3

Sampling and Quantization

3.1 Introduction

In digital communication systems, signal processing tools require the input source to be digitized before being processed through various stages of the network. The digitization process consists of two main stages: sampling the signal and converting the sampled amplitudes into binary (digital) codewords. The difference between the original analogue amplitudes and the digitized ones depend on the number of bits used in the conversion. A 16 bit analogue to digital converter is usually used to sample and digitize the input analogue speech signal. Having digitized the input speech, the speech coding algorithms are used to compress the resultant bit rate where various quantizers are used. In this chapter, after a brief review of the sampling process, quantizers which are used in speech coders are discussed.

3.2 Sampling

As stated above, the digital conversion process can be split into sampling, which discretizes the continuous time, and quantization, which reduces the infinite range of the sampled amplitudes to a finite set of possibilities. The sampled waveform can be represented by,

$$s(n) = s_a(nT) \qquad -\infty < n < \infty \tag{3.1}$$

where s_a is the analogue waveform, n is the integer sample number and T is the sampling time (the time difference between any two adjacent samples, which is determined by the bandwidth or the highest frequency in the input signal).

The sampling theorem states that if a signal $s_a(t)$ has a band-limited Fourier transform $S_a(j\omega)$ given by,

$$S_a(j\omega) = \int_{-\infty}^{\infty} s_a(t)e^{-j\omega t}dt \tag{3.2}$$

such that $S_a(j\omega) = 0$ for $|\omega| \geq 2\pi W$ then the analogue signal can be reconstructed from its sampled version if $T \leq 1/2W$. W is called the *Nyquist frequency*.

The effect of sampling is shown in Figure 3.1. As can be seen from Figures 3.1b and 3.1c, the band-limited Fourier transform of the analogue signal which is shown in Figure 3.1a is duplicated at every multiple of the sampling frequency.

This is because the Fourier transform of the sampled signal is evaluated at multiples of the sampling frequency which forms the relationship,

$$S(e^{j\omega T}) = \frac{1}{T}\sum_{n=-\infty}^{\infty} S_a(j\omega + j2\pi n/T) \tag{3.3}$$

This can also be interpreted by looking into the time domain sampling process where the input signal is regularly (at every sampling interval) multiplied

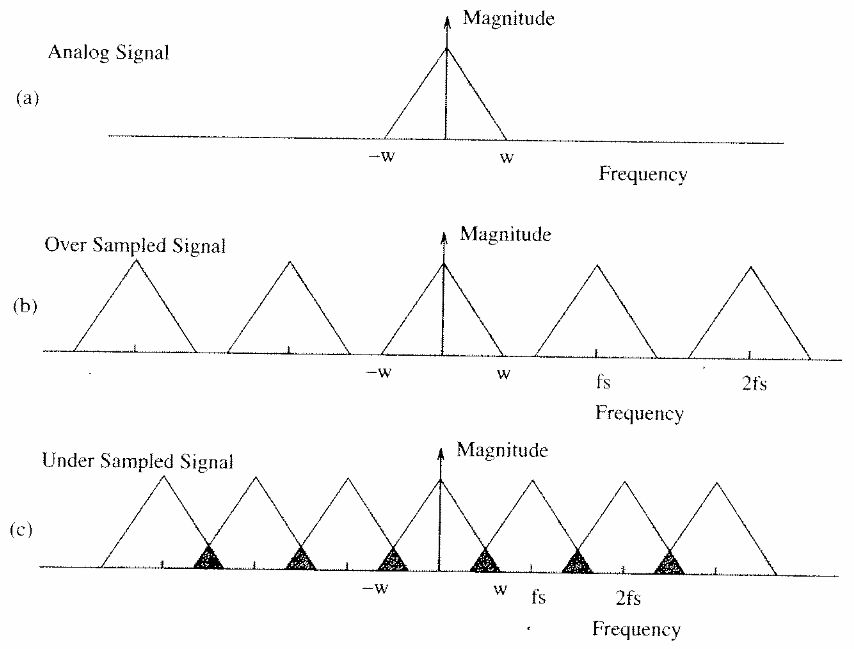

Figure 3.1 Effects of sampling: (a) original signal spectrum, (b) over sampled signal spectrum and (c) under sampled signal spectrum

with a delta function. When converted to the frequency domain, the multiplication becomes convolution and the message spectrum is reproduced at multiples of the sampling frequency.

We can clearly see that if the sampling frequency is less than twice the Nyquist frequency, the spectra of two adjacent multiples of the sampling frequencies will overlap. For example, if $\frac{1}{T} = f_s < 2W$ the analogue signal image centred at $2\pi/T$ overlaps into the base band image. The distortion caused by high frequencies overlapping low frequencies is called *aliasing*. In order to avoid aliasing distortion, either the input analogue signal has to be band-limited to a maximum of half the sampling frequency or the sampling frequency has to be increased to at least twice the highest frequency in the analogue signal.

Given the condition $1/T > 2W$, the Fourier transform of the sampled sequence is proportional to the Fourier transform of the analogue signal in the base band as follows:

$$S(e^{j\omega T}) = \frac{1}{T}S_a(j\omega) \quad |\omega| < \frac{\pi}{T} \tag{3.4}$$

Using the above relationship, the original analogue signal can be obtained from the sampled sequence using interpolation given by [1],

$$s_a(t) = \sum_{n=-\infty}^{\infty} s_a(nT)\frac{sin[\pi(t-nT)/T]}{\pi(t-nT)/T} \tag{3.5}$$

which can be written as,

$$s_a(t) = \sum_{n=-\infty}^{\infty} s_a(nT)sinc(\phi) \tag{3.6}$$

where $\phi = \pi(t-nT)/T$.

Therefore, if the sampling frequency is at least twice the Nyquist frequency, the analogue signal can be recovered completely from its sampled version by adding together *sinc* functions centred on each sampling point and scaled by the sampled value of the analogue signal. The $sinc(\phi)$ function in the above equation represents an ideal low pass filter. In practice, the front end band limitation before sampling is usually achieved by a low pass filter which is less than ideal and may cause aliasing distortion due to its roll-off characteristics. In order to avoid aliasing distortion, the sampling frequency is usually chosen to be higher than twice the Nyquist frequency. In telecommunication networks the analogue speech signal is band-limited to 300 to 3400 Hz and sampled at 8000 Hz. This same band limitation and sampling is used throughout this book unless otherwise specified.

3.3 Scalar Quantization

Quantization converts a continuous-amplitude signal (usually 16 bit, represented by the digitization process) to a discrete-amplitude signal that is different from the continuous-amplitude signal by the quantization error or noise. When each of a set of discrete values is quantized separately the process is known as scalar quantization. The input–output characteristics of a uniform scalar quantizer are shown in Figure 3.2.

Each sampled value of the input analogue signal, which has an infinite range (16 bit digitized), is compared against a finite set of amplitude values and the closest value from the finite set is chosen to represent the amplitude. The distance between the finite set of amplitude levels is called the quantizer step size and is usually represented by Δ. Each discrete amplitude level x_i is represented by a codeword $c(n)$ for transmission purposes. The codeword $c(n)$ indicates to the de-quantizer, which is usually at the receiver, which discrete amplitude is to be used.

Assuming all of the discrete amplitude values in the quantizer are represented by the same number of bits B and the sampling frequency is f_s, the

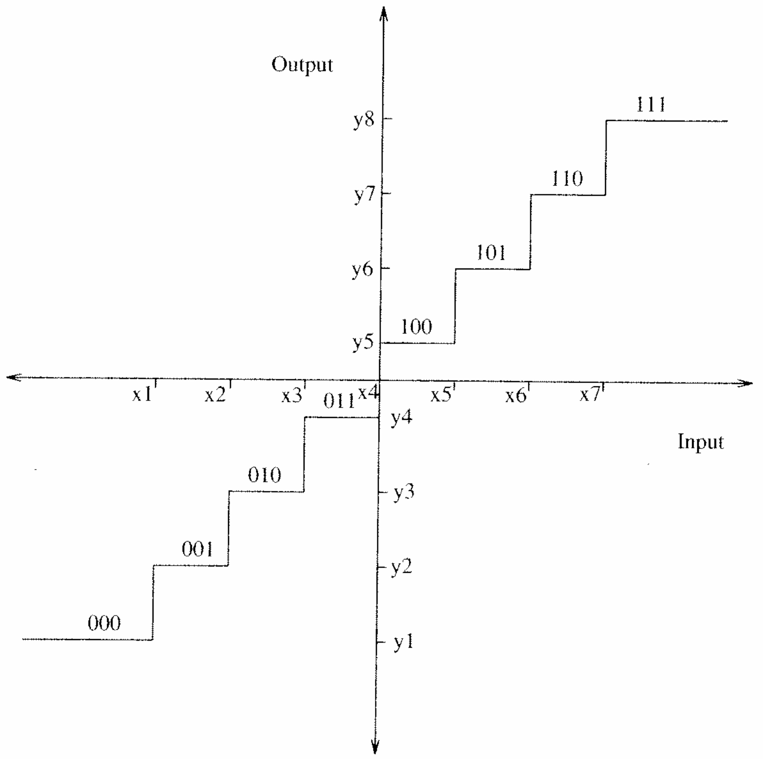

Figure 3.2 The input-output characteristics of a uniform quantizer

channel transmission bit rate is given by,

$$T_c = Bf_s \quad bits/second \tag{3.7}$$

Given a fixed sampling frequency, the only way to reduce the channel bit rate T_c is by reducing the length of the codeword $c(n)$. However, a reduced length $c(n)$ means a smaller set of discrete amplitudes separated by larger Δ and, hence, larger differences between the analogue and discrete amplitudes after quantization, which reduces the quality of reconstructed signal. In order to reduce the bit rate while maintaining good speech quality, various types of scalar quantizer have been designed and used in practice. The main aim of a specific quantizer is to match the input signal characteristics both in terms of its dynamic range and probability density function.

