```
void fft2 (float *x, float *y, int n, int m)
{
    int n1, n2, i, j, k, l;
    float  xt, yt, c, s;
    double e, a;

    /* Loop through all m stages */
    n2 = n;
    for (k = 0; k < m; k++) {
        n1 = n2;
        n2 = n2 / 2;
        e = PI2 / n1;
        for (j = 0; j < n2; j++) {
            /* Compute Twiddle factors */
            a = j * e;
            c = (float) cos (a);
            s = (float) sin (a);

            /* Do the butterflies */
            for (i = j; i < n; i += n1) {
                l = i + n2;
                xt = x[i] - x[l];
                x[i] = x[i] + x[l];
                yt = y[i] - y[l];
                y[i] = y[i] + y[l];
                x[l] = c * xt + s * yt;
                y[l] = c * yt - s * xt;
            }
        }
    }

    /* Bit reversal: descrambling */
    j = 0;
    for (i = 0; i < n - 1; i++) {
        if (i < j) {
            xt = x[j];
            x[j] = x[i];
            x[i] = xt;
            xt = y[j];
            y[j] = y[i];
            y[i] = xt;
        }
        k = n / 2;
        while (k <= j) {
            j -= k;
            k /= 2;
        }
        j += k;
    }
}
```

Figure 5.16 C source for a decimation-in-time radix-2 FFT. Before calling the subroutine, x and y contain the real and imaginary parts of the input signal respectively. After returning from the subroutine, x and y contain the real and imaginary parts of the Fourier transform of the input signal. n is the length of the FFT and is related to m by $n = 2^m$.

To compute the inverse FFT an additional routine is not necessary; it can be computed with the subroutine above. To see that, we expand the DFT in Eq. (5.56) into its real and imaginary parts:

$$X_R[k]+jX_I[k]=\sum_{n=0}^{N-1}\left(x_R[n]+jx_I[n]\right)e^{-j2\pi nk/N} \tag{5.80}$$

take complex conjugate and multiply by j to obtain

$$X_I[k]+jX_R[k]=\sum_{n=0}^{N-1}\left(x_I[n]+jx_R[n]\right)e^{j2\pi nk/N} \tag{5.81}$$

which has the same functional form as the expanded inverse DFT of Eq. (5.57)

$$x_R[k]+jx_I[k]=\frac{1}{N}\sum_{n=0}^{N-1}\left(X_R[n]+jX_I[n]\right)e^{j2\pi nk/N} \tag{5.82}$$

so that the inverse FFT can be computed by calling `fft (xi, xr, n)` other than the $(1/N)$ factor.

Often the input signal $x[n]$ is real, so that we know from the symmetry properties of Table 5.5 that its Fourier transform is Hermitian. This symmetry can be used to compute the length-N FFT more efficiently with a length $(N/2)$ FFT. One way of doing so is to define $f[n]=x[2n]$ to represent the even-indexed samples of $x[n]$, and $g[n]=x[2n+1]$ the odd-indexed samples. We can then define a length $(N/2)$ complex signal $h[n]$ as

$$h[n]=f[n]+jg[n]=x[2n]+jx[2n+1] \tag{5.83}$$

whose DFT is

$$H[k]=F[k]+jG[k]=H_R[k]+jH_I[k] \tag{5.84}$$

Since $f[n]$ and $g[n]$ are real, their transforms are Hermitian and thus

$$H^*[-k]=F^*[-k]-jG^*[-k]=F[k]-jG[k] \tag{5.85}$$

Using Eqs. (5.84) and (5.85), we can obtain $F[k]$ and $G[k]$ as a function of $H_R[k]$ and $H_I[k]$:

$$F[k]=\frac{H[k]+H^*[-k]}{2}=\left(\frac{H_R[k]+H_R[-k]}{2}\right)+j\left(\frac{H_I[k]-H_I[-k]}{2}\right) \tag{5.86}$$

$$G[k]=\frac{H[k]-H^*[-k]}{2j}=\left(\frac{H_I[k]+H_I[-k]}{2}\right)-j\left(\frac{H_R[k]-H_R[-k]}{2}\right) \tag{5.87}$$

As shown in Eq. (5.77), $X[k]$ can be obtained as a function of $F[k]$ and $G[k]$

$$X[k] = F[k] + G[k]W_N^{-k} \tag{5.88}$$

so that the DFT of the real sequence $x[n]$ is obtained through Eqs. (5.83), (5.86), (5.87) and (5.88). The computational complexity is a length ($N/2$) complex FFT plus N real multiplies and $3N$ real adds.

5.3.4. Circular Convolution

The convolution of two periodic signals is not defined according to Eq. (5.12). Given two periodic signals $x_1[n]$ and $x_2[n]$ with period N, we define their *circular convolution* as

$$y[n] = x_1[n] \otimes x_2[n] = \sum_{m=0}^{N-1} x_1[m]x_2[n-m] = \sum_{m=<N>} x_1[m]x_2[n-m] \tag{5.89}$$

where $m =< N >$ in Eq. (5.89) means that the sum lasts only one period. In fact, the sum could be over any N consecutive samples, not just the first N. Moreover, $y[n]$ is also periodic with period N. Furthermore, it is left to the reader to show that

$$Y[k] = X_1[k]X_2[k] \tag{5.90}$$

i.e., the DFT of $y[n]$ is the product of the DFTs of $x_1[n]$ and $x_2[n]$.

An important application of the above result is the computation of a regular convolution using a circular convolution. Let $x_1[n]$ and $x_2[n]$ be two signals such that $x_1[n] = 0$ outside $0 \le n < N_1$, and $x_2[n] = 0$ outside $0 \le n < N_2$. We know that their regular convolution $y[n] = x_1[n] * x_2[n]$ is zero outside $0 \le N_1 + N_2 - 1$. If we choose an integer N such that $N \ge N_1 + N_2 - 1$, we can define two periodic signals $\tilde{x}_1[n]$ and $\tilde{x}_2[n]$ with period N such that

$$\tilde{x}_1[n] = \begin{cases} x_1[n] & 0 \le n < N_1 \\ 0 & N_1 \le n < N \end{cases} \tag{5.91}$$

$$\tilde{x}_2[n] = \begin{cases} x_2[n] & 0 \le n < N_2 \\ 0 & N_2 \le n < N \end{cases} \tag{5.92}$$

where $x_1[n]$ and $x_2[n]$ have been *zero padded*. It can be shown that the circular convolution $\tilde{y}[n] = \tilde{x}_1[n] \otimes \tilde{x}_2[n]$ is identical to $y[n]$ for $0 \le n < N$, which means that $y[n]$ can be obtained as the inverse DFT of $\tilde{Y}[k] = \tilde{X}_1[k]\tilde{X}_2[k]$. This method of computing the regular convolution of two signals is more efficient than the direct calculation when N is large. While the crossover point will depend on the particular implementations of the FFT and convolution, as well as the processor, in practice this has been found beneficial for $N \ge 1024$.

5.3.5. The Discrete Cosine Transform (DCT)

The Discrete Cosine Transform (DCT) is widely used for speech processing. It has several definitions. The DCT-II $C[k]$ of a real signal $x[n]$ is defined by:

$$C[k]=\sum_{n=0}^{N-1}x[n]\cos\left(\pi k(n+1/2)/N\right) \quad \text{for } 0\le k<N \tag{5.93}$$

with its inverse given by

$$x[n]=\frac{1}{N}\left\{C[0]+2\sum_{k=1}^{N-1}C[k]\cos\left(\pi k(n+1/2)/N\right)\right\} \quad \text{for } 0\le n<N \tag{5.94}$$

The DCT-II can be derived from the DFT by assuming $x[n]$ is a real periodic sequence with period $2N$ and with an even symmetry $x[n]=x[2N-1-n]$. It is left to the reader to show, that $X[k]$ and $C[k]$ are related by

$$X[k]=2e^{j\pi k/2N}C[k] \quad \text{for } 0\le k<N \tag{5.95}$$

$$X[2N-k]=2e^{-j\pi k/2N}C[k] \quad \text{for } 0\le k<N \tag{5.96}$$

It is left to the reader to prove Eq. (5.94) is indeed the inverse transform using Eqs. (5.57), (5.95), and (5.96). Other versions of the DCT-II have been defined that differ on the normalization constants but are otherwise the same.

There are eight different ways to extend an N-point sequence and make it both periodic and even, such that can be uniquely recovered. The DCT-II is just one of the ways, with three others being shown in Figure 5.17.

The DCT-II is the most often used discrete cosine transform because of its *energy compaction*, which results in its coefficients being more concentrated at low indices than the DFT. This property allows us to approximate the signal with fewer coefficients [10].

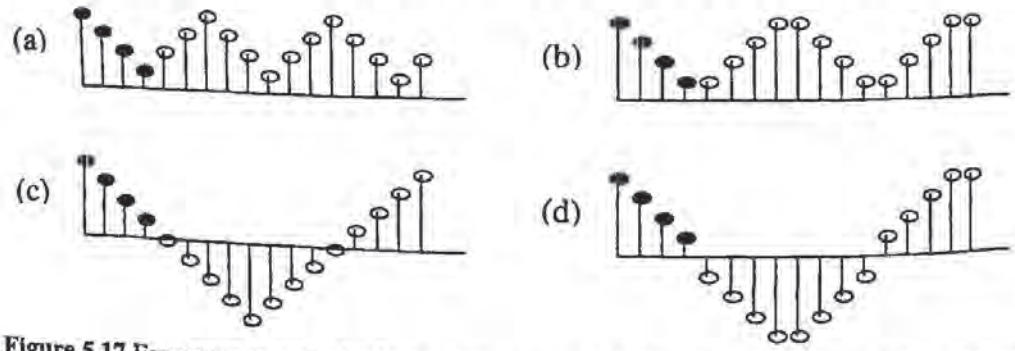


Figure 5.17 Four ways to extend a four-point sequence $x[n]$ to make it both periodic and have even symmetry. The figures in (a), (b), (c) and (d) correspond to the DCT-I, DCT-II, DCT-III and DCT-IV respectively.

From Eq. (5.95) and (5.96) we see that the DCT-II of a real sequence can be computed with a length-2*N* FFT of a real and even sequence, which in turn can be computed with a length (*N*/2) complex FFT and some additional computations. Other fast algorithms have been derived to compute the DCT directly [15], using the principles described in Section 5.3.3.1. Two-dimensional transforms can also be used for image processing.

5.4. DIGITAL FILTERS AND WINDOWS

We describe here the fundamentals of digital filter design and study *finite-impulse response* (FIR) and *infinite-impulse response* (IIR) filters, which are special types of linear time-invariant digital filters. We establish the time-frequency duality and study the ideal low-pass filter (frequency limited) and its dual window functions (time limited).

5.4.1. The Ideal Low-Pass Filter

It is useful to find an impulse response $h[n]$ whose Fourier transform is

$$H(e^{j\omega}) = \begin{cases} 1 & |\omega| < \omega_0 \\ 0 & \omega_0 < |\omega| < \pi \end{cases} \tag{5.97}$$

which is the ideal *low-pass* filter because it lets all frequencies below ω_0 pass through unaffected and completely blocks frequencies above ω_0. Using the definition of Fourier transform, we obtain

$$h[n] = \frac{1}{2\pi}\int_{-\omega_0}^{\omega_0} e^{j\omega n} d\omega = \frac{\left(e^{j\omega_0 n} - e^{-j\omega_0 n}\right)}{2\pi j n} = \frac{\sin \omega_0 n}{\pi n} = \left(\frac{\omega_0}{\pi}\right)\mathrm{sinc}\left(2 f_0 n\right) \tag{5.98}$$

where we have defined the so-called sinc function as

$$\mathrm{sinc}(x) = \frac{\sin \pi x}{\pi x} \tag{5.99}$$

which is a real and even function of x and is plotted in Figure 5.18. Note that the sinc function is *0* when x is a nonzero integer.

Thus, an ideal low-pass filter is noncausal since it has an impulse response with an infinite number of nonzero coefficients.

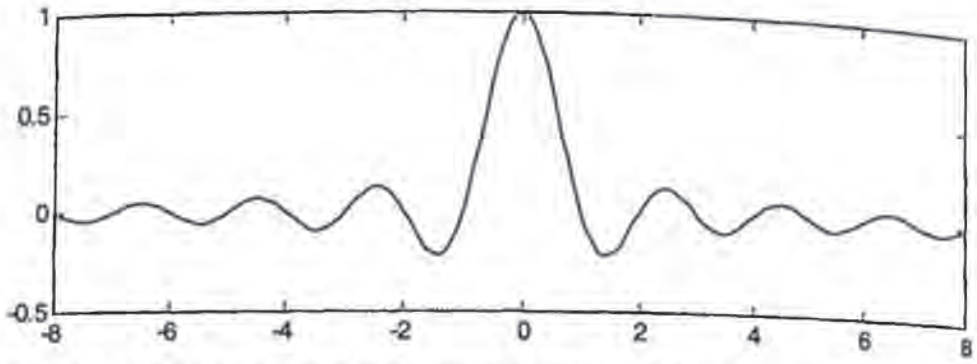


Figure 5.18 A sinc function, which is the impulse response of the ideal low-pass filter with a scale factor.

5.4.2. Window Functions

Window functions are signals that are concentrated in time, often of limited duration. While window functions such as *triangular*, *Kaiser*, *Barlett,* and *prolate spheroidal* occasionally appear in digital speech processing systems, the rectangular, Hanning, and Hamming are the most widely used. Window functions are also concentrated in low frequencies. These window functions are useful in digital filter design and all throughout Chapter 6.

5.4.2.1. The Rectangular Window

The *rectangular* window is defined as

$$h_\pi[n] = u[n] - u[n-N] \tag{5.100}$$

and we refer to it often in this book. Its z-transform is given by

$$H_\pi(z) = \sum_{n=0}^{N-1} z^{-n} \tag{5.101}$$

which results in a polynomial of order $(N-1)$. Multiplying both sides of Eq. (5.101) by z^{-1}, we obtain

$$z^{-1}H_\pi(z) = \sum_{n=1}^{N} z^{-n} = H_\pi(z) - 1 + z^{-N} \tag{5.102}$$

and therefore the sum of the terms of a geometric series can also be expressed as

$$H_\pi(z) = \frac{1-z^{-N}}{1-z^{-1}} \tag{5.103}$$

Although $z=1$ appears to be a pole in Eq. (5.103), it actually isn't because it is canceled by a zero at $z=1$. Since $h_\pi[n]$ has finite length, Eq. (5.25) must be satisfied for $z \neq 0$, so the region of convergence is everywhere but at $z=0$. Moreover, all finite-length sequences have a region of convergence that is the complete z-plane except for possibly $z=0$.

The Fourier transform of the rectangular window is, using Eq. (5.103):

$$H_\pi(e^{j\omega}) = \frac{1-e^{-j\omega N}}{1-e^{-j\omega}} = \frac{\left(e^{j\omega N/2} - e^{-j\omega N/2}\right)e^{-j\omega N/2}}{\left(e^{j\omega/2} - e^{-j\omega/2}\right)e^{-j\omega/2}}$$
$$= \frac{\sin \omega N/2}{\sin \omega/2} e^{-j\omega(N-1)/2} = A(\omega)e^{-j\omega(N-1)/2} \tag{5.104}$$

where $A(\omega)$ is real and even. The function $A(\omega)$, plotted in Figure 5.19 in dB,[8] is 0 for $\omega_k = 2\pi k/N$ with $k \neq \{0, \pm N, \pm 2N, \ldots\}$, and is the discrete-time equivalent of the sinc function.

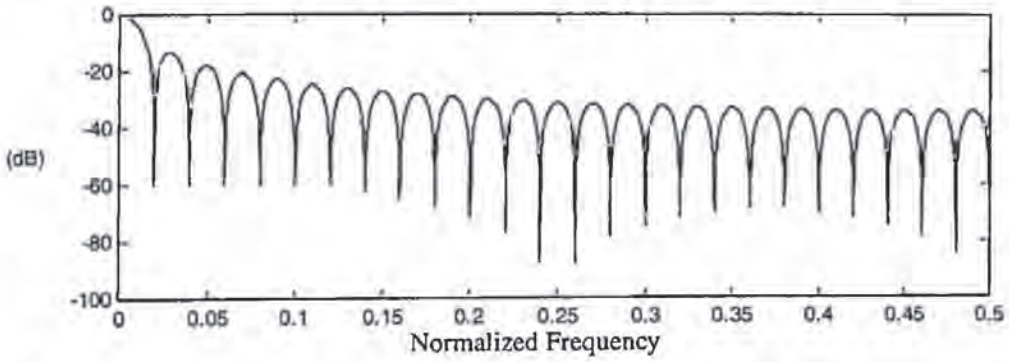


Figure 5.19 Frequency response (magnitude in dB) of the rectangular window with $N = 50$, which is a digital sinc function.

5.4.2.2. The Generalized Hamming Window

The *generalized Hamming window* is defined as

$$h_h[n] = \begin{cases} (1-\alpha) - \alpha\cos(2\pi n/N) & 0 \le n < N \\ 0 & otherwise \end{cases} \tag{5.105}$$

[8] An energy value E is expressed in decibels (dB) as $\overline{E} = 10\log_{10} E$. If the energy value is $2E$, it is therefore 3dB higher. Logarithmic measurements like dB are useful because they correlate well with how the human auditory system perceives volume.

and can be expressed in terms of the rectangular window in Eq. (5.100) as

$$h_h[n] = (1-\alpha)h_\pi[n] - \alpha h_\pi[n]\cos(2\pi n/N) \tag{5.106}$$

whose transform is

$$H_h(e^{j\omega}) = (1-\alpha)H_\pi(e^{j\omega}) - (\alpha/2)H_\pi(e^{j(\omega-2\pi/N)}) - (\alpha/2)H_\pi(e^{j(\omega+2\pi/N)}) \tag{5.107}$$

after using the modulation property in Table 5.5. When $\alpha = 0.5$ the window is known as the *Hanning* window, whereas for $\alpha = 0.46$ it is the *Hamming* window. Hanning and Hamming windows and their magnitude frequency responses are plotted in Figure 5.20.

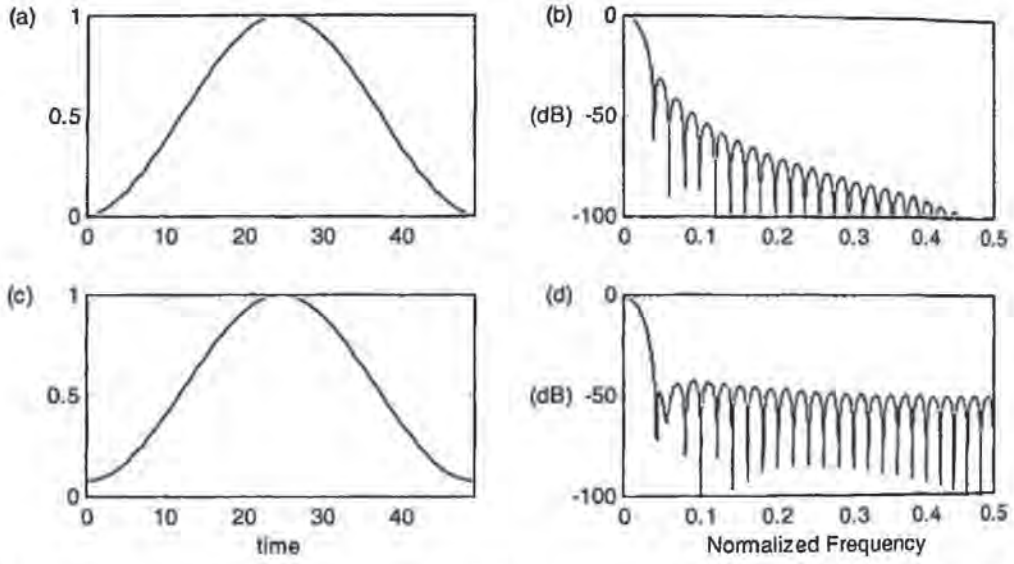


Figure 5.20 (a) Hanning window and (b) the magnitude of its frequency response in dB; (c) Hamming window and (d) the magnitude of its frequency response in dB for $N = 50$.

The main lobe of both Hamming and Hanning is twice as wide as that of the rectangular window, but the attenuation is much greater than that of the rectangular window. The secondary lobe of the Hanning window is 31 dB below the main lobe, whereas for the Hamming window it is 44 dB below. On the other hand, the attenuation of the Hanning window decays with frequency quite rapidly, which is not the case for the Hamming window, whose attenuation stays approximately constant for all frequencies.

5.4.3. FIR Filters

From a practical point of view, it is useful to consider LTI filters whose impulse responses have a limited number of nonzero coefficients:

$$h[n]=\begin{cases} b_n & 0\le n\le M \\ 0 & otherwise \end{cases} \tag{5.108}$$

These types of LTI filters are called *finite-impulse response* (FIR) filters. The input/output relationship in this case is

$$y[n]=\sum_{r=0}^{M} b_r x[n-r] \tag{5.109}$$

The z-transform of $x[n-r]$ is

$$\sum_{n=-\infty}^{\infty} x[n-r]z^{-n}=\sum_{n=-\infty}^{\infty} x[n]z^{-(n+r)}=z^{-r}X(z) \tag{5.110}$$

Therefore, given that the z-transform is linear, $H(z)$ is

$$H(z)=\frac{Y(z)}{X(z)}=\sum_{r=0}^{M} b_r z^{-r}=Az^{-L}\prod_{r=1}^{M-L}\left(1-c_r z^{-1}\right) \tag{5.111}$$

whose region of convergence is the whole z-plane except for possibly $z=0$. Since $\sum_{r=0}^{M}|b_r|$ is finite, FIR systems are always stable, which makes them very attractive. Several special types of FIR filters will be analyzed below: linear-phase, first-order and low-pass FIR filters.

5.4.3.1. Linear-Phase FIR Filters

Linear-phase filters are important because, other than a delay, the phase of the signal is unchanged. Only the magnitude is affected. Therefore, the temporal properties of the input signal are preserved. In this section we show that linear-phase FIR filters can be built if the filter exhibits symmetry.

Let's explore the particular case of $h[n]$ real, $M=2L$, an even number, and $h[n]=h[M-n]$ (called a *Type-I* filter). In this case

$$\begin{aligned} H(e^{j\omega}) &= \sum_{n=0}^{M} h[n]e^{-j\omega n}=h[L]e^{-j\omega L}+\sum_{n=0}^{L-1}\left(h[n]e^{-j\omega n}+h[M-n]e^{-j\omega(2L-n)}\right) \\ &= h[L]e^{-j\omega L}+\sum_{n=0}^{L-1} h[n]\left(e^{-j\omega(n-L)}+e^{j\omega(n-L)}\right)e^{-j\omega L} \\ &= \left(h[L]+\sum_{n=1}^{L} 2h[n+L]\cos(\omega n)\right)e^{-j\omega L}=A(\omega)e^{-j\omega L} \end{aligned} \tag{5.112}$$

where $A(\omega)$ is a real and even function of ω, since the cosine is an even function, and $A(\omega)$ is a linear combination of cosines. Furthermore, we see that the phase $\arg\left\{H(e^{j\omega})\right\}=L\omega$, which is a linear function of ω, and therefore $h[n]$ is called a *linear-*

phase system. It can be shown that if $h[n]=-h[M-n]$, we also get a linear phase system but $A(\omega)$ this time is a pure imaginary and odd function (Type III filter). It is left to the reader to show that in the case of M being odd the system is still linear phase (Types II and IV filters). Moreover, $h[n]$ doesn't have to be real and:

$$h[n]=\pm h^*[M-n] \tag{5.113}$$

is a sufficient condition for $h[n]$ to be linear phase.

5.4.3.2. First-Order FIR Filters

A special case of FIR filters is the first-order filter:

$$y[n]=x[n]+\alpha x[n-1] \tag{5.114}$$

for real values of α, which, unless $\alpha=1$, is not linear phase. Its z-transform is

$$H(z)=1+\alpha z^{-1} \tag{5.115}$$

It is of interest to analyze the magnitude and phase of its frequency response

$$\begin{aligned} |H(e^{j\omega})|^2 &= |1+\alpha(\cos\omega - j\sin\omega)|^2 \\ &= (1+\alpha\cos\omega)^2+(\alpha\sin\omega)^2 = 1+\alpha^2+2\alpha\cos\omega \end{aligned} \tag{5.116}$$

$$\theta(e^{j\omega})=-\arctan\left(\frac{\alpha\sin\omega}{1+\alpha\cos\omega}\right) \tag{5.117}$$

It is customary to display the magnitude response in decibels (dB):

$$10\log|H(e^{j\omega})|^2=10\log\left[(1+\alpha)^2+2\alpha\cos\omega\right] \tag{5.118}$$

as shown in Figure 5.21 for various values of α.

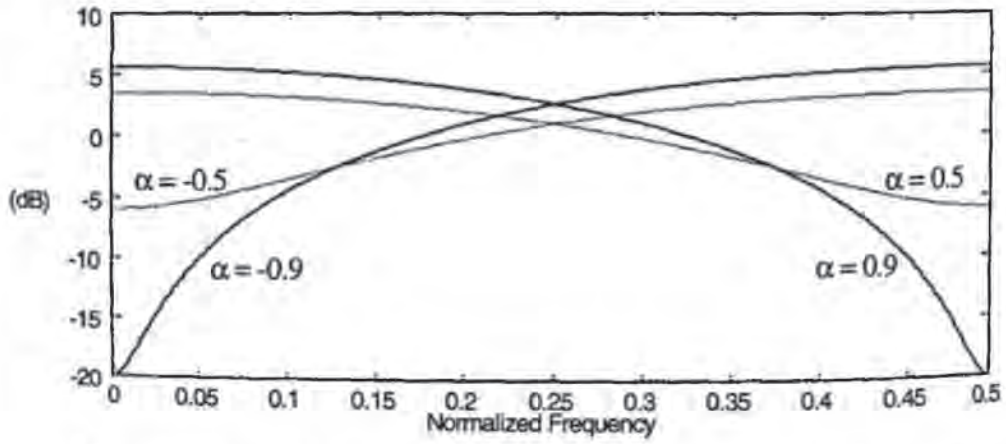


Figure 5.21 Frequency response of the first order FIR filter for various values of α.

We see that for $\alpha > 0$ we have a *low-pass* filter whereas for $\alpha < 0$ it is a *high-pass* filter, also called a *pre-emphasis* filter, since it emphasizes the high frequencies. In general, filters that boost the high frequencies and attenuate the low frequencies are called *high-pass* filters, and filters that emphasize the low frequencies and de-emphasize the high frequencies are called *low-pass* filters. The parameter α controls the slope of the curve.

5.4.3.3. Window Design FIR Lowpass Filters

The ideal lowpass filter lets all frequencies below ω_0 go through and eliminates all energy from frequencies above that range. As we described in Section 5.4.1, the ideal lowpass filter has an infinite impulse response, which poses difficulties for implementation in a practical system, as it requires an infinite number of multiplies and adds.

Since we know that the sinc function decays over time, it is reasonable to assume that a truncated sinc function that keeps a large enough number of samples N could be a good approximation to the ideal low-pass filter. Figure 5.22 shows the magnitude of the frequency response of such a truncated sinc function for different values of N. While the approximation gets better for larger N, the overshoot near ω_0 doesn't go away and in fact stays at about 9% of the discontinuity even for large N. This is known as the *Gibbs phenomenon*, since Yale professor Josiah Gibbs first noticed it in 1899.

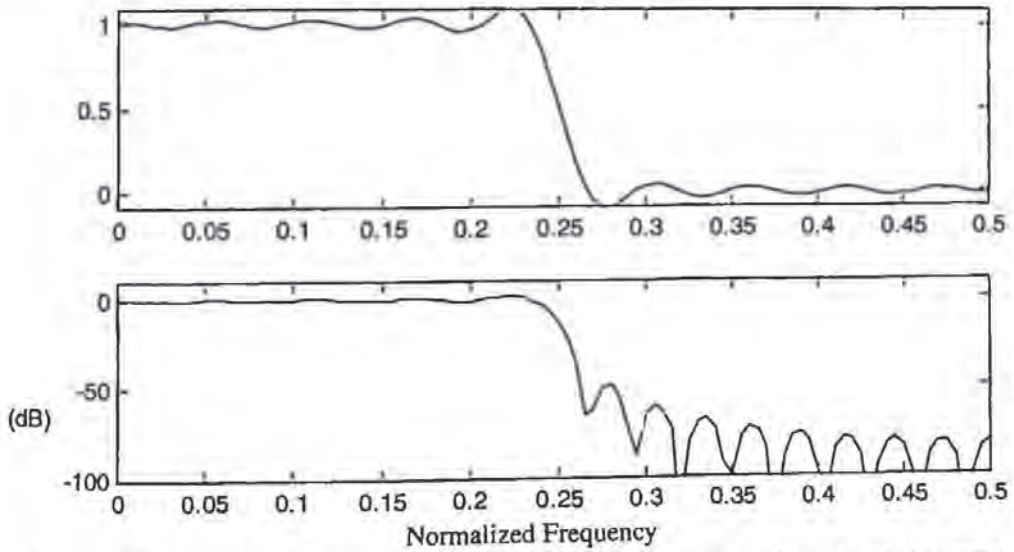


Figure 5.22 Magnitude frequency response of the truncated sinc signal (N=200) for $\omega_0 = \pi/4$. It is an approximation to the ideal low-pass filter, though we see that overshoots are present near the transition. The first graph is linear magnitude and the second is in dB.

In computing the truncated sinc function, we have implicitly multiplied the ideal low-pass filter, the sinc function, by a rectangular window. In the so-called *window design* filter design method, the filter coefficients are obtained by multiplying the ideal sinc function by a

tapering window function, such as the Hamming window. The resulting frequency response is the convolution of the ideal lowpass filter function with the transform of the window (shown in Figure 5.23), and it does not exhibit the overshoots in Figure 5.22, at the expense of a slower transition.

5.4.3.4. Parks McClellan Algorithm

While the window design method is simple, it is hard to predict what the final response will be. Other methods have been proposed whose coefficients are obtained to satisfy some constraints. If our constraints are a maximum ripple of δ_p in the *passband* ($0 \le \omega < \omega_p$), and a minimum attenuation of δ_s in the *stopband* ($\omega_s \le \omega < \pi$), the optimal solution is given by the Parks McClellan algorithm [14].

The transformation

$$x = \cos\omega \tag{5.119}$$

maps the interval $0 \le \omega \le \pi$ into $-1 \le x \le 1$. We note that

$$\cos(n\omega) = T_n(\cos\omega) \tag{5.120}$$

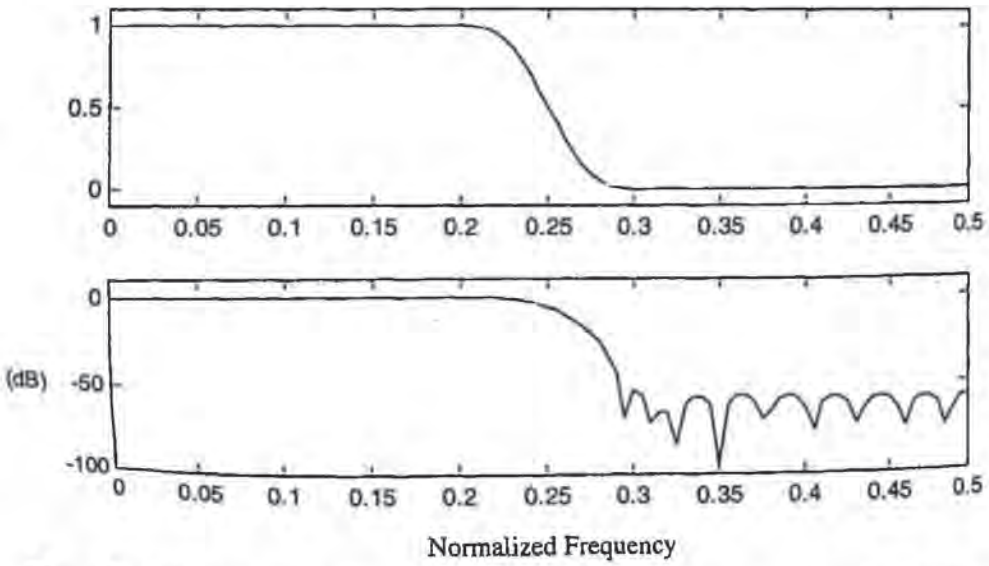


Figure 5.23 Magnitude frequency response of a low-pass filter obtained with the window design method and a Hamming window ($N = 200$). The first graph is linear magnitude and the second is in dB.

where $T_n(x)$ is the n^{th}-order *Chebychev* polynomial. The first two Chebychev polynomials are given by $T_0(x)=1$ and $T_1(x)=x$. If we add the following trigonometric identities

$$\begin{aligned}\cos(n+1)\omega &= \cos n\omega\cos\omega - \sin n\omega\sin\omega \\ \cos(n-1)\omega &= \cos n\omega\cos\omega + \sin n\omega\sin\omega\end{aligned} \tag{5.121}$$

and use Eqs. (5.119) and (5.120), we obtain the following recursion formula:

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x) \quad \text{for} \quad n>1 \tag{5.122}$$

Using Eq. (5.120), the magnitude response of a linear phase Type-I filter in Eq. (5.112) can be expressed as an L^{th}-order polynomial in $\cos\omega$:

$$A(\omega) = \sum_{k=0}^{L} a_k(\cos\omega)^k \tag{5.123}$$

which, using Eq. (5.119) results in a polynomial

$$P(x) = \sum_{k=0}^{L} a_k x^k \tag{5.124}$$

Given that a desired response is $D(x) = D(\cos\omega)$, we define the weighted squared error as

$$E(x) = E(\cos\omega) = W(\cos\omega)[D(\cos\omega) - P(\cos\omega)] = W(x)[D(x) - P(x)] \tag{5.125}$$

where $W(\cos\omega)$ is the weighting in ω. A necessary and sufficient condition for this weighted squared error to be minimized is to have $P(x)$ alternate between minima and maxima. For the case of a low-pass filter,

$$D(\cos\omega) = \begin{cases} 1 & \cos\omega_p \le \cos\omega \le 1 \\ 0 & -1 \le \cos\omega \le \cos\omega_s \end{cases} \tag{5.126}$$

and the weight in the stopband is several times larger than in the passband.

These constraints and the response of a filter designed with such a method are shown in Figure 5.24. We can thus obtain a similar transfer function with fewer coefficients using this method.

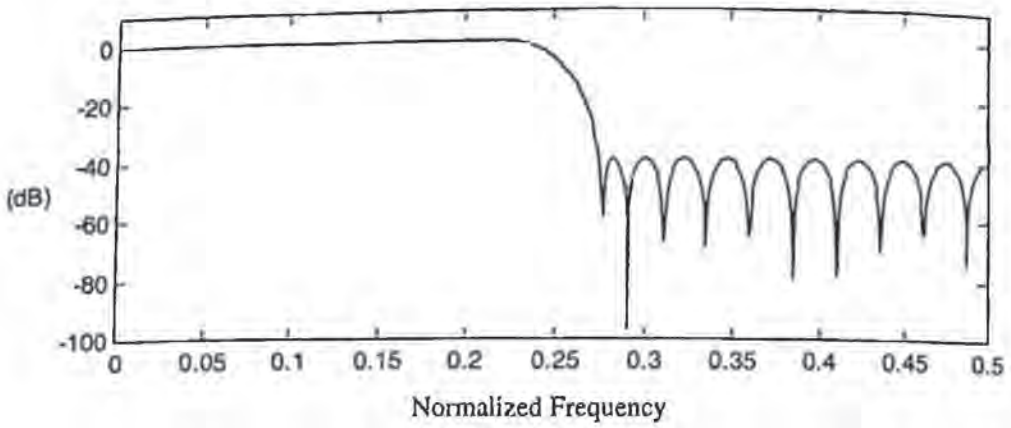


Figure 5.24 Magnitude frequency response of a length-19 lowpass filter designed with the Parks McClellan algorithm.

5.4.4. IIR Filters

Other useful filters are a function of past values of the input and also the output

$$y[n]=\sum_{k=1}^{N} a_k y[n-k]+\sum_{r=0}^{M} b_r x[n-r] \tag{5.127}$$

whose z-transform is given by

$$H(z)=\frac{Y(z)}{X(z)}=\frac{\sum_{r=0}^{M} b_r z^{-r}}{1-\sum_{k=1}^{N} a_k z^{-k}} \tag{5.128}$$

which in turn can be expressed as a function of the roots of the numerator c_r (called *zeros*), and denominator d_k (called *poles*) as

$$H(z)=\frac{Az^{-L}\prod_{r=1}^{M-L}\left(1-c_r z^{-1}\right)}{\prod_{k=1}^{N}\left(1-d_k z^{-1}\right)} \tag{5.129}$$

It is not obvious what the impulse response of such a system is by looking at either Eq. (5.128) or Eq. (5.129). To do that, we can compute the inverse z-transform of Eq. (5.129). If $M < N$ in Eq. (5.129), $H(z)$ can be expanded into partial fractions (see Section 5.2.3.3) as

$$H(z)=\sum_{k=1}^{N}\frac{A_k}{1-d_k z^{-1}} \tag{5.130}$$

and if $M \geq N$

$$H(z) = \sum_{k=1}^{N} \frac{A_k}{1 - d_k z^{-1}} + \sum_{k=0}^{M-N} B_k z^{-k} \tag{5.131}$$

which we can now compute, since we know that the inverse z-transform of $H_k(z) = A_k/(1 - d_k z^{-1})$ is

$$h_k[n] = \begin{cases} A_k d_k^n u[n] & |d_k| < 1 \\ -A_k d_k^n u[-n-1] & |d_k| > 1 \end{cases} \tag{5.132}$$

so that the convergence region includes the unit circle and therefore $h_k[n]$ is stable. Therefore, a necessary and sufficient condition for $H(z)$ to be stable *and* causal simultaneously is that all its poles be inside the unit circle: i.e., $|d_k| < 1$ for all k, so that its impulse response is given by

$$h[n] = B_n + \sum_{k=1}^{N} A_k d_k^n u[n] \tag{5.133}$$

which has an infinite impulse response, and hence its name.

Since IIR systems may have poles outside the unit circle, they are not guaranteed to be stable and causal like their FIR counterparts. This makes IIR filter design more difficult, since only stable and causal filters can be implemented in practice. Moreover, unlike FIR filters, IIR filters do not have linear phase. Despite these difficulties, IIR filters are popular because they are more efficient than FIR filters in realizing steeper roll-offs with fewer coefficients. In addition, as shown in Chapter 6, they represent many physical systems.

5.4.4.1. First-Order IIR Filters

An important type of IIR filter is the first-order filter of the form

$$y[n] = Ax[n] + \alpha y[n-1] \tag{5.134}$$

for α real. Its transfer function is given by

$$H(z) = \frac{A}{1 - \alpha z^{-1}} \tag{5.135}$$

This system has one pole and no zeros. As we saw in our discussion of z-transforms in Section 5.2.3, a necessary condition for this system to be both stable and causal is that $|\alpha| < 1$. Since for the low-pass filter case $0 < \alpha < 1$, it is convenient to define $\alpha = e^{-b}$ where $b > 0$. In addition, the corresponding impulse response is infinite:

$$h[n] = A\alpha^n u[n] \tag{5.136}$$

whose Fourier transform is

$$H(e^{j\omega}) = \frac{A}{1-\alpha e^{-j\omega}} = \frac{A}{1-e^{-b-j\omega}} \tag{5.137}$$

and magnitude square is given by

$$|H(e^{j\omega})|^2 = \frac{|A|^2}{1+\alpha^2-2\alpha\cos\omega} \tag{5.138}$$

which is shown in Figure 5.25 for $\alpha > 0$, which corresponds to a low-pass filter.

The bandwidth of a low-pass filter is defined as the point where its magnitude square is half of its maximum value. Using the first-order Taylor approximation of the exponential function, the following approximation can be used when $b \to 0$:

$$|H(e^{j0})|^2 = \frac{A^2}{|1-e^{-b}|^2} \approx \frac{A^2}{b^2} \tag{5.139}$$

If the bandwidth ω_b is also small, we can similarly approximate

$$|H(e^{j\omega_b})|^2 = \frac{A^2}{|1-e^{-b-j\omega_b}|^2} \approx \frac{A^2}{|b+j\omega_b|^2} = \frac{A^2}{\left(b^2+\omega_b^2\right)} \tag{5.140}$$

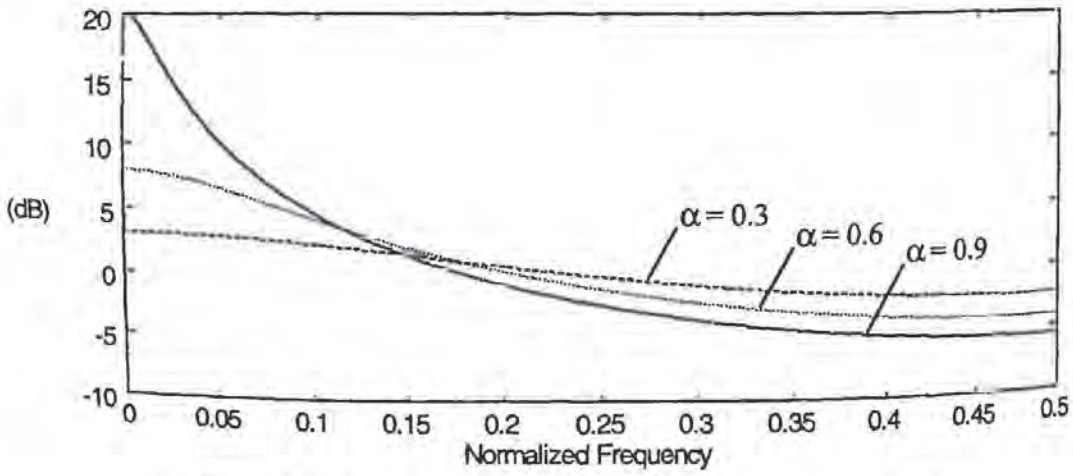


Figure 5.25 Magnitude frequency response of the first-order IIR filter.

so that for $\omega_b = b$ we have $|H(e^{jb})|^2 \approx 0.5\,|H(e^{j0})|^2$. In other words, the bandwidth of this filter equals b, for small values of b. The relative error in this approximation[9] is smaller than 2% for $b < 0.5$, which corresponds to $0.6 < \alpha < 1$. The relationship with the unnormalized bandwidth B is

$$\alpha = e^{-2\pi B/F_s} \tag{5.141}$$

For $\alpha < 0$ it behaves as a high-pass filter, and a similar discussion can be carried out.

5.4.4.2. Second-Order IIR Filters

An important type of IIR filters is the set of second-order filters of the form

$$y[n] = Ax[n] + a_1 y[n-1] + a_2 y[n-2] \tag{5.142}$$

whose transfer function is given by

$$H(z) = \frac{A}{1 - a_1 z^{-1} - a_2 z^{-2}} \tag{5.143}$$

This system has two poles and no zeros. A special case is when the coefficients A, a_1, and a_2 are real. In this case the two poles are given by

$$z = \frac{a_1 \pm \sqrt{a_1^2 + 4a_2}}{2} \tag{5.144}$$

which for the case of $a_1^2 + 4a_2 > 0$ yields two real roots, and is a degenerate case of two first-order systems. The more interesting case is when $a_1^2 + 4a_2 < 0$. In this case we see that the two roots are complex conjugates of each other, which can be expressed in their magnitude and phase notation as

$$z = e^{-\sigma \pm j\omega_0} \tag{5.145}$$

As we mentioned before, $\sigma > 0$ is a necessary and sufficient condition for the poles to be inside the unit circle and thus for the system to be stable. With those values, the z-transform is given by

$$H(z) = \frac{A}{(1 - e^{-\sigma + j\omega_0} z^{-1})(1 - e^{-\sigma - j\omega_0} z^{-1})} = \frac{A}{1 - 2e^{-\sigma}\cos(\omega_0) z^{-1} + e^{-2\sigma} z^{-2}} \tag{5.146}$$

In Figure 5.26 we show the magnitude of its Fourier transform for a value of σ and ω_0. We see that the response is centered around ω_0 and is more concentrated for smaller values of

[9] The exact value is $\omega_b = \arccos\left[2 - \cosh b\right]$, where $\cosh b = \left(e^b + e^{-b}\right)/2$ is the hyperbolic cosine.

σ. This is a type of *bandpass* filter, since it favors frequencies in a band around ω_0. It is left to the reader as an exercise to show that the bandwidth[10] is approximately 2σ. The smaller the ratio σ/ω_0, the sharper the resonance. The filter coefficients can be expressed as a function of the unnormalized bandwidth B and resonant frequency F and the sampling frequency F_s (all expressed in Hz) as

$$a_1 = 2e^{-\pi B/F_s}\cos(2\pi F/F_s) \tag{5.147}$$

$$a_2 = -e^{-2\pi B/F_s} \tag{5.148}$$

These types of systems are also known as second-order resonators and will be of great use for speech synthesis (Chapter 16), particularly for formant synthesis.

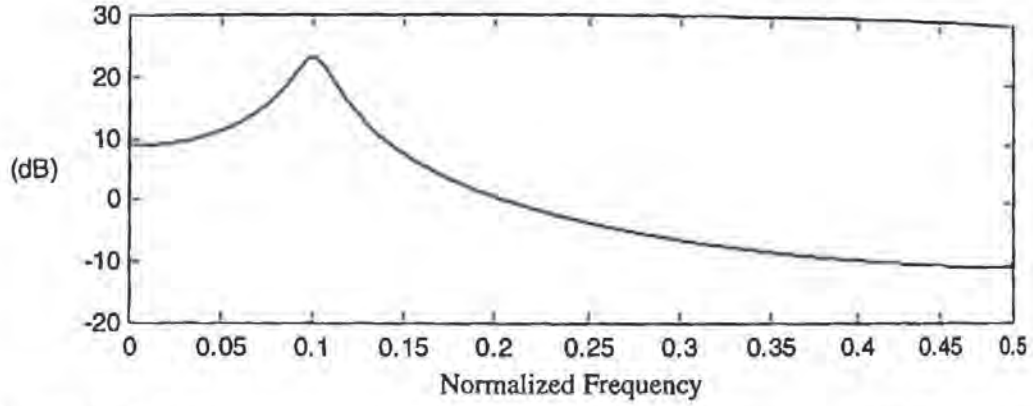


Figure 5.26 Frequency response of the second-order IIR filter for center frequency of $F = 0.1F_s$ and bandwidth $B = 0.01F_s$.

5.5. DIGITAL PROCESSING OF ANALOG SIGNALS

To use digital signal processing methods, it is necessary to convert the speech signal $x(t)$, which is analog, to a digital signal $x[n]$, which is formed by periodically sampling the analog signal $x(t)$ at intervals equally spaced T seconds apart:

$$x[n] = x(nT) \tag{5.149}$$

where T is defined as the sampling period, and its inverse $F_s = 1/T$ as the sampling frequency. In the speech applications considered in this book, F_s can range from 8000 Hz for telephone applications to 44,100 Hz for high-fidelity audio applications. This section explains the sampling theorem, which essentially says that the analog signal $x(t)$ can be

[10] The bandwidth of a bandpass filter is the region between half maximum magnitude squared values.

uniquely recovered given its digital signal $x[n]$ if the analog signal $x(t)$ has no energy for frequencies above the *Nyquist* frequency $F_s/2$.

We not only prove the sampling theorem, but also provide great insight into the analog-digital conversion, which is used in Chapter 7.

5.5.1. Fourier Transform of Analog Signals

The Fourier transform of an analog signal $x(t)$ is defined as

$$X(\Omega) = \int_{-\infty}^{\infty} x(t)e^{-j\Omega t}dt \tag{5.150}$$

with its inverse transform being

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(\Omega)e^{j\Omega t}d\Omega \tag{5.151}$$

They are transform pairs. You can prove similar relations for the Fourier transform of analog signals as for their digital signals counterpart.

5.5.2. The Sampling Theorem

Let's define $x_p(t)$

$$x_p(t) = x(t)p(t) \tag{5.152}$$

as a sampled version of $x(t)$, where

$$p(t) = \sum_{n=-\infty}^{\infty} \delta(t-nT) \tag{5.153}$$

where $\delta(t)$ is the Dirac delta defined in Section 5.3.2.1. Therefore, $x_p(t)$ can also be expressed as

$$x_p(t) = \sum_{n=-\infty}^{\infty} x(t)\delta(t-nT) = \sum_{n=-\infty}^{\infty} x(nT)\delta(t-nT) = \sum_{n=-\infty}^{\infty} x[n]\delta(t-nT) \tag{5.154}$$

after using Eq. (5.149). In other words, $x_p(t)$ can be uniquely specified given the digital signal $x[n]$.

Using the modulation property of Fourier transforms of analog signals, we obtain

$$X_p(\Omega) = \frac{1}{2\pi} X(\Omega) * P(\Omega) \tag{5.155}$$

Following a derivation similar to that in Section 5.3.2.2, one can show that the transform of the impulse train $p(t)$ is given by

$$P(\Omega) = \frac{2\pi}{T} \sum_{k=-\infty}^{\infty} \delta(\Omega - k\Omega_s) \tag{5.156}$$

where $\Omega_s = 2\pi F_s$ and $F_s = 1/T$, so that

$$X_p(\Omega) = \frac{1}{T} \sum_{k=-\infty}^{\infty} X(\Omega - k\Omega_s) \tag{5.157}$$

From Figure 5.27 it can be seen that if

$$X(\Omega) = 0 \text{ for } |\Omega| > \Omega_s / 2 \tag{5.158}$$

then $X(\Omega)$ can be completely recovered from $X_p(\Omega)$ as follows

$$X(\Omega) = R_{\Omega_s}(\Omega) X_p(\Omega) \tag{5.159}$$

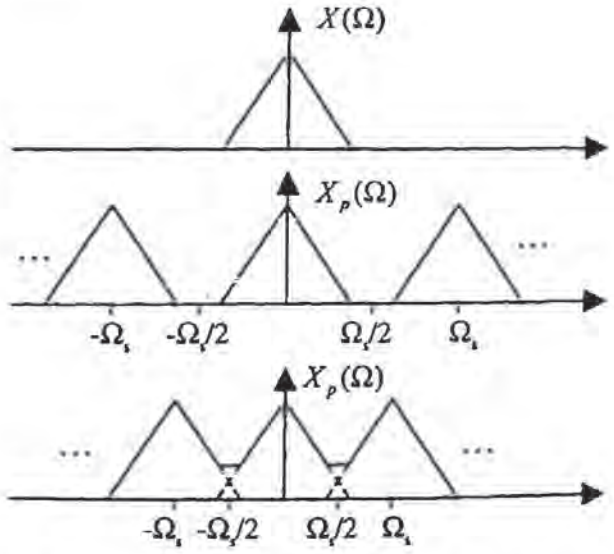


Figure 5.27 $X(\Omega)$, $X_p(\Omega)$ for the case of no aliasing and aliasing.

where

$$R_{\Omega_s}(\Omega) = \begin{cases} T & |\Omega| < \Omega_s/2 \\ 0 & otherwise \end{cases} \tag{5.160}$$

is an ideal lowpass filter. We can also see that if Eq. (5.158) is not met, then *aliasing* will take place and $X(\Omega)$ can no longer be recovered from $X_p(\Omega)$. Since, in general, we cannot be certain that Eq. (5.158) is true, the analog signal is low-pass filtered with an ideal filter given by Eq. (5.160), which is called anti-aliasing filter, prior to sampling. Limiting the bandwidth of our analog signal is the price we have to pay to be able to manipulate it digitally.

The inverse Fourier transform of Eq. (5.160), computed through Eq. (5.151), is a sinc function

$$r_T(t) = \text{sinc}(t/T) = \frac{\sin(\pi t/T)}{\pi t/T} \tag{5.161}$$

so that using the convolution property in Eq. (5.159) we obtain

$$x(t) = r_T(t) * x_p(t) = r_T(t) * \sum_{k=-\infty}^{\infty} x[k]\delta(t-kT) = \sum_{k=-\infty}^{\infty} x[k] r_T(t-kT) \tag{5.162}$$

The *sampling theorem* states that we can recover the continuous time signal $x(t)$ just from its samples $x[n]$ using Eqs. (5.161) and (5.162). The angular frequency $\Omega_s = 2\pi F_s$ is expressed in terms of the sampling frequency F_S. $T = 1/F_s$ is the sampling period, and $F_s/2$ the *Nyquist* frequency. Equation (5.162) is referred to as *bandlimited interpolation* because $x(t)$ is reconstructed by interpolating $x[n]$ with sinc functions that are bandlimited.

Now let's see the relationship between $X_p(\Omega)$ and $X(e^{j\omega})$, the Fourier transform of the discrete sequence $x[n]$. From Eq. (5.154) we have

$$X_p(\Omega) = \sum_{n=-\infty}^{\infty} x[n] e^{-j\Omega nT} \tag{5.163}$$

so that the continuous transform $X_p(\Omega)$ equals the discrete Fourier transform $X(e^{j\omega})$ at $\omega = \Omega T$.

5.5.3. Analog-to-Digital Conversion

The process of converting an analog signal $x(t)$ into a digital signal $x[n]$ is called *Analog-to-Digital conversion*, or A/D for short, and the device that does it is called an *Analog-to-Digital Converter*. In Section 5.5.2 we saw that an ideal low-pass anti-aliasing filter was required on the analog signal, which of course is not realizable in practice so that an approximation has to be used. In practice, sharp analog filters can be implemented on the same

chip using switched capacitor filters, which have attenuations above 60 dB in the stop band so that aliasing tends not to be an important issue for speech signals. The passband is not exactly flat, but this again does not have much significance for speech signals (for other signals, such as those used in modems, this issue needs to be studied more carefully).

Although such sharp analog filters are possible, they can be expensive and difficult to implement. One common solution involves the use of a simple analog low-pass filter with a large attenuation at $MF_s/2$, a multiple of the required cutoff frequency. Then *oversampling* is done at the new rate MF_s, followed by a sharper digital filter with a cut-off frequency of $F_s/2$ and downsampling (see Section 5.6). This is equivalent to having used a sharp analog filter, with the advantage of a lower-cost implementation. This method also allows variable sampling rates with minimal increase in cost and complexity. This topic is discussed in more detail in Chapter 7 in the context of sigma-delta modulators.

In addition, the pulses in Eq. (5.59) cannot be zero length in practice, and therefore the sampling theorem does not hold. However, current hardware allows the pulses to be small enough that the analog signal can be approximately recovered. The signal level is then maintained during T seconds, while the conversion to digital is being carried out.

A real A/D converter cannot provide real numbers for $x[n]$, but rather a set of integers typically represented with 16 bits, which gives a range between –32,768 and 32,767. Such conversion is achieved by comparing the analog signal to a number of different signal levels. This means that *quantization noise* has been added to the digital signal. This is typically not a big problem for speech signals if using 16 bits or more since, as is shown in Chapter 7, other noises will mask the quantization noise anyway. Typically, quantization noise becomes an issue only if 12 or fewer bits are used. A more detailed study of the effects of quantization is presented in Chapter 7.

Finally, A/D subsystems are not exactly linear, which adds another source of distortion. This nonlinearity can be caused by, among things, jitter and drift in the pulses and unevenly spaced comparators. For popular A/D subsystems, such as *sigma-delta* A/D, an offset is typically added to $x[n]$, which in practice is not very important, because speech signals do not contain information at $f = 0$, and thus can be safely ignored.

5.5.4. Digital-to-Analog Conversion

The process of converting the digital signal $x[n]$ back into an analog $x(t)$ is called *digital-to-analog conversion*, or D/A for short. The ideal band-limited interpolation requires ideal sinc functions as shown in Eq. (5.162), which are not realizable. To convert the digital signal to analog, a zero-order hold filter

$$h_0(t) = \begin{cases} 1 & 0 < t < T \\ 0 & \text{otherwise} \end{cases} \tag{5.164}$$

is often used, which produces an analog signal as shown in Figure 5.28. The output of such a filter is given by

$$x_0(t) = h_0(t) * \sum_{n=-\infty}^{\infty} x[n]\delta(t-nT) = \sum_{n=-\infty}^{\infty} x[n]h_0(t-nT) \tag{5.165}$$

The Fourier transform of the zero-hold filter in Eq. (5.164) is, using Eq. (5.150),

$$H_0(\Omega) = \frac{2\sin(\Omega T/2)}{\Omega} e^{-j\Omega T/2} \tag{5.166}$$

and, since we need an ideal lowpass filter to achieve the band-limited interpolation of Eq. (5.162), the signal $x_0(t)$ has to be filtered with a reconstruction filter with transfer function

$$H_r(\Omega) = \begin{cases} \dfrac{\Omega T/2}{\sin(\Omega T/2)} e^{j\Omega T/2} & |\Omega| < \pi/T \\ 0 & |\Omega| > \pi/T \end{cases} \tag{5.167}$$

In practice, the phase compensation is ignored, as it amounts to a delay of $T/2$ seconds. Its magnitude response can be seen in Figure 5.29. In practice, such an analog filter is not realizable and an approximation is made. Since the zero-order hold filter is already low-pass, the reconstruction filter doesn't need to be that sharp.

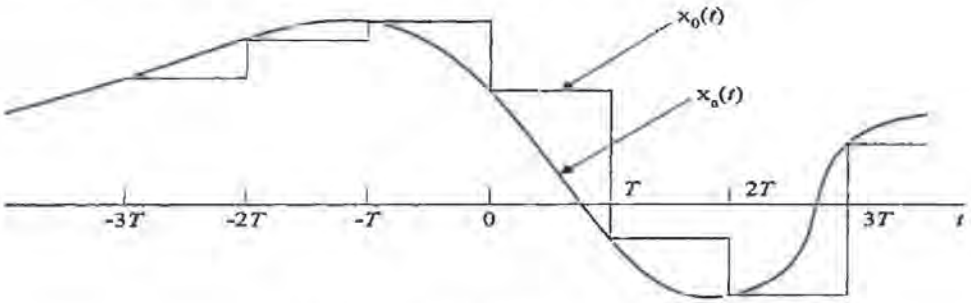


Figure 5.28 Output of a zero-order hold filter.

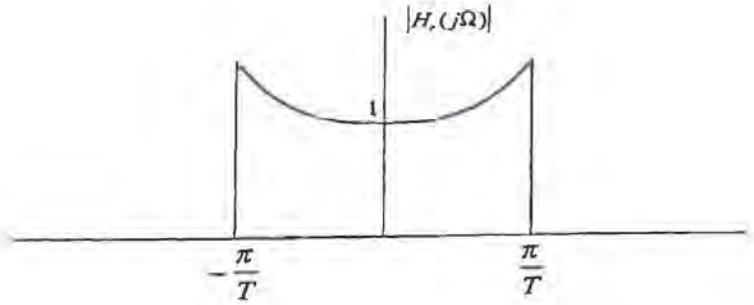


Figure 5.29 Magnitude frequency response of the reconstruction filter used in digital-to-analog converters after a zero-hold filter.

In the above discussion we note that practical A/D and D/A systems introduce distortions, which causes us to wonder whether it is a good idea to go through this process just to manipulate digital signals. It turns out that for most speech processing algorithms described in Chapter 6, the advantages of operating with digital signals outweigh the disadvantage of the distortions described above. Moreover, commercial A/D and D/A systems are such that the errors and distortions can be arbitrarily small. The fact that music in digital format (as in compact discs) has won out over analog format (cassettes) shows that this is indeed the case. Nonetheless, it is important to be aware of the above limitations when designing a system.

5.6. MULTIRATE SIGNAL PROCESSING

The term *Multirate Signal Processing* refers to processing of signals sampled at different rates. A particularly important problem is that of sampling-rate conversion. It is often the case that we have a digital signal $x[n]$ sampled at a sampling rate F_s, and we want to obtain an equivalent signal $y[n]$ but at a different sampling rate F_s'. This often occurs in A/D systems that oversample in order to use smaller quantizers, such as a delta or sigma delta-quantizer (see Chapter 7), and a simpler analog filter, and then have to downsample the signal. Other examples include mixing signals of different sampling rates and downsampling to reduce computation (many signal processing algorithms have a computational complexity proportional to the sampling rate or its square).

A simple solution is to convert the digital signal $x[n]$ into an analog signal $x(t)$ with a D/A system running at F_s and then convert it back to digital with an A/D system running at F_s'. An interesting problem is whether this could be done in the digital domain directly, and the techniques to do so belong to the general class of multi-rate processing.

5.6.1. Decimation

If we want to reduce the sampling rate by a factor of M, i.e., $T' = MT$, we take every M samples. In order to avoid aliasing, we need to lowpass filter the signal to bandlimit it to frequencies $1/T'$. This is shown in Figure 5.30, where the arrow pointing down indicates the decimation.

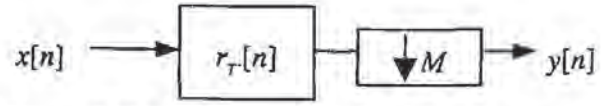


Figure 5.30 Block diagram of the decimation process.

Since the output is not desired at all instants n, but only every M samples, the computation can be reduced by a factor of M over the case where lowpass filtering is done first and *decimation* later. To do this we express the analog signal $x_l(t)$ at the output of the lowpass filter as

$$x_I(t) = \sum_{k=-\infty}^{\infty} x[k] r_{T'}(t - kT) \tag{5.168}$$

and then look at the value $t' = nT'$. The decimated signal $y[n]$ is then given by

$$y[n] = x_I(nT') = \sum_{k=-\infty}^{\infty} x[k] r_{T'}(nT' - kT) = \sum_{k=-\infty}^{\infty} x[k] \text{sinc}\left(\frac{Mn - k}{M}\right) \tag{5.169}$$

which can be expressed as

$$y[n] = \sum_{k=-\infty}^{\infty} x[k] h[Mn - k] \tag{5.170}$$

where

$$h[n] = \text{sinc}(n/M) \tag{5.171}$$

In practice, the ideal lowpass filter $h[n]$ is approximated by an FIR filter with a cutoff frequency of $1/(2M)$.

5.6.2. Interpolation

If we want to increase the sampling rate by a factor of N, so that $T' = T/N$, we do not have any aliasing and no further filtering is necessary. In fact we already know one out of every N output samples

$$y[Nn] = x[n] \tag{5.172}$$

and we just need to compute the $(N-1)$ samples in-between. Since we know that $x[n]$ is a bandlimited signal, we can use the sampling theorem in Eq. (5.162) to reconstruct the analog signal as

$$x_I(t) = \sum_{k=-\infty}^{\infty} x[k] r_T(t - kT) \tag{5.173}$$

and thus the interpolated signal $y[n]$ as

$$y[n] = x(nT') = \sum_{k=-\infty}^{\infty} x[k] r_T(nT' - kT) = \sum_{k=-\infty}^{\infty} x[k] \text{sinc}\left(\frac{n - kN}{N}\right) \tag{5.174}$$

Now let's define

$$x'[k'] = \begin{cases} x[Nk] & k' = Nk \\ 0 & otherwise \end{cases} \tag{5.175}$$

which, inserted into Eq. (5.174), gives

$$y[n] = \sum_{k'=-\infty}^{\infty} x'[k']\mathrm{sinc}\left((n-k')/N\right) \tag{5.176}$$

This can be seen in Figure 5.31, where the block with the arrow pointing up implements Eq. (5.175).

Equation (5.174) can be expressed as

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-kN] \tag{5.177}$$

where we have defined

$$h[n] = \mathrm{sinc}(n/N) \tag{5.178}$$

Again, in practice, the ideal low-pass filter $h[n]$ is approximated by an FIR filter with a cutoff frequency of $1/(2N)$.

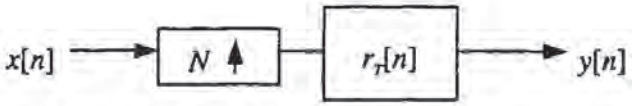


Figure 5.31 Block diagram of the interpolation process.

5.6.3. Resampling

To resample the signal so that $T' = TM/N$, or $F_s' = F_s\left(N/M\right)$, we can first upsample the signal by N and then downsample it by M. However, there is a more efficient way. Proceeding similarly to decimation and interpolation, one can show the output is given by

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[nM - kN] \tag{5.179}$$

where

$$h[n] = \mathrm{sinc}\left(\frac{n}{\max(N,M)}\right) \tag{5.180}$$

for the ideal case. In practice, $h[n]$ is an FIR filter with a cutoff frequency of $1/(2\max(N,M))$. We can see that Eq. (5.179) is a superset of Eqs. (5.170) and (5.177).

5.7. FILTERBANKS

A filterbank is a collection of filters that span the whole frequency spectrum. In this section we describe the fundamentals of filterbanks, which are used in speech and audio coding, echo cancellation, and other applications. We first start with a filterbank with two equal bands, then explain multi-resolution filterbanks, and present the FFT as a filterbank. Finally we introduce the concept of lapped transforms and wavelets.

5.7.1. Two-Band Conjugate Quadrature Filters

A two-band filterbank is shown in Figure 5.32 , where the filters $f_0[n]$ and $g_0[n]$ are low-pass filters, and the filters $f_1[n]$ and $g_1[n]$ are high-pass filters, as shown in Figure 5.33. Since the output of $f_0[n]$ has a bandwidth half of that of $x[n]$, we can sample it at half the rate of $x[n]$. We do that by decimation (throwing out every other sample), as shown in Figure 5.32. The output of such a filter plus decimation is $x_0[m]$. Similar results can be shown for $f_1[n]$ and $x_1[n]$.

