

## 11



# Compression

The increasingly rapid movement of information around the world relies on ingenious methods of data representation, which are in turn made possible by orthogonal transformations. The JPEG format for image representation is based on the Discrete Cosine Transform developed in this chapter. The MPEG-1 and MPEG-2 formats for TV and video data and the H.263 format for video phones are also based on the DCT, but with extra emphasis on compressing in the time dimension.

Sound files can be compressed into a variety of different formats, including MP3, Advanced Audio

Coding (used by Apple's iTunes and XM satellite radio), Microsoft's Windows Media Audio (WMA), and other state-of-the-art methods. What these formats have in common is that the core compression is done by a variant of the DCT called the Modified Discrete Cosine Transform.

**Reality Check**

Reality Check 11 on page 527 explores implementation of the MDCT into a simple, working algorithm to compress audio.

In Chapters 4 and 10, we observed the usefulness of orthogonality to represent and compress data. Here, we introduce the Discrete Cosine Transform (DCT), a variant of the Fourier transform that can be computed in real arithmetic. It is currently the method of choice for compression of sound and image files.

The simplicity of the Fourier transform stems from orthogonality, due to its representation as a complex unitary matrix. The Discrete Cosine Transform has a representation as a real orthogonal matrix, and so the same orthogonality properties make it simple to apply and easy to invert. Its similarity to the Discrete Fourier Transform (DFT) is close enough that fast versions of the DCT exist, in analogy to the Fast Fourier Transform (FFT).

In this chapter, the basic properties of the DCT are explained, and the links to working compression formats are investigated. The well-known JPEG format, for example, applies the two-dimensional DCT to  $8 \times 8$  pixel blocks of an image, and stores the results using Huffman coding. The details of JPEG compression are investigated as a case study in Sections 11.2–11.3.

A modified version of the Discrete Cosine Transform, called the Modified Discrete Cosine Transform (MDCT), is the basis of most modern audio compression formats. The MDCT is the current gold standard for compression of sound files. We will introduce MDCT and investigate its application for coding and decoding, which provides the core technology of file formats such as MP3 and AAC (Advanced Audio Coding).

## 11.1 THE DISCRETE COSINE TRANSFORM

In this section, we introduce the Discrete Cosine Transform. This transform interpolates data, using basis functions that are all cosine functions, and involves only real computations. Its orthogonality characteristics make least squares approximations simple, as in the case of the Discrete Fourier Transform.

### 11.1.1 One-dimensional DCT

Let  $n$  be a positive integer. The one-dimensional Discrete Cosine Transform of order  $n$  is defined by the  $n \times n$  matrix  $C$  whose entries are

$$C_{ij} = \frac{\sqrt{2}}{\sqrt{n}} a_i \cos \frac{i(2j+1)\pi}{2n} \quad (11.1)$$

for  $i, j = 0, \dots, n-1$ , where

$$a_i \equiv \begin{cases} 1/\sqrt{2} & \text{if } i = 0, \\ 1 & \text{if } i = 1, \dots, n-1 \end{cases}$$

or

$$C = \sqrt{\frac{2}{n}} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \cdots & \frac{1}{\sqrt{2}} \\ \cos \frac{\pi}{2n} & \cos \frac{3\pi}{2n} & \cdots & \cos \frac{(2n-1)\pi}{2n} \\ \cos \frac{2\pi}{2n} & \cos \frac{6\pi}{2n} & \cdots & \cos \frac{2(2n-1)\pi}{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \cos \frac{(n-1)\pi}{2n} & \cos \frac{(n-1)3\pi}{2n} & \cdots & \cos \frac{(n-1)(2n-1)\pi}{2n} \end{bmatrix}. \quad (11.2)$$

With two-dimensional images, the convention is to begin with 0 instead of 1. The notation will be easier if we extend this convention to matrix numbering, as we have done in (11.1). *In this chapter, subscripts for  $n \times n$  matrices will go from 0 to  $n-1$ .* For simplicity, we will treat only the case where  $n$  is even in the following discussion.

**DEFINITION 11.1** Let  $C$  be the matrix defined in (11.2). The **Discrete Cosine Transform (DCT)** of  $x = [x_0, \dots, x_{n-1}]^T$  is the  $n$ -dimensional vector  $y = [y_0, \dots, y_{n-1}]^T$ , where

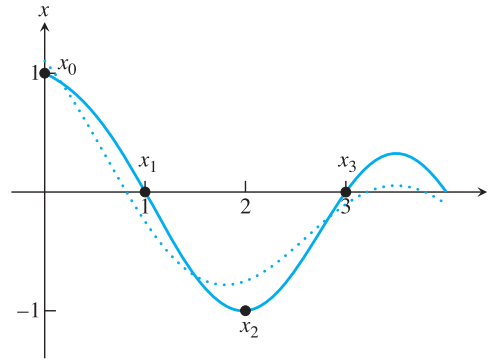
$$y = Cx. \quad (11.3)$$

□

Note that  $C$  is a real orthogonal matrix, meaning that its transpose is its inverse:

$$C^{-1} = C^T = \sqrt{\frac{2}{n}} \begin{bmatrix} \frac{1}{\sqrt{2}} & \cos \frac{\pi}{2n} & \cdots & \cos \frac{(n-1)\pi}{2n} \\ \frac{1}{\sqrt{2}} & \cos \frac{3\pi}{2n} & \cdots & \cos \frac{(n-1)3\pi}{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1}{\sqrt{2}} & \cos \frac{(2n-1)\pi}{2n} & \cdots & \cos \frac{(n-1)(2n-1)\pi}{2n} \end{bmatrix}. \quad (11.4)$$





**Figure 11.1 DCT interpolation and least squares approximation.** The data points are  $(j, x_j)$ , where  $x = [1, 0, -1, 0]$ . The DCT interpolating function  $P_4(t)$  of (11.8) is shown as a solid curve, along with the least squares DCT approximation function  $P_3(t)$  of (11.9) as a dotted curve.

$$\begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ -1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ c + b \\ 2a \\ c - b \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{\sqrt{2-\sqrt{2}}+\sqrt{2+\sqrt{2}}}{2\sqrt{2}} \\ 1 \\ \frac{\sqrt{2-\sqrt{2}}-\sqrt{2+\sqrt{2}}}{2\sqrt{2}} \end{bmatrix} \approx \begin{bmatrix} 0.0000 \\ 0.9239 \\ 1.0000 \\ -0.3827 \end{bmatrix}.$$

According to Theorem 11.2 with  $n = 4$ , the function

$$P_4(t) = \frac{1}{\sqrt{2}} \left[ 0.9239 \cos \frac{(2t+1)\pi}{8} + \cos \frac{2(2t+1)\pi}{8} - 0.3827 \cos \frac{3(2t+1)\pi}{8} \right] \quad (11.8)$$

interpolates the four data points. The function  $P_4(t)$  is plotted as the solid curve in Figure 11.1. ▶

### 11.1.2 The DCT and least squares approximation

Just as the DCT Interpolation Theorem 11.2 is an immediate consequence of Theorem 10.9, the least squares result Theorem 10.11 shows how to find a DCT least squares approximation of the data, using only part of the basis functions. Because of the orthogonality of the basis functions, this can be accomplished by simply dropping the higher frequency terms.

#### SPOTLIGHT ON

##### Orthogonality

The idea behind least squares approximation is that finding the shortest distance from a point to a plane (or subspace in general) means constructing the perpendicular from the point to the plane. This construction is carried out by the normal equations, as we saw in Chapter 4. In Chapters 10 and 11, this concept is applied to approximate data as closely as possible with a relatively small set of basis functions, resulting in compression. The basic message is to choose the basis functions to be orthogonal, as reflected in the rows of the DCT matrix. Then the normal equations become computationally very simple (see Theorem 10.11).

**THEOREM 11.3 DCT Least Squares Approximation Theorem.** Let  $x = [x_0, \dots, x_{n-1}]^T$  be a vector of  $n$  real numbers. Define  $y = [y_0, \dots, y_{n-1}]^T = Cx$ , where  $C$  is the Discrete Cosine Transform matrix. Then, for any positive integer  $m \leq n$ , the choice of coefficients  $y_0, \dots, y_{m-1}$  in

$$P_m(t) = \frac{1}{\sqrt{n}}y_0 + \frac{\sqrt{2}}{\sqrt{n}} \sum_{k=1}^{m-1} y_k \cos \frac{k(2t+1)\pi}{2n}$$

minimizes the squared approximation error  $\sum_{j=0}^{n-1} (P_m(j) - x_j)^2$  of the  $n$  data points. ■

**Proof.** Follows directly from Theorem 10.11. □

Referring to Example 11.1, if we require the best least squares approximation to the same four data points, but use the three basis functions

$$1, \cos \frac{(2t+1)\pi}{8}, \cos \frac{2(2t+1)\pi}{8}$$

only, the solution is

$$P_3(t) = \frac{1}{2} \cdot 0 + \frac{1}{\sqrt{2}} \left[ 0.9239 \cos \frac{(2t+1)\pi}{8} + \cos \frac{2(2t+1)\pi}{8} \right]. \quad (11.9)$$

Figure 11.1 compares the least squares solution  $P_3$  with the interpolating function  $P_4$ .

► **EXAMPLE 11.2** Use the DCT and Theorem 11.3 to find least squares fits to the data  $t = 0, \dots, 7$  and  $x = [-2.2, -2.8, -6.1, -3.9, 0.0, 1.1, -0.6, -1.1]^T$  for  $m = 4, 6$ , and 8.

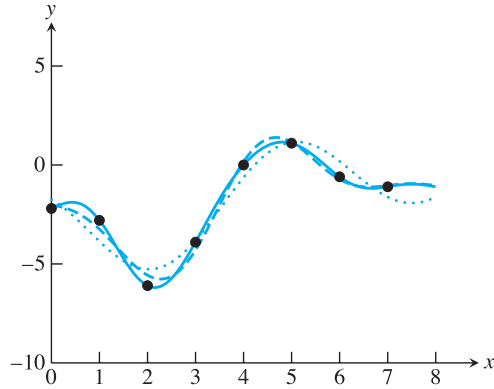
Setting  $n = 8$ , we find that the DCT of the data is

$$y = Cx = \begin{bmatrix} -5.5154 \\ -3.8345 \\ 0.5833 \\ 4.3715 \\ 0.4243 \\ -1.5504 \\ -0.6243 \\ -0.5769 \end{bmatrix}.$$

According to Theorem 11.2, the discrete cosine interpolant of the eight data points is

$$\begin{aligned} P_8(t) = & \frac{1}{\sqrt{8}}(-5.5154) + \frac{1}{2} \left[ -3.8345 \cos \frac{(2t+1)\pi}{16} + 0.5833 \cos \frac{2(2t+1)\pi}{16} \right. \\ & + 4.3715 \cos \frac{3(2t+1)\pi}{16} + 0.4243 \cos \frac{4(2t+1)\pi}{16} \\ & - 1.5504 \cos \frac{5(2t+1)\pi}{16} - 0.6243 \cos \frac{6(2t+1)\pi}{16} \\ & \left. - 0.5769 \cos \frac{7(2t+1)\pi}{16} \right]. \end{aligned}$$

The interpolant  $P_8$  is plotted in Figure 11.2, along with the least squares fits  $P_6$  and  $P_4$ . The latter are obtained, according to Theorem 11.3, by keeping the first six, or first four terms, respectively, of  $P_8$ . ◀



**Figure 11.2 DCT interpolation and least squares approximation.** The solid curve is the DCT interpolant of the data points in Example 11.2. The dashed curve is the least squares fit from the first six terms only, and the dotted curve represents four terms.

