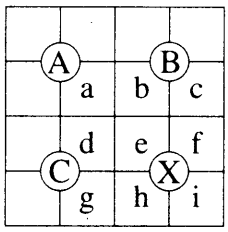

FIGURE 7.12 Pixels used to determine the value of a lower-level pixel.

If the value of this expression is greater than 4.5, the pixel X is tentatively declared to be 1. The table has certain exceptions to this rule to reduce the amount of edge smearing, generally encountered in a filtering operation. There are also exceptions that preserve periodic patterns and dither patterns.

As the lower-resolution layers are obtained from the higher-resolution images, we can use them when encoding the higher-resolution images. The JBIG specification makes use of the lower-resolution images when encoding the higher-resolution images by using the pixels of the lower-resolution images as part of the context for encoding the higher-resolution images. The contexts used for coding the lowest-resolution layer are those shown in Figure 7.10. The contexts used in coding the higher-resolution layer are shown in Figure 7.13.

Ten pixels are used in each context. If we include the 2 bits required to indicate which context template is being used, 12 bits will be used to indicate the context. This means that we can have 4096 different contexts.

Comparison of MH, MR, MMR, and JBIG

In this section we have seen three old facsimile coding algorithms: modified Huffman, modified READ, and modified modified READ. Before we proceed to the more modern techniques found in T.88 and T.42, we compare the performance of these algorithms with the earliest of the modern techniques, namely JBIG. We described the JBIG algorithm as an application of arithmetic coding in Chapter 4. This algorithm has been standardized in ITU-T recommendation T.82. As we might expect, the JBIG algorithm performs better than the MMR algorithm, which performs better than the MR algorithm, which in turn performs better than the MH algorithm. The level of complexity also follows the same trend, although we could argue that MMR is actually less complex than MR.

A comparison of the schemes for some facsimile sources is shown in Table 7.4. The modified READ algorithm was used with $K = 4$, while the JBIG algorithm was used with an adaptive three-line template and adaptive arithmetic coder to obtain the results in this table. As we go from the one-dimensional MH coder to the two-dimensional MMR coder, we get a factor of two reduction in file size for the sparse text sources. We get even more reduction when we use an adaptive coder and an adaptive model, as is true for the JBIG coder. When we come to the dense text, the advantage of the two-dimensional MMR over the one-dimensional MH is not as significant, as the amount of two-dimensional correlation becomes substantially less.

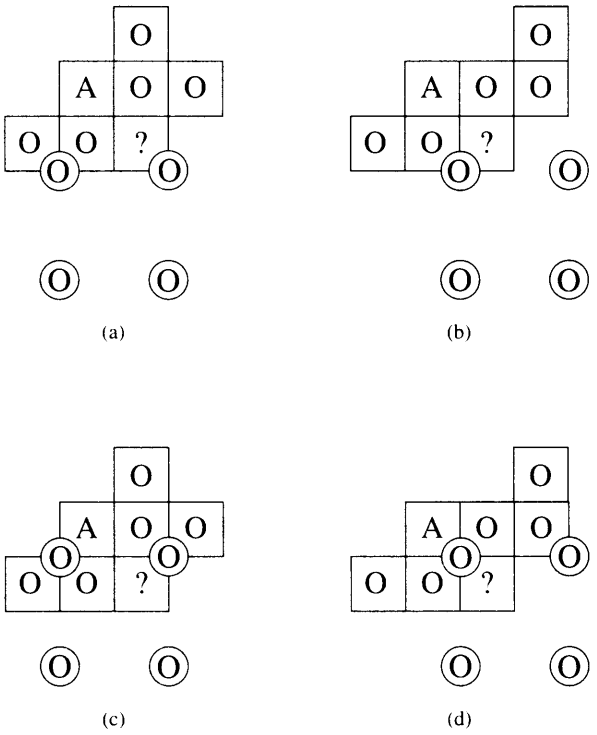

FIGURE 7.13 **Contexts used in the coding of higher-resolution layers.**

TABLE 7.4 **Comparison of binary image coding schemes. Data from [91].**

Source Description	Original Size (pixels)	MH (bytes)	MR (bytes)	MMR (bytes)	JBIG (bytes)
Letter	4352 × 3072	20,605	14,290	8,531	6,682
Sparse text	4352 × 3072	26,155	16,676	9,956	7,696
Dense text	4352 × 3072	135,705	105,684	92,100	70,703

The compression schemes specified in T.4 and T.6 break down when we try to use them to encode halftone images. In halftone images, gray levels are represented using binary pixel patterns. A gray level closer to black would be represented by a pattern that contains more black pixels, while a gray level closer to white would be represented by a pattern with fewer black pixels. Thus, the model that was used to develop the compression schemes specified in T.4 and T.6 is not valid for halftone images. The JBIG algorithm, with its adaptive model and coder, suffers from no such drawbacks and performs well for halftone images also [91].

7.6.4 JBIG2—T.88

The JBIG2 standard was approved in February of 2000. Besides facsimile transmission, the standard is also intended for document storage, archiving, wireless transmission, print spooling, and coding of images on the Web. The standard provides specifications only for the decoder, leaving the encoder design open. This means that the encoder design can be constantly refined, subject only to compatibility with the decoder specifications. This situation also allows for lossy compression, beacuse the encoder can incorporate lossy transformations to the data that enhance the level of compression.

The compression algorithm in JBIG provides excellent compression of a generic bi-level image. The compression algorithm proposed for JBIG2 uses the same arithmetic coding scheme as JBIG. However, it takes advantage of the fact that a significant number of bi-level images contain structure that can be used to enhance the compression performance. A large percentage of bi-level images consist of text on some background, while another significant percentage of bi-level images are or contain halftone images. The JBIG2 approach allows the encoder to select the compression technique that would provide the best performance for the type of data. To do so, the encoder divides the page to be compressed into three types of regions called *symbol regions*, *halftone regions*, and *generic regions*. The symbol regions are those containing text data, the halftone regions are those containing halftone images, and the generic regions are all the regions that do not fit into either category.

The partitioning information has to be supplied to the decoder. The decoder requires that all information provided to it be organized in *segments* that are made up of a segment header, a data header, and segment data. The page information segment contains information about the page including the size and resolution. The decoder uses this information to set up the page buffer. It then decodes the various regions using the appropriate decoding procedure and places the different regions in the appropriate location.

Generic Decoding Procedures

There are two procedures used for decoding the generic regions: the generic region decoding procedure and the generic refinement region decoding procedure. The generic region decoding procedure uses either the MMR technique used in the Group 3 and Group 4 fax standards or a variation of the technique used to encode the lowest-resolution layer in the JBIG recommendation. We describe the operation of the MMR algorithm in Chapter 6. The latter procedure is described as follows.

The second generic region decoding procedure is a procedure called *typical prediction*. In a bi-level image, a line of pixels is often identical to the line above. In typical prediction, if the current line is the same as the line above, a bit flag called $LNTP_n$ is set to 0, and the line is not transmitted. If the line is not the same, the flag is set to 1, and the line is coded using the contexts currently used for the low-resolution layer in JBIG. The value of $LNTP_n$ is encoded by generating another bit, $SLNTP_n$, according to the rule

$$SLNTP_n = !(LNTP_n \oplus LNTP_{n-1})$$

which is treated as a virtual pixel to the left of each row. If the decoder decodes an *LNTP* value of 0, it copies the line above. If it decodes an *LNTP* value of 1, the following bits

in the segment data are decoded using an arithmetic decoder and the contexts described previously.

The generic refinement decoding procedure assumes the existence of a *reference* layer and decodes the segment data with reference to this layer. The standard leaves open the specification of the reference layer.

Symbol Region Decoding

The symbol region decoding procedure is a dictionary-based decoding procedure. The symbol region segment is decoded with the help of a symbol dictionary contained in the symbol dictionary segment. The data in the symbol region segment contains the location where a symbol is to be placed, as well as the index to an entry in the symbol dictionary. The symbol dictionary consists of a set of bitmaps and is decoded using the generic decoding procedures. Note that because JBIG2 allows for lossy compression, the symbols do not have to exactly match the symbols in the original document. This feature can significantly increase the compression performance when the original document contains noise that may preclude exact matches with the symbols in the dictionary.

Halftone Region Decoding

The halftone region decoding procedure is also a dictionary-based decoding procedure. The halftone region segment is decoded with the help of a halftone dictionary contained in the halftone dictionary segment. The halftone dictionary segment is decoded using the generic decoding procedures. The data in the halftone region segment consists of the location of the halftone region and indices to the halftone dictionary. The dictionary is a set of fixed-size halftone patterns. As in the case of the symbol region, if lossy compression is allowed, the halftone patterns do not have to exactly match the patterns in the original document. By allowing for nonexact matches, the dictionary can be kept small, resulting in higher compression.

7.7 MRC—T.44

With the rapid advance of technology for document production, documents have changed in appearance. Where a document used to be a set of black and white printed pages, now documents contain multicolored text as well as color images. To deal with this new type of document, the ITU-T developed the recommendation T.44 for Mixed Raster Content (MRC). This recommendation takes the approach of separating the document into elements that can be compressed using available techniques. Thus, it is more an approach of partitioning a document image than a compression technique. The compression strategies employed here are borrowed from previous standards such as JPEG (T.81), JBIG (T.82), and even T.6.

The T.44 recommendation divides a page into slices where the width of the slice is equal to the width of the entire page. The height of the slice is variable. In the base mode, each

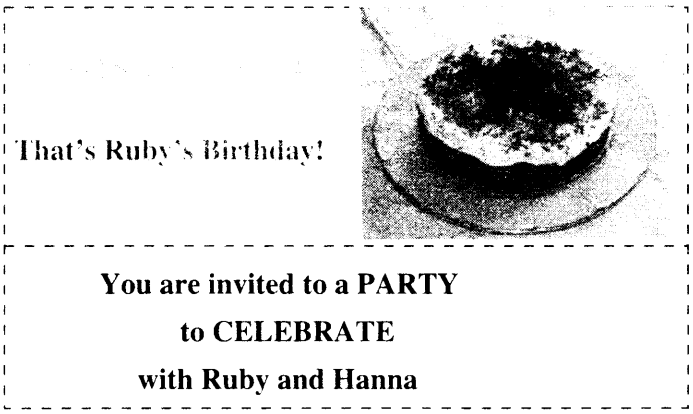

FIGURE 7. 14 Ruby's birthday invitation.

FIGURE 7. 15 The background layer.

slice is represented by three layers: a background layer, a foreground layer, and a mask layer. These layers are used to effectively represent three basic data types: color images (which may be continuous tone or color mapped), bi-level data, and multilevel (multicolor) data. The multilevel image data is put in the background layer, and the mask and foreground layers are used to represent the bi-level and multilevel nonimage data. To work through the various definitions, let us use the document shown in Figure 7.14 as an example. We have divided the document into two slices. The top slice contains the picture of the cake and two lines of writing in two "colors." Notice that the heights of the two slices are not the same and the complexity of the information contained in the two slices is not the same. The top slice contains multicolored text and a continuous tone image whereas the bottom slice contains only bi-level text. Let us take the upper slice first and see how to divide it into the three layers. We will discuss how to code these layers later. The background layer consists of the cake and nothing else. The default color for the background layer is white (though this can be changed). Therefore, we do not need to send the left half of this layer, which contains only white pixels.

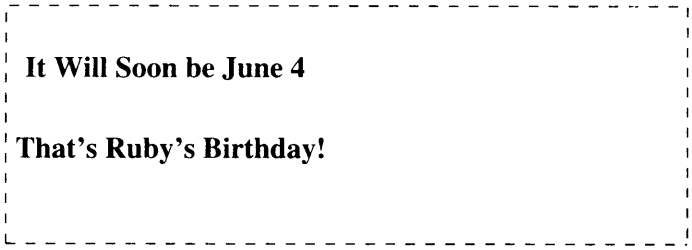

FIGURE 7. 16 The mask layer.

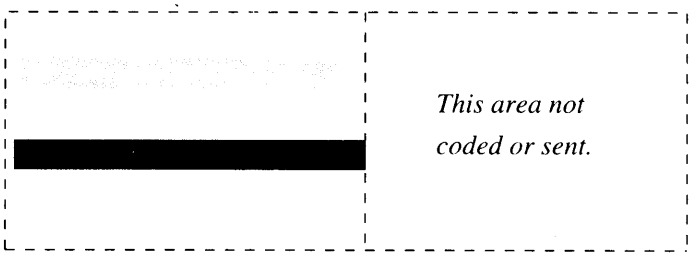

FIGURE 7. 17 The foreground layer.

The mask layer (Figure 7.16) consists of a bi-level representation of the textual information, while the foreground layer contains the colors used in the text. To reassemble the slice we begin with the background layer. We then add to it pixels from the foreground layer using the mask layer as the guide. Wherever the mask layer pixel is black (1) we pick the corresponding pixel from the foreground layer. Wherever the mask pixel is white (0) we use the pixel from the background layer. Because of its role in selecting pixels, the mask layer is also known as the selector layer. During transmission the mask layer is transmitted first, followed by the background and the foreground layers. During the rendering process the background layer is rendered first.

When we look at the lower slice we notice that it contains only bi-level information. In this case we only need the mask layer because the other two layers would be superfluous. In order to deal with this kind of situation, the standard defines three different kinds of stripes. Three-layer stripes (3LS) contain all three layers and is useful when there is both image and textual data in the strip. Two-layer stripes (2LS) only contain two layers, with the third set to a constant value. This kind of stripe would be useful when encoding a stripe with multicolored text and no images, or a stripe with images and bi-level text or line drawings. The third kind of stripe is a one-layer stripe (1LS) which would be used when a stripe contains only bi-level text or line art, or only continuous tone images.

Once the document has been partitioned it can be compressed. Notice that the types of data we have after partitioning are continuous tone images, bi-level information, and multilevel regions. We already have efficient standards for compressing these types of data. For the mask layer containing bi-level information, the recommendation suggests that one of several approaches can be used, including modified Huffman or modified READ

(as described in recomendation T.4), MMR (as described in recommendation T.6) or JBIG (recommendation T.82). The encoder includes information in the datastream about which algorithm has been used. For the continuous tone images and the multilevel regions contained in the foreground and background layers, the recommendation suggests the use of the JPEG standard (recommendation T.81) or the JBIG standard. The header for each slice contains information about which algorithm is used for compression.

7.8 Summary

In this section we have examined a number of ways to compress images. All these approaches exploit the fact that pixels in an image are generally highly correlated with their neighbors. This correlation can be used to predict the actual value of the current pixel. The prediction error can then be encoded and transmitted. Where the correlation is especially high, as in the case of bi-level images, long stretches of pixels can be encoded together using their similarity with previous rows. Finally, by identifying different components of an image that have common characteristics, an image can be partitioned and each partition encoded using the algorithm best suited to it.

Further Reading

1. A detailed survey of lossless image compression techniques can be found in "Lossless Image Compression" by K.P. Subbalakshmi. This chapter appears in the *Lossless Compression Handbook, Academic Press*, 2003.
2. For a detailed description of the LOCO-I and JPEG-LS compression algorithm, see "The LOCO-I Lossless Image Compression Algorithm: Principles and Standardization into JPEG-LS," Hewlett-Packard Laboratories Technical Report HPL-98-193, November 1998 [92].
3. The JBIG and JBIG2 standards are described in a very accessible manner in "Lossless Bilevel Image Compression," by M.W. Hoffman. This chapter appears in the *Lossless Compression Handbook, Academic Press*, 2003.
4. The area of lossless image compression is a very active one, and new schemes are being published all the time. These articles appear in a number of journals, including *Journal of Electronic Imaging*, *Optical Engineering*, *IEEE Transactions on Image Processing*, *IEEE Transactions on Communications*, *Communications of the ACM*, *IEEE Transactions on Computers*, and *Image Communication*, among others.

7.9 Projects and Problems

1. Encode the binary image shown in Figure 7.18 using the modified Huffman scheme.
2. Encode the binary image shown in Figure 7.18 using the modified READ scheme.
3. Encode the binary image shown in Figure 7.18 using the modified modified READ scheme.

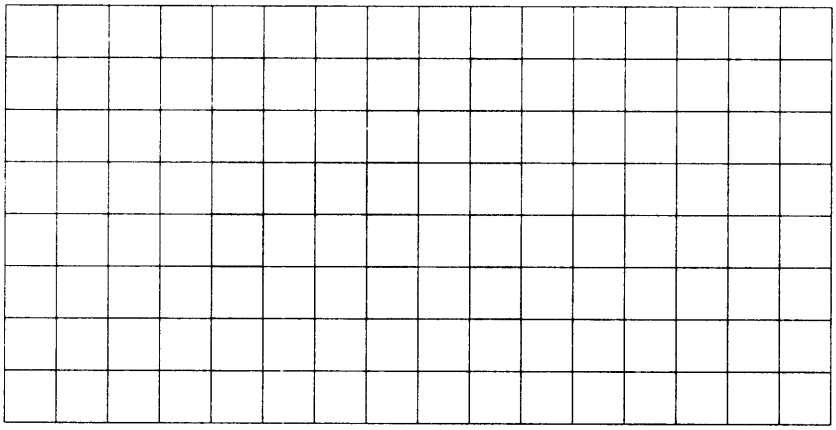

FIGURE 7.18 **An 8 × 16 binary image.**

4. Suppose we want to transmit a 512 × 512, 8-bits-per-pixel image over a 9600 bits per second line.

   (a) If we were to transmit this image using raster scan order, after 15 seconds how many rows of the image will the user have received? To what fraction of the image does this correspond?

   (b) If we were to transmit the image using the method of Example 7.5.1, how long would it take the user to receive the first approximation? How long would it take to receive the first two approximations?

5. An implementation of the progressive transmission example (Example 7.5.1) is included in the programs accompanying this book. The program is called `prog_tran1.c`. Using this program as a template, experiment with different ways of generating approximations (you could use various types of weighted averages) and comment on the qualitative differences (or lack thereof) with using various schemes. Try different block sizes and comment on the practical effects in terms of quality and rate.

6. The program `jpegll_enc.c` generates the residual image for the different JPEG prediction modes, while the program `jpegll_dec.c` reconstructs the original image from the residual image. The output of the encoder program can be used as the input to the public domain arithmetic coding program mentioned in Chapter 4 and the Huffman coding programs mentioned in Chapter 3. Study the performance of different combinations of prediction mode and entropy coder using three images of your choice. Account for any differences you see.

7. Extend `jpegll_enc.c` and `jpegll_dec.c` with an additional prediction mode—be creative! Compare the performance of your predictor with the JPEG predictors.

8. Implement the portions of the CALIC algorithm described in this chapter. Encode the Sena image using your implementation.

Mathematical Preliminaries for Lossy Coding

8.1 Overview

Before we discussed lossless compression, we presented some of the mathematical background necessary for understanding and appreciating the compression schemes that followed. We will try to do the same here for lossy compression schemes. In lossless compression schemes, rate is the general concern. With lossy compression schemes, the loss of information associated with such schemes is also a concern. We will look at different ways of assessing the impact of the loss of information. We will also briefly revisit the subject of information theory, mainly to get an understanding of the part of the theory that deals with the trade-offs involved in reducing the rate, or number of bits per sample, at the expense of the introduction of distortion in the decoded information. This aspect of information theory is also known as rate distortion theory. We will also look at some of the models used in the development of lossy compression schemes.

8.2 Introduction

This chapter will provide some mathematical background that is necessary for discussing lossy compression techniques. Most of the material covered in this chapter is common to many of the compression techniques described in the later chapters. Material that is specific to a particular technique is described in the chapter in which the technique is presented. Some of the material presented in this chapter is not essential for understanding the techniques described in this book. However, to follow some of the literature in this area, familiarity with these topics is necessary. We have marked these sections with a $\star$. If you are primarily interested in the techniques, you may wish to skip these sections, at least on first reading.

On the other hand, if you wish to delve more deeply into these topics, we have included a list of resources at the end of this chapter that provide a more mathematically rigorous treatment of this material.

When we were looking at lossless compression, one thing we never had to worry about was how the reconstructed sequence would differ from the original sequence. By definition, the reconstruction of a losslessly constructed sequence is identical to the original sequence. However, there is only a limited amount of compression that can be obtained with lossless compression. There is a floor (a hard one) defined by the entropy of the source, below which we cannot drive the size of the compressed sequence. As long as we wish to preserve all of the information in the source, the entropy, like the speed of light, is a fundamental limit.

The limited amount of compression available from using lossless compression schemes may be acceptable in several circumstances. The storage or transmission resources available to us may be sufficient to handle our data requirements after lossless compression. Or the possible consequences of a loss of information may be much more expensive than the cost of additional storage and/or transmission resources. This would be the case with the storage and archiving of bank records; an error in the records could turn out to be much more expensive than the cost of buying additional storage media.

If neither of these conditions hold—that is, resources are limited and we do not require absolute integrity—we can improve the amount of compression by accepting a certain degree of loss during the compression process. Performance measures are necessary to determine the efficiency of our *lossy* compression schemes. For the lossless compression schemes we essentially used only the rate as the performance measure. That would not be feasible for lossy compression. If rate were the only criterion for lossy compression schemes, where loss of information is permitted, the best lossy compression scheme would be simply to throw away all the data! Therefore, we need some additional performance measure, such as some measure of the difference between the original and reconstructed data, which we will refer to as the *distortion* in the reconstructed data. In the next section, we will look at some of the more well-known measures of difference and discuss their advantages and shortcomings.

In the best of all possible worlds we would like to incur the minimum amount of distortion while compressing to the lowest rate possible. Obviously, there is a trade-off between minimizing the rate and keeping the distortion small. The extreme cases are when we transmit no information, in which case the rate is zero, or keep all the information, in which case the distortion is zero. The rate for a discrete source is simply the entropy. The study of the situations between these two extremes is called *rate distortion theory*. In this chapter we will take a brief look at some important concepts related to this theory.

Finally, we need to expand the dictionary of models available for our use, for several reasons. First, because we are now able to introduce distortion, we need to determine how to add distortion intelligently. For this, we often need to look at the sources somewhat differently than we have done previously. Another reason is that we will be looking at compression schemes for sources that are analog in nature, even though we have treated them as discrete sources in the past. We need models that more precisely describe the true nature of these sources. We will describe several different models that are widely used in the development of lossy compression algorithms.

We will use the block diagram and notation used in Figure 8.1 throughout our discussions. The output of the source is modeled as a random variable X. The *source coder*

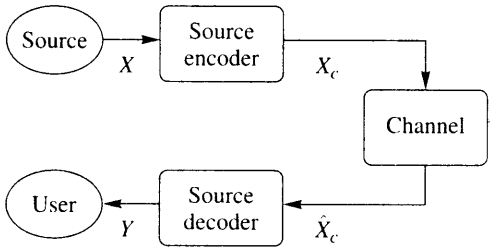

FIGURE 8.1 Block diagram of a generic compression scheme.

takes the source output and produces the compressed representation X_c. The channel block represents all transformations the compressed representation undergoes before the source is reconstructed. Usually, we will take the channel to be the identity mapping, which means $X_c = \hat{X}_c$. The source decoder takes the compressed representation and produces a reconstruction of the source output for the user.

8.3 Distortion Criteria

How do we measure the closeness or fidelity of a reconstructed source sequence to the original? The answer frequently depends on what is being compressed and who is doing the answering. Suppose we were to compress and then reconstruct an image. If the image is a work of art and the resulting reconstruction is to be part of a book on art, the best way to find out how much distortion was introduced and in what manner is to ask a person familiar with the work to look at the image and provide an opinion. If the image is that of a house and is to be used in an advertisement, the best way to evaluate the quality of the reconstruction is probably to ask a real estate agent. However, if the image is from a satellite and is to be processed by a machine to obtain information about the objects in the image, the best measure of fidelity is to see how the introduced distortion affects the functioning of the machine. Similarly, if we were to compress and then reconstruct an audio segment, the judgment of how close the reconstructed sequence is to the original depends on the type of material being examined as well as the manner in which the judging is done. An audiophile is much more likely to perceive distortion in the reconstructed sequence, and distortion is much more likely to be noticed in a musical piece than in a politician's speech.

In the best of all worlds we would always use the end user of a particular source output to assess quality and provide the feedback required for the design. In practice this is not often possible, especially when the end user is a human, because it is difficult to incorporate the human response into mathematical design procedures. Also, there is difficulty in objectively reporting the results. The people asked to assess one person's design may be more easygoing than the people who were asked to assess another person's design. Even though the reconstructed output using one person's design is rated "excellent" and the reconstructed output using the other person's design is only rated "acceptable," switching observers may change the ratings. We could reduce this kind of bias by recruiting a large

number of observers in the hope that the various biases will cancel each other out. This is often the option used, especially in the final stages of the design of compression systems. However, the rather cumbersome nature of this process is limiting. We generally need a more practical method for looking at how close the reconstructed signal is to the original.

A natural thing to do when looking at the fidelity of a reconstructed sequence is to look at the differences between the original and reconstructed values—in other words, the distortion introduced in the compression process. Two popular measures of distortion or difference between the original and reconstructed sequences are the squared error measure and the absolute difference measure. These are called *difference distortion measures*. If $\{x_n\}$ is the source output and $\{y_n\}$ is the reconstructed sequence, then the squared error measure is given by

$$d(x, y) = (x - y)^2 \tag{8.1}$$

and the absolute difference measure is given by

$$d(x, y) = |x - y|. \tag{8.2}$$

In general, it is difficult to examine the difference on a term-by-term basis. Therefore, a number of average measures are used to summarize the information in the difference sequence. The most often used average measure is the average of the squared error measure. This is called the *mean squared error* (mse) and is often represented by the symbol σ^2 or σ_d^2:

$$\sigma^2 = \frac{1}{N}\sum_{n=1}^{N}(x_n - y_n)^2. \tag{8.3}$$

If we are interested in the size of the error relative to the signal, we can find the ratio of the average squared value of the source output and the mse. This is called the *signal-to-noise ratio* (SNR).

$$\text{SNR} = \frac{\sigma_x^2}{\sigma_d^2} \tag{8.4}$$

where σ_x^2 is the average squared value of the source output, or signal, and σ_d^2 is the mse. The SNR is often measured on a logarithmic scale and the units of measurement are *decibels* (abbreviated to dB).

$$\text{SNR(dB)} = 10\log_{10}\frac{\sigma_x^2}{\sigma_d^2} \tag{8.5}$$

Sometimes we are more interested in the size of the error relative to the peak value of the signal x_{peak} than with the size of the error relative to the average squared value of the signal. This ratio is called the *peak-signal-to-noise-ratio* (PSNR) and is given by

$$\text{PSNR(dB)} = 10\log_{10}\frac{x_{\text{peak}}^2}{\sigma_d^2}. \tag{8.6}$$

Another difference distortion measure that is used quite often, although not as often as the mse, is the average of the absolute difference, or

$$d_1 = \frac{1}{N}\sum_{n=1}^{N} |x_n - y_n| . \tag{8.7}$$

This measure seems especially useful for evaluating image compression algorithms.

In some applications, the distortion is not perceptible as long as it is below some threshold. In these situations we might be interested in the maximum value of the error magnitude,

$$d_\infty = \max_n |x_n - y_n| . \tag{8.8}$$

We have looked at two approaches to measuring the fidelity of a reconstruction. The first method involving humans may provide a very accurate measure of perceptible fidelity, but it is not practical and not useful in mathematical design approaches. The second is mathematically tractable, but it usually does not provide a very accurate indication of the perceptible fidelity of the reconstruction. A middle ground is to find a mathematical model for human perception, transform both the source output and the reconstruction to this perceptual space, and then measure the difference in the perceptual space. For example, suppose we could find a transformation $\mathcal{V}$ that represented the actions performed by the human visual system (HVS) on the light intensity impinging on the retina before it is "perceived" by the cortex. We could then find $\mathcal{V}(x)$ and $\mathcal{V}(y)$ and examine the difference between them. There are two problems with this approach. First, the process of human perception is very difficult to model, and accurate models of perception are yet to be discovered. Second, even if we could find a mathematical model for perception, the odds are that it would be so complex that it would be mathematically intractable.

In spite of these disheartening prospects, the study of perception mechanisms is still important from the perspective of design and analysis of compression systems. Even if we cannot obtain a transformation that accurately models perception, we can learn something about the properties of perception that may come in handy in the design of compression systems. In the following, we will look at some of the properties of the human visual system and the perception of sound. Our review will be far from thorough, but the intent here is to present some properties that will be useful in later chapters when we talk about compression of images, video, speech, and audio.

8.3.1 The Human Visual System

The eye is a globe-shaped object with a lens in the front that focuses objects onto the retina in the back of the eye. The retina contains two kinds of receptors, called *rods* and *cones*. The rods are more sensitive to light than cones, and in low light most of our vision is due to the operation of rods. There are three kinds of cones, each of which are most sensitive at different wavelengths of the visible spectrum. The peak sensitivities of the cones are in the red, blue, and green regions of the visible spectrum [93]. The cones are mostly concentrated in a very small area of the retina called the *fovea*. Although the rods are more numerous than the cones, the cones provide better resolution because they are more closely packed in the fovea. The muscles of the eye move the eyeball, positioning the image of the object on

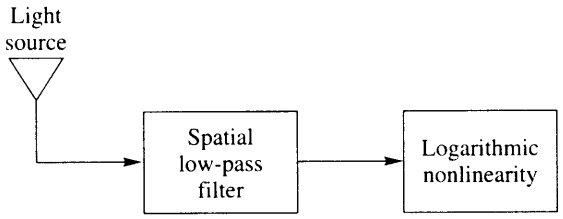

FIGURE 8.2 A model of monochromatic vision.

the fovea. This becomes a drawback in low light. One way to improve what you see in low light is to focus to one side of the object. This way the object is imaged on the rods, which are more sensitive to light.

The eye is sensitive to light over an enormously large range of intensities; the upper end of the range is about 10^{10} times the lower end of the range. However, at a given instant we cannot perceive the entire range of brightness. Instead, the eye adapts to an average brightness level. The range of brightness levels that the eye can perceive at any given instant is much smaller than the total range it is capable of perceiving.