For reconstruction, we upsample $x_0[m]$, by inserting a 0 between every sample. Then we low-pass filter it with filter $g_0[n]$ to complete the interpolation, as we saw in Section 5.6. A similar process can be done with the high pass filters $f_1[n]$ and $g_1[n]$. Adding the two bands produces $\tilde{x}[n]$, which is identical to $x[n]$ if the filters are ideal.

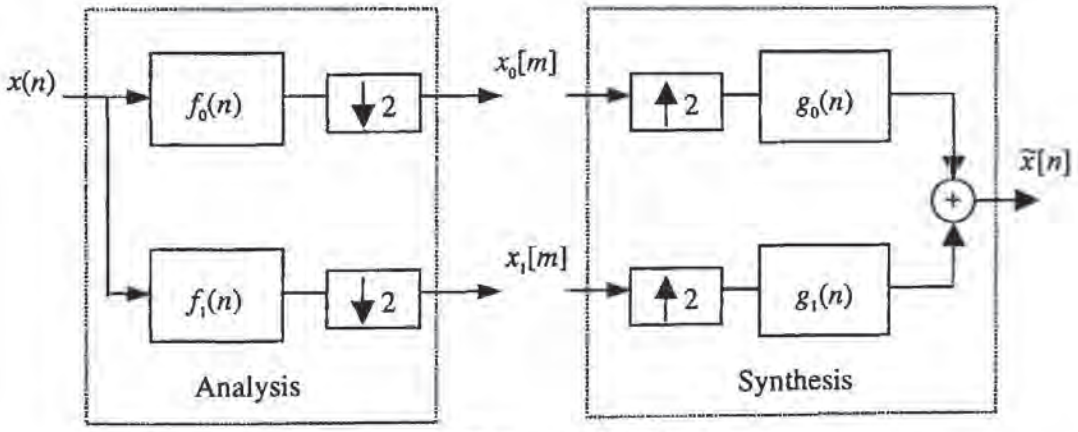


Figure 5.32 Two-band filterbank.

In practice, however, ideal filters such as those in Figure 5.33 are not achievable, so we would like to know if it is possible to build a filterbank that has perfect reconstruction with FIR filters. The answer is affirmative, and in this section we describe conjugate quadrature filters, which are the basis for the solutions.

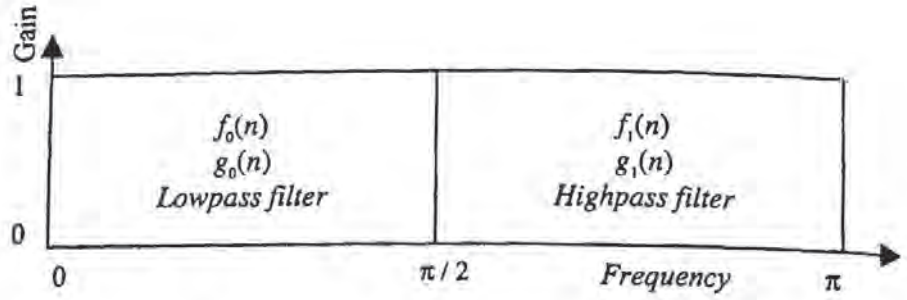


Figure 5.33 Ideal frequency responses of analysis and synthesis filters for the two-band filter-bank.

To investigate this, let's analyze the cascade of a downsampler and an upsampler (Figure 5.34). The output $y[n]$ is a signal whose odd samples are zero and whose even samples are the same as those of the input signal $x[n]$.

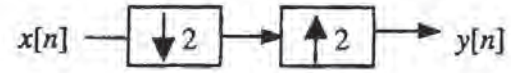


Figure 5.34 Cascade of a downsampler and an upsampler.

The z-transform of the output is given by

$$Y(z) = \sum_{n=-\infty}^{\infty} x[n]z^{-n} = \frac{1}{2}\sum_{n=-\infty}^{\infty} x[n]z^{-n} + \frac{1}{2}\sum_{n=-\infty}^{\infty} (-1)^n x[n]z^{-n}$$
$$= \frac{X(z)+X(-z)}{2} \tag{5.181}$$

Using Eq. (5.181) and the system in Figure 5.32, we can express the z-transform of the output in Figure 5.32 as

$$\hat{X}(z) = \left(\frac{F_0(z)G_0(z)+F_1(z)G_1(z)}{2}\right)X(z)$$
$$+\left(\frac{F_0(-z)G_0(z)+F_1(-z)G_1(z)}{2}\right)X(-z) \tag{5.182}$$
$$= \left(\frac{F_0(z)X(z)+F_0(-z)X(-z)}{2}\right)G_0(z) + \left(\frac{F_1(z)X(z)+F_1(-z)X(-z)}{2}\right)G_1(z)$$

which for perfect reconstruction requires the output to be a delayed version of the input, and thus

$$\begin{aligned} F_0(z)G_0(z)+F_1(z)G_1(z) &= 2z^{-(L-1)} \\ F_0(-z)G_0(z)+F_1(-z)G_1(z) &= 0 \end{aligned} \tag{5.183}$$

These conditions are met if we select the so-called *Conjugate Quadrature Filters* (CQF) [17], which are FIR filters that specify $f_1[n]$, $g_0[n]$, and $g_1[n]$ as a function of $f_0[n]$:

$$\begin{aligned} f_1[n] &= (-1)^n f_0[L-1-n] \\ g_0[n] &= f_0[L-1-n] \\ g_1[n] &= f_1[L-1-n] \end{aligned} \tag{5.184}$$

where $f_0[n]$ is an FIR filter of even length L. The z-transforms of Eq. (5.184) are

$$\begin{aligned} F_1(z) &= -z^{-(L-1)}F_0(-z^{-1}) \\ G_0(z) &= z^{-(L-1)}F_0(z^{-1}) \\ G_1(z) &= -F_0(-z) \end{aligned} \tag{5.185}$$

so that the second equation in Eq.(5.183) is met if L is even. In order to analyze the first equation in Eq. (5.183), let's define $P(z)$ as

$$\begin{aligned} P(z) &= F_0(z)F_0(z^{-1}) \\ p[n] &= \sum_m f_0[m]f_0[m+n] \end{aligned} \tag{5.186}$$

then insert Eq. (5.185) into (5.183), use Eq. (5.186), and obtain the following condition:

$$P(z)+P(-z)=2 \tag{5.187}$$

Taking the inverse z-transform of Eq. (5.187) and using Eq. (5.181), we obtain

$$p[n]=\begin{cases} 1 & n=0 \\ 0 & n=2k \end{cases} \tag{5.188}$$

so that all even samples of the autocorrelation of $f_0[n]$ are zero, except for $n = 0$. Since $f_0[n]$ is a half-band low-pass filter, $p[n]$ is also a half-band low-pass filter. The ideal half-band filter $h[n]$

$$h[n]=\frac{\sin(\pi n/2)}{\pi n} \tag{5.189}$$

satisfies Eq. (5.188), as does any half-band zero-phase filter (a linear phase filter with no delay). Therefore, the steps to build CQF are

1. Design a $(2L - 1)$ tap[11] half-band linear-phase low-pass filter $p[n]$ with any available technique, for an even value of L. For example, one could use the Parks McClellan algorithm, constraining the passband and stopband cutoff frequencies so that $\omega_p = \pi - \omega_s$ and using an error weighting that is the same for the passband and stopband. This results in a half-band linear-phase filter with equal ripple δ in both bands. Another possibility is to multiply the ideal half-band filter in Eq. (5.189) by a window with low-pass characteristics.
2. Add a value δ to $p[0]$ so that we can guarantee that $P(e^{j\omega}) \geq 0$ for all ω and thus is a legitimate power spectral density.
3. Spectrally factor $P(z) = F_0(z)F_0(z^{-1})$ by computing its roots.
4. Compute $f_1[n]$, $g_0[n]$ and $g_1[n]$ from Eq. (5.184).

5.7.2. Multiresolution Filterbanks

While the above filterbank has equal bandwidth for both filters, it may be desirable to have varying bandwidths, since it has been proven to work better in speech recognition systems. In this section we show how to use the two-band conjugate quadrature filters described in the previous section to design a filterbank with more than two bands. In fact, multi-resolution analysis such as that of Figure 5.35, are possible with bands of different bandwidths (see Figure 5.36).

One interesting result is that the product of time resolution and frequency resolution is constant (all the tiles in Figure 5.37 have the same area), since filters with smaller bandwidths do not need to be sampled as often. Instead of using Fourier basis for decomposition, multi-resolution filterbanks allow more flexibility in the tiling of the time-frequency plane.

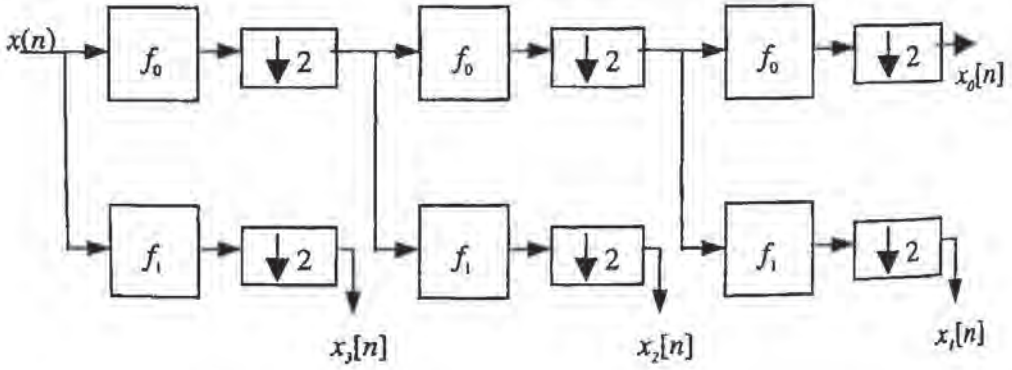


Figure 5.35 Analysis part of a multi-resolution filterbank designed with conjugate quadrature filters. Only $f_0[n]$ needs to be specified.

[11] A filter with N taps is a filter of length N.

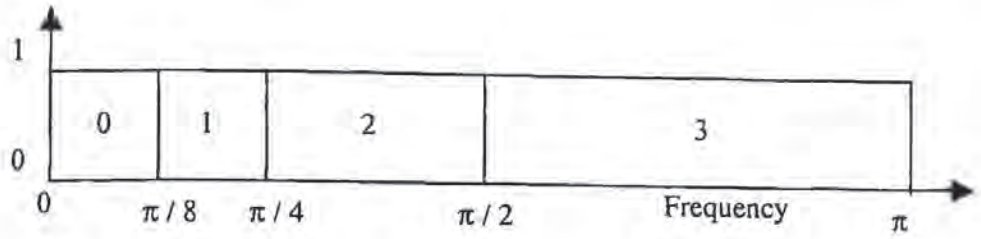


Figure 5.36 Ideal frequency responses of the multi-resolution filterbank of Figure 5.35. Note that $x_0[n]$ and $x_1[n]$ occupy 1/8 of the total bandwidth.

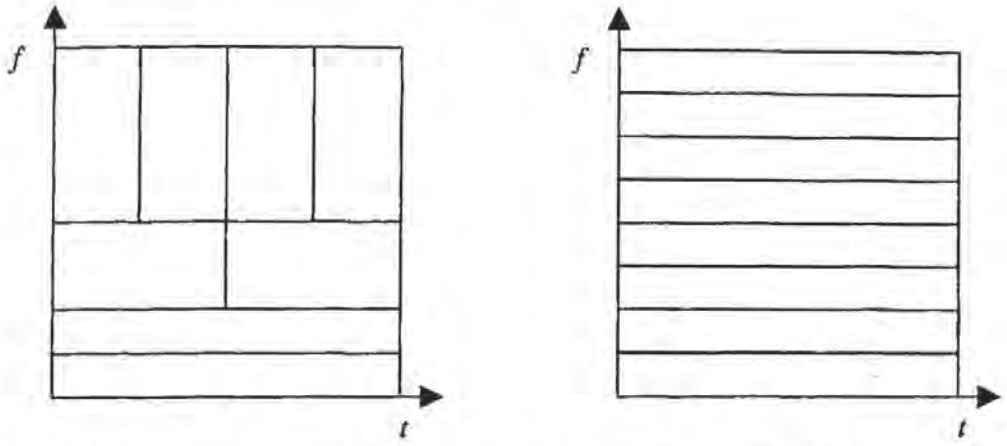


Figure 5.37 Two different time-frequency tilings: the non-uniform filterbank and that obtained through a short-time Fourier transform. Notice that the area of each tile is constant.

5.7.3. The DFT as a Filterbank

It turns out that we can use the Fourier transform to construct a filterbank. To do that, we decompose the input signal $x[n]$ as a sum of *short-time* signals $x_m[n]$

$$x[n] = \sum_{m=-\infty}^{\infty} x_m[n] \tag{5.190}$$

where $x_m[n]$ is obtained as

$$x_m[n] = x[n]w_m[n] \tag{5.191}$$

the product of $x[n]$ by a *window* function $w_m[n]$ of length N. From Eqs. (5.190) and (5.191) we see that the window function has to satisfy

$$\sum_{m=-\infty}^{\infty} w_m[n] = 1 \qquad \forall n \tag{5.192}$$

If the short-term signals $x_m[n]$ are spaced M samples apart, we define the window $w_m[n]$ as:

$$w_m[n] = w[n - Mm] \tag{5.193}$$

where $w[n] = 0$ for $n < 0$ and $n > N$. The windows $w_m[n]$ overlap in time while satisfying Eq. (5.192).

Since $x_m[n]$ has N nonzero values, we can evaluate its length-N DFT as

$$\begin{aligned} X_m[k] &= \sum_{l=0}^{N-1} x_m[Mm+l]e^{-j\omega_k l} \\ &= \sum_{l=0}^{N-1} x[Mm+l]w[l]e^{-j\omega_k l} = \sum_{l=0}^{N-1} x[Mm+l]f_k[-l] \end{aligned} \tag{5.194}$$

where $\omega_k = 2\pi k / N$ and the analysis filters $f_k[l]$ are given by

$$f_k[l] = w[-l]e^{j\omega_k l} \tag{5.195}$$

If we define $\tilde{X}_k[n]$ as

$$\tilde{X}_k[n] = x[n] * f_k[n] = \sum_{r=-\infty}^{\infty} x[n-r]f_k[r] = \sum_{l=0}^{N-1} x[n+l]f_k[-l] \tag{5.196}$$

then Eqs. (5.194) and (5.196) are related by

$$X_m[k] = \tilde{X}_k[mM] \tag{5.197}$$

This manipulation is shown in Figure 5.38, so that the DFT output $X_m[k]$ is $\tilde{X}_k[n]$ decimated by M.

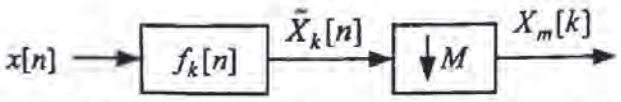


Figure 5.38 Fourier analysis used to build a linear filter.

The short-time signal $x_m[n]$ can be recovered through the inverse DFT of $X_m[k]$ as

$$x_m[mM+l] = h[l]\sum_{k=0}^{N-1} X_m[k]e^{j\omega_k l} \tag{5.198}$$

where $h[n]$ has been defined as

$$h[n] = \begin{cases} 1/N & 0 \le n < N \\ 0 & \text{otherwise} \end{cases} \tag{5.199}$$

so that Eq. (5.198) is valid for all values of l, and not just $0 \le l < N$.

Making the change of variables $mM+l=n$ in Eq. (5.198) and inserting it into Eq. (5.190) results in

$$\begin{aligned} x[n] &= \sum_{m=-\infty}^{\infty} h[n-mM] \sum_{k=0}^{N-1} X_m[k] e^{j\omega_k (n-mM)} \\ &= \sum_{k=0}^{N-1} \sum_{m=-\infty}^{\infty} X_m[k] g_k[n-mM] \end{aligned} \tag{5.200}$$

where the synthesis filters $g_k[n]$ are defined as

$$g_k[n] = h[n] e^{j\omega_k n} \tag{5.201}$$

Now, let's define the upsampled version of $X_m[k]$ as

$$\hat{X}_k[l] = \begin{cases} X_m[k] & l = mM \\ 0 & \text{otherwise} \end{cases} \tag{5.202}$$

which, inserted into Eq. (5.200), yields

$$x[n] = \sum_{k=0}^{N-1} \sum_{l=-\infty}^{\infty} \hat{X}_k[l] g_k[n-l] = \sum_{k=0}^{N-1} \hat{X}_k[n] * g_k[n] \tag{5.203}$$

Thus, the signal can be reconstructed. The block diagram of the analysis/resynthesis filterbank implemented by the DFT can be seen in Figure 5.39, where $x_k[m] = X_m[k]$ and $\tilde{x}[n] = x[n]$.

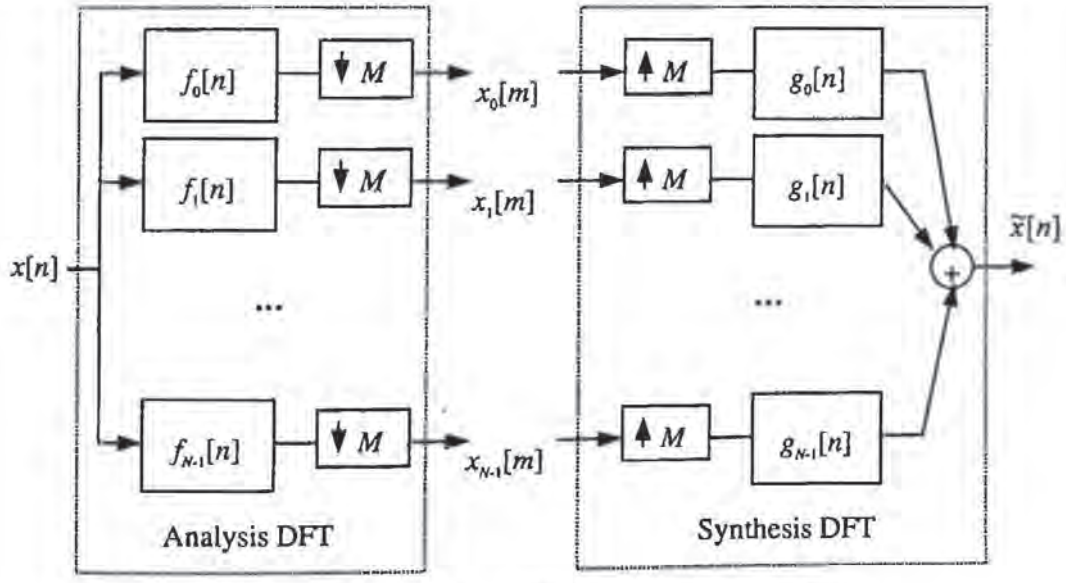


Figure 5.39 A filterbank with N analysis and synthesis filters.

For perfect reconstruction we need $N \geq M$. If $w[n]$ is a rectangular window of length N, the frame rate has to be $M = N$. We can also use overlapping windows with $N = 2M$ (50% overlap), such as Hamming or Hanning windows, and still get perfect reconstruction. The use of such overlapping windows increases the data rate by a factor of 2, but the analysis filters have much less spectral leakage because of the higher attenuation of the Hamming/Hanning window outside the main lobe.

5.7.4. Modulated Lapped Transforms

The filterbank of Figure 5.39 is useful because, as we see in Chapter 7, it is better to quantize the spectral coefficients than the waveform directly. If the DFT coefficients are quantized, there will be some discontinuities at frame boundaries. To solve this problem we can distribute the window $w[n]$ between the analysis and synthesis filters so that

$$w[n] = w_a[n]w_s[n] \tag{5.204}$$

so that the analysis filters are given by

$$f_k[n] = w_a[-n]e^{j\omega_k n} \tag{5.205}$$

and the synthesis filters by

$$g_k[n] = w_s[n]e^{-j\omega_k n} \tag{5.206}$$

This way, if there is a quantization error, the use of a tapering synthesis window will substantially decrease the border effect. A common choice is $w_a[n] = w_s[n]$, which for the case of $w[n]$ being a Hanning window divided by N, results in

$$w_a[n] = w_s[n] = \frac{1}{\sqrt{N}} \sin\left(\frac{\pi n}{N}\right) \quad \text{for} \quad 0 \leq n < N \tag{5.207}$$

so that the analysis and synthesis filters are the reversed versions of each other:

$$f_k[-n] = g_k[n] = \frac{\sin(\pi n/N)}{\sqrt{N}} e^{j2\pi nk/N} \Pi_N[n] = h_k^N[n] \tag{5.208}$$

whose frequency response can be seen in Figure 5.40.

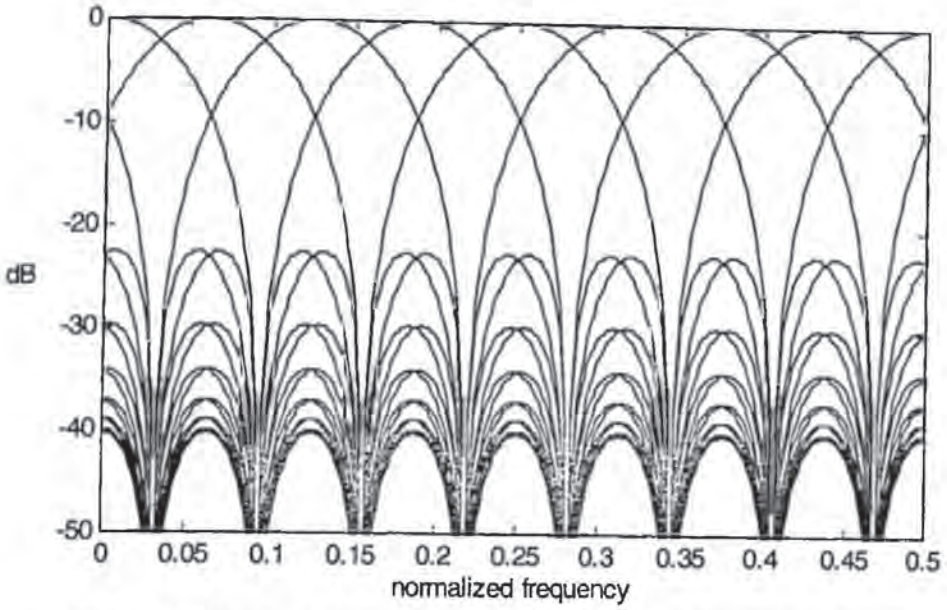


Figure 5.40 Frequency response of the Lapped Orthogonal Transform filterbank.

The functions $h_k^N[n]$ in Eq. (5.208) are sine modulated complex exponentials, which have the property

$$h_k^{N/2}[n] = \sqrt{2} h_k^N[2n] \tag{5.209}$$

which is a property typical of functions called *wavelets*, i.e., they can be obtained from each other by stretching by 2 and scaling them appropriately. Such wavelets can be seen in Figure 5.41.

If instead of modulating a complex exponential we use a cosine sequence, we obtain the *Modulated Lapped Transform* (MLT) [7], also known as the *Modified Discrete Cosine Transform* (MDCT):

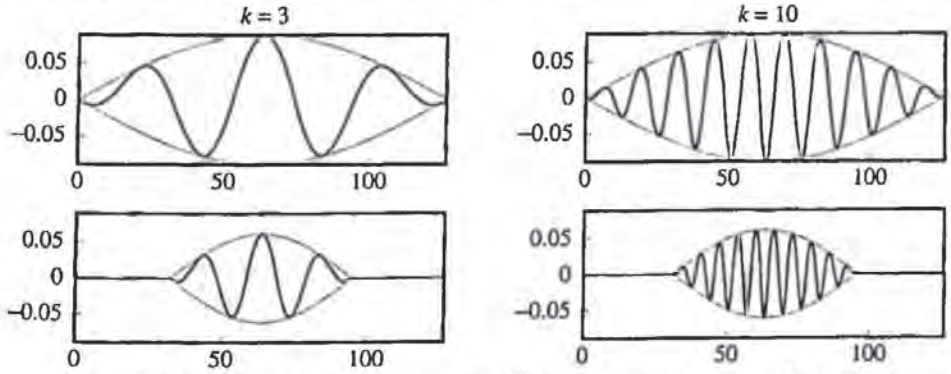


Figure 5.41 Iterations of the wavelet $h_k^N[n]$ for several values of k and N.

$$p_{kn} = f_k[2M-1-n] = g_k[n] = h[n]\sqrt{\frac{2}{M}}\cos\left[\left(k+\frac{1}{2}\right)\left(n+\frac{M+1}{2}\right)\frac{\pi}{M}\right] \tag{5.210}$$

for $k = 0,1,\cdots,M-1$ and $n = 0,1,\cdots,2M-1$. There are M filters with $2M$ taps each, and $h[n]$ is a symmetric window $h[n] = h[2M-1-n]$ that satisfies

$$h^2[n] + h^2[n+M] = 1 \tag{5.211}$$

where the most common choice for $h[n]$ is

$$h[n] = \sin\left[\left(n+\frac{1}{2}\right)\frac{\pi}{2M}\right] \tag{5.212}$$

A fast algorithm can be used to compute these filters based on the DCT, which is called the *Lapped Orthogonal Transform* (LOT).

5.8. STOCHASTIC PROCESSES

While in this chapter we have been dealing with *deterministic signals*, we also need to deal with *noise*, such as the static present in a poorly tuned AM station. To analyze noise signals we need to introduce the concept of stochastic processes, also known as random processes. A *discrete-time* stochastic process $\mathbf{x}[n]$, also denoted by $\mathbf{x}_n$, is a sequence of random variables for each time instant n. *Continuous-time* stochastic processes $\mathbf{x}(t)$, random variables for each value of t, will not be the focus of this book, though their treatment is similar to that of discrete-time processes. We use bold for random variables and regular text for deterministic signals.

Here, we cover the statistics of stochastic processes, defining stationary and ergodic processes and the output of linear systems to such processes.

Example 5.1

We can define a random process $\mathbf{x}[n]$ as

$$\mathbf{x}[n] = \cos[\omega n + \boldsymbol{\varphi}] \tag{5.213}$$

where $\boldsymbol{\varphi}$ is real random variable with a uniform pdf in the interval $(-\pi,\pi)$. Several realizations of this random process are displayed in Figure 5.42.

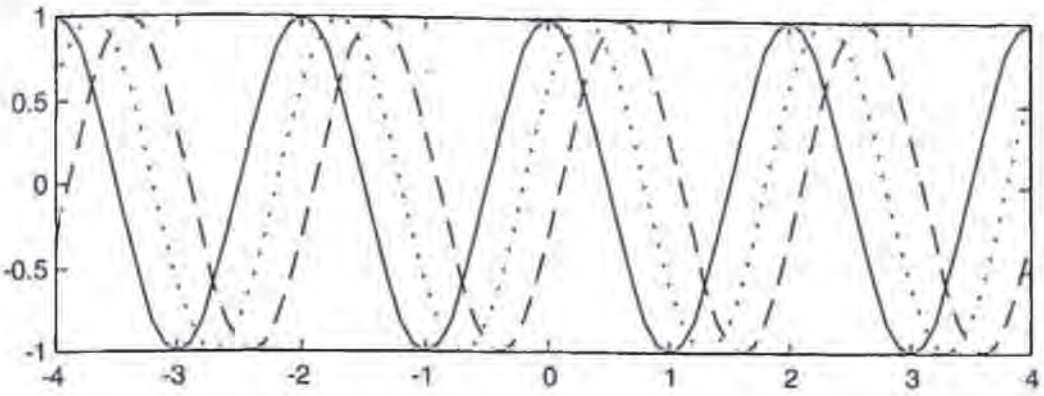


Figure 5.42 Several realizations of a sinusoidal random process with a random phase.

5.8.1. Statistics of Stochastic Processes

In this section we introduce several statistics of stochastic processes such as distribution, density function, mean and autocorrelation. We also define several types of processes depending on these statistics.

For a specific n, $\mathbf{x}[n]$ is a random variable with *distribution*

$$F(x,n) = P\{\mathbf{x}[n] \le x\} \tag{5.214}$$

Its first derivative with respect to x is the first-order density function, or simply the *probability density function* (pdf)

$$f(x,n) = \frac{dF(x,n)}{dx} \tag{5.215}$$

The second-order distribution of the process $\mathbf{x}[n]$ is the joint distribution

$$F(x_1,x_2;n_1,n_2) = P\{\mathbf{x}[n_1] \le x_1, \mathbf{x}[n_2] \le x_2\} \tag{5.216}$$

of the random variables $\mathbf{x}[n_1]$ and $\mathbf{x}[n_2]$. The corresponding density equals

$$f(x_1,x_2;n_1,n_2) = \frac{\partial^2 F(x_1,x_1;n_1,n_2)}{\partial x_1 \partial x_2} \tag{5.217}$$

A complex random process $\mathbf{x}[n] = \mathbf{x}_r[n] + j\mathbf{x}_i[n]$ is specified in terms of the joint statistics of the real processes $\mathbf{x}_r[n]$ and $\mathbf{x}_i[n]$.

The *mean* $\mu[n]$ of $\mathbf{x}[n]$, also called first-order moment, is defined as the expected value of the random variable $\mathbf{x}[n]$ for each value of n:

$$\mu_x[n] = E\{\mathbf{x}[n]\} = \int_{-\infty}^{\infty} \mathbf{x}[n] f(\mathbf{x},n) d\mathbf{x} \tag{5.218}$$

The autocorrelation of complex random process $\mathbf{x}[n]$, also called second-order moment, is defined as

$$R_{xx}[n_1,n_2]=E\{\mathbf{x}[n_1]\mathbf{x}^*[n_2]\}=R_{xx}^*[n_2,n_1] \tag{5.219}$$

which is a statistical average, unlike the autocorrelation of a deterministic signal defined in Eq. (5.45), which was an average over time.

Example 5.2

Let's look at the following sinusoidal random process

$$\mathbf{x}[n]=\mathbf{r}\cos[\omega n+\varphi] \tag{5.220}$$

where $\mathbf{r}$ and φ are independent and φ is uniform in the interval $(-\pi,\pi)$. This process is *zero-mean* because

$$\mu_x[n]=E\{\mathbf{r}\cos[\omega n+\varphi]\}=E\{\mathbf{r}\}E\{\cos[\omega n+\varphi]\}=0 \tag{5.221}$$

since $\mathbf{r}$ and φ are independent and

$$E\{\cos[\omega n+\varphi]\}=\int_{-\pi}^{\pi}\cos[\omega n+\varphi]\frac{1}{2\pi}d\varphi=0 \tag{5.222}$$

Its autocorrelation is given by

$$\begin{aligned}R_{xx}[n_1,n_2]&=E\{\mathbf{r}^2\}\int_{-\pi}^{\pi}\cos[\omega n_1+\varphi]\cos[\omega n_2+\varphi]\frac{1}{2\pi}d\varphi\\&=\frac{1}{2}E\{\mathbf{r}^2\}\int_{-\pi}^{\pi}\{\cos[\omega(n_1+n_2)+\varphi]+\cos[\omega(n_2-n_1)]\}\frac{1}{2\pi}d\varphi\\&=\frac{1}{2}E\{\mathbf{r}^2\}\cos[\omega(n_2-n_1)]\end{aligned} \tag{5.223}$$

which only depends on the time difference n_2-n_1.

An important property of a stochastic process is that its autocorrelation $R_{xx}[n_1,n_2]$ is a *positive-definite* function, i.e., for any a_i,a_j

$$\sum_i\sum_j a_i a_j^* R_{xx}[n_i,n_j]\geq 0 \tag{5.224}$$

which is a consequence of the identity

$$0\leq E\left\{\left|\sum_i a_i\mathbf{x}[n_i]\right|^2\right\}=\sum_i\sum_j a_i a_j^* E\{\mathbf{x}[n_i]\mathbf{x}^*[n_j]\} \tag{5.225}$$

Similarly, the autocovariance of a complex random process is defined as

$$C_{xx}[n_1,n_2]=E\left\{\left(\mathbf{x}[n_1]-\mu_x[n_1]\right)\left(\mathbf{x}[n_2]-\mu_x[n_2]\right)^*\right\}=R_{xx}[n_1,n_2]-\mu_x[n_1]\mu_x^*[n_2] \quad (5.226)$$

The correlation coefficient of process $\mathbf{x}[n]$ is defined as

$$r_{xx}[n_1,n_2]=\frac{C_{xx}[n_1,n_2]}{\sqrt{C_{xx}[n_1,n_1]C_{xx}[n_2,n_2]}} \quad (5.227)$$

An important property of the correlation coefficient is that it is bounded by 1:

$$|r_{xx}[n_1,n_2]|\le 1 \quad (5.228)$$

which is the *Cauchy-Schwarz* inequality. To prove it, we note that for any real number a

$$\begin{aligned}0&\le E\left\{\left|a(\mathbf{x}[n_1]-\mu[n_1])+(\mathbf{x}[n_2]-\mu[n_2])\right|^2\right\}\\&=a^2C_{xx}[n_1,n_1]+2aC_{xx}[n_1,n_2]+C_{xx}[n_2,n_2]\end{aligned} \quad (5.229)$$

Since the quadratic function in Eq. (5.229) is positive for all a, its roots have to be complex, and thus its discriminant has to be negative:

$$C_{xx}^2[n_1,n_2]-C_{xx}[n_1,n_1]C_{xx}[n_2,n_2]\le 0 \quad (5.230)$$

from which Eq. (5.228) is derived.

The cross-correlation of two stochastic processes $\mathbf{x}[n]$ and $\mathbf{y}[n]$ is defined as

$$R_{xy}[n_1,n_2]=E\left\{\mathbf{x}[n_1]\mathbf{y}^*[n_2]\right\}=R_{yx}^*[n_2,n_1] \quad (5.231)$$

where we have explicitly indicated with subindices the random process. Similarly, their cross-covariance is

$$C_{xy}[n_1,n_2]=R_{xy}[n_1,n_2]-\mu_x[n_1]\mu_y^*[n_2] \quad (5.232)$$

Two processes $\mathbf{x}[n]$ and $\mathbf{y}[n]$ are called *orthogonal* iff

$$R_{xy}[n_1,n_2]=0 \quad \text{for every } n_1 \text{ and } n_2 \quad (5.233)$$

They are called *uncorrelated* iff

$$C_{xy}[n_1,n_2]=0 \quad \text{for every } n_1 \text{ and } n_2 \quad (5.234)$$

Independent processes. If two processes $\mathbf{x}[n]$ and $\mathbf{y}[n]$ are such that the random variables $\mathbf{x}[n_1], \mathbf{x}[n_2], \cdots, \mathbf{x}[n_m]$, and $\mathbf{y}[n_1'], \mathbf{y}[n_2'], \cdots, \mathbf{y}[n_m']$ are mutually independent, then these processes are called independent. If two processes are independent, then they are also uncorrelated, though the converse is not generally true.

Gaussian processes. A process $\mathbf{x}[n]$ is called Gaussian if the random variables $\mathbf{x}[n_1], \mathbf{x}[n_2], \cdots, \mathbf{x}[n_m]$ are jointly Gaussian for any m and $n_1, n_2, \cdots, n_m$. If two processes are Gaussian and also uncorrelated, then they are also statistically independent.

5.8.2. Stationary Processes

Stationary processes are those whose statistical properties do not change over time. While truly stationary processes do not exist in speech signals, they are a reasonable approximation and have the advantage of allowing us to use the Fourier transforms defined in Section 5.2.1. In this section we define stationarity and analyze some of its properties.

A stochastic process is called *strict-sense stationary* (SSS) if its statistical properties are invariant to a shift of the origin: i.e., both processes $\mathbf{x}[n]$ and $\mathbf{x}[n+l]$ have the same statistics for any l. Likewise, two processes $\mathbf{x}[n]$ and $\mathbf{y}[n]$ are called *jointly strict-sense stationary* if their joint statistics are the same as those of $\mathbf{x}[n+l]$ and $\mathbf{y}[n+l]$ for any l.

From the definition, it follows that the m^{th}-order density of an SSS process must be such that

$$f(x_1, \cdots, x_m; n_1, \cdots, n_m) = f(x_1, \cdots, x_m; n_1 + l, \cdots, n_m + l) \tag{5.235}$$

for any l. Thus the first-order density satisfies $f(x,n) = f(x,n+l)$ for any l, which means that it is independent of n:

$$f(x,n) = f(x) \tag{5.236}$$

or, in other words, the density function is constant with time.

Similarly, $f(x_1, x_2; n_1 + l, n_2 + l)$ is independent of l, which leads to the conclusion

$$f(x_1, x_2; n_1, n_2) = f(x_1, x_2; m) \qquad m = n_1 - n_2 \tag{5.237}$$

or, in other words, the joint density of $\mathbf{x}[n]$ and $\mathbf{x}[n+m]$ is not a function of n, only of m, the time difference between the two samples.

Let's compute the first two moments of a SSS process:

$$E\{x[n]\} = \int x[n] f(x[n]) = \int x f(x) = \mu \tag{5.238}$$

$$E\{x[n+m]x^*[n]\} = \int x[n+m]x^*[n] f(x[n+m], x[n]) = R_{xx}[m] \tag{5.239}$$

or, in other words, its mean is not a function of time and its autocorrelation depends only on m.

A stochastic process $\mathbf{x}[n]$ that obeys Eq. (5.238) and (5.239) is called *wide-sense stationary* (WSS). From this definition, a SSS process is also a WSS process but the converse is not true in general. Gaussian processes are an important exception, and it can be proved that a WSS Gaussian process is also SSS.

For example, the random process of Eq. (5.213) is WSS, because it has zero mean and its autocorrelation function, as given by Eq. (5.223), is only a function of $m = n_1 - n_2$. By setting $m = 0$ in Eq. (5.239) we see that the average power of a WSS stationary process

$$E\{|x[n]|^2\} = R[0] \tag{5.240}$$

is independent of n.

The autocorrelation of a WSS process is a conjugate-symmetric function, also referred to as a *Hermitian* function:

$$R[-m] = E\{x[n-m]x^*[n]\} = E\{x[n]x^*[n+m]\} = R^*[m] \tag{5.241}$$

so that if $x[n]$ is real, $R[m]$ is even.

From Eqs. (5.226), (5.238), and (5.239) we can compute the autocovariance as

$$C[m] = R[m] - |\mu|^2 \tag{5.242}$$

and its correlation coefficient as

$$r[m] = C[m]/C[0] \tag{5.243}$$

Two processes $\mathbf{x}[n]$ and $\mathbf{y}[n]$ are called jointly WSS if both are WSS and their cross-correlation depends only on $m = n_1 - n_2$:

$$R_{xy}[m] = E\{x[n+m]y^*[n]\} \tag{5.244}$$

$$C_{xy}[m] = R_{xy}[m] - \mu_x \mu_y^* \tag{5.245}$$

5.8.2.1. Ergodic Processes

A critical problem in the theory of stochastic processes is the estimation of their various statistics, such as the mean and autocorrelation given that often only one realization of the random process is available. The first approximation would be to replace the expectation in Eq. (5.218) with its *ensemble average*:

$$\mu[n] \cong \frac{1}{M}\sum_{i=0}^{M-1} x_i[n] \tag{5.246}$$

where $x_i[n]$ are different samples of the random process.

As an example, let $\mathbf{x}[n]$ be the frequency-modulated (FM) random process received by a FM radio receiver:

$$\mathbf{x}[n] = a[n] + \mathbf{v}[n] \tag{5.247}$$

which contains some additive noise $\mathbf{v}[n]$. The realization $x_i[n]$ received by receiver i will be different from the realization $x_j[n]$ for receiver j. We know that each signal has a certain level of noise, so one would hope that by averaging them, we could get the mean of the process for a sufficiently large number of radio receivers.

In many cases, however, only one sample of the process is available. According to Eq. (5.246) this would mean that that the sample signal equals the mean, which does not seem very robust. We could also compute the signal's time average, but this may not tell us much about the random process in general. However, for a special type of random processes called ergodic, their ensemble averages equal appropriate time averages.

A process $\mathbf{x}[n]$ with constant mean

$$E\{\mathbf{x}[n]\} = \mu \tag{5.248}$$

is called *mean-ergodic* if, with probability 1, the ensemble average equals the time average when N approaches infinity:

$$\lim_{N\to\infty} \mu_N = \mu \tag{5.249}$$

where μ_N is the time average

$$\mu_N = \frac{1}{N} \sum_{n=-N/2}^{N/2-1} \mathbf{x}[n] \tag{5.250}$$

which, combined with Eq. (5.248), indicates that μ_N is a random variable with mean μ. Taking expectations in Eq. (5.250) and using Eq. (5.248), it is clear that

$$E\{\mu_N\} = \mu \tag{5.251}$$

so that proving Eq. (5.249) is equivalent to proving

$$\lim_{N\to\infty} \sigma_N^2 = 0 \tag{5.252}$$

with σ_N^2 being the variance of μ_N. It can be shown [12] that a process $\mathbf{x}[n]$ is mean ergodic *iff*

$$\lim_{N\to\infty} \frac{1}{N^2} \sum_{n=-N/2}^{N/2-1} \sum_{m=-N/2}^{N/2-1} C_{xx}[n,m] = 0 \tag{5.253}$$

It can also be shown [12] that a sufficient condition for a WSS process to be mean ergodic is to satisfy

$$\lim_{m\to\infty} C_{xx}[m] = 0 \tag{5.254}$$

which means that if the random variables $\mathbf{x}[n]$ and $\mathbf{x}[n+m]$ are uncorrelated for large m, then process $\mathbf{x}[n]$ is mean ergodic. This is true for many regular processes.

A similar condition can be proven for a WSS process to be covariance ergodic. In most cases in this book we assume ergodicity, first because of convenience for mathematical tractability, and second because it is a good approximation to assume that samples that are far apart are uncorrelated. *Ergodicity allows us to compute means and covariances of random processes by their time averages.*

5.8.3. LTI Systems with Stochastic Inputs

If the WSS random process $\mathbf{x}[n]$ is the input to an LTI system with impulse response $h[n]$, the output

$$\mathbf{y}[n] = \sum_{m=-\infty}^{\infty} h[m]\mathbf{x}[n-m] = \sum_{m=-\infty}^{\infty} h[n-m]\mathbf{x}[m] \tag{5.255}$$

is another WSS random process. To prove this we need to show that the mean is not a function of n:

$$\mu_y[n] = E\{\mathbf{y}[n]\} = \sum_{m=-\infty}^{\infty} h[m]E\{\mathbf{x}[n-m]\} = \mu_x \sum_{m=-\infty}^{\infty} h[m] \tag{5.256}$$

The cross-correlation between input and output is given by

$$\begin{aligned} R_{xy}[m] &= E\{\mathbf{x}[n+m]\mathbf{y}^*[n]\} = \sum_{l=-\infty}^{\infty} h^*[l]E\{\mathbf{x}[n+m]\mathbf{x}^*[n-l]\} \\ &= \sum_{l=-\infty}^{\infty} h^*[l]R_{xx}[m+l] = \sum_{l=-\infty}^{\infty} h^*[-l]R_{xx}[m-l] = h^*[-m]*R_{xx}[m] \end{aligned} \tag{5.257}$$

and the autocorrelation of the output

$$\begin{aligned} R_{yy}[m] &= E\{\mathbf{y}[n+m]\mathbf{y}^*[n]\} = \sum_{l=-\infty}^{\infty} h[l]E\{\mathbf{x}[n+m-l]\mathbf{y}^*[n]\} \\ &= \sum_{l=-\infty}^{\infty} h[l]R_{xy}[m-l] = h[m]*R_{xy}[m] = h[m]*h^*[-m]*R_{xx}[m] \end{aligned} \tag{5.258}$$

is only a function of m.

5.8.4. Power Spectral Density

The Fourier transform of a WSS random process $\mathbf{x}[n]$ is a stochastic process in the variable ω

$$\mathbf{X}(\omega) = \sum_{n=-\infty}^{\infty} \mathbf{x}[n]e^{-j\omega n} \tag{5.259}$$

whose autocorrelation is given by

$$\begin{aligned} E\{\mathbf{X}(\omega+u)\mathbf{X}^*(\omega)\} &= E\left\{\sum_{l=-\infty}^{\infty} \mathbf{x}[l]e^{-j(\omega+u)l} \sum_{m=-\infty}^{\infty} \mathbf{x}^*[m]e^{j\omega m}\right\} \\ &= \sum_{n=-\infty}^{\infty} e^{-j(\omega+u)n} \sum_{m=-\infty}^{\infty} E\{\mathbf{x}[m+n]\mathbf{x}^*[m]\}e^{-jum} \end{aligned} \tag{5.260}$$

where we made a change of variables $l = n+m$ and changed the order of expectation and summation. Now, if $\mathbf{x}[n]$ is WSS

$$R_{xx}[n] = E\{\mathbf{x}[m+n]\mathbf{x}^*[m]\} \tag{5.261}$$

and if we set $u = 0$ in Eq. (5.260) together with Eq. (5.261), then we obtain

$$S_{xx}(\omega) = E\{|\mathbf{X}(\omega)|^2\} = \sum_{n=-\infty}^{\infty} R_{xx}[n]e^{-j\omega n} \tag{5.262}$$

$S_{xx}(\omega)$ is called the *power spectral density* of the WSS random process $\mathbf{x}[n]$, and it is the Fourier transform of its autocorrelation function $R_{xx}[n]$, with the inversion formula being

$$R_{xx}[n] = \frac{1}{2\pi}\int_{-\infty}^{\infty} S_{xx}(\omega)e^{j\omega n}d\omega \tag{5.263}$$

Note that Eqs. (5.48) and (5.263) are identical, though in one case we compute the autocorrelation of a signal as a time average, and the other is the autocorrelation of a random process as an ensemble average. For an ergodic process both are the same.

Just as we take Fourier transforms of deterministic signals, we can also compute the power spectral density of a random process as long as it is wide-sense stationary, which is why these wide-sense stationary processes are so useful.

If the random process $\mathbf{x}[n]$ is real then $R_{xx}[n]$ is real and even and, using properties in Table 5.5, $S_{xx}(\omega)$ is also real and even.

Parseval's theorem for random processes also applies here:

$$E\{|\mathbf{x}[n]|^2\} = R_{xx}[0] = \frac{1}{2\pi}\int_{-\pi}^{\pi} S_{xx}(\omega)d\omega \tag{5.264}$$

so that we can compute the signal's energy from the area under $S_{xx}(\omega)$. Let's get a physical interpretation of $S_{xx}(\omega)$. In order to do that we can similarly derive the cross-power spectrum $S_{xy}(\omega)$ of two WSS random processes $\mathbf{x}[n]$ and $\mathbf{y}[n]$ as the Fourier transform of their cross-correlation:

$$S_{xy}(\omega) = \sum_{n=-\infty}^{\infty} R_{xy}[n] e^{-j\omega n} \tag{5.265}$$

which allows us, taking Fourier transforms in Eq. (5.257), to obtain the cross-power spectrum between input and output to a linear system as

$$S_{xy}(\omega) = S_{xx}(\omega) H^*(\omega) \tag{5.266}$$

Now, taking the Fourier transform of Eq. (5.258), the power spectrum of the output is thus given by

$$S_{yy}(\omega) = S_{xy}(\omega) H(\omega) = S_{xx}(\omega) |H(\omega)|^2 \tag{5.267}$$

Finally, suppose we filter $\mathbf{x}[n]$ through the ideal bandpass filter

$$H_b(\omega) = \begin{cases} \sqrt{\pi / c} & \omega_0 - c < \omega < \omega_0 + c \\ 0 & \textit{otherwise} \end{cases} \tag{5.268}$$

The energy of the output process is

$$0 \le E\left\{ |\mathbf{y}[n]|^2 \right\} = R_{yy}[0] = \frac{1}{2\pi} \int_{-\pi}^{\pi} S_{yy}(\omega) d\omega = \frac{1}{2c} \int_{\omega_0 - c}^{\omega_0 + c} S_{xx}(\omega) d\omega \tag{5.269}$$

so that taking the limit when $c \to 0$ results in

$$0 \le \lim_{c \to 0} \frac{1}{2c} \int_{\omega_0 - c}^{\omega_0 + c} S_{xx}(\omega) d\omega = S_{xx}(\omega_0) \tag{5.270}$$

which is the *Wiener-Khinchin* theorem and says that the power spectrum of a WSS process $\mathbf{x}[n]$, real or complex, is always positive for any ω. Equation (5.269) also explains the name power spectral density, because $S_{xx}(\omega)$ represents the density of power at any given frequency ω.

5.8.5. Noise

A process $\mathbf{x}[n]$ is *white noise* if, and only if, its samples are uncorrelated:

$$C_{xx}[n_1, n_2] = C[n_1] \delta[n_1 - n_2] \tag{5.271}$$

and is zero-mean $\mu_x[n] = 0$.

If in addition $\mathbf{x}[n]$ is WSS, then

$$C_{xx}[n] = R_{xx}[n] = q\delta[n] \tag{5.272}$$

which has a flat power spectral density

$$S_{xx}(\omega) = q \quad \text{for all } \omega \tag{5.273}$$

The thermal noise phenomenon in metallic resistors can be accurately modeled as white Gaussian noise. White noise doesn't have to be Gaussian (white Poisson impulse noise is one of many other possibilities).

Colored noise is defined as a zero-mean WSS process whose samples are correlated with autocorrelation $R_{xx}[n]$. Colored noise can be generated by passing white noise through a filter $h[n]$ such that $S_{xx}(\omega) = |H(\omega)|^2$. A type of colored noise that is very frequently encountered in speech signals is the so-called *pink noise*, whose power spectral density decays with ω. A more in-depth discussion of noise and its effect on speech signals is included in Chapter 10.

5.9. Historical Perspective and Further Reading

It is impossible to cover the field of Digital Signal Processing in just one chapter. The book by Oppenheim and Schafer [10] is one of the most widely used as a comprehensive treatment. For a more in-depth coverage of digital filter design, you can read the book by Parks and Burrus [13]. A detailed study of the FFT is provided by Burrus and Parks [2]. The theory of signal processing for analog signals can be found in Oppenheim and Willsky [11]. The theory of random signals can be found in Papoulis [12]. Multirate processing is well studied in Crochiere and Rabiner [4]. Razavi [16] covers analog-digital conversion. Software programs, such as MATLAB [1], contain a large number of packaged subroutines. Malvar [7] has extensive coverage of filterbanks and lapped transforms.

The field of Digital Signal Processing has a long history. The greatest advances in the field started in the 17th century. In 1666, English mathematician and physicist Sir *Isaac Newton* (1642-1727) invented differential and integral calculus, which was independently discovered in 1675 by German mathematician *Gottfried Wilhelm Leibniz* (1646-1716). They both developed discrete mathematics and numerical methods to solve such equations when closed-form solutions were not available. In the 18th century, these techniques were further extended. Swiss brothers *Johann* (1667-1748) and *Jakob Bernoulli* (1654-1705) invented the calculus of variations and polar coordinates. French mathematician *Joseph Louis Lagrange* (1736-1813) developed algorithms for numerical integration and interpolation of continuous functions. The famous Swiss mathematician *Leonhard Euler* (1707-1783) developed the theory of complex numbers and number theory so useful in the DSP field, in addition to the first full analytical treatment of algebra, the theory of equations, trigonometry and analytical geometry. In 1748, Euler examined the motion of a vibrating string and discovered that sinusoids are eigenfunctions for linear systems. Swiss scientist *Daniel Bernoulli* (1700-1782),

son of Johann Bernoulli, also conjectured in 1753 that all physical motions of a string could be represented by linear combinations of normal modes. However, both Euler and Bernoulli, and later Lagrange, discarded the use of trigonometric series because *it was impossible to represent signals with corners*. The 19th century brought us the theory of harmonic analysis. One of those who contributed most to the field of Digital Signal Processing is *Jean Baptiste Joseph Fourier* (1768-1830), a French mathematician who in 1822 published *The Analytical Theory of Heat*, where he derived a mathematical formulation for the phenomenon of heat conduction. In this treatise, he also developed the concept of Fourier series and harmonic analysis and the Fourier transform. One of Fourier's disciples, the French mathematician *Simeon-Denis Poisson* (1781-1840), studied the convergence of Fourier series together with countryman *Augustin Louis Cauchy* (1789-1857). Nonetheless, it was German *Peter Dirichlet* (1805-1859) who gave the first set of conditions sufficient to guarantee the convergence of a Fourier series. French mathematician *Pierre Simon Laplace* (1749-1827) invented the Laplace transform, a transform for continuous-time signals over the whole complex plane. French mathematician *Marc-Antoine Parseval* (1755-1836) derived the theorem that carries his name. German *Leopold Kronecker* (1823-1891) did work with discrete delta functions. French mathematician *Charles Hermite* (1822-1901) discovered complex conjugate matrices. American *Josiah Willard Gibbs* (1839-1903) studied the phenomenon of Fourier approximations to periodic square waveforms.

Until the early 1950s, all signal processing was analog, including the *long-playing* (LP) record first released in 1948. Pulse Code Modulation (PCM) had been invented by *Paul M. Rainey* in 1926 and independently by *Alan H. Reeves* in 1937, but it wasn't until 1948 when *Oliver, Pierce,* and *Shannon* [9] laid the groundwork for PCM (see Chapter 7 for details). Bell Labs engineers developed a PCM system in 1955, the so-called T-1 carrier system, which was put into service in 1962 as the world's first common-carrier digital communications system and is still used today. The year 1948 also saw the invention of the transistor at Bell Labs and a small prototype computer at Manchester University and marked the birth of modern Digital Signal Processing. In 1958, *Jack Kilby* of Texas Instruments invented the integrated circuit and in 1970, researchers at Lincoln Laboratories developed the first real-time DSP computer, which performed signal processing tasks about 100 times faster than general-purpose computers of the time. In 1978, Texas Instruments introduced *Speak & Spell*™, a toy that included an integrated circuit especially designed for speech synthesis. Intel Corporation introduced in 1971 the 4-bit Intel 4004, the first general-purpose microprocessor chip, and in 1972 they introduced the 8-bit 8008. In 1982 Texas Instruments introduced the TMS32010, the first commercially viable single-chip Digital Signal Processor (DSP), a microprocessor specially designed for fast signal processing operations. At a cost of about $100, the TMS32010 was a 16-bit fixed-point chip with a hardware multiplier built-in that executed 5 million instructions per second (MIPS). Gordon Moore, Intel's founder, came up with the law that carries his name stating that computing power doubles every 18 months, allowing ever faster processors. By the end of the 20th century, DSP chips could perform floating-point operations at a rate over 1000MIPS and had a cost below $5, so that today they are found in many devices from automobiles to cellular phones.

While hardware improvements significantly enabled the development of the field, digital algorithms were also needed. The 1960s saw the discovery of many of the concepts described in this chapter. In 1965, *James W. Cooley* and *John W. Tukey* [3] discovered the FFT, although it was later found [6] that German mathematician *Carl Friedrich Gauss* (1777-1855) had already invented it over a century earlier. The FFT sped up calculations by orders of magnitude, which opened up many possible algorithms for the slow computers of the time. *James F. Kaiser, Bernard Gold,* and *Charles Rader* published key papers on digital filtering. *John Stockham* and *Howard Helms* independently discovered fast convolution by doing convolution with FFTs.

An association that has had a large impact on the development of modern Digital Signal Processing is the Institute of Electrical and Electronic Engineers (IEEE), which has over 350,000 members in 150 nations and is the world's largest technical organization. It was founded in 1884 as the American Institute of Electrical Engineers (AIEE). IEEE's other parent organization, the Institute of Radio Engineers (IRE), was founded in 1912, and the two merged in 1963. The IEEE Signal Processing Society is a society within the IEEE devoted to Signal Processing. Originally founded in 1948 as the Institute of Radio Engineers Professional Group on Audio, it was later renamed the IEEE Group on Audio (1964), the IEEE Audio and Electroacoustics group (1965), the IEEE group on Acoustics Speech and Signal Processing (1974), the Acoustic, Speech and Signal Processing Society (1976), and finally IEEE Signal Processing Society (1989). In 1976 the society initiated its practice of holding an annual conference, the International Conference on Acoustic, Speech and Signal Processing (ICASSP), which has been held every year since, and whose proceedings constitute an invaluable reference. Frederik Nebeker [8] provides a history of the society's first 50 years rich in insights from the pioneers.

REFERENCES

[1] Burrus, C.S., *et al.*, *Computer-Based Exercises for Signal Processing Using Matlab*, 1994, Upper Saddle River, NJ, Prentice Hall.

[2] Burrus, C.S. and T.W. Parks, *DFT/FFT and Convolution Algorithms: Theory and Implementation*, 1985, New York, John Wiley.

[3] Cooley, J.W. and J.W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," *Mathematics of Computation*, 1965, **19**(Apr.), pp. 297-301.

[4] Crochiere, R.E. and L.R. Rabiner, *Multirate Digital Signal Processing*, 1983, Upper Saddle River, NJ, Prentice-Hall.

[5] Duhamel, P. and H. Hollman, "Split Radix FFT Algorithm," *Electronic Letters*, 1984, **20**(January), pp. 14-16.

[6] Heideman, M.T., D.H. Johnson, and C.S. Burrus, "Gauss and the History of the Fast Fourier Transform," *IEEE ASSP Magazine*, 1984, **1**(Oct), pp. 14-21.

[7] Malvar, H., *Signal Processing with Lapped Transforms*, 1992, Artech House.

[8] Nebeker, F., *Fifty Years of Signal Processing: The IEEE Signal Processing Society and Its Technologies*, 1998, IEEE.

[9] Oliver, B.M., J.R. Pierce, and C. Shannon, "The Philosophy of PCM," *Proc. Institute of Radio Engineers*, 1948, **36**, pp. 1324-1331.

[10] Oppenheim, A.V., R.W. Schafer, and J.R. Buck, *Discrete-Time Signal Processing*, 2nd ed., 1999, Prentice-Hall, Upper Saddle River, NJ.

[11] Oppenheim, A.V. and A.S. Willsky, *Signals and Systems*, 1997, Upper Saddle River, NJ, Prentice-Hall.

[12] Papoulis, A., *Probability, Random Variables, and Stochastic Processes*, 3rd ed., 1991, New York, McGraw-Hill.

[13] Parks, T.W. and C.S. Burrus, *Digital Filter Design*, 1987, New York, NY, John Wiley.

[14] Parks, T.W. and J.H. McClellan, "A Program for the Design of Linear Phase Finite Impulse Response Filters," *IEEE Trans. on Audio Electroacoustics*, 1972, **AU-20**(Aug), pp. 195-199.

[15] Rao, K.R. and P. Yip, *Discrete Cosine Transform: Algorithms, Advantages and Applications*, 1990, San Diego, CA, Academic Press.

[16] Razavi, B., *Principles of Data Conversión System Design*, 1995, IEEE Press.

[17] Smith, M.J.T. and T.P. Barnwell, "A Procedure for Designing Exact Reconstruction Filter Banks for Tree Structured Subband Coders," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1984, San Diego, CA, pp. 27.1.1-27.1.4.

CHAPTER 6

Speech Signal Representations

This chapter presents several representations for speech signals useful in speech coding, synthesis, and recognition. The central theme is the decomposition of the speech signal as a source passed through a linear time-varying filter. This filter can be derived from models of speech production based on the theory of acoustics where the source represents the air flow at the vocal cords, and the filter represents the resonances of the vocal tract which change over time. Such a source-filter model is illustrated in Figure 6.1. We describe methods to compute both the source or *excitation* $e[n]$ and the filter $h[n]$ from the speech signal $x[n]$.

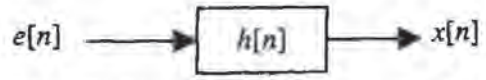


Figure 6.1 Basic source-filter model for speech signals.

To estimate the filter we present methods inspired by speech production models (such as linear predictive coding and cepstral analysis) as well as speech perception models (such

as mel-frequency cepstrum). Once the filter has been estimated, the source can be obtained by passing the speech signal through the inverse filter. Separation between source and filter is one of the most difficult challenges in speech processing.

It turns out that phoneme classification (either by human or by machines) is mostly dependent on the characteristics of the filter. Traditionally, speech recognizers estimate the filter characteristics and ignore the source. Many speech synthesis techniques use a source-filter model because it allows flexibility in altering the pitch and the filter. Many speech coders also use this model because it allows a low bit rate.

We first introduce the spectrogram as a representation of the speech signal that highlights several of its properties and describe the short-time Fourier analysis, which is the basic tool to build the spectrograms of Chapter 2. We then introduce several techniques used to separate source and filter: LPC and cepstral analysis, perceptually motivated models, formant tracking, and pitch tracking.

6.1. SHORT-TIME FOURIER ANALYSIS

In Chapter 2, we demonstrated how useful *spectrograms* are to analyze phonemes and their transitions. A spectrogram of a time signal is a special two-dimensional representation that displays time in its horizontal axis and frequency in its vertical axis. A gray scale is typically used to indicate the energy at each point (t, f) with white representing low energy and black high energy. In this section we cover short-time Fourier analysis, the basic tool with which to compute them.

The idea behind a spectrogram, such as that in Figure 6.2, is to compute a Fourier transform every 5 milliseconds or so, displaying the energy at each time/frequency point. Since some regions of speech signals shorter than, say, 100 milliseconds often appear to be periodic, we use the techniques discussed in Chapter 5. However, the signal is no longer periodic when longer segments are analyzed, and therefore the exact definition of Fourier transform cannot be used. Moreover, that definition requires knowledge of the signal for infinite time. For both reasons, a new set of techniques called *short-time analysis* are proposed. These techniques decompose the speech signal into a series of short segments, referred to as *analysis frames*, and analyze each one independently.

In Figure 6.2 (a), note the assumption that the signal can be approximated as periodic within X and Y is reasonable. In regions (Z, W) and (H, G), the signal is not periodic and looks like *random noise*. The signal in (Z, W) appears to have different noisy characteristics than those of segment (H, G). The use of an analysis frame implies that the region is short enough for the behavior (periodicity or noise-like appearance) of the signal to be approximately constant. If the region where speech seems periodic is too long, the pitch period is not constant and not all the periods in the region are similar. In essence, the speech region has to be short enough so that the signal is *stationary* in that region: i.e., the signal characteristics (whether periodicity or noise-like appearance) are uniform in that region. A more formal definition of stationarity is given in Chapter 5.

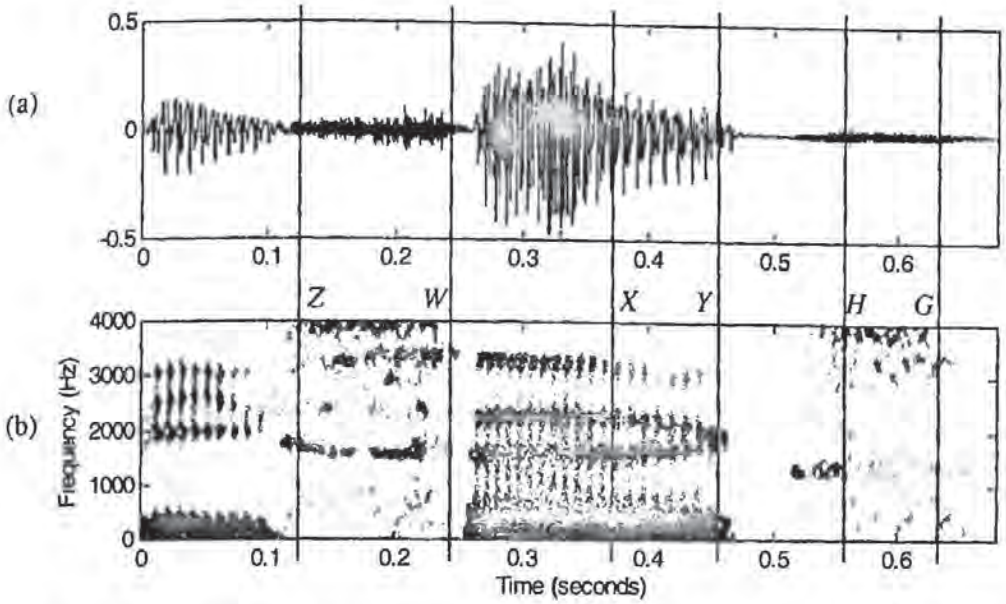


Figure 6.2 (a) Waveform with (b) its corresponding wideband spectrogram. Darker areas mean higher energy for that time and frequency. Note the vertical lines spaced by pitch periods.

Similarly to the filterbanks described in Chapter 5, given a speech signal $x[n]$, we define the short-time signal $x_m[n]$ of frame m as

$$x_m[n] = x[n]w_m[n] \tag{6.1}$$

the product of $x[n]$ by a *window* function $w_m[n]$, which is zero everywhere except in a small region.

While the window function can have different *values* for different frames m, a popular choice is to keep it constant for all frames:

$$w_m[n] = w[m-n] \tag{6.2}$$

where $w[n] = 0$ for $|n| > N/2$. In practice, the window length is on the order of 20 to 30 ms.

With the above framework, the short-time Fourier representation for frame m is defined as

$$X_m(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x_m[n]e^{-j\omega n} = \sum_{n=-\infty}^{\infty} w[m-n]x[n]e^{-j\omega n} \tag{6.3}$$

with all the properties of Fourier transforms studied in Chapter 5.