3.3.1 Quantization Error

When estimating the quantization error, we cannot assume that $\Delta_i = \Delta_{i+n}$ if the quantizer is not uniform [2]. Therefore, the signal lying in the i^{th} interval,

$$x_i - \frac{\Delta_i}{2} \le s(n) < x_i + \frac{\Delta_i}{2} \tag{3.8}$$

is represented by the quantized amplitude x_i and the difference between the input and quantized values is a function of Δ_i. The instantaneous squared error, for the signal lying in the i^{th} interval is $(s(n) - x_i)^2$. The mean squared error of the signal can then be written by including the likelihood of the signal being in the i^{th} interval as,

$$E_i^2 = \int_{x_i-\frac{\Delta_i}{2}}^{x_i+\frac{\Delta_i}{2}} (x - x_i)^2 p(x)dx \tag{3.9}$$

where $s(n)$ has been replaced by x for ease of notation and $p(x)$ represents the probability density function of x. Assuming the step size Δ_i is small, enabling very fine quantization, we can assume that $p(x)$ is flat within the interval $x_i - \frac{\Delta}{2}$ to $x_i + \frac{\Delta}{2}$. Representing the flat region of $p(x)$ by its value at the centre, $p(x_i)$, the above equation can be written as,

$$E_i^2 = p(x_i) \int_{-\frac{\Delta_i}{2}}^{\frac{\Delta_i}{2}} y^2 dy = \frac{\Delta_i^3}{12} p(x_i) \tag{3.10}$$

The probability of the signal falling in the i^{th} interval is,

$$\Gamma_i = \int_{x_i-\frac{\Delta_i}{2}}^{x_i+\frac{\Delta_i}{2}} p(x)dx = p(x_i)\Delta_i \tag{3.11}$$

The above is true only if the quantization levels are very small and, hence, $p(x)$ in each interval can be assumed to be uniform. Substituting (3.11) into (3.10) for $p(x_i)$ we get,

$$E_i^2 = \frac{\Delta_i^2}{12}\Gamma_i \tag{3.12}$$

The total mean squared error is therefore given by,

$$E^2 = \frac{1}{12}\sum_{i=1}^{N}\Gamma_i\Delta_i^2 \tag{3.13}$$

where N is the total number of levels in the quantizer. In the case of a uniform quantizer where each step size is the same, Δ, the total mean squared error becomes,

$$E^2 = \frac{\Delta^2}{12}\sum_{i=1}^{N}\Gamma_i = \frac{\Delta^2}{12} \tag{3.14}$$

where we assume that the signal amplitude is always in the quantizer range and, hence, $\sum_{i=1}^{N}\Gamma_i = 1$.

3.3.2 Uniform Quantizer

The input–output characteristics of a uniform quantizer are shown in Figure 3.2. As can be seen from its input–output characteristics, all of the quantizer intervals (steps) are the same width. A uniform quantizer can be defined by two parameters: the number of quantizer levels and the quantizer step size Δ. The number of levels is generally chosen to be of the form 2^B, to make the most efficient use of B bit binary codewords. Δ and B must be chosen together to cover the range of input samples. Assuming $|x| \leq X_{max}$ and that the probability density function of x is symmetrical, then,

$$2X_{max} = \Delta 2^B \tag{3.15}$$

From the above equation it is easily seen that once the number of bits to be used, B, is known, then the step size, Δ, can be calculated by,

$$\Delta = \frac{2X_{max}}{2^B} \tag{3.16}$$

The quantization error $e_q(n)$ is bounded by,

$$-\frac{\Delta}{2} \leq e_q(n) \leq \frac{\Delta}{2} \tag{3.17}$$

In a uniform quantizer, the only way to reduce the quantization error is by increasing the number of bits. When a uniform quantizer is used, it is assumed that the input signal has a uniform probability density function varying between $\pm X_{max}$ with a constant height of $\frac{1}{2X_{max}}$. From this, the power of the input signal can be written as,

$$P_x = \int_{-X_{max}}^{X_{max}} x^2 p(x)dx = \frac{X_{max}^2}{3} \tag{3.18}$$

Using the result of (3.14), the signal to noise ratio can be written as,

$$SNR = \frac{P_x}{P_n} = \frac{X_{max}^2/3}{\Delta^2/12} \tag{3.19}$$

Substituting (3.16) for Δ we get,

$$SNR = \frac{P_x}{P_n} = 2^{2B} \tag{3.20}$$

Taking the log,

$$SNR(dB) = 10\log_{10}(2^{2B}) = 20B\log_{10}(2) = 6.02B\ dB \tag{3.21}$$

The above result is useful both in determining the number of bits needed in the quantizer for certain signal to quantization noise ratio and in estimating the performance of a uniform quantizer for a given bit rate.

3.3.3 Optimum Quantizer

When choosing the levels of a quantizer, positioning of these levels has to be selected so that the quantization error is minimized. In order to maximize the ratio of signal to quantization noise for a given number of bits per sample, levels of the quantizer must be selected to match the probability density function of the signal to be quantized. This is because speech-like signals do not have a uniform probability density function, and the probability of smaller amplitudes occurring is much higher than that of large amplitudes. Consequently, to cover the signal dynamic range as accurately as possible, the optimum quantizer should have quantization levels with nonuniform spacing. The input–output characteristics of a typical nonuniform quantizer where the step size of the quantizer intervals is increasing for higher input signal values is shown in Figure 3.3. The noise contribution of each interval depends on the probability of the signal falling into a certain quantization interval. The nonuniform spacing of the quantization levels is equivalent to a nonlinear compressor $C(x)$ followed by a uniform quantizer. The nonlinear

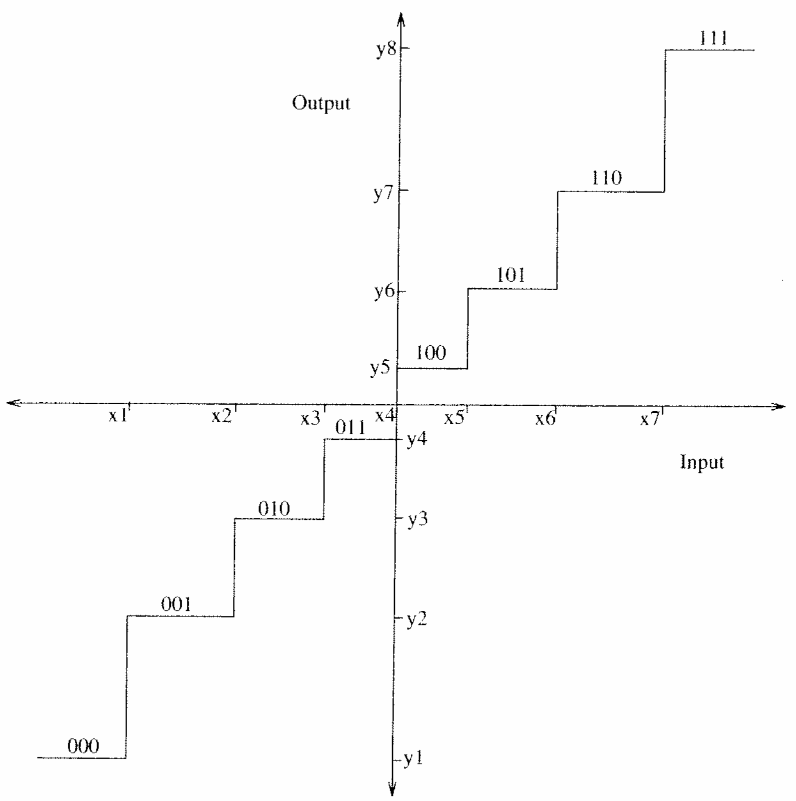

Figure 3.3 The input-output characteristics of a nonuniform quantizer

compressor, $C(x)$, compresses the input samples depending on their statistical properties. In other words, the less likely higher sample values are compressed more than the more likely low amplitude samples. The compressed samples are then quantized using a uniform quantizer. The effect of compression is reversed at the receiver by applying the inverse $C^{-1}(x)$ expansion to the de-quantized samples. The compression and expansion processes do not introduce any signal distortions.

It is quite important to select the best compression–expansion combination for a given input signal probability density function. Panter and Dite [3] used analysis based on the assumption that the quantization is sufficiently fine and that the amplitude probability density function of the input samples is constant within the quantization intervals. Their results show significant improvement in the signal to noise ratio over uniform quantization if the input samples have a *peak* to root mean squared (*rms*) ratio greater than 4.

In designing an optimum quantizer, Max [4] discovered how to optimally choose the output levels for nonuniform input quantizer levels. His analysis required prior knowledge of the probability density function together with the

Table 3.1 Max quantizer input and output levels for 1, 2, 3, 4, and 5 bit quantizers

Max quantizer thresholds									
1 bit		2 bit		3 bit		4 bit		5 bit	
i/p	*o/p*	*i/p*	*o/p*	*i/p*	*o/p*	*i/p*	*o/p*	*i/p*	*o/p*
0.0000	0.7980	0.0000	0.4528	0.0000	0.2451	0.0000	0.1284	0.0000	0.0659
		0.9816	1.5100	0.5006	0.7560	0.2582	0.3881	0.1320	0.1981
				1.0500	1.3440	0.5224	0.6568	0.2648	0.3314
				1.7480	2.1520	0.7996	0.9424	0.3991	0.4668
						1.0990	1.2560	0.5359	0.6050
						1.4370	1.6180	0.6761	0.7473
						1.8440	2.0690	0.8210	0.8947
						2.4010	2.7330	0.9718	1.0490
								1.1300	1.2120
								1.2990	1.3870
								1.4820	1.5770
								1.6820	1.7880
								1.9080	2.0290
								2.1740	2.3190
								2.5050	2.6920
								2.9770	3.2630

variance, σ_x^2, of the input signal but made no assumption of fine quantization. The quantizer input–output threshold values for 1–5 bit Max quantizers are tabulated in Table 3.1 [4]. The quantizers in Table 3.1 are for a unit variance signal with a normal probability density function. Each quantizer has the same threshold values in the corresponding negative side of the quantizer.

Nonuniform quantization is advantageous in speech coding, both in coarse and fine quantization cases, for two reasons. Firstly, a nonuniform quantizer matches the speech probability density function better and hence produces higher signal to noise ratio than a uniform quantizer. Secondly, lower amplitudes, which contribute more to the intelligibility of speech, are quantized more accurately in a nonuniform quantizer.

