$$s(n) = \begin{cases} x(n) & 0 \le n \le N-1 \\ x(2N-n-1) & N < n \le 2N-1 \end{cases}.$$

Show that DFT coefficients of $s(n)$ are given by

$$S(k) = W_{2N}^{-k/2} 2 \sum_{n=0}^{N-1} x(n) \cos\left[\frac{\pi(2n+1)k}{2N}\right] = W_{2N}^{-k/2} V(k) \text{ for } 0 \le k \le 2N-1,$$

where $W_{2N} = e^{-j2\pi/2N}$.

10.39 In Problem 10.38, show that

(a) $S(N) = 0$.

(b) $s(n) = \frac{1}{N} Re\left\{ \frac{S(0)}{2} + \sum_{k=1}^{N-1} S(k) W_{2N}^{-kn} \right\}$ for $0 \leq k \leq 2N-1$.

10.40 Using results from Problems 10.38 and 10.39, show that

$$x(n) = \frac{1}{N} \left\{ \frac{V(0)}{2} + \sum_{k=1}^{N-1} V(k) \cos\left[\frac{\pi(2n+1)k}{2N} \right] \right\} \text{for} \, 0 \leq k \leq N-1.$$