## 11.1 Exercises

- Use the  $2 \times 2$  DCT matrix and Theorem 11.2 to find the DCT interpolating function for the data points.

$$(a) \begin{array}{c|c} t & x \\ \hline 0 & 3 \\ 1 & 3 \end{array} \quad (b) \begin{array}{c|c} t & x \\ \hline 0 & 2 \\ 1 & -2 \end{array} \quad (c) \begin{array}{c|c} t & x \\ \hline 0 & 3 \\ 1 & 1 \end{array} \quad (d) \begin{array}{c|c} t & x \\ \hline 0 & 4 \\ 1 & -1 \end{array}$$

- Describe the  $m = 1$  least squares DCT approximation in terms of the input data  $(0, x_0), (1, x_1)$ .
- Find the DCT of the following data vectors  $x$ , and find the corresponding interpolating function  $P_n(t)$  for the data points  $(i, x_i), i = 0, \dots, n - 1$  (you may state your answers in terms of the  $b$  and  $c$  defined in (11.7)):

$$(a) \begin{array}{c|c} t & x \\ \hline 0 & 1 \\ 1 & 0 \\ 2 & 1 \\ 3 & 0 \end{array} \quad (b) \begin{array}{c|c} t & x \\ \hline 0 & 1 \\ 1 & 1 \\ 2 & 1 \\ 3 & 1 \end{array} \quad (c) \begin{array}{c|c} t & x \\ \hline 0 & 1 \\ 1 & 0 \\ 2 & 0 \\ 3 & 0 \end{array} \quad (d) \begin{array}{c|c} t & x \\ \hline 0 & 1 \\ 1 & 2 \\ 2 & 3 \\ 3 & 4 \end{array}$$

- Find the DCT least squares approximation with  $m = 2$  terms for the data in Exercise 3.
- Carry out the trigonometry needed to establish equations (11.6) and (11.7).
- (a) Prove the trigonometric formula  $\cos(x + y) + \cos(x - y) = 2 \cos x \cos y$  for any  $x, y$ .  
(b) Show that the columns of  $C^T$  are eigenvectors of the matrix  $T$  in (11.5), and identify the eigenvalues. (c) Show that the columns of  $C^T$  are unit vectors.
- Extend the DCT Interpolation Theorem 11.2 to the interval  $[c, d]$  as follows. Let  $n$  be a positive integer and set  $\Delta_t = (d - c)/n$ . Use the DCT to produce a polynomial  $P_n(t)$  that satisfies  $P_n(c + j\Delta_t) = x_j$  for  $j = 0, \dots, n - 1$ .

## 11.1 Computer Problems

- Plot the data from Exercise 3, along with the DCT interpolant and the DCT least squares approximation with  $m = 2$  terms.

2. Plot the data along with the  $m = 4, 6,$  and  $8$  DCT least squares approximations.

$t$	$x$	$t$	$x$	$t$	$x$	$t$	$x$
0	3	0	4	0	3	0	4
1	5	1	1	1	-1	1	2
2	-1	2	-3	2	-1	2	-4
(a) 3	3	(b) 3	0	(c) 3	3	(d) 3	2
4	1	4	0	4	3	4	4
5	3	5	2	5	-1	5	2
6	-2	6	-4	6	-1	6	-4
7	4	7	0	7	3	7	2

3. Plot the function  $f(t)$ , the data points  $(j, f(j))$ ,  $j = 0, \dots, 7$ , and the DCT interpolation function. (a)  $f(t) = e^{-t/4}$  (b)  $f(t) = \cos \frac{\pi}{2}t$ .

## 11.2 TWO-DIMENSIONAL DCT AND IMAGE COMPRESSION

The two-dimensional Discrete Cosine Transform is often used to compress small blocks of an image, as small as  $8 \times 8$  pixels. The compression is lossy, meaning that some information from the block is ignored. The key feature of the DCT is that it helps organize the information so that the part that is ignored is the part that the human eye is least sensitive to. More precisely, the DCT will show us how to interpolate the data with a set of basis functions that are in descending order of importance as far as the human visual system is concerned. The less important interpolation terms can be dropped if desired, just as a newspaper editor cuts a long story on deadline.

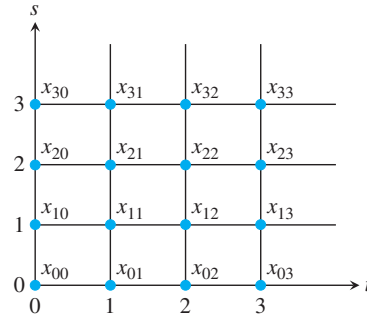
Later, we will apply what we have learned about the DCT to compress images. Using the added tools of quantization and Huffman coding, each  $8 \times 8$  block of an image can be reduced to a bit stream that is stored with bit streams from the other blocks of the image. The complete bit stream is decoded, when the image needs to be uncompressed and displayed, by reversing the encoding process. We will describe this approach, called Baseline JPEG, the default method for storing JPEG images.

### 11.2.1 Two-dimensional DCT

The two-dimensional Discrete Cosine Transform is simply the one-dimensional DCT applied in two dimensions, one after the other. It can be used to interpolate or approximate data given on a two-dimensional grid, in a straightforward analogy to the one-dimensional case. In the context of image processing, the two-dimensional grid represents a block of pixel values—say, grayscale intensities or color intensities.

*In this chapter only, we will list the vertical coordinate first and the horizontal coordinate second when referring to a two-dimensional point, as shown in Figure 11.3.* The goal is to be consistent with the usual matrix convention, where the  $i$  index of entry  $x_{ij}$  changes along the vertical direction, and  $j$  along the horizontal. A major application of this section is to pixel files representing images, which are most naturally viewed as matrices of numbers.

Figure 11.3 shows a grid of  $(s, t)$  points in the two-dimensional plane with assigned values  $x_{ij}$  at each rectangular grid point  $(s_i, t_j)$ . For concreteness, we will use the integer grid  $s_i = \{0, 1, \dots, n - 1\}$  (remember, along the vertical axis) and  $t_j = \{0, 1, \dots, n - 1\}$  along the horizontal axis. The purpose of the two-dimensional DCT is to construct an interpolating function  $F(s, t)$  that fits the  $n^2$  points  $(s_i, t_j, x_{ij})$  for  $i, j = 0, \dots, n - 1$ . The 2D-DCT accomplishes this in an optimal way from the point of view of least squares, meaning that the fit degrades gracefully as basis functions are dropped from the interpolating function.



**Figure 11.3 Two-dimensional grid of data points.** The 2D-DCT can be used to interpolate function values on a square grid, such as pixel values of an image.

The 2D-DCT is the one-dimensional DCT applied successively to both horizontal and vertical directions. Consider the matrix  $X$  consisting of the values  $x_{ij}$ , as in Figure 11.3. To apply the 1D-DCT in the horizontal  $s$ -direction, we first need to transpose  $X$ , then multiply by  $C$ . The resulting columns are the 1D-DCT's of the rows of  $X$ . Each column of  $CX^T$  corresponds to a fixed  $t_i$ . To do a 1D-DCT in the  $t$ -direction means moving across the rows; so, again, transposing and multiplying by  $C$  yields

$$C(CX^T)^T = CXC^T. \tag{11.10}$$

**DEFINITION 11.4** The **two-dimensional Discrete Cosine Transform** (2D-DCT) of the  $n \times n$  matrix  $X$  is the matrix  $Y = CXC^T$ , where  $C$  is defined in (11.1). □

► **EXAMPLE 11.3** Find the 2D Discrete Cosine Transform of the data in Figure 11.4(a).

From the definition and (11.6), the 2D-DCT is the matrix

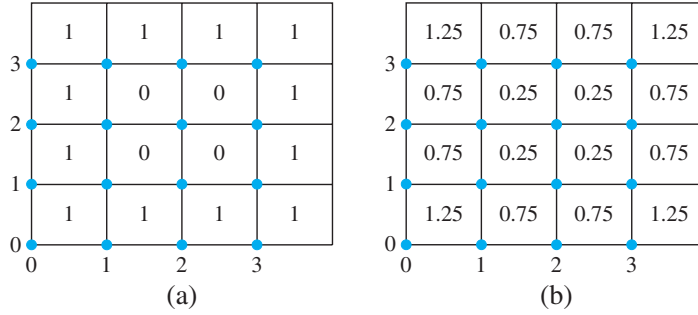
$$\begin{aligned} Y = CXC^T &= \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} a & b & a & c \\ a & c & -a & -b \\ a & -c & -a & b \\ a & -b & a & -c \end{bmatrix} \\ &= \begin{bmatrix} 3 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}. \end{aligned} \tag{11.11}$$

The inverse of the 2D-DCT is easy to express in terms of the DCT matrix  $C$ . Since  $Y = CXC^T$  and  $C$  is orthogonal, the  $X$  is recovered as  $X = C^T Y C$ .

**DEFINITION 11.5** The **inverse two-dimensional Discrete Cosine Transform** of the  $n \times n$  matrix  $Y$  is the matrix  $X = C^T Y C$ . □

As we have seen, there is a close connection between inverting an orthogonal transform (like the 2D-DCT) and interpolation. The goal of interpolation is to recover the original data points from functions that are constructed with the interpolating coefficients that came out of the transform. Since  $C$  is an orthogonal matrix,  $C^{-1} = C^T$ . The inversion of the 2D-DCT can be written as a fact about interpolation,  $X = C^T Y C$ , since in this equation the  $x_{ij}$  are being expressed in terms of products of cosines.





**Figure 11.4** Two-dimensional data for Example 11.3. (a) The 16 data points  $(i, j, x_{ij})$ . (b) Values of the least squares approximation (11.14) at the grid points.

To write a useful expression for the interpolating function, recall the definition of  $C$  in (11.1),

$$C_{ij} = \frac{\sqrt{2}}{\sqrt{n}} a_i \cos \frac{i(2j+1)\pi}{2n} \quad (11.12)$$

for  $i, j = 0, \dots, n-1$ , where

$$a_i \equiv \begin{cases} 1/\sqrt{2} & \text{if } i = 0, \\ 1 & \text{if } i = 1, \dots, n-1. \end{cases}$$

According to the rules of matrix multiplication, the equation  $X = C^T Y C$  translates to

$$\begin{aligned} x_{ij} &= \sum_{k=0}^{n-1} \sum_{l=0}^{n-1} C_{ik}^T y_{kl} C_{lj} \\ &= \sum_{k=0}^{n-1} \sum_{l=0}^{n-1} C_{ki} y_{kl} C_{lj} \\ &= \frac{2}{n} \sum_{k=0}^{n-1} \sum_{l=0}^{n-1} y_{kl} a_k a_l \cos \frac{k(2i+1)\pi}{2n} \cos \frac{l(2j+1)\pi}{2n}. \end{aligned} \quad (11.13)$$

This is exactly the interpolation statement we were looking for.

**THEOREM 11.6 2D-DCT Interpolation Theorem.** Let  $X = (x_{ij})$  be a matrix of  $n^2$  real numbers. Let  $Y = (y_{kl})$  be the two-dimensional Discrete Cosine Transform of  $X$ . Define  $a_0 = 1/\sqrt{2}$  and  $a_k = 1$  for  $k > 0$ . Then the real function

$$P_n(s, t) = \frac{2}{n} \sum_{k=0}^{n-1} \sum_{l=0}^{n-1} y_{kl} a_k a_l \cos \frac{k(2s+1)\pi}{2n} \cos \frac{l(2t+1)\pi}{2n}$$

satisfies  $P_n(i, j) = x_{ij}$  for  $i, j = 0, \dots, n-1$ . ■

Returning to Example 11.3, the only nonzero interpolation coefficients are  $y_{00} = 3$ ,  $y_{02} = y_{20} = 1$ , and  $y_{22} = -1$ . Writing out the interpolation function in the Theorem 11.6 yields

$$\begin{aligned}
P_4(s, t) &= \frac{2}{4} \left[ \frac{1}{2} y_{00} + \frac{1}{\sqrt{2}} y_{02} \cos \frac{2(2t+1)\pi}{8} + \frac{1}{\sqrt{2}} y_{20} \cos \frac{2(2s+1)\pi}{8} \right. \\
&\quad \left. + y_{22} \cos \frac{2(2s+1)\pi}{8} \cos \frac{2(2t+1)\pi}{8} \right] \\
&= \frac{1}{2} \left[ \frac{1}{2} (3) + \frac{1}{\sqrt{2}} (1) \cos \frac{2(2t+1)\pi}{8} + \frac{1}{\sqrt{2}} (1) \cos \frac{2(2s+1)\pi}{8} \right. \\
&\quad \left. + (-1) \cos \frac{2(2s+1)\pi}{8} \cos \frac{2(2t+1)\pi}{8} \right] \\
&= \frac{3}{4} + \frac{1}{2\sqrt{2}} \cos \frac{(2t+1)\pi}{4} + \frac{1}{2\sqrt{2}} \cos \frac{(2s+1)\pi}{4} \\
&\quad - \frac{1}{2} \cos \frac{(2s+1)\pi}{4} \cos \frac{(2t+1)\pi}{4}.
\end{aligned}$$

Checking the interpolation, we get, for example,

$$P_4(0, 0) = \frac{3}{4} + \frac{1}{4} + \frac{1}{4} - \frac{1}{4} = 1$$

and

$$P_4(1, 2) = \frac{3}{4} - \frac{1}{4} - \frac{1}{4} - \frac{1}{4} = 0,$$

agreeing with the data in Figure 11.4. The constant term  $y_{00}/n$  of the interpolation function is called the “DC” component of the expansion (for “direct current”). It is the simple average of the data; the nonconstant terms contain the fluctuations of the data about this average value. In this example, the average of the 12 ones and 4 zeros is  $y_{00}/4 = 3/4$ .