If we illuminate a screen with a certain intensity I and shine a spot on it with different intensity, the spot becomes visible when the difference in intensity is ΔI. This is called the *just noticeable difference* (jnd). The ratio $\frac{\Delta I}{I}$ is known as the *Weber fraction* or *Weber ratio*. This ratio is known to be constant at about 0.02 over a wide range of intensities in the absence of background illumination. However, if the background illumination is changed, the range over which the Weber ratio remains constant becomes relatively small. The constant range is centered around the intensity level to which the eye adapts.

If $\frac{\Delta I}{I}$ is constant, then we can infer that the sensitivity of the eye to intensity is a logarithmic function ($d(\log I) = dI/I$). Thus, we can model the eye as a receptor whose output goes to a logarithmic nonlinearity. We also know that the eye acts as a spatial low-pass filter [94, 95]. Putting all of this information together, we can develop a model for monochromatic vision, shown in Figure 8.2.

How does this description of the human visual system relate to coding schemes? Notice that the mind does not perceive everything the eye sees. We can use this knowledge to design compression systems such that the distortion introduced by our lossy compression scheme is not noticeable.

8.3.2 Auditory Perception

The ear is divided into three parts, creatively named the outer ear, the middle ear, and the inner ear. The outer ear consists of the structure that directs the sound waves, or pressure waves, to the *tympanic membrane*, or eardrum. This membrane separates the outer ear from the middle ear. The middle ear is an air-filled cavity containing three small bones that provide coupling between the tympanic membrane and the *oval window*, which leads into the inner ear. The tympanic membrane and the bones convert the pressure waves in the air to acoustical vibrations. The inner ear contains, among other things, a snail-shaped passage called the *cochlea* that contains the transducers that convert the acoustical vibrations to nerve impulses.

The human ear can hear sounds from approximately 20 Hz to 20 kHz, a 1000:1 range of frequencies. The range decreases with age; older people are usually unable to hear the higher frequencies. As in vision, auditory perception has several nonlinear components. One is that loudness is a function not only of the sound level, but also of the frequency. Thus, for example, a pure 1 kHz tone presented at a 20 dB intensity level will have the same apparent loudness as a 50 Hz tone presented at a 50 dB intensity level. By plotting the amplitude of tones at different frequencies that sound equally loud, we get a series of curves called the *Fletcher-Munson curves* [96].

Another very interesting audio phenomenon is that of *masking*, where one sound blocks out or masks the perception of another sound. The fact that one sound can drown out another seems reasonable. What is not so intuitive about masking is that if we were to try to mask a pure tone with noise, only the noise in a small frequency range around the tone being masked contributes to the masking. This range of frequencies is called the *critical band*. For most frequencies, when the noise just masks the tone, the ratio of the power of the tone divided by the power of the noise in the critical band is a constant [97]. The width of the critical band varies with frequency. This fact has led to the modeling of auditory perception as a bank of band-pass filters. There are a number of other, more complicated masking phenomena that also lend support to this theory (see [97, 98] for more information). The limitations of auditory perception play a major role in the design of audio compression algorithms. We will delve further into these limitations when we discuss audio compression in Chapter 16.

8.4 Information Theory Revisited ★

In order to study the trade-offs between rate and the distortion of lossy compression schemes, we would like to have rate defined explicitly as a function of the distortion for a given distortion measure. Unfortunately, this is generally not possible, and we have to go about it in a more roundabout way. Before we head down this path, we need a few more concepts from information theory.

In Chapter 2, when we talked about information, we were referring to letters from a single alphabet. In the case of lossy compression, we have to deal with two alphabets, the source alphabet and the reconstruction alphabet. These two alphabets are generally different from each other.

Example 8.4.1:

A simple lossy compression approach is to drop a certain number of the least significant bits from the source output. We might use such a scheme between a source that generates monochrome images at 8 bits per pixel and a user whose display facility can display only 64 different shades of gray. We could drop the two least significant bits from each pixel before transmitting the image to the user. There are other methods we can use in this situation that are much more effective, but this is certainly simple.

Suppose our source output consists of 4-bit words $\{0, 1, 2, \ldots, 15\}$. The source encoder encodes each value by shifting out the least significant bit. The output alphabet for the source coder is $\{0, 1, 2, \ldots, 7\}$. At the receiver we cannot recover the original value exactly. However,

we can get an approximation by shifting in a 0 as the least significant bit, or in other words, multiplying the source encoder output by two. Thus, the reconstruction alphabet is $\{0, 2, 4, \ldots, 14\}$, and the source and reconstruction do not take values from the same alphabet. ♦

As the source and reconstruction alphabets can be distinct, we need to be able to talk about the information relationships between two random variables that take on values from two different alphabets.

8.4.1 Conditional Entropy

Let X be a random variable that takes values from the source alphabet $\mathcal{X} = \{x_0, x_1, \ldots, x_{N-1}\}$. Let Y be a random variable that takes on values from the reconstruction alphabet $\mathcal{Y} = \{y_0, y_1, \ldots, y_{M-1}\}$. From Chapter 2 we know that the entropy of the source and the reconstruction are given by

$$H(X) = -\sum_{i=0}^{N-1} P(x_i) \log_2 P(x_i)$$

and

$$H(Y) = -\sum_{j=0}^{M-1} P(y_j) \log_2 P(y_j).$$

A measure of the relationship between two random variables is the *conditional entropy* (the average value of the conditional self-information). Recall that the self-information for an event A was defined as

$$i(A) = \log \frac{1}{P(A)} = -\log P(A).$$

In a similar manner, the conditional self-information of an event A, given that another event B has occurred, can be defined as

$$i(A|B) = \log \frac{1}{P(A|B)} = -\log P(A|B).$$

Suppose B is the event "Frazer has not drunk anything in two days," and A is the event "Frazer is thirsty." Then $P(A|B)$ should be close to one, which means that the conditional self-information $i(A|B)$ would be close to zero. This makes sense from an intuitive point of view as well. If we know that Frazer has not drunk anything in two days, then the statement that Frazer is thirsty would not be at all surprising to us and would contain very little information.

As in the case of self-information, we are generally interested in the average value of the conditional self-information. This average value is called the conditional entropy. The conditional entropies of the source and reconstruction alphabets are given as

$$H(X|Y) = -\sum_{i=0}^{N-1} \sum_{j=0}^{M-1} P(x_i|y_j) P(y_j) \log_2 P(x_i|y_j) \tag{8.9}$$

and

$$H(Y|X) = -\sum_{i=0}^{N-1}\sum_{j=0}^{M-1} P(x_i|y_j)P(y_j)\log_2 P(y_j|x_i). \tag{8.10}$$

The conditional entropy $H(X|Y)$ can be interpreted as the amount of uncertainty remaining about the random variable X, or the source output, given that we know what value the reconstruction Y took. The additional knowledge of Y should reduce the uncertainty about X, and we can show that

$$H(X|Y) \leq H(X) \tag{8.11}$$

(see Problem 5).

Example 8.4.2:

Suppose we have the 4-bits-per-symbol source and compression scheme described in Example 8.4.1. Assume that the source is equally likely to select any letter from its alphabet. Let us calculate the various entropies for this source and compression scheme.

As the source outputs are all equally likely, $P(X = i) = \frac{1}{16}$ for all $i \in \{0, 1, 2, \ldots, 15\}$, and therefore

$$H(X) = -\sum_i \frac{1}{16}\log\frac{1}{16} = \log 16 = 4 \text{ bits}. \tag{8.12}$$

We can calculate the probabilities of the reconstruction alphabet:

$$P(Y = j) = P(X = j) + P(X = j+1) = \frac{1}{16} + \frac{1}{16} = \frac{1}{8}. \tag{8.13}$$

Therefore, $H(Y) = 3$ bits. To calculate the conditional entropy $H(X|Y)$, we need the conditional probabilities $\{P(x_i|y_j)\}$. From our construction of the source encoder, we see that

$$P(X = i|Y = j) = \begin{cases} \frac{1}{2} & \text{if } i = j \text{ or } i = j+1, \text{ for } j = 0, 2, 4, \ldots, 14 \\ 0 & \text{otherwise.} \end{cases} \tag{8.14}$$

Substituting this in the expression for $H(X|Y)$ in Equation (8.9), we get

$$\begin{aligned} H(X|Y) &= -\sum_i\sum_j P(X = i|Y = j)P(Y = j)\log P(X = i|Y = j) \\ &= -\sum_j [P(X = j|Y = j)P(Y = j)\log P(X = j|Y = j) \\ &\quad + P(X = j+1|Y = j)P(Y = j)\log P(X = j+1|Y = j)] \\ &= -8\left[\frac{1}{2}\cdot\frac{1}{8}\log\frac{1}{2} + \frac{1}{2}\cdot\frac{1}{8}\log\frac{1}{2}\right] \qquad (8.15) \\ &= 1. \qquad (8.16) \end{aligned}$$

Let us compare this answer to what we would have intuitively expected the uncertainty to be, based on our knowledge of the compression scheme. With the coding scheme described

here, knowledge of Y means that we know the first 3 bits of the input X. The only thing about the input that we are uncertain about is the value of the last bit. In other words, if we know the value of the reconstruction, our uncertainty about the source output is 1 bit. Therefore, at least in this case, our intuition matches the mathematical definition.

To obtain $H(Y|X)$, we need the conditional probabilities $\{P(y_j|x_i)\}$. From our knowledge of the compression scheme, we see that

$$P(Y=j|X=i)=\begin{cases}1 & \text{if } i=j \text{ or } i=j+1, \text{ for } j=0,2,4,\ldots,14\\ 0 & \text{otherwise.}\end{cases} \tag{8.17}$$

If we substitute these values into Equation (8.10), we get $H(Y|X)=0$ bits (note that $0\log 0=0$). This also makes sense. For the compression scheme described here, if we know the source output, we know 4 bits, the first 3 of which are the reconstruction. Therefore, in this example, knowledge of the source output at a specific time completely specifies the corresponding reconstruction. ♦

8.4.2 Average Mutual Information

We make use of one more quantity that relates the uncertainty or entropy of two random variables. This quantity is called the *mutual information* and is defined as

$$i(x_k;y_j)=\log\left[\frac{P(x_k|y_j)}{P(x_k)}\right]. \tag{8.18}$$

We will use the average value of this quantity, appropriately called the *average mutual information*, which is given by

$$I(X;Y)=\sum_{i=0}^{N-1}\sum_{j=0}^{M-1}P(x_i,y_j)\log\left[\frac{P(x_i|y_j)}{P(x_i)}\right] \tag{8.19}$$

$$=\sum_{i=0}^{N-1}\sum_{j=0}^{M-1}P(x_i|y_j)P(y_j)\log\left[\frac{P(x_i|y_j)}{P(x_i)}\right]. \tag{8.20}$$

We can write the average mutual information in terms of the entropy and the conditional entropy by expanding the argument of the logarithm in Equation (8.20).

$$I(X;Y)=\sum_{i=0}^{N-1}\sum_{j=0}^{M-1}P(x_i,y_j)\log\left[\frac{P(x_i|y_j)}{P(x_i)}\right] \tag{8.21}$$

$$=\sum_{i=0}^{N-1}\sum_{j=0}^{M-1}P(x_i,y_j)\log P(x_i|y_j)-\sum_{i=0}^{N-1}\sum_{j=0}^{M-1}P(x_i,y_j)\log P(x_i) \tag{8.22}$$

$$=H(X)-H(X|Y) \tag{8.23}$$

where the second term in Equation (8.22) is $H(X)$, and the first term is $-H(X|Y)$. Thus, the average mutual information is the entropy of the source minus the uncertainty that remains

about the source output after the reconstructed value has been received. The average mutual information can also be written as

$$I(X; Y) = H(Y) - H(Y|X) = I(Y; X). \tag{8.24}$$

Example 8.4.3:

For the source coder of Example 8.4.2, $H(X) = 4$ bits, and $H(X|Y) = 1$ bit. Therefore, using Equation (8.23), the average mutual information $I(X; Y)$ is 3 bits. If we wish to use Equation (8.24) to compute $I(X; Y)$, we would need $H(Y)$ and $H(Y|X)$, which from Example 8.4.2 are 3 and 0, respectively. Thus, the value of $I(X; Y)$ still works out to be 3 bits. ♦

8.4.3 Differential Entropy

Up to this point we have assumed that the source picks its outputs from a discrete alphabet. When we study lossy compression techniques, we will see that for many sources of interest to us this assumption is not true. In this section, we will extend some of the information theoretic concepts defined for discrete random variables to the case of random variables with continuous distributions.

Unfortunately, we run into trouble from the very beginning. Recall that the first quantity we defined was self-information, which was given by $\log \frac{1}{P(x_i)}$, where $P(x_i)$ is the probability that the random variable will take on the value x_i. For a random variable with a continuous distribution, this probability is zero. Therefore, if the random variable has a continuous distribution, the "self-information" associated with any value is infinity.

If we do not have the concept of self-information, how do we go about defining entropy, which is the average value of the self-information? We know that many continuous functions can be written as limiting cases of their discretized version. We will try to take this route in order to define the entropy of a continuous random variable X with probability density function *(pdf)* $f_X(x)$.

While the random variable X cannot generally take on a particular value with nonzero probability, it can take on a value in an *interval* with nonzero probability. Therefore, let us divide the range of the random variable into intervals of size Δ. Then, by the mean value theorem, in each interval $[(i-1)\Delta, i\Delta)$, there exists a number x_i, such that

$$f_X(x_i)\Delta = \int_{(i-1)\Delta}^{i\Delta} f_X(x)\,dx. \tag{8.25}$$

Let us define a discrete random variable X_d with *pdf*

$$P(X_d = x_i) = f_X(x_i)\Delta. \tag{8.26}$$

Then we can obtain the entropy of this random variable as

$$H(X_d) = -\sum_{i=-\infty}^{\infty} P(x_i)\log P(x_i) \tag{8.27}$$

$$= -\sum_{i=-\infty}^{\infty} f_X(x_i)\Delta \log f_X(x_i)\Delta \tag{8.28}$$

$$= -\sum_{i=-\infty}^{\infty} f_X(x_i)\Delta \log f_X(x_i) - \sum_{i=-\infty}^{\infty} f_X(x_i)\Delta \log \Delta \tag{8.29}$$

$$= -\sum_{i=-\infty}^{\infty} [f_X(x_i) \log f_X(x_i)]\Delta - \log \Delta. \tag{8.30}$$

Taking the limit as $\Delta \to 0$ of Equation (8.30), the first term goes to $-\int_{-\infty}^{\infty} f_X(x) \log f_X(x)\, dx$, which looks like the analog to our definition of entropy for discrete sources. However, the second term is $-\log \Delta$, which goes to plus infinity when Δ goes to zero. It seems there is not an analog to entropy as defined for discrete sources. However, the first term in the limit serves some functions similar to that served by entropy in the discrete case and is a useful function in its own right. We call this term the *differential entropy* of a continuous source and denote it by $h(X)$.

Example 8.4.4:

Suppose we have a random variable X that is uniformly distributed in the interval $[a, b)$. The differential entropy of this random variable is given by

$$h(X) = -\int_{-\infty}^{\infty} f_X(x) \log f_X(x)\, dx \tag{8.31}$$

$$= -\int_a^b \frac{1}{b-a} \log \frac{1}{b-a}\, dx \tag{8.32}$$

$$= \log(b-a). \tag{8.33}$$

Notice that when $b-a$ is less than one, the differential entropy will become negative—in contrast to the entropy, which never takes on negative values. ♦

Later in this chapter, we will find particular use for the differential entropy of the Gaussian source.

Example 8.4.5:

Suppose we have a random variable X that has a Gaussian *pdf*,

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp -\frac{(x-\mu)^2}{2\sigma^2}. \tag{8.34}$$

The differential entropy is given by

$$h(X) = -\int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} \exp -\frac{(x-\mu)^2}{2\sigma^2} \log \left[\frac{1}{\sqrt{2\pi\sigma^2}} \exp -\frac{(x-\mu)^2}{2\sigma^2} \right] dx \tag{8.35}$$

$$= -\log \frac{1}{\sqrt{2\pi\sigma^2}} \int_{-\infty}^{\infty} f_X(x)dx + \int_{-\infty}^{\infty} \frac{(x-\mu)^2}{2\sigma^2} \log e f_X(x)dx \tag{8.36}$$

$$= \frac{1}{2}\log 2\pi\sigma^2 + \frac{1}{2}\log e \tag{8.37}$$

$$= \frac{1}{2}\log 2\pi e\sigma^2. \tag{8.38}$$

Thus, the differential entropy of a Gaussian random variable is directly proportional to its variance. ♦

The differential entropy for the Gaussian distribution has the added distinction that it is larger than the differential entropy for any other continuously distributed random variable with the same variance. That is, for any random variable X, with variance σ^2

$$h(X) \leq \frac{1}{2}\log 2\pi e\sigma^2. \tag{8.39}$$

The proof of this statement depends on the fact that for any two continuous distributions $f_X(X)$ and $g_X(X)$

$$-\int_{-\infty}^{\infty} f_X(x)\log f_X(x)dx \leq -\int_{-\infty}^{\infty} f_X(x)\log g_X(x)dx. \tag{8.40}$$

We will not prove Equation (8.40) here, but you may refer to [99] for a simple proof. To obtain Equation (8.39), we substitute the expression for the Gaussian distribution for $g_X(x)$. Noting that the left-hand side of Equation (8.40) is simply the differential entropy of the random variable X, we have

$$\begin{aligned} h(X) &\leq -\int_{-\infty}^{\infty} f_X(x)\log\frac{1}{\sqrt{2\pi\sigma^2}}\exp-\frac{(x-\mu)^2}{2\sigma^2}dx \\ &= \frac{1}{2}\log(2\pi\sigma^2) + \log e\int_{-\infty}^{\infty} f_X(x)\frac{(x-\mu)^2}{2\sigma^2}dx \\ &= \frac{1}{2}\log(2\pi\sigma^2) + \frac{\log e}{2\sigma^2}\int_{-\infty}^{\infty} f_X(x)(x-\mu)^2dx \\ &= \frac{1}{2}\log(2\pi e\sigma^2). \end{aligned} \tag{8.41}$$

We seem to be striking out with continuous random variables. There is no analog for self-information and really none for entropy either. However, the situation improves when we look for an analog for the average mutual information. Let us define the random variable Y_d in a manner similar to the random variable X_d, as the discretized version of a continuous valued random variable Y. Then we can show (see Problem 4)

$$H(X_d|Y_d) = -\sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty}\left[f_{X|Y}(x_i|y_j)f_Y(y_j)\log f_{X|Y}(x_i|y_j)\right]\Delta\Delta - \log\Delta. \tag{8.42}$$

Therefore, the average mutual information for the discretized random variables is given by

$$I(X_d; Y_d) = H(X_d) - H(X_d|Y_d) \tag{8.43}$$

$$= -\sum_{i=-\infty}^{\infty} f_X(x_i)\Delta \log f_X(x_i) \tag{8.44}$$

$$-\sum_{i=-\infty}^{\infty}\left[\sum_{j=-\infty}^{\infty} f_{X|Y}(x_i|y_j)f_Y(y_j)\log f_{X|Y}(x_i|y_j)\Delta\right]\Delta. \tag{8.45}$$

Notice that the two $\log\Delta$s in the expression for $H(X_d)$ and $H(X_d|Y_d)$ cancel each other out, and as long as $h(X)$ and $h(X|Y)$ are not equal to infinity, when we take the limit as $\Delta \to 0$ of $I(X_d; Y_d)$ we get

$$I(X; Y) = h(X) - h(X|Y). \tag{8.46}$$

The average mutual information in the continuous case can be obtained as a limiting case of the average mutual information for the discrete case and has the same physical significance.

We have gone through a lot of mathematics in this section. But the information will be used immediately to define the rate distortion function for a random source.

8.5 Rate Distortion Theory ★

Rate distortion theory is concerned with the trade-offs between distortion and rate in lossy compression schemes. Rate is defined as the average number of bits used to represent each sample value. One way of representing the trade-offs is via a *rate distortion function R(D)*. The rate distortion function $R(D)$ specifies the lowest rate at which the output of a source can be encoded while keeping the distortion less than or equal to D. On our way to mathematically defining the rate distortion function, let us look at the rate and distortion for some different lossy compression schemes.

In Example 8.4.2, knowledge of the value of the input at time k completely specifies the reconstructed value at time k. In this situation,

$$P(y_j|x_i) = \begin{cases} 1 & \text{for some } j = j_i \\ 0 & \text{otherwise.} \end{cases} \tag{8.47}$$

Therefore,

$$D = \sum_{i=0}^{N-1}\sum_{j=0}^{M-1} P(y_j|x_i)P(x_i)d(x_i, y_j) \tag{8.48}$$

$$= \sum_{i=0}^{N-1} P(x_i)d(x_i, y_{j_i}) \tag{8.49}$$

where we used the fact that $P(x_i, y_j) = P(y_j|x_i)P(x_i)$ in Equation (8.48). The rate for this source coder is the output entropy $H(Y)$ of the source decoder. If this were always the case, the task of obtaining a rate distortion function would be relatively simple. Given a

distortion constraint D^*, we could look at all encoders with distortion less than D^* and pick the one with the lowest output entropy. This entropy would be the rate corresponding to the distortion D^*. However, the requirement that knowledge of the input at time k completely specifies the reconstruction at time k is very restrictive, and there are many efficient compression techniques that would have to be excluded under this requirement. Consider the following example.

Example 8.5.1:

With a data sequence that consists of height and weight measurements, obviously height and weight are quite heavily correlated. In fact, after studying a long sequence of data, we find that if we plot the height along the x axis and the weight along the y axis, the data points cluster along the line $y = 2.5x$. In order to take advantage of this correlation, we devise the following compression scheme. For a given pair of height and weight measurements, we find the orthogonal projection on the $y = 2.5x$ line as shown in Figure 8.3. The point on this line can be represented as the distance to the nearest integer from the origin. Thus, we encode a pair of values into a single value. At the time of reconstruction, we simply map this value back into a pair of height and weight measurements.

For instance, suppose somebody is 72 inches tall and weighs 200 pounds (point A in Figure 8.3). This corresponds to a point at a distance of 212 along the $y = 2.5x$ line. The reconstructed values of the height and weight corresponding to this value are 79 and 197. Notice that the reconstructed values differ from the original values. Suppose we now have

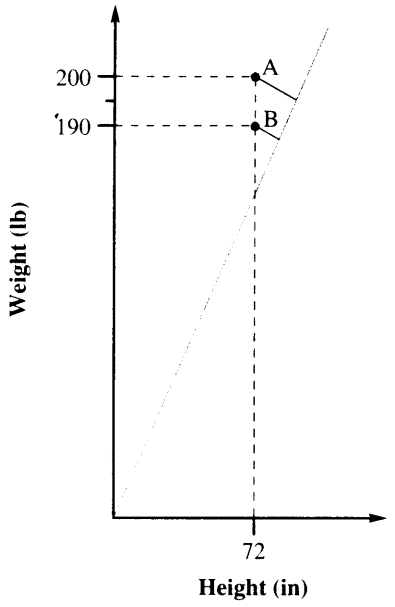

FIGURE 8.3 Compression scheme for encoding height-weight pairs.

another individual who is also 72 inches tall but weighs 190 pounds (point B in Figure 8.3). The source coder output for this pair would be 203, and the reconstructed values for height and weight are 75 and 188, respectively. Notice that while the height value in both cases was the same, the reconstructed value is different. The reason for this is that the reconstructed value for the height depends on the weight. Thus, for this particular source coder, we do not have a conditional probability density function $\{P(y_j|x_i)\}$ of the form shown in Equation (8.47). ♦

Let us examine the distortion for this scheme a little more closely. As the conditional probability for this scheme is not of the form of Equation (8.47), we can no longer write the distortion in the form of Equation (8.49). Recall that the general form of the distortion is

$$D = \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} d(x_i, y_j) P(x_i) P(y_j|x_i). \tag{8.50}$$

Each term in the summation consists of three factors: the distortion measure $d(x_i, y_j)$, the source density $P(x_i)$, and the conditional probability $P(y_j|x_i)$. The distortion measure is a measure of closeness of the original and reconstructed versions of the signal and is generally determined by the particular application. The source probabilities are solely determined by the source. The third factor, the set of conditional probabilities, can be seen as a description of the compression scheme.

Therefore, for a given source with some *pdf* $\{P(x_i)\}$ and a specified distortion measure $d(\cdot, \cdot)$, the distortion is a function only of the conditional probabilities $\{P(y_j|x_i)\}$; that is,

$$D = D(\{P(y_j|x_i)\}). \tag{8.51}$$

Therefore, we can write the constraint that the distortion D be less than some value D^* as a requirement that the conditional probabilities for the compression scheme belong to a set of conditional probabilities Γ that have the property that

$$\Gamma = \{\{P(y_j|x_i\} \text{ such that } D(\{P(y_j|x_i)\}) \leq D^*\}. \tag{8.52}$$

Once we know the set of compression schemes to which we have to confine ourselves, we can start to look at the rate of these schemes. In Example 8.4.2, the rate was the entropy of Y. However, that was a result of the fact that the conditional probability describing that particular source coder took on only the values 0 and 1. Consider the following trivial situation.

Example 8.5.2:

Suppose we have the same source as in Example 8.4.2 and the same reconstruction alphabet. Suppose the distortion measure is

$$d(x_i, y_j) = (x_i - y_j)^2$$

and $D^* = 225$. One compression scheme that satisfies the distortion constraint randomly maps the input to any one of the outputs; that is,

$$P(y_j|x_i) = \frac{1}{8} \qquad \text{for } i = 0, 1, \ldots, 15 \text{ and } j = 0, 2, \ldots, 14.$$

We can see that this conditional probability assignment satisfies the distortion constraint. As each of the eight reconstruction values is equally likely, $H(Y)$ is 3 bits. However, we are not transmitting *any* information. We could get exactly the same results by transmitting 0 bits and randomly picking Y at the receiver. ♦

Therefore, the entropy of the reconstruction $H(Y)$ cannot be a measure of the rate. In his 1959 paper on source coding [100], Shannon showed that the minimum rate for a given distortion is given by

$$R(D) = \min_{\{P(y_j|x_i)\} \in \Gamma} I(X; Y). \tag{8.53}$$

To prove this is beyond the scope of this book. (Further information can be found in [3] and [4].) However, we can at least convince ourselves that defining the rate as an average mutual information gives sensible answers when used for the examples shown here. Consider Example 8.4.2. The average mutual information in this case is 3 bits, which is what we said the rate was. In fact, notice that whenever the conditional probabilities are constrained to be of the form of Equation (8.47),

$$H(Y|X) = 0,$$

then

$$I(X; Y) = H(Y),$$

which had been our measure of rate.

In Example 8.5.2, the average mutual information is 0 bits, which accords with our intuitive feeling of what the rate should be. Again, whenever

$$H(Y|X) = H(Y),$$

that is, knowledge of the source gives us no knowledge of the reconstruction,

$$I(X; Y) = 0,$$

which seems entirely reasonable. We should not have to transmit any bits when we are not sending any information.

At least for the examples here, it seems that the average mutual information does represent the rate. However, earlier we had said that the average mutual information between the source output and the reconstruction is a measure of the information conveyed by the reconstruction about the source output. Why are we then looking for compression schemes that *minimize* this value? To understand this, we have to remember that the process of finding the performance of the optimum compression scheme had two parts. In the first part we

specified the desired distortion. The entire set of conditional probabilities over which the average mutual information is minimized satisfies the distortion constraint. Therefore, we can leave the question of distortion, or fidelity, aside and concentrate on minimizing the rate.

Finally, how do we find the rate distortion function? There are two ways: one is a computational approach developed by Arimoto [101] and Blahut [102]. While the derivation of the algorithm is beyond the scope of this book, the algorithm itself is relatively simple. The other approach is to find a lower bound for the average mutual information and then show that we can achieve this bound. We use this approach to find the rate distortion functions for two important sources.

Example 8.5.3: Rate distortion function for the binary source

Suppose we have a source alphabet $\{0, 1\}$, with $P(0) = p$. The reconstruction alphabet is also binary. Given the distortion measure

$$d(x_i, y_j) = x_i \oplus y_j, \tag{8.54}$$

where $\oplus$ is modulo 2 addition, let us find the rate distortion function. Assume for the moment that $p < \frac{1}{2}$. For $D > p$ an encoding scheme that would satisfy the distortion criterion would be not to transmit anything and fix $Y = 1$. So for $D \geq p$

$$R(D) = 0. \tag{8.55}$$

We will find the rate distortion function for the distortion range $0 \leq D < p$.

Find a lower bound for the average mutual information:

$$\begin{aligned} I(X; Y) &= H(X) - H(X|Y) && (8.56)\\ &= H(X) - H(X \oplus Y|Y) && (8.57)\\ &\geq H(X) - H(X \oplus Y) \qquad \text{from Equation (8.11).} && (8.58) \end{aligned}$$

In the second step we have used the fact that if we know Y, then knowing X we can obtain $X \oplus Y$ and vice versa as $X \oplus Y \oplus Y = X$.

Let us look at the terms on the right-hand side of (8.11):

$$H(X) = -p \log_2 p - (1 - p) \log_2 (1 - p) = H_b(p), \tag{8.59}$$

where $H_b(p)$ is called the *binary entropy function* and is plotted in Figure 8.4. Note that $H_b(p) = H_b(1 - p)$.

Given that $H(X)$ is completely specified by the source probabilities, our task now is to find the conditional probabilities $\{P(x_i|y_j)\}$ such that $H(X \oplus Y)$ is maximized while the average distortion $E[d(x_i, y_j)] \leq D$. $H(X \oplus Y)$ is simply the binary entropy function $H_b(P(X \oplus Y = 1))$, where

$$P(X \oplus Y = 1) = P(X = 0, Y = 1) + P(X = 1, Y = 0). \tag{8.60}$$

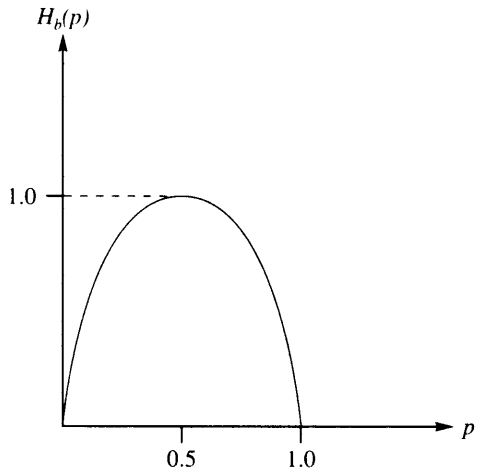

FIGURE 8.4 The binary entropy function.