In Figure 6.3 we show the short-time spectrum of voiced speech. Note that there are a number of peaks in the spectrum. To interpret this, assume the properties of $x_m[n]$ persist outside the window, and that, therefore, the signal is periodic with period M in the true sense. In this case, we know (see Chapter 5) that its spectrum is a sum of impulses

$$X_m(e^{j\omega}) = 2\pi \sum_{k=-\infty}^{\infty} X_m[k]\delta(\omega - 2\pi k / M) \tag{6.4}$$

Given that the Fourier transform of $w[n]$ is

$$W(e^{j\omega}) = \sum_{n=-\infty}^{\infty} w[n]e^{-j\omega n} \tag{6.5}$$

so that the transform of $w[m-n]$ is $W(e^{-j\omega})e^{-j\omega m}$. Therefore, using the convolution property, the transform of $x[n]w[m-n]$ for fixed m is the convolution in the frequency domain

$$X_m(e^{j\omega}) = \sum_{k=-\infty}^{\infty} X_m[k]W(e^{-j(\omega-2\pi k/N)})e^{-j(\omega-2\pi k/N)m} \tag{6.6}$$

which is a sum of weighted $W(e^{j\omega})$, shifted on every harmonic, the narrow peaks seen in Figure 6.3 (b) with a rectangular window. The short-time spectrum of a periodic signal exhibits peaks (equally spaced $2\pi / M$ apart) representing the harmonics of the signal. We estimate $X_m[k]$ from the short-time spectrum $X_m(e^{j\omega})$, and we see the importance of the length and choice of window.

Equation (6.6) indicates that one cannot recover $X_m[k]$ by simply retrieving $X_m(e^{j\omega})$, although the approximation can be reasonable if there is a small value of λ such that

$$W(e^{j\omega}) \approx 0 \text{ for } |\omega - \omega_k| > \lambda \tag{6.7}$$

which is the case outside the main lobe of the window's frequency response.

Recall from Section 5.4.2.1 that, for a rectangular window of length N, $\lambda = 2\pi / N$. Therefore, Eq. (6.7) is satisfied if $N \geq M$, i.e., the rectangular window contains at least one pitch period. The width of the main lobe of the window's frequency response is inversely proportional to the length of the window. The pitch period in Figure 6.3 is $M = 71$ at a sampling rate of 8 kHz. A shorter window is used in Figure 6.3 (c), which results in wider analysis lobes, though still visible.

Also recall from Section 5.4.2.2 that for a Hamming window of length N, $\lambda = 4\pi / N$: twice as wide as that of the rectangular window, which entails $N \geq 2M$. Thus, for Eq. (6.7) to be met, a Hamming window must contain at least two pitch periods. The lobes are visible in Figure 6.3 (d) since $N = 240$, but they are not visible in Figure 6.3 (e) since $N = 120$, and $N < 2M$.

In practice, one cannot know what the pitch period is ahead of time, which often means you need to prepare for the lowest pitch period. A low-pitched voice with a

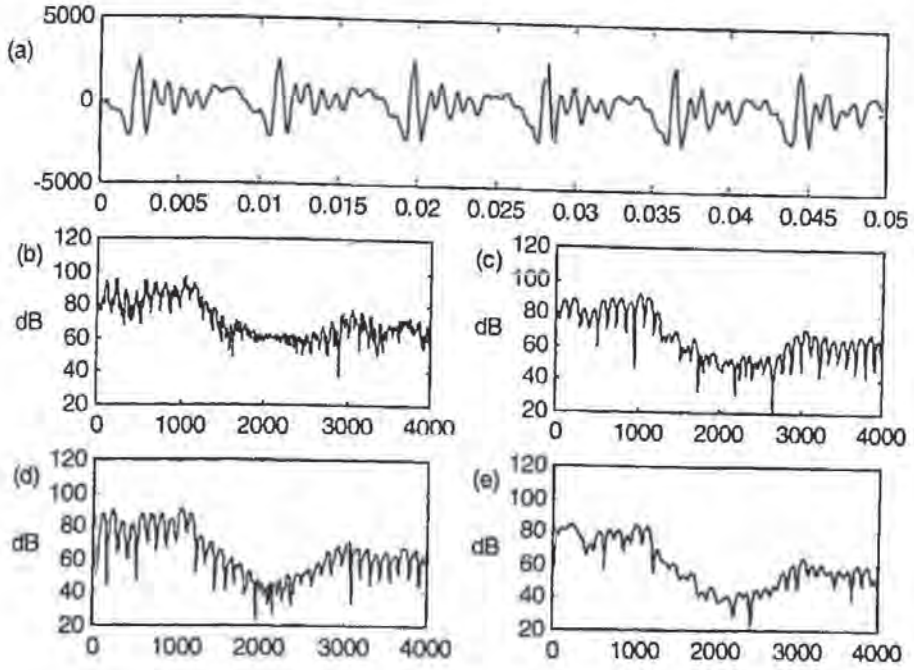


Figure 6.3 Short-time spectrum of male voiced speech (vowel /ah/ with local pitch of 110Hz): (a) time signal, spectra obtained with (b) 30 ms rectangular window and (c) 15 ms rectangular window, (d) 30 ms Hamming window, (e) 15 ms Hamming window. The window lobes are not visible in (e), since the window is shorter than 2 times the pitch period. Note the spectral leakage present in (b).

$F_0 = 50\,\text{Hz}$ requires a rectangular window of at least 20 ms and a Hamming window of at least 40 ms for the condition in Eq. (6.7) to be met. If speech is non-stationary within 40 ms, taking such a long window implies obtaining an average spectrum during that segment instead of several distinct spectra. For this reason, the rectangular window provides better *time resolution* than the Hamming window. Figure 6.4 shows analysis of female speech for which shorter windows are feasible.

But the frequency response of the window is not completely zero outside its main lobe, so one needs to see the effects of this incorrect assumption. From Section 5.4.2.1 note that the second lobe of a rectangular window is only approximately 17 dB below the main lobe. Therefore, for the k^{th} harmonic the value of $X_m(e^{j2\pi k/M})$ contains not $X_m[k]$, but also a weighted sum of $X_m[l]$. This phenomenon is called *spectral leakage* because the amplitude of one harmonic leaks over the rest and masks its value. If the signal's spectrum is white, spectral leakage does not cause a major problem, since the effect of the second lobe on a harmonic is only $10\log_{10}(1+10^{-17/10}) = 0.08\text{dB}$. On the other hand, if the signal's spectrum decays more quickly in frequency than the decay of the window, the spectral leakage results in inaccurate estimates.

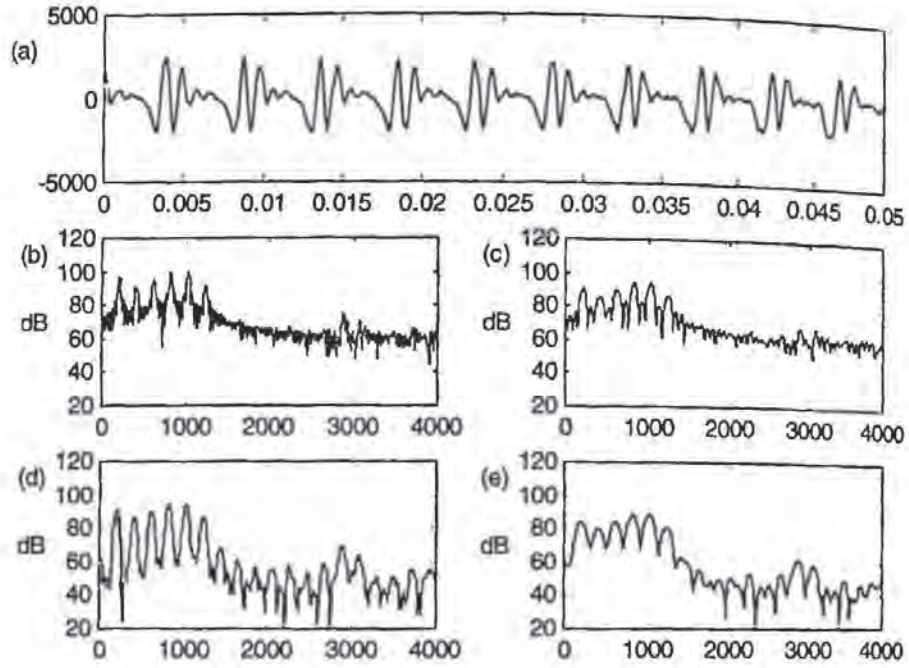


Figure 6.4 Short-time spectrum of female voiced speech (vowel /aa/ with local pitch of 200Hz): (a) time signal, spectra obtained with (b) 30 ms rectangular window and (c) 15 ms rectangular window, (d) 30 ms Hamming window, (e) 15 ms Hamming window. In all cases the window lobes are visible, since the window is longer than 2 times the pitch period. Note the spectral leakage present in (b) and (c).

From Section 5.4.2.2, observe that the second lobe of a Hamming window is approximately 43 dB, which means that the spectral leakage effect is much less pronounced. Other windows, such as Hanning, or triangular windows, also offer less spectral leakage than the rectangular window. This important fact is the reason why, despite their better time resolution, rectangular windows are rarely used for speech analysis. In practice, window lengths are on the order of 20 to 30 ms. This choice is a compromise between the stationarity assumption and the frequency resolution.

In practice, the Fourier transform in Eq. (6.3) is obtained through an FFT. If the window has length N, the FFT has to have a length greater than or equal to N. Since FFT algorithms often have lengths that are powers of 2 ($L = 2^R$), the windowed signal with length N is augmented with $(L-N)$ zeros either before, after, or both. This process is called *zero-padding*. A larger value of L provides a finer description of the discrete Fourier transform; but it does not increase the analysis frequency resolution: this is the sole mission of the window length N.

In Figure 6.3, observe the broad peaks, resonances or formants, which represent the filter characteristics. For voiced sounds there is typically more energy at low frequencies

than at high frequencies, also called *roll-off*. It is impossible to determine exactly the filter characteristics, because we know only samples at the harmonics, and we have no knowledge of the values in between. In fact, the resonances are less obvious in Figure 6.4 because the harmonics sample the spectral envelope less densely. For high-pitched female speakers and children, it is even more difficult to locate the formant resonances from the short-time spectrum.

Figure 6.5 shows the short-time analysis of unvoiced speech, for which no regularity is observed.

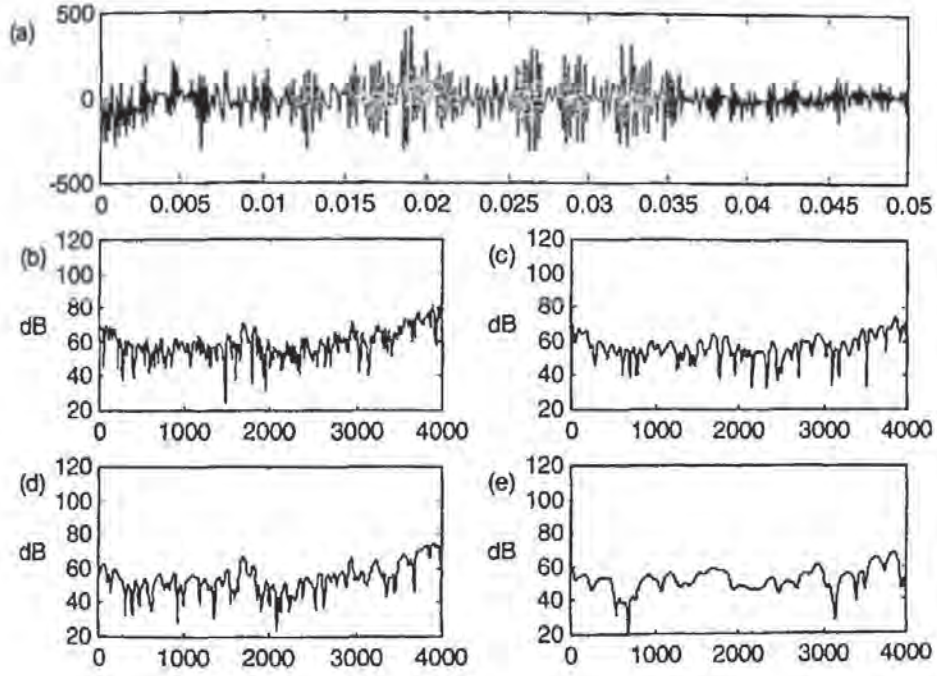


Figure 6.5 Short-time spectrum of unvoiced speech: (a) time signal, (b) 30 ms rectangular window, (c) 15 ms rectangular window, (d) 30 ms Hamming window, (e) 15 ms Hamming window.

6.1.1. Spectrograms

Since the spectrogram displays just the energy and not the phase of the short-term Fourier transform, we compute the energy as

$$\log|X[k]|^2 = \log\left(X_r^2[k] + X_i^2[k]\right) \tag{6.8}$$

with this value converted to a gray scale according to Figure 6.6. Pixels whose values have not been computed are interpolated. The slope controls the contrast of the spectrogram, while the saturation points for white and black control the dynamic range.

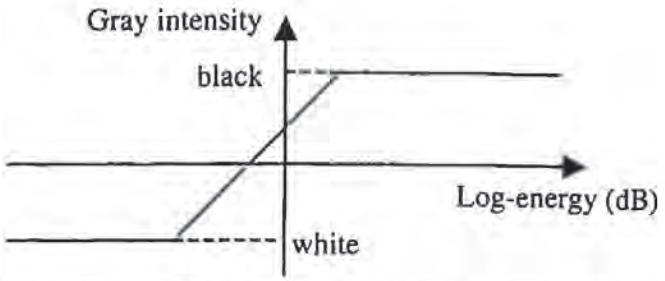


Figure 6.6 Conversion between log-energy values (in the x-axis) and gray scale (in the y-axis). Larger log-energies correspond to a darker gray color. There is a linear region for which more log-energy corresponds to darker gray, but there is saturation at both ends. Typically there is 40 to 60 dB between the pure white and the pure black.

There are two main types of spectrograms: *narrow-band* and *wide-band*. Wide-band spectrograms use relatively short windows (< 10 ms) and thus have good time resolution at the expense of lower frequency resolution, since the corresponding filters have wide bandwidths (> 200 Hz) and the harmonics cannot be seen. Note the vertical stripes in Figure 6.2, due to the fact that some windows are centered at the high part of a pitch pulse, and others in between have lower energy. Spectrograms can aid in determining formant frequencies and fundamental frequency, as well as voiced and unvoiced regions.

Narrow-band spectrograms use relatively long windows (> 20 ms), which lead to filters with narrow bandwidth (< 100 Hz). On the other hand, time resolution is lower than for wide-band spectrograms (see Figure 6.7). Note that the harmonics can be clearly seen, because some of the filters capture the energy of the signal's harmonics, and filters in between have little energy.

Some implementation details also need to be taken into account. Since speech signals are real, the Fourier transform is Hermitian, and its power spectrum is also even. Thus, it is only necessary to display values for $0 \leq k \leq N/2$ for N even. In addition, while the traditional spectrogram uses a gray scale, a color scale can also be used, or even a 3-D representation. In addition, to make the spectrograms easier to read, sometimes the signal is first preemphasized (typically with a first-order difference FIR filter) to boost the high frequencies to counter the roll-off of natural speech.

By inspecting both narrow-band and wide-band spectrograms, we can learn the filter's magnitude response and whether the source is voiced or not. Nonetheless it is very difficult to separate source and filter due to nonstationarity of the speech signal, spectral leakage, and the fact that only the filter's magnitude response can be known at the signal's harmonics.

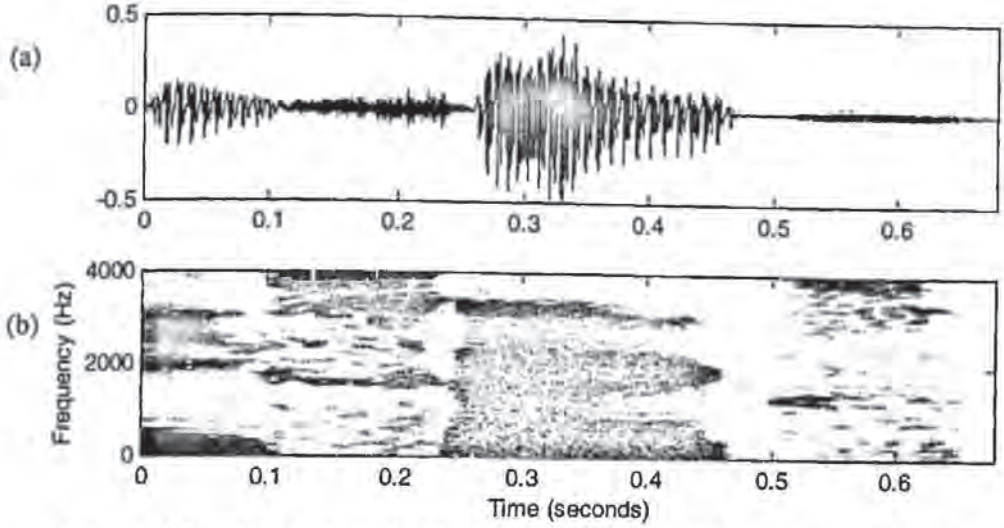


Figure 6.7 Waveform (a) with its corresponding narrowband spectrogram (b). Darker areas mean higher energy for that time and frequency. The harmonics can be seen as horizontal lines spaced by fundamental frequency. The corresponding wideband spectrogram can be seen in Figure 6.2.

6.1.2. Pitch-Synchronous Analysis

In the previous discussion, we assumed that the window length is fixed, and we saw the tradeoffs between a window that contained several pitch periods (narrow-band spectrograms) and a window that contained less than a pitch period (wide-band spectrograms). One possibility is to use a rectangular window whose length is exactly one pitch period; this is called *pitch-synchronous* analysis. To reduce spectral leakage a tapering window, such as Hamming or Hanning, can be used, with the window covering exactly two pitch periods. This latter option provides a very good compromise between time and frequency resolution. In this representation, no stripes can be seen in either time or frequency. The difficulty in computing pitch synchronous analysis is that, of course, we need to know the local pitch period, which, as we see in Section 6.7, is not an easy task.

6.2. ACOUSTICAL MODEL OF SPEECH PRODUCTION

Speech is a sound wave created by vibration that is propagated in the air. Acoustic theory analyzes the laws of physics that govern the propagation of sound in the vocal tract. Such a theory should consider three-dimensional wave propagation, the variation of the vocal tract shape with time, losses due to heat conduction and viscous friction at the vocal tract walls, softness of the tract walls, radiation of sound at the lips, nasal coupling, and excitation of sound. While a detailed model that considers all of the above is not yet available, some

models provide a good approximation in practice, as well as a good understanding of the physics involved.

6.2.1. Glottal Excitation

As discussed in Chapter 2, the vocal cords constrict the path from the lungs to the vocal tract. This is illustrated in Figure 6.8. As lung pressure is increased, air flows out of the lungs and through the opening between the vocal cords (*glottis*). At one point the vocal cords are together, thereby blocking the airflow, which builds up *pressure* behind them. Eventually the pressure reaches a level sufficient to force the vocal cords to open and thus allow air to flow through the glottis. Then, the pressure in the glottis falls and, if the tension in the vocal cords is properly adjusted, the reduced pressure allows the cords to come together, and the cycle is repeated. This condition of sustained oscillation occurs for voiced sounds. The *closed-phase* of the oscillation takes place when the glottis is closed and the *volume velocity* is zero. The *open-phase* is characterized by a non-zero volume velocity, in which the lungs and the vocal tract are coupled.

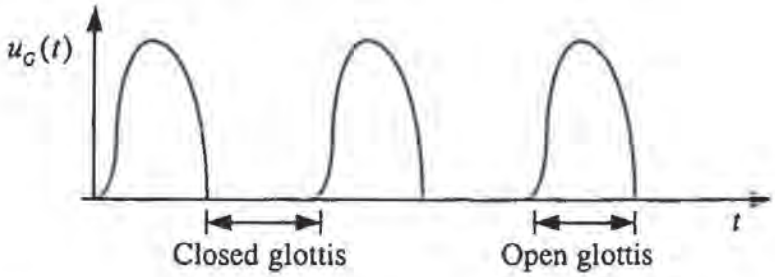


Figure 6.8 Glottal excitation: volume velocity is zero during the closed-phase, during which the vocal cords are closed.

Rosenberg's glottal model [39] defines the shape of the glottal volume velocity with the *open quotient*, or duty cycle, as the ratio of pulse duration to pitch period, and the *speed quotient* as the ratio of the rising to falling pulse durations.

6.2.2. Lossless Tube Concatenation

A widely used model for speech production is based on the assumption that the vocal tract can be represented as a concatenation of lossless tubes, as shown in Figure 6.9. The constant cross-sectional areas $\{A_k\}$ of the tubes approximate the area function $A(x)$ of the vocal tract. If a large number of tubes of short length are used, we reasonably expect the frequency response of the concatenated tubes to be close to those of a tube with continuously varying area function.

For frequencies corresponding to wavelengths that are long compared to the dimensions of the vocal tract, it is reasonable to assume plane wave propagation along the axis of

the tubes. If in addition we assume that there are no losses due to viscosity or thermal conduction, and that the area A does not change over time, the sound waves in the tube satisfy the following pair of differential equations:

$$-\frac{\partial p(x,t)}{\partial x} = \frac{\rho}{A}\frac{\partial u(x,t)}{\partial t}$$
$$-\frac{\partial u(x,t)}{\partial x} = \frac{A}{\rho c^2}\frac{\partial p(x,t)}{\partial t} \tag{6.9}$$

where $p(x,t)$ is the sound pressure in the tube at position x and time t, $u(x,t)$ is the volume velocity flow in the tube at position x and time t, ρ is the density of air in the tube, c is the velocity of sound, and A is the cross-sectional area of the tube.

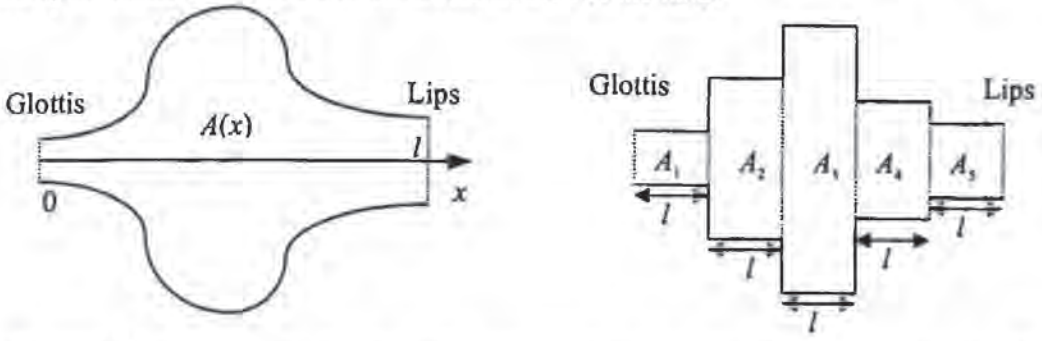


Figure 6.9 Approximation of a tube with continuously varying area $A(x)$ as a concatenation of 5 lossless acoustic tubes.

Since Eqs. (6.9) are linear, the pressure and volume velocity in the k^{th} tube are related by

$$u_k(x,t) = u_k^+(t - x/c) - u_k^-(t + x/c)$$
$$p_k(x,t) = \frac{\rho c}{A_k}\left[u_k^+(t - x/c) + u_k^-(t + x/c)\right] \tag{6.10}$$

where $u_k^+(t - x/c)$ and $u_k^-(t - x/c)$ are the traveling waves in the positive and negative directions respectively and x is the distance measured from the left-hand end of tube k^{th}: $0 \le x \le l$. The reader can prove that this is indeed the solution by substituting Eq. (6.10) into (6.9).

When there is a junction between two tubes, as in Figure 6.10, part of the wave is reflected at the junction, as measured by r_k, the reflection coefficient

$$r_k = \frac{A_{k+1} - A_k}{A_{k+1} + A_k} \tag{6.11}$$

so that the larger the difference between the areas the more energy is reflected. The proof [9] is beyond the scope of this book. Since A_k and A_{k+1} are positive, it is easy to show that r_k satisfies the condition

$$-1 \le r_k \le 1 \tag{6.12}$$

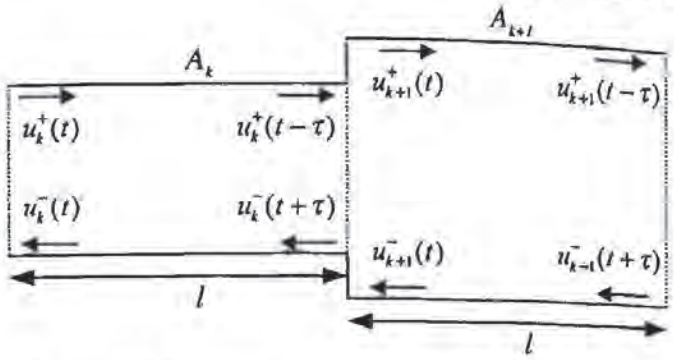


Figure 6.10 Junction between two lossless tubes.

A relationship between the z-transforms of the volume velocity at the glottis $u_G[n]$ and the lips $u_L[n]$ for a concatenation of N lossless tubes can be derived [9] using a discrete-time version of Eq. (6.10) and taking into account boundary conditions for every junction:

$$V(z)=\frac{U_L(z)}{U_G(z)}=\frac{0.5z^{-N/2}\left(1+r_G\right)\prod_{k=1}^{N}\left(1+r_k\right)}{\begin{bmatrix}1 & -r_G\end{bmatrix}\left(\prod_{k=1}^{N}\begin{bmatrix}1 & -r_k\\ -r_k z^{-1} & z^{-1}\end{bmatrix}\right)\begin{bmatrix}1\\ 0\end{bmatrix}} \tag{6.13}$$

where r_G is the reflection coefficient at the glottis and $r_N=r_L$ is the reflection coefficient at the lips. Equation (6.11) is still valid for the glottis and lips, where $A_0=\rho c/Z_G$ is the equivalent area at the glottis and $A_{N+1}=\rho c/Z_L$ the equivalent area at the lips. Z_G and Z_L are the equivalent impedances at the glottis and lips, respectively. Such impedances relate the volume velocity and pressure, for the lips the expression is

$$U_L(z)=P_L(z)/Z_L \tag{6.14}$$

In general, the concatenation of N lossless tubes results in an N-pole system as shown in Eq. (6.13). For a concatenation of N tubes, there are at most $N/2$ complex conjugate poles, or resonances or formants. These resonances occur when a given frequency gets *trapped* in the vocal tract because it is reflected back at the lips and then again back at the glottis.

Since each tube has length l and there are N of them, the total length is $L=lN$. The propagation delay in each tube $\tau=l/c$, and the sampling period is $T=2\tau$, the round trip in a tube. We can find a relationship between the number of tubes N and the sampling frequency $F_s=1/T$:

$$N=\frac{2LF_s}{c} \tag{6.15}$$

For example, for F_s = 8000 kHz, c = 34000 cm/s, and L = 17 cm, the average length of a male adult vocal tract, we obtain $N = 8$, or alternatively 4 formants. Experimentally, the vocal tract transfer function has been observed to have approximately 1 formant per kilohertz. Shorter vocal tract lengths (females or children) have fewer resonances per kilohertz and vice versa.

The pressure at the lips has been found to approximate the derivative of volume velocity, particularly at low frequencies. Thus, $Z_L(z)$ can be approximated by

$$Z_L(z) \approx R_0(1 - z^{-1}) \tag{6.16}$$

which is 0 for low frequencies and reaches R_0 asymptotically. This dependency upon frequency results in a reflection coefficient that is also a function of frequency. For low frequencies, $r_L = 1$, and no loss occurs. At higher frequencies, loss by radiation translates into widening of formant bandwidths.

Similarly, the glottal impedance is also a function of frequency in practice. At high frequencies, Z_G is large and $r_G \approx 1$ so that all the energy is transmitted. For low frequencies, $r_G < 1$, whose main effect is an increase of bandwidth for the lower formants.

Moreover, energy is lost as a result of vibration of the tube walls, which is more pronounced at low frequencies. Energy is also lost, to a lesser extent, as a result of viscous friction between the air and the walls of the tube, particularly at frequencies above 3 kHz. The yielding walls tend to raise the resonance frequencies while the viscous and thermal losses tend to lower them. The net effect in the transfer function is a broadening of the resonances' bandwidths.

Despite thermal losses, yielding walls in the vocal tract, and the fact that both r_L and r_G are functions of frequency, the all-pole model of Eq. (6.13) for $V(z)$ has been found to be a good approximation in practice [13]. In Figure 6.11 we show the measured area function of a vowel and its corresponding frequency response obtained using the approximation as a concatenation of 10 lossless tubes with a constant r_L. The measured formants and corresponding bandwidths match quite well with this model despite all the approximations made. Thus, this concatenation of lossless tubes model represents reasonably well the acoustics inside the vocal tract. Inspired by the above results, we describe in Section 6.3 "Linear Predictive Coding," an all-pole model for speech.

In the production of the nasal consonants, the velum is lowered to trap the nasal tract to the pharynx, whereas a complete closure is formed in the oral tract (/m/ at the lips, /n/ just back of the teeth and /ng/ just forward of the velum itself. This configuration is shown in Figure 6.12, which shows two branches, one of them completely closed. For nasals, the radiation occurs primarily at the nostrils. The set of resonances is determined by the shape and length of the three tubes. At certain frequencies, the wave reflected in the closure cancels the wave at the pharynx, preventing energy from appearing at nostrils. The result is that for nasal sounds, the vocal tract transfer function $V(z)$ has anti-resonances (zeros) in addition to resonances. It has also been observed that nasal resonances have broader bandwidths than non-nasal voiced sounds, due to the greater viscous friction and thermal loss because of the large surface area of the nasal cavity.

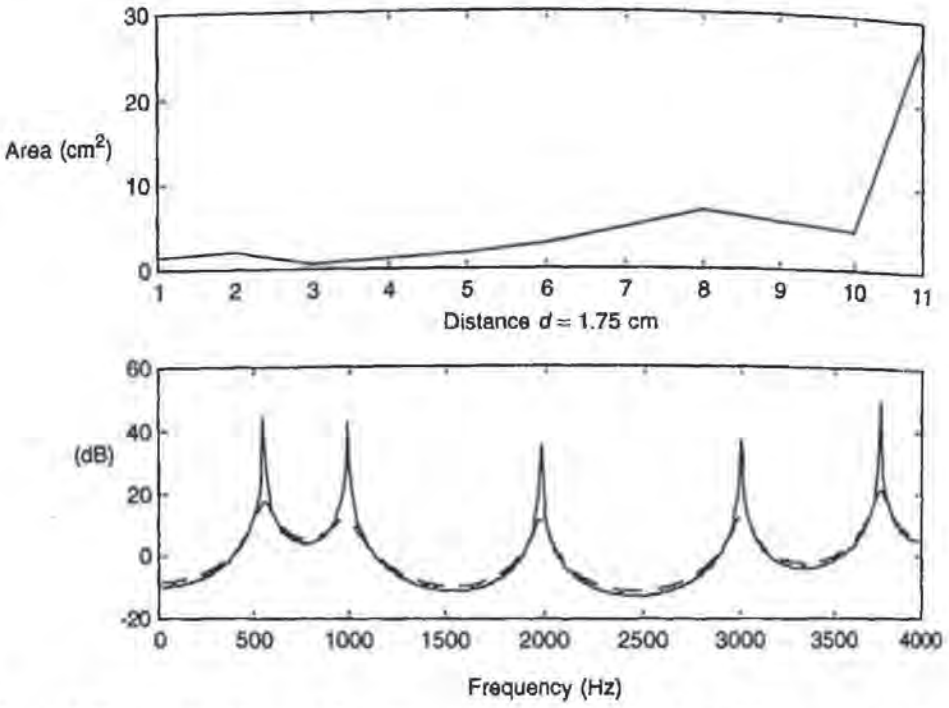


Figure 6.11 Area function and frequency response for vowel /a/ and its approximation as a concatenation of 10 lossless tubes. A reflection coefficient at the load of $k = 0.72$ (dotted line) is displayed. For comparison, the case of $k = 1.0$ (solid line) is also shown.

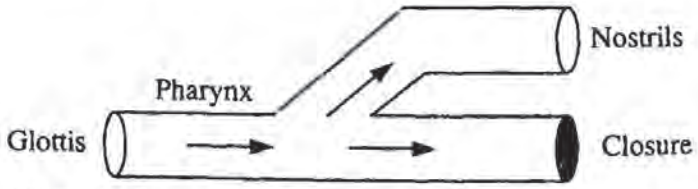


Figure 6.12 Coupling of the nasal cavity with the oral cavity.

6.2.3. Source-Filter Models of Speech Production

As shown in Chapter 10, speech signals are captured by microphones that respond to changes in air pressure. Thus, it is of interest to compute the pressure at the lips $P_L(z)$, which can be obtained as

$$P_L(z) = U_L(z)Z_L(z) = U_G(z)V(z)Z_L(z) \tag{6.17}$$

For voiced sounds we can model $u_G[n]$ as an impulse train convolved with $g[n]$, the glottal pulse (see Figure 6.13). Since $g[n]$ is of finite length, its z-transform is an all-zero system.

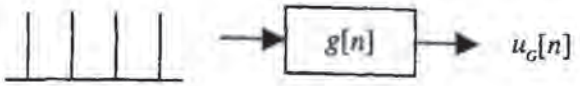


Figure 6.13 Model of the glottal excitation for voiced sounds.

The complete model for both voiced and unvoiced sounds is shown in Figure 6.14. We have modeled $u_G[n]$ in unvoiced sounds as random noise.

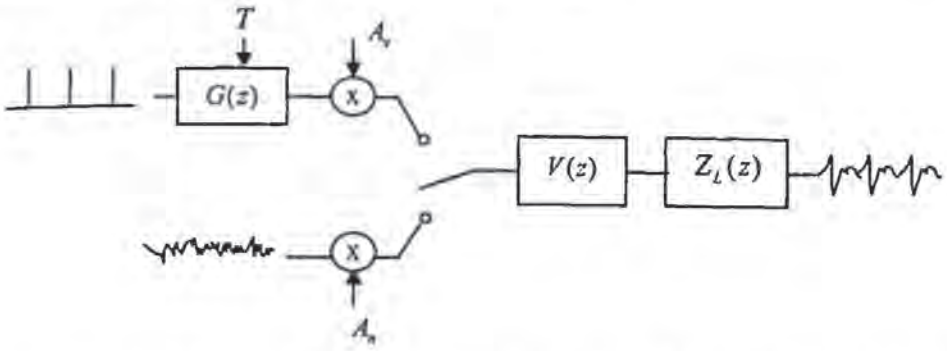


Figure 6.14 General discrete-time model of speech production. The excitation can be either an impulse train with period T and amplitude A_v driving a filter $G(z)$ or random noise with amplitude A_n.

We can simplify the model in Figure 6.14 by grouping $G(z)$, $V(z)$, and $Z_L(z)$ into $H(z)$ for voiced sounds, and $V(z)$ and $Z_L(z)$ into $H(z)$ for unvoiced sounds. The simplified model is shown in Figure 6.15, where we make explicit the fact that the filter changes over time.

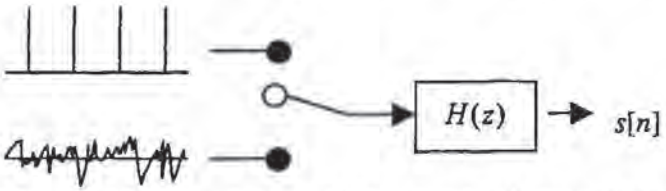


Figure 6.15 Source-filter model for voiced and unvoiced speech.

This model is a decent approximation, but fails on voiced fricatives, since those sounds contain both a periodic component and an aspirated component. In this case, a *mixed excitation* model can be applied, using for voiced sounds a sum of both an impulse train and colored noise (Figure 6.16).

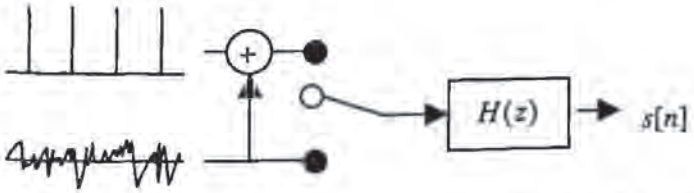


Figure 6.16 A mixed excitation source-filter model of speech.

The model in Figure 6.15 is appealing because the source is white (has a flat spectrum) and all the *coloring* is in the filter. Other source-filter decompositions attempt to model the source as the signal at the glottis, in which the source is definitely not white. Since $G(z)$, $Z_L(z)$ contain zeros, and $V(z)$ can also contain zeros for nasals, $H(z)$ is no longer all-pole. However, recall in Chapter 5, we state that the z-transform of $x[n]=a^n u[n]$ is

$$X(z)=\sum_{n=0}^{\infty} a^n z^{-n}=\frac{1}{1-az^{-1}} \quad \text{for} \quad |a|<|z| \tag{6.18}$$

so that by inverting Eq. (6.18) we see that a zero can be expressed with infinite poles. This is the reason why all-pole models are still reasonable approximations as long as a large enough number of poles is used. Fant [12] showed that on the average the speech spectrum contains one pole per kHz. Setting the number of poles p to F_s+2, where F_s is the sampling frequency expressed in kHz, has been found to work well in practice.

6.3. LINEAR PREDICTIVE CODING

A very powerful method for speech analysis is based on *linear predictive coding* (LPC) [4, 7, 19, 24, 27], also known as LPC analysis or *auto-regressive* (AR) modeling. This method is widely used because it is fast and simple, yet an effective way of estimating the main parameters of speech signals.

As shown in Section 6.2, an all-pole filter with a sufficient number of poles is a good approximation for speech signals. Thus, we could model the filter $H(z)$ in Figure 6.15 as

$$H(z)=\frac{X(z)}{E(z)}=\frac{1}{1-\sum_{k=1}^{p} a_k z^{-k}}=\frac{1}{A(z)} \tag{6.19}$$

where p is the order of the LPC analysis. The *inverse filter* $A(z)$ is defined as

$$A(z)=1-\sum_{k=1}^{p} a_k z^{-k} \tag{6.20}$$

Taking inverse z-transforms in Eq. (6.19) results in

$$x[n]=\sum_{k=1}^{p} a_k x[n-k]+e[n] \tag{6.21}$$

Linear predictive coding gets its name from the fact that it predicts the current sample as a linear combination of its past p samples:

$$\tilde{x}[n]=\sum_{k=1}^{p} a_k x[n-k] \tag{6.22}$$

The prediction error when using this approximation is

$$e[n] = x[n] - \tilde{x}[n] = x[n] - \sum_{k=1}^{p} a_k x[n-k] \tag{6.23}$$

6.3.1. The Orthogonality Principle

To estimate the predictor coefficients from a set of speech samples, we use the short-term analysis technique. Let's define $x_m[n]$ as a segment of speech selected in the vicinity of sample m:

$$x_m[n] = x[m+n] \tag{6.24}$$

We define the short-term prediction error for that segment as

$$E_m = \sum_n e_m^2[n] = \sum_n \left(x_m[n] - \tilde{x}_m[n]\right)^2 = \sum_n \left(x_m[n] - \sum_{j=1}^{p} a_j x_m[n-j] \right)^2 \tag{6.25}$$

In the absence of knowledge about the probability distribution of a_i, a reasonable estimation criterion is minimum mean squared error, introduced in Chapter 4. Thus, given a signal $x_m[n]$, we estimate its corresponding LPC coefficients as those that minimize the total prediction error E_m. Taking the derivative of Eq. (6.25) with respect to a_i and equating to 0, we obtain:

$$< \mathbf{e}_m, \mathbf{x}_m^i >= \sum_n e_m[n] x_m[n-i] = 0 \qquad 1 \le i \le p \tag{6.26}$$

where we have defined $\mathbf{e}_m$ and $\mathbf{x}_m^i$ as vectors of samples, and their inner product has to be 0. This condition, known as *orthogonality principle*, says that the predictor coefficients that minimize the prediction error are such that the error must be orthogonal to the past vectors, and is seen in Figure 6.17.

Equation (6.26) can be expressed as a set of p linear equations

$$\sum_n x_m[n-i] x_m[n] = \sum_{j=1}^{p} a_j \sum_n x_m[n-i] x_m[n-j] \qquad i = 1, 2, \ldots, p \tag{6.27}$$

For convenience, we can define the correlation coefficients as

$$\phi_m[i,j] = \sum_n x_m[n-i] x_m[n-j] \tag{6.28}$$

so that Eqs. (6.27) and (6.28) can be combined to obtain the so-called *Yule-Walker* equations:

$$\sum_{j=1}^{p} a_j \phi_m[i,j] = \phi_m[i,0] \qquad i = 1,2,\ldots,p \tag{6.29}$$

Solution of the set of p linear equations results in the p LPC coefficients that minimize the prediction error. With a_i satisfying Eq. (6.29), the total prediction error in Eq. (6.25) takes on the following value:

$$E_m = \sum_n x_m^2[n] - \sum_{j=1}^{p} a_j \sum_n x_m[n] x_m[n-j] = \phi[0,0] - \sum_{j=1}^{p} a_j \phi[0,j] \tag{6.30}$$

It is convenient to define a normalized prediction error $u[n]$ with unity energy

$$\sum_n u_m^2[n] = 1 \tag{6.31}$$

and a gain G, such that

$$e_m[n] = G u_m[n] \tag{6.32}$$

The gain G can be computed from the short-term prediction error

$$E_m = \sum_n e_m^2[n] = G^2 \sum_n u_m^2[n] = G^2 \tag{6.33}$$

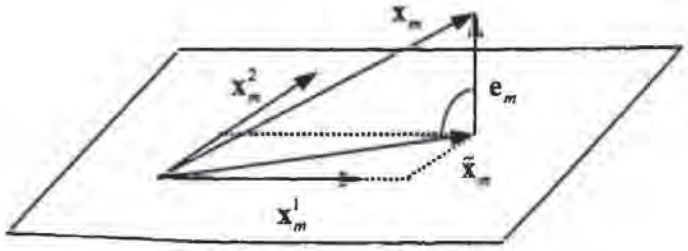


Figure 6.17 The orthogonality principle. The prediction error is orthogonal to the past samples.

6.3.2. Solution of the LPC Equations

The solution of the Yule-Walker equations in Eq. (6.29) can be achieved with any standard matrix inversion package. Because of the special form of the matrix here, some efficient solutions are possible, as described below. Also, each solution offers a different insight so we present three different algorithms: the covariance method, the autocorrelation method, and the lattice method.

6.3.2.1. Covariance Method

The covariance method [4] is derived by defining directly the interval over which the summation in Eq. (6.28) takes place:

$$E_m = \sum_{n=0}^{N-1} e_m^2[n] \tag{6.34}$$

so that $\phi_m[i,j]$ in Eq. (6.28) becomes

$$\phi_m[i,j] = \sum_{n=0}^{N-1} x_m[n-i]x_m[n-j] = \sum_{n=-i}^{N-1-j} x_m[n]x_m[n+i-j] = \phi_m[j,i] \tag{6.35}$$

and Eq. (6.29) becomes

$$\begin{pmatrix} \phi_m[1,1] & \phi_m[1,2] & \phi_m[1,3] & \cdots & \phi_m[1,p] \\ \phi_m[2,1] & \phi_m[2,2] & \phi_m[2,3] & \cdots & \phi_m[2,p] \\ \phi_m[3,1] & \phi_m[3,2] & \phi_m[3,3] & \cdots & \phi_m[3,p] \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \phi_m[p,1] & \phi_m[p,2] & \phi_m[p,3] & \cdots & \phi_m[p,p] \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ \cdots \\ a_p \end{pmatrix} = \begin{pmatrix} \phi_m[1,0] \\ \phi_m[2,0] \\ \phi_m[3,0] \\ \cdots \\ \phi_m[p,0] \end{pmatrix} \tag{6.36}$$

which can be expressed as the following matrix equation

$$\Phi \mathbf{a} = \psi \tag{6.37}$$

where the matrix Φ in Eq. (6.37) is symmetric and *positive definite*, for which efficient methods are available, such as the Cholesky decomposition. For this method, also called the squared root method, the matrix Φ is expressed as

$$\Phi = \mathbf{V}\mathbf{D}\mathbf{V}^t \tag{6.38}$$

where $\mathbf{V}$ is a lower triangular matrix (whose main diagonal elements are 1's), and $\mathbf{D}$ is a diagonal matrix. So each element of Φ can be expressed as

$$\phi[i,j] = \sum_{k=1}^{j} V_{ik} d_k V_{jk} \qquad 1 \le j < i \tag{6.39}$$

or alternatively

$$V_{ij} d_j = \phi[i,j] - \sum_{k=1}^{j-1} V_{ik} d_k V_{jk} \qquad 1 \le j < i \tag{6.40}$$

and for the diagonal elements

$$\phi[i,i] = \sum_{k=1}^{i} V_{ik} d_k V_{ik} \tag{6.41}$$

or alternatively

$$d_i = \phi[i,i] - \sum_{k=1}^{i-1} V_{ik}^2 d_k \,, \qquad i \geq 2 \tag{6.42}$$

with

$$d_1 = \phi[1,1] \tag{6.43}$$

The Cholesky decomposition starts with Eq. (6.43) then alternates between Eqs. (6.40) and (6.42). Once the matrices $\mathbf{V}$ and $\mathbf{D}$ have been determined, the LPC coefficients are solved in a two-step process. The combination of Eqs. (6.37) and (6.38) can be expressed as

$$\mathbf{VY} = \psi \tag{6.44}$$

with

$$\mathbf{Y} = \mathbf{DV'a} \tag{6.45}$$

or alternatively

$$\mathbf{V'a} = \mathbf{D^{-1}Y} \tag{6.46}$$

Therefore, given matrix $\mathbf{V}$ and Eq. (6.44), $\mathbf{Y}$ can be solved recursively as

$$Y_i = \psi_i - \sum_{j=1}^{i-1} V_{ij} Y_j \,, \qquad 2 \leq i \leq p \tag{6.47}$$

with the initial condition

$$Y_1 = \psi_1 \tag{6.48}$$

Having determined $\mathbf{Y}$, Eq. (6.46) can be solved recursively in a similar way

$$a_i = Y_i / d_i - \sum_{j=i+1}^{p} V_{ji} a_j \,, \qquad 1 \leq i < p \tag{6.49}$$

with the initial condition

$$a_p = Y_p / d_p \tag{6.50}$$

where the index i in Eq. (6.49) proceeds backwards.

The term covariance analysis is somewhat of a misnomer, since we know from Chapter 5 that the covariance of a signal is the correlation of that signal with its mean removed. It was so called because the matrix in Eq. (6.36) has the properties of a covariance matrix, though this algorithm is more like a cross-correlation.

6.3.2.2. Autocorrelation Method

The summation in Eq. (6.28) had no specific range. In the autocorrelation method [24, 27], we assume that $x_m[n]$ is 0 outside the interval $0 \le n < N$:

$$x_m[n] = x[m+n]w[n] \tag{6.51}$$

with $w[n]$ being a window (such as a Hamming window) which is 0 outside the interval $0 \le n < N$. With this assumption, the corresponding prediction error $e_m[n]$ is non-zero over the interval $0 \le n < N+p$, and, therefore, the total prediction error takes on the value

$$E_m = \sum_{n=0}^{N+p-1} e_m^2[n] \tag{6.52}$$

With this range, Eq. (6.28) can be expressed as

$$\phi_m[i,j] = \sum_{n=0}^{N+p-1} x_m[n-i]x_m[n-j] = \sum_{n=0}^{N-1-(i-j)} x_m[n]x_m[n+i-j] \tag{6.53}$$

or alternatively

$$\phi_m[i,j] = R_m[i-j] \tag{6.54}$$

with $R_m[k]$ being the autocorrelation sequence of $x_m[n]$:

$$R_m[k] = \sum_{n=0}^{N-1-k} x_m[n]x_m[n+k] \tag{6.55}$$

Combining Eqs. (6.54) and (6.29), we obtain

$$\sum_{j=1}^{p} a_j R_m[|i-j|] = R_m[i] \tag{6.56}$$

which corresponds to the following matrix equation

$$\begin{pmatrix} R_m[0] & R_m[1] & R_m[2] & \cdots & R_m[p-1] \\ R_m[1] & R_m[0] & R_m[1] & \cdots & R_m[p-2] \\ R_m[2] & R_m[1] & R_m[0] & \cdots & R_m[p-3] \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ R_m[p-1] & R_m[p-2] & R_m[p-3] & \cdots & R_m[0] \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ \cdots \\ a_p \end{pmatrix} = \begin{pmatrix} R_m[1] \\ R_m[2] \\ R_m[3] \\ \cdots \\ R_m[p] \end{pmatrix} \tag{6.57}$$

The matrix in Eq. (6.57) is symmetric and all the elements in its diagonals are identical. Such matrices are called *Toeplitz*. Durbin's recursion exploits this fact resulting in a very efficient algorithm (for convenience, we omit the subscript m of the autocorrelation function), whose proof is beyond the scope of this book:

1. Initialization

$$E^0 = R[0] \tag{6.58}$$

2. Iteration. For $i = 1, \cdots, p$ do the following recursion:

$$k_i = \left(R[i] - \sum_{j=1}^{i-1} a_j^{i-1} R[i-j] \right) / E^{i-1} \tag{6.59}$$

$$a_i^i = k_i \tag{6.60}$$

$$a_j^i = a_j^{i-1} - k_i a_{i-j}^{i-1}, \qquad 1 \le j < i \tag{6.61}$$

$$E^i = (1 - k_i^2) E^{i-1} \tag{6.62}$$

3. Final solution:

$$a_j = a_j^p \qquad 1 \le j \le p \tag{6.63}$$

where the coefficients k_i, called *reflection coefficients*, are bounded between –1 and 1 (see Section 6.3.2.1.3). In the process of computing the predictor coefficients of order p, the recursion finds the solution of the predictor coefficients for all orders less than p.

Replacing $R[j]$ by the normalized autocorrelation coefficients $r[j]$, defined as

$$r[j] = R[j] / R[0] \tag{6.64}$$

results in identical LPC coefficients, and the recursion is more robust to problems with arithmetic precision. Likewise, the normalized prediction error at iteration i is defined by dividing Eq. (6.30) by $R[0]$, which, using Eq.(6.54), results in

$$V^i = \frac{E^i}{R[0]} = 1 - \sum_{j=1}^{i} a_j r[j] \tag{6.65}$$

The normalized prediction error is, using Eqs. (6.62) and (6.65),

$$V^p = \prod_{i=1}^{p} (1 - k_i^2) \tag{6.66}$$

6.3.2.3. Lattice Formulation

In this section we derive the lattice formulation [7, 19], an equivalent algorithm to the Levinson Durbin recursion, which has some precision benefits. It is advantageous to define the *forward prediction error* obtained at stage i of the Levinson Durbin procedure as

$$e^i[n] = x[n] - \sum_{k=1}^{i} a_k^i x[n-k] \tag{6.67}$$

whose z-transform is given by

$$E^i(z) = A^i(z)X(z) \tag{6.68}$$

with $A^i(z)$ being defined by

$$A^i(z) = 1 - \sum_{k=1}^{i} a_k^i z^{-k} \tag{6.69}$$

which, combined with Eq. (6.61), results in the following recursion:

$$A^i(z) = A^{i-1}(z) - k_i z^{-i} A^{i-1}(z^{-1}) \tag{6.70}$$

Similarly, we can define the so-called *backward prediction error* as

$$b^i[n] = x[n-i] - \sum_{k=1}^{i} a_k^i x[n+k-i] \tag{6.71}$$

whose z-transform is

$$B^i(z) = z^{-i} A^i(z^{-1})X(z) \tag{6.72}$$

Now combining Eqs. (6.68), (6.70), and (6.72), we obtain

$$E^i(z) = A^{i-1}(z)X(z) - k_i z^{-i} A^{i-1}(z^{-1})X(z) = E^{i-1}(z) - k_i z^{-i} B^{i-1}(z) \tag{6.73}$$

whose inverse z-transform is given by

$$e^i[n] = e^{i-1}[n] - k_i b^{i-1}[n-1] \tag{6.74}$$

Also, substituting Eq. (6.70) into (6.72) and using Eq. (6.68), we obtain

$$B^i(z) = z^{-1} B^{i-1}(z) - k_i E^{i-1}(z) \tag{6.75}$$

whose inverse z-transform is given by

$$b^i[n] = b^{i-1}[n-1] - k_i e^{i-1}[n] \tag{6.76}$$

Equations (6.74) and (6.76) define the forward and backward prediction error sequences for an i^{th}-order predictor in terms of the corresponding forward and backward prediction errors of an $(i-1)^{th}$-order predictor. We initialize the recursive algorithm by noting that the 0^{th}-order predictor is equivalent to using no predictor at all; thus

$$e^0[n] = b^0[n] = x[n] \tag{6.77}$$

and the final prediction error is $e[n] = e^p[n]$.

A block diagram of the lattice method is given in Figure 6.18, which resembles a lattice, hence its name.

While the computation of the k_i coefficients can be done through the Levinson Durbin recursion of Eqs. (6.59) through (6.62), it can be shown that an equivalent calculation can be found as a function of the forward and backward prediction errors. To do so we minimize the sum of the forward prediction errors

$$E^i = \sum_{n=0}^{N-1} \left(e^i[n]\right)^2 \tag{6.78}$$

by substituting Eq. (6.74) in (6.78), taking the derivative with respect to k_i, and equating to 0:

$$k_i = \frac{\sum_{n=0}^{N-1} e^{i-1}[n] b^{i-1}[n-1]}{\sum_{n=0}^{N-1} \left(b^{i-1}[n-1]\right)^2} \tag{6.79}$$

Using Eqs. (6.67) and (6.71), it can be shown that

$$\sum_{n=0}^{N-1} \left(e^{i-1}[n]\right)^2 = \sum_{n=0}^{N-1} \left(b^{i-1}[n-1]\right)^2 \tag{6.80}$$

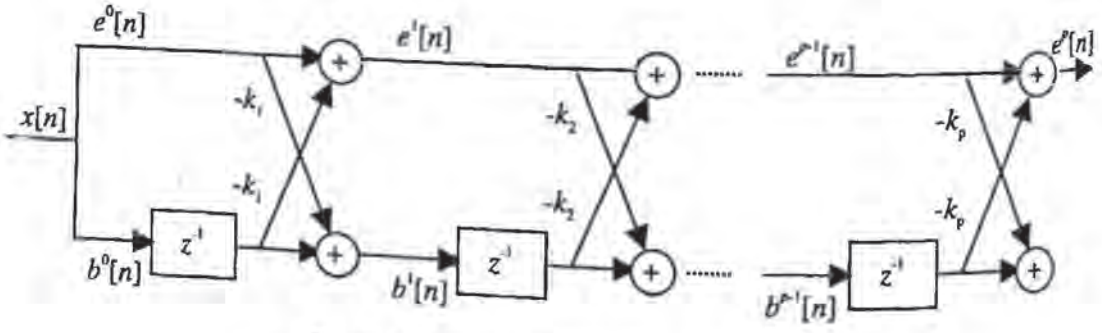


Figure 6.18 Block diagram of the lattice filter.

since minimization of both yields identical Yule-Walker equations. Thus Eq. (6.79) can be alternatively expressed as

$$k_l = \frac{\sum_{n=0}^{N-1} e^{l-1}[n]b^{l-1}[n-1]}{\sqrt{\sum_{n=0}^{N-1}\left(e^{l-1}[n]\right)^2 \sum_{n=0}^{N-1}\left(b^{l-1}[n-1]\right)^2}} = \frac{<\mathbf{e}^{l-1}, \mathbf{b}^{l-1}>}{\left|\mathbf{e}^{l-1}\right|\left|\mathbf{b}^{l-1}\right|} \tag{6.81}$$

where we have defined the vectors $\mathbf{e}^l = \left(e^l[0]\cdots e^l[N-1]\right)$ and $\mathbf{b}^l = \left(b^l[0]\cdots b^l[N-1]\right)$. The inner product of two vectors $\mathbf{x}$ and $\mathbf{y}$ is defined as

$$<\mathbf{x},\mathbf{y}>=\sum_{n=0}^{N-1} x[n]y[n] \tag{6.82}$$

and its norm as

$$|\mathbf{x}|^2 =<\mathbf{x},\mathbf{x}>=\sum_{n=0}^{N-1} x^2[n] \tag{6.83}$$

Equation (6.81) has the form of a normalized cross-correlation function, and, therefore, the reflection coefficients are also called *partial correlation coefficients* (PARCOR). As with any normalized cross-correlation function, the k_l coefficients are bounded by

$$-1 \le k_l \le 1 \tag{6.84}$$

This is a necessary and sufficient condition for all the roots of the polynomial $A(z)$ to be inside the unit circle, therefore guaranteeing a stable filter. This condition can be checked to avoid numerical imprecision by stopping the recursion if the condition is not met. The inverse lattice filter can be seen in Figure 6.19, which resembles the lossless tube model. This is why the k_l are also called *reflection coefficients*.

Lattice filters are often used in fixed-point implementation, because lack of precision doesn't result in unstable filters. Any error that may take place – for example due to quantization – is generally not be sufficient to cause k_l to fall outside the range in Eq. (6.84). If, owing to round-off error, the reflection coefficient falls outside the range, the lattice filter can be ended at the previous step.

More importantly, linearly varying coefficients can be implemented in this fashion. While, typically, the reflection coefficients are constant during the analysis frame, we can implement a linear interpolation of the reflection coefficients to obtain the error signal. If the coefficients of both frames are in the range in Eq. (6.84), the linearly interpolated reflection coefficients also have that property, and thus the filter is stable. This is a property that the predictor coefficients don't have.

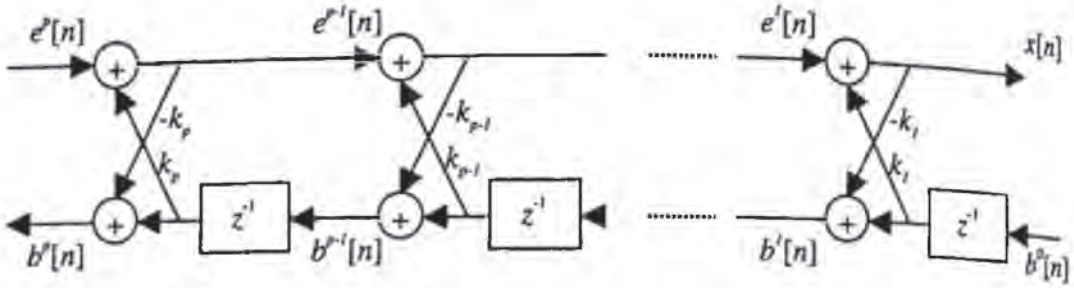


Figure 6.19 Inverse lattice filter used to generate the speech signal, given its residual.

6.3.3. Spectral Analysis via LPC

Let's now analyze the frequency-domain behavior of the LPC analysis by evaluating

$$H(e^{j\omega}) = \frac{G}{1-\sum_{k=1}^{p} a_k e^{-j\omega k}} = \frac{G}{A(e^{j\omega})} \tag{6.85}$$

which is an *all-pole* or IIR filter. If we plot $H(e^{j\omega})$, we expect to see peaks at the roots of the denominator. Figure 6.20 shows the 14-order LPC spectrum of the vowel of Figure 6.3 (d).

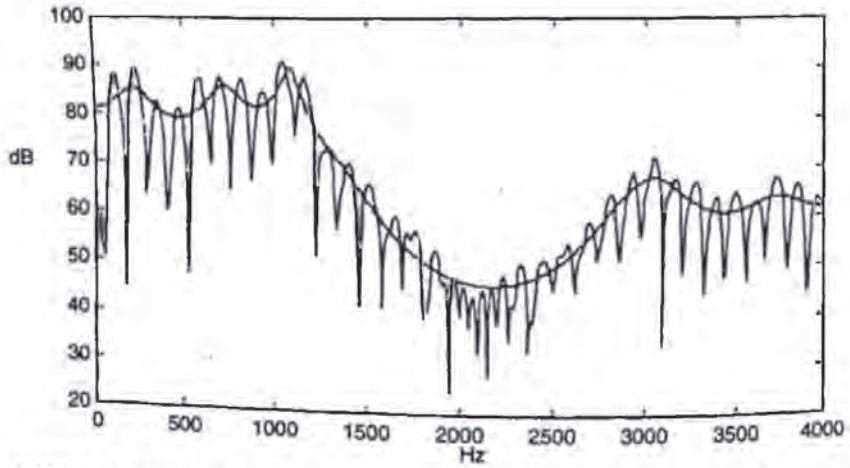


Figure 6.20 LPC spectrum of the */ah/* phoneme in the word *lives* of Figure 6.3. Used here are a 30-ms Hamming window and the autocorrelation method with $p = 14$. The short-time spectrum is also shown.

For the autocorrelation method, the squared error of Eq. (6.52) can be expressed, using Eq. (6.85) and Parseval's theorem, as

$$E_m = \frac{G^2}{2\pi}\int_{-\pi}^{\pi}\frac{|X_m(e^{j\omega})|^2}{|H(e^{j\omega})|^2}d\omega \tag{6.86}$$

Since the integrand in Eq. (6.86) is positive, minimizing E_m is equivalent to minimizing the ratio of the energy spectrum of the speech segment $|X_m(e^{j\omega})|^2$ to the magnitude squared of the frequency response of the linear system $|H(e^{j\omega})|^2$. The LPC spectrum matches more closely the peaks than the valleys (see Figure 6.20), because the regions where $|X_m(e^{j\omega})| > |H(e^{j\omega})|$ contribute more to the error than those where $|H(e^{j\omega})| > |X_m(e^{j\omega})|$.

Even nasals, which have zeros in addition to poles, can be represented with an infinite number of poles. In practice, if p is large enough we can approximate the signal spectrum with arbitrarily small error. Figure 6.21 shows different fits for different values of p. The higher p, the more details of the spectrum are preserved.

The prediction order is not known for arbitrary speech, so we need to set it to balance spectral detail with estimation errors.

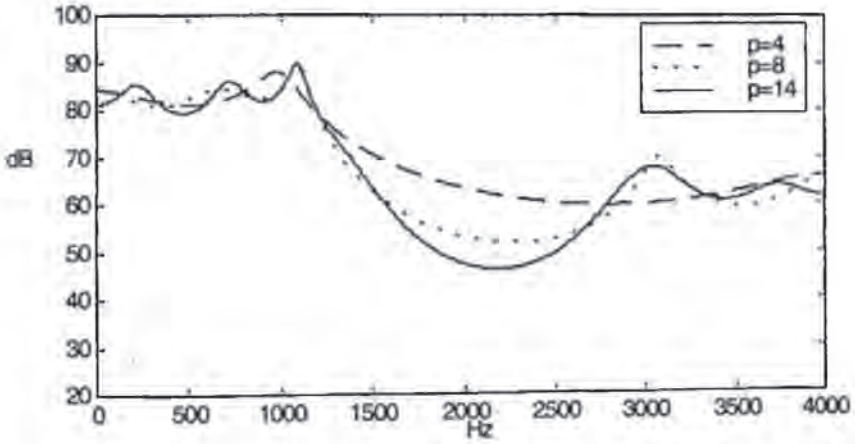


Figure 6.21 LPC spectra of Figure 6.20 for various values of the predictor order p.

6.3.4. The Prediction Error

So far, we have concentrated on the filter component of the source-filter model. Using Eq. (6.23), we can compute the prediction error signal, also called the *excitation*, or *residual* signal. For unvoiced speech we expect the residual to be approximately white noise. In practice, this approximation is quite good, and replacement of the residual by white noise followed by the LPC filter typically results in no audible difference. For voiced speech we

expect the residual to approximate an impulse train. In practice, this is not the case, because the all-pole assumption is not altogether valid; thus, the residual, although it contains spikes, is far from an impulse train. Replacing the residual by an impulse train, followed by the LPC filter, results in speech that sounds somewhat robotic, partly because real speech is not perfectly periodic (it has a random component as well), and because the zeroes are not modeled with the LPC filter. Residual signals computed from inverse LPC filters for several vowels are shown in Figure 6.22.

How do we choose p? This is an important design question. Larger values of p lead to lower prediction errors (see Figure 6.23). Unvoiced speech has higher error than voiced speech, because the LPC model is more accurate for voiced speech. In general, the normalized error rapidly decreases, and then converges to a value of around 12–14 for 8 kHz speech. If we use a large value of p, we are fitting the individual harmonics; thus the LPC filter is modeling the source, and the separation between source and filter is not going to be so good. The more coefficients we have to estimate, the larger the variance of their estimates, since the number of available samples is the same. A rule of thumb is to use 1 complex pole per kHz plus 2–4 poles to model the radiation and glottal effects.

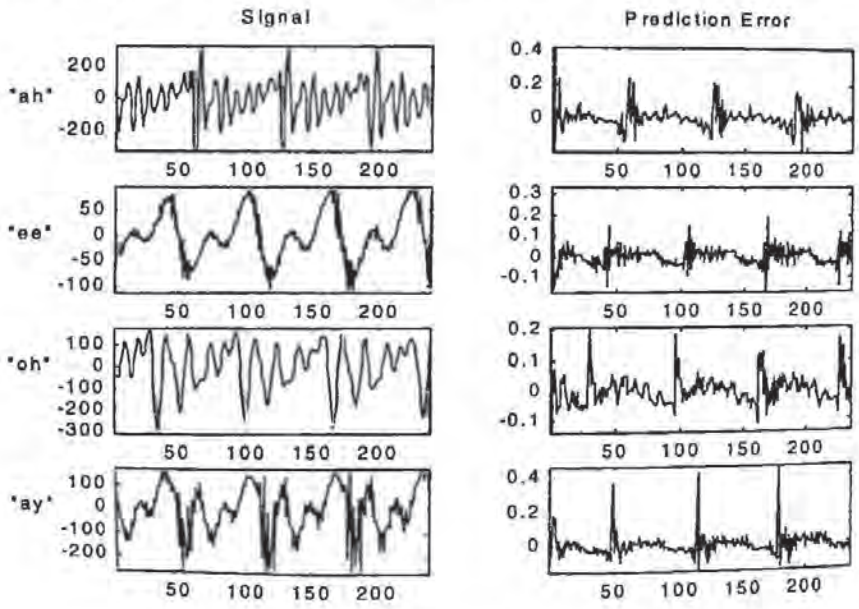


Figure 6.22 LPC prediction error signals for several vowels.