Least squares approximations with the 2D-DCT are done in the same way as with the 1D-DCT. For example, implementing a low-pass filter would mean simply deleting the “high-frequency” components, those whose coefficients have larger indices, from the interpolating function. In Example 11.3, the best least squares fit to the basis functions

$$\cos \frac{i(2s+1)\pi}{8} \cos \frac{j(2t+1)\pi}{8}$$

for  $i + j \leq 2$  is given by dropping all terms that do not satisfy  $i + j \leq 2$ . In this case, the only nonzero “high-frequency” term is the  $i = j = 2$  term, leaving

$$P_2(s, t) = \frac{3}{4} + \frac{1}{2\sqrt{2}} \cos \frac{(2t+1)\pi}{4} + \frac{1}{2\sqrt{2}} \cos \frac{(2s+1)\pi}{4}. \quad (11.14)$$

This least squares approximation is shown in Figure 11.4(b).

Defining the DCT matrix  $C$  in MATLAB can be done through the code fragment

```

for i=1:n
    for j=1:n
        C(i,j)=cos((i-1)*(2*j-1)*pi/(2*n));
    end
end
C=sqrt(2/n)*C;
C(1,:)=C(1,:)/sqrt(2);

```

Alternatively, if MATLAB’s Signal Processing Toolbox is available, the one-dimensional DCT of a vector  $x$  can be computed as

```
>> y=dct(x);
```

To carry out the 2D-DCT of a matrix  $X$ , we fall back on equation (11.10), or

```
>> Y=C*X*C'
```

If MATLAB's `dct` is available, the command

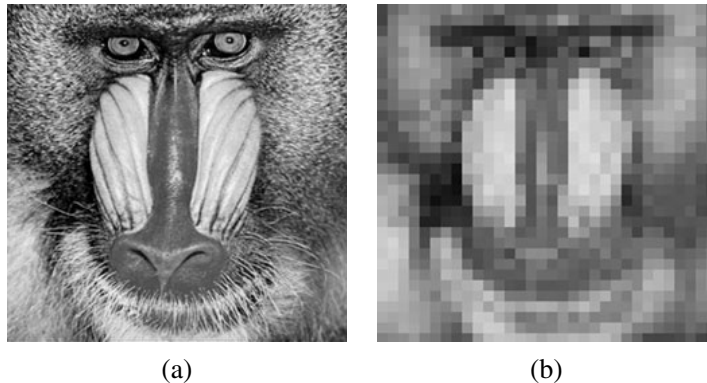
```
>> Y=dct(dct(X'))'
```

computes the 2D-DCT with two applications of the 1D-DCT.

### 11.2.2 Image compression

The concept of orthogonality, as represented in the Discrete Cosine Transform, is crucial to performing image compression. Images consist of pixels, each represented by a number (or three numbers, for color images). The convenient way that methods like the DCT can carry out least squares approximation makes it easy to reduce the number of bits needed to represent the pixel values, while degrading the picture only slightly, and perhaps imperceptibly to human viewers.

Figure 11.5(a) shows a grayscale rendering of a  $256 \times 256$  array of pixels. The grayness of each pixel is represented by one byte, a string of 8 bits representing  $0 = 00000000$  (black) to  $255 = 11111111$  (white). We can think of the information shown in the figure as a  $256 \times 256$  array of integers. Represented in this way, the picture holds  $(256)^2 = 2^{16} = 64\text{K}$  bytes of information.



**Figure 11.5 Grayscale image.** (a) Each pixel in the  $256 \times 256$  grid is represented by an integer between 0 and 255. (b) Crude compression—each  $8 \times 8$  square of pixels is colored by its average grayscale value.

MATLAB imports grayscale or RGB (Red-Green-Blue) values of images from standard image formats. For example, given a grayscale image file `picture.jpg`, the command

```
>> x = imread('picture.jpg');
```

puts the matrix of grayscale values into the double precision variable `x`. If the JPEG file is a color image, the array variable will have a third dimension to index the three colors. We will restrict attention to gray scale to begin our discussion; extension to color is straightforward.

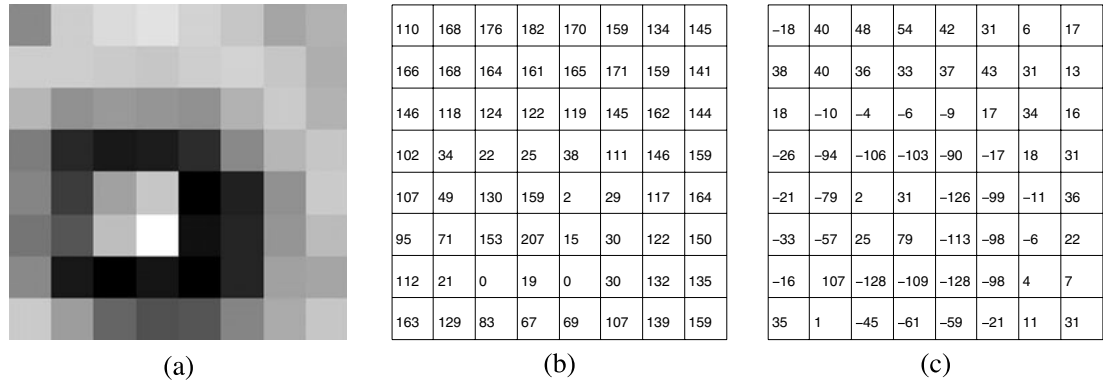
An  $m \times n$  matrix of grayscale values can be rendered by MATLAB with the commands

```
>> imagesc(x); colormap(gray)
```

while an  $m \times n \times 3$  matrix of RGB color is rendered with the `imagesc(x)` command alone. A common formula for converting a color RGB image to gray scale is

$$X_{\text{gray}} = 0.2126R + 0.7152G + 0.0722B, \quad (11.15)$$

or in MATLAB code,



**Figure 11.6** Example of  $8 \times 8$  block. (a) Grayscale view (b) Grayscale pixel values (c) Grayscale pixel values minus 128.

```
>> x=double(x);
>> r=x(:,:,1);g=x(:,:,2);b=x(:,:,3);
>> xgray=0.2126*r+0.7152*g+0.0722*b;
>> xgray=uint8(xgray);
>> imagesc(xgray);colormap(gray)
```

Note that we have converted the default MATLAB data type `uint8`, or unsigned integers, to double precision reals before we do the computation. It is best to convert back to `uint8` type before rendering the picture with `imagesc`.

Figure 11.5(b) shows a crude method of compression, where each  $8 \times 8$  pixel block is replaced by its average pixel value. The amount of data compression is considerable—there are only  $(32)^2 = 2^{10}$  blocks, each now represented by a single integer—but the resulting image quality is poor. Our goal is to compress less harshly, by replacing each  $8 \times 8$  block with a few integers that better carry the information of the original image.

To begin, we simplify the problem to a single  $8 \times 8$  block of pixels, as shown in Figure 11.6(a). The block was taken from the center of the subject's left eye in Figure 11.5. Figure 11.6(b) shows the one-byte integers that represent the grayscale intensities of the 64 pixels. In Figure 11.6(c), we have subtracted  $256/2 = 128$  from the pixel numbers to make them approximately centered around zero. This step is not essential, but better use of the 2D-DCT will result because of this centering.

To compress the  $8 \times 8$  pixel block shown, we will transform the matrix of grayscale pixel values

$$X = \begin{bmatrix} -18 & 40 & 48 & 54 & 42 & 31 & 6 & 17 \\ 38 & 40 & 36 & 33 & 37 & 43 & 31 & 13 \\ 18 & -10 & -4 & -6 & -9 & 17 & 34 & 16 \\ -26 & -94 & -106 & -103 & -90 & -17 & 18 & 31 \\ -21 & -79 & 2 & 31 & -126 & -99 & -11 & 36 \\ -33 & -57 & 25 & 79 & -113 & -98 & -6 & 22 \\ -16 & -107 & -128 & -109 & -128 & -98 & 4 & 7 \\ 35 & 1 & -45 & -61 & -59 & -21 & 11 & 31 \end{bmatrix} \quad (11.16)$$

and rely on the 2D-DCT's ability to sort information according to its importance to the human visual system. We calculate the 2D-DCT of  $X$  to be

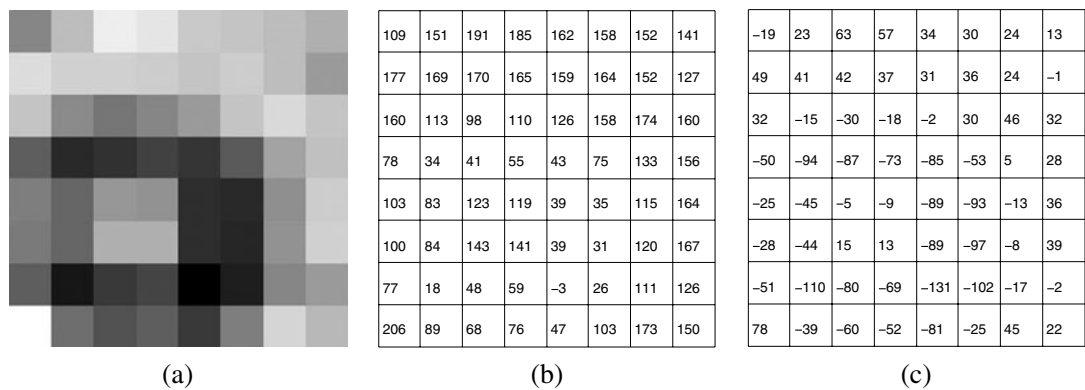
$$Y = C_8 X C_8^T = \begin{bmatrix} -121 & -66 & 127 & -65 & 27 & 98 & 7 & -25 \\ 200 & 22 & -124 & 34 & -36 & -62 & 5 & 6 \\ 113 & 43 & -32 & 55 & -25 & -75 & -21 & 12 \\ -10 & 35 & -69 & -131 & 28 & 54 & -4 & -24 \\ -14 & -18 & 16 & 1 & -5 & -27 & 14 & -6 \\ -124 & -74 & 47 & 60 & -1 & -16 & -8 & 13 \\ 81 & 35 & -57 & -54 & -7 & 6 & 1 & -16 \\ -16 & 11 & 5 & -15 & 11 & 12 & -1 & 9 \end{bmatrix}, \quad (11.17)$$

after rounding to the nearest integer for simplicity. This rounding adds a small amount of extra error and is not strictly necessary, but again it will help the compression. Note that due to the larger amplitudes, there is a tendency for more of the information to be stored in the top left part of the transform matrix  $Y$ , compared with the lower right. The lower right represents higher frequency basis functions that are often less important to the visual system. Nevertheless, because the 2D-DCT is an invertible transform, the information in  $Y$  can be used to completely reconstruct the original image, up to the rounding.

The first compression strategy we try will be a form of low-pass filtering. As discussed in the last section, least squares approximation with the 2D-DCT is just a matter of dropping terms from the interpolation function  $P_8(s, t)$ . For example, we can cut off the contribution of functions with relatively high spatial frequency by setting all  $y_{kl} = 0$  for  $k + l \geq 7$  (recall that we continue to number matrix entries as  $0 \leq k, l \leq 7$ ). After low-pass filtering, the transform coefficients are

$$Y_{\text{low}} = \begin{bmatrix} -121 & -66 & 127 & -65 & 27 & 98 & 7 & 0 \\ 200 & 22 & -124 & 34 & -36 & -62 & 0 & 0 \\ 113 & 43 & -32 & 55 & -25 & 0 & 0 & 0 \\ -10 & 35 & -69 & -131 & 0 & 0 & 0 & 0 \\ -14 & -18 & 16 & 0 & 0 & 0 & 0 & 0 \\ -124 & -74 & 0 & 0 & 0 & 0 & 0 & 0 \\ 81 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \quad (11.18)$$

To reconstruct the image, we apply the inverse 2D-DCT as  $C_8^T Y_{\text{low}} C_8$  and get the grayscale pixel values shown in Figure 11.7. The image in part (a) is similar to the original in Figure 11.6(a), but different in detail.



**Figure 11.7 Result of low-pass filtering.** (a) Filtered image (b) Grayscale pixel values, after transforming and adding 128 (c) Inverse transformed data.

How much have we compressed the information from the  $8 \times 8$  block? The original picture can be reconstructed (losslessly, except for the integer rounding) by inverse transforming the 2D-DCT (11.17) and adding back the 128. In doing the low-pass filtering with matrix (11.17), we have cut the storage requirements approximately in half, while retaining most of the qualitative visual aspects of the block.