Therefore, to maximize $H(X \oplus Y)$, we would want $P(X \oplus Y = 1)$ to be as close as possible to one-half. However, the selection of $P(X \oplus Y)$ also has to satisfy the distortion constraint. The distortion is given by

$$\begin{aligned} E[d(x_i, y_j)] &= 0 \times P(X = 0, Y = 0) + 1 \times P(X = 0, Y = 1) \\ &\quad + 1 \times P(X = 1, Y = 0) + 0 \times P(X = 1, Y = 1) \\ &= P(X = 0, Y = 1) + P(X = 1, Y = 0) \\ &= P(Y = 1|X = 0)p + P(Y = 0|X = 1)(1 - p). \end{aligned} \tag{8.61}$$

But this is simply the probability that $X \oplus Y = 1$. Therefore, the maximum value that $P(X \oplus Y = 1)$ can have is D. Our assumptions were that $D < p$ and $p \leq \frac{1}{2}$, which means that $D < \frac{1}{2}$. Therefore, $P(X \oplus Y = 1)$ is closest to $\frac{1}{2}$ while being less than or equal to D when $P(X \oplus Y = 1) = D$. Therefore,

$$I(X; Y) \geq H_b(p) - H_b(D). \tag{8.62}$$

We can show that for $P(X = 0|Y = 1) = P(X = 1|Y = 0) = D$, this bound is achieved. That is, if $P(X = 0|Y = 1) = P(X = 1|Y = 0) = D$, then

$$I(X; Y) = H_b(p) - H_b(D). \tag{8.63}$$

Therefore, for $D < p$ and $p \leq \frac{1}{2}$,

$$R(D) = H_b(p) - H_b(D). \tag{8.64}$$

Finally, if $p > \frac{1}{2}$, then we simply switch the roles of p and $1-p$. Putting all this together, the rate distortion function for a binary source is

$$R(D) = \begin{cases} H_b(p) - H_b(D) & \text{for } D < \min\{p, 1-p\} \\ 0 & \text{otherwise.} \end{cases} \tag{8.65}$$

♦

Example 8.5.4: Rate distortion function for the Gaussian source

Suppose we have a continuous amplitude source that has a zero mean Gaussian *pdf* with variance σ^2. If our distortion measure is given by

$$d(x, y) = (x - y)^2, \tag{8.66}$$

our distortion constraint is given by

$$E\left[(X - Y)^2\right] \leq D. \tag{8.67}$$

Our approach to finding the rate distortion function will be the same as in the previous example; that is, find a lower bound for $I(X; Y)$ given a distortion constraint, and then show that this lower bound can be achieved.

First we find the rate distortion function for $D < \sigma^2$.

$$\begin{aligned} I(X; Y) &= h(X) - h(X|Y) && (8.68) \\ &= h(X) - h(X - Y|Y) && (8.69) \\ &\geq h(X) - h(X - Y) && (8.70) \end{aligned}$$

In order to minimize the right-hand side of Equation (8.70), we have to maximize the second term subject to the constraint given by Equation (8.67). This term is maximized if $X - Y$ is Gaussian, and the constraint can be satisfied if $E\left[(X - Y)^2\right] = D$. Therefore, $h(X - Y)$ is the differential entropy of a Gaussian random variable with variance D, and the lower bound becomes

$$\begin{aligned} I(X; Y) &\geq \frac{1}{2}\log(2\pi e\sigma^2) - \frac{1}{2}\log(2\pi eD) && (8.71) \\ &= \frac{1}{2}\log\frac{\sigma^2}{D}. && (8.72) \end{aligned}$$

This average mutual information can be achieved if Y is zero mean Gaussian with variance $\sigma^2 - D$, and

$$f_{X|Y}(x|y) = \frac{1}{\sqrt{2\pi D}} \exp\frac{-x^2}{2D}. \tag{8.73}$$

For $D > \sigma^2$, if we set $Y = 0$, then

$$I(X; Y) = 0 \tag{8.74}$$

and

$$E\left[(X-Y)^2\right] = \sigma^2 < D. \tag{8.75}$$

Therefore, the rate distortion function for the Gaussian source can be written as

$$R(D) = \begin{cases} \frac{1}{2}\log\frac{\sigma^2}{D} & \text{for } D < \sigma^2 \\ 0 & \text{for } D > \sigma^2. \end{cases} \tag{8.76}$$

♦

Like the differential entropy for the Gaussian source, the rate distortion function for the Gaussian source also has the distinction of being larger than the rate distortion function for any other source with a continuous distribution and the same variance. This is especially valuable because for many sources it can be very difficult to calculate the rate distortion function. In these situations, it is helpful to have an upper bound for the rate distortion function. It would be very nice if we also had a lower bound for the rate distortion function of a continuous random variable. Shannon described such a bound in his 1948 paper [7], and it is appropriately called the *Shannon lower bound.* We will simply state the bound here without derivation (for more information, see [4]).

The Shannon lower bound for a random variable X and the magnitude error criterion

$$d(x, y) = |x - y| \tag{8.77}$$

is given by

$$R_{SLB}(D) = h(X) - \log(2eD). \tag{8.78}$$

If we used the squared error criterion, the Shannon lower bound is given by

$$R_{SLB}(D) = h(X) - \frac{1}{2}\log(2\pi eD). \tag{8.79}$$

In this section we have defined the rate distortion function and obtained the rate distortion function for two important sources. We have also obtained upper and lower bounds on the rate distortion function for an arbitrary *iid* source. These functions and bounds are especially useful when we want to know if it is possible to design compression schemes to provide a specified rate and distortion given a particular source. They are also useful in determining the amount of performance improvement that we could obtain by designing a better compression scheme. In these ways the rate distortion function plays the same role for lossy compression that entropy plays for lossless compression.

8.6 Models

As in the case of lossless compression, models play an important role in the design of lossy compression algorithms; there are a variety of approaches available. The set of models we can draw on for lossy compression is much wider than the set of models we studied for

lossless compression. We will look at some of these models in this section. What is presented here is by no means an exhaustive list of models. Our only intent is to describe those models that will be useful in the following chapters.

8.6.1 Probability Models

An important method for characterizing a particular source is through the use of probability models. As we shall see later, knowledge of the probability model is important for the design of a number of compression schemes.

Probability models used for the design and analysis of lossy compression schemes differ from those used in the design and analysis of lossless compression schemes. When developing models in the lossless case, we tried for an exact match. The probability of each symbol was estimated as part of the modeling process. When modeling sources in order to design or analyze lossy compression schemes, we look more to the general rather than exact correspondence. The reasons are more pragmatic than theoretical. Certain probability distribution functions are more analytically tractable than others, and we try to match the distribution of the source with one of these "nice" distributions.

Uniform, Gaussian, Laplacian, and Gamma distribution are four probability models commonly used in the design and analysis of lossy compression systems:

- **Uniform Distribution:** As for lossless compression, this is again our ignorance model. If we do not know anything about the distribution of the source output, except possibly the range of values, we can use the uniform distribution to model the source. The probability density function for a random variable uniformly distributed between a and b is

$$f_X(x) = \begin{cases} \frac{1}{b-a} & \text{for } a \le x \le b \\ 0 & \text{otherwise.} \end{cases} \tag{8.80}$$

- **Gaussian Distribution:** The Gaussian distribution is one of the most commonly used probability models for two reasons: it is mathematically tractable and, by virtue of the central limit theorem, it can be argued that in the limit the distribution of interest goes to a Gaussian distribution. The probability density function for a random variable with a Gaussian distribution and mean μ and variance σ^2 is

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp - \frac{(x-\mu)^2}{2\sigma^2}. \tag{8.81}$$

- **Laplacian Distribution:** Many sources that we deal with have distributions that are quite peaked at zero. For example, speech consists mainly of silence. Therefore, samples of speech will be zero or close to zero with high probability. Image pixels themselves do not have any attraction to small values. However, there is a high degree of correlation among pixels. Therefore, a large number of the pixel-to-pixel differences will have values close to zero. In these situations, a Gaussian distribution is not a very close match to the data. A closer match is the Laplacian distribution, which is peaked

at zero. The distribution function for a zero mean random variable with Laplacian distribution and variance σ^2 is

$$f_X(x) = \frac{1}{\sqrt{2\sigma^2}} \exp \frac{-\sqrt{2}|x|}{\sigma}. \tag{8.82}$$

- **Gamma Distribution:** A distribution that is even more peaked, though considerably less tractable, than the Laplacian distribution is the Gamma distribution. The distribution function for a Gamma distributed random variable with zero mean and variance σ^2 is given by

$$f_X(x) = \frac{\sqrt[4]{3}}{\sqrt{8\pi\sigma|x|}} \exp \frac{-\sqrt{3}|x|}{2\sigma}. \tag{8.83}$$

The shapes of these four distributions, assuming a mean of zero and a variance of one, are shown in Figure 8.5.

One way of obtaining the estimate of the distribution of a particular source is to divide the range of outputs into "bins" or intervals I_k. We can then find the number of values n_k that fall into each interval. A plot of $\frac{n_k}{n_T}$, where n_T is the total number of source outputs being considered, should give us some idea of what the input distribution looks like. Be aware that this is a rather crude method and can at times be misleading. For example, if we were not careful in our selection of the source output, we might end up modeling some local peculiarities of the source. If the bins are too large, we might effectively filter out some important properties of the source. If the bin sizes are too small, we may miss out on some of the gross behavior of the source.

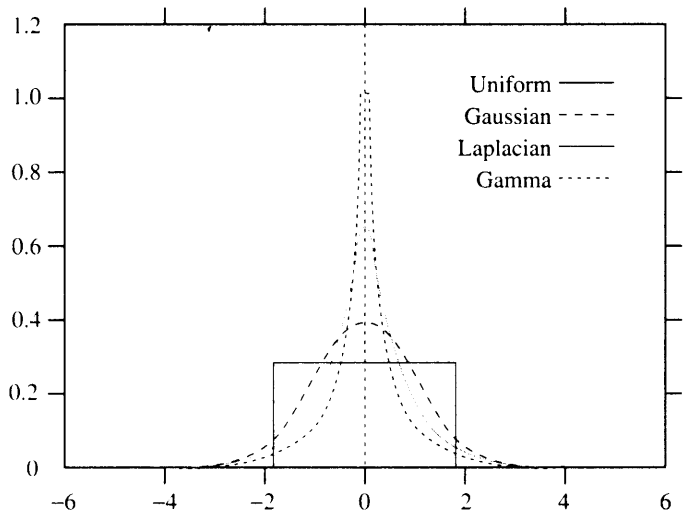

FIGURE 8.5 Uniform, Gaussian, Laplacian, and Gamma distributions.

Once we have decided on some candidate distributions, we can select between them using a number of sophisticated tests. These tests are beyond the scope of this book but are described in [103].

Many of the sources that we deal with when we design lossy compression schemes have a great deal of structure in the form of sample-to-sample dependencies. The probability models described here capture none of these dependencies. Fortunately, we have a lot of models that can capture most of this structure. We describe some of these models in the next section.

8.6.2 Linear System Models

A large class of processes can be modeled in the form of the following difference equation:

$$x_n = \sum_{i=1}^{N} a_i x_{n-i} + \sum_{j=1}^{M} b_j \epsilon_{n-j} + \epsilon_n, \tag{8.84}$$

where $\{x_n\}$ are samples of the process we wish to model, and $\{\epsilon_n\}$ is a white noise sequence. We will assume throughout this book that we are dealing with real valued samples. Recall that a zero-mean wide-sense-stationary noise sequence $\{\epsilon_n\}$ is a sequence with autocorrelation function

$$R_{\epsilon\epsilon}(k) = \begin{cases} \sigma_\epsilon^2 & \text{for } k = 0 \\ 0 & \text{otherwise.} \end{cases} \tag{8.85}$$

In digital signal-processing terminology, Equation (8.84) represents the output of a linear discrete time invariant filter with N poles and M zeros. In the statistical literature, this model is called an autoregressive moving average model of order (N,M), or an ARMA (N,M) model. The autoregressive label is because of the first summation in Equation (8.84), while the second summation gives us the moving average portion of the name.

If all the b_j were zero in Equation (8.84), only the autoregressive part of the ARMA model would remain:

$$x_n = \sum_{i=1}^{N} a_i x_{n-i} + \epsilon_n. \tag{8.86}$$

This model is called an Nth-order autoregressive model and is denoted by AR(N). In digital signal-processing terminology, this is an *all pole filter*. The AR(N) model is the most popular of all the linear models, especially in speech compression, where it arises as a natural consequence of the speech production model. We will look at it a bit more closely.

First notice that for the AR(N) process, knowing all the past history of the process gives no more information than knowing the last N samples of the process; that is,

$$P(x_n|x_{n-1}, x_{n-2}, \ldots) = P(x_n|x_{n-1}, x_{n-2}, \ldots, x_{n-N}), \tag{8.87}$$

which means that the AR(N) process is a Markov model of order N.

The autocorrelation function of a process can tell us a lot about the sample-to-sample behavior of a sequence. A slowly decaying autocorrelation function indicates a high sample-to-sample correlation, while a fast decaying autocorrelation denotes low sample-to-sample

correlation. In the case of *no* sample-to-sample correlation, such as white noise, the autocorrelation function is zero for lags greater than zero, as seen in Equation (8.85). The autocorrelation function for the AR(N) process can be obtained as follows:

$$R_{xx}(k) = E[x_n x_{n-k}] \tag{8.88}$$

$$= E\left[\left(\sum_{i=1}^{N} a_i x_{n-i} + \epsilon_n\right)(x_{n-k})\right] \tag{8.89}$$

$$= E\left[\sum_{i=1}^{N} a_i x_{n-i} x_{n-k}\right] + E[\epsilon_n x_{n-k}] \tag{8.90}$$

$$= \begin{cases} \sum_{i=1}^{N} a_i R_{xx}(k-i) & \text{for } k > 0 \\ \sum_{i=1}^{N} a_i R_{xx}(i) + \sigma_\epsilon^2 & \text{for } k = 0. \end{cases} \tag{8.91}$$

Example 8.6.1:

Suppose we have an AR(3) process. Let us write out the equations for the autocorrelation coefficient for lags 1, 2, 3:

$$R_{xx}(1) = a_1 R_{xx}(0) + a_2 R_{xx}(1) + a_3 R_{xx}(2)$$
$$R_{xx}(2) = a_1 R_{xx}(1) + a_2 R_{xx}(0) + a_3 R_{xx}(1)$$
$$R_{xx}(3) = a_1 R_{xx}(2) + a_2 R_{xx}(1) + a_3 R_{xx}(0).$$

If we know the values of the autocorrelation function $R_{xx}(k)$, for $k = 0, 1, 2, 3$, we can use this set of equations to find the AR(3) coefficients $\{a_1, a_2, a_3\}$. On the other hand, if we know the model coefficients and σ_ϵ^2, we can use the above equations along with the equation for $R_{xx}(0)$ to find the first four autocorrelation coefficients. All the other autocorrelation values can be obtained by using Equation (8.91). ♦

To see how the autocorrelation function is related to the temporal behavior of the sequence, let us look at the behavior of a simple AR(1) source.

Example 8.6.2:

An AR(1) source is defined by the equation

$$x_n = a_1 x_{n-1} + \epsilon_n. \tag{8.92}$$

The autocorrelation function for this source (see Problem 8) is given by

$$R_{xx}(k) = \frac{1}{1 - a_1^2} a_1^k \sigma_\epsilon^2. \tag{8.93}$$

From this we can see that the autocorrelation will decay more slowly for larger values of a_1. Remember that the value of a_1 in this case is an indicator of how closely the current

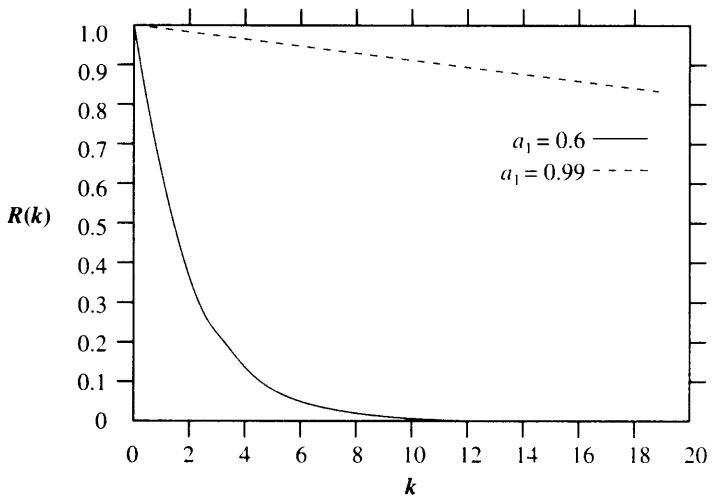

FIGURE 8.6 Autocorrelation function of an AR(1) process with two values of a_1.

sample is related to the previous sample. The autocorrelation function is plotted for two values of a_1 in Figure 8.6. Notice that for a_1 close to 1, the autocorrelation function decays extremely slowly. As the value of a_1 moves farther away from 1, the autocorrelation function decays much faster.

Sample waveforms for $a_1 = 0.99$ and $a_1 = 0.6$ are shown in Figures 8.7 and 8.8. Notice the slower variations in the waveform for the process with a higher value of a_1. Because

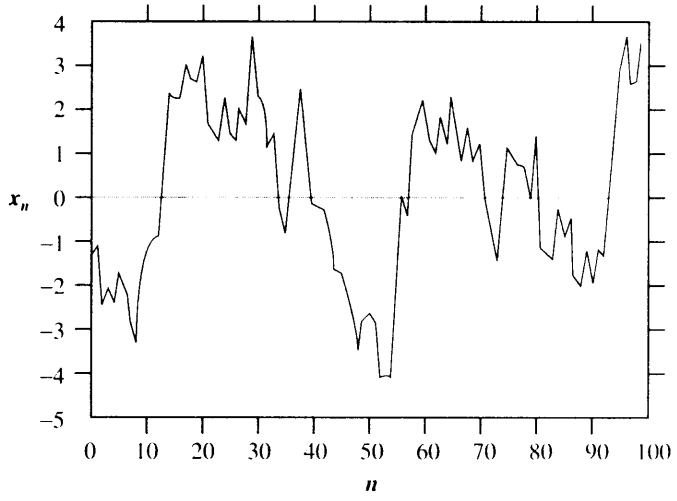

FIGURE 8.7 Sample function of an AR(1) process with $a_1 = 0.99$.

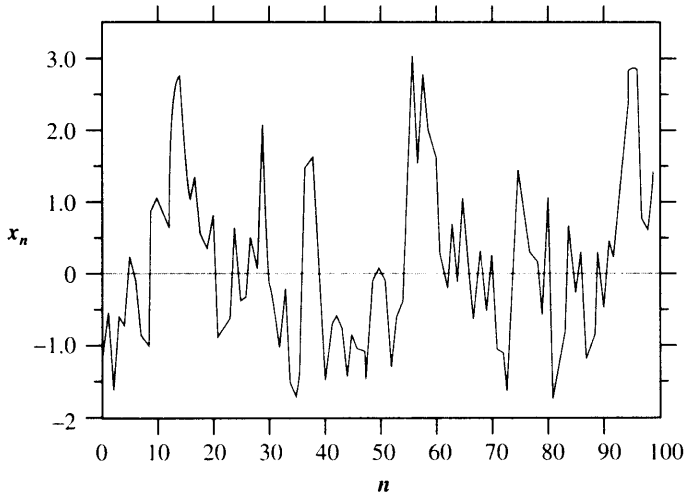

FIGURE 8.8 Sample function of an AR(1) process with $a_1 = 0.6$.

the waveform in Figure 8.7 varies more slowly than the waveform in Figure 8.8, samples of this waveform are much more likely to be close in value than the samples of the waveform of Figure 8.8.

Let's look at what happens when the AR(1) coefficient is negative. The sample waveforms are plotted in Figures 8.9 and 8.10. The sample-to-sample variation in these waveforms

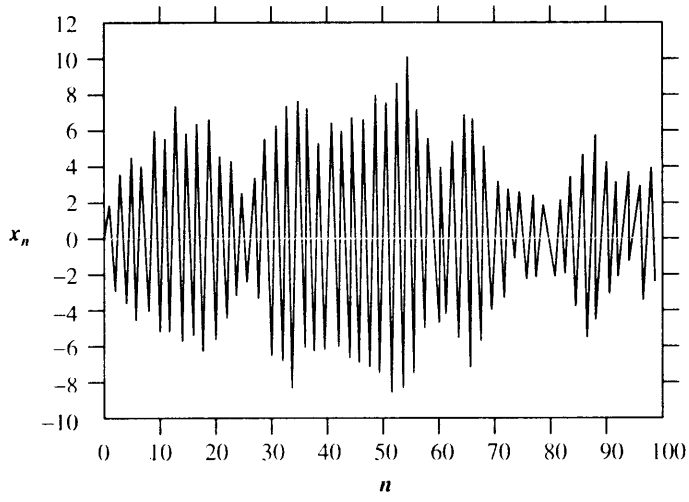

FIGURE 8.9 Sample function of an AR(1) process with $a_1 = -0.99$.

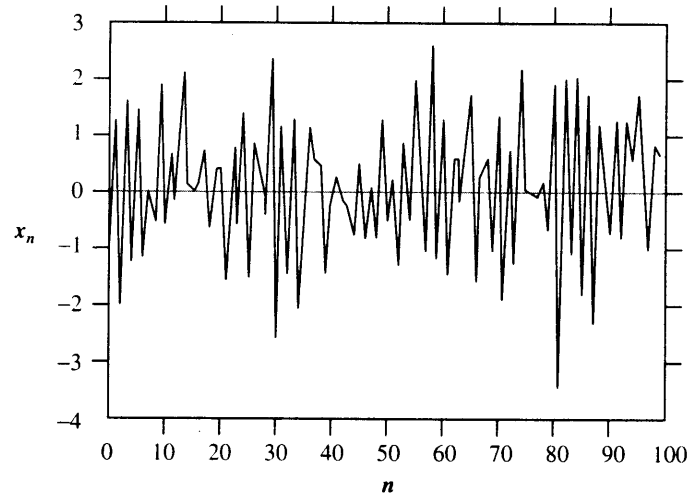

FIGURE 8.10 **Sample function of an AR(1) process with $a_1 = -0.6$.**

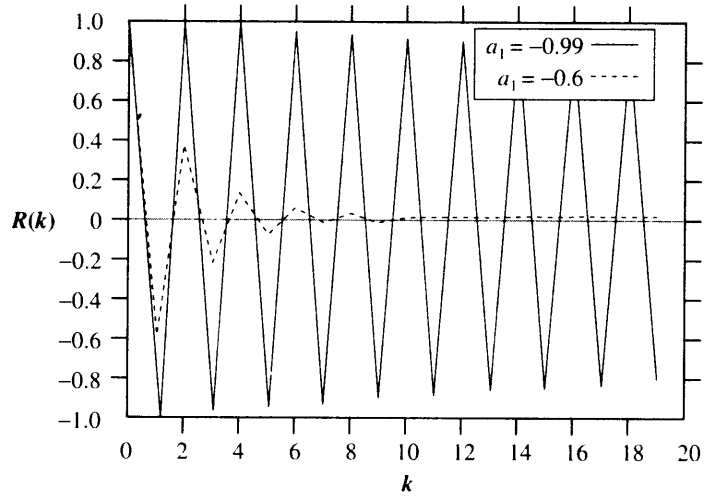

FIGURE 8.11 **Autocorrelation function of an AR(1) process with two negative values of a_1.**

is much higher than in the waveforms shown in Figures 8.7 and 8.8. However, if we were to look at the variation in magnitude, we can see that the higher value of a_1 results in magnitude values that are closer together.

This behavior is also reflected in the autocorrelation function, shown in Figure 8.11, as we might expect from looking at Equation (8.93). ♦

In Equation (8.84), instead of setting all the $\{b_j\}$ coefficients to zero, if we set all the $\{a_i\}$ coefficients to zero, we are left with the moving average part of the ARMA process:

$$x_n = \sum_{j=1}^{M} b_j \epsilon_{n-j} + \epsilon_n. \tag{8.94}$$

This process is called an Mth-order moving average process. This is a weighted average of the current and M past samples. Because of the form of this process, it is most useful when modeling slowly varying processes.

8.6.3 Physical Models

Physical models are based on the physics of the source output production. The physics are generally complicated and not amenable to a reasonable mathematical approximation. An exception to this rule is speech generation.

Speech Production

There has been a significant amount of research conducted in the area of speech production [104], and volumes have been written about it. We will try to summarize some of the pertinent aspects in this section.

Speech is produced by forcing air first through an elastic opening, the vocal cords, and then through cylindrical tubes with nonuniform diameter (the laryngeal, oral, nasal, and pharynx passages), and finally through cavities with changing boundaries such as the mouth and the nasal cavity. Everything past the vocal cords is generally referred to as the *vocal tract*. The first action generates the sound, which is then modulated into speech as it traverses through the vocal tract.

We will often be talking about filters in the coming chapters. We will try to describe filters more precisely at that time. For our purposes at present, a filter is a system that has an input and an output, and a rule for converting the input to the output, which we will call the *transfer function*. If we think of speech as the output of a filter, the sound generated by the air rushing past the vocal cords can be viewed as the input, while the rule for converting the input to the output is governed by the shape and physics of the vocal tract.

The output depends on the input and the transfer function. Let's look at each in turn. There are several different forms of input that can be generated by different conformations of the vocal cords and the associated cartilages. If the vocal cords are stretched shut and we force air through, the vocal cords vibrate, providing a periodic input. If a small aperture is left open, the input resembles white noise. By opening an aperture at different locations along the vocal cords, we can produce a white-noise–like input with certain dominant frequencies that depend on the location of the opening. The vocal tract can be modeled as a series of tubes of unequal diameter. If we now examine how an acoustic wave travels through this series of tubes, we find that the mathematical model that best describes this process is an autoregressive model. We will often encounter the autoregressive model when we discuss speech compression algorithms.

8.7 Summary

In this chapter we have looked at a variety of topics that will be useful to us when we study various lossy compression techniques, including distortion and its measurement, some new concepts from information theory, average mutual information and its connection to the rate of a compression scheme, and the rate distortion function. We have also briefly looked at some of the properties of the human visual system and the auditory system—most importantly, visual and auditory masking. The masking phenomena allow us to incur distortion in such a way that the distortion is not perceptible to the human observer. We also presented a model for speech production.

Further Reading

There are a number of excellent books available that delve more deeply in the area of information theory:

1. *Information Theory*, by R.B. Ash [15].
2. *Information Transmission*, by R.M. Fano [16].
3. *Information Theory and Reliable Communication*, by R.G. Gallagher [11].
4. *Entropy and Information Theory*, by R.M. Gray [17].
5. *Elements of Information Theory*, by T.M. Cover and J.A. Thomas [3].
6. *The Theory of Information and Coding*, by R.J. McEliece [6].

The subject of rate distortion theory is discussed in very clear terms in *Rate Distortion Theory*, by T. Berger [4].

For an introduction to the concepts behind speech perception, see *Voice and Speech Processing*, by T. Parsons [105].

8.8 Projects and Problems

1. Although SNR is a widely used measure of distortion, it often does not correlate with perceptual quality. In order to see this we conduct the following experiment. Using one of the images provided, generate two "reconstructed" images. For one of the reconstructions add a value of 10 to each pixel. For the other reconstruction, randomly add either $+10$ or -10 to each pixel.
   (a) What is the SNR for each of the reconstructions? Do the relative values reflect the difference in the perceptual quality?
   (b) Devise a mathematical measure that will better reflect the difference in perceptual quality for this particular case.
2. Consider the following lossy compression scheme for binary sequences. We divide the binary sequence into blocks of size M. For each block we count the number

of 0s. If this number is greater than or equal to $M/2$, we send a 0; otherwise, we send a 1.

(a) If the sequence is random with $P(0) = 0.8$, compute the rate and distortion (use Equation (8.54)) for $M = 1, 2, 4, 8, 16$. Compare your results with the rate distortion function for binary sources.

(b) Repeat assuming that the output of the encoder is encoded at a rate equal to the entropy of the output.

3. Write a program to implement the compression scheme described in the previous problem.

(a) Generate a random binary sequence with $P(0) = 0.8$, and compare your simulation results with the analytical results.

(b) Generate a binary first-order Markov sequence with $P(0|0) = 0.9$, and $P(1|1) = 0.9$. Encode it using your program. Discuss and comment on your results.

4. Show that

$$H(X_d|Y_d) = -\sum_{j=-\infty}^{\infty}\sum_{i=-\infty}^{\infty} f_{X|Y}(x_i|y_j)f_Y(y_j)\Delta\Delta \log f_{X|Y}(x_i|y_j) - \log\Delta. \tag{8.95}$$

5. For two random variables X and Y, show that

$$H(X|Y) \leq H(X)$$

with equality if X is independent of Y.
Hint: $E[\log(f(x))] \leq \log\{E[f(x)]\}$ (Jensen's inequality).

6. Given two random variables X and Y, show that $I(X; Y) = I(Y; X)$.

7. For a binary source with $P(0) = p$, $P(X = 0|Y = 1) = P(X = 1|Y = 0) = D$, and distortion measure

$$d(x_i, y_j) = x_i \oplus y_j,$$

show that

$$I(X; Y) = H_b(p) - H_b(D). \tag{8.96}$$

8. Find the autocorrelation function in terms of the model coefficients and σ_ϵ^2 for

(a) an AR(1) process,

(b) an MA(1) process, and

(c) an AR(2) process.

Scalar Quantization

9.1 Overview

In this chapter we begin our study of quantization, one of the simplest and most general ideas in lossy compression. We will look at scalar quantization in this chapter and continue with vector quantization in the next chapter. First, the general quantization problem is stated, then various solutions are examined, starting with the simpler solutions, which require the most assumptions, and proceeding to more complex solutions that require fewer assumptions. We describe uniform quantization with fixed-length codewords, first assuming a uniform source, then a source with a known probability density function (*pdf*) that is not necessarily uniform, and finally a source with unknown or changing statistics. We then look at *pdf*-optimized nonuniform quantization, followed by companded quantization. Finally, we return to the more general statement of the quantizer design problem and study entropy-coded quantization.