For unvoiced speech, both the autocorrelation and the covariance methods provide similar results. For voiced speech, however, the covariance method can provide better estimates if the analysis window is shorter than the local pitch period and the window only includes samples from the closed phase (when the vocal tract is closed at the glottis and speech signal is due mainly to free resonances). This is called *pitch synchronous* analysis

and results in lower prediction error, because the true excitation is close to zero during the whole analysis window. During the open phase, the trachea, the vocal folds, and the vocal tract are acoustically coupled, and this coupling will change the free resonances. Additionally, the prediction error is higher for both the autocorrelation and the covariance methods if samples from the open phase are included in the analysis window, because the prediction during those instants is poor.

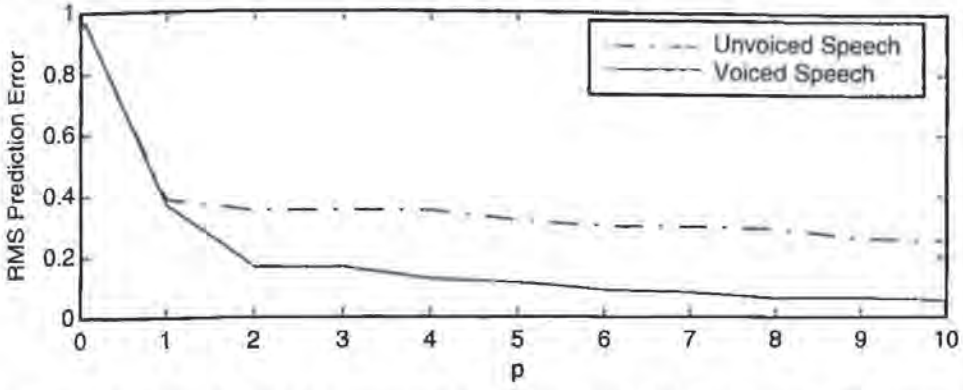


Figure 6.23 Variation of the normalized prediction error with the number of prediction coefficients p for the voiced segment of Figure 6.3 and the unvoiced speech of Figure 6.5. The autocorrelation method was used with a 30 ms Hamming window, and a sampling rate of 8 kHz.

6.3.5. Equivalent Representations

There are a number of alternate useful representations of the predictor coefficients. The most important are the line spectrum frequencies, reflection coefficients, log-area ratios, and the roots of the predictor polynomial.

6.3.5.1. Line Spectral Frequencies

Line Spectral Frequencies (LSF) [18] provide an equivalent representation of the predictor coefficients that is very popular in speech coding. It is derived from computing the roots of the polynomials $P(z)$ and $Q(z)$ defined as

$$P(z) = A(z) + z^{-(p+1)} A(z^{-1}) \tag{6.87}$$

$$Q(z) = A(z) - z^{-(p+1)} A(z^{-1}) \tag{6.88}$$

To gain insight on these roots, look at a second-order predictor filter with a pair of complex roots:

$$A(z) = 1 - a_1 z^{-1} - a_2 z^{-2} = 1 - 2\rho_0 \cos(2\pi f_0) z^{-1} + \rho_0^2 z^{-2} \tag{6.89}$$

where $0 < \rho_0 < 1$ and $0 < f_0 < 0.5$. Inserting Eq. (6.89) into (6.87) and (6.88) results in

$$\begin{aligned} P(z) &= 1-(a_1+a_2)z^{-1}-(a_1+a_2)z^{-2}+z^{-3} \\ Q(z) &= 1-(a_1-a_2)z^{-1}+(a_1-a_2)z^{-2}-z^{-3} \end{aligned} \tag{6.90}$$

From Eq. (6.90) we see that $z=-1$ is a root of $P(z)$ and $z=1$ a root of $Q(z)$, which can be divided out and results in

$$\begin{aligned} P(z) &= (1+z^{-1})(1-2\beta_1 z^{-1}+z^{-2}) \\ Q(z) &= (1-z^{-1})(1-2\beta_2 z^{-1}+z^{-2}) \end{aligned} \tag{6.91}$$

where β_1 and β_2 are given by

$$\begin{aligned} \beta_1 &= \frac{a_1+a_2+1}{2} = \rho_0\cos(2\pi f_0)+\frac{1-\rho_0^2}{2} \\ \beta_2 &= \frac{a_1-a_2-1}{2} = \rho_0\cos(2\pi f_0)-\frac{1-\rho_0^2}{2} \end{aligned} \tag{6.92}$$

It can be shown that $|\beta_1|<1$ and $|\beta_2|<1$ for all possible values of f_0 and ρ_0. With this property, the roots of $P(z)$ and $Q(z)$ in Eq. (6.91) are complex and given by $\beta_1 \pm j\sqrt{1-\beta_1^2}$ and $\beta_2 \pm j\sqrt{1-\beta_2^2}$, respectively. Because they lie in the unit circle, they can be uniquely represented by their angles

$$\begin{aligned} \cos(2\pi f_1) &= \rho_0\cos(2\pi f_0)+\frac{1-\rho_0^2}{2} \\ \cos(2\pi f_2) &= \rho_0\cos(2\pi f_0)-\frac{1-\rho_0^2}{2} \end{aligned} \tag{6.93}$$

where f_1 and f_2 are the *line spectral frequencies* of $A(z)$. Since $|\rho_0|<1$, $\cos(2\pi f_2) < \cos(2\pi f_0)$, and thus $f_2 > f_0$. It's also the case that $\cos(2\pi f_1) > \cos(2\pi f_0)$ and thus $f_1 < f_0$. Furthermore, as $\rho_0 \to 1$, we see from Eq. (6.93) that $f_1 \to f_0$ and $f_2 \to f_0$. We conclude that, given a pole at f_0, the two line spectral frequencies bracket it, i.e., $f_1 < f_0 < f_2$, and that they are closer together as the pole of the second-order resonator gets closer to the unit circle.

We have proven that for a second-order predictor, the roots of $P(z)$ and $Q(z)$ lie in the unit circle, that ± 1 are roots, and that, once sorted, the roots of $P(z)$ and $Q(z)$ alternate. Although we do not prove it here, it can be shown that these conclusions hold for other predictor orders, and, therefore, the p predictor coefficients can be transformed into p line spectral frequencies. We also know that $z=1$ is always a root of $Q(z)$, whereas $z=-1$ is a root of $P(z)$ for even p and a root of $Q(z)$ for odd p.

To compute the LSF for $p>2$, we replace $z=\cos(\omega)$ and compute the roots of $P(\omega)$ and $Q(\omega)$ by any available root finding method. A popular technique, given that

there are p roots which are real in ω and bounded between 0 and 0.5, is to bracket them by observing changes in sign of both functions in a dense grid. To compute the predictor coefficients from the LSF coefficients we can factor $P(z)$ and $Q(z)$ as a product of second-order filters as in Eq. (6.91), and then $A(z)=\big(P(z)+Q(z)\big)/2$.

In practice, LSF are useful because of *sensitivity* (a quantization of one coefficient generally results in a spectral change only around that frequency) and *efficiency* (LSF result in low spectral distortion). This doesn't occur with other representations. As long as the LSF coefficients are ordered, the resulting LPC filter is stable, though the proof is beyond the scope of this book. LSF coefficients are used extensively in Chapter 7.

6.3.5.2. Reflection Coefficients

For the autocorrelation method, the predictor coefficients may be obtained from the reflection coefficients by the following recursion:

$$\begin{aligned} a_i^i &= k_i \qquad\qquad i=1,\cdots,p \\ a_j^i &= a_j^{i-1}-k_i a_{i-j}^{i-1} \qquad 1\le j<i \end{aligned} \tag{6.94}$$

where $a_i=a_i^p$. Similarly, the reflection coefficients may be obtained from the prediction coefficients using a backward recursion of the form

$$\begin{aligned} k_i &= a_i^i \qquad\qquad i=p,\cdots,1 \\ a_j^{i-1} &= \frac{a_j^i+a_i^i a_{i-j}^i}{1-k_i^2} \qquad 1\le j<i \end{aligned} \tag{6.95}$$

where we initialize $a_i^p=a_i$.

Reflection coefficients are useful when implementing LPC filters whose values are interpolated over time, because, unlike the predictor coefficients, they are guaranteed to be stable at all times as long as the anchors satisfy Eq. (6.84).

6.3.5.3. Log-Area Ratios

The *log-area ratio* coefficients are defined as

$$g_i=\ln\left(\frac{1-k_i}{1+k_i}\right) \tag{6.96}$$

with the inverse being given by

$$k_i=\frac{1-e^{g_i}}{1+e^{g_i}} \tag{6.97}$$

The log-area ratio coefficients are equal to the natural logarithm of the ratio of the areas of adjacent sections of a lossless tube equivalent of the vocal tract having the same transfer function. Since for stable predictor filters $-1 < k_i < 1$, we have from Eq. (6.96) that $-\infty < g_i < \infty$. For speech signals, it is not uncommon to have some reflection coefficients close to 1, and quantization of those values can cause a large change in the predictor's transfer function. On the other hand, the log-area ratio coefficients have relatively flat spectral sensitivity (i.e., a small change in their values causes a small change in the transfer function) and thus are useful in coding.

6.3.5.4. Roots of the Polynomial

An alternative to the predictor coefficients results from computing the complex roots of the predictor polynomial:

$$A(z) = 1 - \sum_{k=1}^{p} a_k z^{-k} = \prod_{k=1}^{p} (1 - z_k z^{-1}) \tag{6.98}$$

These roots can be represented as

$$z_k = e^{(-\pi b_k + j2\pi f_k)/F_s} \tag{6.99}$$

where b_k, f_k, and F_s represent the bandwidth, center frequency, and sampling frequency, respectively. Since a_k are real, all complex roots occur in conjugate pairs so that if (b_k, f_k) is a root, so is $(b_k, -f_k)$. The bandwidths b_k are always positive, because the roots are inside the unit circle ($|z_k| < 1$) for a stable predictor. Real roots $z_k = e^{-\pi b_k / F_s}$ can also occur. While algorithms exist to compute the complex roots of a polynomial, in practice there are sometimes numerical difficulties in doing so.

If the roots are available, it is straightforward to compute the predictor coefficients by using Eq. (6.98). Since the roots of the predictor polynomial represent resonance frequencies and bandwidths, they are used in the formant synthesizers of Chapter 16.

6.4. CEPSTRAL PROCESSING

A *homomorphic* transformation $\hat{x}[n] = D(x[n])$ is a transformation that converts a convolution

$$x[n] = e[n] * h[n] \tag{6.100}$$

into a sum

$$\hat{x}[n] = \hat{e}[n] + \hat{h}[n] \tag{6.101}$$

In this section we introduce the *cepstrum* as one homomorphic transformation [32] that allows us to separate the source from the filter. We show that we can find a value N such that the cepstrum of the filter $\hat{h}[n] \approx 0$ for $n \geq N$, and that the cepstrum of the excitation $\hat{e}[n] \approx 0$ for $n < N$. With this assumption, we can approximately recover both $e[n]$ and $h[n]$ from $\hat{x}[n]$ by homomorphic filtering. In Figure 6.24, we show how to recover $h[n]$ with a homomorphic filter:

$$l[n] = \begin{cases} 1 & |n| < N \\ 0 & |n| \geq N \end{cases} \tag{6.102}$$

where D is the cepstrum operator.

The excitation signal can be similarly recovered with a homomorphic filter given by

$$l[n] = \begin{cases} 1 & |n| \geq N \\ 0 & |n| < N \end{cases} \tag{6.103}$$

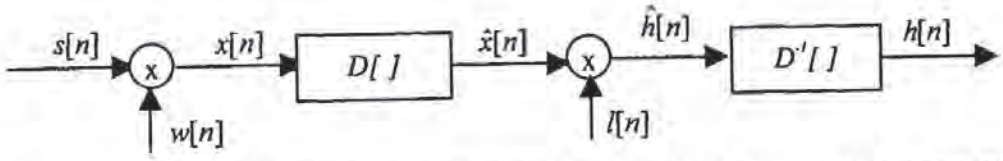


Figure 6.24 Homomorphic filtering to recover the filter's response from a periodic signal. We have used the homomorphic filter of Eq. (6.102).

6.4.1. The Real and Complex Cepstrum

The *real cepstrum* of a digital signal $x[n]$ is defined as

$$c[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} \ln |X(e^{j\omega})| \, e^{j\omega n} d\omega \tag{6.104}$$

and the *complex cepstrum* of $x[n]$ is defined as

$$\hat{x}[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} \ln X(e^{j\omega}) \, e^{j\omega n} d\omega \tag{6.105}$$

where the complex logarithm is used:

$$\hat{X}(e^{j\omega}) = \ln X(e^{j\omega}) = \ln |X(e^{j\omega})| + j\theta(\omega) \tag{6.106}$$

and the phase $\theta(\omega)$ is given by

$$\theta(\omega) = \arg\left[X(e^{j\omega})\right] \tag{6.107}$$

You can see from Eqs. (6.104) and (6.105) that both the real and the complex cepstrum satisfy Eq. (6.101) and thus they are homomorphic transformations.

If the signal $x[n]$ is real, both the real cepstrum $c[n]$ and the complex cepstrum $\hat{x}[n]$ are also real signals. Therefore the term complex cepstrum doesn't mean that it is a complex signal but rather that the complex logarithm is taken.

It can easily be shown that $c[n]$ is the even part of $\hat{x}[n]$:

$$c[n] = \frac{\hat{x}[n] + \hat{x}[-n]}{2} \tag{6.108}$$

From here on, when we refer to cepstrum without qualifiers, we are referring to the real cepstrum, since it is the most widely used in speech technology.

The cepstrum was invented by Bogert et al. [6], and its term was coined by reversing the first syllable of the word spectrum, given that it is obtained by taking the inverse Fourier transform of the log-spectrum. Similarly, they defined the term *quefrency* to represent the independent variable n in $c[n]$. The quefrency has dimension of time.

6.4.2. Cepstrum of Pole-Zero Filters

A very general type of filters are those with rational transfer functions

$$H(z) = \frac{Az^r \prod_{k=1}^{M_i}(1-a_k z^{-1}) \prod_{k=1}^{M_o}(1-u_k z)}{\prod_{k=1}^{N_i}(1-b_k z^{-1}) \prod_{k=1}^{N_o}(1-v_k z)} \tag{6.109}$$

with the magnitudes of a_k, b_k, u_k, and v_k all less than 1. Therefore, $(1-a_k z^{-1})$ and $(1-b_k z^{-1})$ represent the zeros and poles inside the unit circle, whereas $(1-u_k z)$ and $(1-v_k z)$ represent the zeros and poles outside the unit circle, and z^r is a shift from the time origin. Thus, the complex logarithm is

$$\begin{aligned} \hat{H}(z) &= \ln[A] + \ln[z^r] + \sum_{k=1}^{M_i} \ln(1-a_k z^{-1}) \\ &- \sum_{k=1}^{N_i} \ln(1-b_k z^{-1}) + \sum_{k=1}^{M_o} \ln(1-u_k z) - \sum_{k=1}^{N_o} \ln(1-v_k z) \end{aligned} \tag{6.110}$$

where the term $\ln[z']$ contributes to the imaginary part of the complex cepstrum only with a term $j\omega r$. Since it just carries information about the time origin, it's typically ignored. We use the Taylor series expansion

$$\ln(1-x) = -\sum_{n=1}^{\infty} \frac{x^n}{n} \tag{6.111}$$

in Eq. (6.110) and take inverse z-transforms to obtain

$$\hat{h}[n] = \begin{cases} \log[A] & n = 0 \\ \sum_{k=1}^{N_i} \frac{b_k^n}{n} - \sum_{k=1}^{M_i} \frac{a_k^n}{n} & n > 0 \\ \sum_{k=1}^{M_o} \frac{u_k^n}{n} - \sum_{k=1}^{N_o} \frac{v_k^n}{n} & n < 0 \end{cases} \tag{6.112}$$

If the filter's impulse response doesn't have zeros or poles outside the unit circle, the so-called *minimum phase* signals, then $\hat{h}[n] = 0$ for $n < 0$. *Maximum phase* signals are those with $\hat{h}[n] = 0$ for $n > 0$. If a signal is minimum phase, its complex cepstrum can be uniquely determined from its real cepstrum:

$$\hat{h}[n] = \begin{cases} 0 & n < 0 \\ c[n] & n = 0 \\ 2c[n] & n > 0 \end{cases} \tag{6.113}$$

It is easy to see from Eq. (6.112) that both the real and complex cepstrum are decaying sequences, which is the reason why, typically, a finite number of coefficients are sufficient to approximate it, and, therefore, people refer to the truncated cepstrum signal as a *cepstrum vector*.

6.4.2.1. LPC-Cepstrum

The case when the rational transfer function in Eq. (6.109) has been obtained with an LPC analysis is particularly interesting, since LPC analysis is such a widely used method. While Eq. (6.112) applies here, too, it is useful to find a recursion which doesn't require us to compute the roots of the predictor polynomial. Given the LPC filter

$$H(z) = \frac{G}{1 - \sum_{k=1}^{p} a_k z^{-k}} \tag{6.114}$$

we take the logarithm

$$\hat{H}(z) = \ln G - \ln\left(1 - \sum_{l=1}^{p} a_l z^{-l}\right) = \sum_{k=-\infty}^{\infty} \hat{h}[k] z^{-k} \tag{6.115}$$

and the derivative of both sides with respect to z

$$\frac{-\sum_{n=1}^{p} n a_n z^{-n-1}}{1 - \sum_{l=1}^{p} a_l z^{-l}} = -\sum_{k=-\infty}^{\infty} k\hat{h}[k] z^{-k-1} \tag{6.116}$$

Multiplying both sides by $-z\left(1 - \sum_{l=1}^{p} a_l z^{-l}\right)$, we obtain

$$\sum_{n=1}^{p} n a_n z^{-n} = \sum_{n=-\infty}^{\infty} n\hat{h}[n] z^{-n} - \sum_{l=1}^{p} \sum_{k=-\infty}^{\infty} k\hat{h}[k] a_l z^{-k-l} \tag{6.117}$$

which, after replacing $l = n - k$, and equating terms in z^{-n}, results in

$$\begin{aligned} n a_n &= n\hat{h}[n] - \sum_{k=1}^{n-1} k\hat{h}[k] a_{n-k} \qquad 0 < n \le p \\ 0 &= n\hat{h}[n] - \sum_{k=n-p}^{n-1} k\hat{h}[k] a_{n-k} \qquad n > p \end{aligned} \tag{6.118}$$

so that the complex cepstrum can be obtained from the LPC coefficients by the following recursion:

$$\hat{h}[n] = \begin{cases} 0 & n < 0 \\ \ln G & n = 0 \\ a_n + \sum_{k=1}^{n-1} \left(\frac{k}{n}\right) \hat{h}[k] a_{n-k} & 0 < n \le p \\ \sum_{k=n-p}^{n-1} \left(\frac{k}{n}\right) \hat{h}[k] a_{n-k} & n > p \end{cases} \tag{6.119}$$

where the value for $n = 0$ can be obtained from Eqs. (6.115) and (6.111). We note that, while there are a finite number of LPC coefficients, the number of cepstrum coefficients is infinite. Speech recognition researchers have shown empirically that a finite number is sufficient: 12–20 depending on the sampling rate and whether or not frequency warping is done. In Chapter 8 we discuss the use of the cepstrum in speech recognition.

This recursion should not be used in the reverse mode to compute the LPC coefficients from *any* set of cepstrum coefficients, because the recursion in Eq. (6.119) assumes an all-pole model with all poles inside the unit circle, and that might not be the case for an arbitrary cepstrum sequence, so that the recursion might yield a set of unstable LPC coefficients. In some experiments it has been shown that quantized LPC-cepstrum can yield unstable LPC coefficients over 5% of the time.

6.4.3. Cepstrum of Periodic Signals

It is important to see what the cepstrum of periodic signals looks like. To do so, let's consider the following signal:

$$x[n] = \sum_{k=0}^{M-1} \alpha_k \delta[n - kN] \tag{6.120}$$

which can be viewed as an impulse train of period N multiplied by an analysis window, so that only M impulses remain. Its z-transform is

$$X(z) = \sum_{k=0}^{M-1} \alpha_k z^{-kN} \tag{6.121}$$

which is a polynomial in z^{-N} rather than z^{-1}. Therefore, $X(z)$ can be expressed as a product of factors of the form $(1 - a_k z^{-Nk})$ and $(1 - u_k z^{Nk})$. Following the derivation in Section 6.4.2, it is clear that its complex cepstrum is nonzero only at integer multiples of N:

$$\hat{x}[n] = \sum_{k=-\infty}^{\infty} \beta_k \delta[n - kN] \tag{6.122}$$

A particularly interesting case is when $\alpha_k = \alpha^k$ with $0 < \alpha < 1$, so that Eq. (6.121) can be expressed as

$$X(z) = 1 + \alpha z^{-N} + \cdots + (\alpha z^{-N})^{M-1} = \frac{1 - (\alpha z^{-N})^M}{1 - \alpha z^{-N}} \tag{6.123}$$

so that taking the logarithm of Eq. (6.123) and expanding it in Taylor series using Eq. (6.111) results in

$$\hat{X}(z) = \ln X(z) = \sum_{r=1}^{\infty} \frac{\alpha^r}{r} z^{-rN} - \sum_{l=1}^{\infty} \frac{\alpha^{lM}}{l} z^{-lMN} = \sum_{n=1}^{\infty} \hat{x}[n] z^{-n} \tag{6.124}$$

which lets us compute the complex cepstrum as

$$\hat{x}[n]=\sum_{r=1}^{\infty}\frac{\alpha^r}{r}\delta[n-rN]-\sum_{l=1}^{\infty}\frac{\alpha^{lM}}{l}\delta[n-lMN] \tag{6.125}$$

An infinite impulse train can be obtained by making $\alpha \to 1$ and $M \to \infty$ in Eq. (6.125).

$$\hat{x}[n]=\sum_{r=1}^{\infty}\frac{\delta[n-rN]}{r} \tag{6.126}$$

We see from Eq. (6.126) that the cepstrum of an impulse train goes to 0 as n increases. This justifies our assumption of homomorphic filtering.

6.4.4. Cepstrum of Speech Signals

We can compute the cepstrum of a speech segment by windowing the signal with a window of length N. In practice, the cepstrum is not computed through Eq. (6.112), since root-finding algorithms are slow and offer numerical imprecision for the large values of N used. Instead, we can compute the cepstrum directly through its definition of Eq. (6.105), using the DFT as follows:

$$X_a[k]=\sum_{n=0}^{N-1}x[n]e^{-j2\pi nk/N}, \quad 0\le k<N \tag{6.127}$$

$$\tilde{X}_a[k]=\ln X_a[k], \quad 0\le k<N \tag{6.128}$$

$$\hat{x}_a[n]=\frac{1}{N}\sum_{n=0}^{N-1}\hat{X}_a[k]e^{-j2\pi nk/N}, \quad 0\le n<N \tag{6.129}$$

The subscript a means that the new complex cepstrum $\hat{x}_a[n]$ is an aliased version of $\hat{x}[n]$ given by

$$\hat{x}_a[n]=\sum_{r=-\infty}^{\infty}\hat{x}[n+rN] \tag{6.130}$$

which can be derived by using the sampling theorem of Chapter 5, by reversing the concepts of time and frequency.

This aliasing introduces errors in the estimation that can be reduced by choosing a large value for N.

Computation of the complex cepstrum requires computing the complex logarithm and, in turn, the phase. However, given the principal value of the phase $\theta_p[k]$, there are infinite possible values for $\theta[k]$:

$$\theta[k]=\theta_p[k]+2\pi n_k \tag{6.131}$$

From Chapter 5 we know that if $x[n]$ is real, $\arg\left[X(e^{j\omega})\right]$ is an odd function and also continuous. Thus we can do *phase unwrapping* by choosing n_k to guarantee that $\theta[k]$ is a smooth function, i.e., by forcing the difference between adjacent values to be small:

$$|\theta[k]-\theta[k-1]|<\pi \tag{6.132}$$

A linear phase term r as in Eq. (6.110), would contribute to the phase difference in Eq. (6.132) with $2\pi r/N$, which may result in errors in the phase unwrapping if $\theta[k]$ is changing sufficiently rapidly. In addition, there could be large changes in the phase difference if $X_a[k]$ is noisy. To guarantee that we can track small phase differences, a value of N several times larger than the window size is required: i.e., the input signal has to be zero-padded prior to the FFT computation. Finally, the delay r in Eq. (6.109), can be obtained by forcing the phase to be an odd function, so that:

$$\theta[N/2]=\pi r \tag{6.133}$$

For unvoiced speech, the unwrapped phase is random, and therefore only the real cepstrum has meaning. In practical situations, even voiced speech has some frequencies at which noise dominates (typically very low and high frequencies), which results in phase $\theta[k]$ that changes drastically from frame to frame. Because of this, the complex cepstrum in Eq. (6.105) is rarely used for real speech signals. Instead, the real cepstrum is used much more often:

$$C_a[k]=\ln|X_a[k]|, \quad 0\le k<N \tag{6.134}$$

$$c_a[n]=\frac{1}{N}\sum_{n=0}^{N-1}C_a[k]e^{-j2\pi nk/N}, \quad 0\le n<N \tag{6.135}$$

Similarly, it can be shown that for the new real cepstrum $c_a[n]$ is an aliased version of $c[n]$ given by

$$c_a[n]=\sum_{r=-\infty}^{\infty}c[n+rN] \tag{6.136}$$

which again has aliasing that can be reduced by choosing a large value for N.

6.4.5. Source-Filter Separation via the Cepstrum

We have seen that, if the filter is a rational transfer function, and the source is an impulse train, the homomorphic filtering of Figure 6.24 can approximately separate them. Because of problems in estimating the phase in speech signals (see Section 6.4.4), we generally compute the real cepstrum using Eqs. (6.127), (6.134), and (6.135), and then compute the complex cepstrum under the assumption of a minimum phase signal according to Eq. (6.113). The result of separating source and filter using this cepstral deconvolution is shown in Figure 6.25 for voiced speech and Figure 6.26 for unvoiced speech.

The real cepstrum of white noise $x[n]$ with an expected magnitude spectrum $|X(e^{j\omega})|=1$ is 0. If colored noise is present, the cepstrum of the observed colored noise $\hat{y}[n]$ is identical to the cepstrum of the coloring filter $\hat{h}[n]$, except for a gain factor. The above is correct if we take an infinite number of noise samples, but in practice, this cannot be done and a limited number have to be used, so that this is only an approximation, though it is often used in speech processing algorithms.

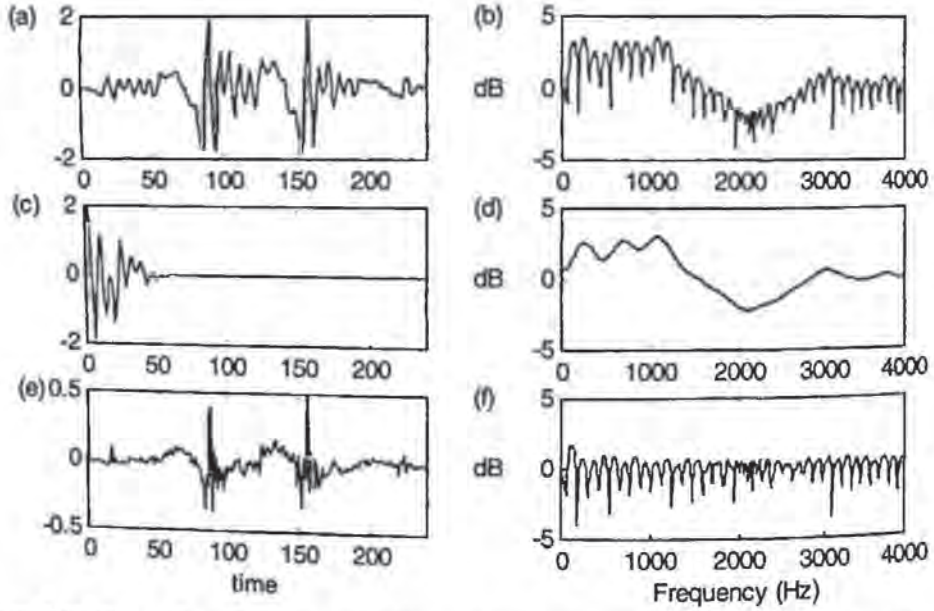


Figure 6.25 Separation of source and filter using homomorphic filtering for voiced speech with the scheme of Figure 6.24 with $N = 20$ in the homomorphic filter of Eq. (6.102) with the real cepstrum: (a) windowed signal, (b) log-spectrum, (c) filter's impulse response, (d) smoothed log-spectrum, (e) windowed excitation signal, (f) log-spectrum of high-part of cepstrum. Note that the windowed excitation is not a windowed impulse train because of the minimum phase assumption.

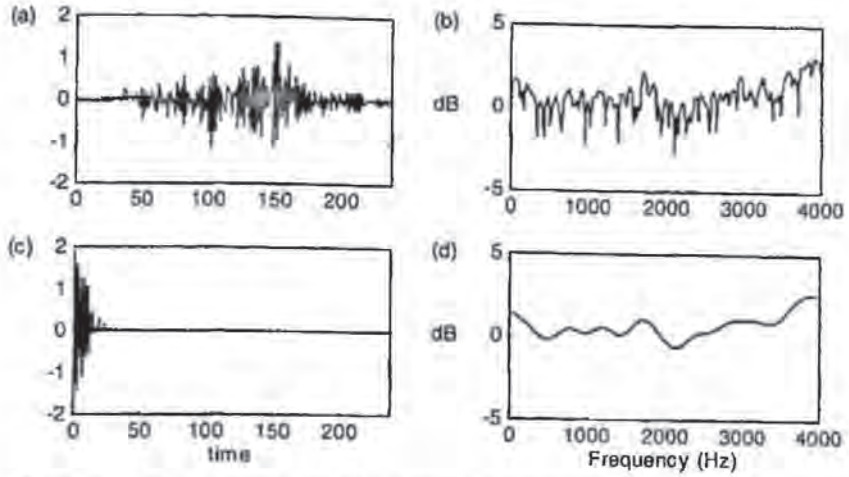


Figure 6.26 Separation of source and filter using homomorphic filtering for unvoiced speech with the scheme of Figure 6.24 with $N = 20$ in the homomorphic filter of Eq. (6.102) with the real cepstrum: (a) windowed signal, (b) log-spectrum, (c) filter's impulse response, (d) smoothed log-spectrum.

6.5. PERCEPTUALLY MOTIVATED REPRESENTATIONS

In this section we describe some aspects of human perception, and methods motivated by the behavior of the human auditory system: bilinearly transformed cepstrum, Mel-Frequency Cepstrum Coefficients (MFCC), and Perceptual Linear Prediction (PLP). These methods have been successfully used in speech recognition.

6.5.1. The Bilinear Transform

The transformation

$$s = \frac{z^{-1} - \alpha}{1 - \alpha z^{-1}} \tag{6.137}$$

for $0 < \alpha < 1$ belongs to the class of *bilinear* transforms. It is a mapping in the complex plane that maps the unit circle onto itself. The frequency transformation is obtained by making the substitution $z = e^{j\omega}$ and $s = e^{j\Omega}$:

$$\Omega = \omega + 2\arctan\left[\frac{\alpha \sin(\omega)}{1 - \alpha\cos(\omega)}\right] \tag{6.138}$$

This transformation is very similar to the Bark and mel scale for an appropriate choice of the parameter α (see Chapter 2). Oppenheim [31] showed that the advantage of this transformation is that it can be used to transform a time sequence in the linear frequency into another time sequence in the warped frequency, as shown in Figure 6.27. This bilinear transform has been successfully applied to cepstral and autocorrelation coefficients.

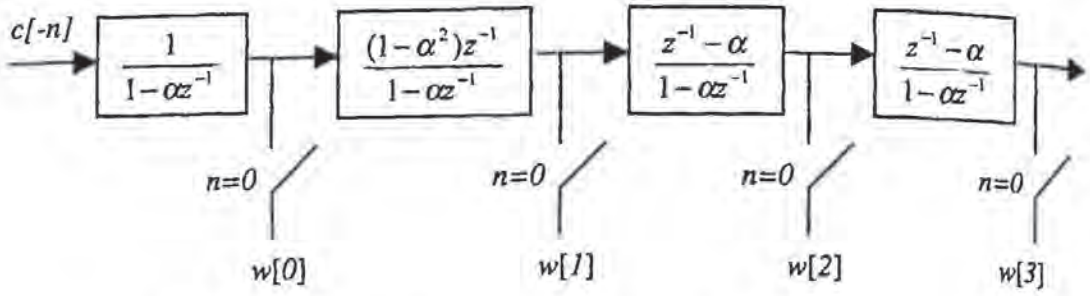


Figure 6.27 Implementation of the frequency-warped cepstral coefficients as a function of the linear-frequency cepstrum coefficients. Both sets of coefficients are causal. The input is the time-reversed cepstrum sequence, and the output can be obtained by sampling the outputs of the filters at time $n = 0$. The filters used for $w[m]$ $m > 2$ are the same. Note that, for a finite-length cepstrum, an infinite-length warped cepstrum results.

For a finite number of cepstral coefficients the bilinear transform in Figure 6.27 results in an infinite number of warped cepstral coefficients. Since truncation is usually done in practice, the bilinear transform is equivalent to a matrix multiplication, where the matrix is a function of the warping parameter α. Shikano [43] showed these warped cepstral coefficients were beneficial for speech recognition.

6.5.2. Mel-Frequency Cepstrum

The *Mel-Frequency Cepstrum Coefficients* (MFCC) is a representation defined as the real cepstrum of a windowed short-time signal derived from the FFT of that signal. The difference from the real cepstrum is that a nonlinear frequency scale is used, which approximates the behavior of the auditory system. Davis and Mermelstein [8] showed the MFCC representation to be beneficial for speech recognition.

Given the DFT of the input signal

$$X_a[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi nk/N}, \quad 0 \le k < N \tag{6.139}$$

we define a filterbank with M filters ($m = 1, 2, \cdots, M$), where filter m is triangular filter given by:

$$H_m[k]=\begin{cases}0 & k<f[m-1]\\ \dfrac{2(k-f[m-1])}{(f[m+1]-f[m-1])(f[m]-f[m-1])} & f[m-1]\le k\le f[m]\\ \dfrac{2(f[m+1]-k)}{(f[m+1]-f[m-1])(f[m+1]-f[m])} & f[m]\le k\le f[m+1]\\ 0 & k>f[m+1]\end{cases} \tag{6.140}$$

Such filters compute the average spectrum around each center frequency with increasing bandwidths, and they are displayed in Figure 6.28.

Alternatively, the filters can be chosen as

$$H_m'[k]=\begin{cases}0 & k<f[m-1]\\ \dfrac{(k-f[m-1])}{(f[m]-f[m-1])} & f[m-1]\le k\le f[m]\\ \dfrac{(f[m+1]-k)}{(f[m+1]-f[m])} & f[m]\le k\le f[m+1]\\ 0 & k>f[m+1]\end{cases} \tag{6.141}$$

which satisfies $\sum_{m=1}^{M} H_m'[k]=1$. The mel-cepstrum computed with $H_m[k]$ or $H_m'[k]$ will differ by a constant vector for all inputs, so the choice becomes unimportant when used in a speech recognition system that has been trained with the same filters.

Let's define f_l and f_h to be the lowest and highest frequencies of the filterbank in Hz, F_s the sampling frequency in Hz, M the number of filters, and N the size of the FFT. The boundary points $f[m]$ are uniformly spaced in the mel-scale:

$$f[m]=\left(\frac{N}{F_s}\right)B^{-1}\left(B(f_l)+m\frac{B(f_h)-B(f_l)}{M+1}\right) \tag{6.142}$$

where the mel-scale B is given by Eq. (2.6), and B^{-1} is its inverse

$$B^{-1}(b)=700\left(\exp(b/1125)-1\right) \tag{6.143}$$

We then compute the log-energy at the output of each filter as

$$S[m]=\ln\left[\sum_{k=0}^{N-1}\left|X_a[k]\right|^2 H_m[k]\right], \qquad 0<m\le M \tag{6.144}$$

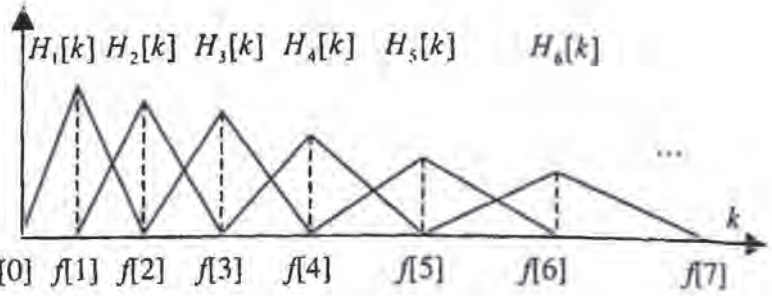


Figure 6.28 Triangular filters used in the computation of the mel-cepstrum using Eq. (6.140).

The mel-frequency cepstrum is then the discrete cosine transform of the M filter outputs:

$$c[n] = \sum_{m=0}^{M-1} S[m] \cos\left(\pi n (m - 1/2)/M\right) \qquad 0 \le n < M \tag{6.145}$$

where M varies for different implementations from 24 to 40. For speech recognition, typically only the first 13 cepstrum coefficients are used. It is important to note that the MFCC representation is no longer a homomorphic transformation. It would be if the order of summation and logarithms in Eq. (6.144) were reversed:

$$S[m] = \sum_{k=0}^{N-1} \ln\left(|X_a[k]|^2 H_m[k]\right) \qquad 0 < m \le M \tag{6.146}$$

In practice, however, the MFCC representation is approximately homomorphic for filters that have a smooth transfer function. The advantage of the MFCC representation using (6.144) instead of (6.146) is that the filter energies are more robust to noise and spectral estimation errors. This algorithm has been used extensively as a feature vector for speech recognition systems.

While the definition of cepstrum in Section 6.4.1 uses an inverse DFT, since $S[m]$ is even, a DCT-II can be used instead (see Chapter 5).

6.5.3. Perceptual Linear Prediction (PLP)

Perceptual Linear Prediction (PLP) [16] uses the standard Durbin recursion of Section 6.3.2.1.2 to compute LPC coefficients, and typically the LPC coefficients are transformed to LPC-cepstrum using the recursion in Section 6.4.2.1. But unlike standard linear prediction, the autocorrelation coefficients are not computed in the time domain through Eq. (6.55).

The autocorrelation $R_x[n]$ is the inverse Fourier transform of the power spectrum $|X(\omega)|^2$ of the signal. We cannot compute the continuous-frequency Fourier transform eas-

ily, but we can take an FFT to compute $X[k]$, so that the autocorrelation can be obtained as the inverse Fourier transform of $|X[k]|^2$. Since the discrete Fourier transform is not performing linear convolution but circular convolution, we need to make sure that the FFT size is larger than twice the window length (see Section 5.3.4) for this to hold. This alternate way of computing autocorrelation coefficients, entailing two FFTs and N multiplies and adds, should yield identical results. Since normally only a small number p of autocorrelation coefficients are needed, this is generally not a cost-effective way to do it, unless the first FFT has to be computed for other reasons.

Perceptual linear prediction uses the above method, but replaces $|X[k]|^2$ by a perceptually motivated power spectrum. The most important aspect is the non-linear frequency scaling, which can be achieved through a set of filterbanks similar to those described in Section 6.5.2, so that this critical-band power spectrum can be sampled in approximately 1-Bark intervals. Another difference is that, instead of taking the logarithm on the filterbank energy outputs, a different non-linearity compression is used, often the cubic root. It is reported [16] that the use of this different non-linearity is beneficial for speech recognizers in noisy conditions.

6.6. FORMANT FREQUENCIES

Formant frequencies are the resonances in the vocal tract and, as we saw in Chapter 2, they convey the differences between different sounds. Expert spectrogram readers are able to recognize speech by looking at a spectrogram, particularly at the formants. It has been argued that they are very useful features for speech recognition, but they haven't been widely used because of the difficulty in estimating them.

One way of obtaining formant candidates at a frame level is to compute the roots of a p^{th}-order LPC polynomial [3, 26]. There are standard algorithms to compute the complex roots of a polynomial with real coefficients [36], though convergence is not guaranteed. Each complex root z_i can be represented as

$$z_i = \exp(-\pi b_i + j2\pi f_i) \tag{6.147}$$

where f_i and b_i are the formant frequency and bandwidth, respectively, of the i^{th} root. Real roots are discarded and complex roots are sorted by increasing f, discarding negative values. The remaining pairs (f_i, b_i) are the formant candidates. Traditional formant trackers discard roots whose bandwidths are higher than a threshold [46], say 200 Hz.

Closed-phase analysis of voiced speech [5] uses only the regions for which the glottis is closed and thus there is no excitation. When the glottis is open, there is a coupling of the vocal tract with the lungs and the resonance bandwidths are somewhat larger. Determination of the closed-phase regions directly from the speech signal is difficult, so often an *electroglottograph* (EGG) signal is used [23]. EGG signals, obtained by placing electrodes at the speaker's throat, are very accurate in determining the times when the glottis is closed. Using samples in the closed-phase covariance analysis can yield accurate results [46]. For female

speech, the closed-phase is short, and sometimes non-existent, so such analysis can be a challenge. EGG signals are useful also for pitch tracking and are described in more detail in Chapter 16.

Another common method consists of finding the peaks on a smoothed spectrum, such as that obtained through an LPC analysis [26, 40]. The advantage of this method is that you can always compute the peaks and it is more computationally efficient than extracting the complex roots of a polynomial. On the other hand, this procedure generally doesn't estimate the formant's bandwidth. The first three formants are typically estimated this way for formant synthesis (see Chapter 16), since they are the ones that allow sound classification, whereas the higher formants are more speaker dependent.

Sometimes, the signal goes through some *conditioning*, which includes sampling rate conversion to remove frequencies outside the range we are interested in. For example, if we are interested only in the first three formants, we can safely downsample the input signal to 8 kHz, since we know all three formants should be below 4 kHz. This downsampling reduces computation and the chances of the algorithm to find formant values outside the expected range (otherwise peaks or roots could be chosen above 4 kHz which we know do not correspond to any of the first three formants). Pre-emphasis filtering is also often used to whiten the signal.

Because of the thresholds imposed above, it is possible that the formants are not continuous. For example, when the vocal tract's spectral envelope is changing rapidly, bandwidths obtained through the above methods are overestimates of the true bandwidths, and they may exceed the threshold and thus be rejected. It is also possible for the peak-picking algorithm to classify a harmonic as a formant during some regions where it is much stronger than the other harmonics. Due to the thresholds used, a given frame could have no formants, only one formant (either first, second, or third), two, three, or more. Formant alignment from one frame to another has often been done using heuristics to prevent such discontinuities.

6.6.1. Statistical Formant Tracking

It is desirable to have an approach that does not use any thresholds on formant candidates and uses a probabilistic model to do the tracking instead of heuristics [1]. The formant candidates can be obtained from roots of the LPC polynomial, peaks in the smoothed spectrum, or even from a dense sample of possible points. If the first n formants are desired, and we have $(p/2)$ formant candidates, a maximum of r n-tuples are considered, where r is given by

$$r = \binom{p/2}{n} \tag{6.148}$$

A Viterbi search (see Chapter 8) is then carried out to find the most likely path of formant n-tuples given a model with some a priori knowledge of formants. The prior distribution for formant targets is used to determine which formant candidate to use of all possible choices for the given phoneme (i.e., we know that F1 for an AE should be around 800 Hz).

Formant continuity is imposed through the prior distribution of the formant slopes. This algorithm produces n formants for every frame, including silence.

Since we are interested in obtaining the first three formants ($n = 3$) and F3 is known to be lower than 4 kHz, it is advantageous to downsample the signal to 8 kHz in order to avoid obtaining formant candidates above 4 kHz and to let us use a lower-order analysis which offers fewer numerical problems when computing the roots. With $p = 14$, it results in a maximum of $r = 35$ triplets for the case of no real roots.

Let $\mathbf{X}$ be a sequence of T feature vectors $\mathbf{x}_t$ of dimension n:

$$\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T)' \tag{6.149}$$

where the prime denotes transpose.

We estimate the formants with the knowledge of what sound occurs at that particular time, for example by using a speech recognizer that segments the waveform into different phonemes (see Chapter 9) or states q_t within a phoneme. In this case we assume that the output distribution of each state i is modeled by one Gaussian density function with a mean μ_i and covariance matrix Σ_i. We can define up to N states, with λ being the set of all means and covariance matrices for all:

$$\lambda = (\mu_1, \Sigma_1, \mu_2, \Sigma_2, \cdots, \mu_N, \Sigma_N) \tag{6.150}$$

Therefore, the log-likelihood for $\mathbf{X}$ is given by

$$\ln p(\mathbf{X} \mid \hat{\mathbf{q}}, \lambda) = -\frac{TM}{2}\ln(2\pi) - \frac{1}{2}\sum_{t=1}^{T}\ln|\Sigma_{q_t}| - \frac{1}{2}\sum_{t=1}^{T}(\mathbf{x}_t - \mu_{q_t})'\Sigma_{q_t}^{-1}(\mathbf{x}_t - \mu_{q_t}) \tag{6.151}$$

Maximizing $\mathbf{X}$ in Eq. (6.151) leads to the trivial solution $\tilde{\mathbf{X}} = (\mu_{q_1}, \mu_{q_2}, \ldots, \mu_{q_T})'$, a piecewise function whose value is that of the best n-tuple candidate. This function has discontinuities at state boundaries and thus is not likely to represent well the physical phenomena of speech.

This problem arises because the slopes at state boundaries do not match the slopes of natural speech. To avoid these discontinuities, we would like to match not only the target formants at each state, but also the formant slopes at each state. To do that, we augment the feature vector $\mathbf{x}_t$ at frame t with the delta vector $\mathbf{x}_t - \mathbf{x}_{t-1}$. Thus, we increase the parameter space of λ with the corresponding means δ_t and covariance matrices Γ_t of these delta parameters, and assume statistical independence among them. The corresponding new log-likelihood has the form

$$\begin{aligned}\ln p(\mathbf{X} \mid \hat{\mathbf{q}}, \lambda) = K &- \frac{1}{2}\sum_{t=1}^{T}\ln|\Sigma_{q_t}| - \frac{1}{2}\sum_{t=2}^{T}\ln|\Gamma_{q_t}| \\ &- \frac{1}{2}\sum_{t=1}^{T}(\mathbf{x}_t - \mu_{q_t})'\Sigma_{q_t}^{-1}(\mathbf{x}_t - \mu_{q_t}) - \frac{1}{2}\sum_{t=2}^{T}(\mathbf{x}_t - \mathbf{x}_{t-1} - \delta_{q_t})'\Gamma_{q_t}^{-1}(\mathbf{x}_t - \mathbf{x}_{t-1} - \delta_{q_t})\end{aligned} \tag{6.152}$$

Maximization of Eq. (6.152) with respect to $\mathbf{x}_t$ requires solving several sets of linear equations. If Γ_i and Σ_i are diagonal covariance matrices, it results in a set of linear equations for each of the M dimensions

$$\mathbf{BX} = \mathbf{c} \tag{6.153}$$

where $\mathbf{B}$ is a tridiagonal matrix (all values are zero except for those in the main diagonal and its two adjacent diagonals), which leads to a very efficient solution [36]. For example, the values of $\mathbf{B}$ and $\mathbf{c}$ for $T = 3$ are given by

$$\mathbf{B} = \begin{pmatrix} \frac{1}{\sigma_{q_1}^2} + \frac{1}{\gamma_{q_2}^2} & -\frac{1}{\gamma_{q_2}^2} & 0 \\ -\frac{1}{\gamma_{q_2}^2} & \frac{1}{\sigma_{q_2}^2} + \frac{1}{\gamma_{q_2}^2} + \frac{1}{\gamma_{q_3}^2} & -\frac{1}{\gamma_{q_3}^2} \\ 0 & -\frac{1}{\gamma_{q_3}^2} & \frac{1}{\sigma_{q_3}^2} + \frac{1}{\gamma_{q_3}^2} \end{pmatrix} \tag{6.154}$$

$$\mathbf{c} = \begin{pmatrix} \frac{\mu_{q_1}}{\sigma_{q_1}^2} - \frac{\delta_{q_2}}{\gamma_{q_2}^2} & \frac{\mu_{q_2}}{\sigma_{q_2}^2} + \frac{\delta_{q_2}}{\gamma_{q_2}^2} - \frac{\delta_{q_3}}{\gamma_{q_3}^2} & \frac{\mu_{q_3}}{\sigma_{q_3}^2} + \frac{\delta_{q_3}}{\gamma_{q_3}^2} \end{pmatrix}' \tag{6.155}$$

where just one dimension is represented, and the process is repeated for all dimensions with a computational complexity of $O(TM)$.

The maximum likelihood sequence $\hat{\mathbf{x}}_t$ is close to the targets μ_i while keeping the slopes close to δ_i for a given state i, thus estimating a continuous function. Because of the delta coefficients, the solution depends on all the parameters of all states and not just the current state. This procedure can be performed for the formants as well as the bandwidths.

The parameters μ_i, Σ_i, δ_i, and Γ_i can be re-estimated using the EM algorithm described in Chapter 8. In [1] it is reported that two or three iterations are sufficient for speaker-dependent data.

The formant track obtained through this method can be rough, and it may be desired to smooth it. Smoothing without knowledge about the speech signal would result in either blurring the sharp transitions that occur in natural speech, or maintaining ragged formant tracks where the underlying physical phenomena vary slowly with time. Ideally we would like a larger adjustment to the raw formant when the error in the estimate is large relative to the variance of the corresponding state within a phoneme. This can be done by modeling the formant measurement error as a Gaussian distribution. Figure 6.29 shows an utterance from a male speaker with the smoothed formant tracks, and Figure 6.30 compares the raw and smoothed formants. When no real formant is visible from the spectrogram, the algorithm tends to assign a large bandwidth (not shown in the figure).

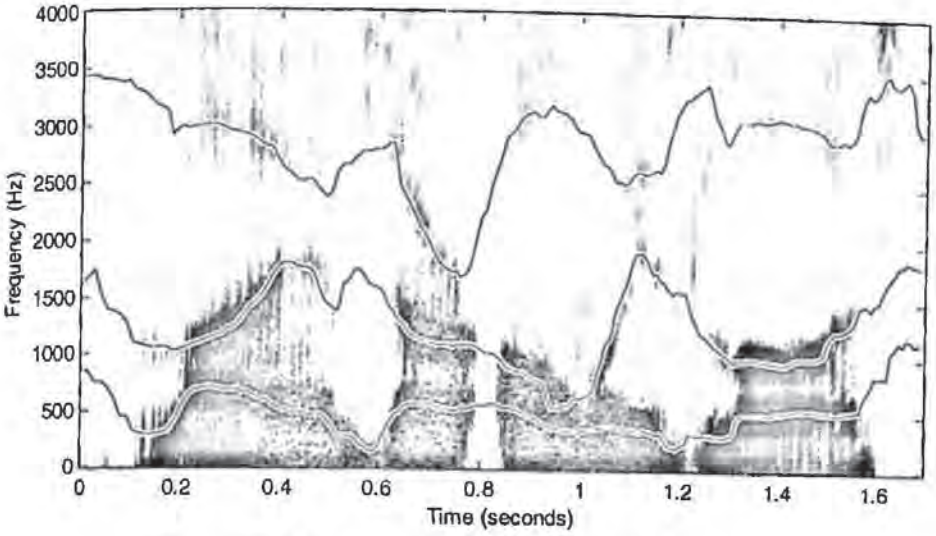


Figure 6.29 Spectrogram and three smoothed formants.

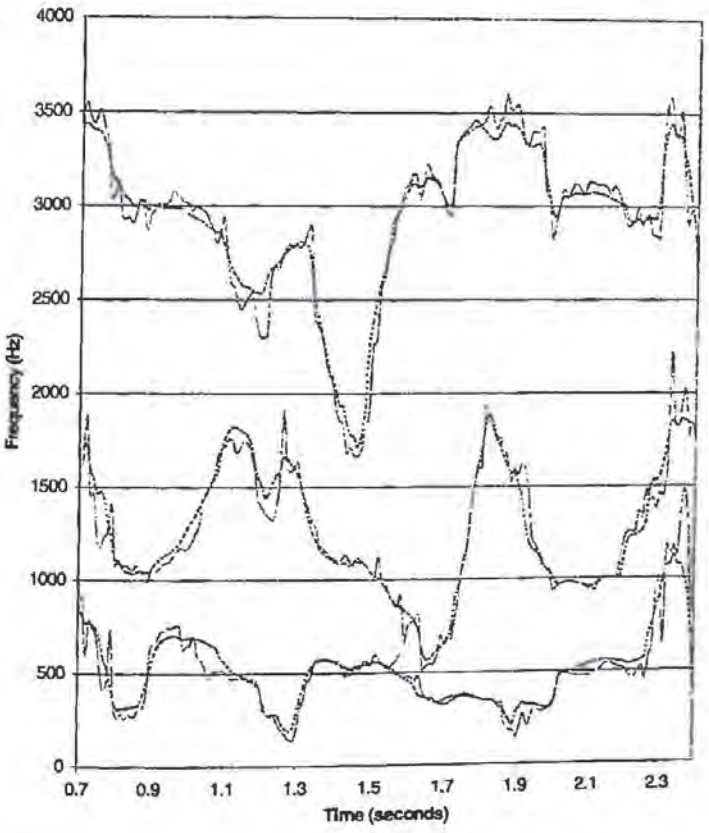


Figure 6.30 Raw formants (ragged gray line) and smoothed formants (dashed line).

6.7. THE ROLE OF PITCH

Pitch determination is very important for many speech processing algorithms. The concatenative speech synthesis methods of Chapter 16 require pitch tracking on the desired speech segments if prosody modification is to be done. Chinese speech recognition systems use pitch tracking for tone recognition, which is important in disambiguating the myriad of homophones. Pitch is also crucial for prosodic variation in text-to-speech systems (see Chapter 15) and spoken language systems (see Chapter 17). While in the previous sections we have dealt with features representing the filter, pitch represents the source of the model illustrated in Figure 6.1.

Pitch determination algorithms also use short-term analysis techniques, which means that for every frame $\mathbf{x}_m$ we get a score $f(T \mid \mathbf{x}_m)$ that is a function of the candidate pitch periods T. These algorithms determine the optimal pitch by maximizing

$$T_m = \arg\max_T f(T \mid \mathbf{x}_m) \tag{6.156}$$

We describe several such functions computed through the autocorrelation method and the normalized cross-correlation method, as well as the signal conditioning that is often performed. Other approaches based on cepstrum [28] have also been used successfully. A good summary of techniques used for pitch tracking is provided by [17, 45].

Pitch determination using Eq. (6.156) is error prone, and a smoothing stage is often done. This smoothing, described in Section 6.7.4, takes into consideration that the pitch does not change quickly over time.

6.7.1. Autocorrelation Method

A commonly used method to estimate pitch is based on detecting the highest value of the autocorrelation function in the region of interest. This region must exclude $m = 0$, as that is the absolute maximum of the autocorrelation function [37]. As discussed in Chapter 5, the statistical autocorrelation of a sinusoidal random process

$$\mathbf{x}[n] = \cos(\omega_0 n + \varphi) \tag{6.157}$$

is given by

$$R[m] = E\{\mathbf{x}^*[n]\mathbf{x}[n+m]\} = \frac{1}{2}\cos(\omega_0 m) \tag{6.158}$$

which has maxima for $m = lT_0$, the pitch period and its harmonics, so that we can find the pitch period by computing the highest value of the autocorrelation. Similarly, it can be shown that any WSS periodic process $\mathbf{x}[n]$ with period T_0 also has an autocorrelation $R[m]$ which exhibits its maxima at $m = lT_0$.

In practice, we need to obtain an estimate $\hat{R}[m]$ from knowledge of only N samples. If we use a window $w[n]$ of length N on $\mathbf{x}[n]$ and assume it to be real, the empirical autocorrelation function is given by

$$\hat{R}[m] = \frac{1}{N} \sum_{n=0}^{N-1-|m|} w[n]\mathbf{x}[n]w[n+|m|]\mathbf{x}[n+|m|] \tag{6.159}$$

whose expected value can be shown to be

$$E\left\{\hat{R}[m]\right\} = R[m]\left(w[m] * w[-m]\right) \tag{6.160}$$

where

$$w[m] * w[-m] = \sum_{n=0}^{N-|m|-1} w[n]w[n+|m|] \tag{6.161}$$

which, for the case of a rectangular window of length N, is given by

$$w[m] * w[-m] = \begin{cases} 1 - \dfrac{|m|}{N} & |m| < N \\ 0 & |m| \geq N \end{cases} \tag{6.162}$$

which means that $\hat{R}[m]$ is a biased estimator of $R[m]$. So, if we compute the peaks based on Eq. (6.159), the estimate of the pitch will also be biased. Although the variance of the estimate is difficult to compute, it is easy to see that as m approaches N, fewer and fewer samples of $x[n]$ are involved in the calculation, and thus the variance of the estimate is expected to increase. If we multiply Eq. (6.159) by $N/(N-m)$, the estimate will be unbiased but the variance will be larger.

Using the empirical autocorrelation in Eq. (6.159) for the random process in Eq. (6.157) results in an expected value of

$$E\left\{\hat{R}[m]\right\} = \left(1 - \frac{|m|}{N}\right)\frac{\cos(\omega_0 m)}{2}, \quad |m| < N \tag{6.163}$$

whose maximum coincides with the pitch period for $m > m_0$.

Since pitch periods can be as low as 40 Hz (for a very low-pitched male voice) or as high as 600 Hz (for a very high-pitched female or child's voice), the search for the maximum is conducted within a region. This F0 detection algorithm is illustrated in Figure 6.31 where the lag with highest autocorrelation is plotted for every frame. In order to see periodicity present in the autocorrelation, we need to use a window that contains at least two pitch periods, which, if we want to detect a 40 Hz pitch, implies 50 ms (see Figure 6.32). For window lengths so long, the assumption of stationarity starts to fail, because a pitch period at the beginning of the window can be significantly different than at the end of the window.

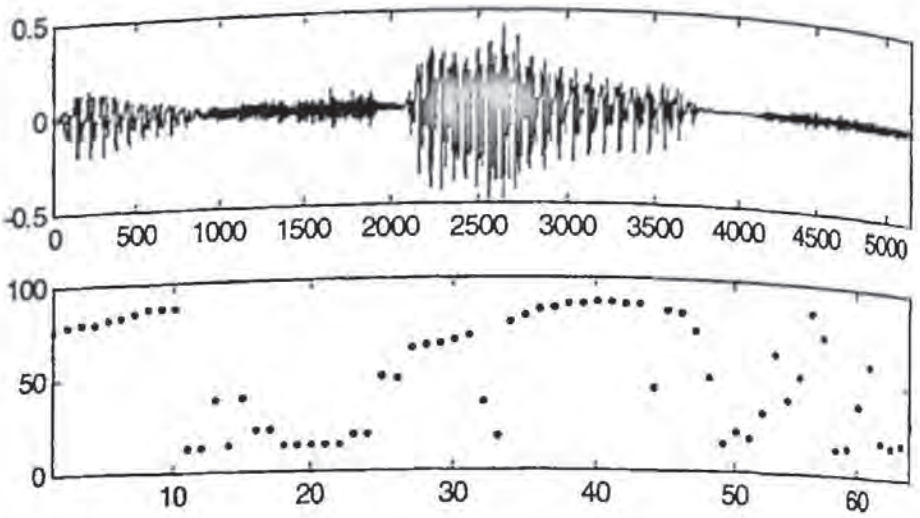


Figure 6.31 Waveform and unsmoothed pitch track with the autocorrelation method. A frame shift of 10 ms, a Hamming window of 30 ms, and a sampling rate of 8 kHz were used. Notice that two frames in the voiced region have an incorrect pitch. The pitch values in the unvoiced regions are essentially random.

One possible solution to this problem is to estimate the autocorrelation function with different window lengths for different lags m.

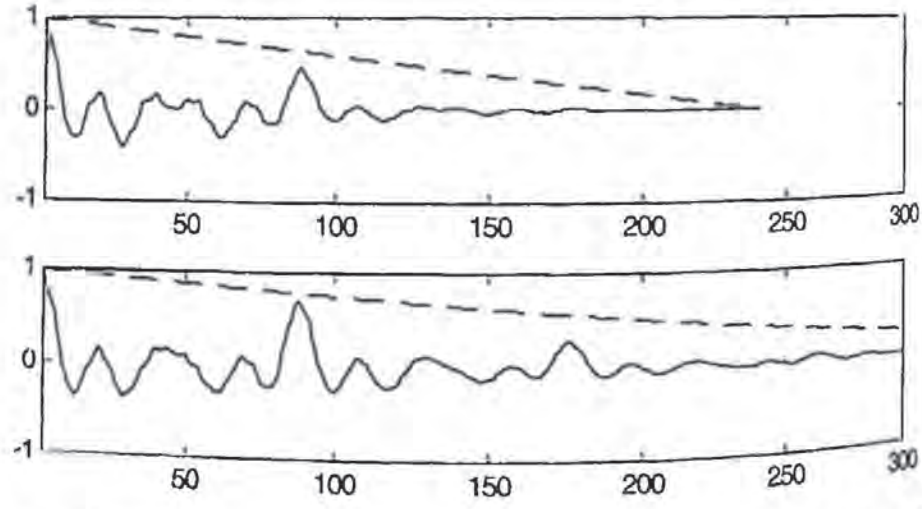


Figure 6.32 Autocorrelation function for frame 40 in Figure 6.31. The maximum occurs at 89 samples. A sampling frequency of 8 kHz and window shift of 10 ms are used. The top figure is using a window length of 30 ms, whereas the bottom one is using 50 ms. Notice the quasi-periodicity in the autocorrelation function.

The candidate pitch periods in Eq. (6.156) can be simply $T_m = m$; i.e., the pitch period is any integer number of samples. For low values of T_m, the frequency resolution is lower than for high values. To maintain a relatively constant frequency resolution, we do not have to search all the pitch periods for large T_m. Alternatively, if the sampling frequency is not high, we may need to use fractional pitch periods (often done in the speech coding algorithms of Chapter 7).

The autocorrelation function can be efficiently computed by taking a signal, windowing it, and taking an FFT and then the square of the magnitude.

6.7.2. Normalized Cross-Correlation Method

A method that is free from these border problems and has been gaining in popularity is based on the *normalized cross-correlation* [2]

$$\alpha_t(T) = \cos(\theta) = \frac{< \mathbf{x}_t, \mathbf{x}_{t-T} >}{|\mathbf{x}_t||\mathbf{x}_{t-T}|} \tag{6.164}$$

where $\mathbf{x}_t = \{x[t-N/2], x[t-N/2+1], \cdots, x[t+N/2-1]\}$ is a vector of N samples centered at time t, and $< \mathbf{x}_t, \mathbf{x}_{t-T} >$ is the inner product between the two vectors defined as

$$< \mathbf{x}_n, \mathbf{y}_l > \sum_{m=-N/2}^{N/2-1} x[n+m]y[l+m] \tag{6.165}$$

so that, using Eq. (6.165), the normalized cross-correlation can be expressed as

$$\alpha_t(T) = \frac{\sum_{n=-N/2}^{N/2-1} x[t+n]x[t+n-T]}{\sqrt{\sum_{n=-N/2}^{N/2-1} x^2[t+n] \sum_{m=-N/2}^{N/2-1} x^2[t+m+T]}} \tag{6.166}$$

where we see that the numerator in Eq. (6.166) is very similar to the autocorrelation in Section 6.7.1, but where N terms are used in the summation for all values of T.

The maximum of the normalized cross-correlation method is shown in Figure 6.33 (b). Unlike the autocorrelation method, the estimate of the normalized cross-correlation is not biased by the term $(1-m/N)$. For perfectly periodic signals, this results in identical values of the normalized cross-correlation function for kT. This can result in pitch halving, where $2T$ can be chosen as the pitch period, which happens in Figure 6.33 (b) at the beginning of the utterance. Using a decaying bias $(1-m/M)$ with $M \gg N$, can be useful in reducing pitch halving, as we see in Figure 6.33 (c).