### 11.2.3 Quantization

The idea of quantization will allow the effects of low-pass filtering to be achieved in a more selective way. Instead of completely ignoring coefficients, we will retain low-accuracy versions of some coefficients at a lower storage cost. This idea exploits the same aspects of the human visual system—that it is less sensitive to higher spatial frequencies. The main idea is to assign fewer bits to store information about the lower right corner of the transform matrix  $Y$ , instead of throwing it away.

#### Quantization modulo $q$

$$\begin{aligned} \text{Quantization: } z &= \text{round}\left(\frac{y}{q}\right) \\ \text{Dequantization: } \bar{y} &= qz \end{aligned} \quad (11.19)$$

Here, “round” means “to the nearest integer.” The **quantization error** is the difference between the input  $y$  and the output  $\bar{y}$  after quantizing and dequantizing. The maximum error of quantization modulo  $q$  is  $q/2$ .

► **EXAMPLE 11.4** Quantize the numbers  $-10$ ,  $3$ , and  $65$  modulo  $8$ .

The quantized values are  $-1$ ,  $0$ , and  $8$ . Upon dequantizing, the results are  $-8$ ,  $0$ , and  $64$ . The errors are  $|-2|$ ,  $|3|$ , and  $|1|$ , respectively, each less than  $q/2 = 4$ . ◀

Returning to the image example, the number of bits allowed for each frequency can be chosen arbitrarily. Let  $Q$  be an  $8 \times 8$  matrix called the **quantization matrix**. The entries  $q_{kl}$ ,  $0 \leq k, l \leq 7$  will regulate how many bits we assign to each entry of the transform matrix  $Y$ . Replace  $Y$  by the compressed matrix

$$Y_Q = \left[ \text{round}\left(\frac{y_{kl}}{q_{kl}}\right) \right], 0 \leq k, l \leq 7. \quad (11.20)$$

The matrix  $Y$  is divided entrywise by the quantization matrix. The subsequent rounding is where the loss occurs, and makes this method a form of lossy compression. Note that the larger the entry of  $Q$ , the more is potentially lost to quantization.

As a first example, **linear quantization** is defined by the matrix

$$q_{kl} = 8p(k + l + 1) \text{ for } 0 \leq k, l \leq 7 \quad (11.21)$$

for some constant  $p$ , called the **loss parameter**. Thus,

$$Q = p \begin{bmatrix} 8 & 16 & 24 & 32 & 40 & 48 & 56 & 64 \\ 16 & 24 & 32 & 40 & 48 & 56 & 64 & 72 \\ 24 & 32 & 40 & 48 & 56 & 64 & 72 & 80 \\ 32 & 40 & 48 & 56 & 64 & 72 & 80 & 88 \\ 40 & 48 & 56 & 64 & 72 & 80 & 88 & 96 \\ 48 & 56 & 64 & 72 & 80 & 88 & 96 & 104 \\ 56 & 64 & 72 & 80 & 88 & 96 & 104 & 112 \\ 64 & 72 & 80 & 88 & 96 & 104 & 112 & 120 \end{bmatrix}.$$

In MATLAB, the linear quantization matrix can be defined by `Q=p*8./hilb(8);`

The loss parameter  $p$  is a knob that can be turned to trade bits for visual accuracy. The smaller the loss parameter, the better the reconstruction will be. The resulting set of numbers in the matrix  $Y_Q$  represents the new quantized version of the image.

To decompress the file, the  $Y_Q$  matrix is dequantized by reversing the process, which is entrywise multiplication by  $Q$ . This is the lossy part of image coding. Replacing the entries  $y_{kl}$  by dividing by  $q_{kl}$  and rounding, and then reconstructing by multiplying by  $q_{kl}$ , one has potentially added error of size  $q_{kl}/2$  to  $y_{kl}$ . This is the quantization error. The larger the  $q_{kl}$ , the larger the potential error in reconstructing the image. On the other hand, the larger the  $q_{kl}$ , the smaller the integer entries of  $Y_Q$ , and the fewer bits will be needed to store them. This is the trade-off between image accuracy and file size.

In fact, quantization accomplishes two things: Many small contributions from higher frequencies are immediately set to zero by (11.20), and the contributions that remain nonzero are reduced in size, so that they can be transmitted or stored by using fewer bits. The resulting set of numbers are converted to a bit stream with the use of Huffman coding, discussed in the next section.

Next, we demonstrate the complete series of steps for compression of a matrix of pixel values in MATLAB. The output of MATLAB's `imread` command is an  $m \times n$  matrix of 8-bit integers for a grayscale photo, or three such matrices for a color photo. (The three matrices carry information for red, green, and blue, respectively; we discuss color in more detail below.) An 8-bit integer is called a `uint8`, to distinguish it from a `double`, as studied in Chapter 0, which requires 64 bits of storage. The command `double(x)` converts the `uint8` number  $x$  into the `double` format, and the command `uint8(x)` does the reverse by rounding  $x$  to the nearest integer between 0 and 255.

The following four commands carry out the conversion, centering, transforming, and quantization of a square  $n \times n$  matrix  $X$  of `uint8` numbers, such as the  $8 \times 8$  pixel matrices considered above. Denote by  $C$  the  $n \times n$  DCT matrix.

```
>> Xd=double(X);
>> Xc=Xd-128;
>> Y=C*Xc*C';
>> Yq=round(Y./Q);
```

At this point the resulting  $Y_q$  is stored or transmitted. To recover the image requires undoing the four steps in reverse order:

```
>> Ydq=Yq.*Q;
>> Xdq=C'*Ydq*C;
>> Xe=Xdq+128;
>> Xf=uint8(Xe);
```

After dequantization, the inverse DCT transform is applied, the offset 128 is added back, and the `double` format is converted back to a matrix  $X_f$  of `uint8` integers.

When linear quantization is applied to (11.17) with  $p = 1$ , the resulting coefficients are

$$Y_Q = \begin{bmatrix} -15 & -4 & 5 & -2 & 1 & 2 & 0 & 0 \\ 13 & 1 & -4 & 1 & -1 & -1 & 0 & 0 \\ 5 & 1 & -1 & 1 & 0 & -1 & 0 & 0 \\ 0 & 1 & -1 & -2 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & -1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \quad (11.22)$$

The reconstructed image block, formed by dequantizing and inverse-transforming  $Y_Q$ , is shown in Figure 11.8(a). Small differences can be seen in comparison with the original block, but it is more faithful than the low-pass filtering reconstruction.

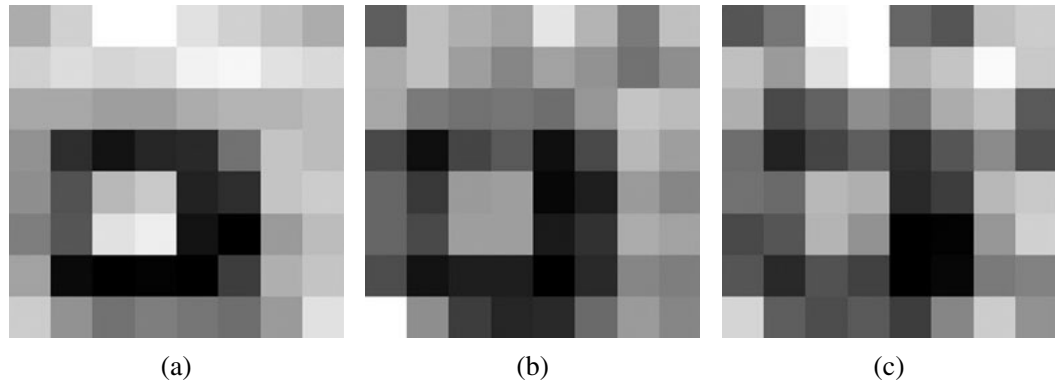


Figure 11.8 Result of linear quantization. Loss parameter is (a)  $p = 1$  (b)  $p = 2$  (c)  $p = 4$ .

After linear quantization with  $p = 2$ , the quantized transform coefficients are

$$Y_Q = \begin{bmatrix} -8 & -2 & 3 & -1 & 0 & 1 & 0 & 0 \\ 6 & 0 & -2 & 0 & 0 & -1 & 0 & 0 \\ 2 & 1 & 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad (11.23)$$

and after linear quantization with  $p = 4$ , the quantized transform coefficients are

$$Y_Q = \begin{bmatrix} -4 & -1 & 1 & -1 & 0 & 1 & 0 & 0 \\ 3 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \quad (11.24)$$

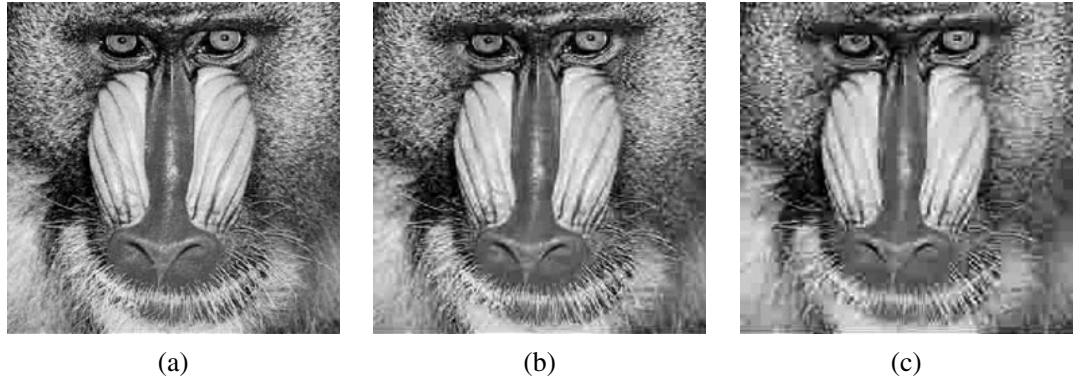
Figure 11.8 shows the result of linear quantization for the three different values of loss parameter  $p$ . Notice that the larger the value of the loss parameter  $p$ , the more entries of the matrix  $Y_Q$  are zeroed by the quantization procedure, the smaller are the data requirements for representing the pixels, and the less faithfully the original image has been reconstructed.

Next, we quantize all  $32 \times 32 = 1024$  blocks of the image in Figure 11.5. That is, we carry out 1024 independent versions of the previous example. The results for loss parameter  $p = 1, 2$ , and 4 are shown in Figure 11.9. The image has begun to deteriorate significantly by  $p = 4$ .

We can make a rough calculation to quantify the amount of image compression due to quantization. The original image uses a pixel value from 0 to 255, which is one byte, or 8 bits. For each  $8 \times 8$  block, the total number of bits needed without compression is  $8(8)^2 = 512$  bits.

Now, assume that linear quantization is used with loss parameter  $p = 1$ . Assume that the maximum entry of the transform  $Y$  is 255. Then the largest possible entries of  $Y_Q$ , after quantization by  $Q$ , are





**Figure 11.9** Result of linear quantization for all 1024  $8 \times 8$  blocks. Loss parameters are (a)  $p = 1$  (b)  $p = 2$  (c)  $p = 4$ .

$$\begin{bmatrix} 32 & 16 & 11 & 8 & 6 & 5 & 5 & 4 \\ 16 & 11 & 8 & 6 & 5 & 5 & 4 & 4 \\ 11 & 8 & 6 & 5 & 5 & 4 & 4 & 3 \\ 8 & 6 & 5 & 5 & 4 & 4 & 3 & 3 \\ 6 & 5 & 5 & 4 & 4 & 3 & 3 & 3 \\ 5 & 5 & 4 & 4 & 3 & 3 & 3 & 2 \\ 5 & 4 & 4 & 3 & 3 & 3 & 2 & 2 \\ 4 & 4 & 3 & 3 & 3 & 2 & 2 & 2 \end{bmatrix}$$

Since both positive and negative entries are possible, the number of bits necessary to store each entry is

$$\begin{bmatrix} 7 & 6 & 5 & 5 & 4 & 4 & 4 & 4 \\ 6 & 5 & 5 & 4 & 4 & 4 & 4 & 4 \\ 5 & 5 & 4 & 4 & 4 & 4 & 4 & 3 \\ 5 & 4 & 4 & 4 & 4 & 4 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 3 & 3 & 3 & 3 & 3 & 3 \end{bmatrix}$$

The sum of these 64 numbers is 249, or  $249/64 \approx 3.89$  bits/pixel, which is less than one-half the number of bits (512, or 8 bits/pixel) needed to store the original pixel values of the  $8 \times 8$  image matrix. The corresponding statistics for other values of  $p$  are shown in the following table:

$p$	total bits	bits/pixel
1	249	3.89
2	191	2.98
4	147	2.30

As seen in the table, the number of bits necessary to represent the image is reduced by a factor of 2 when  $p = 1$ , with little recognizable change in the image. This compression is due to quantization. In order to compress further, we can take advantage of the fact that many of the high-frequency terms in the transform are zero after quantization. This is most efficiently done by using Huffman and run-length coding, introduced in the next section.