9.2 Introduction

In many lossy compression applications we are required to represent each source output using one of a small number of codewords. The number of possible distinct source output values is generally much larger than the number of codewords available to represent them. The process of representing a large—possibly infinite—set of values with a much smaller set is called *quantization*.

Consider a source that generates numbers between -10.0 and 10.0. A simple quantization scheme would be to represent each output of the source with the integer value closest to it. (If the source output is equally close to two integers, we will randomly pick one of them.) For example, if the source output is 2.47, we would represent it as 2, and if the source output is 3.1415926, we would represent it as 3.

This approach reduces the size of the alphabet required to represent the source output; the infinite number of values between -10.0 and 10.0 are represented with a set that contains only 21 values ($\{-10, \ldots, 0, \ldots, 10\}$). At the same time we have also forever lost the original value of the source output. If we are told that the reconstruction value is 3, we cannot tell whether the source output was 2.95, 3.16, 3.057932, or any other of an infinite set of values. In other words, we have lost some information. This loss of information is the reason for the use of the word "lossy" in many lossy compression schemes.

The set of inputs and outputs of a quantizer can be scalars or vectors. If they are scalars, we call the quantizers *scalar quantizers*. If they are vectors, we call the quantizers *vector quantizers*. We will study scalar quantizers in this chapter and vector quantizers in Chapter 10.

9.3 The Quantization Problem

Quantization is a very simple process. However, the design of the quantizer has a significant impact on the amount of compression obtained and loss incurred in a lossy compression scheme. Therefore, we will devote a lot of attention to issues related to the design of quantizers.

In practice, the quantizer consists of two mappings: an encoder mapping and a decoder mapping. The encoder divides the range of values that the source generates into a number of intervals. Each interval is represented by a distinct codeword. The encoder represents all the source outputs that fall into a particular interval by the codeword representing that interval. As there could be many—possibly infinitely many—distinct sample values that can fall in any given interval, the encoder mapping is irreversible. Knowing the code only tells us the interval to which the sample value belongs. It does not tell us which of the many values in the interval is the actual sample value. When the sample value comes from an analog source, the encoder is called an analog-to-digital (A/D) converter.

The encoder mapping for a quantizer with eight reconstruction values is shown in Figure 9.1. For this encoder, all samples with values between -1 and 0 would be assigned the code 011. All values between 0 and 1.0 would be assigned the codeword 100, and so on. On the two boundaries, all inputs with values greater than 3 would be assigned the code 111, and all inputs with values less than -3.0 would be assigned the code 000. Thus, any input

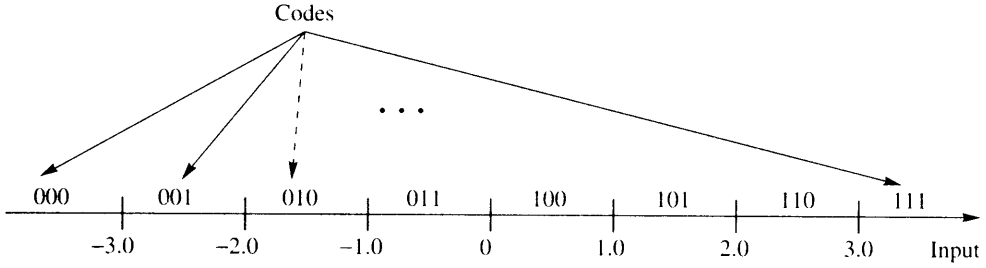

FIGURE 9.1 Mapping for a 3-bit encoder.

Input Codes	Output
000	−3.5
001	−2.5
010	−1.5
011	−0.5
100	0.5
101	1.5
110	2.5
111	3.5

FIGURE 9.2 Mapping for a 3-bit D/A converter.

that we receive will be assigned a codeword depending on the interval in which it falls. As we are using 3 bits to represent each value, we refer to this quantizer as a 3-bit quantizer.

For every codeword generated by the encoder, the decoder generates a reconstruction value. Because a codeword represents an entire interval, and there is no way of knowing which value in the interval was actually generated by the source, the decoder puts out a value that, in some sense, best represents all the values in the interval. Later, we will see how to use information we may have about the distribution of the input in the interval to obtain a representative value. For now, we simply use the midpoint of the interval as the representative value generated by the decoder. If the reconstruction is analog, the decoder is often referred to as a digital-to-analog (D/A) converter. A decoder mapping corresponding to the 3-bit encoder shown in Figure 9.1 is shown in Figure 9.2.

Example 9.3.1:

Suppose a sinusoid $4\cos(2\pi t)$ was sampled every 0.05 second. The sample was digitized using the A/D mapping shown in Figure 9.1 and reconstructed using the D/A mapping shown in Figure 9.2. The first few inputs, codewords, and reconstruction values are given in Table 9.1. Notice the first two samples in Table 9.1. Although the two input values are distinct, they both fall into the same interval in the quantizer. The encoder, therefore, represents both inputs with the same codeword, which in turn leads to identical reconstruction values.

TABLE 9.1 Digitizing a sine wave.

t	$4\cos(2\pi t)$	A/D Output	D/A Output	Error
0.05	3.804	111	3.5	0.304
0.10	3.236	111	3.5	−0.264
0.15	2.351	110	2.5	−0.149
0.20	1.236	101	1.5	−0.264

♦

Construction of the intervals (their location, etc.) can be viewed as part of the design of the encoder. Selection of reconstruction values is part of the design of the decoder. However, the fidelity of the reconstruction depends on both the intervals and the reconstruction values. Therefore, when designing or analyzing encoders and decoders, it is reasonable to view them as a pair. We call this encoder-decoder pair a *quantizer*. The quantizer mapping for the 3-bit encoder-decoder pair shown in Figures 9.1 and 9.2 can be represented by the input-output map shown in Figure 9.3. The quantizer accepts sample values, and depending on the interval in which the sample values fall, it provides an output codeword and a representation value. Using the map of Figure 9.3, we can see that an input to the quantizer of 1.7 will result in an output of 1.5, and an input of -0.3 will result in an output of -0.5.

From Figures 9.1–9.3 we can see that we need to know how to divide the input range into intervals, assign binary codes to these intervals, and find representation or output values for these intervals in order to specify a quantizer. We need to do all of this while satisfying distortion and rate criteria. In this chapter we will define distortion to be the average squared difference between the quantizer input and output. We call this the mean squared quantization error (msqe) and denote it by σ_q^2. The rate of the quantizer is the average number of bits

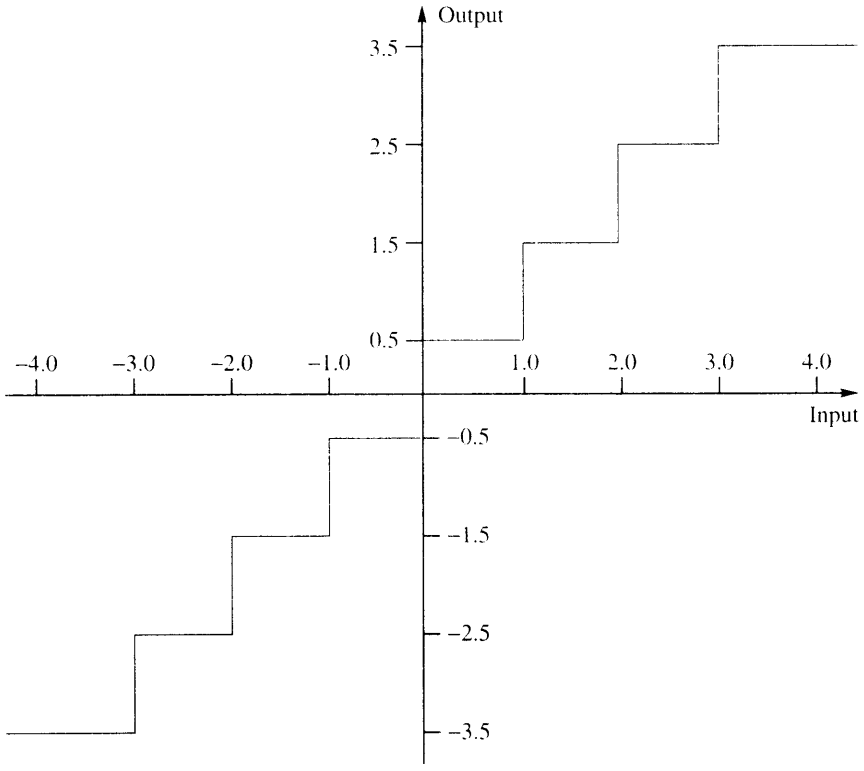

FIGURE 9.3 Quantizer input-output map.

required to represent a single quantizer output. We would like to get the lowest distortion for a given rate, or the lowest rate for a given distortion.

Let us pose the design problem in precise terms. Suppose we have an input modeled by a random variable X with *pdf* $f_X(x)$. If we wished to quantize this source using a quantizer with M intervals, we would have to specify $M+1$ endpoints for the intervals, and a representative value for each of the M intervals. The endpoints of the intervals are known as *decision boundaries*, while the representative values are called *reconstruction levels*. We will often model discrete sources with continuous distributions. For example, the difference between neighboring pixels is often modeled using a Laplacian distribution even though the differences can only take on a limited number of discrete values. Discrete processes are modeled with continuous distributions because it can simplify the design process considerably, and the resulting designs perform well in spite of the incorrect assumption. Several of the continuous distributions used to model source outputs are unbounded—that is, the range of values is infinite. In these cases, the first and last endpoints are generally chosen to be $\pm\infty$.

Let us denote the decision boundaries by $\{b_i\}_{i=0}^{M}$, the reconstruction levels by $\{y_i\}_{i=1}^{M}$, and the quantization operation by $Q(\cdot)$. Then

$$Q(x) = y_i \quad \text{iff} \quad b_{i-1} < x \leq b_i. \tag{9.1}$$

The mean squared quantization error is then given by

$$\sigma_q^2 = \int_{-\infty}^{\infty} (x - Q(x))^2 f_X(x)\,dx \tag{9.2}$$

$$= \sum_{i=1}^{M} \int_{b_{i-1}}^{b_i} (x - y_i)^2 f_X(x)\,dx. \tag{9.3}$$

The difference between the quantizer input x and output $y = Q(x)$, besides being referred to as the quantization error, is also called the *quantizer distortion* or *quantization noise*. But the word "noise" is somewhat of a misnomer. Generally, when we talk about noise we mean a process external to the source process. Because of the manner in which the quantization error is generated, it is dependent on the source process and, therefore, cannot be regarded as external to the source process. One reason for the use of the word "noise" in this context is that from time to time we will find it useful to model the quantization process as an additive noise process as shown in Figure 9.4.

If we use fixed-length codewords to represent the quantizer output, then the size of the output alphabet immediately specifies the rate. If the number of quantizer outputs is M, then the rate is given by

$$R = \lceil \log_2 M \rceil. \tag{9.4}$$

For example, if $M = 8$, then $R = 3$. In this case, we can pose the quantizer design problem as follows:

> Given an input *pdf* $f_X(x)$ and the number of levels M in the quantizer, find the decision boundaries $\{b_i\}$ and the reconstruction levels $\{y_i\}$ so as to minimize the mean squared quantization error given by Equation (9.3).

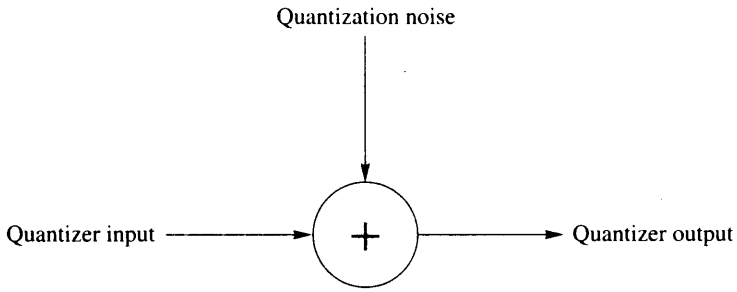

FIGURE 9.4 Additive noise model of a quantizer.

TABLE 9.2 Codeword assignment for an eight-level quantizer.

y_1	1110
y_2	1100
y_3	100
y_4	00
y_5	01
y_6	101
y_7	1101
y_8	1111

However, if we are allowed to use variable-length codes, such as Huffman codes or arithmetic codes, along with the size of the alphabet, the selection of the decision boundaries will also affect the rate of the quantizer. Consider the codeword assignment for the output of an eight-level quantizer shown in Table 9.2.

According to this codeword assignment, if the output y_4 occurs, we use 2 bits to encode it, while if the output y_1 occurs, we need 4 bits to encode it. Obviously, the rate will depend on how often we have to encode y_4 versus how often we have to encode y_1. In other words, the rate will depend on the probability of occurrence of the outputs. If l_i is the length of the codeword corresponding to the output y_i, and $P(y_i)$ is the probability of occurrence of y_i, then the rate is given by

$$R = \sum_{i=1}^{M} l_i P(y_i). \tag{9.5}$$

However, the probabilities $\{P(y_i)\}$ depend on the decision boundaries $\{b_i\}$. For example, the probability of y_i occurring is given by

$$P(y_i) = \int_{b_{i-1}}^{b_i} f_X(x)dx.$$

Therefore, the rate R is a function of the decision boundaries and is given by the expression

$$R = \sum_{i=1}^{M} l_i \int_{b_{i-1}}^{b_i} f_X(x)dx. \tag{9.6}$$

From this discussion and Equations (9.3) and (9.6), we see that for a given source input, the partitions we select and the representation for those partitions will determine the distortion incurred during the quantization process. The partitions we select and the binary codes for the partitions will determine the rate for the quantizer. Thus, the problem of finding the optimum partitions, codes, and representation levels are all linked. In light of this information, we can restate our problem statement:

Given a distortion constraint

$$\sigma_q^2 \leq D^* \tag{9.7}$$

find the decision boundaries, reconstruction levels, and binary codes that minimize the rate given by Equation (9.6), while satisfying Equation (9.7).

Or, given a rate constraint

$$R \leq R^* \tag{9.8}$$

find the decision boundaries, reconstruction levels, and binary codes that minimize the distortion given by Equation (9.3), while satisfying Equation (9.8).

This problem statement of quantizer design, while more general than our initial statement, is substantially more complex. Fortunately, in practice there are situations in which we can simplify the problem. We often use fixed-length codewords to encode the quantizer output. In this case, the rate is simply the number of bits used to encode each output, and we can use our initial statement of the quantizer design problem. We start our study of quantizer design by looking at this simpler version of the problem, and later use what we have learned in this process to attack the more complex version.

9.4 Uniform Quantizer

The simplest type of quantizer is the uniform quantizer. All intervals are the same size in the uniform quantizer, except possibly for the two outer intervals. In other words, the decision boundaries are spaced evenly. The reconstruction values are also spaced evenly, with the same spacing as the decision boundaries; in the inner intervals, they are the midpoints of the intervals. This constant spacing is usually referred to as the step size and is denoted by Δ. The quantizer shown in Figure 9.3 is a uniform quantizer with $\Delta = 1$. It does not have zero as one of its representation levels. Such a quantizer is called a *midrise quantizer*. An alternative uniform quantizer could be the one shown in Figure 9.5. This is called a *midtread quantizer*. As the midtread quantizer has zero as one of its output levels, it is especially useful in situations where it is important that the zero value be represented—for example,

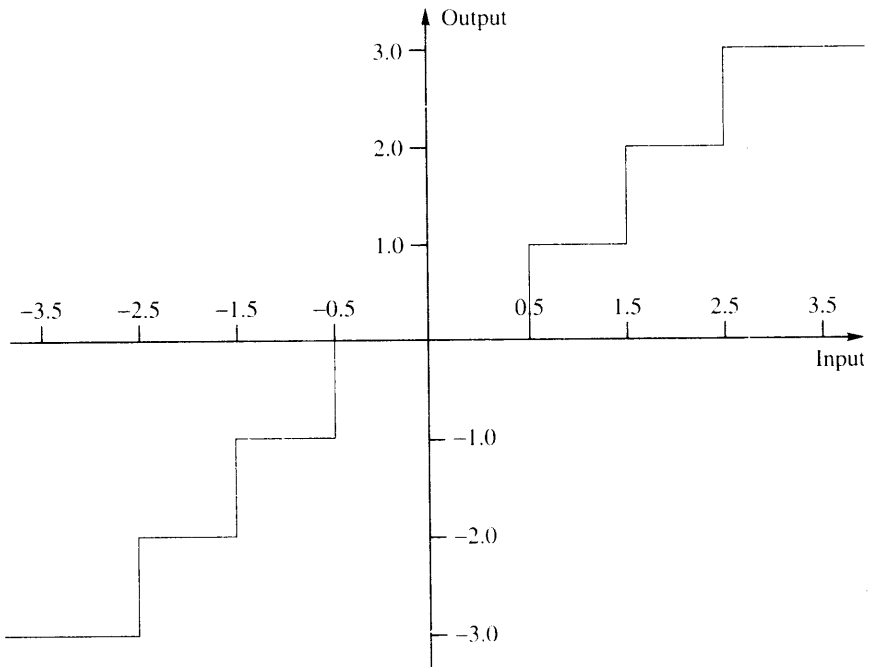

FIGURE 9.5 A midtread quantizer.

control systems in which it is important to represent a zero value accurately, and audio coding schemes in which we need to represent silence periods. Notice that the midtread quantizer has only seven intervals or levels. That means that if we were using a fixed-length 3-bit code, we would have one codeword left over.

Usually, we use a midrise quantizer if the number of levels is even and a midtread quantizer if the number of levels is odd. For the remainder of this chapter, unless we specifically mention otherwise, we will assume that we are dealing with midrise quantizers. We will also generally assume that the input distribution is symmetric around the origin and the quantizer is also symmetric. (The optimal minimum mean squared error quantizer for a symmetric distribution need not be symmetric [106].) Given all these assumptions, the design of a uniform quantizer consists of finding the step size Δ that minimizes the distortion for a given input process and number of decision levels.

Uniform Quantization of a Uniformly Distributed Source

We start our study of quantizer design with the simplest of all cases: design of a uniform quantizer for a uniformly distributed source. Suppose we want to design an M-level uniform quantizer for an input that is uniformly distributed in the interval $[-X_{\max}, X_{\max}]$. This means

we need to divide the $[-X_{\max}, X_{\max}]$ interval into M equally sized intervals. In this case, the step size Δ is given by

$$\Delta = \frac{2X_{\max}}{M}. \tag{9.9}$$

The distortion in this case becomes

$$\sigma_q^2 = 2\sum_{i=1}^{\frac{M}{2}} \int_{(i-1)\Delta}^{i\Delta} \left(x - \frac{2i-1}{2}\Delta\right)^2 \frac{1}{2X_{\max}} dx. \tag{9.10}$$

If we evaluate this integral (after some suffering), we find that the msqe is $\Delta^2/12$.

The same result can be more easily obtained if we examine the behavior of the quantization error q given by

$$q = x - Q(x). \tag{9.11}$$

In Figure 9.6 we plot the quantization error versus the input signal for an eight-level uniform quantizer, with an input that lies in the interval $[-X_{\max}, X_{\max}]$. Notice that the quantization error lies in the interval $[-\frac{\Delta}{2}, \frac{\Delta}{2}]$. As the input is uniform, it is not difficult to establish that the quantization error is also uniform over this interval. Thus, the mean squared quantization error is the second moment of a random variable uniformly distributed in the interval $[-\frac{\Delta}{2}, \frac{\Delta}{2}]$:

$$\sigma_q^2 = \frac{1}{\Delta}\int_{-\frac{\Delta}{2}}^{\frac{\Delta}{2}} q^2 dq \tag{9.12}$$

$$= \frac{\Delta^2}{12}. \tag{9.13}$$

Let us also calculate the signal-to-noise ratio for this case. The signal variance σ_x^2 for a uniform random variable, which takes on values in the interval $[-X_{\max}, X_{\max}]$, is $\frac{(2X_{\max})^2}{12}$.

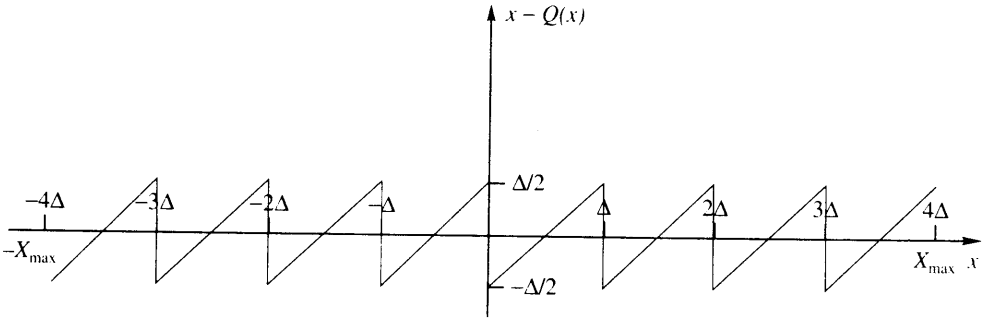

FIGURE 9.6 **Quantization error for a uniform midrise quantizer with a uniformly distributed input.**

The value of the step size Δ is related to $X_{\max}$ and the number of levels M by

$$\Delta = \frac{2X_{\max}}{M}.$$

For the case where we use a fixed-length code, with each codeword being made up of n bits, the number of codewords or the number of reconstruction levels M is 2^n. Combining all this, we have

$$\begin{aligned}
\text{SNR(dB)} &= 10\log_{10}\left(\frac{\sigma_s^2}{\sigma_q^2}\right) && (9.14)\\
&= 10\log_{10}\left(\frac{(2X_{\max})^2}{12}\cdot\frac{12}{\Delta^2}\right) && (9.15)\\
&= 10\log_{10}\left(\frac{(2X_{\max})^2}{12}\frac{12}{(\frac{2X_{\max}}{M})^2}\right) && (9.16)\\
&= 10\log_{10}(M^2)\\
&= 20\log_{10}(2^n)\\
&= 6.02n\,\text{dB}. && (9.17)
\end{aligned}$$

This equation says that for every additional bit in the quantizer, we get an increase in the signal-to-noise ratio of 6.02 dB. This is a well-known result and is often used to get an indication of the maximum gain available if we increase the rate. However, remember that we obtained this result under some assumptions about the input. If the assumptions are not true, this result will not hold true either.

Example 9.4.1: Image compression

A probability model for the variations of pixels in an image is almost impossible to obtain because of the great variety of images available. A common approach is to declare the pixel values to be uniformly distributed between 0 and $2^b - 1$, where b is the number of bits per pixel. For most of the images we deal with, the number of bits per pixel is 8; therefore, the pixel values would be assumed to vary uniformly between 0 and 255. Let us quantize our test image Sena using a uniform quantizer.

If we wanted to use only 1 bit per pixel, we would divide the range [0, 255] into two intervals, [0, 127] and [128, 255]. The first interval would be represented by the value 64, the midpoint of the first interval; the pixels in the second interval would be represented by the pixel value 196, the midpoint of the second interval. In other words, the boundary values are $\{0, 128, 255\}$, while the reconstruction values are $\{64, 196\}$. The quantized image is shown in Figure 9.7. As expected, almost all the details in the image have disappeared. If we were to use a 2-bit quantizer, with boundary values $\{0, 64, 128, 196, 255\}$ and reconstruction levels $\{32, 96, 160, 224\}$, we get considerably more detail. The level of detail increases as the use of bits increases until at 6 bits per pixel, the reconstructed image is indistinguishable from the original, at least to a casual observer. The 1-, 2-, and 3-bit images are shown in Figure 9.7.

FIGURE 9. 7 **Top left: original Sena image; top right: 1 bit/pixel image; bottom left: 2 bits/pixel; bottorm right: 3 bits/pixel.**

Looking at the lower-rate images, we notice a couple of things. First, the lower-rate images are darker than the original, and the lowest-rate reconstructions are the darkest. The reason for this is that the quantization process usually results in scaling down of the dynamic range of the input. For example, in the 1-bit-per-pixel reproduction, the highest pixel value is 196, as opposed to 255 for the original image. As higher gray values represent lighter shades, there is a corresponding darkening of the reconstruction. The other thing to notice in the low-rate reconstruction is that wherever there were smooth changes in gray values there are now abrupt transitions. This is especially evident in the face and neck area, where gradual shading has been transformed to blotchy regions of constant values. This is because a range of values is being mapped to the same value, as was the case for the first two samples of the sinusoid in Example 9.3.1. For obvious reasons, this effect is called *contouring*. The perceptual effect of contouring can be reduced by a procedure called *dithering* [107]. ♦

Uniform Quantization of Nonuniform Sources

Quite often the sources we deal with do not have a uniform distribution; however, we still want the simplicity of a uniform quantizer. In these cases, even if the sources are bounded, simply dividing the range of the input by the number of quantization levels does not produce a very good design.

Example 9.4.2:

Suppose our input fell within the interval $[-1, 1]$ with probability 0.95, and fell in the intervals $[-100, 1)$, $(1, 100]$ with probability 0.05. Suppose we wanted to design an eight-level uniform quantizer. If we followed the procedure of the previous section, the step size would be 25. This means that inputs in the $[-1, 0)$ interval would be represented by the value -12.5, and inputs in the interval $[0, 1)$ would be represented by the value 12.5. The maximum quantization error that can be incurred is 12.5. However, at least 95% of the time, the *minimum* error that will be incurred is 11.5. Obviously, this is not a very good design. A much better approach would be to use a smaller step size, which would result in better representation of the values in the $[-1, 1]$ interval, even if it meant a larger maximum error. Suppose we pick a step size of 0.3. In this case, the maximum quantization error goes from 12.5 to 98.95. However, 95% of the time the quantization error will be less than 0.15. Therefore, the average distortion, or msqe, for this quantizer would be substantially less than the msqe for the first quantizer. ♦

We can see that when the distribution is no longer uniform, it is not a good idea to obtain the step size by simply dividing the range of the input by the number of levels. This approach becomes totally impractical when we model our sources with distributions that are unbounded, such as the Gaussian distribution. Therefore, we include the *pdf* of the source in the design process.

Our objective is to find the step size that, for a given value of M, will minimize the distortion. The simplest way to do this is to write the distortion as a function of the step size, and then minimize this function. An expression for the distortion, or msqe, for an M-level uniform quantizer as a function of the step size can be found by replacing the b_is and y_is in Equation (9.3) with functions of Δ. As we are dealing with a symmetric condition, we need only compute the distortion for positive values of x; the distortion for negative values of x will be the same.

From Figure 9.8, we see that the decision boundaries are integral multiples of Δ, and the representation level for the interval $[(k-1)\Delta, k\Delta)$ is simply $\frac{2k-1}{2}\Delta$. Therefore, the expression for msqe becomes

$$\sigma_q^2 = 2\sum_{i=1}^{\frac{M}{2}-1}\int_{(i-1)\Delta}^{i\Delta}\left(x-\frac{2i-1}{2}\Delta\right)^2 f_X(x)dx + 2\int_{(\frac{M}{2}-1)\Delta}^{\infty}\left(x-\frac{M-1}{2}\Delta\right)^2 f_X(x)dx. \tag{9.18}$$

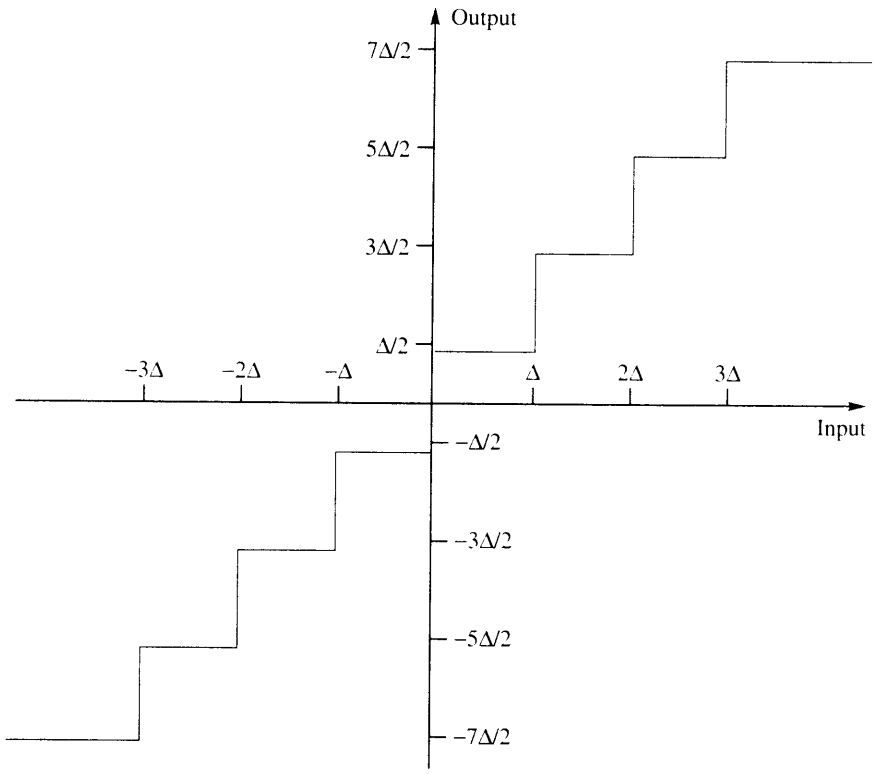

FIGURE 9.8 A uniform midrise quantizer.

To find the optimal value of Δ, we simply take a derivative of this equation and set it equal to zero [108] (see Problem 1).

$$\frac{\delta\sigma_q^2}{\delta\Delta} = -\sum_{i=1}^{\frac{M}{2}-1}(2i-1)\int_{(i-1)\Delta}^{i\Delta}\left(x-\frac{2i-1}{2}\Delta\right)f_X(x)dx$$
$$-(M-1)\int_{(\frac{M}{2}-1)\Delta}^{\infty}\left(x-\frac{M-1}{2}\Delta\right)f_X(x)dx = 0. \qquad (9.19)$$

This is a rather messy-looking expression, but given the *pdf* $f_X(x)$, it is easy to solve using any one of a number of numerical techniques (see Problem 2). In Table 9.3, we list step sizes found by solving (9.19) for nine different alphabet sizes and three different distributions.