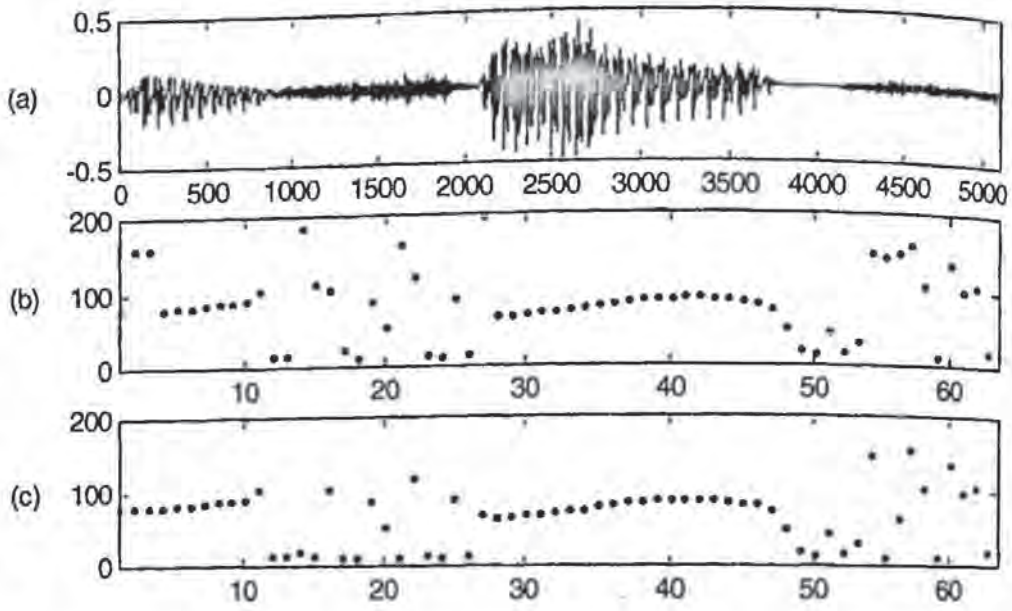


Figure 6.33 (a) Waveform and (b, c) unsmoothed pitch tracks with the normalized cross-correlation method. A frame shift of 10 ms, window length of 10 ms, and sampling rate of 8 kHz were used. (b) is the standard normalized cross-correlation method, whereas (c) has a decaying term. If we compare it to the autocorrelation method of Figure 6.31, the middle voiced region is correctly identified in both (b) and (c), but two frames at the beginning of (b) that have pitch halving are eliminated with the decaying term. Again, the pitch values in the unvoiced regions are essentially random.

Because the number of samples involved in the calculation is constant, this estimate is unbiased and has lower variance than that of the autocorrelation. Unlike the autocorrelation method, the window length could be lower than the pitch period, so that the assumption of stationarity is more accurate and it has more time resolution. While pitch trackers based on the normalized cross-correlation typically perform better than those based on the autocorrelation, they also require more computation, since all the autocorrelation lags can be efficiently computed through 2 FFTs and N multiplies and adds (see Section 5.3.4).

Let's gain some insight about the normalized cross-correlation. If $x[n]$ is periodic with period T, then we can predict it from a vector T samples in the past as:

$$\mathbf{x}_t = \rho \mathbf{x}_{t-T} + \mathbf{e}_t \tag{6.167}$$

where ρ is the prediction gain. The normalized cross-correlation measures the angle between the two vectors, as can be seen in Figure 6.34, and since it is a cosine, it has the property that $-1 \le \alpha_n(P) \le 1$.

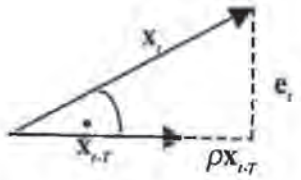


Figure 6.34 The prediction of $\mathbf{x}_t$ with $\mathbf{x}_{t-T}$ results in an error $\mathbf{e}_t$.

If we choose the value of the prediction gain ρ so as to minimize the prediction error

$$|\mathbf{e}_t|^2 = |\mathbf{x}_t|^2 - |\mathbf{x}_t|^2 \cos^2(\theta) = |\mathbf{x}_t|^2 - |\mathbf{x}_t|^2 \alpha_t^2(T) \tag{6.168}$$

and assume $\mathbf{e}_t$ is a zero-mean Gaussian random vector with a standard deviation $\sigma|\mathbf{x}_t|$, then

$$\ln f(\mathbf{x}_t \mid T) = K + \frac{\alpha_t^2(T)}{2\sigma^2} \tag{6.169}$$

so that the maximum likelihood estimate corresponds to finding the value T with highest normalized cross-correlation. Using Eq. (6.166), it is possible that $\alpha_t(T) < 0$. In this case, there is negative correlation between $\mathbf{x}_t$ and $\mathbf{x}_{t-T}$, and it is unlikely that T is a good choice for pitch. Thus, we need to force $\rho > 0$, so that Eq. (6.169) is converted into

$$\ln f(\mathbf{x}_t \mid T) = K + \frac{\left(\max(0, \alpha_t(T))\right)^2}{2\sigma^2} \tag{6.170}$$

The normalized cross-correlation of Eq. (6.164) predicts the current frame with a frame that occurs T samples before. Voiced speech may exhibit low correlation with a previous frame at a spectral discontinuity, such as those appearing at stops. To account for this, an enhancement can be done to consider not only the *backward* normalized cross-correlation, but also the *forward* normalized cross-correlation, by looking at a frame that occurs T samples ahead of the current frame, and taking the highest of both.

$$\ln f(\mathbf{x}_t \mid T) = K + \frac{\left(\max(0, \alpha_t(T), \alpha_t(-T))\right)^2}{2\sigma^2} \tag{6.171}$$

6.7.3. Signal Conditioning

Noise in the signal tends to make pitch estimation less accurate. To reduce this effect, signal conditioning or pre-processing has been proposed prior to pitch estimation [44]. Typically

this involves bandpass filtering to remove frequencies above 1 or 2 kHz, and below 100 Hz or so. High frequencies do not have much voicing information and have significant noise energy, whereas low frequencies can have 50/60 Hz interference from power lines or nonlinearities from some A/D subsystems that can also mislead a pitch estimation algorithm.

In addition to the noise in the very low frequencies and aspiration at high bands, the stationarity assumption is not as valid at high frequencies. Even a slowly changing pitch, say, nominal 100 Hz increasing 5 Hz in 10 ms, results in a fast-changing harmonic: the 30th harmonic at 3000 Hz changes 150 Hz in 10 ms. The corresponding short-time spectrum no longer shows peaks at those frequencies.

Because of this, it is advantageous to filter out such frequencies prior to the computation of the autocorrelation or normalized cross-correlation. If an FFT is used to compute the autocorrelation, this filter is easily done by setting to 0 the undesired frequency bins.

6.7.4. Pitch Tracking

Pitch tracking using the above methods typically fails in several cases:

- *Sub-harmonic errors.* If a signal is periodic with period T, it is also periodic with period $2T$, $3T$, etc. Thus, we expect the scores also to be high for the multiples of T, which can mislead the algorithm. Because the signal is never perfectly stationary, those multiples, or sub-harmonics, tend to have slightly lower scores than the fundamental. If the pitch is identified as $2T$, pitch halving is said to occur.
- *Harmonic errors.* If harmonic M dominates the signal's total energy, the score at pitch period T/M will be large. This can happen if the harmonic falls in a formant frequency that boosts its amplitude considerably compared to that of the other harmonics. If the pitch is identified as $T/2$, pitch doubling is said to occur.
- *Noisy conditions.* When the SNR is low, pitch estimates are quite unreliable for most methods.
- *Vocal fry.* While pitch is generally continuous, for some speakers it can suddenly change and even halve, particularly at the end of an unstressed voiced region. The pitch here is really not well defined and imposing smoothness constraints can hurt the system.
- *F0 jumps* up or down by an octave occasionally.
- *Breathy-voiced speech* is difficult to distinguish from periodic background noise.
- *Narrow-band filtering* of unvoiced excitations by certain vocal tract configurations can lead to signals that appear periodic.

For these reasons, pitch trackers do not determine the pitch value at frame m based exclusively on the signal at that frame. For a frame where there are several pitch candidates with similar scores, the fact that pitch does not change abruptly with time is beneficial in disambiguation, because the following frame possibly has a clearer pitch candidate, which can help.

To integrate the normalized cross-correlation into a probabilistic framework, you can combine tracking with the use of a priori information [10]. Let's define $\mathbf{X}=\{\mathbf{x}_0,\mathbf{x}_1,\ldots,\mathbf{x}_{M-1}\}$ as a sequence of input vectors for M consecutive frames centered at equally spaced time instants, say 10 ms. Furthermore, if we assume that the $\mathbf{x}_i$ are independent of each other, the joint distribution takes on the form:

$$f(\mathbf{X}\,|\,\mathbf{T})=\prod_{i=0}^{M-1} f(\mathbf{x}_i\,|\,T_i) \tag{6.172}$$

where $\mathbf{T}=\{T_0,T_1,\ldots,T_{M-1}\}$ is the pitch track for the input. The *maximum a posteriori* (MAP) estimate of the pitch track is:

$$\mathbf{T}_{MAP}=\max_{\mathbf{T}} f(\mathbf{T}\,|\,\mathbf{X})=\max_{\mathbf{T}}\frac{f(\mathbf{T})f(\mathbf{X}\,|\,\mathbf{T})}{f(\mathbf{X})}=\max_{\mathbf{T}} f(\mathbf{T})f(\mathbf{X}\,|\,\mathbf{T}) \tag{6.173}$$

according to Bayes' rule, with the term $f(\mathbf{X}\,|\,\mathbf{T})$ being given by Eq. (6.172) and $f(\mathbf{x}_i\,|\,T_i)$ by Eq. (6.169), for example.

The function $f(\mathbf{T})$ constitutes the a priori statistics for the pitch and can help disambiguate the pitch, by avoiding pitch doubling or halving given knowledge of the speaker's average pitch, and by avoiding rapid transitions given a model of how pitch changes over time. One possible approximation is given by assuming that the a priori probability of the pitch period at frame i depends only on the pitch period for the previous frame:

$$f(\mathbf{T})=f(T_0,T_1,\ldots,T_{M-1})=f(T_{M-1}\,|\,T_{M-2})f(T_{M-2}\,|\,T_{M-3})\cdots f(T_1\,|\,T_0)f(T_0) \tag{6.174}$$

One possible choice for $f(T_t\,|\,T_{t-1})$ is to decompose it into a component that depends on T_t and another that depends on the difference (T_t-T_{t-1}). If we approximate both as Gaussian densities, we obtain

$$\ln f(T_t\,|\,T_{t-1})=K'-\frac{(T_t-\mu)^2}{2\beta^2}-\frac{(T_t-T_{t-1}-\delta)^2}{2\gamma^2} \tag{6.175}$$

so that when Eqs. (6.170) and (6.175) are combined, the log-probability of transitioning to T_i at time t from pitch T_j at time $t-1$ is given by

$$S_t(T_i,T_j)=\frac{\left(\max(0,\alpha_t(T_i))\right)^2}{2\sigma^2}-\frac{(T_i-\mu)^2}{2\beta^2}-\frac{(T_i-T_j-\delta)^2}{2\gamma^2} \tag{6.176}$$

so that the log-likelihood in Eq. (6.173) can be expressed as

$$\ln f(\mathbf{T})f(\mathbf{X}\mid\mathbf{T})=\left(\max(0.\alpha_0(T_0))\right)^2+\max_{i_t}\sum_{t=1}^{M-1}S_t(T_{i_t},T_{i_{t+1}}) \tag{6.177}$$

which can be maximized through dynamic programming. For a region where pitch is not supposed to change, $\delta=0$, the term $(T_i-T_j)^2$ in Eq. (6.176) acts as a penalty that keeps the pitch track from jumping around. A mixture of Gaussians can be used instead to model different rates of pitch change, as in the case of Mandarin Chinese with four tones characterized by different slopes. The term $(T_i-\mu)^2$ attempts to get the pitch close to its expected value to avoid pitch doubling or halving, with the average μ being different for male and female speakers. Pruning can be done during the search without loss of accuracy (see Chapter 12).

Pitch trackers also have to determine whether a region of speech is voiced or unvoiced. A good approach is to build a statistical classifier with techniques described in Chapter 8 based on energy and the normalized cross-correlation described above. Such classifiers, i.e., an HMM, penalize jumps between voiced and unvoiced frames to avoid voiced regions having isolated unvoiced frames inside and vice versa. A threshold can be used on the a posteriori probability to distinguish voiced from unvoiced frames.

6.8. HISTORICAL PERSPECTIVE AND FURTHER READING

In 1978, Lawrence R. Rabiner and Ronald W. Schafer [38] wrote a book summarizing the work to date on digital processing of speech, which remains a good source for the reader interested in further reading in the field. The book by Deller, Hansen, and Proakis [9] includes more recent work and is also an excellent reference. O'Shaughnessy [33] also has a thorough description of the subject. Malvar [25] covers filterbanks and lapped transforms extensively.

The extensive wartime interest in sound spectrography led Koenig and his colleagues at Bell Laboratories [22] in 1946 to the invaluable development of a tool that has been used for speech analysis since then: the spectrogram. Potter et al. [35] showed the usefulness of the analog spectrogram in analyzing speech. The spectrogram facilitated research in the field and led Peterson and Barney [34] to publish in 1952 a detailed study of formant values of different vowels. The development of computers and the FFT led Oppenheim, in 1970 [30], to develop digital spectrograms, which imitated the analog counterparts.

The MIT Acoustics Lab started work in speech in 1948 with Leo R. Beranek, who in 1954 published the seminal book *Acoustics*, where he studied sound propagation in tubes. In

1950, Kenneth N. Stevens joined the lab and started work on speech perception. Gunnar Fant visited the lab at that time and as a result started a strong speech production effort at KTH in Sweden.

The 1960s marked the birth of digital speech processing. Two books, Gunnar Fant's *Acoustical Theory of Speech Production* [13] in 1960 and James Flanagan's *Speech Analysis: Synthesis and Perception* [14] in 1965, had a great impact and sparked interest in the field. The advent of the digital computer prompted Kelly and Gertsman to create in 1961 the first digital speech synthesizer [21]. Short-time Fourier analysis, cepstrum, LPC analysis, and pitch and formant tracking were the fruit of that decade.

Short-time frequency analysis was first proposed for analog signals by Fano [11] in 1950 and later by Schroeder and Atal [42].

The mathematical foundation behind linear predictive coding dates to the auto-regressive models of George Udny Yule (1927) and Gilbert Walker (1931), which led to the well-known Yule-Walker equations. These equations resulted in a Toeplitz matrix, named after Otto Toeplitz (1881-1940) who studied it extensively. N. Levinson suggested in 1947 an efficient algorithm to invert such a matrix, which J. Durbin refined in 1960 and is now known as the Levinson-Durbin recursion. The well-known LPC analysis consisted of the application of the above results to speech signals, as developed by Bishnu Atal [4], J. Burg [7], Fumitada Itakura and S. Saito [19] in 1968, and Markel [27] and John Makhoul [24] in 1973.

The cepstrum was first proposed in 1964 by Bogert, Healy, and John Tukey [6] and further studied by Alan V. Oppenheim [29] in 1965. The popular mel-frequency cepstrum was proposed by Davis and Mermelstein [8] in 1980, combining the advantages of cepstrum with knowledge of the non-linear perception of frequency by the human auditory system that had been studied by E. Zwicker [47] in 1961.

Formant tracking was first investigated by Ken Stevens and James Flanagan in the late 1950s, with the foundations for most modern techniques being developed by Schafer and Rabiner [40], Itakura [20], and Markel [26]. Pitch tracking through digital processing was first studied by B. Gold [15] in 1962 and then improved by A. M. Noll [28], M. Schroeder [41], and M. Sondhi [44] in the late 1960s.

REFERENCES

[1] Acero, A., "Formant Analysis and Synthesis Using Hidden Markov Models," *Eurospeech*, 1999, Budapest pp. 1047-1050.

[2] Atal, B.S., *Automatic Speaker Recognition Based on Pitch Contours*, PhD Thesis, 1968, Polytechnic Institute of Brooklyn.

[3] Atal, B.S. and L. Hanauer, "Speech Analysis and Synthesis by Linear Prediction of the Speech Wave," *Journal of the Acoustical Society of America*, 1971, **50**, pp. 637-655.

[4] Atal, B.S. and M.R. Schroeder, "Predictive Coding of Speech Signals," *Report of the 6th Int. Congress on Acoustics*, 1968, Tokyo, Japan.

[5] Berouti, M.G., D.G. Childers, and A. Paige, "Glottal Area versus Glottal Volume Velocity," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1977, Hartford, Conn, pp. 33-36.

[6] Bogert, B., M. Healy, and J. Tukey, "The Quefrency Alanysis of Time Series for Echoes," *Proc. Symp. on Time Series Analysis*, 1963, New York, J. Wiley, pp. 209-243.

[7] Burg, J., "Maximum Entropy Spectral Analysis," *Proc. of the 37th Meeting of the Society of Exploration Geophysicists*, 1967.

[8] Davis, S. and P. Mermelstein, "Comparison of Parametric Representations for Monosyllable Word Recognition in Continuously Spoken Sentences," *IEEE Trans. on Acoustics, Speech and Signal Processing*, 1980, **28**(4), pp. 357-366.

[9] Deller, J.R., J.H.L. Hansen, and J.G. Proakis, *Discrete-Time Processing of Speech Signals*, 2000, IEEE Press.

[10] Droppo, J. and A. Acero, "Maximum a Posteriori Pitch Tracking," *Int. Conf. on Spoken Language Processing*, 1998, Sydney, Australia, pp. 943-946.

[11] Fano, R.M., "Short-time Autocorrelation Functions and Power Spectra," *Journal of the Acoustical Society of America*, 1950, **22**(Sep), pp. 546-550.

[12] Fant, G., "On the Predictability of Formant Levels and Spectrum Envelopes from Formant Frequencies" in *For Roman Jakobson*, M. Halle, ed., 1956, The Hague, NL, Mouton & Co., pp. 109-120.

[13] Fant, G., *Acoustic Theory of Speech Production*, 1970, The Hague, NL, Mouton.

[14] Flanagan, J., *Speech Analysis Synthesis and Perception*, 1972, New York, Springer-Verlag.

[15] Gold, B., "Computer Program for Pitch Extraction," *Journal of the Acoustical Society of America*, 1962, **34**(7), pp. 916-921.

[16] Hermansky, H., "Perceptual Linear Predictive (PLP) Analysis of Speech," *Journal of the Acoustical Society of America*, 1990, **87**(4), pp. 1738-1752.

[17] Hess, W., *Pitch Determination of Speech Signals*, 1983, New York, Springer-Verlag.

[18] Itakura, F., "Line Spectrum Representation of Linear Predictive Coefficients," *Journal of the Acoustical Society of America*, 1975, **57**(4), pp. 535.

[19] Itakura, F. and S. Saito, "Analysis Synthesis Telephony Based on the Maximum Likelihood Method," *Proc. 6th Int. Congress on Acoustics*, 1968, Tokyo, Japan.

[20] Itakura, F. and S. Saito, "A Statistical Method for Estimation of Speech Spectral Density and Formant Frequencies," *Elec. and Comm. in Japan*, 1970, **53-A**(1), pp. 36-43.

[21] Kelly, J.L. and L.J. Gerstman, "An Artificial Talker Driven From Phonetic Input," *Journal of Acoustical Society of America*, 1961, **33**, pp. 835.

[22] Koenig, R., H.K. Dunn, and L.Y. Lacy, "The Sound Spectrograph," *Journal of the Acoustical Society of America*, 1946, **18**, pp. 19-49.

[23] Krishnamurthy, A.K. and D.G. Childers, "Two Channel Speech Analysis," *IEEE Trans. on Acoustics, Speech and Signal Processing*, 1986, **34**, pp. 730-743.

[24] Makhoul, J., "Spectral Analysis of Speech by Linear Prediction," *IEEE Trans. on Acoustics, Speech and Signal Processing*, 1973, **21**(3), pp. 140-148.

[25] Malvar, H., *Signal Processing with Lapped Transforms*, 1992, Artech House.

[26] Markel, J.D., "Digital Inverse Filtering—A New Tool for Formant Trajectory Estimation," *IEEE Trans. on Audio and Electroacoustics*, 1972, **AU-20**(June), pp. 129-137.

[27] Markel, J.D. and A.H. Gray, "On Autocorrelation Equations as Applied to Speech Analysis," *IEEE Trans. on Audio and Electroacoustics*, 1973, **AU-21**(April), pp. 69-79.

[28] Noll, A.M., "Cepstrum Pitch Determination," *Journal of the Acoustical Society of America*, 1967, **41**, pp. 293-309.

[29] Oppenheim, A.V., *Superposition in a Class of Nonlinear Systems*, 1965, Research Lab. of Electronics, MIT, Cambridge, Massachusetts.

[30] Oppenheim, A.V., "Speech Spectrograms Using the Fast Fourier Transform," *IEEE Spectrum*, 1970, 7(Aug), pp. 57-62.

[31] Oppenheim, A.V. and D.H. Johnson, "Discrete Representation of Signals," *The Proc. of the IEEE*, 1972, **60**(June), pp. 681-691.

[32] Oppenheim, A.V., R.W. Schafer, and T.G. Stockham, "Nonlinear Filtering of Multiplied and Convolved Signals," *Proc. of the IEEE*, 1968, **56**, pp. 1264-1291.

[33] O'Shaughnessy, D., *Speech Communication—Human and Machine*, 1987, Addison-Wesley.

[34] Peterson, G.E. and H.L. Barney, "Control Methods Used in a Study of the Vowels," *Journal of the Acoustical Society of America*, 1952, 24(2), pp. 175-184.

[35] Potter, R.K., G.A. Kopp, and H.C. Green, *Visible Speech*, 1947, New York, D. Van Nostrand Co. Republished by Dover Publications, Inc., 1966.

[36] Press, W.H., *et al.*, *Numerical Recipes in C*, 1988, New York, NY, Cambridge University Press.

[37] Rabiner, L.R., "On the Use of Autocorrelation Analysis for Pitch Detection," *IEEE Trans. on Acoustics, Speech and Signal Processing*, 1977, **25**, pp. 24-33.

[38] Rabiner, L.R. and R.W. Schafer, *Digital Processing of Speech Signals*, 1978, Englewood Cliffs, NJ, Prentice-Hall.

[39] Rosenberg, A.E., "Effect of Glottal Pulse Shape on the Quality of Natural Vowels," *Journal of the Acoustical Society of America*, 1971, **49**, pp. 583-590.

[40] Schafer, R.W. and L.R. Rabiner, "System for Automatic Formant Analysis of Voiced Speech," *Journal of the Acoustical Society of America*, 1970, **47**, pp. 634-678.

[41] Schroeder, M., "Period Histogram and Product Spectrum: New Methods for Fundamental Frequency Measurement," *Journal of the Acoustical Society of America*, 1968, **43**(4), pp. 829-834.

[42] Schroeder, M.R. and B.S. Atal, "Generalized Short-Time Power Spectra and Autocorrelation," *Journal of the Acoustical Society of America*, 1962, **34**(Nov), pp. 1679-1683.

[43] Shikano, K., K.-F. Lee, and R. Reddy, "Speaker Adaptation through Vector Quantization," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1986, Tokyo, Japan, pp. 2643-2646.

[44] Sondhi, M.M., "New Methods for Pitch Extraction," *IEEE Trans. on Audio and Electroacoustics*, 1968, **16**(June), pp. 262-268.

[45] Talkin, D., "A Robust Algorithm for Pitch Tracking" in *Speech Coding and Synthesis*, W.B. Kleijn and K.K. Paliwal, eds., 1995, Amsterdam, Elsevier, pp. 485-518, .

[46] Yegnanarayana, B. and R.N.J. Veldhuis, "Extraction of Vocal-Tract System Characteristics from Speech Signals," *IEEE Trans. on Speech and Audio Processing*, 1998, **6**(July), pp. 313-327.

[47] Zwicker, E., "Subdivision of the Audible Frequency Range into Critical Bands," *Journal of the Acoustical Society of America*, 1961, **33**(Feb), p. 248.

CHAPTER 7

Speech Coding

Transmission of speech using data networks requires the speech signal to be digitally encoded. Voice over IP has become very popular because of the Internet, where bandwidth limitations make it necessary to compress the speech signal. Digital storage of audio signals, which can result in higher quality and smaller size than the analog counterpart, is commonplace in compact discs, digital video discs, and MP3 files. Many spoken language systems also use coded speech for efficient communication. For these reasons we devote a chapter to speech and audio coding techniques.

Rather than exhaustively cover all the existing speech and audio coding algorithms, we uncover their underlying technology and enumerate some of the most popular standards. The coding technology discussed in this chapter has a strong link to both speech recognition and speech synthesis. For example, the speech synthesis algorithms described in Chapter 16 use many techniques described here.

7.1. SPEECH CODERS ATTRIBUTES

How do we compare different speech or audio coders? We can refer to a number of factors, such as signal bandwidth, bit rate, quality of reconstructed speech, noise robustness, computational complexity, delay, channel-error sensitivity, and standards.

Speech signals can be bandlimited to 10 kHz without significantly affecting the hearer's perception. The telephone network limits the bandwidth of speech signals to between 300 and 3400 Hz, which gives *telephone speech* a lower quality. Telephone speech is typically sampled at 8 kHz. The term *wideband speech* is used for a bandwidth of 50–7000 Hz and a sampling rate of 16 kHz. Finally, *audio coding* is used in dealing with high-fidelity audio signals, in which case the signal is sampled at 44.1 kHz.

Reduction in bit rate is the primary purpose of speech coding. The previous bit stream can be compressed to a lower rate by removing redundancy in the signal, resulting in savings in storage and transmission bandwidth. If only redundancy is removed, the original signal can be recovered exactly (*lossless* compression). In *lossy* compression, the signal cannot be recovered exactly, though hopefully it will sound similar to the original.

Depending on system and design constraints, fixed-rate or variable-rate speech coders can be used. Variable-rate coders are used for non-real time applications, such as voice storage (silence can be coded with fewer bits than fricatives, which in turn use fewer bits than vowels), or for packet voice transmissions, such as CDMA cellular for better channel utilization. Transmission of coded speech through a noisy channel may require devoting more bits to channel coding and fewer to source coding. For most real-time communication systems, a maximum bit rate is specified.

The quality of the reconstructed speech signal is a fundamental attribute of a speech coder. Bit rate and quality are intimately related: the lower the bit rate, the lower the quality. While the bit rate is inherently a number, it is difficult to quantify the quality. The most widely used measure of quality is the *Mean Opinion Score* (MOS) [25], which is the result of averaging opinion scores for a set of between 20 and 60 untrained subjects. Each listener characterizes each set of utterances with a score on a scale from 1 (unacceptable quality) to 5 (excellent quality), as shown in Table 7.1. An MOS of 4.0 or higher defines *good* or *toll* quality, where the reconstructed speech signal is generally indistinguishable from the original signal. An MOS between 3.5 and 4.0 defines *communication* quality, which is sufficient for telephone communications. We show in Section 7.2.1 that if each sample is quantized with 16 bits, the resulting signal has *toll* quality (essentially indistinguishable from the unquantized signal). See Chapter 16 for more details on perceptual quality measurements.

Table 7.1 Mean Opinion Score (MOS) is a numeric value computed as an average for a number of subjects, where each number maps to the above subjective quality.

Excellent	Good	Fair	Poor	Bad
5	4	3	2	1

Another measure of quality is the *signal-to-noise ratio* (SNR), defined as the ratio between the signal's energy and the noise's energy in terms of dB:

$$SNR = \frac{\sigma_x^2}{\sigma_e^2} = \frac{E\{x^2[n]\}}{E\{e^2[n]\}} \tag{7.1}$$

The MOS rating of a codec on noise-free speech is often higher than its MOS rating for noisy speech. This is generally caused by specific assumptions in the speech coder that tend to be violated when a significant amount of noise is present in the signal. This phenomenon is more accentuated for lower-bit-rate coders that need to make more assumptions.

The computational complexity and memory requirements of a speech coder determine the cost and power consumption of the hardware on which it is implemented. In most cases, real-time operation is required at least for the decoder. Speech coders can be implemented in inexpensive *Digital Signal Processors* (DSP) that form part of many consumer devices, such as answering machines and DVD players, for which storage tends to be relatively more expensive than processing power. DSPs are also used in cellular phones because bit rates are limited.

All speech coders have some delay, which, if excessive, can affect the dynamics of a two-way communication. For instance, delays over 150 ms can be unacceptable for highly interactive conversations. Coder delay is the sum of different types of delay. The first is the *algorithmic delay* arising because speech coders usually operate on a block of samples, called a *frame,* which needs to be accumulated before processing can begin. Often the speech coder requires some additional *look-ahead* beyond the frame to be encoded. The *computational delay* is the time that the speech coder takes to process the frame. For real-time operation, the computational delay has to be smaller than the algorithmic delay. A block of bits is generally assembled by the encoder prior to transmission, possibly to add error-correction properties to the bit stream, which cause *multiplexing delay.* Finally, there is the *transmission delay,* due to the time it takes for the frame to traverse the channel. The decoder will incur a *decoder delay* to reconstruct the signal. In practice, the total delay of many speech coders is at least three frames.

If the coded speech needs to be transmitted over a channel, we need to consider possible channel errors, and our speech decoder should be insensitive to at least some of them. There are two types of errors: random errors and burst errors, and they could be handled differently. One possibility to increase the robustness against such errors is to use channel coding techniques, such as those proposed in Chapter 3. Joint source and channel coding allows us to find the right combination of bits to devote to speech coding with the right amount devoted to channel coding, adjusting this ratio adaptively depending on the channel. Since channel coding will only reduce the number of errors, and not eliminate them, graceful degradation of speech quality under channel errors is typically a design factor for speech coders. When the channel is the Internet, complete frames may be missing because they have not arrived in time. Therefore, we need techniques that degrade gracefully with missing frames.

7.2. SCALAR WAVEFORM CODERS

In this section we describe several waveform coding techniques, such as linear PCM, μ-law, and A-law PCM, APCM, DPCM, DM, and ADPCM, that quantize each sample using scalar quantization. These techniques attempt to approximate the waveform, and, if a large enough bit rate is available, will get arbitrarily close to it.

7.2.1. Linear Pulse Code Modulation (PCM)

Analog-to-digital converters perform both sampling and quantization simultaneously. To better understand how this process affects the signal it's better to study them separately. We analyzed the effects of sampling in Chapter 5, so now we analyze the effects of quantization, which encodes each sample with a fixed number of bits. With B bits, it is possible to represent 2^B separate quantization levels. The output of the quantizer $\hat{x}[n]$ is given by

$$\hat{x}[n] = Q\{x[n]\} \tag{7.2}$$

Linear *Pulse Code Modulation* (PCM) is based on the assumption that the input discrete signal $x[n]$ is bounded

$$|x[n]| \leq X_{max} \tag{7.3}$$

and that we use *uniform quantization* with quantization step size Δ which is constant for all levels x_l

$$x_l - x_{l-1} = \Delta \tag{7.4}$$

The input/output characteristics are shown by Figure 7.1 for the case of a 3-bit uniform quantizer. The so-called *mid-riser* quantizer has the same number of positive and negative levels, whereas the *mid-tread* quantizer has one more negative than positive levels. The code $c[n]$ is expressed in two's complement representation, which for Figure 7.1 varies between –4 and +3. For the mid-riser quantizer the output $\hat{x}[n]$ can be obtained from the code $c[n]$ through

$$\hat{x}[n] = sign(c[n])\frac{\Delta}{2} + c[n]\Delta \tag{7.5}$$

and for the mid-tread quantizer

$$\hat{x}[n] = c[n]\Delta \tag{7.6}$$

which is often used in computer systems that use two's complement representation.

There are two independent parameters for a uniform quantizer: the number of levels $N = 2^B$, and the step size Δ. Assuming Eq. (7.3), we have the relationship

$$2X_{max} = \Delta 2^B \tag{7.7}$$

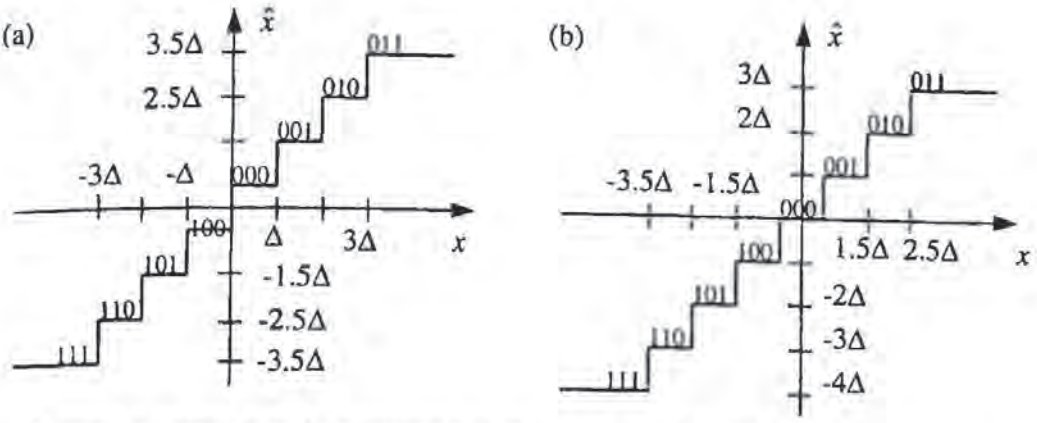


Figure 7.1 Three-bit uniform quantization characteristics: (a) mid-riser, (b) mid-tread.

In quantization, it is useful to express the relationship between the unquantized sample $x[n]$ and the quantized sample $\hat{x}[n]$ as

$$\hat{x}[n] = x[n] + e[n] \tag{7.8}$$

with $e[n]$ being the quantization noise. If we choose Δ and B to satisfy Eq. (7.7), then

$$-\frac{\Delta}{2} \le e[n] \le \frac{\Delta}{2} \tag{7.9}$$

While there is obviously a deterministic relationship between $e[n]$ and $x[n]$, it is convenient to assume a probabilistic model for the quantization noise:

1. $e[n]$ is white: $E\{e[n]e[n+m]\} = \sigma_e^2 \delta[m]$
2. $e[n]$ and $x[n]$ are uncorrelated: $E\{x[n]e[n+m]\} = 0$
3. $e[n]$ is uniformly distributed in the interval $(-\Delta/2, \Delta/2)$

These assumptions are unrealistic for some signals, except in the case of speech signals, which rapidly fluctuate between different quantization levels. The assumptions are reasonable if the step size Δ is small enough, or alternatively the number of levels is large enough (say, more than 2^6).

The variance of such uniform distribution (see Chapter 3) is

$$\sigma_e^2 = \frac{\Delta^2}{12} = \frac{X_{\max}^2}{3 \times 2^{2B}} \tag{7.10}$$

after using Eq. (7.7). The SNR is given by

$$SNR(dB) = 10 \log_{10}\left(\frac{\sigma_x^2}{\sigma_e^2}\right) = (20 \log_{10} 2)B + 10 \log_{10} 3 - 20 \log_{10}\left(\frac{X_{\max}}{\sigma_x}\right) \tag{7.11}$$

which implies that each bit contributes to 6 dB of *SNR*, since $20 \log_{10} 2 \equiv 6$.

Speech samples can be approximately described as following a *Laplacian* distribution [40]

$$p(x) = \frac{1}{\sqrt{2}\sigma_x} e^{-\frac{\sqrt{2}|x|}{\sigma_x}} \tag{7.12}$$

and the probability of x falling outside the range $(-4\sigma_x, 4\sigma_x)$ is 0.35%. Thus, using $X_{max} = 4\sigma_x$, $B = 7$ bits in Eq. (7.11) results in an *SNR* of 35 dB, which would be acceptable in a communications system. Unfortunately, signal energy can vary over 40 dB, due to variability from speaker to speaker as well as variability in transmission channels. Thus, in practice, it is generally accepted that 11 bits are needed to achieve an SNR of 35 dB while keeping the clipping to a minimum.

Digital audio stored in computers (Windows WAV, Apple AIF, Sun AU, and SND formats among others) use 16-bit linear PCM as their main format. The *Compact Disc-Digital Audio* (CD-DA or simply CD) also uses 16-bit linear PCM. Invented in the late 1960s by James T. Russell, it was launched commercially in 1982 and has become one of the most successful examples of consumer electronics technology: there were about 700 million audio CD players in 1997. A CD can store up to 74 minutes of music, so the total amount of digital data that must be stored on a CD is 44,100 samples/(channel*second) * 2 bytes/sample * 2 channels * 60 seconds/minute * 74 minutes = 783,216,000 bytes. This 747 MB are stored in a disk only 12 centimeters in diameter and 1.2 mm thick. CD-ROMs can record only 650 MB of computer data because they use the remaining bits for error correction.

7.2.2. μ-law and A-law PCM

Human perception is affected by SNR, because adding noise to a signal is not as noticeable if the signal energy is large enough. Ideally, we want SNR to be constant for all quantization levels, which requires the step size to be proportional to the signal value. This can be done by using a logarithmic *compander*[1]

$$y[n] = \ln|x[n]| \tag{7.13}$$

followed by a uniform quantizer on $y[n]$ so that

$$\hat{y}[n] = y[n] + \varepsilon[n] \tag{7.14}$$

and, thus,

$$\hat{x}[n] = \exp\{\hat{y}[n]\}\text{sign}\{x[n]\} = x[n]\exp\{\varepsilon[n]\} \tag{7.15}$$

[1] A compander is a nonlinear function that compands one part of the x-axis.

after using Eqs. (7.13) and (7.14). If $\varepsilon[n]$ is small, then Eq. (7.15) can be expressed as

$$\hat{x}[n] \equiv x[n](1+\varepsilon[n]) = x[n] + x[n]\varepsilon[n] \tag{7.16}$$

and, thus, the $SNR = 1/\sigma_\varepsilon^2$ is constant for all levels. This type of quantization is not practical, because an infinite number of quantization steps would be required. An approximation is the so-called μ-law [51]:

$$y[n] = X_{\max} \frac{\log\left[1+\mu\dfrac{|x[n]|}{X_{\max}}\right]}{\log[1+\mu]} \operatorname{sign}\{x[n]\} \tag{7.17}$$

which is approximately logarithmic for large values of $x[n]$ and approximately linear for small values of $x[n]$. A related compander called A-law is also used

$$y[n] = X_{\max} \frac{1+\log\left[\dfrac{A|x[n]|}{X_{\max}}\right]}{1+\log A} \operatorname{sign}\{x[n]\} \tag{7.18}$$

which has greater resolution than μ-law for small sample values, but a range equivalent to 12 bits. In practice, they both offer similar quality. The μ-law curve can be seen in Figure 7.2.

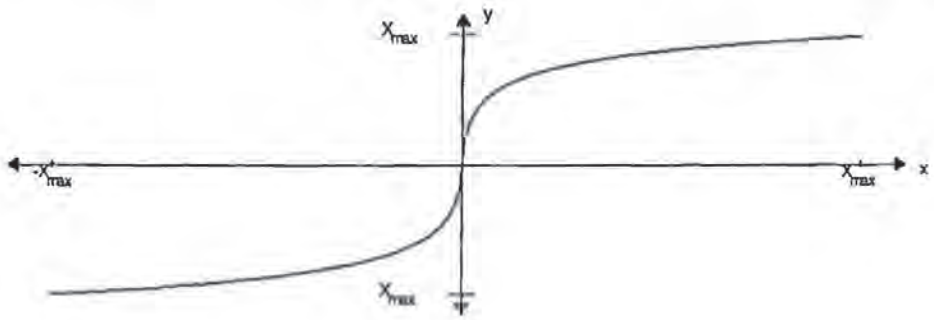


Figure 7.2 Nonlinearity used in the μ-law compression.

In 1972 the ITU-T[2] recommendation G.711 standardized telephone speech coding at 64 kbps for digital transmission of speech through telephone networks. It uses 8 bits per sample and an 8-kHz sampling rate with either μ-law or *A*-law. In North America and Japan, μ-law with $\mu = 255$ is used, whereas, in the rest of the world, *A*-law with $A = 87.56$ is used. Both compression characteristics are very similar and result in an approximate SNR of 35 dB. Without the logarithmic compressor, a uniform quantizer requires approximately 12 bits

[2] The International Telecommunication Union (ITU) is a part of the United Nations Economic, Scientific and Cultural Organization (UNESCO). ITU-T is the organization within ITU responsible for setting global telecommunication standards. Within ITU-T, Study Group 15 (SG15) is responsible for formulating speech coding standards. Prior to 1993, telecommunication standards were set by the *Comité Consultatif International Téléphonique et Télégraphique* (CCITT), which was reorganized into the ITU-T that year.

per sample to achieve the same level of quality. All the speech coders for telephone speech described in this chapter use G.711 as a baseline reference, whose quality is considered *toll*, and an MOS of about 4.3. G.711 is used by most digital central office switches, so that when you make a telephone call using your plain old telephone service (POTS), your call is encoded with G.711.

7.2.3. Adaptive PCM

When quantizing speech signals we confront a dilemma. On the one hand, we want the quantization step size to be large enough to accommodate the maximum peak-to-peak range of the signal and avoid clipping. On the other hand, we need to make the step size small to minimize the quantization noise. One possible solution is to adapt the step size to the level of the input signal.

The basic idea behind *Adaptive PCM* (APCM) is to let the step size $\Delta[n]$ be proportional to the standard deviation of the signal $\sigma[n]$:

$$\Delta[n] = \Delta_0 \sigma[n] \tag{7.19}$$

An equivalent method is to use a fixed quantizer but have a time-varying gain $G[n]$, which is inversely proportional to the signal's standard deviation

$$G[n] = G_0 / \sigma[n] \tag{7.20}$$

Estimation of the signal's variance, or short-time energy, is typically done by low-pass filtering $x^2[n]$. With a first-order IIR filter, the variance $\sigma^2[n]$ is computed as

$$\sigma^2[n] = \alpha\sigma^2[n-1] + (1-\alpha)x^2[n-1] \tag{7.21}$$

with α controlling the time constant of the filter $T = -1/(F_s \ln\alpha)$, F_s the sampling rate, and $0 < \alpha < 1$. In practice, α is chosen so that the time constant ranges between 1 ms ($\alpha = 0.88$ at 8 kHz) and 10 ms ($\alpha = 0.987$ at 8 kHz).

Alternatively, $\sigma^2[n]$ can be estimated from the past M samples:

$$\sigma^2[n] = \frac{1}{M}\sum_{m=n-M}^{n-1} x^2[m] \tag{7.22}$$

In practice, it is advantageous to set limits on the range of values of $\Delta[n]$ and $G[n]$:

$$\Delta_{min} \le \Delta[n] \le \Delta_{max} \tag{7.23}$$

$$G_{min} \le G[n] \le G_{max} \tag{7.24}$$

with the ratios $\Delta_{max}/\Delta_{min}$ and G_{max}/G_{min} determining the dynamic range of the system. If our objective is to obtain a relatively constant SNR over a range of 40 dB, these ratios can be 100.

Feedforward adaptation schemes require us to transmit, in addition to the quantized signal, either the step size $\Delta[n]$ or the gain $G[n]$. Because these values evolve slowly with time, they can be sampled and quantized at a low rate. The overall rate will be the sum of the bit rate required to transmit the quantized signal plus the bit rate required to transmit either the gain or the step size.

Another class of adaptive quantizers use *feedback adaptation* to avoid having to send information about the step size or gain. In this case, the step size and gain are estimated from the quantizer output, so that they can be recreated at the decoder without any extra information. The corresponding short-time energy can then be estimated through a first-order IIR filter as in Eq. (7.21) or a rectangular window as in Eq. (7.22), but replacing $x^2[n]$ by $\hat{x}^2[n]$.

Another option is to adapt the step size

$$\Delta[n] = P\Delta[n-1] \tag{7.25}$$

where $P > 1$ if the previous codeword corresponds to the largest positive or negative quantizer level, and $P < 1$ if the previous codeword corresponds to the smallest positive or negative quantizer level. A similar process can be done for the gain.

APCM exhibits an improvement between 4–8 dB over μ-law PCM for the same bit rate.

7.2.4. Differential Quantization

Speech coding is about finding redundancy in the signal and removing it. We know that there is considerable correlation between adjacent samples, because on the average the signal doesn't change rapidly from sample to sample. A simple way of capturing this is to quantize the difference $d[n]$ between the current sample $x[n]$ and its predicted value $\tilde{x}[n]$

$$d[n] = x[n] - \tilde{x}[n] \tag{7.26}$$

with its quantized value represented as

$$\hat{d}[n] = Q\{d[n]\} = d[n] + e[n] \tag{7.27}$$

where $e[n]$ is the quantization error. Then, the quantized signal is the sum of the predicted signal $\tilde{x}[n]$ and the quantized difference $\hat{d}[n]$

$$\hat{x}[n] = \tilde{x}[n] + \hat{d}[n] = x[n] + e[n] \tag{7.28}$$

If the prediction is good, Eq. (7.28) tells us that the quantization error will be small. Statistically, we need the variance of $e[n]$ to be lower than that of $x[n]$ for differential coding to provide any gain. Systems of this type are generically called *Differential Pulse Code Modulation* (DPCM) [11] and can be seen in Figure 7.3.

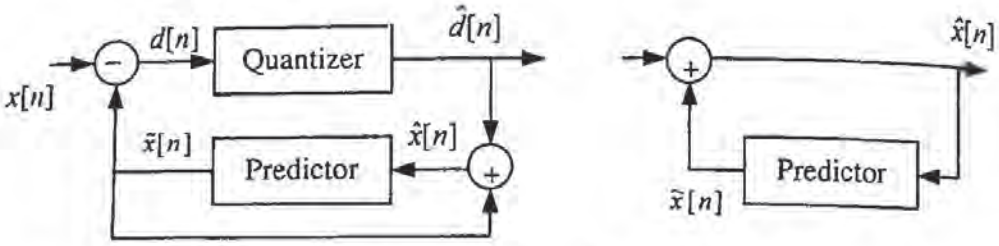


Figure 7.3 Block diagram of a DPCM encoder and decoder with feedback prediction.

Delta Modulation (DM) [47] is a 1-bit DPCM, which predicts the current sample to be the same as the past sample:

$$\tilde{x}[n] = x[n-1] \tag{7.29}$$

so that we transmit whether the current sample is above or below the previous sample.

$$d[n] = \begin{cases} \Delta & x[n] > x[n-1] \\ -\Delta & x[n] \le x[n-1] \end{cases} \tag{7.30}$$

with Δ being the step size. If Δ is too small, the reconstructed signal will not increase as fast as the original signal, a condition known as *slope overload distortion*. When the slope is small, the step size Δ also determines the peak error; this is known as *granular noise*. Both quantization errors can be seen in Figure 7.4. The choice of Δ that minimizes the mean squared error will be a tradeoff between slope overload and granular noise.

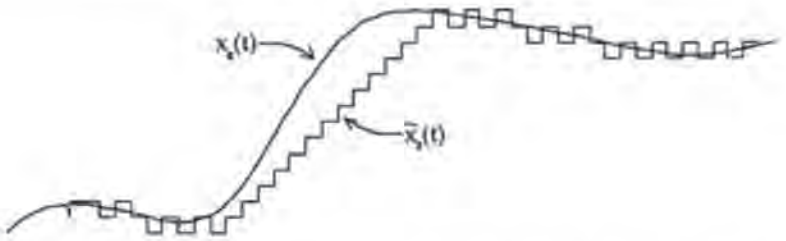


Figure 7.4 An example of slope overload distortion and granular noise in a DM encoder.

If the signal is oversampled by a factor N, and the step size is reduced by the same amount (i.e., Δ/N), the slope overload will be the same, but the granular noise will decrease by a factor N. While the coder is indeed very simple, sampling rates of over 200 kbps are needed for SNRs comparable to PCM, so DM is rarely used as a speech coder.

However, delta modulation is useful in the design of analog-digital converters, in a variant called sigma-delta modulation [44] shown in Figure 7.5. First the signal is lowpass filtered with a simple analog filter, and then it is oversampled. Whenever the predicted signal $\tilde{x}[n]$ is below the original signal $x[n]$, the difference $d[n]$ is positive. This difference $d[n]$

is averaged over time with a digital integrator whose output is $e[n]$. If this situation persists, the accumulated error $e[n]$ will exceed a positive value A, which causes a 1 to be encoded into the stream $q[n]$. A digital-analog converter is used in the loop which increments by one the value of the predicted signal $\tilde{x}[n]$. The system acts in the opposite way if the predicted signal $\tilde{x}[n]$ is above the original signal $x[n]$ for an extended period of time. Since the signal is oversampled, it changes very slowly from one sample to the next, and this quantization can be accurate. The advantages of this technique as an analog-digital converter are that inexpensive analog filters can be used and only a simple 1-bit A/D is needed. The signal can next be low-passed filtered with a more accurate digital filter and then downsampled.

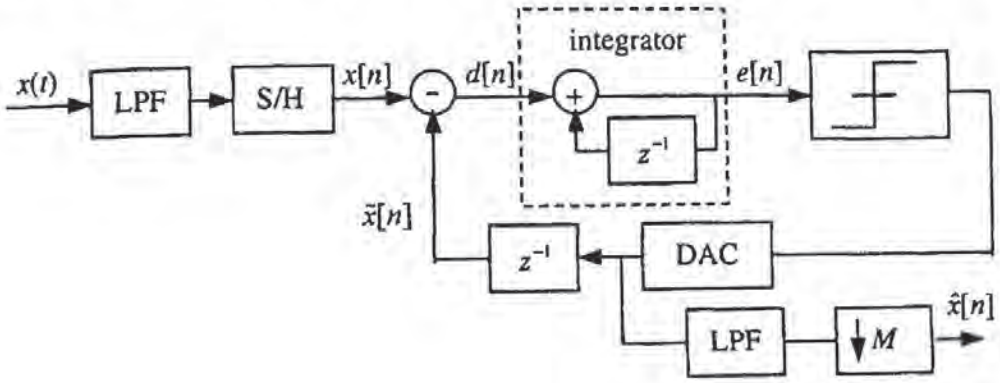


Figure 7.5 A sigma-delta modulator used in an oversampling analog-digital converter.

Adaptive Delta Modulation (ADM) combines ideas from adaptive quantization and delta modulation with the so-called *Continuously Variable Slope Delta Modulation* (CVSDM) [22] having a step size that increases

$$\Delta[n]=\begin{cases}\alpha\Delta[n-1]+k_1 & \text{if } e[n],e[n-1] \text{ and } e[n-2] \text{ have same sign}\\ \alpha\Delta[n-1]+k_2 & \text{otherwise}\end{cases} \tag{7.31}$$

with $0<\alpha<1$ and $0<k_2<<k_1$. The step size increases if the last three errors have the same sign and decreases otherwise.

Improved DPCM is achieved through linear prediction in which $\tilde{x}[n]$ is a linear combination of past quantized values $\hat{x}[n]$

$$\tilde{x}[n]=\sum_{k=1}^{p}a_k\hat{x}[n-k] \tag{7.32}$$

DPCM systems with fixed prediction coefficients can provide from 4 to 11 dB improvement over direct linear PCM, for prediction orders up to $p = 4$, at the expense of increased computational complexity. Larger improvements can be obtained by adapting the

prediction coefficients. The coefficients can be transmitted in a feedforward fashion or not transmitted if the feedback scheme is selected.

ADPCM [6] combines differential quantization with adaptive step-size quantization. ITU-T Recommendation G.726 uses ADPCM at bit rates of 40, 32, 24, and 16 kbps, with 5, 4, 3, and 2 bits per sample, respectively. It employs an adaptive feedback quantizer and an adaptive feedback pole-zero predictor. Speech at bit rates of 40 and 32 kbps offer toll quality, while the other rates don't. G.727 is called embedded ADPCM because the 2-bit quantizer is embedded into the 3-bit quantizer, which is embedded into the 4-bit quantizer, and into the 5-bit quantizer. This makes it possible for the same codec to use a lower bit rate, with a graceful degradation in quality, if channel capacity is temporarily limited. Earlier standards G.721 [7, 13] (created in 1984) and G.723 have been subsumed by G.726 and G.727. G.727 has a MOS of 4.1 for 32 kbps and is used in submarine cables. The Windows WAV format also supports a variant of ADPCM. These standards are shown in Table 7.2.

Table 7.2 Common scalar waveform standards used.

Standard	Bit Rate (kbits/sec)	MOS	Algorithm	Sampling Rate (kHz)
Stereo CD Audio	1411	5.0	16-bit linear PCM	44.1
WAV, AIFF, SND	Variable	-	16/8-bit linear PCM	8, 11.025, 16, 22.05, 44.1, 48
G.711	64	4.3	μ-law/A-law PCM	8
G.727	40, 32, 24, 16	4.2 (32k)	ADPCM	8
G.722	64, 56, 48		Subband ADPCM	16

Wideband speech (50–7000 Hz) increases intelligibility of fricatives and overall perceived quality. In addition, it provides more subject presence and adds a feeling of transparent communication. ITU-T Recommendation G.722 encodes wideband speech with bit rates of 48, 56, and 64 kbps. Speech is divided into two subbands with QMF filters (see Chapter 5). The upper band is encoded using a 16-kbps ADPCM similar to the G.727 standard. The lower band is encoded using a 48-kbps ADPCM with the 4- and 5-bit quantizers embedded in the 6-bit quantizer. The quality of this system scores almost 1 MOS higher than that of telephone speech.

7.3. SCALAR FREQUENCY DOMAIN CODERS

Frequency domain is advantageous because:

1. The samples of a speech signal have a great deal of correlation among them, whereas frequency domain components are approximately uncorrelated and

2. The perceptual effects of masking described in Chapter 2 can be more easily implemented in the frequency domain. These effects are more pronounced for high-bandwidth signals, so frequency-domain coding has been mostly used for CD-quality signals and not for 8-kHz speech signals.

7.3.1. Benefits of Masking

As discussed in Chapter 2, masking is a phenomenon by which human listeners cannot perceive a sound if it is below a certain level. The consequence is that we don't need to encode such sound. We now illustrate how this masked threshold is computed for MPEG[3]-1 layer 1. Given an input signal $s[n]$ quantized with b bits, we obtain the normalized signal $x[n]$ as

$$x[n] = \frac{s[n]}{N2^{b-1}} \tag{7.33}$$

where $N = 512$ is the length of the DFT. Then, using a Hanning window,

$$w[n] = 0.5 - 0.5\cos(2\pi n / N) \tag{7.34}$$

we obtain the log-power spectrum as

$$P[k] = P_0 + 10\log_{10}\left|\sum_{n=0}^{N-1} w[n]x[n]e^{-j2\pi nk/N}\right|^2 \tag{7.35}$$

where P_0 is the playback SPL, which, in the absence of any volume information, is defined as 90 dB.

Tonal components are identified in Eq. (7.35) as local maxima, which exceed neighboring components within a certain Bark distance by at least 7 dB. Specifically, bin k is tonal if and only if

$$P[k] > P[k \pm 1] \tag{7.36}$$

and

$$P[k] > P[k \pm l] + 7dB \tag{7.37}$$

where $1 < l \le \Delta_k$, and Δ_k is given by

$$\Delta_k = \begin{cases} 2 & 2 < k < 63 \quad (170\,\text{Hz} - 5.5\,\text{kHz}) \\ 3 & 63 \le k < 127 \quad (5.5\,\text{kHz}, 11\,\text{kHz}) \\ 6 & 127 \le k \le 256 \quad (11\,\text{kHz}, 22\,\text{kHz}) \end{cases} \tag{7.38}$$

[3] MPEG (Moving Picture Experts Group) is the nickname given to a family of International Standards for coding audiovisual information.

so that the power of that tonal masker is computed as the sum of the power in that bin and its left and right adjacent bins:

$$P_{TM}[k] = 10\log_{10}\left(\sum_{j=-1}^{1} 10^{0.1P[k+j]}\right) \tag{7.39}$$

The noise maskers are computed as the sum of power spectrum of the remaining frequency bins $\bar{k}$ in a critical band not within a neighborhood Δ_k of the tonal maskers:

$$P_{NM}[\bar{k}] = 10\log_{10}\left(\sum_{j} 10^{0.1P[j]}\right) \tag{7.40}$$

where j spans a critical band.

To compute the overall masked threshold we need to sum all masking thresholds contributed by each frequency bin i, which is approximately equal to the maximum (see Chapter 2):

$$T[k] = \max\left(T_h[k], \max_i\left(T_i[k]\right)\right) \tag{7.41}$$

In Chapter 2 we saw that whereas temporal postmasking can last from 50 to 300 ms, temporal premasking tends to last about 5 ms. This is also important because when a frequency transform is quantized, the blocking effects of transform's coders can introduce noise above the temporal premasking level that can be audible, since 1024 points corresponds to 23 ms at a 44-kHz sampling rate. To remove this pre-echo distortion, audible in the presence of castanets and other abrupt transient signals, subband filtering has been proposed, whose time constants are well below the 5-ms premasking time constant.

7.3.2. Transform Coders

We now use the *Adaptive Spectral Entropy Coding* (ASPEC) of High Quality Music Signals algorithm, which is the basis for the MPEG1 Layer 1 audio coding standard [24], to illustrate how transform coders work. The DFT coefficients are grouped into 128 subbands, and 128 scalar quantizers are used to transmit all the DFT coefficients. It has been empirically found that a difference of less than 1 dB between the original amplitude and the quantized value cannot be perceived. Each subband j has a quantizer having k_j levels and step size of T_j as

$$k_j = 1 + 2\times \mathrm{rnd}\left(P_j / T_j\right) \tag{7.42}$$

where T_j is the quantized JND threshold, P_j is the quantized magnitude of the largest real or imaginary component of the j^{th} subband, and rnd( ) is the nearest integer rounding function. Entropy coding (see Chapter 3) is used to encode the coefficients of that subband. Both

T_j and P_j are quantized on a dB scale using 8-bit uniform quantizers with a 170-dB dynamic range, thus with a step size of 0.66 dB. Then they are transmitted as side information.

There are two main methods of obtaining a frequency-domain representation:

1. Through subband filtering via a filterbank (see Chapter 5). When a filterbank is used, the bandwidth of each band is chosen to increase with frequency following a perceptual scale, such as the Bark scale. As shown in Chapter 5, such filterbanks yield perfect reconstruction in the absence of quantization.
2. Through frequency-domain transforms. Instead of using a DFT, higher efficiency can be obtained by the use of an MDCT (see Chapter 5).

The exact implementation of the MPEG1 Layer I standard is much more complicated and beyond the scope of this book, though it follows the main ideas described here; the same is true for the popular MPEG1 Layer III, also known as MP3. Implementation details can be found in [42].

7.3.3. Consumer Audio

Dolby Digital, MPEG, DTS, and the Perceptual Audio Coder (PAC) [28] are all audio coders based on frequency-domain coding. Except for MPEG-1, which supports only stereo signals, the rest support multichannel.

Dolby Digital is multichannel digital audio, using lossy AC-3 [54] coding technology from original PCM with a sample rate of 48 kHz at up to 24 bits. The bit rate varies from 64 to 448 kbps, with 384 being the normal rate for 5.1 channels and 192 the normal rate for stereo (with or without surround encoding). Most Dolby Digital decoders support up to 640 kbps. Dolby Digital is the format used for audio tracks on almost all Digital Video/Versatile Discs (DVD). A DVD-5 with only one surround stereo audio stream (at 192 kbps) can hold over 55 hours of audio. A DVD-18 can hold over 200 hours.

MPEG was established in 1988 as part of the joint ISO (International Standardization Organization) / IEC (International Electrotechnical Commission) Technical Committee on Information Technology. MPEG-1 was approved in 1992 and MPEG-2 in 1994. Layers I to III define several specifications that provide better quality at the expense of added complexity. MPEG-1 audio is limited to 384 kbps. MPEG1 Layer III audio [23], also known as MP3, is very popular on the Internet, and many compact players exist.

MPEG-2 audio, one of the audio formats used in DVD, is multichannel digital audio, using lossy compression from 16-bit linear PCM at 48 kHz. Tests have shown that for nearly all types of speech and music, at a data rate of 192 kbps and over, on a stereo channel, scarcely any difference between original and coded versions was observable (ranking of coded item > 4.5), with the original signal needing 1.4 Mbps on a CD (reduction by a factor of 7). One advantage of the MPEG audio technique is that future findings regarding psychoacoustic effects can be incorporated later, so it can be expected that today's quality level

using 192 kbps will be achievable at lower data rates in the future. A variable bit rate of 32 to 912 kbps is supported for DVDs.

DTS (Digital Theater Systems) Digital Surround is another multi-channel (5.1) digital audio format, using lossy compression derived from 20-bit linear PCM at 48 kHz. The compressed data rate varies from 64 to 1536 kbps, with typical rates of 768 and 1536 kbps.

7.3.4. Digital Audio Broadcasting (DAB)

Digital Audio Broadcasting (DAB) is a means of providing current AM and FM listeners with a new service that offers: sound quality comparable to that of compact discs, increased service availability (especially for reception in moving vehicles), flexible coverage scenarios, and high spectrum efficiency.

Different approaches have been considered for providing listeners with such a service. Currently, the most advanced system is one commonly referred to as Eureka 147 DAB, which has been under development in Europe under the Eureka Project EU147 since 1988. Other approaches include various American in-band systems (IBOC, IBAC, IBRC, FMDigital, and FMeX) still in development, as well as various other systems promising satellite delivery, such as WorldSpace and CD Radio, still in development as well. One satellite-delivery system called MediaStar (formerly Archimedes) proposes to use the Eureka 147 DAB signal structure, such that a single receiver could access both terrestrial and satellite broadcasts.

DAB has been under development since 1981 at the Institut für Rundfunktechnik (IRT) and since 1987 as part of a European research project (Eureka 147). The Eureka 147 DAB specification was standardized by the European Telecommunications Standards Institute (ETSI) in February 1995 as document ETS 300 401, with a draft second edition issued in June 1996. In December 1994, the International Telecommunication Union—Radiocommunication (ITU-R) recommended that this technology, referred to as Digital System A, be used for implementing DAB services.

The Eureka 147 DAB signal consists of multiple carriers within a 1.536-MHz channel bandwidth. Four possible modes of operation define the channel coding configuration, specifying the total number of carriers, the carrier spacing, and also the guard interval duration. Each channel provides a raw data rate of 2304 kbps; after error protection, a useful data rate of anywhere between approximately 600 kbps up to 1800 kbps is available to the service provider, depending on the user-specified multiplex configuration. This useful data rate can be divided into an infinite number of possible configurations of audio and data programs. All audio programs are individually compressed using MUSICAM (MPEG-1 Layer II).

For each useful bit, 1 1/3 ... 4 bits are transmitted. This extensive redundancy makes it possible to reconstruct the transmitted bit sequence in the receiver, even if part of it is disrupted during transmission (FEC—forward error correction). In the receiver, error concealment can be carried out at the audio reproduction stage, so that residual transmission errors which could not be corrected do not always cause disruptive noise.

7.4. CODE EXCITED LINEAR PREDICTION (CELP)

The use of linear predictors removes redundancy in the signal, so that coding of the residual signal can be done with simpler quantizers. We first introduce the LPC vocoder and then introduce coding of the residual signal with a very popular technique called CELP.

7.4.1. LPC Vocoder

A typical model for speech production is shown in Figure 7.6, which has a source, or excitation, driving a linear time-varying filter. For voiced speech, the excitation is an impulse train spaced P samples apart. For unvoiced speech, the source is white random noise. The filter $h_m[n]$ for frame m changes at regular intervals, say every 10 ms. If this filter is represented with linear predictive coding, it is called an *LPC vocoder* [3].

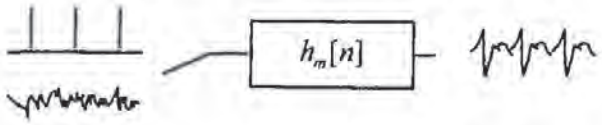


Figure 7.6 Block diagram of an LPC vocoder.

In addition to transmitting the gain and LPC coefficients, the encoder has to determine whether the frame is voiced or unvoiced, as well as the pitch period P for voiced frames.

The LPC vocoder produces reasonable quality for unvoiced frames, but often results in somewhat mechanical sound for voiced sounds, and a buzzy quality for voiced fricatives. More importantly, the LPC vocoder is quite sensitive to voicing and pitch errors, so that an accurate pitch tracker is needed for reasonable quality. The LPC vocoder also performs poorly in the presence of background noise. Nonetheless, it can be highly intelligible. The Federal Standard 1015 [55], proposed for secure communications, is based on a 2.4-kbps LPC vocoder.

It's also possible to use linear predictive coding techniques together with Huffman coding [45] to achieve lossless compression of up to 50%.

7.4.2. Analysis by Synthesis

Code Excited Linear Prediction (CELP) [5] is an umbrella term for a family of techniques that quantize the LPC residual using VQ, thus the term *code excited*, using analysis by synthesis. In addition CELP uses the fact that the residual of voiced speech has periodicity and can be used to predict the residual of the current frame. In CELP coding the LPC coefficients are quantized and transmitted (feedforward prediction), as well as the codeword index. The prediction using LPC coefficients is called *short-term prediction*. The prediction of the residual based on pitch is called *long-term prediction*. To compute the quantized coefficients we use an *analysis-by-synthesis* technique, which consists of choosing the combina-

tion of parameters whose reconstructed signal is closest to the analysis signal. In practice, not all coefficients of a CELP coder are estimated in an analysis-by-synthesis manner.