Linear quantization with  $p = 1$  is close to the default JPEG quantization. The quantization matrix that provides the most compression with the least image degradation has

been the subject of much research and discussion. The JPEG standard includes an appendix called “Annex K: Examples and Guidelines,” which contains a  $Q$  based on experiments with the human visual system. The matrix

$$Q_Y = p \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix} \quad (11.25)$$

is widely used in currently distributed JPEG encoders. Setting the loss parameter  $p = 1$  should give virtually perfect reconstruction as far as the human visual system is concerned, while  $p = 4$  usually introduces noticeable defects. To some extent, the visual quality depends on the pixel size: If the pixels are small, some errors may go unnoticed.

So far, we have discussed grayscale images only. It is fairly easy to extend application to color images, which can be expressed in the RGB color system. Each pixel is assigned three integers, one each for red, green, and blue intensity. One approach to image compression is to repeat the preceding processing independently for each of the three colors, treating each as if it were gray scale, and then to reconstitute the image from its three colors at the end.

Although the JPEG standard does not take a position on how to treat color, the method often referred to as Baseline JPEG uses a more delicate approach. Define the **luminance**  $Y = 0.299R + 0.587G + 0.114B$  and the **color differences**  $U = B - Y$  and  $V = R - Y$ . This transforms the RGB color data to the YUV system. This is a completely reversible transform, since the RGB values can be found as  $B = U + Y$ ,  $R = V + Y$ , and  $G = (Y - 0.299R - 0.114B)/(0.587)$ . Baseline JPEG applies the DCT filtering previously discussed independently to  $Y$ ,  $U$ , and  $V$ , using the quantization matrix  $Q_Y$  from Annex K for the luminance variable  $Y$  and the quantization matrix

$$Q_C = \begin{bmatrix} 17 & 18 & 24 & 47 & 99 & 99 & 99 & 99 \\ 18 & 21 & 26 & 66 & 99 & 99 & 99 & 99 \\ 24 & 26 & 56 & 99 & 99 & 99 & 99 & 99 \\ 47 & 66 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \end{bmatrix} \quad (11.26)$$

for the color differences  $U$  and  $V$ . After reconstructing  $Y$ ,  $U$ , and  $V$ , they are put back together and converted back to RGB to reconstitute the image.

Because of the less important roles of  $U$  and  $V$  in the human visual system, more aggressive quantization is allowed for them, as seen in (11.26). Further compression can be derived from an array of additional ad hoc tricks—for example, by averaging the color differences and treating them on a less fine grid.

## 11.2 Exercises

- Find the 2D-DCT of the following data matrices  $X$ , and find the corresponding interpolating function  $P_2(s, t)$  for the data points  $(i, j, x_{ij})$ ,  $i, j = 0, 1$ :

$$(a) \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \quad (b) \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} \quad (c) \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \quad (d) \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

2. Find the 2D-DCT of the data matrix  $X$ , and find the corresponding interpolating function  $P_n(s, t)$  for the data points  $(i, j, x_{ij}), i, j = 0, \dots, n - 1$ .

$$(a) \begin{bmatrix} 1 & 0 & -1 & 0 \\ 1 & 0 & -1 & 0 \\ 1 & 0 & -1 & 0 \\ 1 & 0 & -1 & 0 \end{bmatrix} \quad (b) \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$(c) \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (d) \begin{bmatrix} 3 & 3 & 3 & 3 \\ 3 & -1 & -1 & 3 \\ 3 & 3 & 3 & 3 \\ 3 & -1 & -1 & 3 \end{bmatrix}$$

3. Find the least squares approximation, using the basis functions  $1, \cos \frac{(2s+1)\pi}{8}, \cos \frac{(2t+1)\pi}{8}$  for the data in Exercise 2.
4. Use the quantization matrix  $Q = \begin{bmatrix} 10 & 20 \\ 20 & 100 \end{bmatrix}$  to quantize the matrices that follow. State the quantized matrix, the (lossy) dequantized matrix, and the matrix of quantization errors.

$$(a) \begin{bmatrix} 24 & 24 \\ 24 & 24 \end{bmatrix} \quad (b) \begin{bmatrix} 32 & 28 \\ 28 & 45 \end{bmatrix} \quad (c) \begin{bmatrix} 54 & 54 \\ 54 & 54 \end{bmatrix}$$

## 11.2 Computer Problems

1. Find the 2D-DCT of the data matrix  $X$ .

$$(a) \begin{bmatrix} -1 & 1 & -1 & 1 \\ -2 & 2 & -2 & 2 \\ -3 & 3 & -3 & 3 \\ -4 & 4 & -4 & 4 \end{bmatrix} \quad (b) \begin{bmatrix} 1 & 2 & -1 & -2 \\ -1 & -2 & 1 & 2 \\ 1 & 2 & -1 & -2 \\ -1 & -2 & 1 & 2 \end{bmatrix}$$

$$(c) \begin{bmatrix} 1 & 3 & 1 & -1 \\ 2 & 1 & 0 & 1 \\ 1 & -1 & 2 & 3 \\ 3 & 2 & 1 & 0 \end{bmatrix} \quad (d) \begin{bmatrix} -3 & -2 & -1 & 0 \\ -2 & -1 & 0 & 1 \\ -1 & 0 & 1 & 2 \\ 0 & 1 & 2 & 3 \end{bmatrix}$$

2. Using the 2D-DCT from Computer Problem 1, find the least squares low-pass filtered approximation to  $X$  by setting all transform values  $Y_{kl} = 0$  for  $k + l \geq 4$ .
3. Obtain a grayscale image file of your choice, and use the `imread` command to import into MATLAB. Crop the resulting matrix so that each dimension is a multiple of 8. If necessary, converting a color RGB image to gray scale can be accomplished by the standard formula (11.15).
- Extract an  $8 \times 8$  pixel block, for example, by using the MATLAB command `xb=x(81:88, 81:88)`. Display the block with the `imagesc` command.
  - Apply the 2D-DCT.
  - Quantize by using linear quantization with  $p = 1, 2,$  and  $4$ . Print out each  $Y_Q$ .

- (d) Reconstruct the block by using the inverse 2D-DCT, and compare with the original. Use MATLAB commands `colormap(gray)` and `imagesc(X, [0 255])`.
  - (e) Carry out (a)–(d) for all  $8 \times 8$  blocks, and reconstitute the image in each case.
4. Carry out the steps of Computer Problem 3, but quantize by the JPEG-suggested matrix (11.25) with  $p = 1$ .
  5. Obtain a color image file of your choice. Carry out the steps of Computer Problem 3 for colors R, G, and B separately, using linear quantization, and recombine as a color image.
  6. Obtain a color image, and transform the RGB values to luminance/color difference coordinates. Carry out the steps of Computer Problem 3 for  $Y$ ,  $U$ , and  $V$  separately by using JPEG quantization, and recombine as a color image.

## 11.3 HUFFMAN CODING

Lossy compression for images requires making a trade of accuracy for file size. If the reductions in accuracy are small enough to be unnoticeable for the intended purpose of the image, the trade may be worthwhile. The loss of accuracy occurs at the quantization step, after transforming to separate the image into its spatial frequencies. Lossless compression refers to further compression that may be applied without losing any more accuracy, simply due to efficient coding of the DCT-transformed, quantized image.

In this section, we discuss lossless compression. As a relevant application, there are simple, efficient methods for turning the quantized DCT transform matrix from the last section into a JPEG bit stream. Finding out how to do this will take us on a short tour of basic information theory.

### 11.3.1 Information theory and coding

Consider a message consisting of a string of symbols. The symbols are arbitrary; let us assume that they come from a finite set. In this section, we consider efficient ways to encode such a string in binary digits, or bits. The shorter the string of bits, the easier and cheaper it will be to store or transmit the message.

► **EXAMPLE 11.5** Encode the message ABAACDAB as a binary string.

Since there are four symbols, a convenient binary coding might associate two bits with each letter. For example, we could choose the correspondence

A	00
B	01
C	10
D	11

Then the message would be coded as

$$(00)(01)(00)(00)(10)(11)(00)(01).$$

With this code, a total of 16 bits is required to store or transmit the message. ◀

It turns out that there are more efficient coding methods. To understand them, we first have to introduce the idea of information. Assume that there are  $k$  different symbols, and denote by  $p_i$  the probability of the appearance of symbol  $i$  at any point in the string.

The probability might be known a priori, or it may be estimated empirically by dividing the number of appearances of symbol  $i$  in the string by the length of the string.

**DEFINITION 11.7** The **Shannon information**, or **Shannon entropy** of the string is  $I = -\sum_{i=1}^k p_i \log_2 p_i$ . □

The definition is named after C. Shannon of Bell Laboratories, who did seminal work on information theory in the middle of the 20th century. The Shannon information of a string is considered an average of the number of bits per symbol that is needed, at minimum, to code the message. The logic is as follows: On average, if a symbol appears  $p_i$  of the time, then one expects to need  $-\log_2 p_i$  bits to represent it. For example, a symbol that appears  $1/8$  of the time could be represented by one of the  $-\log_2(1/8) = 3$ -bit symbols 000, 001,  $\dots$ , 111, of which there are 8. To find the average bits per symbol over all symbols, we should weight the bits per symbol  $i$  by its probability  $p_i$ . This means that the average number of bits/symbol for the entire message is the sum  $I$  in the definition.

► **EXAMPLE 11.6** Find the Shannon information of the string ABAACDAB.

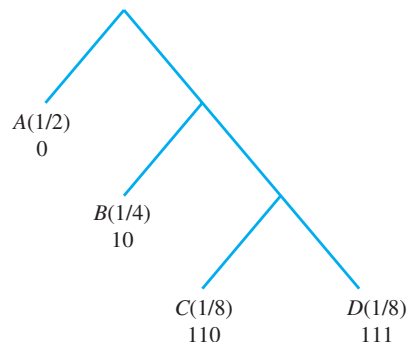
The empirical probabilities of appearance of the symbols  $A, B, C, D$  are  $p_1 = 4/8 = 2^{-1}$ ,  $p_2 = 2/8 = 2^{-2}$ ,  $p_3 = 1/8 = 2^{-3}$ ,  $p_4 = 2^{-3}$ , respectively. The Shannon information is

$$-\sum_{i=1}^4 p_i \log_2 p_i = \frac{1}{2}1 + \frac{1}{4}2 + \frac{1}{8}3 + \frac{1}{8}3 = \frac{7}{4}.$$

Thus, Shannon information estimates that at least 1.75 bits/symbol are needed to code the string. Since the string has length 8, the optimal total number of bits should be  $(1.75)(8) = 14$ , not 16, as we coded the string earlier.

In fact, the message can be sent in the predicted 14 bits, using the method known as **Huffman coding**. The goal is to assign a unique binary code to each symbol that reflects the probability of encountering the symbol, with more common symbols receiving shorter codes.

The algorithm works by building a tree from which the binary code can be read. Begin with two symbols with the smallest probability, and consider the “combined” symbol, assigning to it the combined probability. The two symbols form one branching of the tree. Then repeat this step, combining symbols and working up the branches of the tree, until there is only one symbol group left, which corresponds to the top of the tree. Here, we first combined the least probable symbols C and D into a symbol CD with probability  $1/4$ . The remaining probabilities are A ( $1/2$ ), B ( $1/4$ ), and CD ( $1/4$ ). Again, we combine the two least likely symbols to get A ( $1/2$ ), BCD ( $1/2$ ). Finally, combining the remaining two gives ABCD (1). Each combination forms a branch of the Huffman tree:



Once the tree is completed, the Huffman code for each symbol can be read by traversing the tree from the top, writing a 0 for a branch to the left and a 1 for a branch to the right, as shown above. For example, A is represented by 0, and C is represented by two rights and a left, 110. Now the string of letters ABAACDAB can be translated to a bit stream of length 14:

$$(0)(10)(0)(0)(110)(111)(0)(10).$$

The Shannon information of the message provides a lower bound for the bits/symbol of the binary coding. In this case, the Huffman code has achieved the Shannon information bound of  $14/8 = 1.75$  bits/symbol. Unfortunately, this is not always possible, as the next example shows.

► **EXAMPLE 11.7** Find the Shannon information and a Huffman coding of the message ABRA CADABRA.