Before we discuss the results in Table 9.3, let's take a look at the quantization noise for the case of nonuniform sources. Nonuniform sources are often modeled by *pdf*s with unbounded support. That is, there is a nonzero probability of getting an unbounded input. In practical situations, we are not going to get inputs that are unbounded, but often it is very convenient to model the source process with an unbounded distribution. The classic example of this is measurement error, which is often modeled as having a Gaussian distribution.

TABLE 9.3 Optimum step size and SNR for uniform quantizers for different distributions and alphabet sizes [108, 109].

Alphabet Size	Uniform		Gaussian		Laplacian	
	Step Size	SNR	Step Size	SNR	Step Size	SNR
2	1.732	6.02	1.596	4.40	1.414	3.00
4	0.866	12.04	0.9957	9.24	1.0873	7.05
6	0.577	15.58	0.7334	12.18	0.8707	9.56
8	0.433	18.06	0.5860	14.27	0.7309	11.39
10	0.346	20.02	0.4908	15.90	0.6334	12.81
12	0.289	21.60	0.4238	17.25	0.5613	13.98
14	0.247	22.94	0.3739	18.37	0.5055	14.98
16	0.217	24.08	0.3352	19.36	0.4609	15.84
32	0.108	30.10	0.1881	24.56	0.2799	20.46

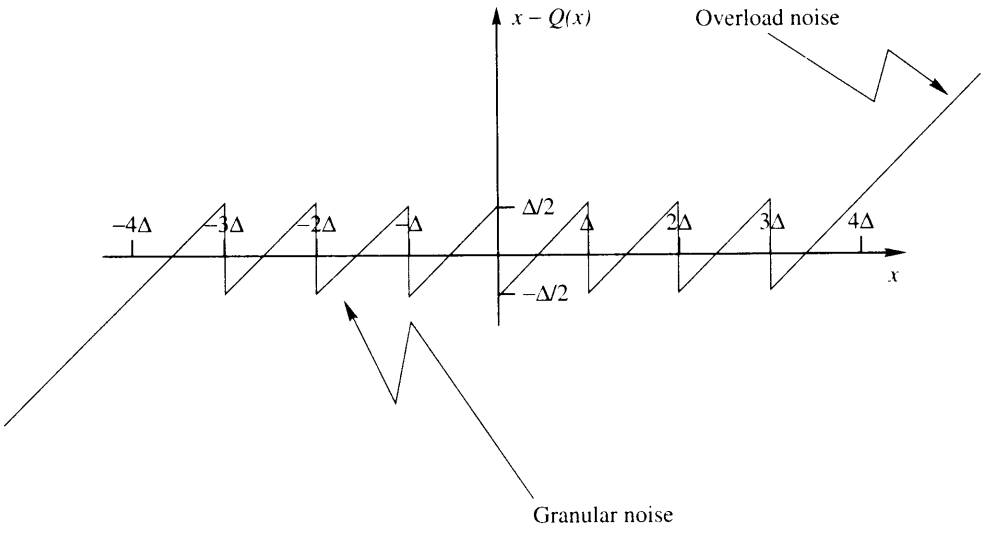

FIGURE 9.9 Quantization error for a uniform midrise quantizer.

even when the measurement error is known to be bounded. If the input is unbounded, the quantization error is no longer bounded either. The quantization error as a function of input is shown in Figure 9.9. We can see that in the inner intervals the error is still bounded by $\frac{\Delta}{2}$; however, the quantization error in the outer intervals is unbounded. These two types of quantization errors are given different names. The bounded error is called *granular error* or *granular noise*, while the unbounded error is called *overload error* or *overload noise*. In the expression for the msqe in Equation (9.18), the first term represents the granular noise, while the second term represents the overload noise. The probability that the input will fall into the overload region is called the *overload probability* (Figure 9.10).

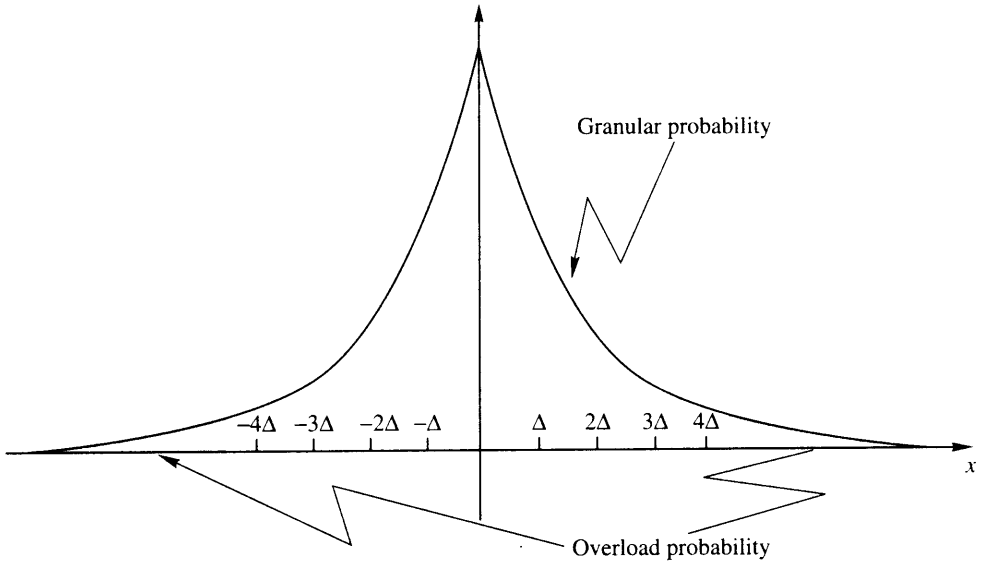

FIGURE 9. 10 Overload and granular regions for a 3-bit uniform quantizer.

The nonuniform sources we deal with have probability density functions that are generally peaked at zero and decay as we move away from the origin. Therefore, the overload probability is generally much smaller than the probability of the input falling in the granular region. As we see from Equation (9.19), an increase in the size of the step size Δ will result in an increase in the value of $\left(\frac{M}{2} - 1\right)\Delta$, which in turn will result in a decrease in the overload probability and the second term in Equation (9.19). However, an increase in the step size Δ will also increase the granular nóise, which is the first term in Equation (9.19). The design process for the uniform quantizer is a balancing of these two effects. An important parameter that describes this trade-off is the loading factor f_l, defined as the ratio of the maximum value the input can take in the granular region to the standard deviation. A common value of the loading factor is 4. This is also referred to as 4σ *loading*.

Recall that when quantizing an input with a uniform distribution, the SNR and bit rate are related by Equation (9.17), which says that for each bit increase in the rate there is an increase of 6.02 dB in the SNR. In Table 9.3, along with the step sizes, we have also listed the SNR obtained when a million input values with the appropriate *pdf* are quantized using the indicated quantizer.

From this table, we can see that, although the SNR for the uniform distribution follows the rule of a 6.02 dB increase in the signal-to-noise ratio for each additional bit, this is not true for the other distributions. Remember that we made some assumptions when we obtained the $6.02n$ rule that are only valid for the uniform distribution. Notice that the more peaked a distribution is (that is, the further away from uniform it is), the more it seems to vary from the 6.02 dB rule.

We also said that the selection of Δ is a balance between the overload and granular errors. The Laplacian distribution has more of its probability mass away from the origin in

its tails than the Gaussian distribution. This means that for the same step size and number of levels there is a higher probability of being in the overload region if the input has a Laplacian distribution than if the input has a Gaussian distribution. The uniform distribution is the extreme case, where the overload probability is zero. For the same number of levels, if we increase the step size, the size of the overload region (and hence the overload probability) is reduced at the expense of granular noise. Therefore, for a given number of levels, if we were picking the step size to balance the effects of the granular and overload noise, distributions that have heavier tails will tend to have larger step sizes. This effect can be seen in Table 9.3. For example, for eight levels the step size for the uniform quantizer is 0.433. The step size for the Gaussian quantizer is larger (0.586), while the step size for the Laplacian quantizer is larger still (0.7309).

Mismatch Effects

We have seen that for a result to hold, the assumptions we used to obtain the result have to hold. When we obtain the optimum step size for a particular uniform quantizer using Equation (9.19), we make some assumptions about the statistics of the source. We assume a certain distribution and certain parameters of the distribution. What happens when our assumptions do not hold? Let's try to answer this question empirically.

We will look at two types of mismatches. The first is when the assumed distribution type matches the actual distribution type, but the variance of the input is different from the assumed variance. The second mismatch is when the actual distribution type is different from the distribution type assumed when obtaining the value of the step size. Throughout our discussion, we will assume that the mean of the input distribution is zero.

In Figure 9.11, we have plotted the signal-to-noise ratio as a function of the ratio of the actual to assumed variance of a 4-bit Gaussian uniform quantizer, with a Gaussian

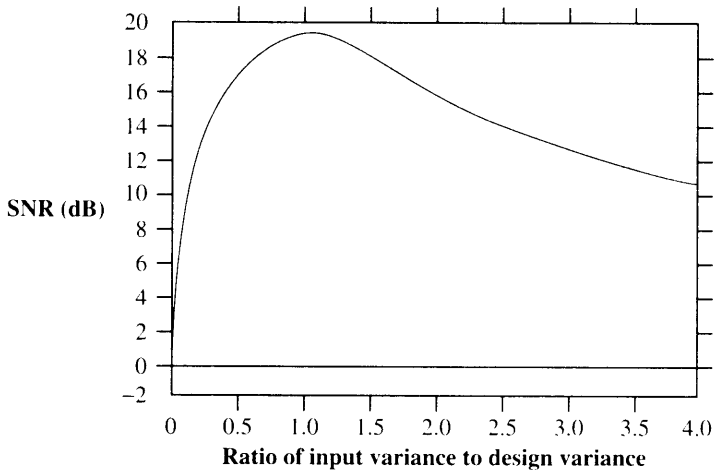

FIGURE 9.11 **Effect of variance mismatch on the performance of a 4-bit uniform quantizer.**

input. (To see the effect under different conditions, see Problem 5.) Remember that for a distribution with zero mean, the variance is given by $\sigma_x^2 = E[X^2]$, which is also a measure of the power in the signal X. As we can see from the figure, the signal-to-noise ratio is maximum when the input signal variance matches the variance assumed when designing the quantizer. From the plot we also see that there is an asymmetry; the SNR is considerably worse when the input variance is lower than the assumed variance. This is because the SNR is a ratio of the input variance and the mean squared quantization error. When the input variance is smaller than the assumed variance, the mean squared quantization error actually drops because there is less overload noise. However, because the input variance is low, the ratio is small. When the input variance is higher than the assumed variance, the msqe increases substantially, but because the input power is also increasing, the ratio does not decrease as dramatically. To see this more clearly, we have plotted the mean squared error versus the signal variance separately in Figure 9.12. We can see from these figures that the decrease in signal-to-noise ratio does not always correlate directly with an increase in msqe.

The second kind of mismatch is where the input distribution does not match the distribution assumed when designing the quantizer. In Table 9.4 we have listed the SNR when inputs with different distributions are quantized using several different eight-level quantizers. The quantizers were designed assuming a particular input distribution.

Notice that as we go from left to right in the table, the designed step size becomes progressively larger than the "correct" step size. This is similar to the situation where the input variance is smaller than the assumed variance. As we can see when we have a mismatch that results in a smaller step size relative to the optimum step size, there is a greater drop in performance than when the quantizer step size is larger than its optimum value.

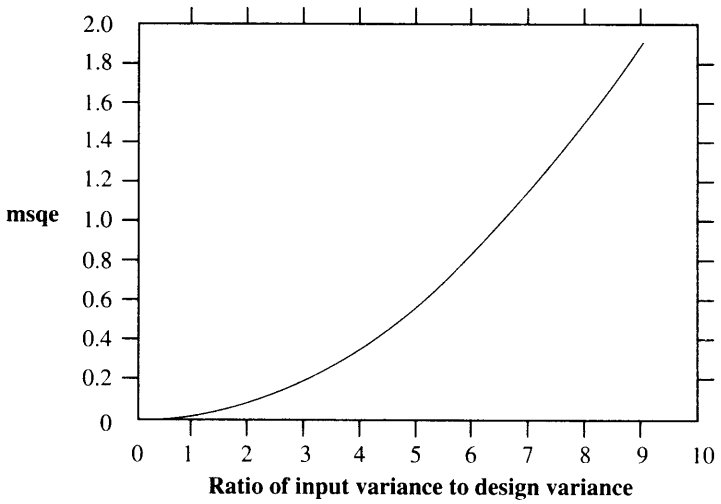

FIGURE 9.12 **The msqe as a function of variance mismatch with a 4-bit uniform quantizer.**

TABLE 9.4 Demonstration of the effect of mismatch using eight- level quantizers (dB).

Input Distribution	Uniform Quantizer	Gaussian Quantizer	Laplacian Quantizer	Gamma Quantizer
Uniform	18.06	15.56	13.29	12.41
Gaussian	12.40	14.27	13.37	12.73
Laplacian	8.80	10.79	11.39	11.28
Gamma	6.98	8.06	8.64	8.76

9.5 Adaptive Quantization

One way to deal with the mismatch problem is to adapt the quantizer to the statistics of the input. Several things might change in the input relative to the assumed statistics, including the mean, the variance, and the *pdf*. The strategy for handling each of these variations can be different, though certainly not exclusive. If more than one aspect of the input statistics changes, it is possible to combine the strategies for handling each case separately. If the mean of the input is changing with time, the best strategy is to use some form of differential encoding (discussed in some detail in Chapter 11). For changes in the other statistics, the common approach is to adapt the quantizer parameters to the input statistics.

There are two main approaches to adapting the quantizer parameters: an *off-line* or *forward adaptive* approach, and an *on-line* or *backward adaptive* approach. In forward adaptive quantization, the source output is divided into blocks of data. Each block is analyzed before quantization, and the quantizer parameters are set accordingly. The settings of the quantizer are then transmitted to the receiver as *side information*. In backward adaptive quantization, the adaptation is performed based on the quantizer output. As this is available to both transmitter and receiver, there is no need for side information.

9.5.1 Forward Adaptive Quantization

Let us first look at approaches for adapting to changes in input variance using the forward adaptive approach. This approach necessitates a delay of at least the amount of time required to process a block of data. The insertion of side information in the transmitted data stream may also require the resolution of some synchronization problems. The size of the block of data processed also affects a number of other things. If the size of the block is too large, then the adaptation process may not capture the changes taking place in the input statistics. Furthermore, large block sizes mean more delay, which may not be tolerable in certain applications. On the other hand, small block sizes mean that the side information has to be transmitted more often, which in turn means the amount of overhead per sample increases. The selection of the block size is a trade-off between the increase in side information necessitated by small block sizes and the loss of fidelity due to large block sizes (see Problem 7).

The variance estimation procedure is rather simple. At time n we use a block of N future samples to compute an estimate of the variance

$$\hat{\sigma}_q^2 = \frac{1}{N} \sum_{i=0}^{N-1} x_{n+i}^2. \tag{9.20}$$

Note that we are assuming that our input has a mean of zero. The variance information also needs to be quantized so that it can be transmitted to the receiver. Usually, the number of bits used to quantize the value of the variance is significantly larger than the number of bits used to quantize the sample values.

Example 9.5.1:

In Figure 9.13 we show a segment of speech quantized using a fixed 3-bit quantizer. The step size of the quantizer was adjusted based on the statistics of the entire sequence. The sequence was the `testm.raw` sequence from the sample data sets, consisting of about 4000 samples of a male speaker saying the word "test." The speech signal was sampled at 8000 samples per second and digitized using a 16-bit A/D.

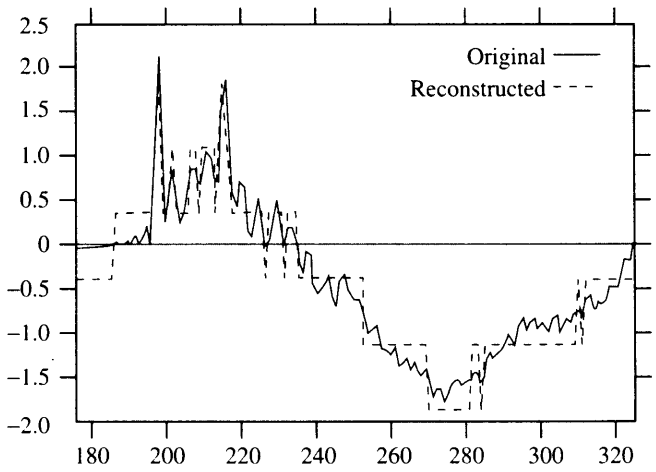

FIGURE 9.13 Original 16-bit speech and compressed 3-bit speech sequences.

We can see from the figure that, as in the case of the example of the sinusoid earlier in this chapter, there is a considerable loss in amplitude resolution. Sample values that are close together have been quantized to the same value.

The same sequence quantized with a forward adaptive quantizer is shown in Figure 9.14. For this example, we divided the input into blocks of 128 samples. Before quantizing the samples in a block, the standard deviation for the samples in the block was obtained. This value was quantized using an 8-bit quantizer and sent to both the transmitter and receiver.

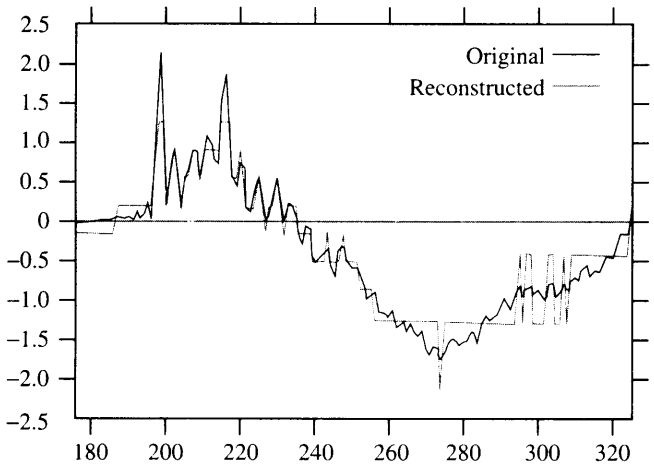

FIGURE 9.14 **Original 16-bit speech sequence and sequence obtained using an eight-level forward adaptive quantizer.**

The samples in the block were then normalized using this value of the standard deviation. Notice that the reconstruction follows the input much more closely, though there seems to be room for improvement, especially in the latter half of the displayed samples. ♦

Example 9.5.2:

In Example 9.4.1, we used a uniform quantizer with the assumption that the input is uniformly distributed. Let us refine this source model a bit and say that while the source is uniformly distributed over different regions, the range of the input changes. In a forward adaptive quantization scheme, we would obtain the minimum and maximum values for each block of data, which would be transmitted as side information. In Figure 9.15, we see the Sena image quantized with a block size of 8×8 using 3-bit forward adaptive uniform quantization. The side information consists of the minimum and maximum values in each block, which require 8 bits each. Therefore, the overhead in this case is $\frac{16}{8\times 8}$ or 0.25 bits per pixel, which is quite small compared to the number of bits per sample used by the quantizer.

The resulting image is hardly distinguishable from the original. Certainly at higher rates, forward adaptive quantization seems to be a very good alternative. ♦

9.5.2 Backward Adaptive Quantization

In backward adaptive quantization, only the past quantized samples are available for use in adapting the quantizer. The values of the input are only known to the encoder; therefore, this information cannot be used to adapt the quantizer. How can we get information about mismatch simply by examining the output of the quantizer without knowing what the input was? If we studied the output of the quantizer for a long period of time, we could get some idea about mismatch from the distribution of output values. If the quantizer step size Δ is

FIGURE 9. 15 **Sena image quantized to 3.25 bits per pixel using forward adaptive quantization.**

well matched to the input, the probability that an input to the quantizer would land in a particular interval would be consistent with the *pdf* assumed for the input. However, if the actual *pdf* differs from the assumed *pdf*, the number of times the input falls in the different quantization intervals will be inconsistent with the assumed *pdf*. If Δ is smaller than what it should be, the input will fall in the outer levels of the quantizer an excessive number of times. On the other hand, if Δ is larger than it should be for a particular source, the input will fall in the inner levels an excessive number of times. Therefore, it seems that we should observe the output of the quantizer for a long period of time, then expand the quantizer step size if the input falls in the outer levels an excessive number of times, and contract the step size if the input falls in the inner levels an excessive number of times.

Nuggehally S. Jayant at Bell Labs showed that we did not need to observe the quantizer output over a long period of time [110]. In fact, we could adjust the quantizer step size after observing a single output. Jayant named this quantization approach "quantization with one word memory." The quantizer is better known as the *Jayant quantizer*. The idea behind the Jayant quantizer is very simple. If the input falls in the outer levels, the step size needs to be expanded, and if the input falls in the inner quantizer levels, the step size needs to be reduced. The expansions and contractions should be done in such a way that once the quantizer is matched to the input, the product of the expansions and contractions is unity.

The expansion and contraction of the step size is accomplished in the Jayant quantizer by assigning a *multiplier* M_k to each interval. If the $(n-1)$th input falls in the kth interval, the step size to be used for the nth input is obtained by multiplying the step size used for the $(n-1)$th input with M_k. The multiplier values for the inner levels in the quantizer are less than one, and the multiplier values for the outer levels of the quantizer are greater than one.

Therefore, if an input falls into the inner levels, the quantizer used to quantize the next input will have a smaller step size. Similarly, if an input falls into the outer levels, the step size will be multiplied with a value greater than one, and the next input will be quantized using a larger step size. Notice that the step size for the current input is modified based on the previous quantizer output. The previous quantizer output is available to both the transmitter and receiver, so there is no need to send any additional information to inform the receiver about the adaptation. Mathematically, the adaptation process can be represented as

$$\Delta_n = M_{l(n-1)}\Delta_{n-1} \tag{9.21}$$

where $l(n-1)$ is the quantization interval at time $n-1$.

In Figure 9.16 we show a 3-bit uniform quantizer. We have eight intervals represented by the different quantizer outputs. However, the multipliers for symmetric intervals are identical because of symmetry:

$$M_0 = M_4 \quad M_1 = M_5 \quad M_2 = M_6 \quad M_3 = M_7$$

Therefore, we only need four multipliers. To see how the adaptation proceeds, let us work through a simple example using this quantizer.

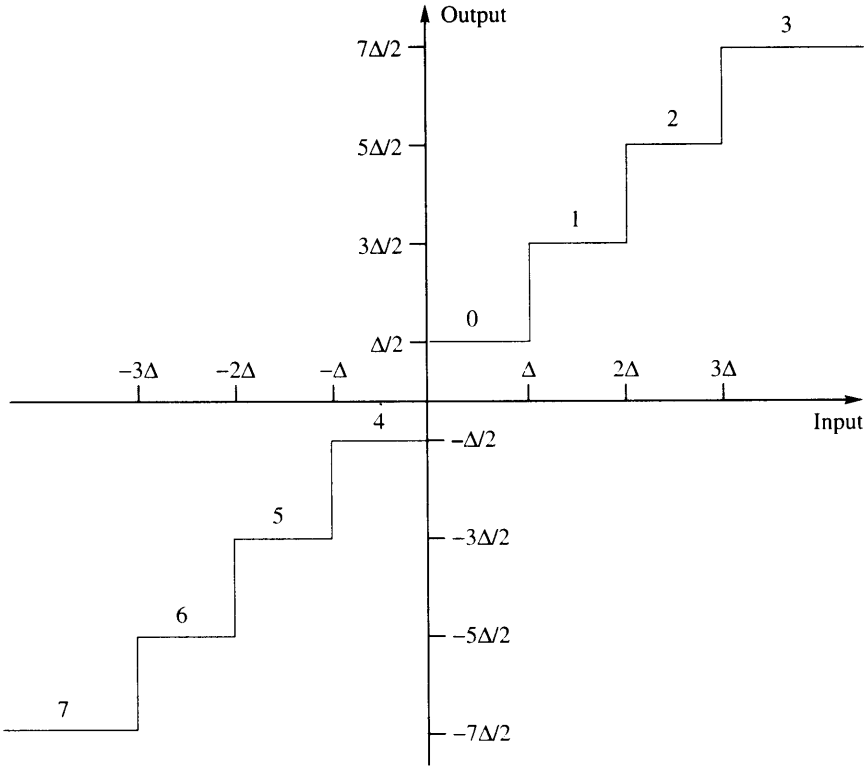

FIGURE 9.16 Output levels for the Jayant quantizer.

Example 9.5.3: Jayant quantizer

For the quantizer in Figure 9.16, suppose the multiplier values are $M_0 = M_4 = 0.8$, $M_1 = M_5 = 0.9$, $M_2 = M_6 = 1$, $M_3 = M_7 = 1.2$; the initial value of the step size, Δ_0, is 0.5; and the sequence to be quantized is $0.1, -0.2, 0.2, 0.1, -0.3, 0.1, 0.2, 0.5, 0.9, 1.5, \ldots$. When the first input is received, the quantizer step size is 0.5. Therefore, the input falls into level 0, and the output value is 0.25, resulting in an error of 0.15. As this input fell into the quantizer level 0, the new step size Δ_1 is $M_0 \times \Delta_0 = 0.8 \times 0.5 = 0.4$. The next input is -0.2, which falls into level 4. As the step size at this time is 0.4, the output is -0.2. To update, we multiply the current step size with M_4. Continuing in this fashion, we get the sequence of step sizes and outputs shown in Table 9.5.

TABLE 9.5 Operation of a Jayant quantizer.

n	Δ_n	Input	Output Level	Output	Error	Update Equation
0	0.5	0.1	0	0.25	0.15	$\Delta_1 = M_0 \times \Delta_0$
1	0.4	−0.2	4	−0.2	0.0	$\Delta_2 = M_4 \times \Delta_1$
2	0.32	0.2	0	0.16	0.04	$\Delta_3 = M_0 \times \Delta_2$
3	0.256	0.1	0	0.128	0.028	$\Delta_4 = M_0 \times \Delta_3$
4	0.2048	−0.3	5	−0.3072	−0.0072	$\Delta_5 = M_5 \times \Delta_4$
5	0.1843	0.1	0	0.0922	−0.0078	$\Delta_6 = M_0 \times \Delta_5$
6	0.1475	0.2	1	0.2212	0.0212	$\Delta_7 = M_1 \times \Delta_6$
7	0.1328	0.5	3	0.4646	−0.0354	$\Delta_8 = M_3 \times \Delta_7$
8	0.1594	0.9	3	0.5578	−0.3422	$\Delta_9 = M_3 \times \Delta_8$
9	0.1913	1.5	3	0.6696	−0.8304	$\Delta_{10} = M_3 \times \Delta_9$
10	0.2296	1.0	3	0.8036	0.1964	$\Delta_{11} = M_3 \times \Delta_{10}$
11	0.2755	0.9	3	0.9643	0.0643	$\Delta_{12} = M_3 \times \Delta_{11}$

Notice how the quantizer adapts to the input. In the beginning of the sequence, the input values are mostly small, and the quantizer step size becomes progressively smaller, providing better and better estimates of the input. At the end of the sample sequence, the input values are large and the step size becomes progressively bigger. However, the size of the error is quite large during the transition. This means that if the input was changing rapidly, which would happen if we had a high-frequency input, such transition situations would be much more likely to occur, and the quantizer would not function very well. However, in cases where the statistics of the input change slowly, the quantizer could adapt to the input. As most natural sources such as speech and images tend to be correlated, their values do not change drastically from sample to sample. Even when some of this structure is removed through some transformation, the residual structure is generally enough for the Jayant quantizer (or some variation of it) to function quite effectively. ♦

The step size in the initial part of the sequence in this example is progressively getting smaller. We can easily conceive of situations where the input values would be small for a long period. Such a situation could occur during a silence period in speech-encoding systems,

or while encoding a dark background in image-encoding systems. If the step size continues to shrink for an extended period of time, in a finite precision system it would result in a value of zero. This would be catastrophic, effectively replacing the quantizer with a zero output device. Usually, a minimum value $\Delta_{\min}$ is defined, and the step size is not allowed to go below this value to prevent this from happening. Similarly, if we get a sequence of large values, the step size could increase to a point that, when we started getting smaller values, the quantizer would not be able to adapt fast enough. To prevent this from happening, a maximum value $\Delta_{\max}$ is defined, and the step size is not allowed to increase beyond this value.

The adaptivity of the Jayant quantizer depends on the values of the multipliers. The further the multiplier values are from unity, the more adaptive the quantizer. However, if the adaptation algorithm reacts too fast, this could lead to instability. So how do we go about selecting the multipliers?

First of all, we know that the multipliers correponding to the inner levels are less than one, and the multipliers for the outer levels are greater than one. If the input process is stationary and P_k represents the probability of being in quantizer interval k (generally estimated by using a fixed quantizer for the input data), then we can impose a stability criterion for the Jayant quantizer based on our requirement that once the quantizer is matched to the input, the product of the expansions and contractions are equal to unity. That is, if n_k is the number of times the input falls in the kth interval,

$$\prod_{k=0}^{M} M_k^{n_k} = 1. \tag{9.22}$$

Taking the Nth root of both sides (where N is the total number of inputs) we obtain

$$\prod_{k=0}^{M} M_k^{\frac{n_k}{N}} = 1,$$

or

$$\prod_{k=0}^{M} M_k^{P_k} = 1 \tag{9.23}$$

where we have assumed that $P_k = n_k/N$.

There are an infinite number of multiplier values that would satisfy Equation (9.23). One way to restrict this number is to impose some structure on the multipliers by requiring them to be of the form

$$M_k = \gamma^{l_k} \tag{9.24}$$

where γ is a number greater than one and l_k takes on only integer values [111, 112]. If we substitute this expression for M_k into Equation (9.23), we get

$$\prod_{k=0}^{M} \gamma^{l_k P_k} = 1, \tag{9.25}$$

which implies that

$$\sum_{k=0}^{M} l_k P_k = 0. \tag{9.26}$$

The final step is the selection of γ, which involves a significant amount of creativity. The value we pick for γ determines how fast the quantizer will respond to changing statistics. A large value of γ will result in faster adaptation, while a smaller value of γ will result in greater stability.