In speech coding, Max's quantizer [4] is widely used to normalize the input samples to unit variance, which guarantees the input dynamic range. In many other cases, specific nonuniform quantizers are designed by optimizing the quantizer intervals using a large number of samples of the signal to be quantized. Although, these specific quantizers are not generally applicable, they give the best performance for a given signal with a given probability density function and variance. In cases where the variance of the signal has a large dynamic range, the variance of the signal is transmitted separately at known time intervals enabling a unit variance nonuniform quantizer to be used. These quantizers are called forward adaptive nonuniform quantizers.

3.3.4 Logarithmic Quantizer

As was discussed above, an optimum quantizer is advantageous if the dynamic range (or variance) of the input signal is fixed to a small known range. However, the performance of such a quantizer deteriorates rapidly as the power of the signal moves away from the value that the quantizer is designed for. Although, this can be controlled by normalizing the input signal to unit variance, this process requires the transmission of the signal variance at known time intervals for correct scaling of the de-quantized signal amplitudes.

In order to cater for the wide dynamic range of the input speech signal, Cattermole [2] suggested two companding laws called A-Law and μ-Law Pulse Code Modulation (PCM). In both schemes, the signal to quantization noise performance can be very close to that of a uniform quantizer, but their performances do not change significantly with changing signal variance and remain relatively constant over a wide range of input speech levels. When compared with uniform quantizers, companded quantizers require fewer bits per input sample for a specified signal dynamic range and signal to quantization noise ratio. In a companding quantizer, quantizer levels are closely spaced for small amplitudes which progressively increase as the input signal range increases. This ensures that, when quantizing speech signals where the probability density function is zero mean and maximum at the origin, the frequently occurring small amplitudes are more accurately quantized than the less frequent large amplitudes, achieving a significantly better performance than a uniform quantizer.

The A-Law compression is defined by:

$$A_{Law}(x) = \frac{Ax}{1+\log_{10}(A)} \quad for\ 0 \le x \le \frac{1}{A} \tag{3.22}$$

$$A_{Law}(x) = \frac{1+\log_{10}(Ax)}{1+\log_{10}(A)} \quad for\ \frac{1}{A} \le x \le 1 \tag{3.23}$$

where A is the compression parameter with typical values of 86 for 7 bit (North American) PCM and 87.56 for 8 bit (European) PCM speech quantizers.

The μ-Law compression on the other hand is defined by:

$$\mu_{Law}(x) = sign(x)\frac{V_o \log_{10}\left[1+\frac{\mu|x|}{V_o}\right]}{\log_{10}[1+\mu]} \tag{3.24}$$

where V_o is given by $V_o = L\sigma_x$ in which L is the loading factor and σ_x is the *rms* value of the input speech signal.

A typical value of the compression factor μ is 255. The above expressions show that the A-Law is a combination of a logarithmic curve for large amplitudes and a linear curve for small amplitudes. The μ-Law on the other hand is not exactly linear or logarithmic in any range but it is approximately linear for small amplitudes and logarithmic for large amplitudes. A comparison made in [5] between a μ-Law quantizer and an optimum quantizer showed that the optimum quantizer can be as much as 4 dB better. However, an optimum quantizer may have more background noise when the channel is idle and its dynamic range is limited to a smaller input signal range. For these two reasons, logarithmic quantizers are usually preferred.

3.3.5 Adaptive Quantizer

As we have seen from the already discussed quantization schemes, the dynamic range of the input signal plays a crucial role in determining the performance of a quantizer. Although, the probability density function of speech can easily be estimated and used in a quantizer design process, the variations in its dynamic range, which can be as much as 30 dB, reduces the performance of any quantizer. This can be overcome by controlling the dynamic range of the input signal. As was briefly mentioned earlier, one way of achieving this is by estimating the variance of the speech segment prior to quantization and hence, adjusting the quantizer levels accordingly. The adjustment of the quantizer levels is equivalent to designing the quantizer for unit variance and normalizing the input signal before quantization. This is called forward adaptation. A forward adaptive quantizer block diagram is shown in Figure 3.4. Assuming the speech is stationary during K samples, the *rms* is given by:

$$\sigma_x = \sqrt{\frac{1}{K}\sum_{n=1}^{K} x(n)^2} \tag{3.25}$$

where the speech samples in the block are represented by $x(n)$ and mean is assumed to be zero. However, the choice of block length K is very important because the probability density function of the normalized input signal can be affected by K. As K increases the probability density of the normalized speech signal changes from Gaussian ($K \le 128$) to Laplacian ($K > 512$) [6]. This method requires the transmission of the speech block variances to the de-quantizer for correct signal amplitude adjustment. In order to make the normalization and de-normalization compatible, a quantized version of the speech *rms*, σ_x, is used at both the quantizer and the de-quantizer.

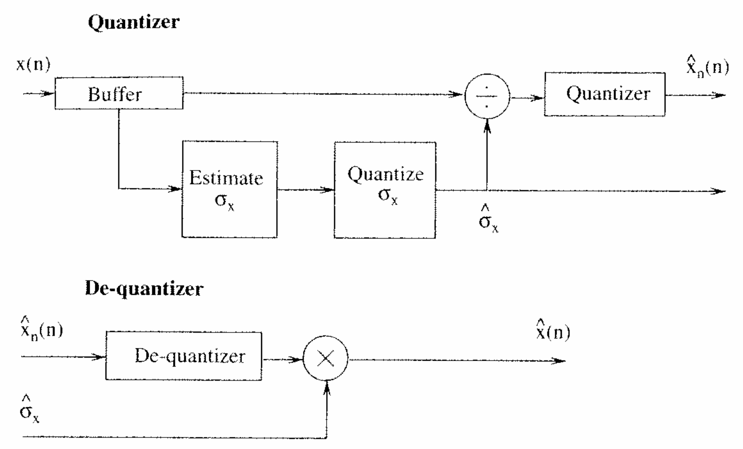

Figure 3.4 Block diagram of a forward adaptive quantizer

Another adaptation scheme which does not require transmission of the speech variance to the de-quantizer is called backward adaptation. Here, before quantizing each sample, the *rms* of the input signal is estimated from N previously quantized samples. Thus, the normalizing factor for the n^{th} sample is:

$$\sigma_x(n) = \sqrt{\frac{a_1}{N}\sum_{i=1}^{N}\hat{x}^2(n-i)} \tag{3.26}$$

where $\hat{x}$ represents the quantized values of the past samples and a_1 is a tuning factor [6].

It has been shown [7] that for a band-limited stationary zero-mean Gaussian input, as the period N increases, the obtained signal to noise ratio tends to an asymptotic maximum. However, N must be such that the power of the signal is fairly constant during the samples of estimation. On average, the backward adaptive quantizer has 3–5 dB more signal to noise ratio compared with a logarithmic quantizer. A block diagram of a backward adaptive quantizer is shown in Figure 3.5.

An adaptation scheme called *one word memory* [8] has also been suggested. It looks at only one previously quantized sample and either expands or compresses the quantizer intervals as shown in Figure 3.6. Thus at the $(n+1)^{th}$ sample the value of the quantizer step size Δ is:

$$\Delta_{n+1} = \Delta_n M_i(|\hat{x}(n)|) \tag{3.27}$$

where, M_i is one of i fixed coefficients corresponding to quantizer levels which control the expansion–compression processes.

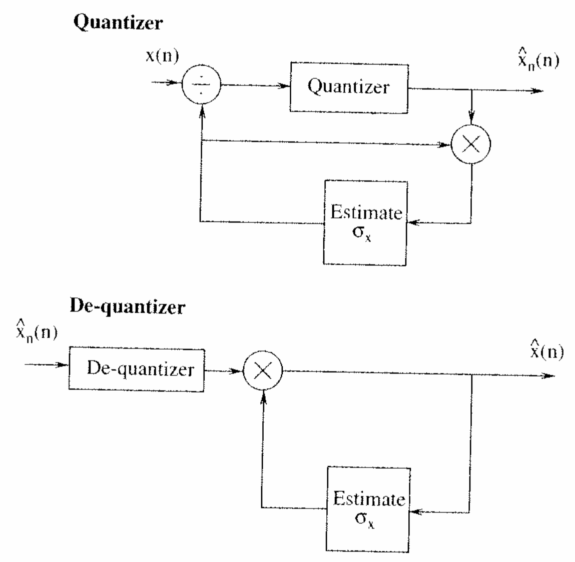

Figure 3.5 Block diagram of a backward adaptive quantizer

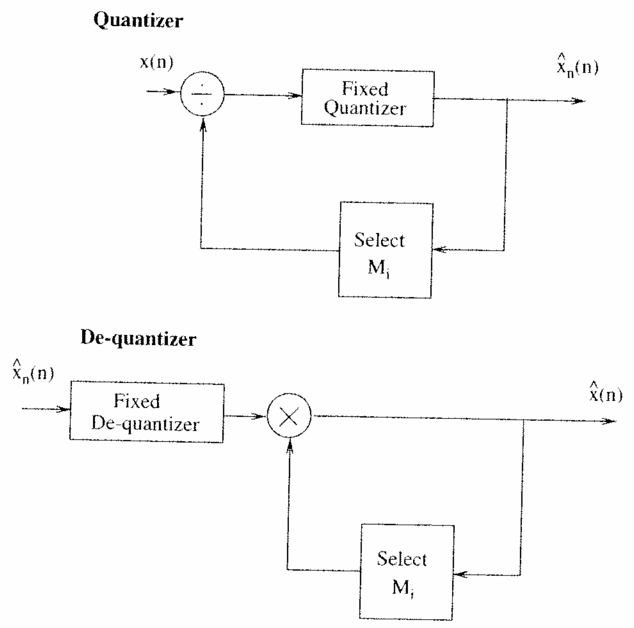

Figure 3.6 Block diagram of one word memory (Jayant) quantizer

For large quantized previous samples, multiplier values are greater than one and for small previously quantized samples multiplier values are less than one. A typical set of step size multiplier values for 2, 3 and 4 bit quantizers are shown in Table 3.2.

Table 3.2 Step size multiplier values for 2, 3, and 4 bit quantizers [9]

Adaptation multiplier values			
Previous o/p levels	2 bit	3 bit	4 bit
L1	0.60	0.85	0.80
L2	2.20	1.00	0.80
L3		1.00	0.80
L4		1.50	0.80
L5			1.20
L6			1.60
L7			2.00
L8			2.40

The recommended step size multiplier values [9] do not, in general, constitute critical target values. As can be seen from Table 3.2 [9], the middle values are fairly constant. What is critical, however, is that the step size increase should be more rapid than its decrease. This is very important for preventing quantizer overload.