We first estimate the p^{th}-order LPC coefficients from the samples $x[n]$ for frame t using the autocorrelation method, for example. We then quantize the LPC coefficients to $(a_1, a_2, \cdots, a_p)$ with the techniques described in Section 7.4.5. The residual signal $e[n]$ is obtained by inverse filtering $x[n]$ with the quantized LPC filter

$$e[n] = x[n] - \sum_{i=1}^{p} a_i x[n-i] \tag{7.43}$$

Given the transfer function of the LPC filter

$$H(z) = \frac{1}{A(z)} = \frac{1}{1 - \sum_{i=1}^{p} a_i z^{-i}} = \sum_{i=0}^{\infty} h_i z^{-i} \tag{7.44}$$

we can obtain the first M coefficients of the impulse response $h[n]$ of the LPC filter by driving it with an impulse as

$$h[n] = \begin{cases} 1 & n = 0 \\ \sum_{i=1}^{n} a_i h[n-i] & 0 < n < p \\ \sum_{i=1}^{p} a_i h[n-i] & p \le n < M \end{cases} \tag{7.45}$$

so that if we quantize a frame of M samples of the residual $\mathbf{e} = (e[0], e[1], \cdots e[M-1])^T$ to $\mathbf{e}_i = (e_i[0], e_i[1], \cdots e_i[M-1])^T$, we can compute the reconstructed signal $\hat{x}_i[n]$ as

$$\hat{x}_i[n] = \sum_{m=0}^{n} h[m] e_i[n-m] + \sum_{m=n+1}^{\infty} h[m] e[n-m] \tag{7.46}$$

where the second term in the sum depends on the residual for previous frames, which we already have. Let's define signal $r_0[n]$ as the second term of Eq (7.46):

$$r_0[n] = \sum_{m=n+1}^{\infty} h[m] e[n-m] \tag{7.47}$$

which is the output of the LPC filter when there is no excitation for frame t. The important thing to note is that $r_0[n]$ does not depend on $e_i[n]$.

It is convenient to express Eqs. (7.46) and (7.47) in matrix form as

$$\hat{\mathbf{x}}_i = \mathbf{H}\mathbf{e}_i + \mathbf{r}_0 \tag{7.48}$$

where matrix $\mathbf{H}$ corresponds to the LPC filtering operation with its memory set to 0:

$$\mathbf{H}=\begin{bmatrix} h_0 & 0 & \cdots & 0 & 0 \\ h_1 & h_0 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ h_{M-1} & h_{M-2} & \cdots & h_0 & 0 \\ h_M & h_{M-1} & \cdots & h_1 & h_0 \end{bmatrix} \tag{7.49}$$

Given the large dynamic range of the residual signal, we use gain-shape quantization, where we quantize the gain and the gain-normalized residual separately:

$$\mathbf{e}_i = \lambda \mathbf{c}_i \tag{7.50}$$

where λ is the gain and $\mathbf{c}_i$ is the codebook entry i. This codebook is known as the *fixed codebook* because its vectors do not change from frame to frame. Usually the size of the codebook is selected as 2^N so that full use is made of all N bits. Codebook sizes typically vary from 128 to 1024. Combining Eq. (7.48) with Eq. (7.50), we obtain

$$\hat{\mathbf{x}}_i = \lambda \mathbf{H}\mathbf{c}_i + \mathbf{r}_0 \tag{7.51}$$

The error between the original signal $\mathbf{x}$ and the reconstructed signal $\hat{\mathbf{x}}_i$ is

$$\boldsymbol{\varepsilon}_i = \mathbf{x} - \hat{\mathbf{x}}_i \tag{7.52}$$

The optimal gain λ and codeword index i are the ones that minimize the squared error between the original signal and the reconstructed[4] signal:

$$E(i,\lambda) = \left|\mathbf{x}-\hat{\mathbf{x}}_i\right|^2 = \left|\mathbf{x}-\lambda\mathbf{H}\mathbf{c}_i-\mathbf{r}_0\right|^2 = \left|\mathbf{x}-\mathbf{r}_0\right|^2 + \lambda^2\mathbf{c}_i^T\mathbf{H}^T\mathbf{H}\mathbf{c}_i - 2\lambda\mathbf{c}_i^T\mathbf{H}^T(\mathbf{x}-\mathbf{r}_0) \tag{7.53}$$

where the term $\left|\mathbf{x}-\mathbf{r}_0\right|^2$ does not depend on λ or i and can be neglected in the minimization. For a given $\mathbf{c}_i$, the gain λ_i that minimizes Eq. (7.53) is given by

$$\lambda_i = \frac{\mathbf{c}_i^T\mathbf{H}^T(\mathbf{x}-\mathbf{r}_0)}{\mathbf{c}_i^T\mathbf{H}^T\mathbf{H}\mathbf{c}_i} \tag{7.54}$$

Inserting Eq. (7.54) into (7.53) lets us compute the index j as the one that minimizes

$$j = \arg\min_i \left\{ -\frac{\left(\mathbf{c}_i^T\mathbf{H}^T(\mathbf{x}-\mathbf{r}_0)\right)^2}{\mathbf{c}_i^T\mathbf{H}^T\mathbf{H}\mathbf{c}_i} \right\} \tag{7.55}$$

So we first obtain the codeword index j according to Eq. (7.55) and then the gain λ_j according to Eq. (7.54), which is scalarly quantized to $\hat{\lambda}_j$. Both codeword index j and $\hat{\lambda}_j$ are transmitted. In the algorithm described here, we first chose the quantized LPC coeffi-

[4] A beginner's mistake is to find the codebook index that minimizes the squared error of the residual. This does not minimize the difference between the original signal and the reconstructed signal.

cients $(a_1, a_2, \cdots, a_p)$ independently of the gains and codeword index, and then we chose the codeword index independently of the quantized gain $\hat{\lambda}_j$. This procedure is called *open-loop* estimation, because some parameters are obtained independently of the others. This is shown in Figure 7.7. *Closed-loop* estimation [49] means that all possible combinations of quantized parameters are explored. Closed-loop is more computationally expensive but yields lower squared error.

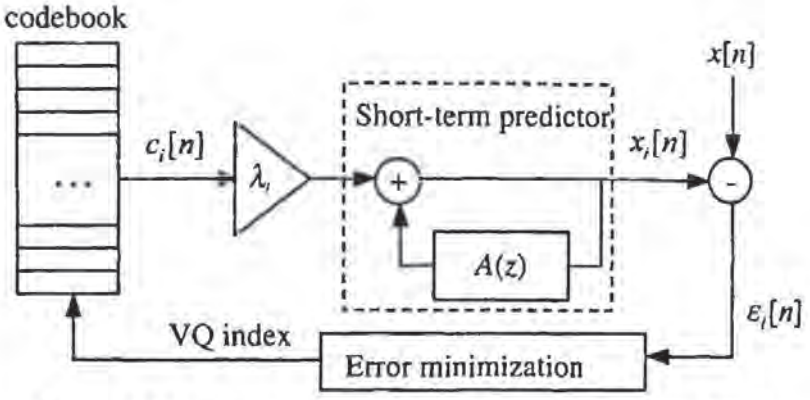


Figure 7.7 Analysis-by-synthesis principle used in a basic CELP.

7.4.3. Pitch Prediction: Adaptive Codebook

The fact that speech is highly periodic during voiced segments can also be used to reduce redundancy in the signal. This can be done by predicting the residual signal $e[n]$ at the current vector with samples from the past residual signal shifted a pitch period t:

$$e[n] = \lambda_i^a e[n-t] + \lambda_i^f c_i^f[n] = \lambda_i^a c_i^a[n] + \lambda_i^f c_i^f[n] \tag{7.56}$$

Using the matrix framework we described before, Eq. (7.56) can be expressed as

$$\mathbf{e}_{ti} = \lambda_i^a \mathbf{c}_t^a + \lambda_i^f \mathbf{c}_i^f \tag{7.57}$$

where we have made use of an *adaptive codebook* [31], where $\mathbf{c}_t^a$ is the adaptive codebook entry j with corresponding gain λ^a, and $\mathbf{c}_i^f$ is the fixed or stochastic codebook entry i with corresponding gain λ^f. The adaptive codebook entries are segments of the recently synthesized excitation signal

$$\mathbf{c}_t^a = (e[-t], e[1-t], \cdots, e[M-1-t])^T \tag{7.58}$$

where t is the delay which specifies the start of the adaptive codebook entry t. The range of t is often between 20 and 147, since this can be encoded with 7 bits. This corresponds to a range in pitch frequency between 54 and 400 Hz for a sampling rate of 8 kHz.

The contribution of the adaptive codebook is much larger than that of the stochastic codebook for voiced sounds. So we generally search for the adaptive codebook first, using Eq. (7.58) and a modified version of Eqs. (7.55) and (7.54), replacing i by t. Closed-loop search of both t and gain here often yields a much larger error reduction.

7.4.4. Perceptual Weighting and Postfiltering

The objective of speech coding is to reduce the bit rate while maintaining a perceived level of quality; thus, minimization of the error is not necessarily the best criterion. A perceptual weighting filter tries to shape the noise so that it gets masked by the speech signal (see Chapter 2). This generally means that most of the quantization noise energy is located in spectral regions where the speech signal has most of its energy. A common technique [4] consists in approximating this perceptual weighting with a linear filter

$$W(z) = \frac{A(z/\beta)}{A(z/\gamma)} \tag{7.59}$$

where $A(z)$ is the predictor polynomial

$$A(z) = 1 - \sum_{i=1}^{p} a_i z^{-i} \tag{7.60}$$

Choosing γ and β so that and $0 < \gamma < \beta \leq 1$, implies that the roots of $A(z/\beta)$ and $A(z/\gamma)$ will move closer to the origin of the unit circle than the roots of $A(z)$, thus resulting in a frequency response with wider resonances. This perceptual filter therefore deemphasizes the contribution of the quantization error near the formants. A common choice of parameters is $\beta = 1.0$ and $\gamma = 0.8$, since it simplifies the implementation. This filter can easily be included in the matrix **H**, and a CELP coder incorporating the perceptual weighting is shown in Figure 7.8.

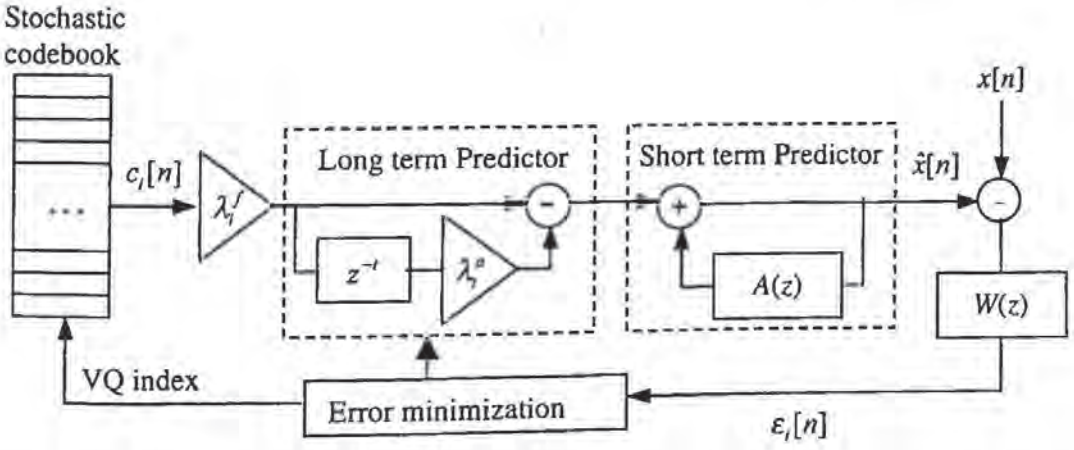


Figure 7.8 Diagram of a CELP coder. Both long-term and short-term predictors are used, together with a perceptual weighting.

Despite the perceptual weighting filter, the reconstructed signal still contains audible noise. This filter reduces the noise in those frequency regions that are perceptually irrelevant without degrading the speech signal. The postfilter generally consists of a short-term postfilter to emphasize the formant structure and a long-term postfilter to enhance the periodicity of the signal [10]. One possible implementation follows Eq. (7.59) with values of $\beta = 0.5$ and $\gamma = 0.75$.

7.4.5. Parameter Quantization

To achieve a low bit rate, all the coefficients need to be quantized. Because of its coding efficiency, vector quantization is the compression technique of choice to quantize the predictor coefficients. The LPC coefficients cannot be quantized directly, because small errors produced in the quantization process may result in large changes in the spectrum and possibly unstable filters. Thus, equivalent representations that guarantee stability are used, such as reflection coefficients, log-area ratios, and the line spectral frequencies (LSF) described in Chapter 6. LSF are used most often, because it has been found empirically that they behave well when they are quantized and interpolated [2]. For 8 kHz, 10 predictor coefficients are often used, which makes using a single codebook impractical because of the large dimension of the vector. Split-VQ [43] is a common choice, where the vectors are divided into several subvectors, and each is vector quantized. Matrix quantization can also be used to exploit the correlation of these subvectors across consecutive time frames. *Transparent quality*, defined as average spectral distortion below 1 dB with no frames above 4 dB, can be achieved with fewer than 25 bits per frame.

A frame typically contains around 20 to 30 milliseconds, which at 8 kHz represents 160–240 samples. Because of the large vector dimension, it is impractical to quantize a whole frame with a single codebook. To reduce the dimensionality, the frame is divided into four or more nonoverlapping sub-frames. The LSF coefficients for each subframe are linearly interpolated between the two neighboring frames.

The typical range of the pitch prediction for an 8-kHz sampling rate goes from 54 to 400 Hz, from 20 to 147 samples, and from 2.5 ms to 18.375 ms, which can be encoded with 7 bits. An additional bit is often used to encode fractional delays for the lower pitch periods. These fractional delays can be implemented through upsampling as described in Chapter 5. The subframe gain of the adaptive codebook can be effectively encoded with 3 or 4 bits. Alternatively, the gains of all sub-frames within a frame can be encoded through VQ, resulting in more efficient compression.

The fixed codebook can be trained from data using the techniques described in Chapter 4. This will offer the lowest distortion for the training set but doesn't guarantee low distortion for mismatched test signals. Also, it requires additional storage, and full search increases computation substantially.

Since subframes should be approximately white, the codebook can be populated from samples of a white process. A way of reducing computation is to let those noise samples be only +1, 0, or –1, because only additions are required. Codebooks of a specific type, known as *algebraic codebooks* [1], offer even more computational savings because they contain

many 0s. Locations for the 4 pulses per subframe under the G.729 standard are shown in Table 7.3.

Full search can efficiently be done with this codebook structure. Algebraic codebooks can provide almost as low distortion as trained codebooks can, with low computational complexity.

Table 7.3 Algebraic codebooks for the G.729 standard. Each of the four codebooks has one pulse in one possible location indicated by 3 bits for the first three codebooks and 4 bits for the last codebook. The sign is indicated by an additional bit. A total of 17 bits are needed to encode a 40-sample subframe.

Amplitude	Positions
±1	0, 5, 10, 15, 20, 25, 30, 35
±1	1, 6, 11, 16, 21, 26, 31, 36
±1	2, 7, 12. 17, 22, 27, 32, 37
±1	3, 8, 13, 18, 23, 28, 33, 38 4, 9, 14, 19, 24, 29, 34, 39

7.4.6. CELP Standards

There are many standards for speech coding based on CELP, offering various points in the bit-rate/quality plane, mostly depending on when they were created and how refined the technology was at that time.

Voice over Internet Protocol (Voice over IP) consists of transmission of voice through data networks such as the Internet. H.323 is an umbrella standard which references many other ITU-T recommendations. H.323 provides the system and component descriptions, call model descriptions, and call signaling procedures. For audio coding, G.711 is mandatory, while G.722, G.728, G.723.1, and G.729 are optional. G.728 is a low-delay CELP coder that offers toll quality at 16 kbps [9], using a feedback 50th-order predictor, but no pitch prediction. G.729 [46] offers toll quality at 8 kbps, with a delay of 10 ms. G.723.1, developed by DSP Group, including Audiocodes Ltd., France Telecom, and the University of Sherbrooke, has slightly lower quality at 5.3 and 6.3 kbps, but with a delay of 30 ms. These standards are shown in Table 7.4.

Table 7.4 Several CELP standards used in the H.323 specification used for teleconferencing and voice streaming through the Internet.

Standard	Bit Rate (kbps)	MOS	Algorithm	H.323	Comments
G.728	16	4.0	No pitch prediction	Optional	Low-delay
G.729	8	3.9	ACELP	Optional	
G.723.1	5.3, 6.3	3.9	ACELP for 5.3k	Optional	

In 1982, the Conference of European Posts and Telegraphs (CEPT) formed a study group called the Groupe Spécial Mobile (GSM) to study and develop a pan-European public land mobile system. In 1989, GSM responsibility was transferred to the European Telecommunication Standards Institute (ETSI), and the phase I GSM specifications were published in 1990. Commercial service was started in mid 1991, and by 1993 there were 36 GSM networks in 22 countries, with 25 additional countries considering or having already selected GSM. This is not only a European standard; South Africa, Australia, and many Middle and Far East countries have chosen GSM. The acronym GSM now stands for Global System for Mobile telecommunications. The GSM group studied several voice coding algorithms on the basis of subjective speech quality and complexity (which is related to cost, processing delay, and power consumption once implemented) before arriving at the choice of a Regular Pulse Excited–Linear Predictive Coder (RPE-LPC) with a Long Term Predictor loop [56]. Neither the original full-rate at 13 kbps [56] nor the half-rate at 5.6 kbps [19] achieves toll quality, though the enhanced full-rate (EFR) standard based on ACELP [26] has toll quality at the same rates.

The *Telecommunication Industry Association* (TIA) and the *Electronic Industries Alliance* (EIA) are organizations accredited by the *American National Standards Institute* (ANSI) to develop voluntary industry standards for a wide variety of telecommunication products. TR-45 is the working group within TIA devoted to mobile and personal communication systems. Time Division Multiple Access (TDMA) is a digital wireless technology that divides a narrow radio channel into framed time slots (typically 3 or 8) and allocates a slot to each user. The TDMA Interim Standard 54, or TIA/EIA/IS54, was released in early 1991 by both TIA and EIA. It is available in North America at both the 800-MHz and 1900-MHz bands. IS54 [18] at 7.95 kbps is used in North America's TDMA (Time Division Multiple Access) digital telephony and has quality similar to the original full-rate GSM. TDMA IS-136 is an update released in 1994.

Table 7.5 CELP standards used in cellular telephony.

Standard	Bit Rate (kbps)	MOS	Algorithm	Cellular	Comments
Full-rate GSM	13	3.6	VSELP RTE-LTP	GSM	
EFR GSM	12.2	4.5	ACELP	GSM	
IS-641	7.4	4.1	ACELP	PCS1900	
IS-54	7.95	3.9	VSELP	TDMA	
IS-96a	max 8.5	3.9	QCELP	CDMA	Variable-rate

Code Division Multiple Access (CDMA) is a form of *spread spectrum*, a family of digital communication techniques that have been used in military applications for many years. The core principle is the use of noiselike carrier waves, and, as the name implies,

bandwidths much wider than that required for simple point-to-point communication at the same data rate. Originally there were two motivations: either to resist enemy efforts to jam the communications (anti-jam, or AJ) or to hide the fact that communication was even taking place, sometimes called low probability of intercept (LPI). The service started in 1996 in the United States, and by the end of 1999 there were 50 million subscribers worldwide. IS-96 QCELP [14], used in North America's CDMA, offers variable-rate coding at 8.5, 4, 2, and 0.8 kbps. The lower bit rate is transmitted when the coder detects background noise. TIA/EIA/IS-127-2 is a standard for an enhanced variable-rate codec, whereas TIA/EIA/IS-733-1 is a standard for high-rate. Standards for CDMA, TDMA, and GSM are shown in Table 7.5.

Third generation (3G) is the generic term used for the next generation of mobile communications systems. 3G systems will provide enhanced services to those—such as voice, text, and data—predominantly available today. The Universal Mobile Telecommunications System (UMTS) is a part of ITU's International Mobile Telecommunications (IMT)-2000 vision of a global family of third-generation mobile communications systems. It has been assigned to the frequency bands 1885–2025 and 2110–2200 MHz. The first networks are planned to launch in Japan in 2001, with European countries following in early 2002. A major part of 3G is General Packet Radio Service (GPRS), under which carriers charge by the packet rather than by the minute. The speech coding standard for CDMA2000, the umbrella name for the third-generation standard in the United States, gained approval for its first phase in 2000. An adaptive multi-rate wideband speech codec has also been proposed for the GSM's 3G [16], which has five modes of operation from 24 kbps down to 9.1 kbps.

While most of the work described above uses a sampling rate of 8 kHz, there has been growing interest in using CELP techniques for high bandwidth and particularly in a scalable way so that a basic layer contains the lower frequency and the higher layer either is a full-band codec [33] or uses a parametric model [37].

7.5. LOW-BIT RATE SPEECH CODERS

In this section we describe a number of low-bit-rate speech coding techniques including the mixed-excitation LPC vocoder, harmonic coding, and waveform interpolation. These coding techniques are also used extensively in speech synthesis.

Waveform-approximating coders are designed to minimize the difference between the original signal and the coded signal. Therefore, they produce a reconstructed signal whose SNR goes to infinity as the bit rate increases, and they also behave well when the input signal is noisy or music. In this category we have the scalar waveform coders of Section 7.2, the frequency-domain coders of Section 7.3, and the CELP coders of Section 7.4.

Low-bit-rate coders, on the other hand, do not attempt to minimize the difference between the original signal and the quantized signal. Since these coders are designed to operate at low bit rates, their SNR does not generally approach infinity even if a large bit rate is used. The objective is to compress the original signal with another one that is perceptually equivalent. Because of the reliance on an inaccurate model, these low-bit-rate coders often

distort the speech signal even if the parameters are not quantized. In this case, the distortion can consist of more than quantization noise. Furthermore, these coders are more sensitive to the presence of noise in the signal, and they do not perform as well on music.

In Figure 7.9 we compare the MOS of waveform approximating coders and low-bit-rate coders as a function of the bit rate. CELP uses a model of speech to obtain as much prediction as possible, yet allows for the model not to be exact, and thus is a waveform-approximating coder. CELP is a robust coder that works reasonably well when the assumption of only a clean speech signal breaks either because of additive noise or because there is music in the background. Researchers are working on the challenging problem of creating more scalable coders that offer best performance at all bit rates.

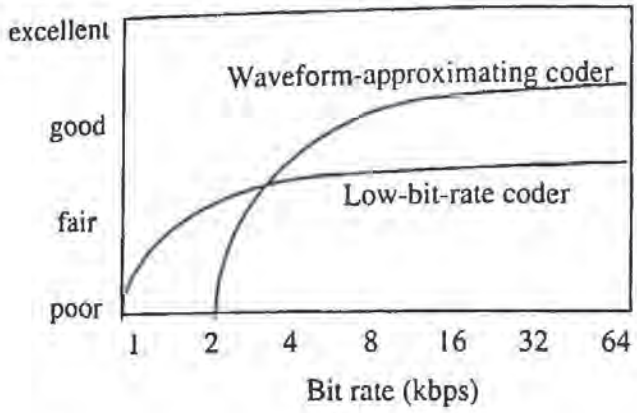


Figure 7.9 Typical subjective performance of waveform-approximating and low-bit-rate coders as a function of the bit rate. Note that waveform-approximating coders are a better choice for bit rates higher than about 3 kbps, whereas parametric coders are a better choice for lower bit rates. The exact cutoff point depends on the specific algorithms compared.

7.5.1. Mixed-Excitation LPC Vocoder

The main weakness of the LPC vocoder is the binary decision between voiced and unvoiced speech, which results in errors especially for noisy speech and voiced fricatives. By having a separate voicing decision for each of a number of frequency bands, the performance can be enhanced significantly [38]. The new proposed U.S. Federal Standard at 2.4 kbps is a Mixed Excitation Linear Prediction (MELP) LPC vocoder [39], which has a MOS of about 3.3. This exceeds the quality of the older 4800-bps Federal Standard 1016 [8] based on CELP. The bit rate of the proposed standard can be reduced while maintaining the same quality by jointly quantizing several frames together [57]. A hybrid codec that uses MELP in strongly voiced regions and CELP in weakly voiced and unvoiced regions [53] has shown to yield lower bit rates. MELP can also be combined with the waveform interpolation technique of Section 7.5.3 [50].

7.5.2. Harmonic Coding

Sinusoidal coding decomposes the speech signal [35] or the LP residual signal [48] into a sum of sinusoids. The case where these sinusoids are harmonically related is of special interest for speech synthesis (see Chapter 16), so we will concentrate on it in this section, even though a similar treatment can be followed for the case where the sinusoids are not harmonically related. In fact, a combination of harmonically related and nonharmonically related sinusoids can also be used [17]. We show in Section 7.5.2.2 that we don't need to transmit the phase of the sinusoids, only the magnitude.

As shown in Chapter 5, a periodic signal $\tilde{s}[n]$ with period T_0 can be expressed as a sum of T_0 harmonic sinusoids

$$\tilde{s}[n] = \sum_{l=0}^{T_0-1} A_l \cos(nl\omega_0 + \phi_l) \tag{7.61}$$

whose frequencies are multiples of the fundamental frequency $\omega_0 = 2\pi / T_0$, and where A_l and ϕ_l are the sinusoid amplitudes and phases, respectively. If the pitch period T_0 has fractional samples, the sum in Eq. (7.61) includes only the integer part of T_0 in the summation. Since a real signal $s[n]$ will not be perfectly periodic in general, we have a modeling error

$$e[n] = s[n] - \tilde{s}[n] \tag{7.62}$$

We can use short-term analysis to estimate these parameters from the input signal $s[n]$ at frame k, in the neighborhood of $t = kN$, where N is the frame shift:

$$s_k[n] = s[n]w_k[n] = s[n]w[kN - n] \tag{7.63}$$

if we make the assumption that the sinusoid parameters for frame k (ω_0^k, A_l^k and ϕ_l^k) are constant within the frame.

At resynthesis time, there will be discontinuities at unit boundaries, due to the block processing, unless we specifically smooth the parameters over time. One way of doing this is with overlap-add method between frames $(k-1)$ and k:

$$\hat{s}[n] = w[n]\tilde{s}_{k-1}[n] + w[n-N]\tilde{s}_k[n-N], \quad 0 \le n < N \tag{7.64}$$

where the window $w[n]$ must be such that

$$w[n] + w[n-N] = 1, \quad 0 \le n < N \tag{7.65}$$

to achieve perfect reconstruction. This is the case for the common Hamming and Hanning windows.

This harmonic model [35] is similar to the classic filterbank, though rather than the whole spectrum we transmit only the fundamental frequency ω_0 and the amplitudes A_l and phases ϕ_l of the harmonics. This reduced representation doesn't result in loss of quality for a frame shift N that corresponds to 12 ms or less. For unvoiced speech, using a default pitch of 100 Hz results in acceptable quality.

7.5.2.1. Parameter Estimation

For simplicity in the calculations, let's define $\tilde{s}[n]$ as a sum of complex exponentials

$$\tilde{s}[n] = \sum_{l=0}^{T_0-1} A_l \exp\{j(nl\omega_0 + \phi_l)\} \tag{7.66}$$

and perform short-time Fourier transform with a window $w[n]$

$$\tilde{S}_W(\omega) = \sum_{l=0}^{T_0-1} A_l e^{j\phi_l} W(\omega - l\omega_0) \tag{7.67}$$

where $W(\omega)$ is the Fourier transform of the window function. The goal is to estimate the sinusoid parameters as those that minimize the squared error:

$$E = |S(\omega) - \tilde{S}_W(\omega)|^2 \tag{7.68}$$

If the main lobes of the analysis window do not overlap, we can estimate the phases ϕ_l as

$$\phi_l = \arg S(l\omega_0) \tag{7.69}$$

and the amplitudes A_l as

$$A_l = \frac{|S(l\omega_0)|}{W(0)} \tag{7.70}$$

For example, the Fourier transform of a $(2N + 1)$ point rectangular window centered around the origin is given by

$$W(\omega) = \frac{\sin((2N+1)\omega/2)}{\sin(\omega/2)} \tag{7.71}$$

whose main lobes will not overlap in Eq. (7.67) if $2T_0 < 2N+1$: i.e., the window contains at least two pitch periods. The implicit assumption in the estimates of Eqs. (7.69) and (7.70) is that there is no spectral leakage, but a rectangular window does have significant spectral leakage, so a different window is often used in practice. For windows such as Hanning or Hamming, which reduce the leakage significantly, it has been found experimentally that these estimates are acceptable if the window contains at least two and a half pitch periods.

Typically, the window is centered around 0 (nonzero in the interval $-N \le n \le N$) to avoid numerical errors in estimating the phases.

Another implicit assumption in Eqs. (7.69) and (7.70) is that we know the fundamental frequency ω_0 ahead of time. Since, in practice, this is not the case, we can estimate it as the one which minimizes Eq. (7.68). This pitch-estimation method can generate pitch doubling or tripling when a harmonic falls within a formant that accounts for the majority of the signal's energy.

Voiced/unvoiced decisions can be computed from the ratio between the energy of the signal and that of the reconstruction error

$$SNR = \frac{\sum_{n=-N}^{N} |s[n]|^2}{\sum_{n=-N}^{N} |s[n] - \tilde{s}[n]|^2} \tag{7.72}$$

where it has been empirically found that frames with SNR higher than 13 dB are generally voiced and lower than 4 dB unvoiced. In between, the signal is considered to contain a mixed excitation. Since speech is not perfectly stationary within the analysis frame, even noise-free periodic signals will yield finite SNR.

For unvoiced speech, a good assumption is to default to a pitch of 100 Hz. The use of fewer sinusoids leads to perceptual artifacts.

Improved quality can be achieved by using an analysis-by-synthesis framework [17, 34] since the closed-loop estimation is more robust to pitch-estimation and voicing decision errors.

7.5.2.2. Phase Modeling

An impulse train $e[n]$, a periodic excitation, can be expressed as a sum of complex exponentials

$$e[n] = T_0 \sum_{k=-\infty}^{\infty} \delta[n - n_0 - kT_0] = \sum_{l=0}^{T_0-1} e^{j(n-n_0)\omega_0 l} \tag{7.73}$$

which, if passed through a filter $H(\omega) = A(\omega)\exp\Phi(\omega)$, will generate

$$s[n] = \sum_{l=0}^{T_0-1} A(l\omega_0)\exp\{j[(n-n_0)\omega_0 l + \Phi(l\omega_0)]\} \tag{7.74}$$

Comparing Eq. (7.66) with (7.74), the phases of our sinusoidal model are given by

$$\phi_l = -n_0\omega_0 l + \Phi(l\omega_0) \tag{7.75}$$

Since the sinusoidal model has too many parameters to lead to low-rate coding, a common technique is to not encode the phases. In Chapter 6 we show that if a system is considered minimum phase, the phases can be uniquely recovered from knowledge of the magnitude spectrum.

The magnitude spectrum is known at the pitch harmonics, and the remaining values can be filled in by interpolation: e.g., linear or cubic splines [36]. This interpolated magnitude spectrum can be approximated through the real cepstrum:

$$|\tilde{A}(\omega)| = c_0 + 2\sum_{k=1}^{K} c_k \cos(k\omega) \tag{7.76}$$

and the phase, assuming a minimum phase system, is given by

$$\tilde{\Phi}(\omega) = -2\sum_{k=1}^{K} c_k \sin(k\omega) \tag{7.77}$$

The phase $\phi_0(t)$ of the first harmonic between frames $(k-1)$ and k can be obtained from the instantaneous frequency $\omega_0(t)$

$$\phi_0(t) = \phi_0((k-1)N) + \int_{(k-1)N}^{t} \omega_0(t)dt \tag{7.78}$$

if we assume the frequency $\omega_0(t)$ in that region to vary linearly between frames $(k-1)$ and k:

$$\omega_0(t) = \omega_0^{k-1} + \frac{\omega_0^k - \omega_0^{k-1}}{N} t \tag{7.79}$$

and insert Eq. (7.79) into (7.78), evaluating at $t = kN$, to obtain

$$\phi_0^k = \phi_0(kN) = \phi_0((k-1)N) + (\omega_0^{k-1} + \omega_0^k)(N/2) \tag{7.80}$$

the phase of the sinusoid at ω_0 as a function of the fundamental frequencies at frames $(k-1)$, k and the phase at frame $(k-1)$:

$$\phi_l^k = \Phi^k(l\omega_0) + l\phi_0^k \tag{7.81}$$

The phases computed by Eqs. (7.80) and (7.81) are a good approximation in practice for perfectly voiced sounds. For unvoiced sounds, random phases are needed, or else the reconstructed speech sounds buzzy. Voiced fricatives and many voiced sounds have an aspiration component, so that a mixed excitation is needed to represent them. In these cases, the source is split into different frequency bands and each band is classified as either voiced or unvoiced. Sinusoids in voiced bands use the phases described above, whereas sinusoids in unvoiced bands have random phases.

7.5.2.3. Parameter Quantization

To quantize the sinusoid amplitudes, we can use an LPC fitting and then quantize the line spectral frequencies. Also we can do a cepstral fit and quantize the cepstral coefficients. To be more effective, a mel scale should be used.

While these approaches help in reducing the number of parameters and in quantizing those parameters, they are not the most effective way of quantizing the sinusoid amplitudes. A technique called *Variable-Dimension Vector Quantization* (VDVQ) [12] has been devised to address this. Each codebook vector $\mathbf{c}_i$ has a fixed dimension N determined by the length of the FFT used. The vector of sinusoid amplitudes $\mathbf{A}$ has a dimension l that depends on the number of harmonics and thus the pitch of the current frame. To compute the distance between $\mathbf{A}$ and $\mathbf{c}_i$, the codebook vectors are resampled to a size l and the distance is computed between two vectors of dimension l. Euclidean distance of the log-amplitudes is often used.

In this method, only the distance at the harmonics is evaluated instead of the distance at the points in the envelope that are not actually present in the signal. Also, this technique does not suffer from inaccuracies of the model used, such as the inability of linear predictive coding to model nasals.

7.5.3. Waveform Interpolation

The main idea behind waveform interpolation (WI) [29] is that the pitch pulse changes slowly over time for voiced speech. During voiced segments, the speech signal is nearly periodic. WI coders can operate as low as 2.4 kbps.

Starting at an arbitrary time instant, it is easy to identify a first pitch cycle $x_1[n]$, a second $x_2[n]$, a third $x_3[n]$, and so on. We then express our signal $x[n]$ as a function of these pitch cycle waveforms $x_m[n]$

$$x[n] = \sum_{m=-\infty}^{\infty} x_m[n - t_m] \tag{7.82}$$

where $P_m = t_m - t_{m-1}$ is the pitch period at time t_m in samples, and the pitch cycle is a windowed version of the input

$$x_m[n] = w_m[n]x[n] \tag{7.83}$$

for example, with a rectangular window. To transmit the signal in a lossless fashion we need to transmit all pitch waveforms $x_m[n]$.

If the signal is perfectly periodic, we need to transmit only one pitch waveform $x_m[n]$ and the pitch period P. In practice, voiced signals are not perfectly periodic, so that we need to transmit more than just one pitch waveform. On the other hand, voiced speech is nearly periodic, and consecutive pitch waveforms are very similar. Thus, we probably do not need to transmit all, and we could send every other pitch waveform, for example.

It is convenient to define a two-dimensional surface $u[n,l]$ (shown in Figure 7.10) such that the pitch waveform $x_m[n]$ can be obtained as

$$x_m[n] = u[n, t_m] \tag{7.84}$$

so that $u[n,l]$ is defined for $l = t_m$, with the remaining points having been computed through interpolation. A frequency representation of the pitch cycle can also be used instead of the time pitch cycle.

This surface can then be sampled at regular time intervals $l = sT$. It has been shown empirically that transmitting the pitch waveform $x_1[n]$ about 40 times per second (a 25-ms interval is equivalent to $T = 200$ samples for an $F_s = 8000$ Hz sampling rate) is sufficient for voiced speech. The so-called *slowly evolving waveform* (SEW) $\tilde{u}[n,l]$ can be generated by low-pass filtering $u[n,l]$ along the l-axis:

$$x_1[n] = \tilde{u}[n, sT] = \frac{\sum_m h[sT - t_m]u[n, t_m]}{\sum_m h[sT - t_m]} \tag{7.85}$$

where $h[n]$ is a low-pass filter and $x_s[n]$ is a sampled version of $\tilde{u}[n,l]$.

The decoder has to reconstruct each pitch waveform $x_m[n]$ from the SEW $x_s[n]$ by interpolation between adjacent pitch waveforms, and thus the name *waveform interpolation (WI) coding*:

$$\tilde{x}_m[n] = \tilde{u}[n,t_m] = \frac{\sum_s h[t_m - sT]x_s[n]}{\sum_s h[t_m - sT]} \tag{7.86}$$

If the sampling period is larger than the local pitch period ($T > P_m$), perfect reconstruction will not be possible, and there will be some error in the approximation

$$x_m[n] = \tilde{x}_m[n] + \hat{x}_m[n] \tag{7.87}$$

or alternatively in the two-dimensional representation

$$u[n,l] = \tilde{u}[n,l] + \hat{u}[n,l] \tag{7.88}$$

where $\hat{x}_m[n]$ and $\hat{u}[n,l]$ represent the *rapidly evolving waveforms* (REW).

Since this technique can also be applied to unvoiced speech, where the concept of pitch waveform doesn't make sense, the more general term *characteristic waveform* is used instead. For unvoiced speech, an arbitrary *period* of around 100 Hz can be used.

For voiced speech, we expect the rapidly varying waveform $\hat{u}[n,l]$ in Eq. (7.88) to have much less energy than the slowly evolving waveform $\tilde{u}[n,l]$. For unvoiced speech the converse is true: $\hat{u}[n,l]$ has more energy than $\tilde{u}[n,l]$. For voiced fricatives, both components may be comparable and thus we want to transmit both.

In Eqs. (7.85) and (7.86) we need to average characteristic waveforms that have, in general, different lengths. To handle this, all characteristic waveforms are typically normalized in length prior to the averaging operation. This length normalization is done by padding with zeros $x_m[n]$ to a certain length M, or truncating $x_m[n]$ if $P_m > M$. Another possible normalization is done via linear resampling. This decomposition is shown in Figure 7.10.

Another representation uses the Fourier transform of $x_m[n]$. This case is related to the harmonic model of Section 7.5.2. In the harmonic model, a relatively long window is needed to average the several pitch waveforms within the window, whereas this waveform interpolation method has higher time resolution. In constructing the characteristic waveforms we have implicitly used a rectangular window of length one pitch period, but other windows can be used, such as a Hanning window that covers two pitch periods. This frequency-domain representation offers advantages in coding both the SEW and the REW, because properties of the human auditory system can help reduce the bit rate. This decomposition is often done on the LPC residual signal.

In particular, the REW $\hat{u}[n,l]$ has the characteristics for noise, and as such only a rough description of its power spectral density is needed. At the decoder, random noise is generated with the transmitted power spectrum. The spectrum of $\hat{u}[n,l]$ can be vector quantized to as few as eight shapes with little or no degradation.

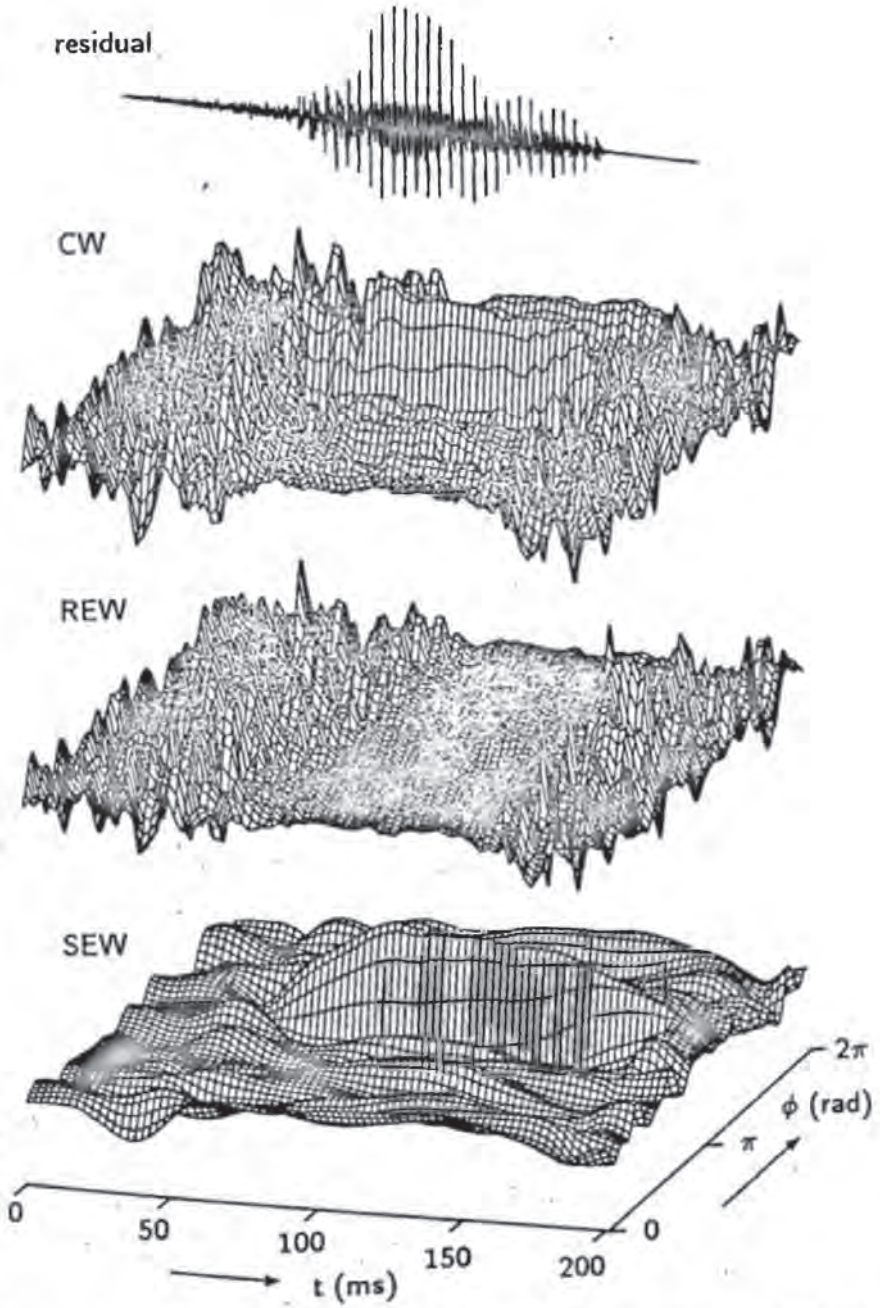


Figure 7.10 LP residual signal and its associated characteristic waveform (CW) $u(t,\phi)$. In the ϕ axis we have a normalized pitch pulse at every given time t. Decomposition of the surface into a slowly evolving waveform (SEW) and a rapidly evolving waveform (REW). (After Kleijn and Haagen [30], reprinted by permission of IEEE).

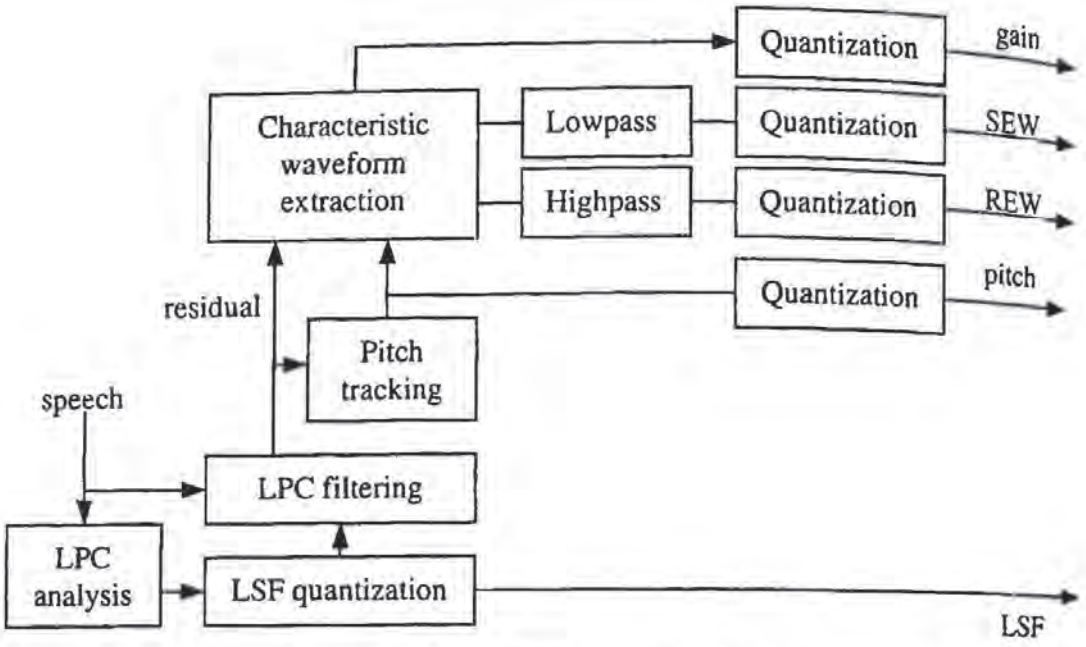


Figure 7.11 Block diagram of the WI encoder.

The SEW $\tilde{u}[n,l]$ is more important perceptually, and for high quality the whole shape needs to be transmitted. Higher accuracy is desired at lower frequencies so that a perceptual frequency scale (mel or Bark) is often used. Since the magnitude of $\tilde{u}[n,l]$ is perceptually more important than the phase, for low bit rates the phase of the SEW is not transmitted. The magnitude spectrum can be quantized with the VDVQ described in Section 7.5.2.3.

To obtain the characteristic waveforms, the pitch needs to be computed. We can find the pitch period such that the energy of the REW is minimized. To do this we use the approaches described in Chapter 6. Figure 7.11 shows a block diagram of the encoder and Figure 7.12 of the decoder.

Parameter estimation using an analysis-by-synthesis framework [21] can yield better results than the open-loop estimation described above.

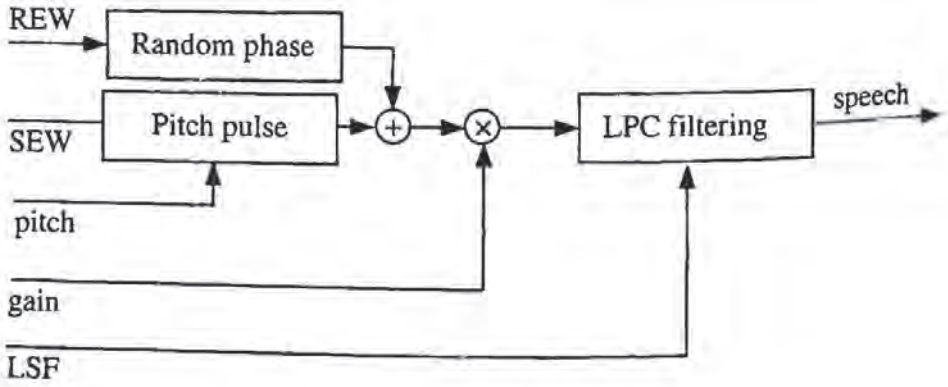


Figure 7.12 Block diagram of the WI decoder.

7.6. HISTORICAL PERSPECTIVE AND FURTHER READING

This chapter is only an introduction to speech and audio coding technologies. The reader is referred to [27, 32, 41, 52] for coverage in greater depth. A good source of the history of speech coding can be found in [20].

In 1939, Homer Dudley of AT&T Bell Labs first proposed the channel vocoder [15], the first analysis-by-synthesis system. This vocoder analyzed slowly varying parameters for both the excitation and the spectral envelope. Dudley thought of the advantages of bandwidth compression and information encryption long before the advent of digital communications.

PCM was first conceived in 1937 by Alex Reeves at the Paris Laboratories of AT&T, and it started to be deployed in the United States Public Switched Telephone Network in 1962. The digital compact disc, invented in the late 1960s by James T. Russell and introduced commercially in 1984, also uses PCM as coding standard. The use of μ-law encoding was proposed by Smith [51] in 1957, but it wasn't standardized for telephone networks (G.711) until 1972. In 1952, Schouten et al. [47] proposed delta modulation and Cutler [11] invented differential PCM. ADPCM was developed by Barnwell [6] in 1974.

Speech coding underwent a fundamental change with the development of linear predictive coding in the early 1970s. Atal [3] proposed the LPC vocoder in 1971, and then CELP [5] in 1984. The majority of coding standards for speech signals today use a variation on CELP.

Sinusoidal coding [35] and waveform interpolation [29] were developed in 1986 and 1991, respectively, for low-bit-rate telephone speech. Transform coders such as MP3 [23], MPEG II, and Perceptual Audio Coder (PAC) [28] have been used primarily in audio coding for high-fidelity applications.

Recently, researchers have been improving the technology for cellular communications by trading off source coding and channel coding. For poor channels more bits are allocated to channel coding and fewer to source coding to reduce dropped calls. Scalable coders that have different layers with increased level of precision, or bandwidth, are also of great interest.

REFERENCES

[1] Adoul, J.P., *et al.*, "Fast CELP Coding Based on Algebraic Codes," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1987, Dallas, TX, pp. 1957-1960.

[2] Atal, B.S., R.V. Cox, and P. Kroon, "Spectral Quantization and Interpolation for CELP Coders," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1989, Glasgow, pp. 69-72.

[3] Atal, B.S. and L. Hanauer, "Speech Analysis and Synthesis by Linear Prediction of the Speech Wave," *Journal of the Acoustical Society of America*, 1971, **50**, pp. 637-655.

[4] Atal, B.S. and M.R. Schroeder, "Predictive Coding of Speech Signals and Subjective Error Criteria," *IEEE Trans. on Acoustics, Speech and Signal Processing*, 1979, **ASSP-27**(3), pp. 247-254.

[5] Atal, B.S. and M.R. Schroeder, "Stochastic Coding of Speech at Very Low Bit Rates," *Proc. Int. Conf. on Comm.*, 1984, Amsterdam, pp. 1610-1613.

[6] Barnwell, T.P., *et al.*, *Adaptive Differential PCM Speech Transmission*, 1974, Rome Air Development Center.

[7] Benvenuto, N., G. Bertocci, and W.R. Daumer, "The 32-kbps ADPCM Coding Standard," *AT&T Technical Journal*, 1986, **65**, pp. 12-22.

[8] Campbell, J.P., T.E. Tremain, and V.C. Welch, "The DoD 4.8 kbps Standard (Proposed Federal Standard 1016)" in *Advances in Speech Coding*, B. Atal, V. Cuperman, and A. Gersho, eds. 1991, Kluwer Academic Publishers, pp. 121-133.

[9] Chen, J.H., *et al.*, "A Low-Delay CELP Coder for the CCITT 16 kbps Speech Coding Standard," *IEEE Journal on Selected Areas Communcations*, 1992, **10**(5), pp. 830-849.

[10] Chen, J.H. and A. Gersho, "Adaptive Postfiltering for Quality Enhancement of Coded Speech," *IEEE Trans. on Speech and Audio Processing*, 1995, **3**(1), pp. 59-71.

[11] Cutler, C.C., *Differential Quantization for Communication Signals*, 1952, US Patent 2,605,361.

[12] Das, A. and A. Gersho, "Variable Dimension Vector Quantization," *IEEE Signal Processing Letters*, 1996, **3**(7), pp. 200-202.

[13] Daumer, W.R., *et al.*, "Overview of the 32kbps ADPCM Algorithm," *Proc. IEEE Global Telecomm*, 1984, pp. 774-777.

[14] DeJaco, P.J.A., W. Gardner, and C. Lee, "QCELP: The North American CDMA Digital Cellular Variable Speech Coding Standard," *Proc. Workshop on Speech Coding for Telecommunications*, 1993, Sainte Adele, Quebec, pp. 5-6.

[15] Dudley, H., "The Vocoder," *Bell Labs Record*, 1939, **17**, pp. 122-126.

[16] Erdmann, C., *et al.*, "An Adaptive Rate Wideband Speech Codec with Adaptive Gain Re-Quantization," *IEEE Workshop on Speech Coding*, 2000, Delavan, Wisconsin.

[17] Etemoglu, C.O., V. Cuperman, and A. Gersho, "Speech Coding with an Analysis-by-Synthesis Sinusoidal Model," *Int. Conf. on Acoustics, Speech and Signal Processing*, 2000, Istanbul, Turkey, pp. 1371-1374.

[18] Gerson, I.A. and M.A. Jasiuk, "Vector Sum Excited Linear Prediction (VSELP)" in *Advances in Speech Coding*, B.S. Atal, V. Cuperman, and A. Gersho, eds. 1991, Boston, MA, Kluwer Academic Publishers, pp. 69-79.

[19] Gerson, I.A. and M.A. Jasiuk., "Techniques for Improving the Performance of CELP-type Speech Coders," *IEEE Journal Selected Areas Communications*, 1991, **10**(5), pp. 858-865.

[20] Gold, B. and N. Morgan, *Speech and Audio Signal Processing: Processing and Perception of Speech and Music*, 2000, New York, John Wiley.

[21] Gottesman, O. and A. Gersho, "High Quality Enhanced Waveform Interpolative Coding at 2.8 kbps," *Int. Conf. on Acoustics, Speech and Signal Processing*, 2000, Istanbul, Turkey, pp. 1363-1366.

[22] Greefkes, J.A., "A Digitally Companded Delta Modulation Modem for Speech Transmission," *Proc. Int. Conf. on Communications*, 1970, pp. 7.33-7.48.

[23] ISO, *Coding of Moving Pictures and Associated Audio—Audio Part*, 1993 Int. Standards Organization.

[24] ISO/IEC, *Information Technology—Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to about 1.5 Mbps, Part 3: Audio (MPEG-1)*, 1992, Int. Standards Organization.

[25] ITU-T, *Methods for Subjective Determination of Transmission Quality*, 1996, Int. Telecommunication Unit.

[26] Jarvinen, K., *et al.*, "GSM Enhanced Full Rate Speech Codec," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1997, Munich, Germany, pp. 771-774.

[27] Jayant, N.S. and P. Noll, *Digital Coding of Waveforms*, 1984. Upper Saddle River, NJ, Prentice Hall.

[28] Johnston, J.D., *et al.*, "ATT Perceptual Audio Coding (PAC)" in *Audio Engineering Society (AES) Collected Papers on Digital Audio Bit Rate Reduction*, N. Gilchrist and C. Grewin, eds. 1996, pp. 73-82.

[29] Kleijn, W.B., "Continuous Representations in Linear Predictive Coding," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1991, Toronto, Canada, pp. 201-204.

[30] Kleijn, W.B. and J. Haagen, "Transformation and Decomposition of the Speech Signal for Coding," *IEEE Signal Processing Letters*, 1994, **1**, pp. 136-138.

[31] Kleijn, W.B., D.J. Krasinski, and R.H. Ketchum, "An Efficient Stochastically Excited Linear Predictive Coding Algorithm for High Quality Low Bit Rate Transmission of Speech," *Speech Communication*, 1988, **7**, pp. 305-316.

[32] Kleijn, W.B. and K.K. Paliwal, *Speech Coding and Synthesis*, 1995, Amsterdam, Netherlands, Elsevier.

[33] Koishida, K., V. Cuperman, and A. Gersho, "A 16-KBIT/S Bandwidth Scalable Audio Coder Based on the G.729 Standard," *Int. Conf. on Acoustics, Speech and Signal Processing*, 2000, Istanbul, Turkey, pp. 1149-1152.

[34] Li, C. and V. Cuperman, "Analysis-by-Synthesis Multimode Harmonic Speech Coding at 4 kbps," *Int. Conf. on Acoustics, Speech and Signal Processing*, 2000, Istanbul, Turkey, pp. 1367-1370.

[35] McAulay, R.J. and T.F. Quateri, "Speech Analysis/Synthesis Based on a Sinusoidal Representation," *IEEE Trans. on Acoustics, Speech and Signal Processing*, 1986, **34**, pp. 744-754.

[36] McAulay, R.J. and T.F. Quateri, "Sinusoidal Coding" in *Speech Coding and Synthesis*, W.B. Kleijn and K.K. Paliwal, eds. 1995, Elsevier, pp. 121-174.

[37] McCree, A., "A 14 kbps Wideband Speech Coder with a Parametric Highband Model," *Int. Conf. on Acoustics, Speech and Signal Processing*, 2000, Istanbul, Turkey, pp. 1153-1156.

[38] McCree, A.V. and T.P. Barnwell, "Improving the Performance of a Mixed-Excitation LPC Vocoder in Acoustic Noise," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1992, San Francisco, pp. II-137-138.

[39] McCree, A.V., *et al.*, "A 2.4 kbit/s MELP Coder Candidate for the New U.S. Federal Standard," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1996, Atlanta, GA, pp. 200-203.

[40] Paez, M.D. and T.H. Glisson, "Minimum Squared-Error Quantization in Speech," *IEEE Trans. on Comm*, 1972, **20**, pp. 225-230.

[41] Painter, T. and A. Spanias, "A Review of Algorithms for Perceptual Coding of Digital Audio Signals," *Proc. Int. Conf. on DSP*, 1997, pp. 179-205.

[42] Painter, T. and A. Spanias, "Perceptual Coding of Digital Audio," *Proc. of IEEE*, 2000(April), pp. 451-513.

[43] Paliwal, K.K. and B. Atal, "Efficient Vector Quantization of LPC Parameters at 24 Bits/Frame," *IEEE Trans. on Speech and Audio Processing*, 1993, **1**(1), pp. 3-14.

[44] Prevez, M.A., H.V. Sorensen, and J.V.D. Spiegel, "An Overview of Sigma-Delta Converters," *IEEE Signal Processing Magazine*, 1996, **13**(1), pp. 61-84.

[45] Robinson, T., *Simple Lossless and Near-Lossless Waveform Compression*, 1994, Cambridge University Engineering Department.

[46] Salami, R., C. Laflamme, and B. Bessette, "Description of ITU-T Recommendation G.729 Annex A: Reduced Complexity 8 kbps CS-ACELP Codec," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1997, Munich, Germany, pp. 775-778.

[47] Schouten, J.S., F.E. DeJager, and J.A. Greefkes, *Delta Modulation, a New Modulation System for Telecommunications*, 1952, Phillips, pp. 237-245.

[48] Shlomot, E., V. Cuperman, and A. Gersho, "Combined Harmonic and Waveform Coding of Speech at Low Bit Rates," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1998, Seattle, WA, pp. 585-588.

[49] Singhal, S. and B.S. Atal, "Improving Performance of Multi-Pulse LPC Coders at Low Bit Rates," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1984, San Diego, pp. 1.3.1-1.3.4.

[50] Skoglund, J., R. Cox, and J. Collura, "A Combined WI and MELP Coder at 5.2KBPS," *Int. Conf. on Acoustics, Speech and Signal Processing*, 2000, Istanbul, Turkey, pp. 1387-1390.

[51] Smith, B., "Instantaneous Companding of Quantized Signals," *Bell Systems Technical Journal*, 1957, 36(3), pp. 653-709.

[52] Spanias, A.S., "Speech Coding: A Tutorial Review," *Proc. of the IEEE*, 1994, 82(10), pp. 1441-1582.

[53] Stachurski, J. and A. McCree, "A 4 kbps Hybrid MELP/CELP Coder with Alignment Phase Encoding and Zero Phase Equalization," *Int. Conf. on Acoustics, Speech and Signal Processing*, 2000, Istanbul, Turkey, pp. 1379-1382.

[54] Todd, C., "AC-3: Flexible Perceptual Coding for Audio Transmission and Storage," *Audio Engineering Society 96th Convention*, 1994.

[55] Tremain, T.E., *The Government Standard Linear Predictive Coding Algorithm*, in *Speech Technology Magazine*, 1982, pp. 40-49.

[56] Vary, P., *et al.*, "A Regular-Pulse Excited Linear Predictive Code," *Speech Communication*, 1988, 7(2), pp. 209-215.

[57] Wang, T., *et al.*, "A 1200 BPS Speech Coder Based on MELP," *Int. Conf. on Acoustics, Speech and Signal Processing*, 2000, Istanbul, Turkey, pp. 1375-1378.

PART III

SPEECH RECOGNITION

CHAPTER 8

Hidden Markov Models

The hidden Markov model (HMM) is a very powerful statistical method of characterizing the observed data samples of a discrete-time series. Not only can it provide an efficient way to build parsimonious parametric models, but can also incorporate the dynamic programming principle in its core for a unified pattern segmentation and pattern classification of time-varying data sequences. The data samples in the time series can be discretely or continuously distributed; they can be scalars or vectors. The underlying assumption of the HMM is that the data samples can be well characterized as a parametric random process, and the parameters of the stochastic process can be estimated in a precise and well-defined framework. The basic HMM theory was published in a series of classic papers by Baum and his colleagues [4]. The HMM has become one of the most powerful statistical methods for modeling speech signals. Its principles have been successfully used in automatic speech recognition, formant and pitch tracking, speech enhancement, speech synthesis, statistical language modeling, part-of-speech tagging, spoken language understanding, and machine translation [3, 4, 8, 10, 12, 18, 23, 37].

8.1. THE MARKOV CHAIN

A *Markov chain* models a class of random processes that incorporates a minimum amount of memory without being completely memoryless. In this subsection we focus on the discrete-time Markov chain only.

Let $\mathbf{X} = X_1, X_2, \ldots X_n$ be a sequence of random variables from a finite discrete alphabet $O = \{o_1, o_2, \ldots, o_M\}$. Based on the Bayes' rule, we have

$$P(X_1, X_2, \ldots, X_n) = P(X_1)\prod_{i=2}^{n} P(X_i \mid X_1^{i-1}) \tag{8.1}$$

where $X_1^{i-1} = X_1, X_2, \ldots, X_{i-1}$. The random variables X are said to form a first-order Markov chain, provided that

$$P(X_i \mid X_1^{i-1}) = P(X_i \mid X_{i-1}) \tag{8.2}$$

As a consequence, for the first-order Markov chain, Eq. (8.1) becomes

$$P(X_1, X_2, \ldots, X_n) = P(X_1)\prod_{i=2}^{n} P(X_i \mid X_{i-1}) \tag{8.3}$$

Equation (8.2) is also known as the *Markov assumption*. This assumption uses very little memory to model dynamic data sequences: the probability of the random variable at a given time depends only on the value at the preceding time. The Markov chain can be used to model time-invariant (stationary) events if we discard the time index i,

$$P(X_i = s \mid X_{i-1} = s') = P(s \mid s') \tag{8.4}$$

If we associate X_i to a state, the Markov chain can be represented by a finite state process with transition between states specified by the probability function $P(s|s')$. Using this finite state representation, the Markov assumption [Eq. (8.2)] is translated to the following: the probability that the Markov chain will be in a particular state at a given time depends only on the state of the Markov chain at the previous time.

Consider a Markov chain with N distinct states labeled by $\{1, \ldots, N\}$, with the state at time t in the Markov chain denoted as s_t; the parameters of a Markov chain can be described as follows:

$$a_{ij} = P(s_t = j \mid s_{t-1} = i) \quad 1 \le i, j \le N \tag{8.5}$$

$$\pi_i = P(s_1 = i) \quad 1 \le i \le N \tag{8.6}$$

where a_{ij} is the transition probability from state i to state j; and π_i is the initial probability that the Markov chain will start in state i. Both transition and initial probabilities are bound to the constraints:

$$\sum_{j=1}^{N} a_{ij} = 1; \quad 1 \le i \le N$$
$$\sum_{j=1}^{N} \pi_j = 1 \tag{8.7}$$

The Markov chain described above is also called the observable Markov model because the output of the process is the set of states at each time instance t, where each state corresponds to an observable event X_i. In other words, there is one-to-one correspondence between the observable event sequence $\mathbf{X}$ and the Markov chain state sequence $S = s_1, s_2, \ldots s_n$. Consider a simple three-state Markov chain for the Dow Jones Industrial average as shown in Figure 8.1. At the end of each day, the Dow Jones Industrial average may correspond to one of the following states:

state 1 – *up* (in comparison to the index of previous day)
state 2 – *down* (in comparison to the index of previous day)
state 3 – *unchanged* (in comparison to the index of previous day)

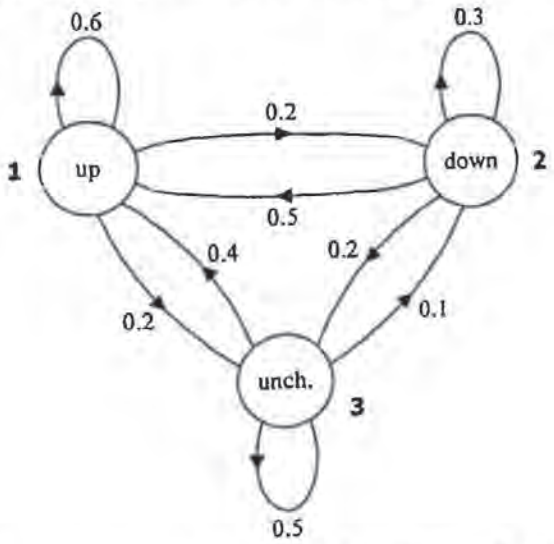


Figure 8.1 A Markov chain for the Dow Jones Industrial average. Three states represent *up*, *down*, and *unchanged*, respectively.

The parameter for this Dow Jones Markov chain may include a state-transition probability matrix

$$A = \{a_{ij}\} = \begin{bmatrix} 0.6 & 0.2 & 0.2 \\ 0.5 & 0.3 & 0.2 \\ 0.4 & 0.1 & 0.5 \end{bmatrix}$$

and an initial state probability matrix

$$\pi = (\pi_i)' = \begin{pmatrix} 0.5 \\ 0.2 \\ 0.3 \end{pmatrix}$$

Suppose you would like to find out the probability for five consecutive *up* days. Since the observed sequence of *up-up-up-up-up* corresponds to the state sequence of (1, 1, 1, 1, 1), the probability will be

$$P(5 \text{ consecutive } up \text{ days}) = P(1,1,1,1,1)$$
$$= \pi_1 a_{11} a_{11} a_{11} a_{11} = 0.5 \times (0.6)^4 = 0.0648$$

8.2. DEFINITION OF THE HIDDEN MARKOV MODEL

In the Markov chain, each state corresponds to a deterministically observable event, i.e., the output of such sources in any given state is not random. A natural extension to the Markov chain introduces a non-deterministic process that generates output observation symbols in any given state. Thus, the observation is a probabilistic function of the state. This new model is known as a *hidden Markov model*, which can be viewed as a double-embedded stochastic process with an underlying stochastic process (the state sequence) not directly observable. This underlying process can only be probabilistically associated with another observable stochastic process producing the sequence of features we can observe.