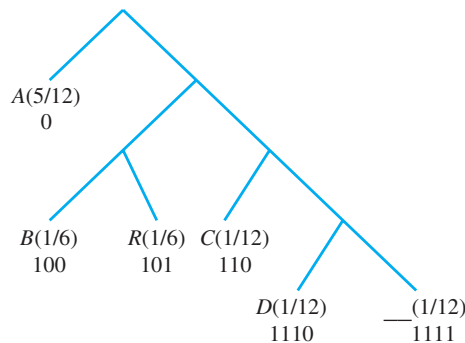
The empirical probabilities of the six symbols are

A	5/12
B	2/12
R	2/12
C	1/12
D	1/12
—	1/12

Note that the space has been treated as a symbol. The Shannon information is

$$-\sum_{i=1}^6 p_i \log_2 p_i = -\frac{5}{12} \log_2 \frac{5}{12} - 2 \frac{1}{6} \log_2 \frac{1}{6} - 3 \frac{1}{12} \log_2 \frac{1}{12} \approx 2.28 \text{ bits/symbol.}$$

This is the theoretical minimum for the average bits/symbol for coding the message ABRA CADABRA. To find the Huffman coding, proceed as already described. We begin by combining the symbols D and —, although any two of the three with probability 1/12 could have been chosen for the lowest branch. The symbol A comes in last, since it has highest probability. One Huffman coding is displayed in the diagram.



Note that A has a short code, due to the fact that it is a popular symbol in the message. The coded binary sequence for ABRA CADABRA is

$$(0)(100)(101)(0)(1111)(110)(0)(1110)(0)(100)(101)(0),$$

which has length 28 bits. The average for this coding is  $28/12 = 2\frac{1}{3}$  bits/symbol, slightly larger than the theoretical minimum previously calculated. Huffman codes cannot always match the Shannon information, but they often come very close. ◀

The secret of a Huffman code is the following: Since each symbol occurs only at the end of a tree branch, no complete symbol code can be the beginning of another symbol code. Therefore, there is no ambiguity when translating the code back into symbols.

### 11.3.2 Huffman coding for the JPEG format

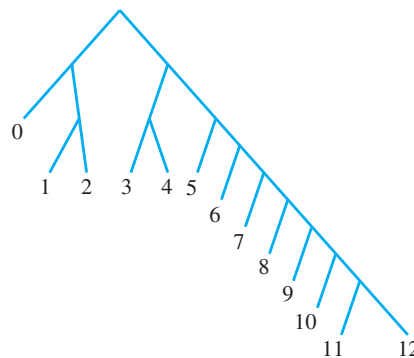
This section is devoted to an extended example of Huffman coding in practice. The JPEG image compression format is ubiquitous in modern digital photography. It makes a fascinating case study due to the juxtaposition of theoretical mathematics and engineering considerations.

The binary coding of transform coefficients for a JPEG image file uses Huffman coding in two different ways, one for the DC component (the (0, 0) entry of the transform matrix) and another for the other 63 entries of the 8 × 8 matrix, the so-called AC components.

**DEFINITION 11.8** Let  $y$  be an integer. The **size** of  $y$  is defined to be

$$L = \begin{cases} \text{floor}(\log_2 |y|) + 1 & \text{if } y \neq 0 \\ 0 & \text{if } y = 0 \end{cases} \quad \square$$

Huffman coding for JPEG has three ingredients: a Huffman tree for the DC components, another Huffman tree for the AC components, and an integer identifier table. The first part of the coding for the entry  $y = y_{00}$  is the binary coding for the size of  $y$ , from the following Huffman tree for DC components, called the **DPCM tree**, for Differential Pulse Code Modulation.



Again, the tree is to be interpreted by coding a 0 or 1 when going down a branch to the left or right, respectively. The first part is followed by a binary string from the following integer identifier table:

$L$	entry	binary
0	0	--
1	-1,1	0,1
2	-3,-2,2,3	00,01,10,11
3	-7,-6,-5,-4,4,5,6,7	000,001,010,011,100,101,110,111
4	-15,-14,...,-8,8,...,14,15	0000,0001,...,0111,1000,...,1110,1111
5	-31,-30,...,-16,16,...,30,31	00000,00001,...,01111,10000,...,11110,11111
6	-63,-62,...,-32,32,...,62,63	000000,000001,...,011111,100000,...,111110,111111
⋮	⋮	⋮

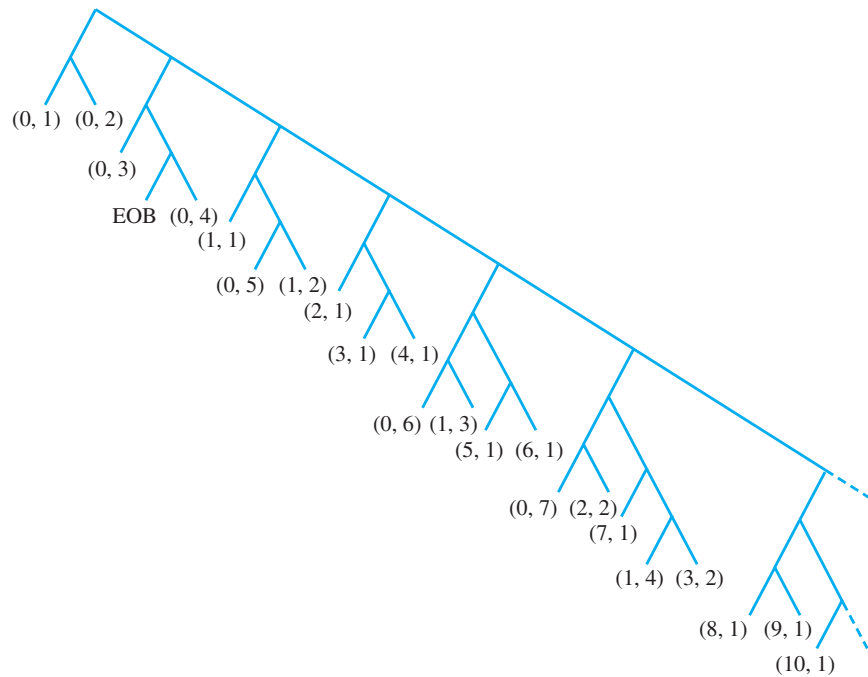
As an example, the entry  $y_{00} = 13$  would have size  $L = 4$ . According to the DPCM tree, the Huffman code for 4 is (101). The table shows that the extra digits for 13 are (1101), so the concatenation of the two parts, 1011101, would be stored for the DC component.

Since there are often correlations between the DC components of nearby 8 × 8 blocks, only the differences from block to block are stored after the first block. The differences are stored, moving from left to right, using the DPCM tree.

For the remaining 63 AC components of the  $8 \times 8$  block, **Run Length Encoding (RLE)** is used as a way to efficiently store long runs of zeros. The conventional order for storing the 63 components is the zigzag pattern

$$\begin{bmatrix} 0 & 1 & 5 & 6 & 14 & 15 & 27 & 28 \\ 2 & 4 & 7 & 13 & 16 & 26 & 29 & 42 \\ 3 & 8 & 12 & 17 & 25 & 30 & 41 & 43 \\ 9 & 11 & 18 & 24 & 31 & 40 & 44 & 53 \\ 10 & 19 & 23 & 32 & 39 & 45 & 52 & 54 \\ 20 & 22 & 33 & 38 & 46 & 51 & 55 & 60 \\ 21 & 34 & 37 & 47 & 50 & 56 & 59 & 61 \\ 35 & 36 & 48 & 49 & 57 & 58 & 62 & 63 \end{bmatrix} \quad (11.27)$$

Instead of coding the 63 numbers themselves, a zero run-length pair  $(n, L)$  is coded, where  $n$  denotes the length of a run of zeros, and  $L$  represents the size of the next nonzero entry. The most common codes encountered in typical JPEG images, and their default codings according to the JPEG standard, are shown in the Huffman tree for AC components.



In the bit stream, the Huffman code from the tree (which only identifies the size of the entry) is immediately followed by the binary code identifying the integer, from the previous table. For example, the sequence of entries  $-5, 0, 0, 0, 2$  would be represented as  $(0, 3) -5 (3, 2) 2$ , where  $(0, 3)$  means no zeros followed by a size 3 number, and  $(3, 2)$  represents 3 zeros followed by a size 2 number. From the Huffman tree, we find that  $(0, 3)$  codes as  $(100)$ , and  $(3, 2)$  as  $(111110111)$ . The identifier for  $-5$  is  $(010)$  and for  $2$  is  $(10)$ , from the integer identifier table. Therefore, the bit stream used to code  $-5, 0, 0, 0, 2$  is  $(100)(010)(111110111)(10)$ .

The preceding Huffman tree shows only the most commonly occurring JPEG run-length codes. Other useful codes are  $(11, 1) = 1111111001$ ,  $(12, 1) = 1111111010$ , and  $(13, 1) = 1111111000$ .

► **EXAMPLE 11.8** Code the quantized DCT transform matrix in (11.24) for a JPEG image file.

The DC entry  $y_{00} = -4$  has size 3, coded as  $(100)$  by the DPCM tree, and extra bits  $(011)$  from the integer identifier table. Next, we consider the AC coefficient string.



According to (11.27), the AC coefficients are ordered as  $-1, 3, 1, 0, 1, -1, -1$ , seven zeros, 1, four zeros,  $-1$ , three zeros,  $-1$ , and the remainder all zeros. The run-length encoding begins with  $-1$ , which has size 1 and so contributes (0, 1) from the run-length code. The next number 3 has size 2 and contributes (0, 2). The zero run-length pairs are

(0, 1)  $-1$  (0, 2) 3 (0, 1) 1 (1, 1) 1 (0, 1)  $-1$  (0, 1)  $-1$   
 (7, 1) 1 (4, 1)  $-1$  (3, 1)  $-1$  EOB.

Here, EOB stands for “end-of-block” and means that the remainder of the entries consists of zeros. Next, we read the bit representatives from the Huffman tree on page 518 and the integer identifier table. The bit stream that stores the  $8 \times 8$  block from the photo in Figure 11.8 (c) is listed below, where the parentheses are included only for human readability:

(100)(011)  
 (00)(0)(01)(11)(00)(1)(1100)(1)(00)(0)(00)(0)  
 (11111010)(1)(111011)(0)(111010)(0)(1010)

The pixel block in Figure 11.8(c), which is a reasonable approximation of the original Figure 11.6(a), is exactly represented by these 54 bits. On a per-pixel basis, this works out to  $54/64 \approx 0.84$  bits/pixel. Note the superiority of this coding to the bits/pixel achieved by low-pass filtering and quantization alone. Given that the pixels started out as 8-bit integers, the  $8 \times 8$  image has been compressed by more than a factor of 9:1. ◀

Decompressing a JPEG file consists of reversing the compression steps. The JPEG reader decodes the bit stream to run-length symbols, which form  $8 \times 8$  DCT transform blocks that in turn are finally converted back to pixel blocks with the use of the inverse DCT.

### 11.3 Exercises

- Find the probability of each symbol and the Shannon information for the messages.  
 (a) BABBCABB (b) ABCACCAB (c) ABABCABA
- Draw a Huffman tree and use it to code the messages in Exercise 1. Compare the Shannon information with the average number of bits needed per symbol.
- Draw a Huffman tree and convert the message, including spaces and punctuation marks, to a bit stream by using Huffman coding. Compare the Shannon information with the average number of bits needed per symbol. (a) AY CARUMBA! (b) COMPRESS THIS MESSAGE (c) SHE SELLS SEASHELLS BY THE SEASHORE
- Translate the transformed, quantized image components (a) (11.22) and (b) (11.23) to bit streams, using JPEG Huffman coding.

## 11.4 MODIFIED DCT AND AUDIO COMPRESSION

We return to the problem of one-dimensional signals and discuss state-of-the-art approaches to audio compression. Although one might think that one dimension is easier to handle than two, the challenge is that the human auditory system is very sensitive in the frequency domain, and unwanted artifacts introduced by compression and decompression are even more readily detected. For that reason, it is common for sound compression methods to make use of sophisticated tricks designed to hide the fact that compression has occurred.





$$= \left[ c_{\frac{n}{2}} \cdots c_{n-1} | -c_{n-1} \cdots -c_{\frac{n}{2}} | -c_{\frac{n}{2}-1} \cdots -c_0 | -c_0 \cdots -c_{\frac{n}{2}-1} \right]. \quad (11.32)$$

For example, the  $n = 4$  MDCT matrix is

$$M = [c_2c_3 | c_4c_5 | c_6c_7 | c_8c_9] = [c_2c_3 | -c_3 - c_2 | -c_1 - c_0 | -c_0 - c_1]$$

To simplify notation, let  $A$  and  $B$  denote the left and right halves of the DCT4 matrix, so that  $E = [A|B]$ . Define the permutation matrix formed by reversing the columns of the identity matrix, left for right:

$$R = \begin{bmatrix} & & & 1 \\ & & \cdot & \\ & \cdot & & \\ 1 & & & \end{bmatrix}.$$

The permutation matrix  $R$  reverses columns right for left when multiplying a matrix on the right. When multiplying on the left, it reverses rows top to bottom. Note that  $R$  is a symmetric orthogonal matrix, since  $R^{-1} = R^T = R$ . Now (11.32) can be written more simply as

$$M = (B | -BR | -AR | -A), \quad (11.33)$$

where  $AR$  and  $BR$  are versions of  $A$  and  $B$  in which the order of the columns has been reversed, left for right.