3.3.6 Differential Quantizer

In a differential quantizer, the final quantized signal, $r(n)$ is the difference between the input samples $x(n)$ and their estimates $x_p(n)$.

$$r(n) = x(n) - x_p(n) \tag{3.28}$$

and

$$x_p(n) = \sum_{k=1}^{p} \hat{x}(n-k)a_k \tag{3.29}$$

where a_k is the weighting used for the previously quantized $(n-k)^{th}$ sample and p is the number of previously quantized samples considered in the estimation process.

The reason for this preprocessing stage to form the prediction residual (prediction error signal) before quantization is that, in speech signals, there is a strong correlation between adjacent samples and, hence, by removing some of the redundancies that speech signals possess, the signal variance is reduced before quantization. This reduces the quantization noise by employing a smaller quantizer step size Δ. Block diagrams of typical adaptive differential quantizers are shown in Figures 3.7 and 3.8.

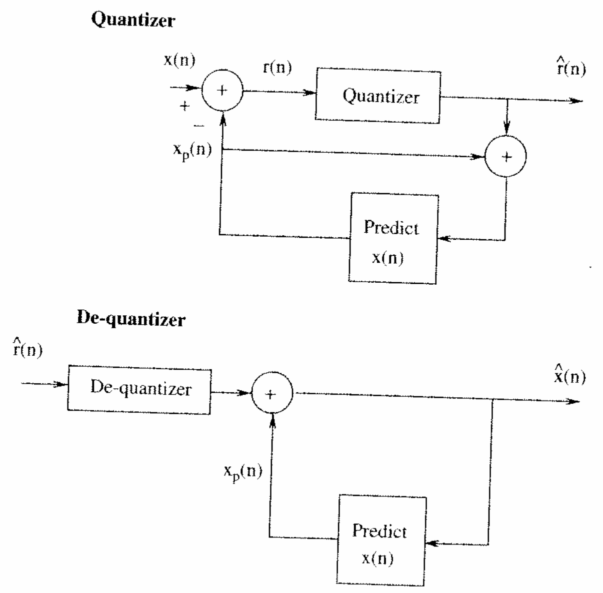

Figure 3.7 Block diagram of a backward adaptive differential quantizer

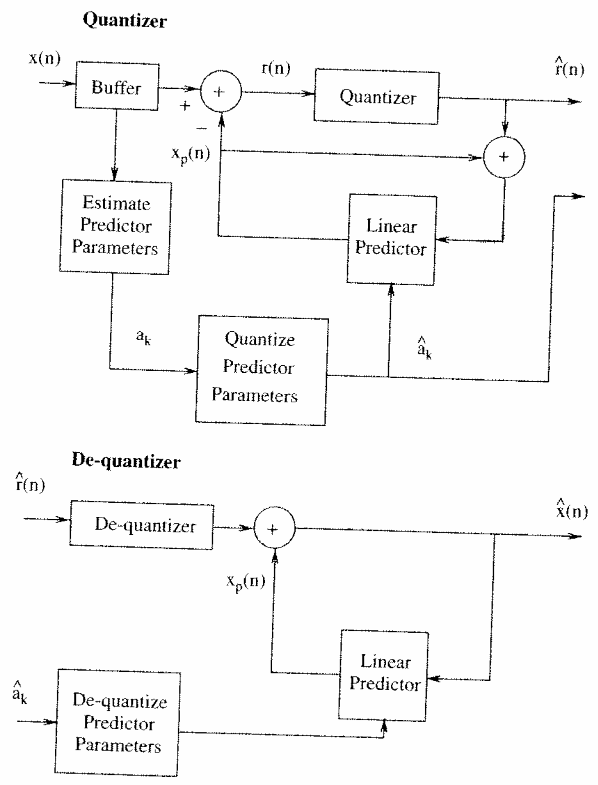

Figure 3.8 Block diagram of a forward adaptive differential quantizer

In order to show the advantage of a differential quantizer over a nondifferential quantizer, consider the following example: Assume that K input samples are to be quantized with a nondifferential quantizer with a total of $K.B_1$ bits. Consider also the same K samples are to be differentially quantized, in which case K error samples e_i are quantized to B_2 bits/sample accuracy. In a differential quantizer, the weighting coefficients a_k can be calculated using backward or forward techniques as shown in Figures 3.7 and 3.8. When backward estimation of the a_k parameters is used, the quantizer does not need to send extra information to the de-quantizer. However, in the case of forward estimation of the a_k parameters, the differential quantizer would also require $K.B_3$ bits to transmit the a_k parameters to the de-quantizer for correct recovery of the quantized signal. As the correlation between the input speech samples is usually high, the variance of the error signal to be quantized by the differential quantizer is much smaller than that of the original speech samples. Therefore, for the same accuracy of quantization, $B_2 < B_1$ and in general $B_3 \ll B_2$ which means $K.B_1 > K(B_2 + B_3)$. This shows that the main advantage of a differential over a nondifferential quantizer is due to the reduction in the speech dynamic range to be quantized.

The performance of a differential quantizer can be approximately defined by its prediction gain (the amount of signal reduction before quantization) and the performance of the residual error quantizer. Assuming that the same type of quantizer is used for both the differential and nondifferential quantization schemes, the difference in performance will depend on the accuracy of the predictor. For simplicity, if we assume a predictor depth of 1, and $\hat{x}(n-1) \simeq x(n-1)$ the residual error signal is obtained as,

$$r(n) = x(n) - ax(n-1) \tag{3.30}$$

where a is the weighting coefficient used on the previous sample to predict the current sample. The squared error is then given by,

$$r^2(n) = [x(n) - ax(n-1)]^2 \tag{3.31}$$

or,

$$r^2(n) = x^2(n) + a^2x^2(n-1) - 2ax(n)x(n-1) \tag{3.32}$$

Assuming, a is updated every N samples,

$$\sum_{n=1}^{N} r^2(n) = \sum_{n=1}^{N} x^2(n) + \sum_{n=1}^{N} a^2x^2(n-1) - 2a\sum_{n=1}^{N} x(n)x(n-1) \tag{3.33}$$

which can simply be written as,

$$\sigma_r^2 = \sigma_x^2 + a^2\sigma_x^2 - 2a\sum_{n=1}^{N} x(n)x(n-1) \tag{3.34}$$

Substituting $\rho = \frac{\sum_{n=1}^{N} x(n)x(n-1)}{\sum_{n=1}^{N} x^2(n)}$ (first order normalized autocorrelation coefficient) in (3.34) gives,

$$\sigma_r^2 = \sigma_x^2 + a^2\sigma_x^2 - 2a\sigma_x^2\rho \tag{3.35}$$

The prediction gain G_p is then found as,

$$G_p = \frac{\sigma_x^2}{\sigma_r^2} = \frac{1}{1 + a^2 - 2a\rho} \tag{3.36}$$

To maximize the prediction gain, the denominator of equation (3.36) should be minimized with respect to a, hence,

$$\frac{\partial(1 + a^2 - 2a\rho)}{\partial a} = 0 = (0 + 2a - 2\rho) \tag{3.37}$$

which gives,

$$a = \rho \tag{3.38}$$

Substituting $a = \rho$ in (3.36)

$$G_p = \frac{1}{1 + \rho^2 - 2\rho\rho} = \frac{1}{1 - \rho^2} \tag{3.39}$$

The above result shows that if the correlation between the adjacent samples is high, then a differential quantizer will perform significantly better than a nondifferential quantizer. In fact, if the signal to be quantized is a nonvarying DC signal, where $\rho = 1$, the gain of the prediction process will be infinite, i.e. no residual error will be left and, hence, no residual information will need to be transmitted. A typical ρ for speech is between 0.8 and 0.9 which may result in 4–7 dB signal reduction before quantization, hence achieving significant increase in quantization performance.

3.4 Vector Quantization

When a set of discrete-time amplitude values is quantized jointly as a single vector, the process is known as vector quantization (VQ), also known as block quantization or pattern-matching quantization. A block diagram of a simple vector quantizer is shown in Figure 3.9.

If we assume $\mathbf{x} = [x_1, x_2, \ldots ., x_N]^T$ is an N dimensional vector with real-valued, continuous-amplitude (short or float representation is assumed to be continuous amplitude) randomly varying components $x_k, 1 \leq k \leq N$ (the

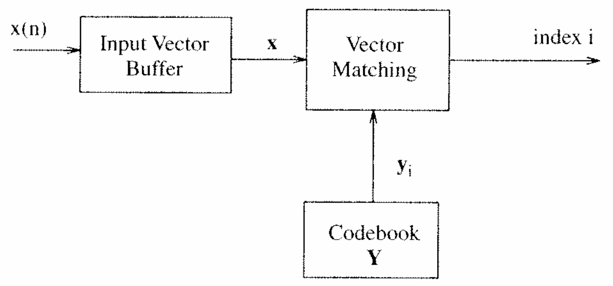

Figure 3.9 Block diagram of a simple vector quantizer

superscript T denotes transpose in vector quantization), this vector is matched with another real-valued, discrete-amplitude, N dimensional vector $\mathbf{y}$. Hence, $\mathbf{x}$ is quantized as $\mathbf{y}$, and $\mathbf{y}$ is used to represent $\mathbf{x}$. Usually, $\mathbf{y}$ is chosen from a finite set of values $\mathbf{Y} = \mathbf{y}_i, 1 \leq i \leq L$, where $\mathbf{y}_i = [y_{i1}, y_{i2}, \ldots ., y_{iN}]^T$. The set $\mathbf{Y}$ is called the codebook or reference templates where L is the size of the codebook, and $\mathbf{y}_i$ are the codebook vectors. The size of the codebook may be considered to be equivalent to the number of levels in a scalar quantizer. In order to design such a codebook, N dimensional space is partitioned into L regions or cells $C_i, 1 \leq i \leq L$ and a vector $\mathbf{y}_i$ is associated with each cell C_i. The quantizer then assigns the codebook vector $\mathbf{y}_i$ if $\mathbf{x}$ is in C_i,

$$q(\mathbf{x}) = \mathbf{y}_i \quad \textit{if } \mathbf{x} \in C_i \tag{3.40}$$

The codebook design process is also known as training or populating the codebook. Figure 3.10 shows an example of the partitioning of a two-dimensional space ($N = 2$) for the purpose of vector quantization. The filled region enclosed by the bold lines is the cell C_i. During vector quantization, any input vector $\mathbf{x}$ that lies in the cell C_i is quantized as $\mathbf{y}_i$. The other codebook vectors corresponding to the other cells are shown by dots.