A hidden Markov model is basically a Markov chain where the output observation is a random variable X generated according to a output probabilistic function associated with each state. Figure 8.2 shows a revised hidden Markov model for the Dow Jones Industrial average. You see that each state now can generate all three output observations: *up*, *down*, and *unchanged*, according to its output pdf. This means that there is no longer a one-to-one correspondence between the observation sequence and the state sequence, so you cannot unanimously determine the state sequence for a given observation sequence, i.e., the state sequence is not observable and therefore hidden. This is why the world *hidden* is placed in front of *Markov models*. Although the state of an HMM is hidden, it often contains salient information about the data we are modeling. For example, the first state in Figure 8.2 indicates a *bull* market, and the second state indicates a *bear* market as specified by the output probability in each state. Formally speaking, a hidden Markov model is defined by:

- $O = \{o_1, o_2, \ldots, o_M\}$—An output observation alphabet.[1] The observation symbols correspond to the physical output of the system being modeled. In the case of the Dow Jones Industrial average HMM, the output observation alphabet is the collection of three categories—$O = \{up, down, unchanged\}$.

[1] Although we use the discrete output observation here, we can extend it to the continuous case with a continuous pdf. You can also use vector quantization to map a continuous vector variable into a discrete alphabet set.

- $\Omega = \{1,2,\ldots,N\}$—A set of states representing the state space. Here s_t is denoted as the state at time t. In the case of the Dow Jones Industrial average HMM, the state may indicate a bull market, a bear market, and a stable market.
- $\mathbf{A} = \{a_{ij}\}$—A transition probability matrix, where a_{ij} is the probability of taking a transition from state i to state j, i.e.,

$$a_{ij} = P(s_t = j | s_{t-1} = i) \tag{8.8}$$

- $\mathbf{B} = \{b_i(k)\}$—An output probability matrix,[2] where $b_i(k)$ is the probability of emitting symbol o_k when state i is entered. Let $\mathbf{X} = X_1, X_2, \ldots, X_t, \ldots$ be the observed output of the HMM. The state sequence $S = s_1, s_2, \ldots, s_t, \ldots$ is not observed (hidden), and $b_i(k)$ can be rewritten as follows:

$$b_i(k) = P(X_t = o_k | s_t = i) \tag{8.9}$$

- $\pi = \{\pi_i\}$—A initial state distribution where

$$\pi_i = P(s_0 = i) \quad 1 \le i \le N \tag{8.10}$$

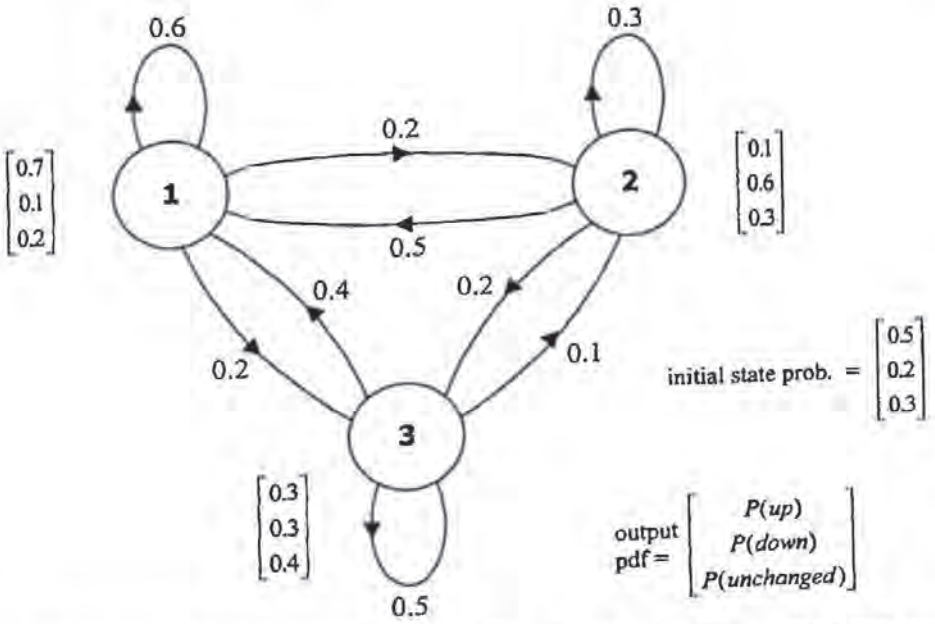


Figure 8.2 A hidden Markov model for the Dow Jones Industrial average. The three states no longer have deterministic meanings as in the Markov chain illustrated in Figure 8.1.

[2] The output distribution can also be transition-dependent. Although these two formulations look different, the state-dependent one can be reformulated as a transition-dependent one with the constraint of all the transitions entering the same state sharing the same output distribution.

Since a_{ij}, $b_i(k)$, and π_i are all probabilities, they must satisfy the following properties:

$$a_{ij} \geq 0, \quad b_i(k) \geq 0, \quad \pi_i \geq 0 \quad \forall \text{ all } i, j, k \tag{8.11}$$

$$\sum_{j=1}^{N} a_{ij} = 1 \tag{8.12}$$

$$\sum_{k=1}^{M} b_i(k) = 1 \tag{8.13}$$

$$\sum_{i=1}^{N} \pi_i = 1 \tag{8.14}$$

To sum up, a complete specification of an HMM includes two constant-size parameters, N and M, representing the total number of states and the size of observation alphabets, observation alphabet O, and three sets (matrices) of probability measures $\mathbf{A}$, $\mathbf{B}$, $\boldsymbol{\pi}$. For convenience, we use the following notation

$$\Phi = (\mathbf{A}, \mathbf{B}, \boldsymbol{\pi}) \tag{8.15}$$

to indicate the whole parameter set of an HMM and sometimes use the parameter set Φ to represent the HMM interchangeably without ambiguity.

In the first-order hidden Markov model discussed above, there are two assumptions. The first is the *Markov assumption* for the Markov chain.

$$P(s_t | s_1^{t-1}) = P(s_t | s_{t-1}) \tag{8.16}$$

where s_1^{t-1} represents the state sequence $s_1, s_2, \ldots, s_{t-1}$. The second is the *output-independence assumption*:

$$P(X_t | X_1^{t-1}, s_1^t) = P(X_t | s_t) \tag{8.17}$$

where X_1^{t-1} represents the output sequence $X_1, X_2, \ldots, X_{t-1}$. The output-independence assumption states that the probability that a particular symbol is emitted at time t depends only on the state s_t and is conditionally independent of the past observations.

You might argue that these assumptions limit the memory of the first-order hidden Markov models and may lead to model deficiency. However, in practice, they make evaluation, decoding, and learning feasible and efficient without significantly affecting the modeling capability, since those assumptions greatly reduce the number of parameters that need to be estimated.

Given the definition of HMMs above, three basic problems of interest must be addressed before they can be applied to real-world applications.

1. **The Evaluation Problem**—Given a model Φ and a sequence of observations $\mathbf{X} = (X_1, X_2, \ldots, X_T)$, what is the probability $P(\mathbf{X}|\Phi)$, i.e., the probability that the model generates the observations?
2. **The Decoding Problem**—Given a model Φ and a sequence of observations $\mathbf{X} = (X_1, X_2, \ldots, X_T)$, what is the most likely state sequence $\mathbf{S} = (s_0, s_1, s_2, \ldots, s_T)$ in the model that produces the observations?
3. **The Learning Problem**—Given a model Φ and a set of observations, how can we adjust the model parameter $\hat{\Phi}$ to maximize the joint probability (likelihood) $\prod_{\mathbf{X}} P(\mathbf{X}|\Phi)$?

If we could solve the *evaluation* problem, we would have a way of evaluating how well a given HMM matches a given observation sequence. Therefore, we could use HMM to do pattern recognition, since the likelihood $P(\mathbf{X}|\Phi)$ can be used to compute posterior probability $P(\Phi|\mathbf{X})$, and the HMM with highest posterior probability can be determined as the desired pattern for the observation sequence. If we could solve the decoding problem, we could find the best matching state sequence given an observation sequence, or, in other words, we could uncover the hidden state sequence. As discussed in Chapters 12 and 13, these are the basics for the decoding in continuous speech recognition. Last but not least, if we could solve the learning problem, we would have the means to automatically estimate the model parameter Φ from an ensemble of training data. These three problems are tightly linked under the same probabilistic framework. The efficient implementation of these algorithms shares the principle of dynamic programming that we briefly discuss next.

8.2.1. Dynamic Programming and DTW

The dynamic programming concept, also known as *dynamic time warping* (DTW) in speech recognition [40], has been widely used to derive the overall distortion between two speech templates. In these template-based systems, each speech template consists of a sequence of speech vectors. The overall distortion measure is computed from the accumulated distance between two feature vectors that are aligned between two speech templates with minimal overall distortion. The DTW method can warp two speech templates $(\mathbf{x}_1\mathbf{x}_2 \ldots \mathbf{x}_N)$ and $(\mathbf{y}_1\mathbf{y}_2 \ldots \mathbf{y}_M)$ in the time dimension to alleviate nonlinear distortion as illustrated in Figure 8.3.

This is roughly equivalent to the problem of finding the minimum distance in the trellis between these two templates. Associated with every pair (i, j) is a distance $d(i, j)$ between two speech vectors $\mathbf{x}_i$ and $\mathbf{y}_j$. To find the optimal path between starting point (1, 1) and end point (N, M) from left to right, we need to compute the optimal accumulated distance $D(N, M)$. We can enumerate all possible accumulated distance from (1,1) to (N, M) and identify the one that has the minimum distance. Since there are M possible moves for each step from left to right in Figure 8.3, all the possible paths from (1, 1) to (N, M) will be

exponential. Dynamic programming principles can drastically reduce the amount of computation by avoiding the enumeration of sequences that cannot possibly be optimal. Since the same optimal path after each step must be based on the previous step, the minimum distance $D(i, j)$ must satisfy the following equation:

$$D(i,j) = \min_{k}\left[D(i-1,k) + d(k,j)\right] \tag{8.18}$$

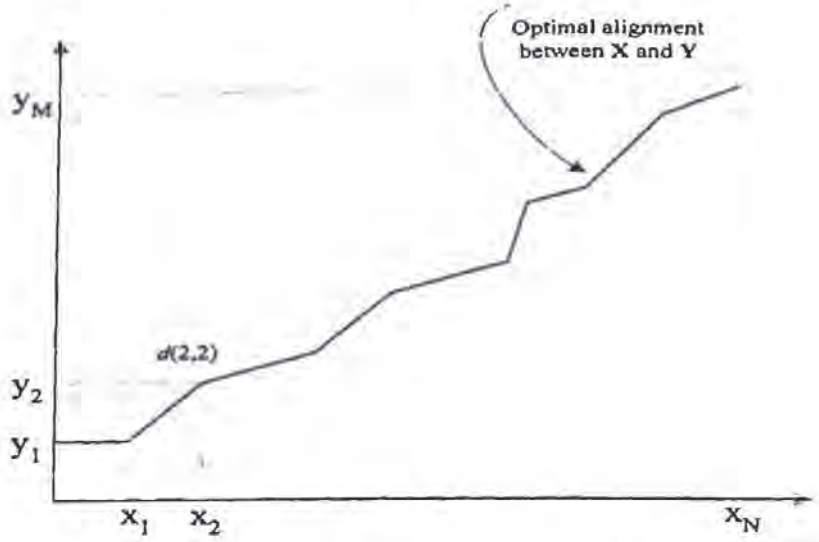


Figure 8.3 Direct comparison between two speech templates $\mathbf{X}=\mathbf{x}_1\mathbf{x}_2...\mathbf{x}_N$ and $\mathbf{Y}=\mathbf{y}_1\mathbf{y}_2...\mathbf{y}_M$.

Equation (8.18) indicates you only need to consider and keep only the best move for each pair although there are M possible moves. The recursion allows the optimal path search to be conducted incrementally from left to right. In essence, dynamic programming delegates the solution recursively to its own sub-problem. The computation proceeds from the small sub-problem ($D(i-1,k)$) to the larger sub-problem ($D(i, j)$). We can identify the optimal match $\mathbf{y}_j$ with respect to $\mathbf{x}_i$ and save the index in a back pointer table $B(i, j)$ as we move forward. The optimal path can be traced back after the optimal path is identified. The algorithm is described in Algorithm 8.1.

The advantage of the dynamic programming lies in the fact that once a sub-problem is solved, the partial result can be stored and never needs to be recalculated. This is a very important principle that you see again and again in building practical spoken language systems.

Speech recognition based on DTW is simple to implement and fairly effective for small-vocabulary speech recognition. Dynamic programming can temporally align patterns to account for differences in speaking rates across talkers as well as across repetitions of the word by the same talker. However, it does not have a principled way to derive an averaged template for each pattern from a large amount of training samples. A multiplicity of reference training tokens is typically required to characterize the variation among different utterances. As such, the HMM is a much better alternative for spoken language processing.

ALGORITHM 8.1: *THE DYNAMIC PROGRAMMING ALGORITHM*

Step 1: Initialization

$D(1,1) = d(1,1), B(1,1) = 1$, for $j = 2,\ldots,M$ compute $D(1,j) = \infty$

Step 2: Iteration

for $i = 2,\ldots,N$ {

for $j = 1,\ldots,M$ compute {

$$D(i,j) = \min_{1 \le p \le M} [D(i-1,p) + d(p,j)]$$

$$B(i,j) = \arg\min_{1 \le p \le M} [D(i-1,p) + d(p,j)] \; \}\}$$

Step 3: Backtracking and Termination

The optimal (minimum) distance is $D(N,M)$ and the optimal path is $(s_1, s_2, \ldots, s_N)$ where $s_N = M$ and $s_i = B(i+1, s_{i+1})$, $i = N-1, N-2, \ldots, 1$

8.2.2. How to Evaluate an HMM—The Forward Algorithm

To calculate the probability (likelihood) $P(\mathbf{X}|\Phi)$ of the observation sequence $\mathbf{X} = (X_1, X_2, \ldots, X_T)$, given the HMM Φ, the most intuitive way is to sum up the probabilities of all possible state sequences:

$$P(\mathbf{X}|\Phi) = \sum_{\text{all } \mathbf{S}} P(\mathbf{S}|\Phi) P(\mathbf{X}|\mathbf{S},\Phi) \tag{8.19}$$

In other words, to compute $P(\mathbf{X}|\Phi)$, we first enumerate all possible state sequences $\mathbf{S}$ of length T, that generate observation sequence $\mathbf{X}$, and then sum all the probabilities. The probability of each path $\mathbf{S}$ is the product of the state sequence probability (first factor) and the joint output probability (the second factor) along the path.

For one particular state sequence $\mathbf{S} = (s_1, s_2, \ldots, s_T)$, where s_1 is the initial state, the state-sequence probability in Eq. (8.19) can be rewritten by applying Markov assumption:

$$P(\mathbf{S}|\Phi) = P(s_1|\Phi) \prod_{t=2}^{T} P(s_t|s_{t-1},\Phi) = \pi_{s_1} a_{s_1 s_2} \ldots a_{s_{T-1} s_T} = a_{s_0 s_1} a_{s_1 s_2} \ldots a_{s_{T-1} s_T} \tag{8.20}$$

where $a_{s_0 s_1}$ denotes π_{s_1} for simplicity. For the same state sequence $\mathbf{S}$, the joint output probability along the path can be rewritten by applying the output-independent assumption:

$$\begin{aligned} P(\mathbf{X}|\mathbf{S},\Phi) &= P(X_1^T|S_1^T,\Phi) = \prod_{t=1}^{T} P(X_t|s_t,\Phi) \\ &= b_{s_1}(X_1) b_{s_2}(X_2) \ldots b_{s_T}(X_T) \end{aligned} \tag{8.21}$$

Substituting Eqs. (8.20) and (8.21) into (8.19), we get:

$$P(\mathbf{X}|\Phi) = \sum_{\text{all S}} P(\mathbf{S}|\Phi)P(\mathbf{X}|\mathbf{S},\Phi)$$
$$= \sum_{\text{all S}} a_{s_0 s_1} b_{s_1}(X_1) a_{s_1 s_2} b_{s_2}(X_2) \ldots a_{s_{T-1} s_T} b_{s_T}(X_T) \tag{8.22}$$

Equation (8.22) can be interpreted as follows. First we enumerate all possible state sequence with length T. For any given state sequence, we start from initial state s_1 with probability π_{s_1} or $a_{s_0 s_1}$. We take a transition from s_{t-1} to s_t with probability $a_{s_{t-1} s_t}$ and generate the observation X_t with probability $b_{s_t}(X_t)$ until we reach the last transition.

However, direct evaluation of Eq. (8.22) according to the interpretation above requires enumeration of $O(N^T)$ possible state sequences, which results in exponential computational complexity. Fortunately, a more efficient algorithm can be used to calculate Eq. (8.22). The trick is to store intermediate results and use them for subsequent state-sequence calculations to save computation. This algorithm is known as the *forward algorithm*.

Based on the HMM assumptions, the calculation of $P(s_t | s_{t-1}, \Phi)P(X_t | s_t, \Phi)$ involves only s_{t-1}, s_t, and X_t, so, it is possible to compute the likelihood with $P(\mathbf{X}|\Phi)$ with recursion on t. Let's define forward probability:

$$\alpha_t(i) = P(X_1^t, s_t = i|\Phi) \tag{8.23}$$

$\alpha_t(i)$ is the probability that the HMM is in state i at time t having generated partial observation X_1^t (namely $X_1 X_2 \ldots X_t$). $\alpha_t(i)$ can be calculated inductively as illustrated in Algorithm 8.2. This inductive procedure shown in Eq. (8.24) can be illustrated in a *trellis*. Figure 8.4 illustrates the computation of forward probabilities α via a trellis framework for the Dow Jones Industrial average HMM shown in Figure 8.2. Given two consecutive *up* days for the Dow Jones Industrial average, we can find the forward probability α based on the model of Figure 8.2. An arrow in Figure 8.4 indicates a transition from its origin state to its destination state. The number inside each cell indicates the forward probability α. We start the α cells from $t = 0$, where the α cells contains exactly the initial probabilities. The other cells are computed in a *time-synchronous* fashion from left to right, where each cell for time t is completely computed before proceeding to time $t+1$. When the states in the last column have been computed, the sum of all probabilities in the final column is the probability of generating the observation sequence. For most speech problems, we need to have the HMM end in some particular exit state (a.k.a final state, S_F), and we thus have $P(\mathbf{X}|\Phi) = \alpha_T(s_F)$.

It is easy to show that the complexity for the forward algorithm is $O(N^2 T)$ rather than exponential. This is because we can make full use of partially computed probabilities for the improved efficiency.

ALGORITHM 8.2: *THE FORWARD ALGORITHM*

Step 1: Initialization

$$\alpha_1(i) = \pi_i b_i(X_1) \qquad 1 \le i \le N$$

Step 2: Induction

$$\alpha_t(j) = \left[\sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij}\right] b_j(X_t) \qquad 2 \le t \le T;\ 1 \le j \le N \tag{8.24}$$

Step 3: Termination

$$P(\mathbf{X}|\Phi) = \sum_{i=1}^{N} \alpha_T(i)$$

If it is required to end in the final state, $P(\mathbf{X}|\Phi) = \alpha_T(s_F)$

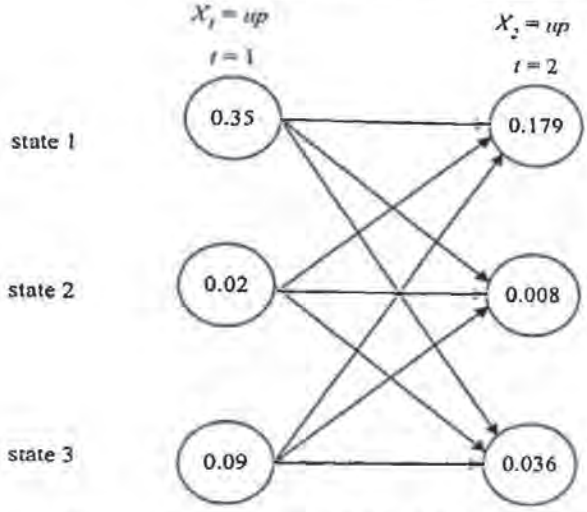


Figure 8.4 The forward trellis computation for the HMM of the Dow Jones Industrial average.

8.2.3. How to Decode an HMM—The Viterbi Algorithm

The forward algorithm, in the previous section, computes the probability that an HMM generates an observation sequence by summing up the probabilities of all possible paths, so it does not provide the best path (or state sequence). In many applications, it is desirable to find such a path. As a matter of fact, finding the best path (state sequence) is the cornerstone for searching in continuous speech recognition. Since the state sequence is hidden (unobserved) in the HMM framework, the most widely used criterion is to find the state sequence that has the highest probability of being taken while generating the observation sequence. In other words, we are looking for the state sequence $\mathbf{S} = (s_1, s_2, \ldots, s_T)$ that maximizes $P(\mathbf{S}, \mathbf{X}|\Phi)$. This problem is very similar to the optimal-path problem in dynamic programming. As a consequence, a formal technique based on dynamic programming, known

as *Viterbi* algorithm [43], can be used to find the best state sequence for an HMM. In practice, the same method can be used to evaluate HMMs that offers an approximate solution close to the case obtained using the forward algorithm described in Section 8.2.2.

The Viterbi algorithm can be regarded as the dynamic programming algorithm applied to the HMM or as a modified forward algorithm. Instead of summing up probabilities from different paths coming to the same destination state, the Viterbi algorithm picks and remembers the best path. To define the best-path probability:

$$V_t(i) = P(X_1^t, S_1^{t-1}, s_t = i \mid \Phi)$$

$V_t(i)$ is the probability of the most likely state sequence at time t, which has generated the observation X_1^t (until time t) and ends in state i. A similar induction procedure for the Viterbi algorithm can be described in Algorithm 8.3.

ALGORITHM 8.3: *THE VITERBI ALGORITHM*

Step 1: Initialization

$V_1(i) = \pi_i b_i(X_1) \qquad 1 \le i \le N$

$B_1(i) = 0$

Step 2: Induction

$$V_t(j) = \underset{1 \le i \le N}{Max}\left[V_{t-1}(i)a_{ij}\right] b_j(X_t) \quad 2 \le t \le T; \quad 1 \le j \le N \tag{8.25}$$

$$B_t(j) = \underset{1 \le i \le N}{Arg\max}\left[V_{t-1}(i)a_{ij}\right] \quad 2 \le t \le T; \quad 1 \le j \le N \tag{8.26}$$

Step 3: Termination

The best score $= \underset{1 \le i \le N}{Max}\left[V_t(i)\right]$

$s_T^* = \underset{1 \le i \le N}{Arg\max}\left[B_T(i)\right]$

Step 4: Backtracking

$s_t^* = B_{t+1}(s_{t+1}^*) \quad t = T-1, T-2, \ldots, 1$

$S^* = (s_1^*, s_2^*, \ldots, s_T^*)$ is the best sequence

This Viterbi algorithm can also be illustrated in a trellis framework similar to the one for the forward algorithm shown in Figure 8.4. Instead of summing up all the paths, Figure 8.5 illustrates the computation of V_t by picking the best path in each cell. The number inside each cell indicates the best score V_t and the best path leading to each cell is indicated by solid lines while the rest of the paths are indicated by dashed line. Again, the computation is done in a *time-synchronous* fashion from left to right. The complexity for the Viterbi algorithm is also $O(N^2T)$.

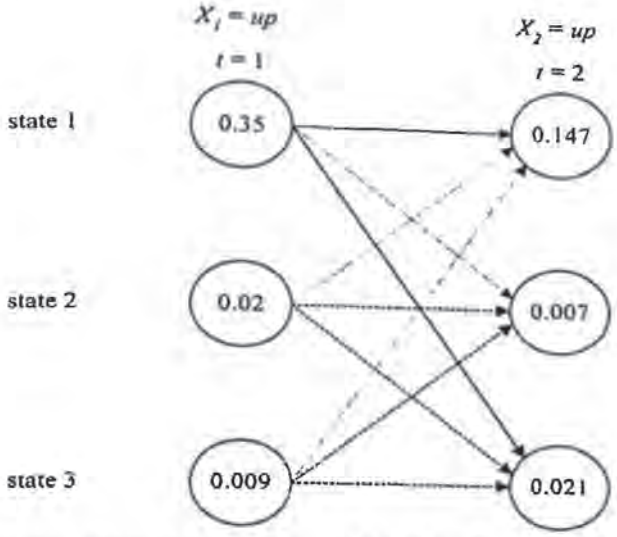


Figure 8.5 The Viterbi trellis computation for the HMM of the Dow Jones Industrial average.

8.2.4. How to Estimate HMM Parameters—Baum-Welch Algorithm

It is very important to estimate the model parameters $\Phi = (\mathbf{A}, \mathbf{B}, \pi)$ to accurately describe the observation sequences. This is by far the most difficult of the three problems, because there is no known analytical method that maximizes the joint probability of the training data in a closed form. Instead, the problem can be solved by the iterative *Baum-Welch* algorithm, also known as the *forward-backward* algorithm.

The HMM learning problem is a typical case of unsupervised learning discussed in Chapter 4, where the data is incomplete because of the hidden state sequence. The EM algorithm is perfectly suitable for this problem. As a matter of fact, Baum and colleagues used the same principle as that of the EM algorithm. Before we describe the formal Baum-Welch algorithm, we first define a few useful terms. In a manner similar to the forward probability, we define backward probability as:

$$\beta_t(i) = P(X_{t+1}^T | s_t = i, \Phi) \tag{8.27}$$

where $\beta_t(i)$ is the probability of generating partial observation X_{t+1}^T (from t+1 to the end) given that the HMM is in state i at time t, $\beta_t(i)$ can then be calculated inductively;

Initialization:

$$\beta_T(i) = 1/N \qquad 1 \le i \le N$$

Induction:

$$\beta_t(i)=\left[\sum_{j=1}^{N} a_{ij} b_j(X_{t+1})\beta_{t+1}(j)\right] \quad t=T-1\ldots1; \quad 1\le i\le N \tag{8.28}$$

The relationship of adjacent α and β (α_{t-1} & α_t and β_t & β_{t+1}) can be best illustrated as shown in Figure 8.6. α is computed recursively from left to right, and β recursively from right to left.

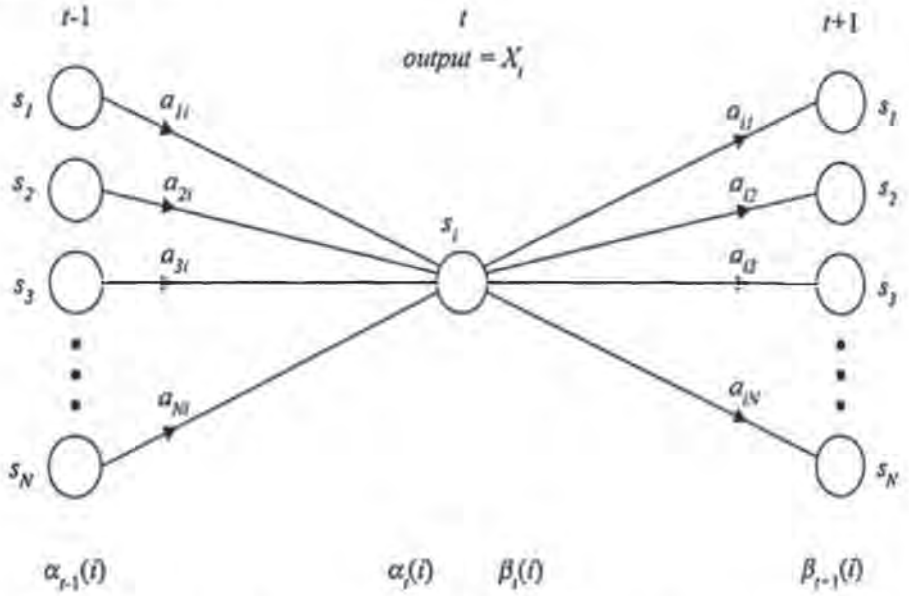


Figure 8.6 The relationship of α_{t-1} and α_t and β_t and β_{t+1} in the forward-backward algorithm.

Next, we define $\gamma_t(i,j)$, which is the probability of taking the transition from state i to state j at time t, given the model and observation sequence, i.e.,

$$\begin{aligned}\gamma_t(i,j) &= P(s_{t-1}=i, s_t=j \mid X_1^T, \Phi)\\ &= \frac{P(s_{t-1}=i, s_t=j, X_1^T \mid \Phi)}{P(X_1^T \mid \Phi)}\\ &= \frac{\alpha_{t-1}(i)a_{ij}b_j(X_t)\beta_t(j)}{\sum_{k=1}^{N}\alpha_T(k)}\end{aligned} \tag{8.29}$$

The equation above can be best illustrated as shown in Figure 8.7.

We can iteratively refine the HMM parameter vector $\Phi=(\mathbf{A},\mathbf{B},\pi)$ by maximizing the likelihood $P(\mathbf{X}|\Phi)$ for each iteration. We use $\hat{\Phi}$ to denote the new parameter vector derived from the parameter vector Φ in the previous iteration. According to the EM algorithm

of Chapter 4, the maximization process is equivalent to maximizing the following Q-function:

$$Q(\Phi,\hat{\Phi}) = \sum_{\text{all } \mathbf{S}} \frac{P(\mathbf{X},\mathbf{S}|\Phi)}{P(\mathbf{X}|\Phi)} \log P(\mathbf{X},\mathbf{S}|\hat{\Phi}) \tag{8.30}$$

where $P(\mathbf{X},\mathbf{S}|\Phi)$ and $\log P(\mathbf{X},\mathbf{S}|\hat{\Phi})$ can be expressed as:

$$P(\mathbf{X},\mathbf{S}|\Phi) = \prod_{t=1}^{T} a_{s_{t-1}s_t} b_{s_t}(X_t) \tag{8.31}$$

$$\log P(\mathbf{X},\mathbf{S}|\Phi) = \sum_{t=1}^{T} \log a_{s_{t-1}s_t} + \sum_{t=1}^{T} \log b_{s_t}(X_t) \tag{8.32}$$

Equation (8.30) can thus be rewritten as

$$Q(\Phi,\hat{\Phi}) = Q_{\mathbf{a}_i}(\Phi,\hat{\mathbf{a}}_i) + Q_{\mathbf{b}_j}(\Phi,\hat{\mathbf{b}}_j) \tag{8.33}$$

where

$$Q_{\mathbf{a}_i}(\Phi,\hat{\mathbf{a}}_i) = \sum_i \sum_j \sum_t \frac{P(\mathbf{X}, s_{t-1}=i, s_t=j|\Phi)}{P(\mathbf{X}|\Phi)} \log \hat{a}_{ij} \tag{8.34}$$

$$Q_{\mathbf{b}_j}(\Phi,\hat{\mathbf{b}}_j) = \sum_j \sum_k \sum_{t\in X_t=o_k} \frac{P(\mathbf{X}, s_t=j|\Phi)}{P(\mathbf{X}|\Phi)} \log \hat{b}_j(k) \tag{8.35}$$

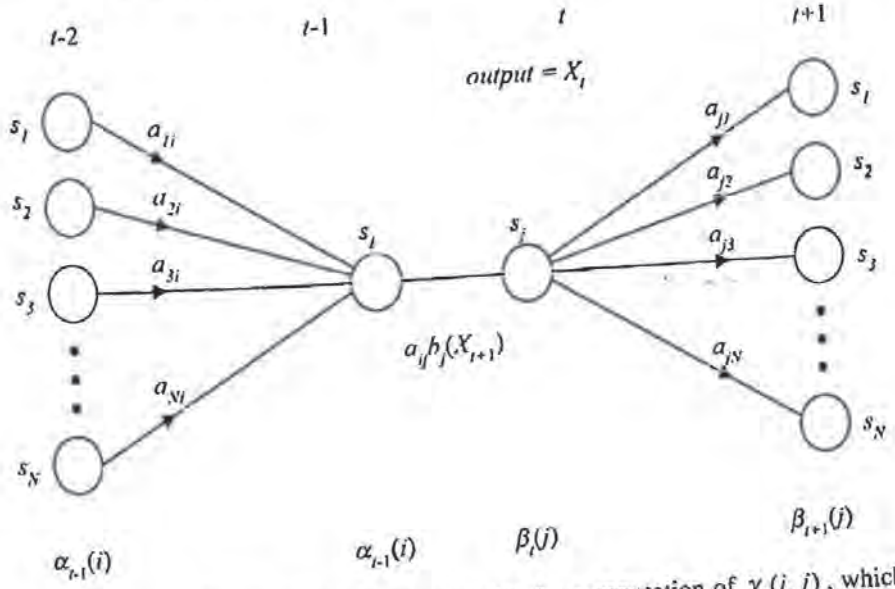


Figure 8.7 Illustration of the operations required for the computation of $\gamma_t(i,j)$, which is the probability of taking the transition from state i to state j at time t.

Since we separate the Q-function into two independent terms, the maximization procedure on $Q(\Phi, \hat{\Phi})$ can be done by maximizing the individual terms separately, subject to probability constraints.

$$\sum_{j=1}^{N} a_{ij} = 1 \quad \forall \text{ all } i \tag{8.36}$$

$$\sum_{k=1}^{M} b_j(k) = 1 \quad \forall \text{ all } j \tag{8.37}$$

Moreover, all these terms in Eqs. (8.34) and (8.35) have the following form:

$$F(x) = \sum_i y_i \log x_i \tag{8.38}$$

where $\sum_i x_i = 1$

By using the Lagrange multipliers, the function above can be proved to achieve maximum value at

$$x_i = \frac{y_i}{\sum_i y_i} \tag{8.39}$$

Using this formation, we obtain the model estimate as[3]:

$$\hat{a}_{ij} = \frac{\frac{1}{P(\mathbf{X}|\Phi)} \sum_{t=1}^{T} P(\mathbf{X}, s_{t-1} = i, s_t = j | \Phi)}{\frac{1}{P(\mathbf{X}|\Phi)} \sum_{t=1}^{T} P(\mathbf{X}, s_{t-1} = i | \Phi)} = \frac{\sum_{t=1}^{T} \gamma_t(i,j)}{\sum_{t=1}^{T} \sum_{k=1}^{N} \gamma_t(i,k)} \tag{8.40}$$

$$\hat{b}_j(k) = \frac{\frac{1}{P(\mathbf{X}|\Phi)} \sum_{t=1}^{T} P(\mathbf{X}, s_t = j | \Phi) \delta(X_t, o_k)}{\frac{1}{P(\mathbf{X}|\Phi)} \sum_{t=1}^{T} P(\mathbf{X}, s_t = j | \Phi)} = \frac{\sum_{t \in X_t = o_k} \sum_i \gamma_t(i,j)}{\sum_{t=1}^{T} \sum_i \gamma_t(i,j)} \tag{8.41}$$

By carefully examining the HMM re-estimation Eqs. (8.40) and (8.41), you can see that Eq. (8.40) is essentially the ratio between the expected number of transitions from state i to state j and the expected number of transitions from state i. For the output probability re-estimation Eq. (8.41), the numerator is the expected number of times the observation data

[3] Notice that the initial probability $\hat{\pi}_i$ can be derived as a special case of the transition probability. $\hat{\pi}_i$ is often fixed (i.e., $\hat{\pi}_i = 1$ for the initial state) for most speech applications.

emitted from state j is the observation symbol o_k, and the denominator is the expected number of times the observation data is emitted from state j.

According to the EM algorithm, the forward-backward (Baum-Welch) algorithm guarantees a monotonic likelihood improvement on each iteration, and eventually the likelihood converges to a local maximum. The forward-backward algorithm can be described in a way similar to the general EM algorithm as shown in Algorithm 8.4.

ALGORITHM 8.4: *THE FORWARD-BACKWARD ALGORITHM*

Step 1: Initialization: Choose an initial estimate Φ.

Step 2: E-step: Compute auxiliary function $Q(\Phi, \hat{\Phi})$ based on Φ.

Step 3: M-step: Compute $\hat{\Phi}$ according to the re-estimation Eqs. (8.40) and (8.41) to maximize the auxiliary Q-function.

Step 4: Iteration: Set $\Phi = \hat{\Phi}$, repeat from step 2 until convergence.

Although the forward-backward algorithm described above is based on one training observation sequence, it can be easily generalized to multiple training observation sequences under the independence assumption between these sequences. To train an HMM from M data sequences is equivalent to finding the HMM parameter vector Φ that maximizes the joint likelihood:

$$\prod_{i=1}^{M} P(\mathbf{X}_i \mid \Phi) \tag{8.42}$$

The training procedure performs the forward-backward algorithm on each independent observation sequence to calculate the expectations (or sometimes referred to as counts) in Eqs. (8.40) and (8.41). These counts in the denominator and numerator, respectively, can be added across M data sequences respectively. Finally, all the model parameters (probabilities) are normalized to make them sum up to one. This constitutes one iteration of Baum-Welch re-estimation; iteration continues until convergence. This procedure is practical and useful because it allows you to train a good HMM in a typical speech recognition scenario where a large amount of training data is available.

For example, if we let $\gamma_t^m(i,j)$ denote the $\gamma_t(i,j)$ from the m^{th} data sequence and T^m denote the corresponding length, Eq. (8.40) can be extended as:

$$\hat{a}_{ij} = \frac{\sum_{m=1}^{M}\sum_{t=1}^{T^m}\gamma_t^m(i,j)}{\sum_{m=1}^{M}\sum_{t=1}^{T^m}\sum_{k=1}^{N}\gamma_t^m(i,k)} \tag{8.43}$$

8.3. CONTINUOUS AND SEMICONTINUOUS HMMS

If the observation does not come from a finite set, but from a continuous space, the discrete output distribution discussed in the previous sections needs to be modified. The difference between the discrete and the continuous HMM lies in a different form of output probability functions. For speech recognition, use of continuous HMMs implies that the quantization procedure to map observation vectors from the continuous space to the discrete space for the discrete HMM is no longer necessary. Thus, the inherent quantization error can be eliminated.

8.3.1. Continuous Mixture Density HMMs

In choosing continuous output probability density functions $b_j(\mathbf{x})$, the first candidate is multivariate Gaussian mixture density functions. This is because they can approximate any continuous probability density function, as discussed in Chapter 3. With M Gaussian mixture density functions, we have:

$$b_j(\mathbf{x}) = \sum_{k=1}^{M} c_{jk} N(\mathbf{x}, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}) = \sum_{k=1}^{M} c_{jk} b_{jk}(\mathbf{x}) \tag{8.44}$$

where $N(\mathbf{x}, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk})$ or $b_{jk}(\mathbf{x})$ denotes a single Gaussian density function with mean vector $\boldsymbol{\mu}_{jk}$ and covariance matrix $\boldsymbol{\Sigma}_{jk}$ for state j, M denotes the number of mixture-components, and c_{jk} is the weight for the k^{th} mixture component satisfying

$$\sum_{k=1}^{M} c_{jk} = 1 \tag{8.45}$$

To take the same *divide and conquer* approach as Eq. (8.33), we need to express $b_j(\mathbf{x})$ with respect to each single mixture component as:

$$p(\mathbf{X}, \mathbf{S} \mid \Phi) = \prod_{t=1}^{T} a_{s_{t-1}s_t} b_{s_t}(\mathbf{x}_t) = \sum_{k_1=1}^{M} \sum_{k_2=1}^{M} \cdots \sum_{k_T=1}^{M} \{ \prod_{t=1}^{T} a_{s_{t-1}s_t} b_{s_t k_t}(\mathbf{x}_{s_t}) c_{s_t k_t} \} \tag{8.46}$$

Equation (8.46) can be considered as the summation of densities with all the possible state sequences S and all the possible mixture components K, defined in Ω^T as the T^{th} Cartesian product of $\Omega = \{1, 2, \ldots, M\}$, as follows:

$$p(\mathbf{X}, \mathbf{S}, \mathbf{K} \mid \Phi) = \prod_{t=1}^{T} a_{s_{t-1}s_t} b_{s_t k_t}(\mathbf{x}_t) c_{s_t k_t} \tag{8.47}$$

Therefore, the joint probability density is

$$p(\mathbf{X}|\Phi)=\sum_{\mathbf{S}}\sum_{\mathbf{K}\in\Omega^T} p(\mathbf{X},\mathbf{S},\mathbf{K}|\Phi) \tag{8.48}$$

An auxiliary function $Q(\Phi,\bar{\Phi})$ of two model points, Φ and $\hat{\Phi}$, given an observation $\mathbf{X}$, can be written as:

$$Q(\Phi,\hat{\Phi})=\sum_{\mathbf{S}}\sum_{\mathbf{K}\in\Omega^T}\frac{p(\mathbf{X},\mathbf{S},\mathbf{K}|\Phi)}{p(\mathbf{X}|\Phi)}\log p(\mathbf{X},\mathbf{S},\mathbf{K}|\hat{\Phi}) \tag{8.49}$$

From (8.47), the following decomposition can be shown:

$$\begin{aligned}&\log p(\mathbf{X},\mathbf{S},\mathbf{K}|\hat{\Phi})\\&=\sum_{t=1}^{T}\log\hat{a}_{s_{t-1}s_t}+\sum_{t=1}^{T}\log\hat{b}_{s_t k_t}(\mathbf{x}_t)+\sum_{t=1}^{T}\log\hat{c}_{s_t k_t}\end{aligned} \tag{8.50}$$

Maximization of the likelihood by way of re-estimation can be accomplished on individual parameter sets owing to the separability shown in (8.50). The separation of (8.50) is the key to the increased versatility of a re-estimation algorithm in accommodating mixture observation densities. The auxiliary function can be rewritten in a separated form in a similar manner as Eq. (8.33):

$$\begin{aligned}Q(\Phi,\hat{\Phi})&=\sum_{\mathbf{S}}\sum_{\mathbf{K}}\frac{p(\mathbf{X},\mathbf{S},\mathbf{K}|\Phi)}{p(\mathbf{X}|\Phi)}\log p(\mathbf{X},\mathbf{S},\mathbf{K}|\hat{\Phi})\\&=\sum_{i=1}^{N}Q_{\mathbf{a}_i}(\Phi,\hat{\mathbf{a}}_i)+\sum_{j=1}^{N}\sum_{k=1}^{M}Q_{\mathbf{b}_{jk}}(\Phi,\hat{\mathbf{b}}_{jk})+\sum_{j=1}^{N}\sum_{k=1}^{M}Q_{\mathbf{c}_{jk}}(\Phi,\hat{\mathbf{c}}_{jk})\end{aligned} \tag{8.51}$$

The only difference we have is:

$$Q_{\mathbf{b}_{jk}}(\Phi,\hat{\mathbf{b}}_{jk})=\sum_{t=1}^{T}p(s_t=j,k_t=k|\mathbf{X},\Phi)\log\hat{b}_{jk}(\mathbf{x}_t), \tag{8.52}$$

and

$$Q_{\mathbf{c}_{jk}}(\Phi,\hat{\mathbf{c}}_{jk})=\sum_{t=1}^{T}p(s_t=j,k_t=k|\mathbf{X},\Phi)\log\hat{c}_{jk} \tag{8.53}$$

The optimization procedure is similar to what is discussed in the discrete HMM. The only major difference is $Q_{\mathbf{b}_{jk}}(\Phi,\hat{\mathbf{b}}_{jk})$. Maximization of $Q_{\mathbf{b}_{jk}}(\Phi,\hat{\mathbf{b}}_{jk})$ with respect to $\hat{\mathbf{b}}_{jk}$ is obtained through differentiation with respect to $\{\pi_{jk},\Sigma_{jk}\}$ that satisfies:

$$\nabla_{\hat{\mathbf{b}}_{jk}}Q_{\mathbf{b}_{qk}}(\Phi,\hat{\mathbf{b}}_{jk})=0 \tag{8.54}$$

The solutions are:

$$\hat{\boldsymbol{\mu}}_{jk} = \frac{\sum_{t=1}^{T} \frac{p(\mathbf{X}, s_t = j, k_t = k \mid \Phi)}{p(\mathbf{X} \mid \Phi)} \mathbf{x}_t}{\sum_{t=1}^{T} \frac{p(\mathbf{X}, s_t = j, k_t = k \mid \Phi)}{p(\mathbf{X} \mid \Phi)}} = \frac{\sum_{t=1}^{T} \zeta_t(j,k) \mathbf{x}_t}{\sum_{t=1}^{T} \zeta_t(j,k)} \tag{8.55}$$

$$\begin{aligned} \hat{\boldsymbol{\Sigma}}_{jk} &= \frac{\sum_{t=1}^{T} \frac{p(\mathbf{X}, s_t = j, k_t = k \mid \Phi)}{p(\mathbf{X} \mid \Phi)} (\mathbf{x}_t - \hat{\boldsymbol{\mu}}_{jk})(\mathbf{x}_t - \hat{\boldsymbol{\mu}}_{jk})'}{\sum_{t=1}^{T} \frac{p(\mathbf{X}, s_t = j, k_t = k \mid \Phi)}{p(\mathbf{X} \mid \Phi)}} \\ &= \frac{\sum_{t=1}^{T} \zeta_t(j,k) (\mathbf{x}_t - \hat{\boldsymbol{\mu}}_{jk})(\mathbf{x}_t - \hat{\boldsymbol{\mu}}_{jk})'}{\sum_{t=1}^{T} \zeta_t(j,k)} \end{aligned} \tag{8.56}$$

where $\zeta_t(j,k)$ is computed as:

$$\zeta_t(j,k) = \frac{p(\mathbf{X}, s_t = j, k_t = k \mid \Phi)}{p(\mathbf{X} \mid \Phi)} = \frac{\sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} c_{jk} b_{jk}(\mathbf{x}_t) \beta_t(j)}{\sum_{i=1}^{N} \alpha_T(i)} \tag{8.57}$$

In a similar manner to the discrete HMM, we can derive the reestimation equation for c_{jk} as follows:

$$\hat{c}_{jk} = \frac{\sum_{t=1}^{T} \zeta_t(j,k)}{\sum_{t=1}^{T} \sum_{k=1}^{M} \zeta_t(j,k)} \tag{8.58}$$

8.3.2. Semicontinuous HMMs

Traditionally, the discrete and the continuous mixture density HMMs have been treated separately. In fact, the gap between them can be bridged under some minor assumptions with the so-called semicontinuous or tied-mixture HMM. It assumes the mixture density functions are tied together across all the models to form a set of shared kernels. In the discrete HMM, a VQ codebook is typically used to map the continuous input feature vector $\mathbf{x}$ to o_k, so we can use the discrete output probability distribution $b_j(k)$. The codebook can

be essentially regarded as one of such shared kernels. Accordingly, Eq. (8.44) can be modified as:

$$b_j(\mathbf{x}) = \sum_{k=1}^{M} b_j(k) f(\mathbf{x} \mid o_k) = \sum_{k=1}^{M} b_j(k) N(\mathbf{x}, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{8.59}$$

where o_k is the kth codeword, $b_j(k)$ is the same output probability distribution in the discrete HMM or the mixture weights for the continuous mixture density function, and $N(\mathbf{x}, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ are assumed to be independent of the Markov model and they are shared across all the Markov models with a very large number of mixtures M.

From the discrete HMM point of view, the needed VQ codebook consists of M continuous probability density functions, and each *codeword* has a mean vector and a covariance matrix. Typical quantization produces a codeword index that has minimum distortion to the given continuous observation $\mathbf{x}$. In the semicontinuous HMM, the quantization operation produces values of continuous probability density functions $f(\mathbf{x} \mid o_k)$ for all the codewords o_k. The structure of the semicontinuous model can be roughly the same as that of the discrete one. However, the output probabilities are no longer used directly as in the discrete HMM. In contrast, the VQ codebook density functions, $N(\mathbf{x}, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$, are combined with the discrete output probability as in Eq. (8.59). This is a combination of *discrete* model-dependent weighting coefficients with the *continuous* codebook probability density functions. Such a representation can be used to re-estimate the original VQ codebook together with the HMM.

The semicontinuous model also resembles the M-mixture continuous HMM with all the continuous output probability density functions shared among all Markov states. Compared with the continuous mixture HMM, the semicontinuous HMM can maintain the modeling ability of large-mixture probability density functions. In addition, the number of free parameters and the computational complexity can be reduced, because all the probability density functions are tied together, thus providing a good compromise between detailed acoustic modeling and trainability.

In practice, because M is large, Eq. (8.59) can be simplified by using the L most significant values $f(\mathbf{x} \mid o_k)$ for each $\mathbf{x}$ without affecting the performance. Experience has shown that values of L in the range of 1-3% of M are adequate. This can be conveniently obtained during the VQ operations by sorting the VQ output and keeping the L most significant values. Let $\eta(\mathbf{x})$ denote the set of L VQ codewords that has the largest $f(\mathbf{x} \mid o_k)$ for the given $\mathbf{x}$. Then we have:

$$b_j(\mathbf{x}) \cong \sum_{k \in \eta(\mathbf{x})} f(\mathbf{x} \mid o_k) b_j(k) \tag{8.60}$$

Since the number of mixture components in $\eta(\mathbf{x})$ is of lower order than M, Eq. (8.60) can significantly reduce the amount of computation. In fact, $\eta(\mathbf{x})$ is the key to bridge the gap between the continuous and discrete HMM. If $\eta(\mathbf{x})$ contains only the most significant

$f(\mathbf{x} \mid o_k)$ (i.e., only the closest codeword to $\mathbf{x}$), the semicontinuous HMM degenerates to the discrete HMM. On the other hand, a large VQ codebook can be used such that each Markov state could contain a number of its own codewords (a mixture of probability density functions). The discrete output probability $b_{ij}(k)$ thus becomes the mixture weights for each state. This would go to the other extreme, a standard continuous mixture density model. We can also define $\eta(\mathbf{x})$ in such a way that we can have partial tying of $f(\mathbf{x} \mid o_k)$ for different phonetic classes. For example, we can have a phone-dependent codebook.

When we have a tied VQ codebook, re-estimation of these mean vectors and covariance matrices of different models will involve interdependencies. If any observation $\mathbf{x}_t$ (no matter what model it is designated for) has a large value of posterior probability $\zeta_t(j,k)$, it will have a large contribution on re-estimation of parameters of codeword o_k. We can compute the posterior probability for each codeword from $\zeta_t(j,k)$ as defined in Eq. (8.57).

$$\zeta_t(k) = p(\mathbf{x}_t = o_k \mid \mathbf{X}, \Phi) = \sum_j \zeta_t(j,k) \tag{8.61}$$

The re-estimation formulas for the tied mixture can be written as:

$$\hat{\mu}_k = \frac{\sum_{t=1}^{T} \zeta_t(k)\mathbf{x}_t}{\sum_{t=1}^{T} \zeta_t(k)} \tag{8.62}$$

$$\hat{\Sigma}_k = \frac{\sum_{t=1}^{T} \zeta_t(k)(\mathbf{x}_t - \hat{\mu}_k)(\mathbf{x}_t - \hat{\mu}_k)'}{\sum_{t=1}^{T} \zeta_t(k)} \tag{8.63}$$

8.4. PRACTICAL ISSUES IN USING HMMS

While the HMM provides a solid framework for speech modeling, there are a number of issues you need to understand to make effective use of spoken language processing. In this section we point out some of the key issues related to practical applications. For expedience, we mostly use the discrete HMM as our example here.

8.4.1. Initial Estimates

In theory, the re-estimation algorithm of the HMM should reach a local maximum for the likelihood function. A key question is how to choose the right initial estimates of the HMM parameters so that the local maximum becomes the global maximum.

In the discrete HMM, if a probability is initialized to be zero, it will remain zero forever. Thus, it is important to have a reasonable set of initial estimates. Empirical study has shown that, for discrete HMMs, you can use a uniform distribution as the initial estimate. It works reasonably well for most speech applications, though good initial estimates are always helpful to compute the output probabilities.

If continuous mixture density HMMs are used, good initial estimates for the Gaussian density functions are essential. There are a number of ways to obtain such initial estimates:

- You can use the *k*-means clustering procedure, as used in vector quantization clustering. The Markov state segmentation can be derived from the discrete HMM, since it is not very sensitive to the initial parameters. Based on the segmented data, you can use the *k*-means algorithm to derive needed Gaussian mean and covariance parameters. The mixture coefficients can be based on the uniform distribution.
- You can estimate the semicontinuous HMM from the discrete HMM. You simply need to estimate an additional covariance matrix for each VQ codeword and run an additional four or five iterations to refine the semicontinuous HMM based on the discrete HMM, which typically requires four or five iterations from the uniform distribution. When the semi-continuous HMM is trained, you take the top *M codewords*, and in each Markov state use them as the initial Gaussian density functions for the continuous density mixture model.
- You can start training a single mixture Gaussian model. You can compute the parameters from previously segmented data. You can then iteratively split the Gaussian density function in a way similar to VQ codebook generation. You typically need two or three iterations to refine the continuous density after each splitting.

8.4.2. Model Topology

Speech is a time-evolving nonstationary signal. Each HMM state has the ability to capture some quasi-stationary segment in the non-stationary speech signal. A left-to-right topology, as illustrated in Figure 8.8, is a natural candidate to model the speech signal. It has a self-transition to each state that can be used to model contiguous speech features belonging to the same state. When the quasi-stationary speech segment evolves, the left-to-right transition enables a natural progression of such evolution. In such a topology, each state has a state-dependent output probability distribution that can be used to interpret the observable speech signal. This topology is, in fact, one of the most popular HMM structures used in state-of-the-art speech recognition systems. The output probability distribution can be either discrete distributions or a mixture of continuous density functions.

For the left-to-right HMM, the most important parameter in determining the topology is the number of states. The choice of model topology depends on available training data and what the model is used for. If each HMM is used to represent a phone, you need to have at least three to five output distributions. If such a model is used to represent a word, more states are generally required, depending on the pronunciation and duration of the word. For example, the word *tetrahydrocannabino* should have a large number of states in comparison to the word *a*. You may use at least 24 states for the former and three states for the latter. If you have the number of states depending on the duration of the signal, you may want to use 15 to 25 states for each second of speech signal. One exception is that, for silence, you may want to have a simpler topology. This is because silence is stationary, and one or two states will be sufficient.

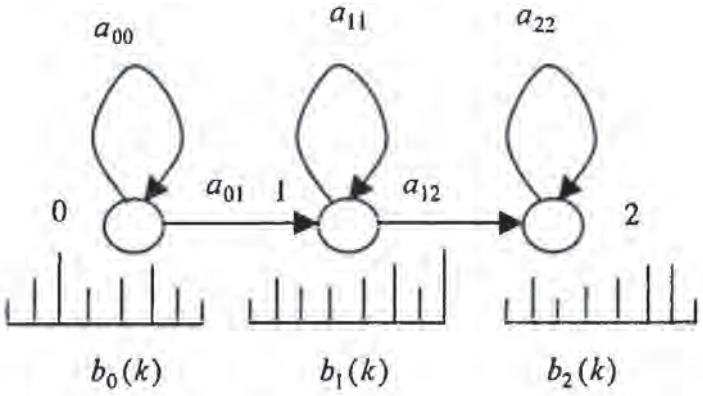


Figure 8.8 A typical hidden Markov model used to model phonemes. There are three states (0-2) and each state has an associated output probability distribution.

In practice, it is convenient to define a *null transition*. This is convenient if we want to simply traverse the HMM without consuming any observation symbol. To incorporate the null arc, you need to slightly modify the basic forward-backward or Viterbi probability equations, provided that no loops of empty transitions exist. If we denote the empty transition between state i and j as a_{ij}^{ε}, they need to satisfy the following constraints:

$$\sum_j a_{ij} + a_{ij}^{\varepsilon} = 1, \forall i \tag{8.64}$$

The forward probability can be augmented as follows:

$$\alpha_t(j) = \left[\sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(\mathbf{x}_t) + \sum_{i=1}^{N} \alpha_t(i) a_{ij}^{\varepsilon} \right] \quad 1 \le t \le T; \quad 1 \le j \le N \tag{8.65}$$

Equation (8.65) appears to have a recursion, but it actually uses the value of the same time column of $\alpha_t(i)$, provided that i is already computed, which is easily achievable if we have left-to-right empty transitions without loops of empty transitions.

8.4.3. Training Criteria

The argument for maximum likelihood estimation (MLE) is based on an assumption that the true distribution of speech is a member of the family of distributions used. This amounts to the assertion that the observed speech is genuinely produced by the HMM being used, and the only unknown parameters are the values. However, this can be challenged. Typical HMMs make many inaccurate assumptions about the speech production process, such as the output-independence assumption, the Markov assumption, and the continuous probability density assumption. Such inaccurate assumptions substantially weaken the rationale for maximum likelihood criteria. For instance, although maximum likelihood estimation is consistent (convergence to the true value), it is meaningless to have such a property if the wrong model is used. The true parameters in such cases will be the true parameters of the wrong models. Therefore, an estimation criterion that can work well in spite of these inaccurate assumptions should offer improved recognition accuracy compared with the maximum likelihood criterion. These alternative criteria include the MCE and MMIE, as discussed in Chapter 4. Finally, if we have prior knowledge about the model distribution, we can employ the Bayes' method such as MAP that can effectively combine both the prior and posterior distributions in a consistent way, which is particularly suitable for adaptation or dealing with insufficient training data.

Among all these criteria, MLE remains one of the most widely used, because of its simplicity and superior performance when appropriate assumptions are made about the system design. MCE and MMIE work well for small- to medium-vocabulary speech recognition [2, 26, 36]. You can train a number of other iterations based on the ML estimates. Neither MCE nor MMIE has been found extremely effective for large-vocabulary speech recognition. However, it is possible to combine the MMIE or MCE model with the MLE model for improved performance. This is because the error patterns generated from these different models are not the same. We can decode the test utterance with these different models and vote for the most consistent results [15, 25]. We discuss MAP methods in Chapter 9, since it is mostly helpful for speaker adaptive speech recognition.

8.4.4. Deleted Interpolation

For improved robustness, it is often necessary to combine well trained general models (such as speaker-independent) with those that are less well trained but more detailed (such as speaker-dependent). For example, you can typically improve speech recognition accuracy with speaker-dependent training. Nevertheless, you may not have sufficient data for a particular speaker so it is desirable to use a speaker-independent model that is more general but less accurate to smooth the speaker-dependent model. One effective way to achieve robust-

ness is to interpolate both models with a technique called deleted interpolation, in which the interpolation weights are estimated using cross-validation data. The objective function is to maximize the probability of the model generating the held-out data.

Now, let us assume that we want to interpolate two sets of models, $P_A(\mathbf{x})$ and $P_B(\mathbf{x})$, which can be either discrete probability distributions or continuous density functions, to form an interpolated model $P_{DI}(\mathbf{x})$. The interpolation procedure can be expressed as follows:

$$P_{DI}(\mathbf{x}) = \lambda P_A(\mathbf{x}) + (1-\lambda) P_B(\mathbf{x}) \tag{8.66}$$

where the interpolation weight λ is what we need to derive from the training data.

Consider that we want to interpolate a speaker-independent model $P_A(\mathbf{x})$ with a speaker-dependent model $P_B(\mathbf{x})$. If we use speaker-independent data to estimate the interpolation weight, we may not capture needed speaker-specific information that should be reflected in the interpolation weights. What is worse is that the interpolation weight for the speaker-independent model should be equal to 1.0 if we use the same speaker-independent data from which the model was derived to estimate the interpolation weight. This is because of the MLE criterion. If we use speaker-dependent data instead, we have the weight for the speaker-dependent model equal 1.0 without achieving the desired smoothing effect. Thus the interpolation weights need to be trained using different data or *deleted* data with the so called cross-validation method.

We can have the training data normally divided into M parts, and train a set of $P_A(\mathbf{x})$ and $P_B(\mathbf{x})$ models using the standard EM algorithm from each combination of M-1 parts, with the deleted part serving as the unseen data to estimate the interpolation weights λ. These M sets of interpolation weights are then averaged to obtain the final weights.

ALGORITHM 8.5: *DELETED INTERPOLATION PROCEDURE*

Step 1: Initialize λ with a guessed estimate.

Step 2: Update λ by the following formula:

$$\hat{\lambda} = \frac{1}{M} \sum_{j=1}^{M} \sum_{t=1}^{n_j} \frac{\lambda P_{A-j}(\mathbf{x}_t^j)}{\lambda P_{A-j}(\mathbf{x}_t^j) + (1-\lambda) P_{B-j}(\mathbf{x}_t^j)} \tag{8.67}$$

where $P_{A-j}(\mathbf{x}_t^j)$ and $P_{B-j}(\mathbf{x}_t^j)$ is $P_A(\mathbf{x})$ and $P_B(\mathbf{x})$ estimated by the entire training corpus except part j, the deleted part, respectively; n_j is the total number of data points in part j that have been aligned to estimate the model; and $\mathbf{x}_t^j$ indicates the t-th data point in the j-th set of the aligned data.

Step 3: If the new value $\hat{\lambda}$ is sufficiently close to the previous value λ, stop. Otherwise, go to Step 2.

In fact, the interpolation weights in Eq. (8.66) are similar to the Gaussian mixture weights, although $P_A(\mathbf{x})$ and $P_B(\mathbf{x})$ may not be Gaussian density functions. When we have M sets of data, we can use the same EM algorithm to estimate the interpolation weights as illustrated in Algorithm 8.5.

The deleted interpolation procedure described above can be applied after each training iteration. Then, for the following iteration of training, the learned interpolation weights can be used as illustrated in Eq. (8.66) to compute the forward-backward paths or the Viterbi maximum path. We can also have more than two distributions interpolated together. Deleted interpolation has been widely used in both acoustic and language modeling where smoothing is needed.

8.4.5. Parameter Smoothing

One simple reality for probabilistic modeling is that as many observations as possible are required to reliably estimate model parameters. However, in reality, only a finite amount of training data is available. If the training data are limited, this will result in some parameters being inadequately trained, and classification based on poorly trained models will result in higher recognition error rate. There are many possible solutions to address the problem of insufficient training data:

- You can increase the size of the training data. *There is no data like more data.*
- You can reduce the number of free parameters to be re-estimated. This has its limitations, because a number of significant parameters are always needed to model physical events.
- You can interpolate one set of parameter estimates with another set of parameter estimates, for which an adequate amount of training data exists. Deleted interpolation, discussed in Section 8.4.4, can be used effectively. In the discrete HMM, one simple approach is to set a floor to both the transition probability and the output probability in order to eliminate possible zero estimates. The same principle applies to the SCHMM as well as the mixing coefficients of the continuous density HMM. Parameter flooring can be regarded as a special case of interpolation with the uniform distribution.
- You can tie parameters together to reduce the number of free parameters. The SCHMM is a typical example of such parameter-tying techniques.

For the continuous mixture HMM, you need to pay extra attention to smoothing the covariance matrices. There are a number of techniques you can use:

- You can interpolate the covariance matrix with those that are better trained or a priori via the MAP method.

- You can tie the Gaussian covariance matrices across different mixture components or across different Markov states. A very general shared Gaussian density model is discussed in [20].
- You can use the diagonal covariance matrices if the correlation among feature coefficients is weak, which is the case if you use uncorrelated features such as the MFCC.
- You can combine these methods together.

In practice, we can reduce the speech recognition error rate by 5-20% with various smoothing techniques, depending on the available amount of training data.

8.4.6. Probability Representations

When we compute the forward and backward probabilities in the forward-backward algorithm, they will approach zero in exponential fashion if the observation sequence length, T, becomes large enough. For sufficiently large T, the dynamic range of these probabilities will exceed the precision range of essentially any machine. Thus, in practice, it will result in underflow on the computer if probabilities are represented directly. We can resolve this implementation problem by scaling these probabilities with some scaling coefficient so that they remain within the dynamic range of the computer. All of these scaling coefficients can be removed at the end of the computation without affecting the overall precision.

For example, let $\alpha_t(i)$ be multiplied by a scaling coefficient, S_t:

$$S_t = 1/\sum_i \alpha_t(i) \tag{8.68}$$

so that $\sum_i S_t\alpha_t(i) = 1$ for $1 \le t \le T$. $\beta_t(i)$ can also be multiplied by S_t for $1 \le t \le T$. The recursion involved in computing the forward and backward variables can be scaled at each stage of time t by S_t. Notice that $\alpha_t(i)$ and $\beta_t(i)$ are computed recursively in exponential fashion; therefore, at time t, the total scale factor applied to the forward variable $\alpha_t(i)$ is

$$Scale_\alpha(t) = \prod_{k=1}^{t} S_k \tag{8.69}$$

and the total scale factor applied to the backward variable $\beta_t(i)$ is

$$Scale_\beta(t) = \prod_{k=t}^{T} S_k \tag{8.70}$$

This is because the individual scaling factors are multiplied together in the forward and backward recursion. Let $\alpha'_t(i)$, $\beta'_t(i)$, and $\gamma'_t(i,j)$ denote their corresponding scaled variables, respectively. Note that

$$\sum_i \alpha'_T(i) = Scale_\alpha(T)\sum_i \alpha_T(i) = Scale_\alpha(T)P(\mathbf{X}\,|\,\Phi) \tag{8.71}$$

The scaled intermediate probability, $\gamma'_t(i,j)$, can then be written as:

$$\gamma'_t(i,j) = \frac{Scale_\alpha(t-1)\alpha_{t-1}(i)a_{ij}b_j(X_t)\beta_t(j)Scale_\beta(t)}{Scale_\alpha(T)\sum_{i=1}^{\Lambda}\alpha_T(i)} = \gamma_t(i,j) \tag{8.72}$$

Thus, the intermediate probabilities can be used in the same way as the unscaled probabilities, because the scaling factor is cancelled out in Eq. (8.72). Therefore, re-estimation formulas can be kept exactly except that $P(\mathbf{X}\,|\,\Phi)$ should be computed according to

$$P(\mathbf{X}\,|\,\Phi) = \sum_i \alpha'_T(i) / Scale_\alpha(T) \tag{8.73}$$

In practice, the scaling operation need not be performed at every observation time. It can be used at any scaling interval for which the underflow is likely to occur. In the unscaled interval, $Scale_\alpha$ can be kept as unity.