Example 9.5.4:

Suppose we have to obtain the multiplier functions for a 2-bit quantizer with input probabilities $P_0 = 0.8$, $P_1 = 0.2$. First, note that the multiplier value for the inner level has to be less than 1. Therefore, l_0 is less than 0. If we pick $l_0 = -1$ and $l_1 = 4$, this would satisfy Equation (9.26), while making M_0 less than 1 and M_1 greater than 1. Finally, we need to pick a value for γ.

In Figure 9.17 we see the effect of using different values of γ in a rather extreme example. The input is a square wave that switches between 0 and 1 every 30 samples. The input is quantized using a 2-bit Jayant quantizer. We have used $l_0 = -1$ and $l_1 = 2$. Notice what happens when the input switches from 0 to 1. At first the input falls in the outer level of the quantizer, and the step size increases. This process continues until Δ is just greater than 1. If γ is close to 1, Δ has been increasing quite slowly and should have a value close to 1 right before its value increases to greater than 1. Therefore, the output at this point is close to 1.5. When Δ becomes greater than 1, the input falls in the inner level, and if γ is close to 1, the output suddenly drops to about 0.5. The step size now decreases until it is just

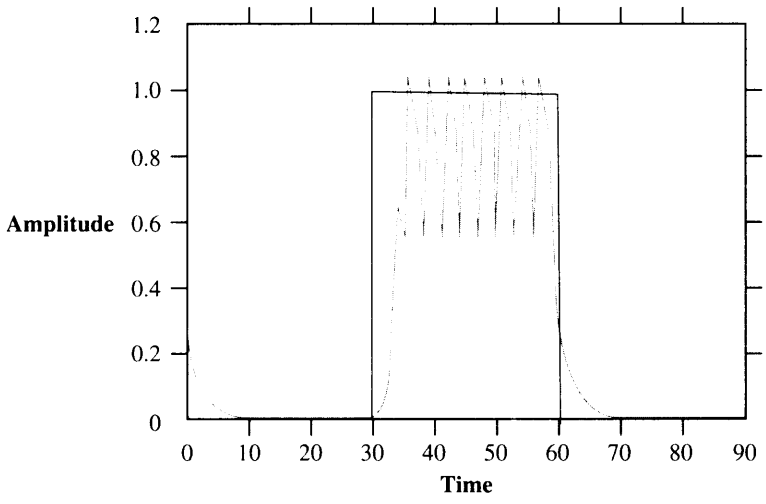

FIGURE 9.17 Effect of γ on the performance of the Jayant quantizer.

below 1, and the process repeats, causing the "ringing" seen in Figure 9.17. As γ increases, the quantizer adapts more rapidly, and the magnitude of the ringing effect decreases. The reason for the decrease is that right before the value of Δ increases above 1, its value is much smaller than 1, and subsequently the output value is much smaller than 1.5. When Δ increases beyond 1, it may increase by a significant amount, so the inner level may be much greater than 0.5. These two effects together compress the ringing phenomenon. Looking at this phenomenon, we can see that it may have been better to have two adaptive strategies, one for when the input is changing rapidly, as in the case of the transitions between 0 and 1, and one for when the input is constant, or nearly so. We will explore this approach further when we describe the quantizer used in the CCITT standard G.726. ♦

When selecting multipliers for a Jayant quantizer, the best quantizers expand more rapidly than they contract. This makes sense when we consider that, when the input falls into the outer levels of the quantizer, it is incurring overload error, which is essentially unbounded. This situation needs to be mitigated with dispatch. On the other hand, when the input falls in the inner levels, the noise incurred is granular noise, which is bounded and therefore may be more tolerable. Finally, the discussion of the Jayant quantizer was motivated by the need for robustness in the face of changing input statistics. Let us repeat the earlier experiment with changing input variance and distributions and see the performance of the Jayant quantizer compared to the *pdf*-optimized quantizer. The results for these experiments are presented in Figure 9.18.

Notice how flat the performance curve is. While the performance of the Jayant quantizer is much better than the nonadaptive uniform quantizer over a wide range of input variances, at the point where the input variance and design variance agree, the performance of the nonadaptive quantizer is significantly better than the performance of the Jayant quantizer.

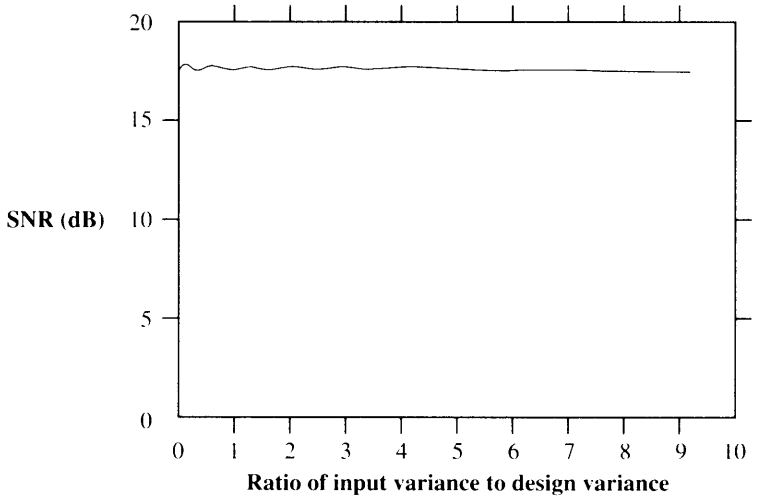

FIGURE 9.18 Performance of the Jayant quantizer for different input variances.

This means that if we know the input statistics and we are reasonably certain that the input statistics will not change over time, it is better to design for those statistics than to design an adaptive system.

9.6 Nonuniform Quantization

As we can see from Figure 9.10, if the input distribution has more mass near the origin, the input is more likely to fall in the inner levels of the quantizer. Recall that in lossless compression, in order to minimize the *average* number of bits per input symbol, we assigned shorter codewords to symbols that occurred with higher probability and longer codewords to symbols that occurred with lower probability. In an analogous fashion, in order to decrease the average distortion, we can try to approximate the input better in regions of high probability, perhaps at the cost of worse approximations in regions of lower probability. We can do this by making the quantization intervals smaller in those regions that have more probability mass. If the source distribution is like the distribution shown in Figure 9.10, we would have smaller intervals near the origin. If we wanted to keep the number of intervals constant, this would mean we would have larger intervals away from the origin. A quantizer that has nonuniform intervals is called a *nonuniform quantizer*. An example of a nonuniform quantizer is shown in Figure 9.19.

Notice that the intervals closer to zero are smaller. Hence the maximum value that the quantizer error can take on is also smaller, resulting in a better approximation. We pay for this improvement in accuracy at lower input levels by incurring larger errors when the input falls in the outer intervals. However, as the probability of getting smaller input values is much higher than getting larger signal values, on the average the distortion will be lower than if we had a uniform quantizer. While a nonuniform quantizer provides lower average distortion, the design of nonuniform quantizers is also somewhat more complex. However, the basic idea is quite straightforward: find the decision boundaries and reconstruction levels that minimize the mean squared quantization error. We look at the design of nonuniform quantizers in more detail in the following sections.

9.6.1 *pdf*-Optimized Quantization

A direct approach for locating the best nonuniform quantizer, if we have a probability model for the source, is to find the $\{b_i\}$ and $\{y_i\}$ that minimize Equation (9.3). Setting the derivative of Equation (9.3) with respect to y_j to zero, and solving for y_j, we get

$$y_j = \frac{\int_{b_{j-1}}^{b_j} x f_X(x) dx}{\int_{b_{j-1}}^{b_j} f_X(x) dx}. \tag{9.27}$$

The output point for each quantization interval is the centroid of the probability mass in that interval. Taking the derivative with respect to b_j and setting it equal to zero, we get an expression for b_j as

$$b_j = \frac{y_{j+1} + y_j}{2}. \tag{9.28}$$

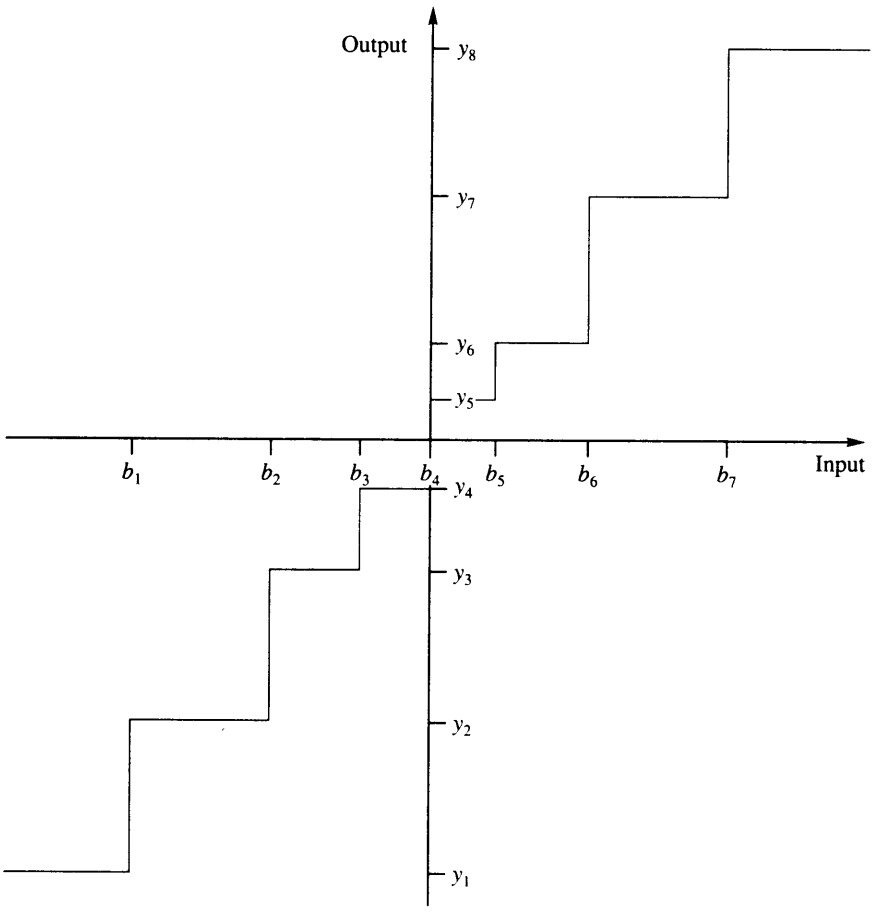

FIGURE 9.19 A nonuniform midrise quantizer.

The decision boundary is simply the midpoint of the two neighboring reconstruction levels. Solving these two equations will give us the values for the reconstruction levels and decision boundaries that minimize the mean squared quantization error. Unfortunately, to solve for y_j, we need the values of b_j and b_{j-1}, and to solve for b_j, we need the values of y_{j+1} and y_j. In a 1960 paper, Joel Max [108] showed how to solve the two equations iteratively. The same approach was described by Stuart P. Lloyd in a 1957 internal Bell Labs memorandum. Generally, credit goes to whomever publishes first, but in this case, because much of the early work in quantization was done at Bell Labs, Lloyd's work was given due credit and the algorithm became known as the Lloyd-Max algorithm. However, the story does not end (begin?) there. Allen Gersho [113] points out that the same algorithm was published by Lukaszewicz and Steinhaus in a Polish journal in 1955 [114]! Lloyd's paper remained unpublished until 1982, when it was finally published in a special issue of the *IEEE Transactions on Information Theory* devoted to quantization [115].

To see how this algorithm works, let us apply it to a specific situation. Suppose we want to design an M-level symmetric midrise quantizer. To define our symbols, we will use Figure 9.20. From the figure, we see that in order to design this quantizer, we need to obtain the reconstruction levels $\{y_1, y_2, \ldots, y_{\frac{M}{2}}\}$ and the decision boundaries $\{b_1, b_2, \ldots, b_{\frac{M}{2}-1}\}$. The reconstruction levels $\{y_{-1}, y_{-2}, \ldots, y_{-\frac{M}{2}}\}$ and the decision boundaries $\{b_{-1}, b_{-2}, \ldots, b_{-(\frac{M}{2}-1)}\}$ can be obtained through symmetry, the decision boundary b_0 is zero, and the decision boundary $b_{\frac{M}{2}}$ is simply the largest value the input can take on (for unbounded inputs this would be ∞).

Let us set j equal to 1 in Equation (9.27):

$$y_1 = \frac{\int_{b_0}^{b_1} x f_X(x)dx}{\int_{b_0}^{b_1} f_X(x)dx}. \tag{9.29}$$

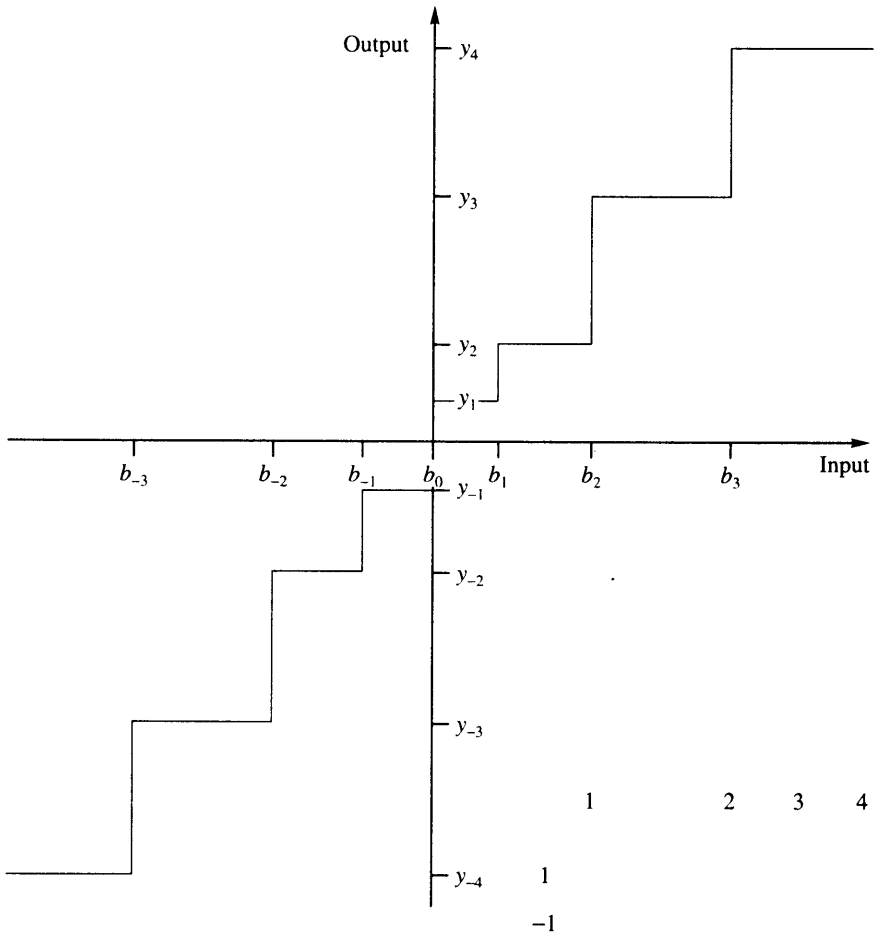

FIGURE 9.20 A nonuniform midrise quantizer.

As b_0 is known to be 0, we have two unknowns in this equation, b_1 and y_1. We make a guess at y_1 and later we will try to refine this guess. Using this guess in Equation (9.29), we numerically find the value of b_1 that satisfies Equation (9.29). Setting j equal to 1 in Equation (9.28), and rearranging things slightly, we get

$$y_2 = 2b_1 + y_1 \tag{9.30}$$

from which we can compute y_2. This value of y_2 can then be used in Equation (9.27) with $j = 2$ to find b_2, which in turn can be used to find y_3. We continue this process, until we obtain a value for $\{y_1, y_2, \ldots, y_{\frac{M}{2}}\}$ and $\{b_1, b_2, \ldots, b_{\frac{M}{2}-1}\}$. Note that the accuracy of all the values obtained to this point depends on the quality of our initial estimate of y_1. We can check this by noting that $y_{\frac{M}{2}}$ is the centroid of the probability mass of the interval $[b_{\frac{M}{2}-1}, b_{\frac{M}{2}}]$. We know $b_{\frac{M}{2}}$ from our knowledge of the data. Therefore, we can compute the integral

$$y_{\frac{M}{2}} = \frac{\int_{b_{\frac{M}{2}-1}}^{b_{\frac{M}{2}}} x f_X(x) dx}{\int_{b_{\frac{M}{2}-1}}^{b_{\frac{M}{2}}} f_X(x) dx} \tag{9.31}$$

and compare it with the previously computed value of $y_{\frac{M}{2}}$. If the difference is less than some tolerance threshold, we can stop. Otherwise, we adjust the estimate of y_1 in the direction indicated by the sign of the difference and repeat the procedure.

Decision boundaries and reconstruction levels for various distributions and number of levels generated using this procedure are shown in Table 9.6. Notice that the distributions that have heavier tails also have larger outer step sizes. However, these same quantizers have smaller inner step sizes because they are more heavily peaked. The SNR for these quantizers is also listed in the table. Comparing these values with those for the *pdf*-optimized uniform quantizers, we can see a significant improvement, especially for distributions further away from the uniform distribution. Both uniform and nonuniform *pdf*-optimized, or Lloyd-Max,

TABLE 9.6 Quantizer boundary and reconstruction levels for nonuniform Gaussian and Laplacian quantizers.

	Gaussian			Laplacian		
Levels	b_i	y_i	SNR	b_i	y_i	SNR
4	0.0	0.4528		0.0	0.4196	
	0.9816	1.510	9.3 dB	1.1269	1.8340	7.54 dB
6	0.0	0.3177		0.0	0.2998	
	0.6589	1.0		0.7195	1.1393	
	1.447	1.894	12.41 dB	1.8464	2.5535	10.51 dB
8	0.0	0.2451		0.0	0.2334	
	0.7560	0.6812		0.5332	0.8330	
	1.050	1.3440		1.2527	1.6725	
	1.748	2.1520	14.62 dB	2.3796	3.0867	12.64 dB

quantizers have a number of interesting properties. We list these properties here (their proofs can be found in [116, 117, 118]):

- **Property 1:** The mean values of the input and output of a Lloyd-Max quantizer are equal.
- **Property 2:** For a given Lloyd-Max quantizer, the variance of the output is always less than or equal to the variance of the input.
- **Property 3:** The mean squared quantization error for a Lloyd-Max quantizer is given by

$$\sigma_q^2 = \sigma_x^2 - \sum_{j=1}^{M} y_j^2 P[b_{j-1} \leq X < b_j] \tag{9.32}$$

  where σ_x^2 is the variance of the quantizer input, and the second term on the right-hand side is the second moment of the output (or variance if the input is zero mean).
- **Property 4:** Let N be the random variable corresponding to the quantization error. Then for a given Lloyd-Max quantizer,

$$E[XN] = -\sigma_q^2. \tag{9.33}$$

- **Property 5:** For a given Lloyd-Max quantizer, the quantizer output and the quantization noise are orthogonal:

$$E[Q(X)N \mid b_0, b_1, \ldots, b_M] = 0. \tag{9.34}$$

Mismatch Effects

As in the case of uniform quantizers, the *pdf*-optimized nonuniform quantizers also have problems when the assumptions underlying their design are violated. In Figure 9.21 we show the effects of variance mismatch on a 4-bit Laplacian nonuniform quantizer.

This mismatch effect is a serious problem because in most communication systems the input variance can change considerably over time. A common example of this is the telephone system. Different people speak with differing amounts of loudness into the telephone. The quantizer used in the telephone system needs to be quite robust to the wide range of input variances in order to provide satisfactory service.

One solution to this problem is the use of adaptive quantization to match the quantizer to the changing input characteristics. We have already looked at adaptive quantization for the uniform quantizer. Generalizing the uniform adaptive quantizer to the nonuniform case is relatively straightforward, and we leave that as a practice exercise (see Problem 8). A somewhat different approach is to use a nonlinear mapping to flatten the performance curve shown in Figure 9.21. In order to study this approach, we need to view the nonuniform quantizer in a slightly different manner.

9.6.2 Companded Quantization

Instead of making the step size small, we could make the interval in which the input lies with high probability large—that is, expand the region in which the input lands with high

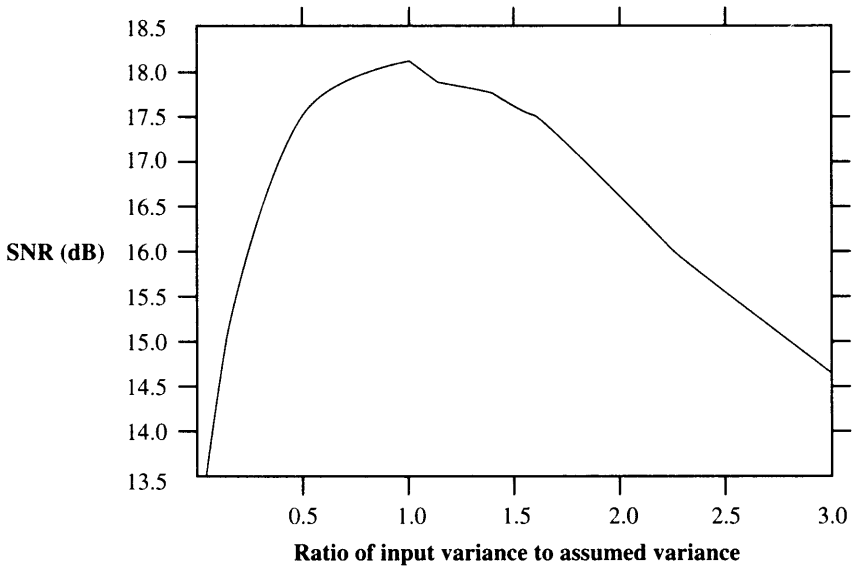

FIGURE 9.21 Effect of mismatch on nonuniform quantization.

probability in proportion to the probability with which the input lands in this region. This is the idea behind companded quantization. This quantization approach can be represented by the block diagram shown in Figure 9.22. The input is first mapped through a *compressor* function. This function "stretches" the high-probability regions close to the origin, and correspondingly "compresses" the low-probability regions away from the origin. Thus, regions close to the origin in the input to the compressor occupy a greater fraction of the total region covered by the compressor. If the output of the compressor function is quantized using a uniform quantizer, and the quantized value transformed via an *expander* function,

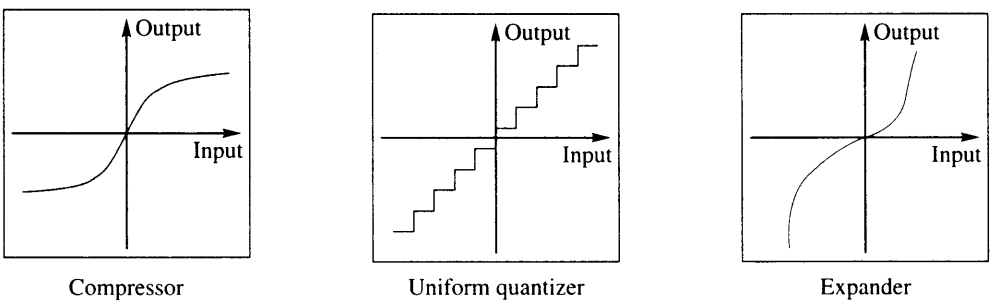

FIGURE 9.22 Block diagram for log companded quantization.

the overall effect is the same as using a nonuniform quantizer. To see this, we devise a simple compander and see how the process functions.

Example 9.6.1:

Suppose we have a source that can be modeled as a random variable taking values in the interval $[-4, 4]$ with more probability mass near the origin than away from it. We want to quantize this using the quantizer of Figure 9.3. Let us try to flatten out this distribution using the following compander, and then compare the companded quantization with straightforward uniform quantization. The compressor characteristic we will use is given by the following equation:

$$c(x) = \begin{cases} 2x & \text{if } -1 \le x \le 1 \\ \frac{2x}{3} + \frac{4}{3} & x > 1 \\ \frac{2x}{3} - \frac{4}{3} & x < -1. \end{cases} \tag{9.35}$$

The mapping is shown graphically in Figure 9.23. The inverse mapping is given by

$$c^{-1}(x) = \begin{cases} \frac{x}{2} & \text{if } -2 \le x \le 2 \\ \frac{3x}{2} - 2 & x > 2 \\ \frac{3x}{2} + 2 & x < -2. \end{cases} \tag{9.36}$$

The inverse mapping is shown graphically in Figure 9.24.

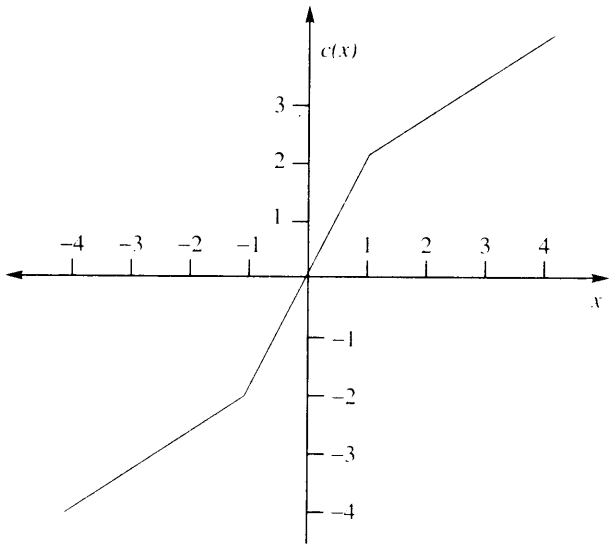

FIGURE 9.23 Compressor mapping.

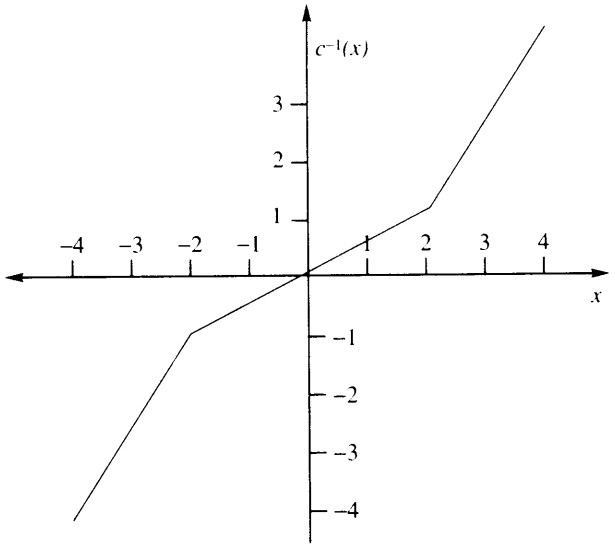

FIGURE 9. 24 Expander mapping.

Let's see how using these mappings affects the quantization error both near and far from the origin. Suppose we had an input of 0.9. If we quantize directly with the uniform quantizer, we get an output of 0.5, resulting in a quantization error of 0.4. If we use the companded quantizer, we first use the compressor mapping, mapping the input value of 0.9 to 1.8. Quantizing this with the same uniform quantizer results in an output of 1.5, with an apparent error of 0.3. The expander then maps this to the final reconstruction value of 0.75, which is 0.15 away from the input. Comparing 0.15 with 0.4, we can see that relative to the input we get a substantial reduction in the quantization error. In fact, for all values in the interval $[-1, 1]$, we will not get any increase in the quantization error, and for most values we will get a decrease in the quantization error (see Problem 6 at the end of this chapter). Of course, this will not be true for the values outside the $[-1, 1]$ interval. Suppose we have an input of 2.7. If we quantized this directly with the uniform quantizer, we would get an output of 2.5, with a corresponding error of 0.2. Applying the compressor mapping, the value of 2.7 would be mapped to 3.13, resulting in a quantized value of 3.5. Mapping this back through the expander, we get a reconstructed value of 3.25, which differs from the input by 0.55.

As we can see, the companded quantizer effectively works like a nonuniform quantizer with smaller quantization intervals in the interval $[-1, 1]$ and larger quantization intervals outside this interval. What is the effective input-output map of this quantizer? Notice that all inputs in the interval $[0, 0.5]$ get mapped into the interval $[0, 1]$, for which the quantizer output is 0.5, which in turn corresponds to the reconstruction value of 0.25. Essentially, all values in the interval $[0, 0.5]$ are represented by the value 0.25. Similarly, all values in

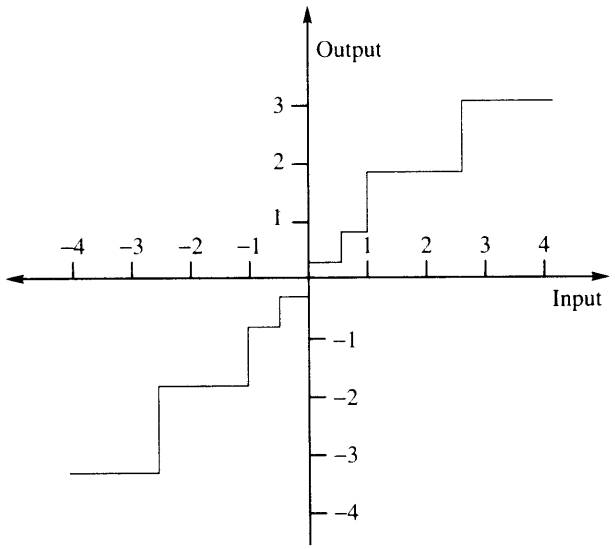

FIGURE 9. 25 Nonuniform companded quantizer.

the interval [0.5, 1] are represented by the value 0.75, and so on. The effective quantizer input-output map is shown in Figure 9.25. ♦

If we bound the source output by some value x_{max}, any nonuniform quantizer can always be represented as a companding quantizer. Let us see how we can use this fact to come up with quantizers that are robust to mismatch. First we need to look at some of the properties of high-rate quantizers, or quantizers with a large number of levels.