If the vector dimension, N, equals one vector quantization reduces to scalar quantization. Scalar quantization has the special property that whilst cells may have different sizes (step sizes) they all have the same shape. In vector quantization, however, cells may have different shapes which gives vector quantization an advantage over scalar quantization.

When $\mathbf{x}$ is quantized as $\mathbf{y}$, a quantization error results and, to measure the performance of a specific codebook, an overall distortion measure D is defined as,

$$D = \frac{1}{M} \sum_{i=1}^{M} d_i[\mathbf{x}, \mathbf{y}] \tag{3.41}$$

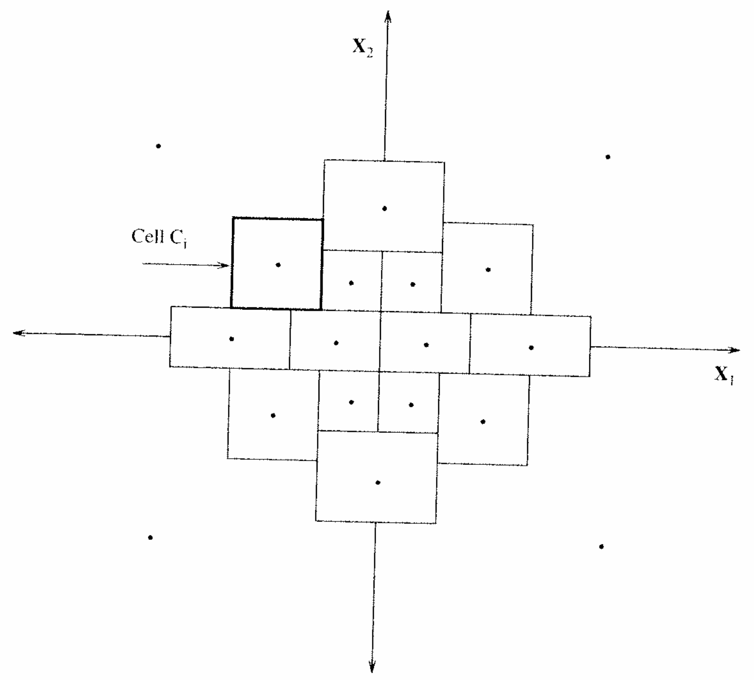

Figure 3.10 Partitioning of a two-dimensional space into 18 cells

where $d_i[\mathbf{x}, \mathbf{y}]$ is the distortion due to the i^{th} vector in the database given by,

$$d_i[\mathbf{x}, \mathbf{y}] = \frac{1}{N} \sum_{k=1}^{N} d[x_{ik}, y_{mk}] \tag{3.42}$$

where M is the number of vectors in the database and $\mathbf{y}_m$ is the quantized version of $\mathbf{x}_i$. For transmission purposes, each vector $\mathbf{y}_i$ is encoded using a codeword of binary digits of length B_i bits. The transmission rate T is given by,

$$T = BF_c \quad \textit{bits/second} \tag{3.43}$$

where,

$$B = \frac{1}{M} \sum_{i=1}^{M} B_i \quad \textit{bits/vector} \tag{3.44}$$

is the average codeword length (usually $B = B_i$), B_i is the number of bits used to encode vector $\mathbf{y}_i$ and F_c is the number of codewords transmitted per

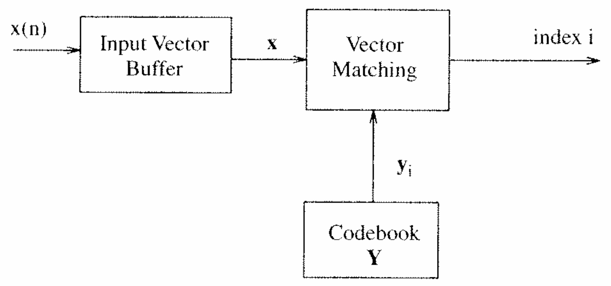

Figure 3.9 Block diagram of a simple vector quantizer

superscript T denotes transpose in vector quantization), this vector is matched with another real-valued, discrete-amplitude, N dimensional vector $\mathbf{y}$. Hence, $\mathbf{x}$ is quantized as $\mathbf{y}$, and $\mathbf{y}$ is used to represent $\mathbf{x}$. Usually, $\mathbf{y}$ is chosen from a finite set of values $\mathbf{Y} = \mathbf{y}_i, 1 \leq i \leq L$, where $\mathbf{y}_i = [y_{i1}, y_{i2}, \ldots ., y_{iN}]^T$. The set $\mathbf{Y}$ is called the codebook or reference templates where L is the size of the codebook, and $\mathbf{y}_i$ are the codebook vectors. The size of the codebook may be considered to be equivalent to the number of levels in a scalar quantizer. In order to design such a codebook, N dimensional space is partitioned into L regions or cells $C_i, 1 \leq i \leq L$ and a vector $\mathbf{y}_i$ is associated with each cell C_i. The quantizer then assigns the codebook vector $\mathbf{y}_i$ if $\mathbf{x}$ is in C_i,

$$q(\mathbf{x}) = \mathbf{y}_i \quad if \ \mathbf{x} \in C_i \tag{3.40}$$

The codebook design process is also known as training or populating the codebook. Figure 3.10 shows an example of the partitioning of a two-dimensional space ($N = 2$) for the purpose of vector quantization. The filled region enclosed by the bold lines is the cell C_i. During vector quantization, any input vector $\mathbf{x}$ that lies in the cell C_i is quantized as $\mathbf{y}_i$. The other codebook vectors corresponding to the other cells are shown by dots.

If the vector dimension, N, equals one vector quantization reduces to scalar quantization. Scalar quantization has the special property that whilst cells may have different sizes (step sizes) they all have the same shape. In vector quantization, however, cells may have different shapes which gives vector quantization an advantage over scalar quantization.

When $\mathbf{x}$ is quantized as $\mathbf{y}$, a quantization error results and, to measure the performance of a specific codebook, an overall distortion measure D is defined as,

$$D = \frac{1}{M} \sum_{i=1}^{M} d_i[\mathbf{x}, \mathbf{y}] \tag{3.41}$$

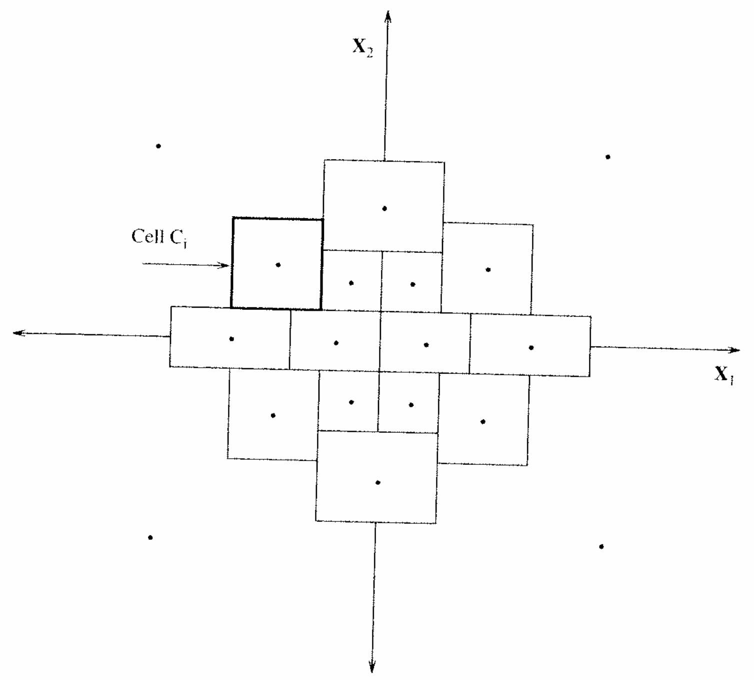

Figure 3.10 Partitioning of a two-dimensional space into 18 cells

where $d_i[\mathbf{x}, \mathbf{y}]$ is the distortion due to the i^{th} vector in the database given by,

$$d_i[\mathbf{x}, \mathbf{y}] = \frac{1}{N} \sum_{k=1}^{N} d[x_{ik}, y_{mk}] \tag{3.42}$$

where M is the number of vectors in the database and $\mathbf{y}_m$ is the quantized version of $\mathbf{x}_i$. For transmission purposes, each vector $\mathbf{y}_i$ is encoded using a codeword of binary digits of length B_i bits. The transmission rate T is given by,

$$T = BF_c \quad bits/second \tag{3.43}$$

where,

$$B = \frac{1}{M} \sum_{i=1}^{M} B_i \quad bits/vector \tag{3.44}$$

is the average codeword length (usually $B = B_i$), B_i is the number of bits used to encode vector $\mathbf{y}_i$ and F_c is the number of codewords transmitted per second. The average number of bits per vector dimension (sample) is,

$$R = \frac{B}{N} \quad \textit{bits/sample} \tag{3.45}$$

When designing a compression system, one tries to design a quantizer in which the distortion between the original and the quantized vectors is minimized for a given digital transmission rate. Therefore, during the design of a quantizer it is important to decide which type of distortion measure is likely to minimize the subjective distortion.

3.4.1 Distortion Measures

A distortion measure should be subjectively relevant, so that the differences in distortion values can be used to indicate similar differences in speech quality. However, a few dB decrease in the distortion may be quite perceptible by the ear in one case but not in another. Whilst objective distortion measures are necessary and useful tools in the design of speech coding systems, decisions on the direction for improving coder performance should be made using subjective quality testing.

Mean Squared Error

The most common distortion measure is the mean squared error (MSE) defined as,

$$d[\mathbf{x},\mathbf{y}] = \frac{1}{N}(\mathbf{x}-\mathbf{y})(\mathbf{x}-\mathbf{y})^T = \frac{1}{N}\sum_{k=1}^{N}[x_k - y_k]^2 \tag{3.46}$$

The popularity of the MSE is due to its simplicity.