The action of MDCT can be expressed in terms of DCT4. Let

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

be a  $2n$ -vector, where each  $x_i$  is a length  $n/2$  vector (remember that  $n$  is even). Then, by the characterization of  $M$  in (11.33),

$$\begin{aligned} Mx &= Bx_1 - BRx_2 - ARx_3 - Ax_4 \\ &= [A|B] \begin{bmatrix} -Rx_3 - x_4 \\ x_1 - Rx_2 \end{bmatrix} = E \begin{bmatrix} -Rx_3 - x_4 \\ x_1 - Rx_2 \end{bmatrix}, \end{aligned} \quad (11.34)$$

where  $E$  is the  $n \times n$  DCT4 matrix and  $Rx_2$  and  $Rx_3$  represent  $x_2$  and  $x_3$  with their entries reversed top to bottom. This is very helpful—we can express the output of  $M$  in terms of an orthogonal matrix  $E$ .

Since the  $n \times 2n$  matrix  $M$  of the MDCT is not a square matrix, it is not invertible. However, two adjacent MDCT's can have rank  $2n$  in total, and working together, can reconstruct the input  $x$ -values perfectly, as we now show.

The “inverse” MDCT is represented by the  $2n \times n$  matrix  $N = M^T$ , which has transposed entries

$$N_{ij} = \sqrt{\frac{2}{n}} \cos \frac{(j + \frac{1}{2})(i + \frac{n}{2} + \frac{1}{2})\pi}{n}. \quad (11.35)$$

It is not an actual inverse, although it is as close as it can be for a rectangular matrix. By transposing (11.33), we have

$$N = \begin{bmatrix} B^T \\ -RB^T \\ -RA^T \\ -A^T \end{bmatrix}, \quad (11.36)$$

using our earlier notation  $E = [A|B]$  for the Discrete Cosine Transform DCT4. We know that since  $E$  is an orthogonal matrix,

$$\begin{aligned} A^T A &= I \\ B^T B &= I \\ A^T B &= B^T A = 0, \end{aligned}$$

where  $I$  denotes the  $n \times n$  identity matrix.

Now we are ready to calculate  $NM$ , to see in what sense  $N$  inverts the MDCT matrix  $M$ . Let  $x$  be partitioned into four parts, as before. According to (11.34) and (11.36), the orthogonality of  $A$  and  $B$ , and the fact that  $R^2 = I$ , we have

$$\begin{aligned} NM \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} &= \begin{bmatrix} B^T \\ -RB^T \\ -RA^T \\ -A^T \end{bmatrix} [A(-Rx_3 - x_4) + B(x_1 - Rx_2)] \\ &= \begin{bmatrix} x_1 - Rx_2 \\ -Rx_1 + x_2 \\ x_3 + Rx_4 \\ Rx_3 + x_4 \end{bmatrix}. \end{aligned} \quad (11.37)$$

In audio compression algorithms, MDCT is applied to vectors of data that overlap. The reason is that any artifacts due to the ends of the vectors will occur with a fixed frequency, because of the constant vector length. The auditory system is even more sensitive to periodic errors than the visual system; after all, an error of fixed frequency is a tone of that frequency, which the ear is designed to pick up. Assume that the data will be presented in overlapped fashion. Let

$$Z_1 = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \text{ and } Z_2 = \begin{bmatrix} x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix}$$

be two  $2n$ -vectors for an even integer  $n$ , where each  $x_i$  is a length  $n/2$  vector. The vectors  $Z_1$  and  $Z_2$  overlap by half of their length. Since (11.37) shows that

$$NMZ_1 = \begin{bmatrix} x_1 - Rx_2 \\ -Rx_1 + x_2 \\ x_3 + Rx_4 \\ Rx_3 + x_4 \end{bmatrix} \text{ and } NMZ_2 = \begin{bmatrix} x_3 - Rx_4 \\ -Rx_3 + x_4 \\ x_5 + Rx_6 \\ Rx_5 + x_6 \end{bmatrix}, \quad (11.38)$$

we can reconstruct the  $n$ -vector  $[x_3, x_4]$  exactly by averaging the bottom half of  $NMZ_1$  and the top half of  $NMZ_2$ :

$$\begin{bmatrix} x_3 \\ x_4 \end{bmatrix} = \frac{1}{2}(NMZ_1)_{n, \dots, 2n-1} + \frac{1}{2}(NMZ_2)_{0, \dots, n-1}. \quad (11.39)$$

This equality is how  $N$  is used to decode the signal after being coded by  $M$ .

This result is summarized in Theorem 11.12.

**THEOREM 11.12 Inversion of MDCT through overlapping.** Let  $M$  be the  $n \times 2n$  MDCT matrix, and  $N = M^T$ . Let  $u_1, u_2, u_3$  be  $n$ -vectors, and set

$$v_1 = M \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \text{ and } v_2 = M \begin{bmatrix} u_2 \\ u_3 \end{bmatrix}.$$

Then the  $n$ -vectors  $w_1, w_2, w_3, w_4$  defined by

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = Nv_1 \text{ and } \begin{bmatrix} w_3 \\ w_4 \end{bmatrix} = Nv_2$$

satisfy  $u_2 = \frac{1}{2}(w_2 + w_3)$ . ■

This is exact reconstruction. Theorem 11.12 is customarily used with a long signal of concatenated  $n$ -vectors  $[u_1, u_2, \dots, u_m]$ . The MDCT is applied to adjacent pairs to get a transformed signal  $(v_1, v_2, \dots, v_{m-1})$ . Now the lossy compression comes in. The  $v_i$  are frequency components, so we can choose to keep certain frequencies and de-emphasize others. We will take up this direction in the next section.

After shrinking the content of the  $v_i$  by quantization or other means,  $(u_2, \dots, u_{m-1})$  can be decompressed by Theorem 11.12. Note that we cannot recover  $u_1$  and  $u_m$ ; they should either be unimportant parts of the signal or padding that is added beforehand.

► **EXAMPLE 11.9** Use the overlapped MDCT to transform the signal  $x = [1, 2, 3, 4, 5, 6]$ . Then invert the transform to reconstruct the middle section  $[3, 4]$ .

We will overlap the vectors  $[1, 2, 3, 4]$  and  $[3, 4, 5, 6]$ . Let  $n = 2$  and set

$$E_2 = \begin{bmatrix} \cos \frac{\pi}{8} & \cos \frac{3\pi}{8} \\ \cos \frac{3\pi}{8} & \cos \frac{9\pi}{8} \end{bmatrix} = \begin{bmatrix} b & c \\ c & -b \end{bmatrix}.$$

Note that our definitions of  $b$  and  $c$  have changed slightly from (11.7) to be compatible with the MDCT. Applying the  $2 \times 4$  MDCT gives

$$v_1 = M \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} = E_2 \begin{bmatrix} -R(3) - 4 \\ 1 - R(2) \end{bmatrix} = E_2 \begin{bmatrix} -7 \\ -1 \end{bmatrix} = \begin{bmatrix} -7b - c \\ b - 7c \end{bmatrix} = \begin{bmatrix} -6.8498 \\ -1.7549 \end{bmatrix}$$

$$v_2 = M \begin{bmatrix} 3 \\ 4 \\ 5 \\ 6 \end{bmatrix} = E_2 \begin{bmatrix} -R(5) - 6 \\ 3 - R(4) \end{bmatrix} = E_2 \begin{bmatrix} -11 \\ -1 \end{bmatrix} = \begin{bmatrix} -11b - c \\ b - 11c \end{bmatrix} = \begin{bmatrix} -10.5454 \\ -3.2856 \end{bmatrix}.$$

The transformed signal is represented by

$$[v_1 | v_2] = \begin{bmatrix} -6.8498 & -10.5454 \\ -1.7549 & -3.2856 \end{bmatrix}.$$

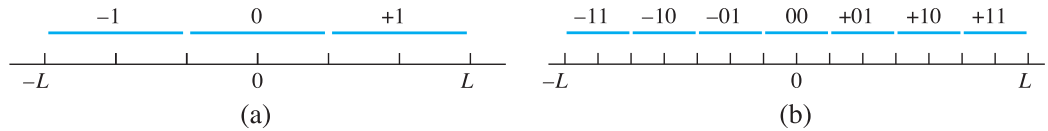
To invert the MDCT, define  $A$  and  $B$  by

$$E_2 = \left[ A \mid B \right] = \begin{bmatrix} b & c \\ c & -b \end{bmatrix}$$

and calculate

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = Nv_1 = \begin{bmatrix} B^T v_1 \\ -RB^T v_1 \\ -RA^T v_1 \\ -A^T v_1 \end{bmatrix} = \begin{bmatrix} c & -b \\ -c & b \\ -b & -c \\ -b & -c \end{bmatrix} \begin{bmatrix} -7b - c \\ b - 7c \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \\ 7 \\ 7 \end{bmatrix}$$

$$\begin{bmatrix} w_3 \\ w_4 \end{bmatrix} = Nv_2 = \begin{bmatrix} B^T v_2 \\ -RB^T v_2 \\ -RA^T v_2 \\ -A^T v_2 \end{bmatrix} = \begin{bmatrix} c & -b \\ -c & b \\ -b & -c \\ -b & -c \end{bmatrix} \begin{bmatrix} -11b - c \\ b - 11c \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \\ 11 \\ 11 \end{bmatrix},$$



**Figure 11.11 Bit quantization.** Illustration of (11.39). (a) 2 bits (b) 3 bits.

where we have used the fact  $b^2 + c^2 = 1$ . The result of Theorem 11.12 is that we can recover the overlap  $[3, 4]$  by

$$u_2 = \frac{1}{2}(w_2 + w_3) = \frac{1}{2} \left( \begin{bmatrix} 7 \\ 7 \end{bmatrix} + \begin{bmatrix} -1 \\ 1 \end{bmatrix} \right) = \begin{bmatrix} 3 \\ 4 \end{bmatrix}. \quad \blacktriangleleft$$

The definition and use of MDCT is less direct than the use of the DCT, discussed earlier in the chapter. Its advantage is that it allows overlapping of adjacent vectors in an efficient way. The effect is to average contributions from two vectors, reducing artifacts from abrupt transitions seen at boundaries. As in the case of DCT, we can filter or quantize the transform coefficients before reconstructing the signal in order to improve or compress the signal. Next, we show how the MDCT can be used for compression by adding a quantization step.

### 11.4.2 Bit quantization

Lossy compression of audio signals is achieved by quantizing the output of a signal's MDCT. In this section, we will expand on the quantization used for image compression, to allow more control over the number of bits used to represent the lossy version of the signal.

Start with the open interval of real numbers  $(-L, L)$ . Assume that the goal is to represent a number in  $(-L, L)$  by  $b$  bits, and that we are willing to live with a little error. We will use one bit for the sign and quantize to a binary integer of  $b - 1$  bits. The formula follows:

**$b$ -bit quantization of  $(-L, L)$**

$$\text{Quantization: } z = \text{round} \left( \frac{y}{q} \right), \text{ where } q = \frac{2L}{2^b - 1}$$

$$\text{Dequantization: } \bar{y} = qz \tag{11.40}$$

As an example, we show how to represent the numbers in the interval  $(-1, 1)$  by 4 bits. Set  $q = 2(1)/(2^4 - 1) = 2/15$ , and quantize by  $q$ . The number  $y = -0.3$  is represented by

$$\frac{-0.3}{2/15} = -\frac{9}{4} \rightarrow -2 \rightarrow -010,$$

and the number  $y = 0.9$  is represented by

$$\frac{0.9}{2/15} = \frac{27}{4} = 6.75 \rightarrow 7 \rightarrow +111.$$

Dequantization reverses the process. The quantized version of  $-0.3$  is dequantized as

$$(-2)q = (-2)(2/15) = -4/15 \approx -0.2667$$

and the quantized version of 0.9 as

$$(7)q = (7)(2/15) = 14/15 \approx 0.9333.$$

In both cases, the quantization error is  $1/30$ .