Define

$$\Delta_k = b_k - b_{k-1}. \tag{9.37}$$

If the number of levels is high, then the size of each quantization interval will be small, and we can assume that the *pdf* of the input $f_X(x)$ is essentially constant in each quantization interval. Then

$$f_X(x) = f_X(y_k) \qquad \text{if } b_{k-1} \le x < b_k. \tag{9.38}$$

Using this we can rewrite Equation (9.3) as

$$\sigma_q^2 = \sum_{i=1}^{M} f_X(y_i) \int_{b_{i-1}}^{b_i} (x - y_i)^2 dx \tag{9.39}$$

$$= \frac{1}{12} \sum_{i=1}^{M} f_X(y_i) \Delta_i^3. \tag{9.40}$$

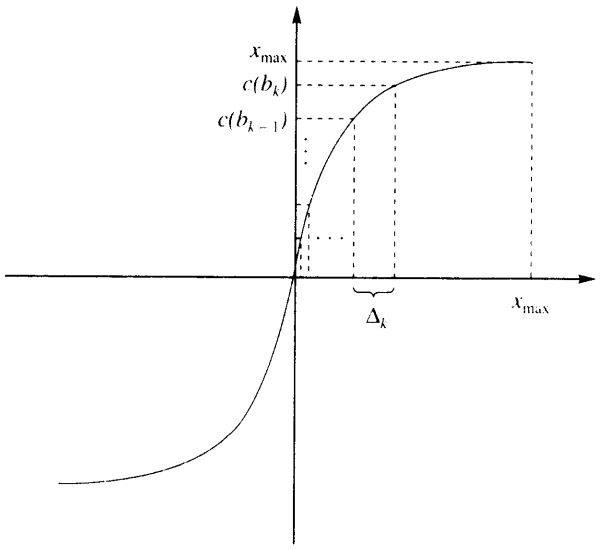

FIGURE 9.26 A compressor function.

Armed with this result, let us return to companded quantization. Let $c(x)$ be a companding characteristic for a symmetric quantizer, and let $c'(x)$ be the derivative of the compressor characteristic with respect to x. If the rate of the quantizer is high, that is, if there are a large number of levels, then within the kth interval, the compressor characteristic can be approximated by a straight line segment (see Figure 9.26), and we can write

$$c'(y_k) = \frac{c(b_k) - c(b_{k-1})}{\Delta_k}. \tag{9.41}$$

From Figure 9.26 we can also see that $c(b_k) - c(b_{k-1})$ is the step size of a uniform M-level quantizer. Therefore,

$$c(b_k) - c(b_{k-1}) = \frac{2x_{max}}{M}. \tag{9.42}$$

Substituting this into Equation (9.41) and solving for Δ_k, we get

$$\Delta_k = \frac{2x_{max}}{Mc'(y_k)}. \tag{9.43}$$

Finally, substituting this expression for Δ_k into Equation (9.40), we get the following relationship between the quantizer distortion, the *pdf* of the input, and the compressor characteristic:

$$\begin{aligned}\sigma_q^2 &= \frac{1}{12}\sum_{i=1}^{M} f_X(y_i)\left(\frac{2x_{\max}}{Mc'(y_i)}\right)^3 \\ &= \frac{x_{\max}^2}{3M^2}\sum_{i=1}^{M}\frac{f_X(y_i)}{c'^2(y_i)}\cdot\frac{2x_{\max}}{Mc'(y_i)} \\ &= \frac{x_{\max}^2}{3M^2}\sum_{i=1}^{M}\frac{f_X(y_i)}{c'^2(y_i)}\Delta_i \end{aligned} \tag{9.44}$$

which for small Δ_i can be written as

$$\sigma_q^2 = \frac{x_{\max}^2}{3M^2}\int_{-x_{\max}}^{x_{\max}}\frac{f_X(x)}{(c'(x))^2}dx. \tag{9.45}$$

This is a famous result, known as the Bennett integral after its discoverer, W.R. Bennett [119], and it has been widely used to analyze quantizers. We can see from this integral that the quantizer distortion is dependent on the *pdf* of the source sequence. However, it also tells us how to get rid of this dependence. Define

$$c'(x) = \frac{x_{\max}}{\alpha|x|}, \tag{9.46}$$

where α is a constant. From the Bennett integral we get

$$\sigma_q^2 = \frac{x_{\max}^2}{3M^2}\frac{\alpha^2}{x_{\max}^2}\int_{-x_{\max}}^{x_{\max}} x^2 f_X(x)dx \tag{9.47}$$

$$= \frac{\alpha^2}{3M^2}\sigma_x^2 \tag{9.48}$$

where

$$\sigma_x^2 = \int_{-x_{\max}}^{x_{\max}} x^2 f_X(x)dx. \tag{9.49}$$

Substituting the expression for σ_q^2 into the expression for SNR, we get

$$\text{SNR} = 10\log_{10}\frac{\sigma_x^2}{\sigma_q^2} \tag{9.50}$$

$$= 10\log_{10}(3M^2) - 20\log_{10}\alpha \tag{9.51}$$

which is independent of the input *pdf*. This means that if we use a compressor characteristic whose derivative satisfies Equation (9.46), then regardless of the input variance, the signal-to-noise ratio will remain constant. This is an impressive result. However, we do need some caveats.

Notice that we are not saying that the mean squared quantization error is independent of the quantizer input. It is not, as is clear from Equation (9.48). Remember also that this

result is valid as long as the underlying assumptions are valid. When the input variance is very small, our assumption about the *pdf* being constant over the quantization interval is no longer valid, and when the variance of the input is very large, our assumption about the input being bounded by x_{max} may no longer hold.

With fair warning, let us look at the resulting compressor characteristic. We can obtain the compressor characteristic by integrating Equation (9.46):

$$c(x) = x_{max} + \beta \log \frac{|x|}{x_{max}} \tag{9.52}$$

where β is a constant. The only problem with this compressor characteristic is that it becomes very large for small x. Therefore, in practice we approximate this characteristic with a function that is linear around the origin and logarithmic away from it.

Two companding characteristics that are widely used today are μ-law companding and A-law companding. The μ-law compressor function is given by

$$c(x) = x_{max} \frac{\ln\left(1 + \mu \frac{|x|}{x_{max}}\right)}{\ln(1+\mu)} \text{sgn}(x). \tag{9.53}$$

The expander function is given by

$$c^{-1}(x) = \frac{x_{max}}{\mu}[(1+\mu)^{\frac{|x|}{x_{max}}} - 1]\text{sgn}(x). \tag{9.54}$$

This companding characteristic with $\mu = 255$ is used in the telephone systems in North America and Japan. The rest of the world uses the A-law characteristic, which is given by

$$c(x) = \begin{cases} \frac{A|x|}{1+\ln A}\text{sgn}(x) & 0 \le \frac{|x|}{x_{max}} \le \frac{1}{A} \\ x_{max}\frac{1+\ln\frac{A|x|}{x_{max}}}{1+\ln A}\text{sgn}(x) & \frac{1}{A} \le \frac{|x|}{x_{max}} \le 1 \end{cases} \tag{9.55}$$

and

$$c^{-1}(x) = \begin{cases} \frac{|x|}{A}(1+\ln A) & 0 \le \frac{|x|}{x_{max}} \le \frac{1}{1+\ln A} \\ \frac{x_{max}}{A}\exp\left[\frac{|x|}{x_{max}}(1+\ln A) - 1\right] & \frac{1}{1+\ln A} \le \frac{|x|}{x_{max}} \le 1. \end{cases} \tag{9.56}$$

9.7 Entropy-Coded Quantization

In Section 9.3 we mentioned three tasks: selection of boundary values, selection of reconstruction levels, and selection of codewords. Up to this point we have talked about accomplishment of the first two tasks, with the performance measure being the mean squared quantization error. In this section we will look at accomplishing the third task, assigning codewords to the quantization interval. Recall that this becomes an issue when we use variable-length codes. In this section we will be looking at the latter situation, with the rate being the performance measure.

We can take two approaches to the variable-length coding of quantizer outputs. We can redesign the quantizer by taking into account the fact that the selection of the decision boundaries will affect the rate, or we can keep the design of the quantizer the same

(i.e., Lloyd-Max quantization) and simply entropy-code the quantizer output. Since the latter approach is by far the simpler one, let's look at it first.

9.7.1 Entropy Coding of Lloyd-Max Quantizer Outputs

The process of trying to find the optimum quantizer for a given number of levels and rate is a rather difficult task. An easier approach to incorporating entropy coding is to design a quantizer that minimizes the msqe, that is, a Lloyd-Max quantizer, then entropy-code its output.

In Table 9.7 we list the output entropies of uniform and nonuniform Lloyd-Max quantizers. Notice that while the difference in rate for lower levels is relatively small, for a larger number of levels, there can be a substantial difference between the fixed-rate and entropy-coded cases. For example, for 32 levels a fixed-rate quantizer would require 5 bits per sample. However, the entropy of a 32-level uniform quantizer for the Laplacian case is 3.779 bits per sample, which is more than 1 bit less. Notice that the difference between the fixed rate and the uniform quantizer entropy is generally greater than the difference between the fixed rate and the entropy of the output of the nonuniform quantizer. This is because the nonuniform quantizers have smaller step sizes in high-probability regions and larger step sizes in low-probability regions. This brings the probability of an input falling into a low-probability region and the probability of an input falling in a high-probability region closer together. This, in turn, raises the output entropy of the nonuniform quantizer with respect to the uniform quantizer. Finally, the closer the distribution is to being uniform, the less difference in the rates. Thus, the difference in rates is much less for the quantizer for the Gaussian source than the quantizer for the Laplacian source.

9.7.2 Entropy-Constrained Quantization ★

Although entropy coding the Lloyd-Max quantizer output is certainly simple, it is easy to see that we could probably do better if we take a fresh look at the problem of quantizer

TABLE 9.7 Output entropies in bits per sample for minimum mean squared error quantizers.

Number of Levels	Gaussian		Laplacian	
	Uniform	Nonuniform	Uniform	Nonuniform
4	1.904	1.911	1.751	1.728
6	2.409	2.442	2.127	2.207
8	2.759	2.824	2.394	2.479
16	3.602	3.765	3.063	3.473
32	4.449	4.730	3.779	4.427

design, this time with the entropy as a measure of rate rather than the alphabet size. The entropy of the quantizer output is given by

$$H(Q) = -\sum_{i=1}^{M} P_i \log_2 P_i \tag{9.57}$$

where P_i is the probability of the input to the quantizer falling in the ith quantization interval and is given by

$$P_i = \int_{b_{i-1}}^{b_i} f_X(x)dx. \tag{9.58}$$

Notice that the selection of the representation values $\{y_j\}$ has no effect on the rate. This means that we can select the representation values solely to minimize the distortion. However, the selection of the boundary values affects both the rate and the distortion. Initially, we found the reconstruction levels and decision boundaries that minimized the distortion, while keeping the rate fixed by fixing the quantizer alphabet size and assuming fixed-rate coding. In an analogous fashion, we can now keep the entropy fixed and try to minimize the distortion. Or, more formally:

> For a given R_o, find the decision boundaries $\{b_j\}$ that minimize σ_q^2 given by Equation (9.3), subject to $H(Q) \leq R_o$.

The solution to this problem involves the solution of the following $M-1$ nonlinear equations [120]:

$$\ln \frac{P_{i-1}}{P_i} = \lambda(y_{k+1} - y_k)(y_{k+1} + y_k - 2b_k) \tag{9.59}$$

where λ is adjusted to obtain the desired rate, and the reconstruction levels are obtained using Equation (9.27). A generalization of the method used to obtain the minimum mean squared error quantizers can be used to obtain solutions for this equation [121]. The process of finding optimum entropy-constrained quantizers looks complex. Fortunately, at higher rates we can show that the optimal quantizer is a uniform quantizer, simplifying the problem. Furthermore, while these results are derived for the high-rate case, it has been shown that the results also hold for lower rates [121].

9.7.3 High-Rate Optimum Quantization ★

At high rates, the design of optimum quantizers becomes simple, at least in theory. Gish and Pierce's work [122] says that at high rates the optimum entropy-coded quantizer is a uniform quantizer. Recall that any nonuniform quantizer can be represented by a compander and a uniform quantizer. Let us try to find the optimum compressor function at high rates that minimizes the entropy for a given distortion. Using the calculus of variations approach, we will construct the functional

$$J = H(Q) + \lambda\sigma_q^2, \tag{9.60}$$

then find the compressor characteristic to minimize it.

For the distortion σ_q^2, we will use the Bennett integral shown in Equation (9.45). The quantizer entropy is given by Equation (9.57). For high rates, we can assume (as we did before) that the *pdf* $f_X(x)$ is constant over each quantization interval Δ_i, and we can replace Equation (9.58) by

$$P_i = f_X(y_i)\Delta_i. \tag{9.61}$$

Substituting this into Equation (9.57), we get

$$H(Q) = -\sum f_X(y_i)\Delta_i \log[f_X(y_i)\Delta_i] \tag{9.62}$$

$$= -\sum f_X(y_i)\log[f_X(y_i)]\Delta_i - \sum f_X(y_i)\log[\Delta_i]\Delta_i \tag{9.63}$$

$$= -\sum f_X(y_i)\log[f_X(y_i)]\Delta_i - \sum f_X(y_i)\log\frac{2x_{\max}/M}{c'(y_i)}\Delta_i \tag{9.64}$$

where we have used Equation (9.43) for Δ_i. For small Δ_i we can write this as

$$H(Q) = -\int f_X(x)\log f_X(x)dx - \int f_X(x)\log\frac{2x_{\max}/M}{c'(x)}dx \tag{9.65}$$

$$= -\int f_X(x)\log f_X(x)dx - \log\frac{2x_{\max}}{M} + \int f_X(x)\log c'(x)dx \tag{9.66}$$

where the first term is the differential entropy of the source $h(X)$. Let's define $g = c'(x)$. Then substituting the value of $H(Q)$ into Equation (9.60) and differentiating with respect to g, we get

$$\int f_X(x)[g^{-1} - 2\lambda\frac{x_{\max}^2}{3M^2}g^{-3}]dx = 0. \tag{9.67}$$

This equation is satisfied if the integrand is zero, which gives us

$$g = \sqrt{\frac{2\lambda}{3}}\frac{x_{\max}}{M} = K(constant). \tag{9.68}$$

Therefore,

$$c'(x) = K \tag{9.69}$$

and

$$c(x) = Kx + \alpha. \tag{9.70}$$

If we now use the boundary conditions $c(0) = 0$ and $c(x_{\max}) = x_{\max}$, we get $c(x) = x$, which is the compressor characteristic for a uniform quantizer. Thus, at high rates the optimum quantizer is a uniform quantizer.

Substituting this expression for the optimum compressor function in the Bennett integral, we get an expression for the distortion for the optimum quantizer:

$$\sigma_q^2 = \frac{x_{\max}^2}{3M^2}. \tag{9.71}$$

Substituting the expression for $c(x)$ in Equation (9.66), we get the expression for the entropy of the optimum quantizer:

$$H(Q) = h(X) - \log \frac{2x_{\max}}{M}. \tag{9.72}$$

Note that while this result provides us with an easy method for designing optimum quantizers, our derivation is only valid if the source *pdf* is entirely contained in the interval $[-x_{\max}, x_{\max}]$, and if the step size is small enough that we can reasonably assume the *pdf* to be constant over a quantization interval. Generally, these conditions can only be satisfied if we have an extremely large number of quantization intervals. While theoretically this is not much of a problem, most of these reconstruction levels will be rarely used. In practice, as mentioned in Chapter 3, entropy coding a source with a large output alphabet is very problematic. One way we can get around this is through the use of a technique called *recursive indexing*.

Recursive indexing is a mapping of a countable set to a collection of sequences of symbols from another set with finite size [76]. Given a countable set $A = \{a_0, a_1, \ldots\}$ and a finite set $B = \{b_0, b_1, \ldots, b_M\}$ of size $M+1$, we can represent any element in A by a sequence of elements in B in the following manner:

1. Take the index i of element a_i of A.
2. Find the quotient m and remainder r of the index i such that
$$i = mM + r.$$
3. Generate the sequence: $\underbrace{b_M b_M \cdots b_M}_{m \text{ times}} b_r$.

B is called the representation set. We can see that given any element in A we will have a unique sequence from B representing it. Furthermore, no representative sequence is a prefix of any other sequence. Therefore, recursive indexing can be viewed as a trivial, uniquely decodable prefix code. The inverse mapping is given by

$$\underbrace{b_M b_M \cdots b_M}_{m \text{ times}} b_r \mapsto a_{mM+r}.$$

Since it is one-to-one, if it is used at the output of the quantizer to convert the index sequence of the quantizer output into the sequence of the recursive indices, the former can be recovered without error from the latter. Furthermore, when the size $M+1$ of the representation set B is chosen appropriately, in effect we can achieve the reduction in the size of the output alphabets that are used for entropy coding.

Example 9.7.1:

Suppose we want to represent the set of nonnegative integers $A = \{0, 1, 2, \ldots\}$ with the representation set $B = \{0, 1, 2, 3, 4, 5\}$. Then the value 12 would be represented by the sequence 5, 5, 2, and the value 16 would be represented by the sequence 5, 5, 5, 1. Whenever the

decoder sees the value 5, it simply adds on the next value until the next value is smaller than 5. For example, the sequence 3, 5, 1, 2, 5, 5, 1, 5, 0 would be decoded as 3, 6, 2, 11, 5. ♦

Recursive indexing is applicable to any representation of a large set by a small set. One way of applying recursive indexing to the problem of quantization is as follows: For a given step size $\Delta > 0$ and a positive integer K, define x_l and x_h as follows:

$$x_l = -\left\lfloor \frac{K-1}{2} \right\rfloor \Delta$$
$$x_h = x_l + (K-1)\Delta$$

where $\lfloor x \rfloor$ is the largest integer not exceeding x. We define a recursively indexed quantizer of size K to be a uniform quantizer with step size Δ and with x_l and x_h being its smallest and largest output levels. (Q defined this way also has 0 as its output level.) The quantization rule Q, for a given input value x, is as follows:

1. If x falls in the interval $(x_l + \frac{\Delta}{2}, x_h - \frac{\Delta}{2})$, then $Q(x)$ is the nearest output level.

2. If x is greater than $x_h - \frac{\Delta}{2}$, see if $x_1 \stackrel{\Delta}{=} x - x_h \in (x_l + \frac{\Delta}{2}, x_h - \frac{\Delta}{2})$. If so, $Q(x) = (x_h, Q(x_1))$. If not, form $x_2 = x - 2x_h$ and do the same as for x_1. This process continues until for some m, $x_m = x - mx_h$ falls in $(x_l + \frac{\Delta}{2}, x_h - \frac{\Delta}{2})$, which will be quantized into

$$Q(x) = (\underbrace{x_h, x_h, \ldots, x_h}_{m \text{ times}}, Q(x_m)). \tag{9.73}$$

3. If x is smaller than $x_l + \frac{\Delta}{2}$, a similar procedure to the above is used; that is, form $x_m = x + mx_l$ so that it falls in $(x_l + \frac{\Delta}{2}, x_h - \frac{\Delta}{2})$, and quantize it to $(x_l, x_l, \ldots, x_l, Q(x_m))$.

In summary, the quantizer operates in two modes: one when the input falls in the range (x_l, x_h), the other when it falls outside of the specified range. The recursive nature in the second mode gives it the name.

We pay for the advantage of encoding a larger set by a smaller set in several ways. If we get a large input to our quantizer, the representation sequence may end up being intolerably large. We also get an increase in the rate. If $H(Q)$ is the entropy of the quantizer output, and γ is the average number of representation symbols per input symbol, then the minimum rate for the recursively indexed quantizer is $\gamma H(Q)$.

In practice, neither cost is too large. We can avoid the problem of intolerably large sequences by adopting some simple strategies for representing these sequences, and the value of γ is quite close to one for reasonable values of M. For Laplacian and Gaussian quantizers, a typical value for M would be 15 [76].

9.8 Summary

The area of quantization is a well-researched area and much is known about the subject. In this chapter, we looked at the design and performance of uniform and nonuniform quantizers for a variety of sources, and how the performance is affected when the assumptions used

in the design process are not correct. When the source statistics are not well known or change with time, we can use an adaptive strategy. One of the more popular approaches to adaptive quantization is the Jayant quantizer. We also looked at the issues involved with entropy-coded quantization.

Further Reading

With an area as broad as quantization, we had to keep some of the coverage rather cursory. However, there is a wealth of information on quantization available in the published literature. The following sources are especially useful for a general understanding of the area:

1. A very thorough coverage of quantization can be found in *Digital Coding of Waveforms*, by N.S. Jayant and P. Noll [123].
2. The paper "Quantization," by A. Gersho, in *IEEE Communication Magazine*, September 1977 [113], provides an excellent tutorial coverage of many of the topics listed here.
3. The original paper by J. Max, "Quantization for Minimum Distortion," *IRE Transactions on Information Theory* [108], contains a very accessible description of the design of *pdf*-optimized quantizers.
4. A thorough study of the effects of mismatch is provided by W. Mauersberger in [124].

9.9 Projects and Problems

1. Show that the derivative of the distortion expression in Equation (9.18) results in the expression in Equation (9.19). You will have to use a result called Leibnitz's rule and the idea of a telescoping series. Leibnitz's rule states that if $a(t)$ and $b(t)$ are monotonic, then

$$\frac{\delta}{\delta t}\int_{a(t)}^{b(t)} f(x,t)dx = \int_{a(t)}^{b(t)} \frac{\delta f(x,t)}{\delta t}dx + f(b(t),t)\frac{\delta b(t)}{\delta t} - f(a(t),t)\frac{\delta a(t)}{\delta t}. \quad (9.74)$$

2. Use the program `falspos` to solve Equation (9.19) numerically for the Gaussian and Laplacian distributions. You may have to modify the function `func` in order to do this.
3. Design a 3-bit uniform quantizer (specify the decision boundaries and representation levels) for a source with a Laplacian *pdf*, with a mean of 3 and a variance of 4.
4. The pixel values in the Sena image are not really distributed uniformly. Obtain a histogram of the image (you can use the `hist_image` routine), and using the fact that the quantized image should be as good an approximation as possible for the original, design 1-, 2-, and 3-bit quantizers for this image. Compare these with the results displayed in Figure 9.7. (For better comparison, you can reproduce the results in the book using the program `uquan_img`.)

5. Use the program `misuquan` to study the effect of mismatch between the input and assumed variances. How do these effects change with the quantizer alphabet size and the distribution type?

6. For the companding quantizer of Example 9.6.1, what are the outputs for the following inputs: $-0.8, 1.2, 0.5, 0.6, 3.2, -0.3$? Compare your results with the case when the input is directly quantized with a uniform quantizer with the same number of levels. Comment on your results.

7. Use the test images Sena and Bookshelf1 to study the trade-offs involved in the selection of block sizes in the forward adaptive quantization scheme described in Example 9.5.2. Compare this with a more traditional forward adaptive scheme in which the variance is estimated and transmitted. The variance information should be transmitted using a uniform quantizer with differing number of bits.

8. Generalize the Jayant quantizer to the nonuniform case. Assume that the input is from a known distribution with unknown variance. Simulate the performance of this quantizer over the same range of ratio of variances as we have done for the uniform case. Compare your results to the fixed nonuniform quantizer and the adaptive uniform quantizer. To get a start on your program, you may wish to use `misnuq.c` and `juquan.c`.

9. Let's look at the rate distortion performance of the various quanitzers.

   (a) Plot the rate-distortion function $R(D)$ for a Gaussian source with mean zero and variance $\sigma_X^2 = 2$.

   (b) Assuming fixed length codewords, compute the rate and distortion for 1, 2, and 3 bit pdf-optimized nonuniform quantizers. Also, assume that X is a Gaussian random variable with mean zero and $\sigma_X^2 = 2$. Plot these values on the same graph with **x**'s.

   (c) For the 2 and 3 bit quantizers, compute the rate and distortion assuming that the quantizer outputs are entropy coded. Plot these on the graph with **o**'s.

Vector Quantization

10.1 Overview

By grouping source outputs together and encoding them as a single block, we can obtain efficient lossy as well as lossless compression algorithms. Many of the lossless compression algorithms that we looked at took advantage of this fact. We can do the same with quantization. In this chapter, several quantization techniques that operate on blocks of data are described. We can view these blocks as vectors, hence the name "vector quantization." We will describe several different approaches to vector quantization. We will explore how to design vector quantizers and how these quantizers can be used for compression.

10.2 Introduction

In the last chapter, we looked at different ways of quantizing the output of a source. In all cases the quantizer inputs were scalar values, and each quantizer codeword represented a single sample of the source output. In Chapter 2 we saw that, by taking longer and longer sequences of input samples, it is possible to extract the structure in the source coder output. In Chapter 4 we saw that, even when the input is random, encoding sequences of samples instead of encoding individual samples separately provides a more efficient code. Encoding sequences of samples is more advantageous in the lossy compression framework as well. By "advantageous" we mean a lower distortion for a given rate, or a lower rate for a given distortion. As in the previous chapter, by "rate" we mean the average number of bits per input sample, and the measures of distortion will generally be the mean squared error and the signal-to-noise ratio.

The idea that encoding sequences of outputs can provide an advantage over the encoding of individual samples was first put forward by Shannon, and the basic results in information

theory were all proved by taking longer and longer sequences of inputs. This indicates that a quantization strategy that works with sequences or blocks of output would provide some improvement in performance over scalar quantization. In other words, we wish to generate a representative set of sequences. Given a source output sequence, we would represent it with one of the elements of the representative set.

In vector quantization we group the source output into blocks or vectors. For example, we can treat L consecutive samples of speech as the components of an L-dimensional vector. Or, we can take a block of L pixels from an image and treat each pixel value as a component of a vector of size or dimension L. This vector of source outputs forms the input to the vector quantizer. At both the encoder and decoder of the vector quantizer, we have a set of L-dimensional vectors called the *codebook* of the vector quantizer. The vectors in this codebook, known as *code-vectors*, are selected to be representative of the vectors we generate from the source output. Each code-vector is assigned a binary index. At the encoder, the input vector is compared to each code-vector in order to find the code-vector closest to the input vector. The elements of this code-vector are the quantized values of the source output. In order to inform the decoder about which code-vector was found to be the closest to the input vector, we transmit or store the binary index of the code-vector. Because the decoder has exactly the same codebook, it can retrieve the code-vector given its binary index. A pictorial representation of this process is shown in Figure 10.1.

Although the encoder may have to perform a considerable amount of computations in order to find the closest reproduction vector to the vector of source outputs, the decoding consists of a table lookup. This makes vector quantization a very attractive encoding scheme for applications in which the resources available for decoding are considerably less than the resources available for encoding. For example, in multimedia applications, considerable

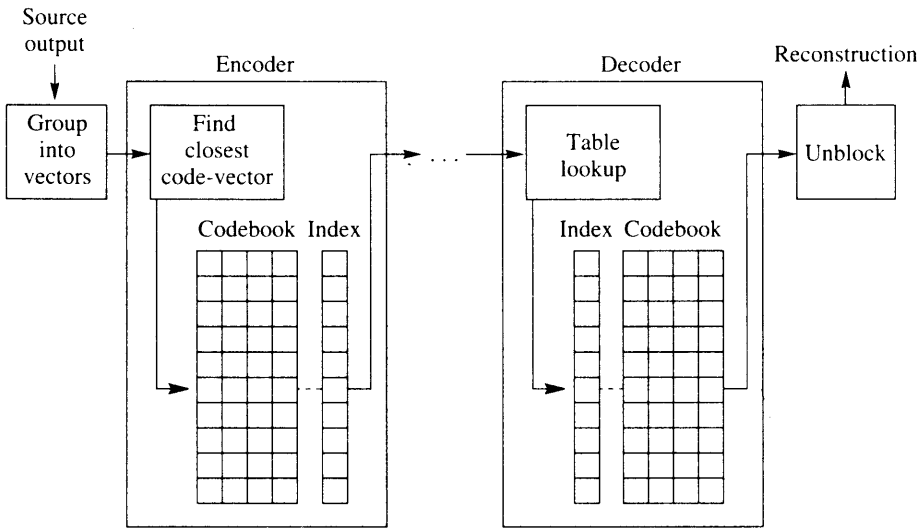

FIGURE 10.1 **The vector quantization procedure.**

computational resources may be available for the encoding operation. However, if the decoding is to be done in software, the amount of computational resources available to the decoder may be quite limited.

Even though vector quantization is a relatively new area, it has developed very rapidly, and now even some of the subspecialties are broad areas of research. If this chapter we will try to introduce you to as much of this fascinating area as we can. If your appetite is whetted by what is available here and you wish to explore further, there is an excellent book by Gersho and Gray [5] devoted to the subject of vector quantization.

Our approach in this chapter is as follows: First, we try to answer the question of why we would want to use vector quantization over scalar quantization. There are several answers to this question, each illustrated through examples. In our discussion, we assume that you are familiar with the material in Chapter 9. We will then turn to one of the most important elements in the design of a vector quantizer, the generation of the codebook. While there are a number of ways of obtaining the vector quantizer codebook, most of them are based on one particular approach, popularly known as the Linde-Buzo-Gray (LBG) algorithm. We devote a considerable amount of time in describing some of the details of this algorithm. Our intent here is to provide you with enough information so that you can write your own programs for design of vector quantizer codebooks. In the software accompanying this book, we have also included programs for designing codebooks that are based on the descriptions in this chapter. If you are not currently thinking of implementing vector quantization routines, you may wish to skip these sections (Sections 10.4.1 and 10.4.2). We follow our discussion of the LBG algorithm with some examples of image compression using codebooks designed with this algorithm, and then with a brief sampling of the many different kinds of vector quantizers. Finally, we describe another quantization strategy, called trellis-coded quantization (TCQ), which, though different in implementation from the vector quantizers, also makes use of the advantage to be gained from operating on sequences.

Before we begin our discussion of vector quantization, let us define some of the terminology we will be using. The amount of compression will be described in terms of the rate, which will be measured in bits per sample. Suppose we have a codebook of size K, and the input vector is of dimension L. In order to inform the decoder of which code-vector was selected, we need to use $\lceil \log_2 K \rceil$ bits. For example, if the codebook contained 256 code-vectors, we would need 8 bits to specify which of the 256 code-vectors had been selected at the encoder. Thus, the number of bits *per vector* is $\lceil \log_2 K \rceil$ bits. As each code-vector contains the reconstruction values for L source output samples, the number of bits *per sample* would be $\frac{\lceil \log_2 K \rceil}{L}$. Thus, the rate for an L-dimensional vector quantizer with a codebook of size K is $\frac{\lceil \log_2 K \rceil}{L}$. As our measure of distortion we will use the mean squared error. When we say that in a codebook $\mathcal{C}$, containing the K code-vectors $\{Y_i\}$, the input vector X is closest to Y_j, we will mean that

$$\|X - Y_j\|^2 \leq \|X - Y_i\|^2 \qquad \text{for all} Y_i \in \mathcal{C} \tag{10.1}$$

where $X = (x_1 x_2 \cdots x_L)$ and

$$\|X\|^2 = \sum_{i=1}^{L} x_i^2. \tag{10.2}$$

The term *sample* will always refer to a scalar value. Thus, when we are discussing compression of images, a sample refers to a single pixel. Finally, the output points of the quantizer are often referred to as *levels*. Thus, when we wish to refer to a quantizer with K output points or code-vectors, we may refer to it as a K-level quantizer.