Weighted Mean Squared Error

In the mean squared error method, it is assumed that the distortion contributed by each element of the vector $\mathbf{x}$ is weighted equally. In general, unequal weights can be introduced to render contributions of certain elements to the distortion more important than others. Hence, a general weighted mean squared error is defined by,

$$d_w[\mathbf{x},\mathbf{y}] = (\mathbf{x}-\mathbf{y})W(\mathbf{x}-\mathbf{y})^T \tag{3.47}$$

where W is a positive weighting matrix.

Perceptually Determined Distortion Measures

For high bit rates and hence small distortions, reasonable distortion measures, including the two mentioned above, perform well with similar performances.

Furthermore, they correlate well with subjective judgements of speech quality. However, as the bit rate decreases and distortion increases, simple distortion measures may not be related to the subjective quality of speech. Since the main application of vector quantization is expected to be at low bit-rates, it is very important to develop and use distortion measures that are better correlated with human auditory behaviour. A number of perceptually based distortion measures have been developed [10, 11, 12]. Since the main aim is to produce the highest speech quality possible at a given bit rate, it is essential to use a distortion measure that correlates well with human perception.

3.4.2 Codebook Design

When designing an L level codebook, N dimensional space is partitioned into L cells C_i, $1 \le i \le L$, and each cell C_i is assigned a vector $\mathbf{y}_i$. The quantizer chooses the codebook vector $\mathbf{y}_i$ if $\mathbf{x}$ is in C_i. To optimize a quantizer, the distortion in equation (3.41) is minimized over all L levels. There are two necessary conditions for optimality. The first condition is that the optimum quantizer finds a matching vector for every input vector by minimizing the distortion criterion. That is, the quantizer chooses the codebook vector that results in the minimum distortion with respect to $\mathbf{x}$ [13].

$$q(\mathbf{x}) = \mathbf{y}_i \quad \textit{if}\, d[\mathbf{x},\mathbf{y}_i] \le d[\mathbf{x},\mathbf{y}_j], \quad j \ne i, \quad 1 \le j \le L. \tag{3.48}$$

The second necessary condition for optimality is that each codebook vector $\mathbf{y}_i$ is optimized to give the minimum average distortion in cell C_i.

$$D_i = E\{[d(\mathbf{x},\mathbf{y}_i)|\ \mathbf{x}\epsilon C_i]\} = \int_{\mathbf{x}\epsilon C_i} d[\mathbf{x},\mathbf{y}_i]p(x)dx \tag{3.49}$$

where $p(x)$ is the probability density function of vectors that result in the quantized vector $\mathbf{y}_i$ in cell (cluster) C_i.

Vector $\mathbf{y}_i$ is called the centroid of the cell C_i. Optimization of the centroid of a particular cell depends on the definition of the distortion measure. For either the mean squared error or the weighted mean squared error, distortion in each cell is minimized by,

$$y_{in} = \frac{1}{M_i}\sum_{k=1}^{M_i} x_{kn} \quad \mathbf{x}\epsilon C_i \tag{3.50}$$

where y_{in} $\{n = 1, 2, \ldots, N\}$ is the n^{th} element of the centroid $\mathbf{y}_i$ of the cluster C_i. That is, $\mathbf{y}_i$ is simply the sample mean of all the training vectors M_i contained in cell C_i. One of the most popular methods for codebook design is an iterative clustering algorithm known as the K-means algorithm [13] (also known as

Lloyd's algorithm [14]). The algorithm divides the set of training vectors into L clusters C_i in such a way that the two necessary conditions for optimality are satisfied.

K-means Algorithm

Given that m is the iteration index and C_{i_m} is the i^{th} cluster at iteration m with $\mathbf{y}_{i_m}$ its centroid:

1. Initialization: Set $m = 0$ and choose a set of initial codebook vectors $\mathbf{y}_{i_0}$, $1 \leq i \leq L$.
2. Classification: Partition the set of training vectors $\mathbf{x}_n$, $1 \leq n \leq M$, into the clusters C_i by the nearest neighbour rule,

$$\mathbf{x} \epsilon C_{i_m} \quad \textit{if} \quad d[\mathbf{x}, \mathbf{y}_{i_m}] \leq d[\mathbf{x}, \mathbf{y}_{j_m}] \quad \textit{for all} \quad j \neq i.$$

3. Codebook updating: $m \to m + 1$. Update the codebook vector of every cluster by computing the centroid of training vectors in each cluster.
4. Termination test: If the decrease in the overall distortion at iteration m relative to $m - 1$ is below a certain threshold, stop; otherwise go to step 2.

Any other reasonable termination test may be used for step 4.

The above algorithm converges to a local optimum [14, 15]. Furthermore, any such solution is in general not unique [16]. Global optimality may be achieved approximately by initializing the codebook vectors to different values and repeating the above algorithm for several sets of initializations and then choosing the codebook that results in the minimum overall distortion.

3.4.3 Codebook Types

Vector quantization can offer substantial performance over scalar quantization at very low bit-rates. However, these advantages are obtained at considerable computational and storage costs. In order to compromise between the computation and storage costs, and quantizer performance, a number of codebook types have been developed. Some codebooks are precomputed and do not change while being used; others may be updated during quantization. Here, we will briefly explain some of the widely-used codebooks in speech coding.

Full Search Codebook

A full search codebook is one where during the quantization process each input vector is compared against all of the candidate vectors in the codebook. This process is called full search or exhaustive search. The computation and

storage requirements of a typical full search codebook can be calculated as follows. If each vector in a full search codebook is represented by $B = RN$ *bits* for transmission, then the number of vectors in the codebook is given by,

$$L = 2^B = 2^{RN} \tag{3.51}$$

where N is the vector dimension in the codebook. In many applications, computing the absolute value of the quantization error may not be necessary as the main concern is to select the best performing vector. So a relative performance rather than the absolute error is required. It is therefore possible to compute the similarity rather than the difference between the input vector and the codebook vectors. Therefore, assuming that the cross-correlation of the input vector with each of the codebook candidates is computed and the one resulting in the highest cross-correlation value is selected as the quantized value of the input vector, the computation cost (assuming that all the vectors are normalized, as differences in the energy levels will give misleading cross-correlation values) is given by,

$$Com_{fs} = N2^{RN} \quad \textit{multiply} - \textit{add per input vector} \tag{3.52}$$

From this, we can also calculate the storage required for the codebook vectors as,

$$M_{fs} = NL = N2^B = N2^{RN} \quad \textit{locations} \tag{3.53}$$

It can be seen from the above expressions that the computation and storage requirements of a full search codebook are exponentially related to the number of bits in the codewords.

For a 16-bit fixed point processor the storage M_{fs} in bytes is given by $2 \times M_{fs}$ and for a 32-bit floating point implementation, storage is $4 \times M_{fs}$. In general, the storage is defined by the required number of words each corresponding to a location. For example if $N = 10$ and $R = 1$ the number of codebook vectors L will be $2^{NR} = 1024$. The number of multiply–add operations needed will be $N2^{RN} = 10 \times 1024 = 10\,240$ per input vector. Assuming a sampling frequency of 8 kHz, the number of vectors per second will be $8000/10 = 800$. Therefore, the computation cost will be $800 \times 10\,240 = 8.192 \times 10^6$ multiply–add per second. The storage requirement will be $N2^{RN} = 10 \times 2^{10} = 10\,240$ words (locations).

Using the K-means algorithm, a full search codebook can be optimized (trained) in two possible ways.

- **Method 1:** The process starts with two initial vectors which may be chosen randomly or calculated as centroids of the two halves of the large training database. The K-means algorithm is used to optimize the initial vectors. After the optimization of each of the two initial vectors $\mathbf{v}_1 = [v_{11}, v_{12}, v_{13}, \ldots, v_{1N}]$ and $\mathbf{v}_2 = [v_{21}, v_{22}, v_{23}, \ldots, v_{2N}]$ with dimensions N, each is split into two further vectors as,

$$\mathbf{v}_3 = \mathbf{v}_1 - \varepsilon_1, \quad \mathbf{v}_4 = \mathbf{v}_1 + \varepsilon_1, \quad \mathbf{v}_5 = \mathbf{v}_2 - \varepsilon_2, \quad \mathbf{v}_6 = \mathbf{v}_2 + \varepsilon_2,$$

where $\varepsilon_1 = [e_{11}, e_{12}, e_{13}, \ldots, e_{1N}]$ and $\varepsilon_2 = [e_{21}, e_{22}, e_{23}, \ldots, e_{2N}]$. In most cases $\varepsilon_1 = \varepsilon_2$.
The vectors from the second stage are again optimized using the K-means algorithm and split into further vectors and so on until the number of optimized vectors is equal to the desired number. The optimization process can also be terminated by comparing the overall quantization noise performance of the codebook against a threshold.
During the optimization of a full search codebook using the above method, it is important to check that all of the optimized vectors are in the densely-populated areas and do not diverge into outer areas where their use will be wasted. In such cases the perturbation vector ε is modified to change the direction of the resultant vector.

- **Method 2:** The second method of optimization starts with randomly-selected vectors from the training database. The number of initial vectors is larger than the final desired number of vectors in the codebook. Using the K-means algorithm these vectors are optimized. After the first optimization process, the least used vectors are discarded from the codebook. The remaining vectors are then optimized and the least used vectors are again discarded from the optimized codebook. This process continues until the final size of the codebook is reached. Here, the number of vectors discarded at each stage and the number of optimization iterations may vary with the application but the initial size of the codebook should at least be 1.5 times the final size and the number of discarding stages should not be fewer than five or six. The number of vectors discarded in each stage should be reduced to increase the accuracy of optimization.

Binary Search Codebook

Binary search [17], known in the pattern recognition literature as hierarchical clustering [14], is a method for partitioning space in such a way that the search for the minimum distortion code-vector is proportional to $\log_2 L$ rather than L. In speech coding literature, binary search codebooks are also called tree codebooks or tree search codebooks.