► **EXAMPLE 11.10** Quantize the MDCT output of Example 11.9 to 4-bit integers. Then dequantize, invert the MDCT, and find the quantization error.

All transform entries lie in the interval  $(-12, 12)$ . Using  $L = 12$ , four-bit quantization requires  $q = 2(12)/(2^4 - 1) = 1.6$ . Then

$$v_1 = \begin{bmatrix} -6.8498 \\ -1.7549 \end{bmatrix} \rightarrow \begin{bmatrix} \text{round}(\frac{-6.8498}{1.6}) \\ \text{round}(\frac{-1.7549}{1.6}) \end{bmatrix} \rightarrow \begin{bmatrix} -4 \\ -1 \end{bmatrix} \rightarrow \begin{matrix} -100 \\ -001 \end{matrix}$$

and

$$v_2 = \begin{bmatrix} -10.5454 \\ -3.2856 \end{bmatrix} \rightarrow \begin{bmatrix} \text{round}(\frac{-10.5454}{1.6}) \\ \text{round}(\frac{-3.2856}{1.6}) \end{bmatrix} \rightarrow \begin{bmatrix} -7 \\ -2 \end{bmatrix} \rightarrow \begin{matrix} -111 \\ -010 \end{matrix}.$$

The transform variables  $v_1, v_2$  can be stored as four 4-bit integers, for a total of 16 bits.

Dequantization with  $q = 1.6$  is

$$\begin{bmatrix} -4 \\ -1 \end{bmatrix} \rightarrow \begin{bmatrix} -6.4 \\ -1.6 \end{bmatrix} = \bar{v}_1$$

and

$$\begin{bmatrix} -7 \\ -2 \end{bmatrix} \rightarrow \begin{bmatrix} -11.2 \\ -3.2 \end{bmatrix} = \bar{v}_2.$$

Applying the inverse MDCT yields

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = N\bar{v}_1 = \begin{bmatrix} -0.9710 \\ 0.9710 \\ 6.5251 \\ 6.5251 \end{bmatrix},$$

$$\begin{bmatrix} w_3 \\ w_4 \end{bmatrix} = N\bar{v}_2 = \begin{bmatrix} -1.3296 \\ 1.3296 \\ 11.5720 \\ 11.5720 \end{bmatrix},$$

and the reconstructed signal

$$u_2 = \frac{1}{2}(w_2 + w_3) = \frac{1}{2} \left( \begin{bmatrix} 6.5251 \\ 6.5251 \end{bmatrix} + \begin{bmatrix} -1.3296 \\ 1.3296 \end{bmatrix} \right) = \begin{bmatrix} 2.5977 \\ 3.9274 \end{bmatrix}.$$

The quantization error is the difference between the original and reconstructed signals:

$$\left| \begin{bmatrix} 2.5977 \\ 3.9274 \end{bmatrix} - \begin{bmatrix} 3 \\ 4 \end{bmatrix} \right| = \begin{bmatrix} 0.4023 \\ 0.0726 \end{bmatrix}.$$

Coding of audio files is usually done by using a preset allocation of bits for prescribed frequency ranges. Reality Check 11 guides the reader through construction of a complete **codec**, or code–decode protocol, that uses the MDCT along with bit quantization. ◀



## 11.4 Exercises

- Find the MDCT of the input. Express the answer in terms of  $b = \cos \pi/8$  and  $c = \cos 3\pi/8$ .  
(a) [1, 3, 5, 7] (b) [-2, -1, 1, 2] (c) [4, -1, 3, 5]
- Find the MDCT of the two overlapping length 4 windows of the given input, as in Example 11.9. Then reconstruct the middle section, using the inverse MDCT.  
(a) [-3, -2, -1, 1, 2, 3] (b) [1, -2, 2, -1, 3, 0] (c) [4, 1, -2, -3, 0, 3]
- Quantize each real number in  $(-1, 1)$  to 4 bits, and then dequantize and compute the quantization error. (a)  $2/3$  (b)  $0.6$  (c)  $3/7$
- Repeat Exercise 3, but quantize to 8 bits.
- Quantize each real number in  $(-4, 4)$  to 8 bits, and then dequantize and compute the quantization error. (a)  $3/2$  (b)  $-7/5$  (c)  $2.9$  (d)  $\pi$
- Show that the DCT4  $n \times n$  matrix is an orthogonal matrix for each even integer  $n$ .
- Reconstruct the middle section of the data in Exercise 2 after quantizing to 4 bits in  $(-6, 6)$ . Compare with the correct middle section.
- Reconstruct the middle section of the data in Exercise 2 after quantizing to 6 bits in  $(-6, 6)$ . Compare with the correct middle section.
- Explain why the  $n$ -dimensional column vector  $c_k$  defined by (11.28) for any integer  $k$  can be expressed in terms of a column  $c_{k'}$  for  $0 \leq k' \leq n - 1$ . Express  $c_{5n}$  and  $c_{6n}$  in this way.
- Find an upper bound for the quantization error (the error caused by quantization, followed by dequantization) when converting a real number to a  $b$ -bit integer in the interval  $(-L, L)$ .

## 11.4 Computer Problems

- Write a MATLAB program to accept as input a vector, apply the MDCT to each of the length  $2n$  windows, and reconstruct the overlapped length  $n$  sections, as in Example 11.9. Demonstrate that it works on the following input signals. (a)  $n = 4, x = [1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12]$   
(b)  $n = 4, x_i = \cos(i\pi/6)$  for  $i = 0, \dots, 11$  (c)  $n = 8, x_i = \cos(i\pi/10)$  for  $i = 0, \dots, 63$
- Adapt your program from Computer Problem 1 to apply  $b$ -bit quantization before reconstructing the overlaps. Then reconstruct the examples from that problem, and compute the reconstruction errors by comparing with the original input.



### 11 A Simple Audio Codec

Efficient transmission and storage of audio files is a key part of modern communications, and the part played by compression is crucial. In this Reality Check, you will put together a bare-bones compression–decompression protocol based on the ability of the MDCT to split the audio signal into its frequency components and the bit quantization method of Section 11.4.2.

The MDCT is applied to an input window of  $2n$  signal values and provides an output of  $n$  frequency components that approximate the data (and together with the next window, interpolates the latter  $n$  input points). The compression part of the algorithm consists of coding the frequency components after quantization to save space, as demonstrated in Example 11.10.

In common audio storage formats, the way the bits are allocated to the various frequency components during quantization is based on **psychoacoustics**, the science of human sound perception. Techniques such as **frequency masking**, the empirical fact that the ear can handle only one dominant sound in each frequency range at a given time, are used to decide which frequency components are most and least important to preserve. More quantization bits are allocated to more important components. Most competitive methods are based on the MDCT and differ on how the psychoacoustic factors are treated. In our description, we will take a simplified approach that ignores most psychoacoustic factors and relies simply on **importance filtering**, the tendency to apportion more bits to frequency components of greater magnitude.

We begin with the reconstruction of a pure tone. Setting  $n = 32$ , the bottom frequency tone catalogued by the MDCT is 64 Hz, at the lower edge of perceptible frequencies for the human ear. A pure 64-Hz tone is represented by  $x(t) = \cos 2\pi(64)t$ , where  $t$  is measured in seconds. If  $F_s$  is the number of samples per second, then  $1/F_s, 2/F_s, \dots, F_s/F_s$  represent one second worth of points in time. The MATLAB commands

```
Fs=8192;
x=cos(2*pi*64*(1:Fs)/Fs);
sound(x,Fs)
```

play one second of a 64-Hz tone. The sampling frequency  $F_s$  of  $8192 = 2^{13}$  bytes/sec is quite common, corresponding to  $2^{16} = 65536$  bits/sec, referred to as a 64Kb/sec sampling rate for an audio file. (Higher quality files are often sampled at two or three times this rate, at 128 or 192 Kbs.)

Higher pitch tones are obtained by replacing 64 by an integer multiple  $64f$ . Setting  $f = 2$  or 4 gives higher octave versions. Setting  $f = 7$  plays a 448-Hz tone, just far enough from concert A (440 Hz) that if you have friends with perfect pitch, it should drive them to distraction in short order.

The MATLAB code fragment shown next applies the MDCT and quantizes, followed by an immediate dequantization and inverse MDCT on the overlapped segments, as described in Section 11.4. In this way, the effect of the quantization error that accompanies lossy compression can be examined.

```
n=32; % length of window
nb=127; % number of windows; must be > 1
b=4; L=5; % quantization information
q=2*L/(2^b-1); % b bits on interval [-L, L]
for i=1:n % form the MDCT matrix
    for j=1:2*n
        M(i,j) = cos((i-1+1/2)*(j-1+1/2+n/2)*pi/n);
    end
end
M=sqrt(2/n)*M;
N=M'; % inverse MDCT
Fs=8192;f=7; % Fs=sampling rate
x=cos((1:4096)*pi*64*f/4096); % test signal
sound(x,Fs) % Matlab's sound command
out=[];
for k=1:nb % loop over windows
    x0=x(1+(k-1)*n:2*n+(k-1)*n)';
    y0=M*x0;
    y1=round(y0/q); % transform components quantized
    y2=y1*q; % and dequantized
    w(:,k)=N*y2; % invert the MDCT
```

```

if (k>1)
    w2=w(n+1:2*n,k-1);w3=w(1:n,k);
    out=[out;(w2+w3)/2];      % collect the reconstructed signal
end
end
pause(1)
sound(out,Fs)                % play the reconstructed tone

```


The code plays the original 1/2-second tone (448 Hz), followed by the reconstructed tone. Compare the effect of changing the number of bits that represent the transform components, given by variable  $b$  in the code.

### Suggested activities:

1. How is the output of the MDCT different for odd integer values of  $f$ , compared with even values? Explain why the number of bits needed to make the reconstructed sound similar to the original differs for odd versus even  $f$ .
2. Add a “window function” to the code. The window function scales the input signal  $x$  smoothly to zero at each end of the window, counteracting the problem that the signal is not exactly periodic. A common choice is to replace  $x_i$  with  $x_i h_i$ , where

$$h_i = \sqrt{2} \sin \frac{(i - \frac{1}{2})\pi}{2n}$$

for a length  $2n$  window. To undo the window function, multiply the inverse MDCT outputs  $w_2$  and  $w_3$  componentwise by the same  $h_i$ ; this uses the orthogonality of sine, since the window function is now offset by 1/4 period. Compare the effect of the window function on the number of bits necessary to reconstruct the tone well.

3. Introduce importance sampling. Make a new test tone that is a combination of pure tones. Modify the code so that each of the 32 frequency components of  $y$  has its own number  $b_k$  of bits for quantization. Propose a method that makes  $b_k$  larger if the contributions  $|y_k|$  are larger, on average. Count the number of bits required to hold the signal, and refine your proposal.
4. Build two separate subprograms, a coder and a decoder. The coder should write a file (or MATLAB variable) of bits representing the quantized output of the MDCT and print the number of bits used. The decoder should load the file written by the coder and reconstruct the signal.
5. Download a .wav file with the MATLAB `wavread` command, or download another audio file of your choice. (Alternatively, `handel` can be used. If you use a stereo file, you will need to work with each channel separately.) Propose and implement a method to determine the best allocation of bits, as represented by the  $b_k$ . Use the coder to compress the audio file and the decoder to decompress. Compare sound quality of different results, where differing amounts of compression have been accomplished.
6. Investigate further tricks the sound industry uses to make compression more effective. For example, in the case of a stereo audio file, is there a better approach than treating the channels  $s_1$  and  $s_2$  separately? Why might it be advantageous to compress  $(s_1 + s_2)/2$  and  $(s_1 - s_2)/2$  instead? 

## Software and Further Reading

---

For good practical introductions to data compression, see Nelson and Gailly [1995], Storer [1988], and Sayood [1996]. General references on image and sound compression are Bhaskaran and Konstantinides [1995]. Rao and Yip [1990] is a good source for information on the Discrete Cosine Transform. The seminal article on Huffman coding is Huffman [1952].

We have introduced the baseline JPEG standard (Wallace [1991]) for image compression. The full standard is available in Pennebaker and Mitchell [1993]. The recently introduced JPEG-2000 standard (Taubman and Marcellin [2002]) allows wavelet compression in place of DCT.

Most protocols for sound compression are based on the Modified Discrete Cosine Transform (Wang and Viterbo [2003], Malvar [1992]). More specific information can be found on the individual formats like MP3 (shorthand for MPEG audio layer 3, see Hacker [2000]), AAC (Advanced Audio Coding, used in Apple iTunes and QuickTime video, and XM satellite radio), and the open-source audio format Ogg-Vorbis.