10.3 Advantages of Vector Quantization over Scalar Quantization

For a given rate (in bits per sample), use of vector quantization results in a lower distortion than when scalar quantization is used at the same rate, for several reasons. In this section we will explore these reasons with examples (for a more theoretical explanation, see [3, 4, 17]).

If the source output is correlated, vectors of source output values will tend to fall in clusters. By selecting the quantizer output points to lie in these clusters, we have a more accurate representation of the source output. Consider the following example.

Example 10.3.1:

In Example 8.5.1, we introduced a source that generates the height and weight of individuals. Suppose the height of these individuals varied uniformly between 40 and 80 inches, and the weight varied uniformly between 40 and 240 pounds. Suppose we were allowed a total of 6 bits to represent each pair of values. We could use 3 bits to quantize the height and 3 bits to quantize the weight. Thus, the weight range between 40 and 240 pounds would be divided into eight intervals of equal width of 25 and with reconstruction values $\{52, 77, \ldots, 227\}$. Similarly, the height range between 40 and 80 inches can be divided into eight intervals of width five, with reconstruction levels $\{42, 47, \ldots, 77\}$. When we look at the representation of height and weight separately, this approach seems reasonable. But let's look at this quantization scheme in two dimensions. We will plot the height values along the x-axis and the weight values along the y-axis. Note that we are not changing anything in the quantization process. The height values are still being quantized to the same eight different values, as are the weight values. The two-dimensional representation of these two quantizers is shown in Figure 10.2.

From the figure we can see that we effectively have a quantizer output for a person who is 80 inches (6 feet 8 inches) tall and weighs 40 pounds, as well as a quantizer output for an individual whose height is 42 inches but weighs more than 200 pounds. Obviously, these outputs will never be used, as is the case for many of the other outputs. A more sensible approach would be to use a quantizer like the one shown in Figure 10.3, where we take account of the fact that the height and weight are correlated. This quantizer has exactly the same number of output points as the quantizer in Figure 10.2; however, the output points are clustered in the area occupied by the input. Using this quantizer, we can no longer quantize the height and weight separately. We have to consider them as the coordinates of a point in two dimensions in order to find the closest quantizer output point. However, this method provides a much finer quantization of the input.

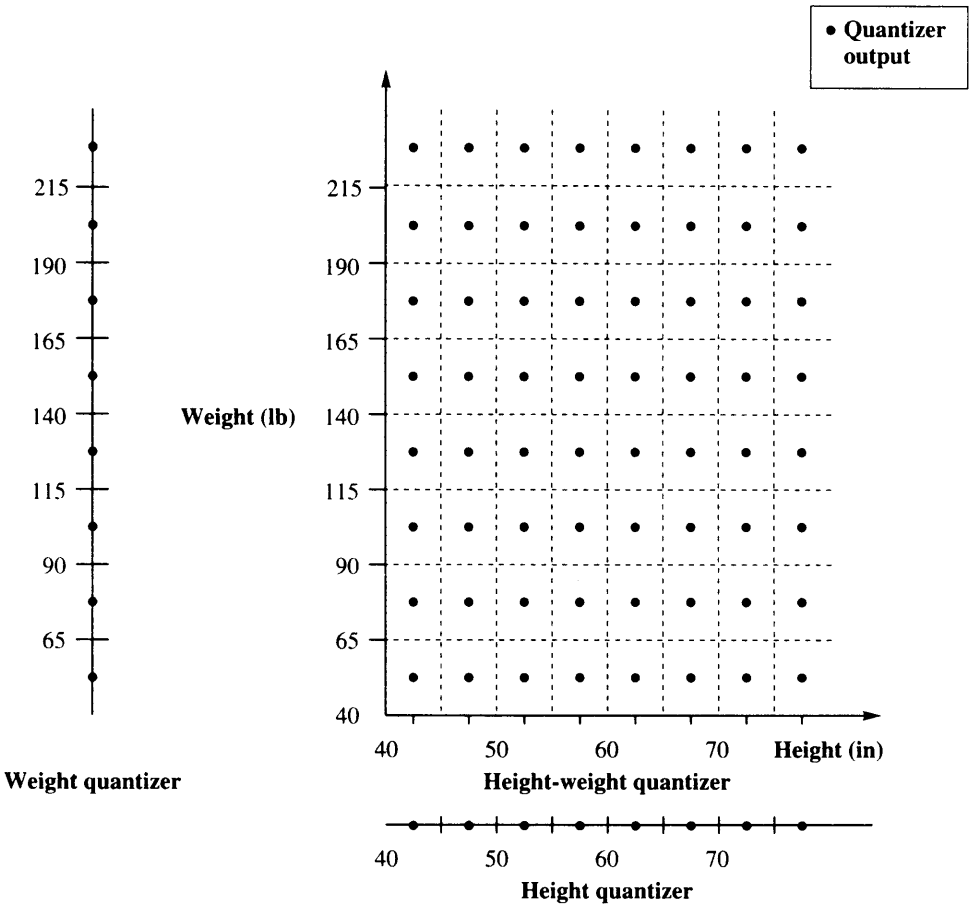

FIGURE 10.2 **The height/weight scalar quantizers when viewed in two dimensions.**

Note that we have not said how we would obtain the locations of the quantizer outputs shown in Figure 10.3. These output points make up the codebook of the vector quantizer, and we will be looking at codebook design in some detail later in this chapter. ♦

We can see from this example that, as in lossless compression, looking at longer sequences of inputs brings out the structure in the source output. This structure can then be used to provide more efficient representations.

We can easily see how structure in the form of correlation between source outputs can make it more efficient to look at sequences of source outputs rather than looking at each sample separately. However, the vector quantizer is also more efficient than the scalar quantizer when the source output values are not correlated. The reason for this is actually

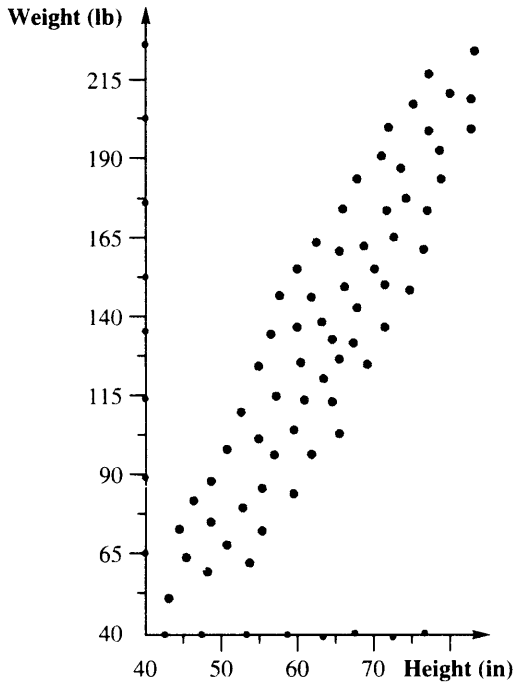

FIGURE 10.3 **The height-weight vector quantizer.**

quite simple. As we look at longer and longer sequences of source outputs, we are afforded more flexibility in terms of our design. This flexibility in turn allows us to match the design of the quantizer to the source characteristics. Consider the following example.

Example 10.3.2:

Suppose we have to design a uniform quantizer with eight output values for a Laplacian input. Using the information from Table 9.3 in Chapter 9, we would obtain the quantizer shown in Figure 10.4, where Δ is equal to 0.7309. As the input has a Laplacian distribution, the probability of the source output falling in the different quantization intervals is not the same. For example, the probability that the input will fall in the interval $[0, \Delta)$ is 0.3242, while the probability that a source output will fall in the interval $[3\Delta, \infty)$ is 0.0225. Let's look at how this quantizer will quantize two consecutive source outputs. As we did in the previous example, let's plot the first sample along the x-axis and the second sample along the y-axis. We can represent this two-dimensional view of the quantization process as shown in Figure 10.5. Note that, as in the previous example, we have not changed the quantization process; we are simply representing it differently. The first quantizer input, which we have represented in the figure as x_1, is quantized to the same eight possible output values as before. The same is true for the second quantizer input, which we have represented in the

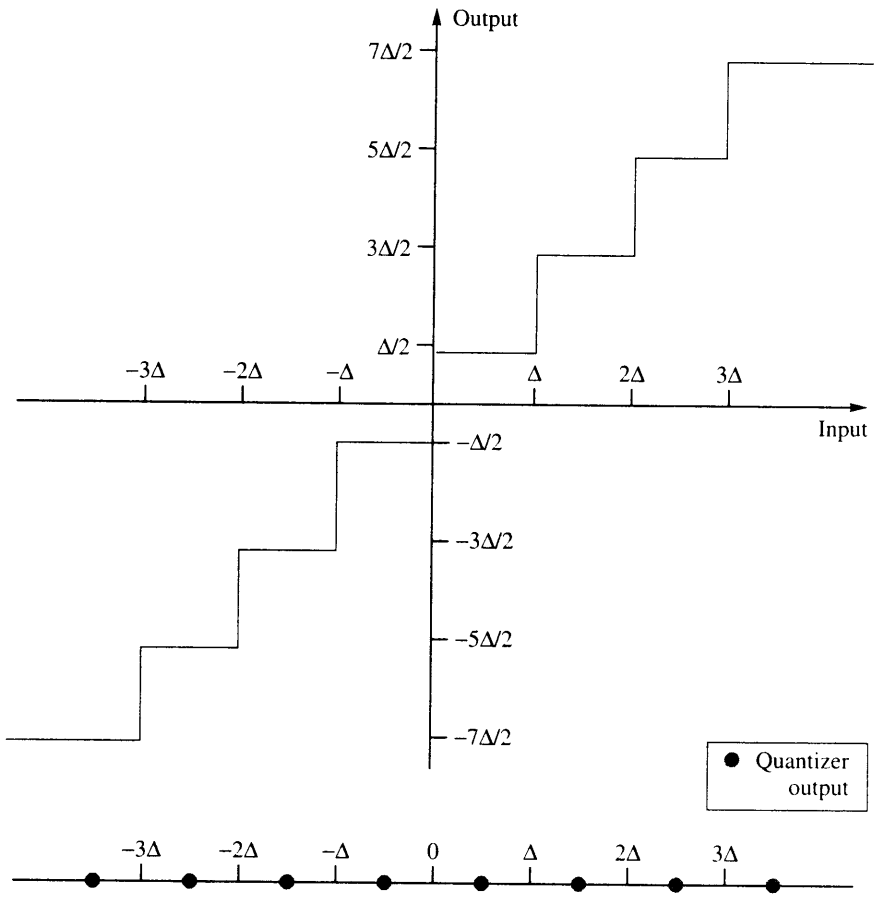

FIGURE 10.4 Two representations of an eight-level scalar quantizer.

figure as x_2. This two-dimensional representation allows us to examine the quantization process in a slightly different manner. Each filled-in circle in the figure represents a sequence of two quantizer outputs. For example, the top rightmost circle represents the two quantizer outputs that would be obtained if we had two consecutive source outputs with a value greater than 3Δ. We computed the probability of a single source output greater than 3Δ to be 0.0225. The probability of two consecutive source outputs greater than 2.193 is simply $0.0225 \times 0.0225 = 0.0005$, which is quite small. Given that we do not use this output point very often, we could simply place it somewhere else where it would be of more use. Let us move this output point to the origin, as shown in Figure 10.6. We have now modified the quantization process. Now if we get two consecutive source outputs with values greater than 3Δ, the quantizer output corresponding to the second source output may not be the same as the first source output.

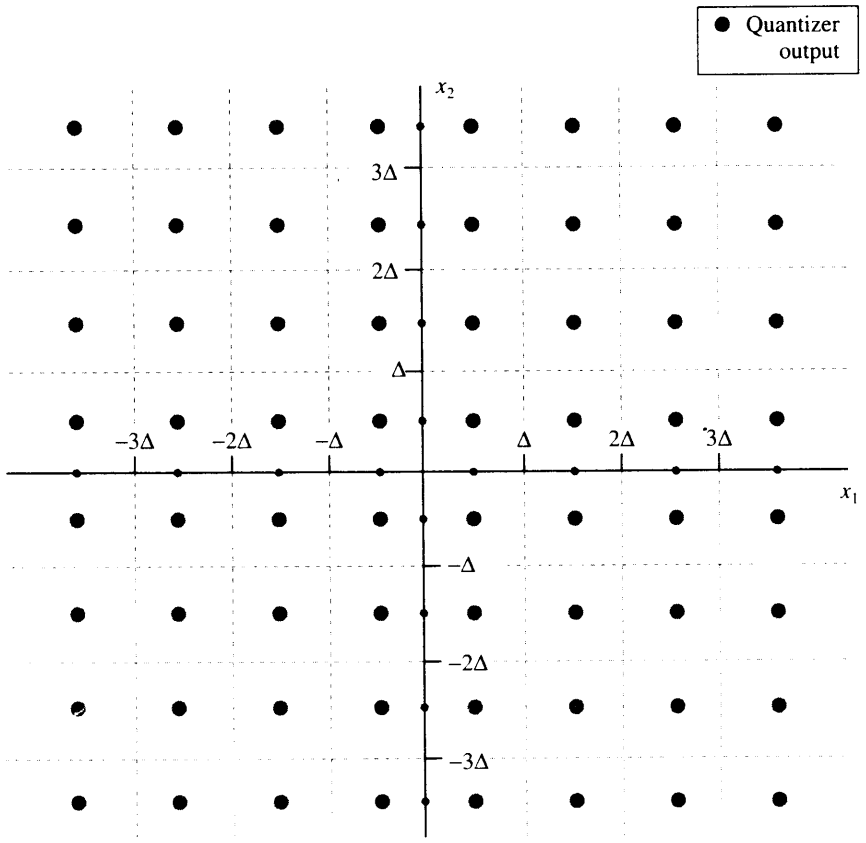

FIGURE 10.5 **Input-output map for consecutive quantization of two inputs using an eight-level scalar quantizer.**

If we compare the rate distortion performance of the two vector quantizers, the SNR for the first vector quantizer is 11.44 dB, which agrees with the result in Chapter 9 for the uniform quantizer with a Laplacian input. The SNR for the modified vector quantizer, however, is 11.73 dB, an increase of about 0.3 dB. Recall that the SNR is a measure of the average squared value of the source output samples and the mean squared error. As the average squared value of the source output is the same in both cases, an increase in SNR means a decrease in the mean squared error. Whether this increase in SNR is significant will depend on the particular application. What is important here is that by treating the source output in groups of two we could effect a positive change with only a minor modification. We could argue that this modification is really not that minor since the uniform characteristic of the original quantizer has been destroyed. However, if we begin with a nonuniform quantizer and modify it in a similar way, we get similar results.

Could we do something similar with the scalar quantizer? If we move the output point at $\frac{7\Delta}{2}$ to the origin, the SNR *drops* from 11.44 dB to 10.8 dB. What is it that permits us to make

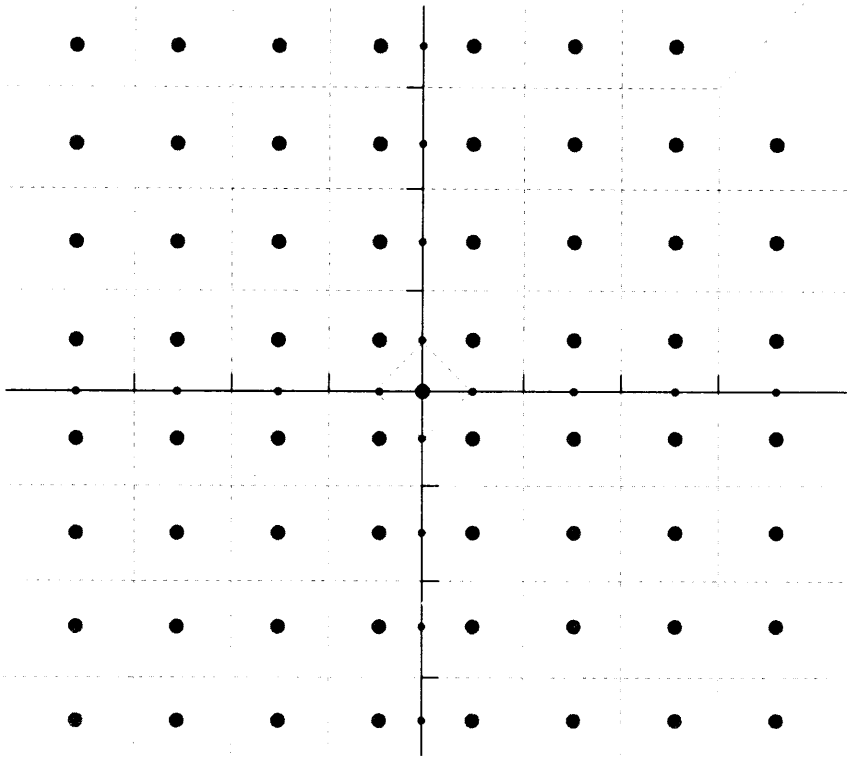

FIGURE 10.6 Modified two-dimensional vector quantizer.

modifications in the vector case, but not in the scalar case? This advantage is caused by the added flexibility we get by viewing the quantization process in higher dimensions. Consider the effect of moving the output point from $\frac{7\Delta}{2}$ to the origin in terms of two consecutive inputs. This one change in one dimension corresponds to moving 15 output points in two dimensions. Thus, modifications at the scalar quantizer level are gross modifications when viewed from the point of view of the vector quantizer. Remember that in this example we have only looked at two-dimensional vector quantizers. As we block the input into larger and larger blocks or vectors, these higher dimensions provide even greater flexibility and the promise of further gains to be made. ♦

In Figure 10.6, notice how the quantization regions have changed for the outputs around the origin, as well as for the two neighbors of the output point that were moved. The decision boundaries between the reconstruction levels can no longer be described as easily as in the case for the scalar quantizer. However, if we know the distortion measure, simply knowing the output points gives us sufficent information to implement the quantization

process. Instead of defining the quantization rule in terms of the decision boundary, we can define the quantization rule as follows:

$$Q(X) = Y_j \quad \text{iff} \quad d(X, Y_j) < d(X, Y_i) \ \forall i \neq j. \tag{10.3}$$

For the case where the input X is equidistant from two output points, we can use a simple tie-breaking rule such as "use the output point with the smaller index." The quantization regions V_j can then be defined as

$$V_j = \{X : d(X, Y_j) < d(X, Y_i) \ \forall i \neq j\}. \tag{10.4}$$

Thus, the quantizer is completely defined by the output points and a distortion measure.

From a multidimensional point of view, using a scalar quantizer for each input restricts the output points to a rectangular grid. Observing several source output values at once allows us to move the output points around. Another way of looking at this is that in one dimension the quantization intervals are restricted to be intervals, and the only parameter that we can manipulate is the size of these intervals. When we divide the input into vectors of some length n, the quantization regions are no longer restricted to be rectangles or squares. We have the freedom to divide the range of the inputs in an infinite number of ways.

These examples have shown two ways in which the vector quantizer can be used to improve performance. In the first case, we exploited the sample-to-sample dependence of the input. In the second case, there was no sample-to-sample dependence; the samples were independent. However, looking at two samples together still improved performance.

These two examples can be used to motivate two somewhat different approaches toward vector quantization. One approach is a pattern-matching approach, similar to the process used in Example 10.3.1, while the other approach deals with the quantization of random inputs. We will look at both of these approaches in this chapter.

10.4 The Linde-Buzo-Gray Algorithm

In Example 10.3.1 we saw that one way of exploiting the structure in the source output is to place the quantizer output points where the source output (blocked into vectors) are most likely to congregate. The set of quantizer output points is called the *codebook* of the quantizer, and the process of placing these output points is often referred to as *codebook design*. When we group the source output in two-dimensional vectors, as in the case of Example 10.3.1, we might be able to obtain a good codebook design by plotting a representative set of source output points and then visually locate where the quantizer output points should be. However, this approach to codebook design breaks down when we design higher-dimensional vector quantizers. Consider designing the codebook for a 16-dimensional quantizer. Obviously, a visual placement approach will not work in this case. We need an automatic procedure for locating where the source outputs are clustered.

This is a familiar problem in the field of pattern recognition. It is no surprise, therefore, that the most popular approach to designing vector quantizers is a clustering procedure known as the k-means algorithm, which was developed for pattern recognition applications.

The k-means algorithm functions as follows: Given a large set of output vectors from the source, known as the *training set*, and an initial set of k representative patterns, assign each element of the training set to the closest representative pattern. After an element is assigned, the representative pattern is updated by computing the centroid of the training set vectors assigned to it. When the assignment process is complete, we will have k groups of vectors clustered around each of the output points.

Stuart Lloyd [115] used this approach to generate the *pdf*-optimized scalar quantizer, except that instead of using a training set, he assumed that the distribution was known. The Lloyd algorithm functions as follows:

1. Start with an initial set of reconstruction values $\{y_i^{(0)}\}_{i=1}^{M}$. Set $k = 0$, $D^{(0)} = 0$. Select threshold ϵ.
2. Find decision boundaries
$$b_j^{(k)} = \frac{y_{j+1}^{(k)} + y_j^{(k)}}{2} \qquad j = 1, 2, \ldots, M-1.$$
3. Compute the distortion
$$D^{(k)} = \sum_{i=1}^{M} \int_{b_{i-1}^{(k)}}^{b_i^{(k)}} (x - y_i)^2 f_X(x)dx.$$
4. If $D^{(k)} - D^{(k-1)} < \epsilon$, stop; otherwise, continue.
5. $k = k + 1$. Compute new reconstruction values
$$y_j^{(k)} = \frac{\int_{b_{j-1}^{(k-1)}}^{b_j^{(k-1)}} x f_X(x)dx}{\int_{b_{j-1}^{(k-1)}}^{b_j^{(k-1)}} f_X(x)dx}.$$
Go to Step 2.

Linde, Buzo, and Gray generalized this algorithm to the case where the inputs are no longer scalars [125]. For the case where the distribution is known, the algorithm looks very much like the Lloyd algorithm described above.

1. Start with an initial set of reconstruction values $\{Y_i^{(0)}\}_{i=1}^{M}$. Set $k = 0$, $D^{(0)} = 0$. Select threshold ϵ.
2. Find quantization regions
$$V_i^{(k)} = \{X : d(X, Y_i) < d(X, Y_j)\ \forall j \neq i\} \qquad j = 1, 2, \ldots, M.$$
3. Compute the distortion
$$D^{(k)} = \sum_{i=1}^{M} \int_{V_i^{(k)}} \|X - Y_i^{(k)}\|^2 f_X(X)dX.$$

4. If $\frac{(D^{(k)}-D^{(k-1)})}{D^{(k)}} < \epsilon$, stop; otherwise, continue.

5. $k = k+1$. Find new reconstruction values $\left\{Y_i^{(k)}\right\}_{i=1}^{M}$ that are the centroids of $\left\{V_i^{(k-1)}\right\}$. Go to Step 2.

This algorithm is not very practical because the integrals required to compute the distortions and centroids are over odd-shaped regions in n dimensions, where n is the dimension of the input vectors. Generally, these integrals are extremely difficult to compute, making this particular algorithm more of an academic interest.

Of more practical interest is the algorithm for the case where we have a training set available. In this case, the algorithm looks very much like the k-means algorithm.

1. Start with an initial set of reconstruction values $\left\{Y_i^{(0)}\right\}_{i=1}^{M}$ and a set of training vectors $\{X_n\}_{n=1}^{N}$. Set $k = 0$, $D^{(0)} = 0$. Select threshold ϵ.

2. The quantization regions $\left\{V_i^{(k)}\right\}_{i=1}^{M}$ are given by

$$V_i^{(k)} = \{X_n : d(X_n, Y_i) < d(X_n, Y_j)\ \forall j \neq i\} \qquad i = 1, 2, \ldots, M.$$

   We assume that none of the quantization regions are empty. (Later we will deal with the case where $V_i^{(k)}$ is empty for some i and k.)

3. Compute the average distortion $D^{(k)}$ between the training vectors and the representative reconstruction value.

4. If $\frac{(D^{(k)}-D^{(k-1)})}{D^{(k)}} < \epsilon$, stop; otherwise, continue.

5. $k = k+1$. Find new reconstruction values $\left\{Y_i^{(k)}\right\}_{i=1}^{M}$ that are the average value of the elements of each of the quantization regions $V_i^{(k-1)}$. Go to Step 2.

This algorithm forms the basis of most vector quantizer designs. It is popularly known as the Linde-Buzo-Gray or LBG algorithm, or the generalized Lloyd algorithm (GLA) [125]. Although the paper of Linde, Buzo, and Gray [125] is a starting point for most of the work on vector quantization, the latter algorithm had been used several years prior by Edward E. Hilbert at the NASA Jet Propulsion Laboratories in Pasadena, California. Hilbert's starting point was the idea of clustering, and although he arrived at the same algorithm as described above, he called it the *cluster compression algorithm* [126].

In order to see how this algorithm functions, consider the following example of a two-dimensional vector quantizer codebook design.

Example 10.4.1:

Suppose our training set consists of the height and weight values shown in Table 10.1. The initial set of output points is shown in Table 10.2. (For ease of presentation, we will always round the coordinates of the output points to the nearest integer.) The inputs, outputs, and quantization regions are shown in Figure 10.7.

TABLE 10.1 Training set for designing vector quantizer codebook.

Height	Weight
72	180
65	120
59	119
64	150
65	162
57	88
72	175
44	41
62	114
60	110
56	91
70	172

TABLE 10.2 Initial set of output points for codebook design.

Height	Weight
45	50
75	117
45	117
80	180

The input (44, 41) has been assigned to the first output point; the inputs (56, 91), (57, 88), (59, 119), and (60, 110) have been assigned to the second output point; the inputs (62, 114), and (65, 120) have been assigned to the third output; and the five remaining vectors from the training set have been assigned to the fourth output. The distortion for this assignment is 387.25. We now find the new output points. There is only one vector in the first quantization region, so the first output point is (44, 41). The average of the four vectors in the second quantization region (rounded up) is the vector (58, 102), which is the new second output point. In a similar manner, we can compute the third and fourth output points as (64, 117) and (69, 168). The new output points and the corresponding quantization regions are shown in Figure 10.8. From Figure 10.8, we can see that, while the training vectors that were initially part of the first and fourth quantization regions are still in the same quantization regions, the training vectors (59,115) and (60,120), which were in quantization region 2, are now in quantization region 3. The distortion corresponding to this assignment of training vectors to quantization regions is 89, considerably less than the original 387.25. Given the new assignments, we can obtain a new set of output points. The first and fourth output points do not change because the training vectors in the corresponding regions have not changed. However, the training vectors in regions 2 and 3 have changed. Recomputing the output points for these regions, we get (57, 90) and (62, 116). The final form of the

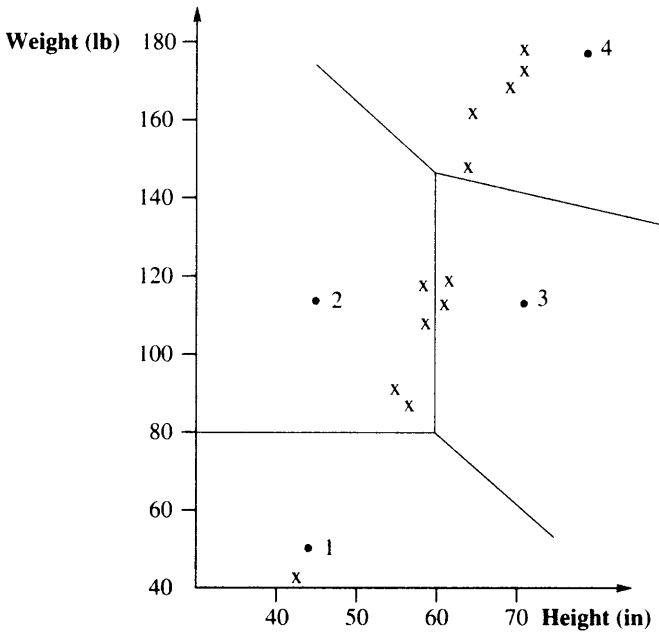

FIGURE 10.7 **Initial state of the vector quantizer.**

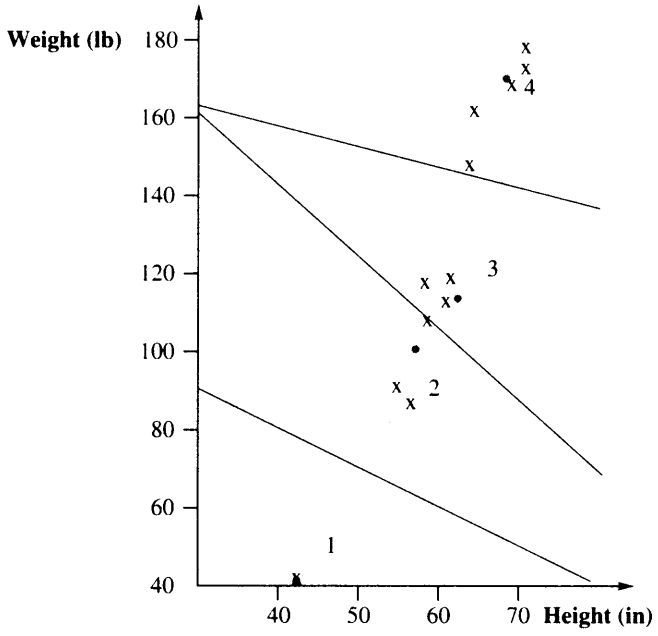

FIGURE 10.8 **The vector quantizer after one iteration.**