In a binary search codebook, N dimensional space is first divided into two regions (using the K-means algorithm with two initial vectors), then each of the two regions is further divided into two subregions, and so on, until the space is divided into L regions or cells. Here, L is restricted to be a power

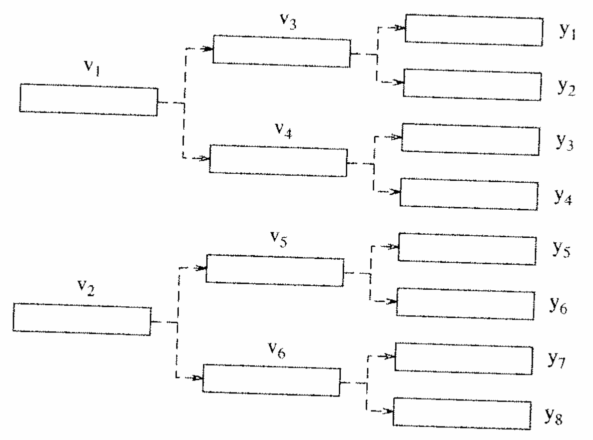

Figure 3.11 Binary splitting into eight cells

of 2, $L = 2^B$, where B is an integer number of bits. Each region is associated with a centroid. Figure 3.11 shows the division of space into $L = 8$ cells. At the first binary division $\mathbf{v}_1$ and $\mathbf{v}_2$ are calculated as the centroids of the two halves of the total space to be covered. At the second binary division four centroids are calculated as $\mathbf{v}_3$ to $\mathbf{v}_6$. The centroids of the regions after the third binary division are the actual codebook vectors $\mathbf{y}_i$. An input vector $\mathbf{x}$ is quantized, searching the tree along a path that gives the minimum distortion at each node in the path. Again assuming N multiply–adds for each distortion computation, the computation cost will be,

$$Com_{bs} = 2N \log_2 L = 2NB \quad multiply - add\ per\ input\ vector \tag{3.54}$$

At each stage, the input vector is compared against only two candidates. This makes the computation cost a linear function of the number of bits in the codewords.

The total storage cost, on the other hand, has gone up significantly,

$$M_{bs} = 2N(L-1) \quad locations \tag{3.55}$$

or,

$$M_{bs} = N \sum_{i=1}^{B} 2^i \quad locations \tag{3.56}$$

A tree search codebook need not be a binary search codebook. In other words the number of splitting stages may be less than the number of bits, B, in the codeword. In this case, each vector from the previous stage may point to more than two vectors in the current stage. This can be seen as a compromise between the extreme cases of low computation cost with high storage (binary codebook) and high computation cost with low storage requirement (full search codebook).

During the training of a binary codebook, at each stage of splitting using the K-means algorithm and method 1, the resultant optimum codebooks are stored. The database is also split into sections represented by each of the resultant vectors. When the vectors are further split, each new pair of vectors is optimized using the section of the database represented by their mother vector. This process continues until the final size codebook is reached and optimized.

Cascaded Codebooks

The major advantage of a binary search codebook is the substantial decrease in its computational cost, relative to a full search codebook, with a relatively small decrease in performance. However, the storage required for a binary search codebook relative to a full search codebook is nearly doubled. Cascaded vector quantization is a method intended to reduce storage as well as computational costs [18, 13]. A two-stage cascaded vector quantization is shown in Figure 3.12. Cascaded vector quantization consists of a sequence of vector quantization stages, each operating on the error signal of the previous stage. The input vector **x** is first quantized using a B_1 bit L_1 level vector

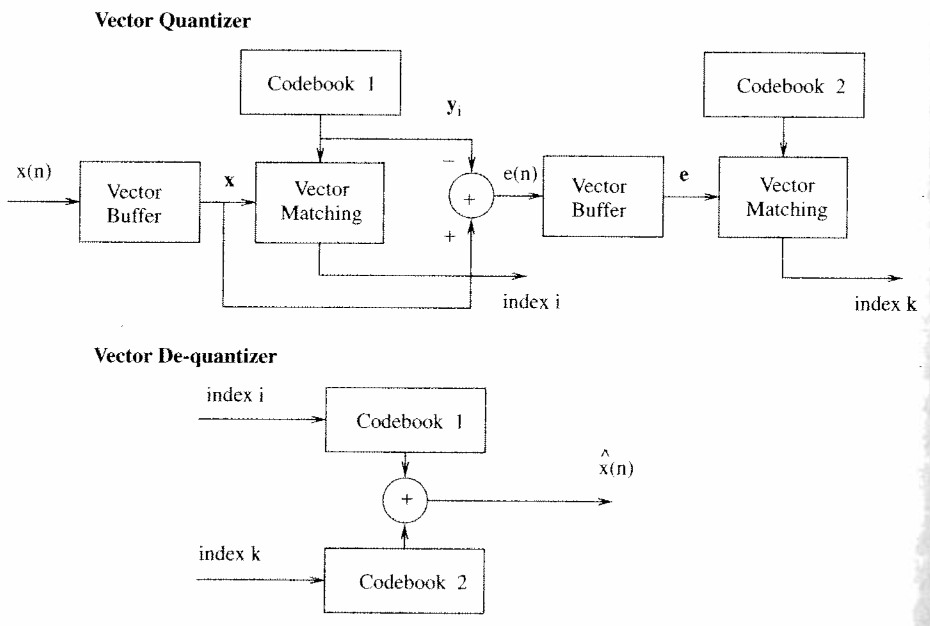

Figure 3.12 A two-stage cascaded vector quantizer

quantizer and the resulting error signal is then used in the input to a B_2 bit L_2 level second vector quantizer. The sum of the two quantized vectors results in the quantized value of the input vector **x**.

The computation and storage costs for a k -stage cascaded vector quantization are respectively,

$$Com_{cc} = N(L_1 + L_2 + \ldots + L_k) \quad multiply - add\ per\ input\ vector \tag{3.57}$$

$$M_{cc} = N(L_1 + L_2 + \ldots + L_k) \quad locations \tag{3.58}$$

Assuming $L_1 = 2^{B_1}$, $L_2 = 2^{B_2}$ and $L_k = 2^{B_k}$ and the total number of bits per input vector $B = B_1 + B_2 \ldots + B_k$, we can see that the number of candidate vectors searched in a cascaded codebook for each input vector is less than in a full search codebook,

$$\sum_{n=1}^{k} 2^{B_n} < 2^B \quad if \quad B = \sum_{n=1}^{k} B_n \quad and \ k > 1 \tag{3.59}$$

We can also see that the storage of a cascaded codebook is less than that required by a binary codebook,

$$N\left(\sum_{n=1}^{k} 2^{B_n}\right) < N\left(\sum_{i=1}^{B} 2^i\right) \quad for \ k > 1 \tag{3.60}$$

Given the condition that the total number of bits used at various stages of a cascaded codebook is B, both computation and storage requirements reduce with an increase in the number of stages.

Split Codebooks

In all of the above codebook types an N dimensional input vector is directly matched with N dimensional codebook entries. In a split vector quantization scheme, an N dimensional input vector is first split into P parts where $P > 1$. For each part of the split vector a separate codebook is used and each part may be vector quantized independently of the other parts using B_p bits. Assuming a vector is split into P equal parts and vector quantized using B_p bits for each part, the computation and storage requirements can be calculated as follows:

$$Com_{ss} = \frac{N}{P}(L_1 + L_2 + \ldots + L_P) \quad multiply - add\ per\ input\ vector \tag{3.61}$$

where $L_p = 2^{B_p}$ for $p = 1, 2, \ldots, P$. Similarly, the storage is given by:

$$M_{ss} = \frac{N}{P}(L_1 + L_2 + \ldots + L_P) \quad locations \tag{3.62}$$

The usefulness of a split vector quantization is in its flexibility in choosing the dimension of each split part and in the allocation of the overall bits per input vector to these parts according to the perceptual importance of the vector elements contained in each split part.

Gain Shape Codebooks

In the earlier discussion of scalar quantization, it was mentioned that the variance of the input speech signal affected the performance of the quantizer. This is also true in the case of a vector quantizer. For example, if the input signal variance is fixed at a certain value, all of the codebook entries will have the same variance and differ only in the shape of vector elements. In addition, if we assume that the same number of shape combinations is repeated with another variance level at the input, the number of codebook entries would have to be doubled to cover the vector shapes at two different energy levels. Therefore, if the input vectors have a large dynamic range, the required codebook size may be too large for practical implementation in both computation and storage. This problem can be overcome by using the same idea that is used in scalar quantization: each input vector is normalized to a certain variance level (usually unity), and then its unit variance shape is vector quantized using a shape codebook containing candidate vectors with unity variance. The original variance of the input vector is separately quantized and transmitted to the de-quantizer for correct scaling. This process is called gain-shape vector quantization. A block diagram of a gain-shape vector quantizer is shown in Figure 3.13. The gain of the input vector is usually calculated and quantized using a scalar quantizer either before or during the search of the shape codebook.

If the gain of the input vector is to be calculated and quantized before finding its shape then the quantized gain is calculated as:

$$\hat{\sigma}_x = Q\left[\sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2}\right] \tag{3.63}$$

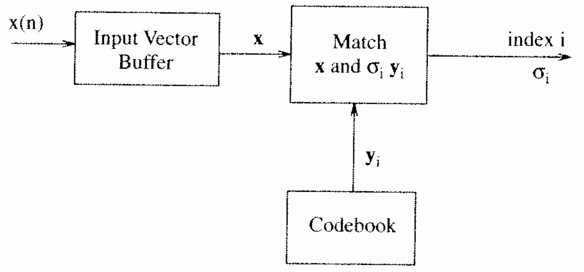

Figure 3.13 Gain-shape vector quantizer

where

$$Q[.]$$

denotes quantization operation. The shape codebook is then searched and the codebook vector which minimizes the expression,

$$D_k = \sum_{i=1}^{N}(x_i - \hat{\sigma}_x y_{ki})^2 \quad k = 1, 2, \ldots, L \tag{3.64}$$

is chosen for transmission. This search scheme, called open loop, is not optimum. Better performance can be achieved with a closed loop scheme where the shape is first found and then the corresponding gain is quantized before computing the final error. Here, we assume an optimum gain σ_k to be used for each of the L shape codebook entries and compute the corresponding distortion D_k as:

$$D_k = \sum_{i=1}^{N}(x_i - \sigma_k y_{ki})^2 \quad k = 1, 2, \ldots, L \tag{3.65}$$

We wish to find a vector $\mathbf{y}_k$ from the shape codebook with a gain value of σ_k such that the corresponding distortion D_k is minimized. However, we have two unknowns, namely, $\mathbf{y}_k$ and σ_k. To find σ_k in terms of $\mathbf{y}_k$ we differentiate (3.65) with respect to σ_k and set it to zero for minimum error gain. This gives the following σ_k for the codebook vector $\mathbf{y}_k$ in relation to an input vector $\mathbf{x}$,

$$\sigma_k = \frac{\sum_{i=1}^{N}(x_i y_{ki})}{\sum_{i=1}^{N} y_{ki}^2} \tag{3.66}$$

If we substitute (3.66) into (3.65) we can write the distortion D_k independently of σ_k as,

$$D_k = \sum_{i=1}^{N}(x_i)^2 - \frac{\left(\sum_{i=1}^{N} x_i y_{ki}\right)^2}{\sum_{i=1}^{N} y_{ki}^2} \quad k = 1, 2, \ldots, L \tag{3.67}$$

The first term of D_k in equation (3.67) does not change with k, and hence it is not computed during the search of the shape. The shape is found by maximizing only the second term in (3.67). During the codebook search process, the most likely shape values are found by maximizing the second term in equation (3.67). Then, corresponding gain values given by (3.66) are computed and quantized. Finally, each shape vector scaled by its quantized gain is compared with the input vector. This whole process can be simplified with only a small increase in the quantization error by computing the second part of equation (3.67) for all k to select the best shape vector without quantizing its gain (assuming that gain quantization noise will not, in general, render other vectors more favourable). In this case only one shape vector is considered which does not require further comparisons after the gain quantization process.

Adaptive Codebooks

The above discussed codebooks do not vary with time. Therefore, it is extremely important to train these codebooks for optimal performance with varying time and hence varying input vector characteristics. One way of making a codebook track the input vector characteristics with time is to make the codebook adaptive. As in the case of an adaptive scalar quantizer, the adaptation of a codebook can be achieved using either forward or backward schemes.

In a forward adaptive vector quantizer, the codebook is updated with respect to the input vectors before the quantization process, which requires some side information to be transmitted to the de-quantizer for compatible adaptation necessary for correct recovery of the signal.

In the case of a backward adaptive quantizer, the codebook is updated by the appropriately transformed most recent quantizer output vectors. In this case, no side information is needed since the same update process can be performed at the de-quantizer using the previously recovered vectors.

An adaptive codebook is usually used in cascade with other (generally, fixed) codebooks, which provide the initial vectors to the adaptive codebook as well as helping to speed up adaptation when significant signal variations occur. An adaptive codebook in a two-stage cascaded vector quantizer is shown in Figure 3.14. The first stage can be an adaptive codebook followed by a fixed second stage codebook. The adaptive codebooks used in these configurations are called *predictor codebooks* and the whole process is called *predictive* or *differential vector quantization*.

3.4.4 Training, Testing and Codebook Robustness

An important part of the codebook design is the training process used to populate the codebook. The training process simply optimizes a codebook for given training data by calculating the centroids of the cells. Because the K-means algorithm is not guaranteed to result in a codebook that is globally optimum, it is often suggested that one repeats the algorithm with a number of different initial sets of codebook vectors [19].

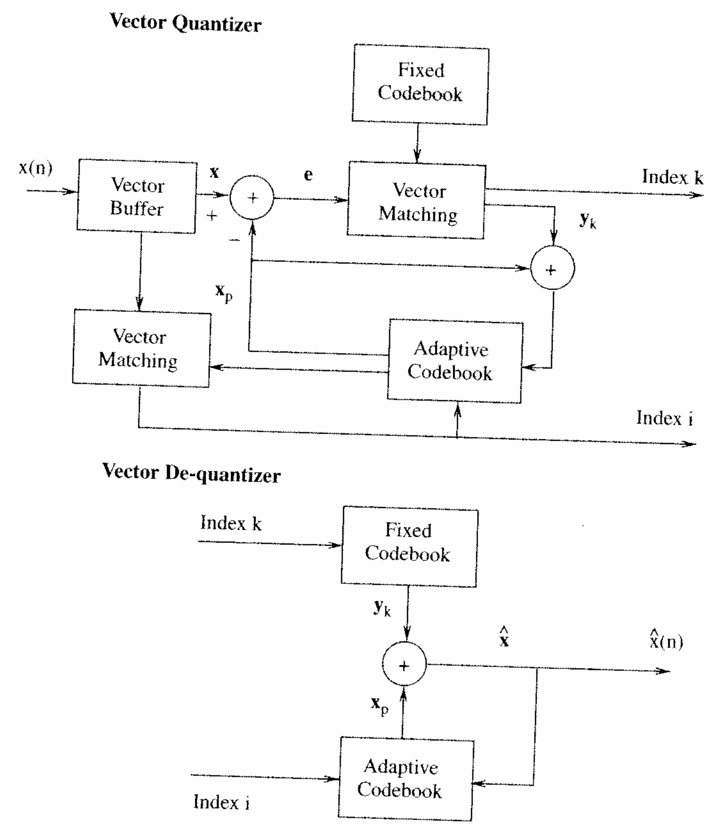

Figure 3.14 Adaptive vector quantizer in a cascaded setup

After designing a codebook to match a given set of training data, it is important to test the performance of that codebook on data that was not used in the training. Testing only on the training data will always give better performance than the codebook will actually give in practice.

The robustness of a codebook can be measured by measuring its performance on data whose distribution is different from that of the training data. In practice, one cannot usually predict all of the situations under which a quantizer will be used and so the distribution of the actual data may be different from that of the training data. There are two major types of variation that affect the design and operational performance of a codebook: input signal variability and digital transmission channel errors.

Signal variability can be further classified as speaker variability and environmental variability. Speaker variability covers the changes in the input signal due to a change in the speaker's voice and may, for example, be due to multiple speakers or the health conditions of each speaker. Environmental variability, on the other hand, refers to the background noise level and type. For a given bit rate and speaker, a speaker-dependent codebook performs better than a speaker-independent codebook. One method of maximizing the performance of a codebook is to design a speaker-independent codebook initially and then, as the system is used, have it adapt to the speech of new speakers [20]. In such a system, automatic adaptation to the background noise environment of the speaker is also possible.

As in the case of a scalar quantizer transmission channel errors affects the performance of a vector quantizer. Channel errors translate directly into distortion at the output, depending on the channel error rate. In general, vector quantization systems tend to be less robust to random channel errors than scalar quantizers, as a single bit error can cause all of the values represented by that vector to be in error.

3.5 Summary

Many quantization schemes have been designed and deployed in practice. With the advancement in the DSP technology which allowed more processing power as well as storage, vector quantization techniques have become widespread. Vector quantization schemes are very effective in reducing the bit rate of the signal that is being quantized at the expense of increased implementation complexity. It is however crucial that the codebooks are trained to match the incoming signal. As the training processes are usually applied off-line they can be allowed to run for a long time so that the best codebooks are obtained. In parallel with significant advances in the DSP technology, the implementation cost of various codebooks has been optimized by developing intelligent search algorithms as well as different types of codebook.

Bibliography

[1] L. Rabiner and R. Schafer (1978) *Digital Processing of Speech Signals.* Englewood Cliffs, NJ: Prentice-Hall

[2] K. W. Cattermole (1973) *Principles of Pulse Code Modulation.* London: Illiffe

[3] P. F. Panter and W. Dite (1951) 'Quantization distortion in PCM with non-uniform spacing of levels', in *Proc. IRE*, 39:44–8.

[4] J. Max (1960) 'Quantising for minimum distortion', in *IRE Trans. on Information Theory*, 6:7–12.

[5] M. D. Paez and T. H. Glisson (1972) 'Minimum mean-squared-error quantization in speech PCM and DPCM systems', in *IEEE Trans. on Communications*, 20(4):225–30.

[6] P. Noll (1974) 'Adaptive quantizing in speech coding systems', in *IEEE Int. Zurich Seminar on Digital Comm.*, pp. B3.1–6, March.

[7] R. W. Stroh (1970) 'Optimum and adaptive DPCM', Ph.D. thesis, Polytechnic Inst. of Brooklyn, USA.

[8] N. S. Jayant (1974) 'Adaptive quantization with a one-bit memory', in *Bell Sys. Technical Journal*, 52.

[9] N. S. Jayant (1974) 'Digital coding of speech waveforms: PCM, DPCM and DM quantizers', in *IEEE Proc.*, 62(5):611–632.

[10] N. S. Jayant and P. Noll (1984) *Digital Coding of Waveforms: Principles and applications to speech and video.* New Jersey: Prentice-Hall

[11] M. Schroeder and B. Atal (1979) 'Predictive coding of speech signals and subjective error criteria', in *IEEE Trans. on Acoust., Speech and Signal Processing*, 27:247–54.

[12] V. Viswanathan *et al.* (1983) 'Objective speech quality evaluation of medium band and narrow band real-time speech coders', in *Proc. of Int. Conf. on Acoust., Speech and Signal Processing*, pp. 543–6.

[13] J. Makhoul, S. Roucos, and H. Gish (1985) 'Vector quantisation in speech coding', in *Proc. of IEEE*, 23:1551–88.

[14] M. R. Anderberg (1973) *Cluster Analysis for Applications*, p. 22. Academic Press

[15] Y. Linde, A. Buzo, and R. Gray (1980) 'An algorithm for vector quantiser design', in *IEEE Trans. on Communications*, 28(1):84–95.

[16] R. Gray and E. Karnin (1982) 'Multiple local in vector quantization', in *IEEE Trans. on Information Theory*, 28:256–61.

[17] A. Buzo, AH Gray Jr., R. M. Gray, and J. D. Markel (1980) 'Speech coding based upon vector quantisation', in *IEEE Trans. on Acoust., Speech and Signal Processing*, 28(5):562–74.

[18] S. Roucos, R. Schwartz, and J. Makhoul (1982) 'Vector quantization for very low bit rate coding of speech', in *Proc. of Globecom*, pp. 1074–8.

[19] R. Gray (1984) 'Vector quantization', in *IEEE ASSP Magazine*, 1:4–28.

[20] D. B. Paul (1983) 'An 800 bps adaptive vector quantisation vocoder using a perceptual distance measure', in *Proc. of Int. Conf. on Acoust., Speech and Signal Processing*, pp. 73–6.