An alternative way to avoid underflow is to use a logarithmic representation for all the probabilities. This not only ensures that scaling is unnecessary, as underflow cannot happen, but also offers the benefit that integers can be used to represent the logarithmic values, thereby changing floating point operations to fixed point ones, which is particularly suitable for Viterbi-style computation, as Eq. (8.25) requires no probability addition.

In the forward-backward algorithm we need to have probability addition. We can keep a table on $\log_b P_2 - \log_b P_1$. If we represent probability P by $\log_b P$, more precision can be obtained by setting b closer to unity. Let us assume that we want to add P_1 and P_2 and that $P_1 \geq P_2$. We have:

$$\log_b(P_1 + P_2) = \log_b P_1 + \log_b(1 + b^{\log_b P_2 - \log_b P_1}) \tag{8.74}$$

If P_2 is many orders of magnitude smaller than P_1, adding the two numbers will just result in P_1. We could store all possible values of $\log_b(1+b^x)$ in a table. Using logarithms introduces errors for addition operation. In practice, double-precision float representation can be used to minimize the impact of the precision problems.

8.5. HMM LIMITATIONS

There are a number of limitations in the conventional HMMs. For example, HMMs assume the duration follows an exponential distribution, the transition probability depends only on the origin and destination, and all observation frames are dependent only on the state that generated them, not on neighboring observation frames. Researchers have proposed a number of techniques to address these limitations, albeit these solutions have not significantly improved speech recognition accuracy for practical applications.

8.5.1. Duration Modeling

One major weakness of conventional HMMs is that they do not provide an adequate representation of the temporal structure of speech. This is because the probability of state occupancy decreases exponentially with time as shown in Eq. (8.75). The probability of t consecutive observations in state i is the probability of taking the self-loop at state i for t times, which can be written as

$$d_i(t) = a_{ii}^t(1-a_{ii}) \tag{8.75}$$

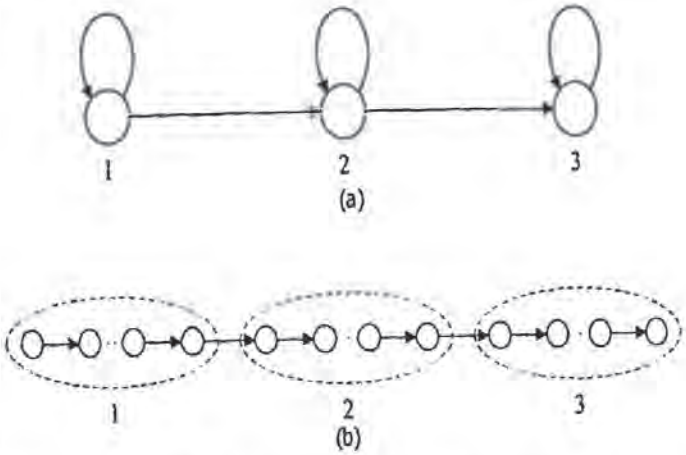


Figure 8.9 A standard HMM (a) and its corresponding explicit duration HMM (b) where the self transitions are replaced with the explicit duration probability distribution for each state.

An improvement to the standard HMM results from the use of HMMs with an explicit time duration distribution for each state [30, 39]. To explain the principle of time duration modeling, a conventional HMM with exponential state duration density and a time duration HMM with specified state duration densities (which can be either a discrete distribution or a continuous density) are illustrated in Figure 8.9. In (a), the state duration probability has an exponential form as in Eq. (8.75). In (b), the self-transition probabilities are replaced with an explicit duration probability distribution. At time t, the process enters state i for duration τ with probability density $d_i(\tau)$, during which the observations $X_{t+1}, X_{t+2} \ldots X_{t+\tau}$ are generated. It then transfers to state j with transition probability a_{ij} only after the appropriate τ observations have occurred in state i. Thus, by setting the time duration probability density to be the exponential density of Eq. (8.75) the time duration HMM can be made equivalent to the standard HMM. The parameters $d_i(\tau)$ can be estimated from observations along with the other parameters of the HMM. For expedience, the duration density is usually truncated at a maximum duration value T_d. To re-estimate the parameters of the HMM with time duration modeling, the forward recursion must be modified as follows:

$$\alpha_t(j) = \sum_{\tau} \sum_{i, i \neq j} \alpha_{t-\tau}(i) a_{ij} d_j(\tau) \prod_{l=1}^{\tau} b_j(X_{t-\tau+l}) \tag{8.76}$$

where the transition from state i to state j depends not only upon the transition probability a_{ij} but also upon all the possible time periods τ that may occur in state j. Intuitively, Eq. (8.76) illustrates that when state j is reached from previous states i, the observations may stay in state j for a period of τ with duration density $d_j(\tau)$, and each observation emits its own output probability. All possible durations must be considered, which leads to summation with respect to τ. The independence assumption of observations results in the Π term of the output probabilities. Similarly, the backward recursion can be written as:

$$\beta_t(i) = \sum_{\tau} \sum_{j, j \neq i} a_{ij} d_j(\tau) \prod_{l=1}^{\tau} b_j(X_{t+l}) \beta_{t+\tau}(j) \tag{8.77}$$

The modified Baum-Welch algorithm can then be used based on Eq. (8.76) and (8.77). The proof of the re-estimation algorithm can be based on the modified Q-function except that $P(\mathbf{X}, \mathbf{S}|\Phi)$ should be replaced with $P(\mathbf{X}, \mathbf{S}, \mathbf{T}|\Phi)$, which denotes the joint probability of observation, $\mathbf{X}$, state sequence, $\mathbf{S} = \{s_1, s_2 \ldots, s_k \ldots s_{N_s}\}$ in terms of state s_k with time duration τ_k, and the corresponding duration sequence, $\mathbf{T} = \{\tau_1, \tau_2, \ldots \tau_k \ldots \tau_{N_s}\}$.

$$Q(\Phi, \hat{\Phi}) = \frac{1}{P(\mathbf{X}|\Phi)} \sum_{\mathbf{T}} \sum_{\mathbf{S}} P(\mathbf{X}, \mathbf{S}, \mathbf{T}|\Phi) \log P(\mathbf{X}, \mathbf{S}, \mathbf{T}|\hat{\Phi}) \tag{8.78}$$

In a manner similar to the standard HMM, $\gamma_{t,\tau}(i, j)$ can be defined as the transition probability from state i at time t to state j with time duration τ in state j. $\gamma_{t,\tau}(i, j)$ can be written as:

$$\gamma_{t,\tau}(i, j) = \alpha_t(i) a_{ij} d_j(\tau) \prod_{l=1}^{\tau} b_j(X_{t+l}) \beta_{t+\tau}(j) / \sum_{k=1}^{N} \alpha_T(k) \tag{8.79}$$

Similarly, the probability of being in state j at time t with duration τ can be computed as:

$$\gamma_{t,\tau}(j) = \sum_{i} \gamma_{t,\tau}(i, j) \tag{8.80}$$

The re-estimation algorithm can be derived from Eq. (8.80), the Viterbi decoding algorithm can be used for the time duration model, and the optimal path can be determined according to:

$$V_t(j) = \max_{i} \max_{\tau} [V_{t-\tau}(i) a_{ij} d_j(\tau) \prod_{l=1}^{\tau} b_j(X_{t-\tau+l})] \tag{8.81}$$

There are drawbacks to the use of the time duration modeling discussed here. One is the great increase in computational complexity by a factor of $O(D^2)$, where D is the time

duration distribution length. Another problem is the large number of additional parameters D that must be estimated. One proposed remedy is to use a continuous density function instead of the discrete distribution $d_j(\tau)$.

In practice, duration models offer only modest improvement for speaker-independent continuous speech recognition. Many systems even eliminate the transition probability completely because the output probabilities are so dominant. Nevertheless, duration information is very effective for pruning unlikely candidates during the large-vocabulary speech recognition decoding process.

8.5.2. First-Order Assumption

As you can see from the previous section, the duration of each stationary segment captured by a single state is inadequately modeled. Another way to alleviate the duration problem is to eliminate the first-order transition assumption and to make the underlying state sequence a second-order Markov chain [32]. As a result, the transition probability between two states at time t depends on the states in which the process was at time t-1 and t-2. For a given state sequence $\mathbf{S} = \{s_1, s_2, \ldots s_T\}$, the probability of the state should be computed as:

$$P(\mathbf{S}) = \prod_t a_{s_{t-2}s_{t-1}s_t} \tag{8.82}$$

where $a_{s_{t-2}s_{t-1}s_t} = P(s_t | s_{t-2}s_{t-1})$ is the transition probability at time t, given the two-order state history. The re-estimation procedure can be readily extended based on Eq. (8.82). For example, the new forward probability can be re-defined as:

$$\alpha_t(j,k) = P(X_1^t, s_{t-1} = j, s_t = k|\lambda) = \sum_i \alpha_{t-1}(i,j) a_{ijk} b_k(X_t) \tag{8.83}$$

where $a_{ijk} = P(s_t = k | s_{t-2} = i, s_{t-1} = j)$. Similarly, we can define the backward probability as:

$$\beta_t(i,j) = P(X_{t+1}^T | s_{t-1} = i, s_t = j, \lambda) = \sum_k a_{ijk} b_k(X_{t+1}) \beta_{t+1}(j,k) \tag{8.84}$$

With Eq. (8.83) and (8.84), the MLE estimates can be derived easily based on the modified $\gamma_t(i,j,k)$:

$$\begin{aligned} \gamma_t(i,j,k) &= P(s_{t-1} = i, s_t = j, s_{t+1} = k, \mathbf{X}|\Phi) \\ &= \alpha_t(i,j) a_{ijk} b_k(X_{t+1}) \beta_{t+1}(j,k) / P(\mathbf{X}|\Phi) \end{aligned} \tag{8.85}$$

In practice, the second-order model is computationally very expensive as we have to consider the increased state space, which can often be realized with an equivalent first-order hidden Markov model on the two-fold product state space. It has not offered significantly improved accuracy to justify its increase in computational complexity for most applications.

8.5.3. Conditional Independence Assumption

The third major weakness in HMMs is that all observation frames are dependent only on the state that generated them, not on neighboring observation frames. The conditional independence assumption makes it hard to effectively handle nonstationary frames that are strongly correlated. There are a number of ways to alleviate the conditional independence assumption [34]. For example, we can assume the output probability distribution depends not only on the state but also on the previous frame. Thus, the probability of a given state sequence can be rewritten as:

$$P(\mathbf{X}|\mathbf{S},\Phi) = \prod_{t=1}^{T} P(X_t|X_{t-1},s_t,\Phi) \tag{8.86}$$

As the parameter space becomes huge, we often need to quantize X_{t-1} into a smaller set of codewords so that we can keep the number of free parameters under control. Thus, Eq. (8.86) can be simplified as:

$$P(\mathbf{X}|\mathbf{S},\Phi) = \prod_{t=1}^{T} P(X_t|\Re(X_{t-1}),s_t,\Phi) \tag{8.87}$$

where $\Re(\,)$ denotes the quantized vector that has a small codebook size, L. Although this can dramatically reduce the space of the free conditional output probability distributions, the total number of free parameters will still increase by L times.

The re-estimation for conditional dependent HMMs can be derived with the modified Q-function, as discussed in the previous sections. In practice, it has not demonstrated convincing accuracy improvement for large-vocabulary speech recognition.

8.6. Historical Perspective and Further Reading

The Markov chain was named after Russian scientist A. Markov for his pioneering work in analyzing the letter sequence in the text of a literary work in 1913 [33]. In the 1960s, Baum and others further developed efficient methods for training the model parameters [4, 5]. When the observation is real valued, the use of continuous or semi-continuous HMMs can improve the overall performance. Baum et al. also developed the method to use continuous density functions that are strictly log concave [5], which was relaxed by Liporace [31] and expanded by Juang to include mixture density functions [27].

The Viterbi algorithm shares the same concept that was independently discovered by researchers in many separate fields [28], including Vintsyuk [42], Needleman and Wunsch [35], Sankoff [41], Sakoe and Chiba [40], and Wagner and Fischer [44].

Jim Baker did his Ph.D. thesis under Raj Reddy at Carnegie Mellon using HMMs for speech recognition [3]. At the same time Fred Jelinek and his colleagues at IBM Research pioneered widespread applications [23]. Since the 1980s, partly because of the DARPA-

funded speech projects, HMMs have become a mainstream technique for modeling speech, as exemplified by advanced systems developed at BBN, Bell Labs, Carnegie Mellon, IBM, Microsoft, SRI, and others [9, 17, 29, 46]. The Ph.D. theses from Kai-Fu Lee [29], Hsiao-Wuen Hon [16], and Mei-Yuh Hwang [22] at Carnegie Mellon addressed many important practical issues in using HMMs for speech recognition. There are also a number of good books on the practical use of HMMs [18, 24, 38, 45].

The choice of different output probabilities depends on a number of factors such as the availability of training data, the feature characteristics, the computational complexity, and the number of free parameters [19] [34]. The semi-continuous model, also known as the tied-mixture model, was independently proposed by Huang and Jack [21] and Bellegarda and Nahamoo [6]. Other improvements include explicit duration modeling [1, 11, 13, 14, 30, 39], high-order and conditional models [7, 32, 34], which have yet to be shown effective for practical speech recognition.

Both Carnegie Mellon University's open speech software[4] and Cambridge University's HTK[5] are a good starting point for those interested in using the existing tools for running experiments.

HMMs have become the most prominent techniques for speech recognition today. Most of the state-of-the-art speech recognition systems on the market are based on HMMs described in this chapter.

REFERENCES

[1] Anastasakos, A., R. Schwartz, and H. Sun, "Duration Modeling in Large Vocabulary Speech Recognition" in *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1995, Detroit, MI, pp. 628-631.

[2] Bahl, L.R., *et al.*, "Speech Recognition with Continuous-Parameter Hidden Markov Models," *Computer Speech and Language*, 1987, **2**, pp. 219-234.

[3] Baker, J.K., "The DRAGON System—An Overview," *Trans. on Acoustics, Speech and Signal Processing*, 1975, **23**(1), pp. 24-29.

[4] Baum, L.E. and J.A. Eagon, "An Inequality with Applications to Statistical Estimation for Probabilistic Functions of Markov Processes and to a Model for Ecology," *Bulletin of American Mathematical Society*, 1967, **73**, pp. 360-363.

[5] Baum, L.E., *et al.*, "A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains," *Annals of Mathematical Statistics*, 1970, **41**, pp. 164-171.

[6] Bellegarda, J.R. and D. Nahamoo, "Tied Mixture Continuous Parameter Models for Large Vocabulary Isolated Speech Recognition," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1989, pp. 13-16.

[7] Brown, P., *The Acoustic-Modeling Problem in Automatic Speech Recognition*, Ph.D. Thesis in *Computer Science*, 1987, Carnegie Mellon University, Pittsburgh.

[4] http://www.speech.cs.cmu.edu/sphinx/

[5] http://htk.eng.cam.ac.uk/

[8] Brown, P.F., *et al.*, "The Mathematics of Statistical Machine Translation: Parameter Estimation," *Computational Linguistics*, 1995, **19**(2), pp. 263-312.

[9] Chou, W., C.H. Lee, and B.H. Juang, "Minimum Error Rate Training of Inter-Word Context Dependent Acoustic Model Units in Speech Recognition" in *Proc. of the Int. Conf. on Spoken Language Processing*, 1994, Yokohama, Japan, pp. 439-442.

[10] Church, K., "A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text," *Proc. of the Second Conf. on Applied Natural Language Processing*, 1988, Austin, Texas, pp. 136-143.

[11] Deng, L., M. Lennig, and P. Mermelstein, "Use of Vowel Duration Information in a Large Vocabulary Word Recognizer," *Journal of the Acoustical Society of America*, 1989, **86**(2 August), pp. 540-548.

[12] DeRose, S.J., "Grammatical Category Disambiguation by Statistical Optimization," *Computational Linguistics*, 1988(1), pp. 31-39.

[13] Dumouchel, P. and D. Shaughnessy, "Segmental Duration and HMM Modeling," *Proc. of the European Conf. on Speech Communication and Technology*, 1995, Madrid, Spain, pp. 803-806.

[14] Ferguson, J.D., "Variable Duration Models for Speech" in *Proc. of the Symposium on the Application of Hidden Markov Models to Text and Speech*, J.D. Ferguson, ed., 1980, Princeton, New Jersey, pp. 143-179.

[15] Fiscus, J., "A Post-Processing System to Yield Reduced Word Error Rates: Recognizer Output Voting Error Reduction (ROVER)," *IEEE Workshop on Automatic Speech Recognition and Understanding*, 1997, Santa Barbara, CA, pp. 347-352.

[16] Hon, H.W., *Vocabulary-Independent Speech Recognition: The VOCIND System*, Ph.D Thesis in *Department of Computer Science* 1992, Carnegie Mellon University, Pittsburgh.

[17] Huang, X.D., *et al.*, "The SPHINX-II Speech Recognition System: An Overview," *Computer Speech and Language*, 1993, pp. 137-148.

[18] Huang, X.D., Y. Ariki, and M.A. Jack, *Hidden Markov Models for Speech Recognition*, 1990, Edinburgh, U.K., Edinburgh University Press.

[19] Huang, X.D., *et al.*, "A Comparative Study of Discrete, Semicontinuous, and Continuous Hidden Markov Models," *Computer Speech and Language*, 1993, 7(4), pp. 359-368.

[20] Huang, X.D., *et al.*, "Deleted Interpolation and Density Sharing for Continuous Hidden Markov Models," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1996.

[21] Huang, X.D. and M.A. Jack, "Semi-Continuous Hidden Markov Models with Maximum Likelihood Vector Quantization" in *IEEE Workshop on Speech Recognition*, 1988.

[22] Hwang, M., *Subphonetic Modeling in HMM-based Speech Recognition Systems*, Ph.D. Thesis in *Computer Science*, 1994, Carnegie Mellon University, Pittsburgh.

[23] Jelinek, F., "Continuous Speech Recognition by Statistical Methods," *Proc. of the IEEE*, 1976, **64**(4), pp. 532-556.

[24] Jelinek, F., *Statistical Methods for Speech Recognition*, 1998, Cambridge, MA, MIT Press.

[25] Jiang, L. and X. Huang, "Unified Decoding and Feature Representation for Improved Speech Recognition," *Proc. of the 6th European Conf. on Speech Communication and Technology*, 1999, Budapest, Hungary, pp. 1331-1334.

[26] Juang, B.H., W. Chou, and C.H. Lee, "Statistical and Discriminative Methods for Speech Recognition" in *Automatic Speech and Speaker Recognition—Advanced Topics*, C.H. Lee, F.K. Soong, and K.K. Paliwal, eds., 1996, Boston, pp. 109-132, Kluwer Academic Publishers.

[27] Juang, B.H., S.E. Levinson, and M.M. Sondhi, "Maximum Likelihood Estimation for Multivariate Mixture Observations of Markov Chains," *IEEE Trans. on Information Theory*, 1986, **IT-32**(2), pp. 307-309.

[28] Kruskal, J., "An Overview of Sequence Comparison" in *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, D. Sankoff and J. Kruskal, eds. 1983, Reading, MA., Addison-Wesley, pp. 1-44.

[29] Lee, K.F., *Large-Vocabulary Speaker-Independent Continuous Speech Recognition: The SPHINX System*, Ph.D. Thesis in Computer Science Dept., 1988, Carnegie Mellon University, Pittsburgh.

[30] Levinson, S.E., "Continuously Variable Duration Hidden Markov Models for Automatic Speech Recognition," *Computer Speech and Language*, 1986, pp. 29-45.

[31] Liporace, L.R., "Maximum Likelihood Estimation for Multivariate Observations of Markov Sources," *IEEE Trans. on Information Theory*, 1982, **28**, pp. 729-734.

[32] Mari, J., J. Haton, and A. Kriouile, "Automatic Word Recognition Based on Second-Order Hidden Markov Models," *IEEE Trans. on Speech and Audio Processing*, 1977, **5**(1), pp. 22-25.

[33] Markov, A.A., "An Example of Statistical Investigation in the Text of 'Eugene Onyegin', Illustrating Coupling of Tests in Chains," *Proc. of the Academy of Sciences of St. Petersburg*, 1913, Russia, pp. 153-162.

[34] Ming, J. and F. Smith, "Modelling of the Interframe Dependence in an HMM Using Conditional Gaussian Mixtures," *Computer Speech and Language*, 1996, **10**(4), pp. 229-247.

[35] Needleman, S. and C. Wunsch, "A General Method Applicable to the Search for Similarities in the Amino-acid Sequence of Two Proteins," *Journal of Molecular Biology*, 1970, **48**, pp. 443-453.

[36] Normandin, Y., "Maximum Mutual Information Estimation of Hidden Markov Models" in *Automatic Speech and Speaker Recognition*, C.H. Lee, F.K. Soong, and K.K. Paliwal, eds., 1996, Norwell, MA, Kluwer Academic Publishers.

[37] Rabiner, L.R., "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," *Proc. of IEEE*, 1989, **77**(2), pp. 257-286.

[38] Rabiner, L.R. and B.H. Juang, *Fundamentals of Speech Recognition*, Prentice Hall Signal Processing Series, ed. A.V. Oppenheim, 1993, Englewood Cliffs, NJ, Prentice-Hall.

[39] Russell, M.J. and R.K. Moore, "Explicit Modeling of State Occupancy in Hidden Markov Models for Automatic Speech Recognition," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1985, pp. 5-8.

[40] Sakoe, H. and S. Chiba, "Dynamic Programming Algorithm Optimization for Spoken Word Recognition," *IEEE Trans. on Acoustics, Speech and Signal Processing*, 1978, **26**(1), pp. 43-49.

[41] Sankoff, D., "Matching Sequences under Deletion-Insertion Constraints," *Proc. of the National Academy of Sciences*, 1972, **69**, pp. 4-6.

[42] Vintsyuk, T.K., "Speech Discrimination by Dynamic Programming," *Cybernetics*, 1968, **4**(1), pp. 52-57.

[43] Viterbi, A.J., "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," *IEEE Trans. on Information Theory*, 1967, **13**(2), pp. 260-269.

[44] Wagner, R. and M. Fischer, "The String-to-String Correction Problem," *Journal of the ACM*, 1974, **21**, pp. 168-173.

[45] Waibel, A.H. and K.F. Lee, *Readings in Speech Recognition*, 1990, San Mateo, CA, Morgan Kaufman Publishers.

[46] Young, S.J. and P.C. Woodland, "The Use of State Tying in Continuous Speech Recognition," *Proc. of Eurospeech*, 1993, Berlin, pp. 2203-2206.

CHAPTER 9

Acoustic Modeling

After years of research and development, accuracy of automatic speech recognition remains one of the most important research challenges. A number of well-known factors determines accuracy; those most noticeable are variations in context, in speaker, and in environment. Acoustic modeling plays a critical role in improving accuracy and is arguably the central part of any speech recognition system.

For the given acoustic observation $\mathbf{X} = X_1 X_2 ... X_n$, the goal of speech recognition is to find out the corresponding word sequence $\hat{\mathbf{W}} = w_1 w_2 ... w_m$ that has the maximum posterior probability $P(\mathbf{W}|\mathbf{X})$ as expressed by Eq. (9.1).

$$\hat{\mathbf{W}} = \arg\max_{\mathbf{W}} P(\mathbf{W}|\mathbf{X}) = \arg\max_{\mathbf{W}} \frac{P(\mathbf{W})P(\mathbf{X}|\mathbf{W})}{P(\mathbf{X})} \tag{9.1}$$

Since the maximization of Eq. (9.1) is carried out with the observation $\mathbf{X}$ fixed, the above maximization is equivalent to maximization of the following equation:

$$\hat{\mathbf{W}} = \arg\max_{\mathbf{W}} P(\mathbf{W})P(\mathbf{X} \mid \mathbf{W}) \tag{9.2}$$

The practical challenge is how to build accurate acoustic models, $P(\mathbf{X}|\mathbf{W})$, and language models, $P(\mathbf{W})$, that can truly reflect the spoken language to be recognized. For large-vocabulary speech recognition, since there are a large number of words, we need to decompose a word into a subword sequence. Thus $P(\mathbf{X}|\mathbf{W})$ is closely related to phonetic modeling. $P(\mathbf{X}|\mathbf{W})$ should take into account speaker variations, pronunciation variations, environmental variations, and context-dependent phonetic coarticulation variations. Last, but not least, any static acoustic or language model will not meet the needs of real applications. So it is vital to dynamically adapt both $P(\mathbf{W})$ and $P(\mathbf{X}|\mathbf{W})$ to maximize $P(\mathbf{W}|\mathbf{X})$ while using spoken language systems. The decoding process of finding the best word sequence $\mathbf{W}$ to match the input speech signal $\mathbf{X}$ in speech recognition systems is more than a simple pattern recognition problem, since in continuous speech recognition you have an infinite number of word patterns to search, as discussed in detail in Chapters 12 and 13.

In this chapter we focus on discussing solutions that work well in practice. To highlight solutions that are effective, we use the Whisper speech recognition system [49] developed at Microsoft Research as a concrete example to illustrate how to build a working system and how various techniques can help to reduce speech recognition errors.[1] We hope that by studying what worked well in the past we can illuminate the possibilities for further improvement of the state of the art.

The hidden Markov model we discussed in Chapter 8 is the underpinning for acoustic phonetic modeling. It provides a powerful way to integrate segmentation, time warping, pattern matching, and context knowledge in a unified manner. The underlying technologies are undoubtedly evolving, and the research community is aggressively searching for more powerful solutions. Most of the techniques discussed in this chapter can be readily derived from the fundamentals discussed in earlier chapters.

9.1. VARIABILITY IN THE SPEECH SIGNAL

The research community has produced technologies that, with some constraints, can accurately recognize spoken input. Admittedly, today's state-of-the-art systems still cannot match humans' performance. Although we can build a very accurate speech recognizer for a particular speaker, in a particular language and speaking style, in a particular environment, and limited to a particular task, it remains a research challenge to build a recognizer that can essentially understand anyone's speech, in any language, on any topic, in any free-flowing style, and in almost any speaking environment.

[1] Most of the experimental results used here are based on a development test set for the 60,000-word speaker-independent continuous dictation task. The training set consists of 35,000 utterances from about 300 speakers. The test set consists of 410 utterances from 10 speakers that were not used in the training data. The language model is derived from 2 billion words of English text corpora.

Accuracy and robustness are the ultimate measures for the success of speech recognition algorithms. There are many reasons why existing algorithms or systems did not deliver what people want. In the sections that follow we summarize the major factors involved.

9.1.1. Context Variability

Spoken language interaction between people requires knowledge of word meanings, communication context, and common sense. Words with widely different meanings and usage patterns may have the same phonetic realization. Consider the challenge represented by the following utterance:

> *Mr. Wright should write to Ms. Wright right away about his Ford or four door Honda.*

For a given word with the same pronunciation, the meaning could be dramatically different, as indicated by *Wright*, *write*, and *right*. What makes it even more difficult is that *Ford or* and *Four Door* are not only phonetically identical, but also semantically relevant. The interpretation is made within a given word boundary. Even with smart linguistic and semantic information, it is still impossible to decipher the correct word sequence, unless the speaker pauses between words or uses intonation to set apart these semantically confusable phrases.

In addition to the context variability at word and sentence level, you can find dramatic context variability at the phonetic level. As illustrated in Figure 9.1, the acoustic realization of phoneme */eel* for word *peat* and *wheel* depends on its left and right context. The dependency becomes more important in fast speech or spontaneous speech conversations, since many phonemes are not fully realized.

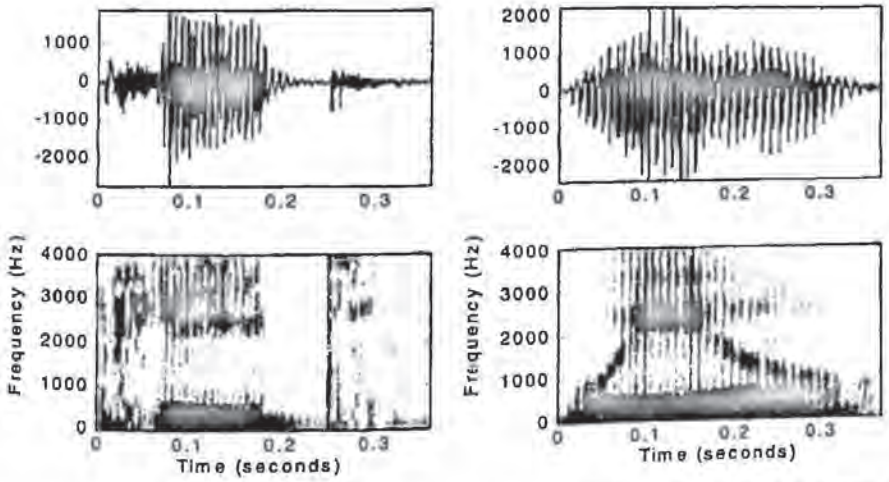


Figure 9.1 Waveforms and spectrograms for words *peat* (left) and *wheel* (right). The phoneme */ee/* is illustrated with two different left and right contexts. This illustrates that different contexts may have different effects on a phone.

9.1.2. Style Variability

To deal with acoustic realization variability, a number of constraints can be imposed on the use of the speech recognizer. For example, we can have an *isolated* speech recognition system, in which users have to pause between each word. Because the pause provides a clear boundary for the word, we can easily eliminate errors such as *Ford or* and *four door*. In addition, isolated speech provides a correct silence context to each word so that it is easier to model and decode the speech, leading to a significant reduction in computational complexity and error rate. In practice, the word-recognition error rate of an isolated speech recognizer can typically be reduced by more than a factor of three (from 7% to 2%) as compared with to a comparable continuous speech recognition system [5]. The disadvantage is that such an isolated speech recognizer is unnatural to most people. The throughput is also significantly lower than that for continuous speech.

In continuous speech recognition, the error rate for casual, spontaneous speech, as occurs in our daily conversation, is much higher than for carefully articulated read-aloud speech. The rate of speech also affects the word recognition rate. It is typical that the higher the *speaking rate* (words/minute), the higher the error rate. If a person whispers, or shouts, to reflect his or her emotional changes, the variation increases even more significantly.

9.1.3. Speaker Variability

Every individual speaker is different. The speech he or she produces reflects the physical vocal tract size, length and width of the neck, a range of physical characteristics, age, sex, dialect, health, education, and personal style. As such, one person's speech patterns can be entirely different from those of another person. Even if we exclude these interspeaker differences, the same speaker is often unable to precisely produce the same utterance. Thus, the shape of the vocal tract movement and rate of delivery may vary from utterance to utterance, even with dedicated effort to minimize the variability.

For *speaker-independent* speech recognition, we typically use more than 500 speakers to build a combined model. Such an approach exhibits large performance fluctuations among new speakers because of possible mismatches in the training data between exiting speakers and new ones [50]. In particular, speakers with accents have a tangible error-rate increase of 2 to 3 times.

To improve the performance of a speaker-independent speech recognizer, a number of constraints can be imposed on its use. For example, we can have a user enrollment that requires the user to speak for about 30 minutes. With the *speaker-dependent* data and training, we may be able to capture various speaker-dependent acoustic characteristics that can significantly improve the speech recognizer's performance. In practice, speaker-dependent speech recognition offers not only improved accuracy but also improved speed, since decoding can be more efficient with an accurate acoustic and phonetic model. A typical speaker-dependent speech recognition system can reduce the word recognition error by more than 30% as compared with a comparable speaker-independent speech recognition system.

The disadvantage of speaker-dependent speech recognition is that it takes time to collect speaker-dependent data, which may be impractical for some applications such as an automatic telephone operator. Many applications have to support walk-in speakers, so speaker-independent speech recognition remains an important feature. When the amount of speaker-dependent data is limited, it is important to make use of both speaker-dependent and speaker-independent data using *speaker-adaptive* training techniques, as discussed in Section 9.6. Even for speaker-independent speech recognition, you can still use speaker-adaptive training based on recognition results to quickly adapt to each individual speaker during usage.

9.1.4. Environment Variability

The world we live in is full of sounds of varying loudness from different sources. When we interact with computers, we may have people speaking in the background. Someone may slam the door, or the air conditioning may start humming without notice. If speech recognition is embedded in mobile devices, such as PDAs (personal digital assistants) or cellular phones, the spectrum of noises varies significantly because the owner moves around. These external parameters, such as the characteristics of the environmental noise and the type and placement of the microphone, can greatly affect speech recognition system performance. In addition to the background noises, we have to deal with noises made by speakers, such as lip smacks and noncommunication words. Noise may also be present from the input device itself, such as microphone and A/D interference noises.

In a similar manner to speaker-independent training, we can build a system by using a large amount of data collected from a number of environments; this is referred to as *multistyle training* [70]. We can use adaptive techniques to normalize the mismatch across different environment conditions in a manner similar to speaker-adaptive training, as discussed in Chapter 10. Despite the progress being made in the field, environment variability remains as one of the most severe challenges facing today's state-of-the-art speech systems.

9.2. How to Measure Speech Recognition Errors

It is critical to evaluate the performance of speech recognition systems. The word recognition error rate is widely used as one of the most important measures. When you compare different acoustic modeling algorithms, it is important to compare their relative error reduction. Empirically, you need to have a test data set that contains more than 500 sentences (with 6 to 10 words for each sentence) from 5 to 10 different speakers to reliably estimate the recognition error rate. Typically, you need to have more than 10% relative error reduction to consider adopting a new algorithm.

As a sanity check, you may want to use a small sample from the training data to measure the performance of the *training set*, which is often much better than what you can get from testing new data. Training-set performance is useful in the development stage to identify potential implementation bugs. Eventually, you need to use a *development set* that typically consists of data never used in training. Since you may tune a number of parameters

with your development set, it is important to evaluate performance of a *test set* after you decide the optimal parameter setting. The test set should be completely new with respect to both training and parameter tuning.

There are typically three types of word recognition errors in speech recognition:

- *Substitution*: an incorrect word was substituted for the correct word
- *Deletion*: a correct word was omitted in the recognized sentence
- *Insertion*: an extra word was added in the recognized sentence[2]

For instance, a speech recognition system may produce an incorrect result as follows, where substitutions are ***bold***, insertions are underlined, and deletions are denoted as **. There are four errors in this example.

> **Correct:** *Did mob mission area of the Copeland ever go to m4 in nineteen eighty one*
> **Recognized:** *Did mob mission area ** **the copy** land ever go to m4 in nineteen **east** one*

To determine the minimum error rate, you can't simply compare two word sequences one by one. For example, suppose you have utterance *The effect is clear* recognized as *Effect is not clear*. If you compare word to word, the error rate is 75% (*The* vs. *Effect*, *effect* vs. *is*, *is* vs. *not*). In fact, the error rate is only 50% with one deletion (*The*) and one insertion (*not*). In general, you need to align a recognized word string against the correct word string and compute the number of substitutions (*Subs*), deletions (*Dels*), and insertions (*Ins*). The *Word Error Rate* is defined as:

$$\text{Word Error Rate} = 100\% \times \frac{Subs + Dels + Ins}{\text{No. of words in the correct sentenc}} \tag{9.3}$$

This alignment is also known as the *maximum substring matching* problem, which can be easily handled by the dynamic programming algorithm discussed in Chapter 8.

Let the correct word string be $w_1 w_2 \cdots w_n$, where w_i denotes the ith word in the correct word string, and the recognized word string be $\hat{w}_1 \hat{w}_2 \cdots \hat{w}_m$, where $\hat{w}_i$ denotes the i^{th} word in the recognized word string. We denote $R[i, j]$ as the minimum error of aligning substring $w_1 w_2 \cdots w_n$ against substring $\hat{w}_1 \hat{w}_2 \cdots \hat{w}_m$. The optimal alignment and the associated word error rate $R[n, m]$ for correct word string $w_1 w_2 \cdots w_n$ and the recognized word string $\hat{w}_1 \hat{w}_2 \cdots \hat{w}_m$ are obtained via the dynamic programming algorithm illustrated in Algorithm 9.1. The accumulated cost function $R[i, j]$ progresses from $R[1, 1]$ to $R[n, m]$ corresponding to the minimum distance from (1, 1) to (n, m). We store the back pointer information $B[i, j]$ as we move along. When we reach the final grid (n, m), we back trace along the optimal path to find out if there are substitutions, deletions, or insertions on the matched path, as stored in $B[i, j]$.

[2] Even for isolated speech recognition, you may still have the insertion error, since the word boundary needs to be detected in most applications. It is possible that one isolated utterance is recognized as two words.

ALGORITHM 9.1: ALGORITHM TO MEASURE THE WORD ERROR RATE

Step 1: *Initialization* $R[0,0] = 0$ $R[i,j] = \infty$ if $(i < 0)$ or $(j < 0)$ $B[0,0] = 0$

Step 2: *Iteration*

for $i = 1,\ldots,n$ {

for $j = 1,\ldots,m$ {

$$R[i,j] = \min \begin{bmatrix} R[i-1,j]+1 \text{ (deletion)} \\ R[i-1,j-1] \text{ (match)} \\ R[i-1,j-1]+1 \text{ (substitution)} \\ R[i,j-1]+1 \text{ (insertion)} \end{bmatrix}$$

$$B[i,j] = \begin{cases} 1 & \text{if deletion} \\ 2 & \text{if insertion} \\ 3 & \text{if match} \\ 4 & \text{if substitution} \end{cases} \quad \} \}$$

Step 3: *Backtracking and termination*

$$\text{word error rate} = 100\% \times \frac{R(n,m)}{n}$$

$$\text{optimal backward path} = (s_1, s_2, \ldots, 0)$$

$$\text{where } s_1 = B[n,m],\ s_t = \begin{bmatrix} B[i-1,j] \text{ if } s_{t-1} = 1 \\ B[i,j-1] \text{ if } s_{t-1} = 2 \\ B[i-1,j-1] \text{ if } s_{t-1} = 3 \text{ or } 4 \end{bmatrix} \text{ for } t = 2,\ldots \text{ until } s_t = 0$$

For applications involved with rejection, such as word confidence measures as discussed in Section 9.7, you need to measure both false rejection rate and false acceptance rate. In speaker or command verification, the false acceptance of a valid user/command is also referred to as Type I error, as opposed to the false rejection of a valid user/command (Type II) [17]. A higher false rejection rate generally leads to a lower false acceptance rate. A plot of the false rejection rate versus the false acceptance rate, widely used in communication theory, is called the *receiver operating characteristic* (ROC) curve.

9.3. SIGNAL PROCESSING—EXTRACTING FEATURES

The role of a signal processing module, as illustrated in Figure 1.2, is to reduce the data rate, to remove noises, and to extract salient features that are useful for subsequent acoustic matching. Using as building blocks the topics we discussed in earlier chapters, we briefly illustrate here what is important in modeling speech to deal with variations we must address. More advanced environment normalization techniques are discussed in Chapter 10.

9.3.1. Signal Acquisition

Today's computers can handle most of the necessary speech signal acquisition tasks in software. For example, most PC sound cards have direct memory access, and the speech can be digitized to memory without burdening the CPU with input/output interrupts. The operating system can correctly handle most of the necessary AD/DA functions in real time.

To perform speech recognition, a number of components—such as digitizing speech, feature extraction and transformation, acoustic matching, and language model-based search—can be pipelined time-synchronously from left to right. Most operating systems can supply mechanisms for organizing pipelined programs in a multitasking environment. Buffers must be appropriately allocated so that you can ensure time-synchronous processing of each component. Large buffers are generally required on slow machines because of potential delays in processing an individual component. The right buffer size can be easily determined by experimentally tuning the system with different machine load situations to find a balance between resource use and relative delay.

For speech signal acquisition, the needed buffer typically ranges from 4 to 64 kB with 16-kHz sampling rate and 16-bit A/D precision. In practice, 16-kHz sampling rate is sufficient for the speech bandwidth (8 kHz). Reduced bandwidth, such as telephone channel, generally increases speech recognition error rate. Table 9.1 shows some empirical relative word recognition error increase using a number of different sampling rates. If we take the 8-kHz sampling as our baseline, we can reduce the word recognition error with a comparable recognizer by about 10% if we increase the sampling rate to 11 kHz. If we further increase the sampling rate to 16 kHz, the word recognition error rate can be further reduced by an additional 10%. Further increasing the sampling rate to 22 kHz does not have any additional impact on the word recognition errors, because most of the salient speech features are within an 8-kHz bandwidth.

Table 9.1 Relative error rate reduction with different sampling rates. The reduction is relative to that of the preceding row.

Sampling Rate	Relative Error-Rate Reduction
8 kHz	Baseline
11 kHz	+10%
16 kHz	+10%
22 kHz	+0%

9.3.2. End-Point Detection

To activate speech signal capture, you can use a number of modes including either *push to talk* or *continuously listening*. The push-to-talk mode uses a special push event to activate or

deactivate speech capture, which is immune to the potential background noise and can eliminate unnecessary use of processing resources to detect speech events. This mode sometimes also requires you to *push and hold while talking*. You push to indicate speech's beginning and then release to indicate the end of speech capture. The disadvantage is the necessity to activate the application each time the person speaks.

The continuously listening model listens all the time and automatically detects whether there is a speech signal or not. It needs a so-called *speech end-point detector*, which is typically based on an extremely efficient two-class pattern classifier. Such a classifier is used to filter out obvious silence, but the ultimate decision on the utterance boundary is left to the speech recognizer. In comparison to the push-to-talk mode, the continuously listening mode requires more processing resources, also with potential classification errors.

The endpoint detector is often based on an energy threshold that is a function of time. The logarithm of the energy threshold can be dynamically generated based on energy profiles across a certain period of time. Constraints on word duration can also be imposed to better classify a sequence of frames so that extremely short spikes can be eliminated.

It is not critical for the automatic end-point detector to offer exact end-point accuracy. The key feature required of it is a low rejection rate (i.e., the automatic end-point detector should not interpret speech segments as silence/noise segments). Any false rejection leads to an error in the speech recognizer. On the other hand, a possible false acceptance (i.e., the automatic end-point detector interprets noise segments as speech segments) may be rescued by the speech recognizer later if the recognizer has appropriate noise models, such as specific models for clicks, lip smacks, and background noise.

Explicit end-point detectors work reasonably well with recordings exhibiting a signal-to-noise ratio of 30 dB or greater, but they fail considerably on noisier speech. As discussed, speech recognizers can be used to determine the end points by aligning the vocabulary words preceded and followed by a silence/noise model. This scheme is generally much more reliable than any threshold-based explicit end-point detection, because recognition can jointly detect both the end points and words or other explicit noise classes, but requires more computational resources. A compromise is to use a simple adaptive two-class (speech vs. silence/noise) classifier to locate speech activities (with enough buffers at both ends) and notify the speech recognizer for subsequent processing. For the two-class classifier, we can use both the log-energy and delta log-energy as the feature. Two Gaussian density functions, $\{\Phi_1, \Phi_2\} = \Phi$, can be used to model the background stationary noise and speech, respectively. The parameters of the Gaussian density can be estimated using the labeled speech and noise data or estimated in an unsupervised manner.

When enough frames are classified as speech segments by the efficient two-class classifier, the speech recognizer is notified to start recognizing the signal. As shown in Figure 9.2, we should include enough frames before the beginning frame, t_b, for the speech recognizer to minimize the possible detection error. In the same manner, when enough noise/silence frames are detected at t_e, we should keep providing the speech recognizer with enough frames for processing before declaring that the end of the utterance has been reached.



Figure 9.2 End-point detection boundary t_b and t_e may need extra buffering for subsequent speech recognition.

Since there are only two classes, these parameters can be dynamically adapted using the EM algorithm during runtime. As discussed in Chapter 4, the EM algorithm can iteratively estimate the Gaussian parameters without having a precise segmentation between speech and noise segments. This is very important, because we need to keep the parameters dynamic for robust end-point detection in constantly changing environments.

To track the varying background noises, we use an exponential window to give weight to the most recent signal:

$$w_k = \exp(-\alpha k) \tag{9.4}$$

where α is a constant that controls the adaptation rate, and k is the index of the time. In fact, you could use different rates for noise and speech when you use the EM algorithm to estimate the two-class Gaussian parameters. It is advantageous to use a smaller time constant for noise than for speech. With such a weighting window, the means of the Gaussian density, as discussed in Chapter 4, can be rewritten as:

$$\hat{\boldsymbol{\mu}}_k = \frac{\sum_{i=-\infty}^{t} w_i \dfrac{c_k P(\mathbf{x}_i \mid \Phi_k)\mathbf{x}_i}{\sum_{k=1}^{2} P(\mathbf{x}_i \mid \Phi_k)}}{\sum_{i=-\infty}^{t} w_i \dfrac{c_k P(\mathbf{x}_i \mid \Phi_k)}{\sum_{k=1}^{2} P(\mathbf{x}_i \mid \Phi_k)}}, k \in \{0,1\} \tag{9.5}$$

9.3.3. MFCC and Its Dynamic Features

The extraction of reliable features is one of the most important issues in speech recognition. There are a large number of features we can use. However, as discussed in Chapter 4, the

curse-of-dimensionality problem reminds us that the amount of training data is always limited. Therefore, incorporation of additional features may not lead to any measurable error reduction. This does not necessarily mean that the additional features are poor ones, but rather that we may have insufficient data to reliably model those features.

The first feature we use is the speech waveform itself. In general, time-domain features are much less accurate than frequency-domain features such as the mel-frequency cepstral coefficients (MFCC) discussed in Chapter 6 [23]. This is because many features such as formants, useful in discriminating vowels, are better characterized in the frequency domain with a low-dimension feature vector.

As discussed in Chapter 2, temporal changes in the spectra play an important role in human perception. One way to capture this information is to use *delta coefficients* that measure the change in coefficients over time. Temporal information is particularly complementary to HMMs, since HMMs assume each frame is independent of the past, and these dynamic features broaden the scope of a frame. It is also easy to incorporate new features by augmenting the static feature.

When 16-kHz sampling rate is used, a typical state-of-the-art speech system can be build based on the following features.

- 13th-order MFCC $\mathbf{c}_k$
- 13th-order 40-msec 1st-order delta MFCC computed from $\Delta\mathbf{c}_k = \mathbf{c}_{k+2} - \mathbf{c}_{k-2}$
- 13th-order 2nd-order delta MFCC computed from $\Delta\Delta\mathbf{c}_k = \Delta\mathbf{c}_{k+1} - \Delta\mathbf{c}_{k-1}$

A short-time analysis Hamming window of 25 ms is typically used to compute the MFCC $\mathbf{c}_k$. The window shift is typically 10 ms. Please note that $c_k[0]$ is included in the feature vector, which has a role similar to that of the log power. The feature vector used for speech recognition is typically a combination of these features

$$\mathbf{x}_k = \begin{pmatrix} \mathbf{c}_k \\ \Delta\mathbf{c}_k \\ \Delta\Delta\mathbf{c}_k \end{pmatrix} \tag{9.6}$$

The relative error reduction with a typical speech recognition system is illustrated in Table 9.2. As you can see from the table, the 13th-order MFCC outperforms 13th-order LPC cepstrum coefficients, which indicates that perceptually motivated mel-scale representation indeed helps recognition. In a similar manner, perceptually based LPC features such as PLP can achieve similar improvement. The MFCC order has also been studied experimentally for speech recognition. The higher-order MFCC does not further reduce the error rate in comparison with the 13th-order MFCC, which indicates that the first 13 coefficients already contain most salient information needed for speech recognition. In addition to mel-scale representation, another perceptually motivated feature such as the first- and second-order delta features can significantly reduce the word recognition error, while the higher-order delta features provide no further information.

Feature extraction in these experiments is typically optimized together with the classifier, since there are a number of modeling assumptions, such as the diagonal covariance in the Gaussian density function, that are closely related to what features to use. It is possible that these relative error reductions would vary if a different speech recognizer were used.

Table 9.2 Relative error reduction with different features. The reduction is relative to that of the preceding row.

Feature Set	Relative Error Reduction
13th-order LPC cepstrum coefficients	Baseline
13th-order MFCC	+10%
16th-order MFCC	+0%
+1st- and 2nd-order dynamic features	+20%
+3rd-order dynamic features	+0%

9.3.4. Feature Transformation

Before you use feature vectors such as MFCC for recognition, you can preprocess or transform them into a new space that alleviates environment noise, channel distortion, and speaker variations. You can also transform the features that are most effective for preserving class separability so that you can further reduce the recognition error rate. Since we devote Chapter 10 completely to environment and channel normalization, we briefly discuss here how we can transform the feature vectors to improve class separability.

To further reduce the dimension of the feature vector, you can use a number of dimension reduction techniques to map the feature vector into more effective representations. If the mapping is linear, the mapping function is well defined and you can find the coefficients of the linear function so as to optimize your objective functions. For example, when you combine the first- and second-order dynamic features with the static MFCC vector, you can use *principal-component analysis* (PCA) (also known as *Karhunen-Loeve* transform) [32] to map the combined feature vector into a smaller dimensional vector. The optimum basis vectors of the principal-component analysis are the eigenvectors of the covariance matrix of a given distribution. In practice, you can compute the eigenvectors of the autocorrelation matrix as the basis vectors. The effectiveness of the transformed vector, in terms of representing the original feature vector, is determined by the corresponding eigenvalue of each value in the vector. You can discard the feature with the smallest eigenvalue, since the mean-square error between the transformed vector and the original vector is determined by the eigenvalue of each feature in the vector. In addition, the transformed feature vector is uncorrelated. This is particularly suitable for the Gaussian probability density function with a diagonal covariance matrix.

The recognition error is the best criterion for deciding what feature sets to use. However, it is hard to obtain such an estimate to evaluate feature sets systematically. A simpler

approach is to use within-class and between-class scatter matrices to formulate criteria of class separability, which is also called *Linear Discriminant Analysis* (LDA) transformation. We can compute the within-class scatter matrix as:

$$S_w = \sum_{\mathbf{x}\in\omega_i} P(\omega_i)E\{(\mathbf{x}-\boldsymbol{\mu}_i)(\mathbf{x}-\boldsymbol{\mu}_i)' \mid \omega_i\} = \sum_{\mathbf{x}\in\omega_i} P(\omega_i)\Sigma_i \tag{9.7}$$

where the sum is for all the data $\mathbf{x}$ within the class ω_i. This is the scatter of samples around their respective class mean. On the other hand, the between-class scatter matrix is the scatter of the expected vectors around the mixture mean:

$$S_B = \sum_{\boldsymbol{\mu}_i\in\omega_i} P(\omega_i)(\boldsymbol{\mu}_i - \mathbf{m}_0)(\boldsymbol{\mu}_i - \mathbf{m}_0)' \tag{9.8}$$

where $\mathbf{m}_0$ represents the expected mean vector of the mixture distribution:

$$\mathbf{m}_0 = E\{\mathbf{x}\} = \sum_{\omega_i} P(\omega_i)\mathbf{m}_i \tag{9.9}$$

To formulate criteria to transform feature vector $\mathbf{x}$, we need to derive the linear transformation matrix $\mathbf{A}$. One of the measures can be the trace of $S_w^{-1}S_B$:

$$J = tr(S_w^{-1}S_B) \tag{9.10}$$

The trace is the sum of the eigenvalues of $S_w^{-1}S_B$ and hence the sum of the variances in the principal directions. The number is larger when the between-class scatter is large or the within-class scatter is small. You can derive the transformation matrix based on the eigenvectors of $S_w^{-1}S_B$. In a manner similar to PCA, you can reduce the dimension of the original input feature vector by discarding the smallest eigenvalues [16, 54].

Researchers have used the LDA method to measure the effectiveness of several feature vectors for speaker normalization [41]. Other feature processing techniques designed for speaker normalization include neural-network-based speaker mapping [51], frequency warping for vocal tract normalization (VTN) via mel-frequency scaling [67, 100], and bilinear transformation [2].

To reduce interspeaker variability by a speaker-specific frequency warping, you can simply shift the center frequencies of the mel-spaced filter bank. Let $k\Delta f_{mel}$, $k = 1, \ldots, K$, denote the center frequencies in mel-scale. Then the center frequencies in hertz for a warping factor of α are computed by Eq. (9.11) before the cosine transformation of the MFCC feature vector.

$$f_{Hz}^{\alpha}(k\Delta f_{mel}) = 700(10^{k\Delta f_{mel}/2595} - 1)/\alpha \tag{9.11}$$

The warping factor is estimated for each speaker by computing the likelihood of the training data for feature sets obtained with different warping factors using the HMM. The relative error reduction based on the feature transformation method has been limited, typically under 10%.

9.4. PHONETIC MODELING—SELECTING APPROPRIATE UNITS

As discussed in Chapter 2, the phonetic system is related to a particular language. We focus our discussion on language-independent technologies but use English in our examples to illustrate how we can use the language-independent technologies to model the salient phonetic information in the language. For general-purpose large-vocabulary speech recognition, it is difficult to build whole-word models because:

- Every new task contains novel words without any available training data, such as proper nouns and newly invented jargons.
- There are simply too many words, and these different words may have different acoustic realizations, as illustrated in Chapter 2. It is unlikely that we have sufficient repetitions of these words to build context-dependent word models.

How to select the most basic units to represent salient acoustic and phonetic information for the language is an important issue in designing a workable system. At a high level, there are a number of issues we must consider in choosing appropriate modeling units.

- The unit should be *accurate*, to represent the acoustic realization that appears in different contexts.
- The unit should be *trainable*. We should have enough data to estimate the parameters of the unit. Although words are accurate and representative, they are the least trainable choice in building a working system, since it is nearly impossible to get several hundred repetitions for all the words, unless we are using a speech recognizer that is domain specific, such as a recognizer designed for digits only.
- The unit should be *generalizable*, so that any new word can be derived from a predefined unit inventory for task-independent speech recognition. If we have a fixed set of word models, there is no obvious way for us to derive the new word model.

A practical challenge is how to balance these selection criteria for speech recognition. In this section we compare a number of units and point out their strengths and weaknesses in practical applications.

9.4.1. Comparison of Different Units

What is a unit of language? In English, words are typically considered as a principal carrier of meaning and are seen as the smallest unit that is capable of independent use. As the most natural unit of speech. whole-word models have been widely used for many speech recognition systems. A distinctive advantage of using word models is that we can capture the phonetic coarticulation inherent within these words. When the vocabulary is small, we can create word models that are context dependent.

For example, if the vocabulary is English digits, we can have different word models for the word *one* to represent the word in different contexts. Thus each word model is dependent on its left and right context. If someone says *three one two*, the recognizer uses the word model *one* that specifically depends on the left context *three* and right context *two*. Since the vocabulary is small (10), we need to have only 10*10*10=1000 word models, which is achievable when you collect enough training data. With context-dependent, or even context-independent, word models, a wide range of phonological variations can be automatically accommodated. When these word models are adequately trained, they usually yield the best recognition performance in comparison to other modeling units. Therefore, for small vocabulary recognition, whole-word models are widely used, since they are both *accurate* and *trainable*, and there is no need to be *generalizable*.

While words are suitable units for small-vocabulary speech recognition, they are not a practical choice for large-vocabulary continuous speech recognition. First, each word has to be treated individually, and data cannot be shared across word models; this implies a prohibitively large amount of training data and storage. Second, for some task configurations, the recognition vocabulary may consist of words that never appeared in the training data. As a result, some form of word-model composition technique is required to generate word models. Third, it is very expensive to model interword coarticulation effects or adapt a word-based system for a new speaker, a new channel, or new context usage.

To summarize, word models are *accurate* if enough data are available. Thus, they are *trainable* only for small tasks. They are typically not *generalizable*.

Alternatively, there are only about 50 phones in English, and they can be sufficiently trained with just a few hundred sentences. Unlike word models, phonetic models provide no training problem. Moreover, they are also vocabulary independent by nature and can be trained on one task and tested on another. Thus, phones are more *trainable* and *generalizable*. However, the phonetic model is inadequate because it assumes that a phoneme in any context is identical. Although we may try to say each word as a concatenated sequence of independent phonemes, these phonemes are not produced independently, because our articulators cannot move instantaneously from one position to another. Thus, the realization of a phoneme is strongly affected by its immediately neighboring phonemes. For example, if context-independent phonetic models are used, the same model for *t* must capture various events, such as flapping, unreleased stops, and realizations in */t s/* and */t r/*. Then, if */t s/* is the only context in which *t* occurs in the training, while */t r/* is the only context in the testing,

the model used is highly inappropriate. While word models are not generalizable, phonetic models overgeneralize and, thus, lead to less accurate models.

A compromise between the word and phonetic model is to use larger units such as *syllables*. These units encompass phone clusters that contain the most variable contextual effects. However, while the central portions of these units have no contextual dependencies, the beginning and ending portions are still susceptible to some contextual effects. There are only about 1200 tone-dependent syllables in Chinese and approximately 50 syllables in Japanese, which makes syllable a suitable unit for these languages. Unfortunately, the large number of syllables (over 30,000) in English presents a challenge in terms of trainability.

9.4.2. Context Dependency

If we make units context dependent, we can significantly improve the recognition accuracy, provided there are enough training data to estimate these context-dependent parameters. Context-dependent phonemes have been widely used for large-vocabulary speech recognition, thanks to its significantly improved accuracy and trainability. A context usually refers to the immediate left and/or right neighboring phones.

A *triphone* model is a phonetic model that takes into consideration both the left and the right neighboring phones. If two phones have the same identity but different left or right contexts, they are considered different triphones. We call different realizations of a phoneme *allophones*. Triphones are an example of allophones.

The left and right contexts used in triphones, while important, are only two of many important contributing factors that affect the realization of a phone. Triphone models are powerful because they capture the most important coarticulatory effects. They are generally much more consistent than context-independent phone models. However, as context-dependent models generally have increased parameters, trainability becomes a challenging issue. We need to balance trainability and accuracy with a number of parameter-sharing techniques.

Modeling interword context-dependent phones is complicated. For example, in the word *speech*, pronounced */s p iy ch/*, both left and right contexts for */p/* and */iy/* are known, while the left context for */s/* and the right context for */ch/* are dependent on the preceding and following words in actual sentences. The juncture effect on word boundaries is one of the most serious coarticulation phenomena in continuous speech, especially with short function words like *the* or *a*. Even with the same left and right context identities, there may be significantly different realizations for a phone at different word positions (the *beginning, middle,* or *end* of a word). For example, the phone */t/* in *that rock* is almost extinct, while the phone */t/* in the middle of *theatrical* sounds like */ch/*. This implies that different word positions have an effect on the realization of the same triphone.

In addition to the context, stress also plays an important role in the realization of a particular phone. Stressed vowels tend to have longer duration, higher pitch, and more intensity, while unstressed vowels appear to move toward a neutral, central *schwa*-like phoneme. Agreement about the phonetic identity of a syllable has been reported to be greater in

stressed syllables for both humans and automatic phone recognizers. In English, *word-level* stress is referred to as *free stress*, because the stressed syllable can take on any position within a word, in contrast to *bound stress* found in languages such as French and Polish, where the position of the stressed syllable is fixed within a word. Therefore, stress in English can be used as a constraint for lexical access. In fact, stress can be used as a unique feature to distinguish a set of word pairs, such as ***import*** vs. ***import***, and ***export*** vs. ***export***. For example, the phone set used for Whisper, such as */er/-/axr/* and */ah/-/ix/-/ax/*, describes these stressed and unstressed vowels. One example illustrating how stress can significantly affect the realization of phone is demonstrated in Figure 9.3, where phone */t/* in word *Italy* vs. *Italian* is pronounced differently in American English due the location of the stress, albeit the triphone context is identical for both words.

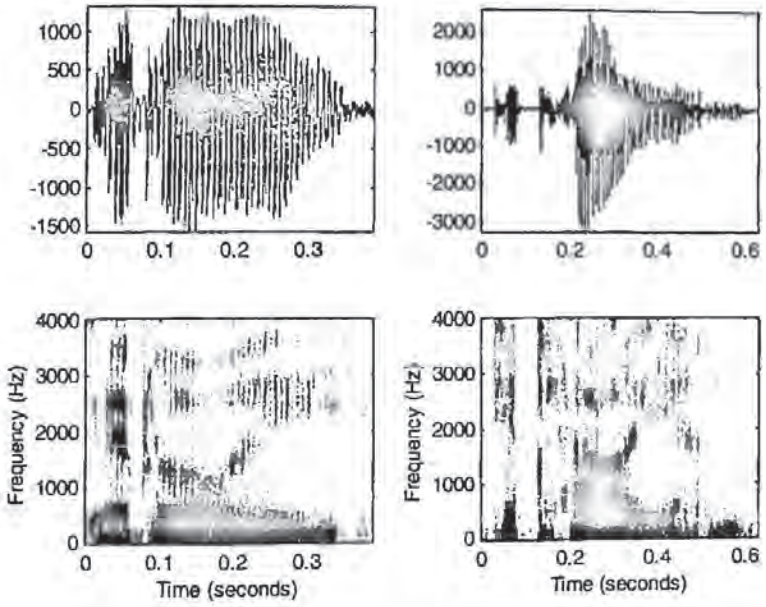


Figure 9.3 The importance of stress is illustrated in *Italy* vs. *Italian* for phone */t/*. The realizations are quite different, even though they share the same left and right context.

Sentence-level stress, on the other hand, represents the overall stress pattern of continuous speech. While sentence-level stress does not change the meaning of any particular lexicon item, it usually increases the relative prominence of portions of the utterance for the purpose of contrast or emphasis. *Contrastive stress* is normally used to coordinate constructions such as *there are **import** records and there are **domestic** ones*, as well as for the purpose of correction, as in *I said **import**, not **export**. Emphatic* stress is commonly used to distinguish a sentence from its negation, e.g., *I **did** have dinner*. Sentence-level stress is very hard to model without incorporating high-level semantic and pragmatic knowledge. In most state-of- the-art speech recognition systems, only word-level stress is used for creating allophones.

9.4.3. Clustered Acoustic-Phonetic Units

Triphone modeling assumes that every triphone context is different. Actually, many phones have similar effects on the neighboring phones. The position of our articulators has an important effect on how we pronounce neighboring vowels. For example, /b/ and /p/ are both labial stops and have similar effects on the following vowel, while /r/ and /w/ are both liquids and have similar effects on the following vowel. Contrary to what we illustrate in Figure 9.1, Figure 9.4 illustrates this phenomenon. It is desirable to find instances of similar contexts and merge them. This would lead to a much more manageable number of models that can be better trained.

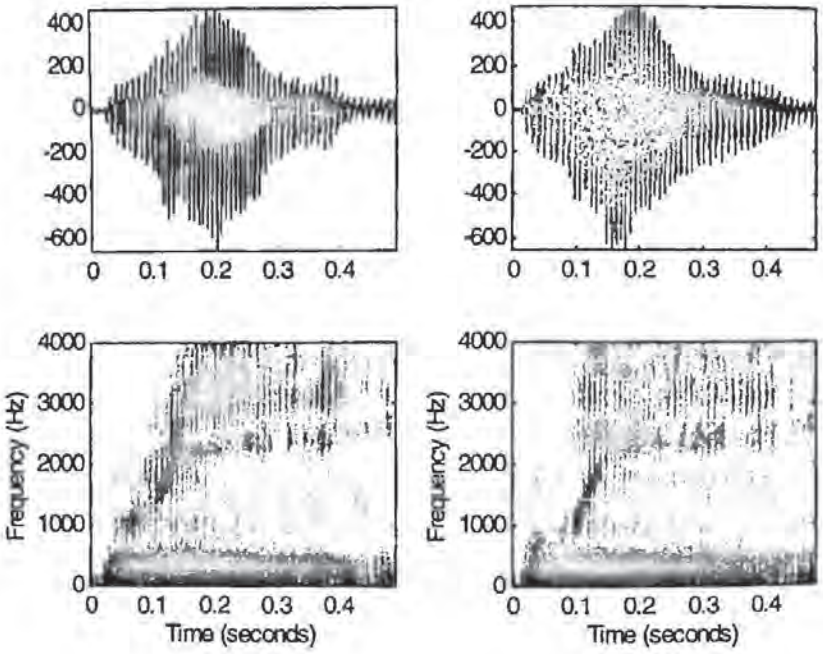


Figure 9.4 The spectrograms for the phoneme /iy/ with two different *left-contexts* are illustrated. Note that /r/ and /w/ have similar effects on /iy/. This illustrates that different left-contexts may have similar effects on a phone.

The trainability and accuracy balance between phonetic and word models can be generalized further to model subphonetic events. In fact, both phonetic and subphonetic units have the same benefits, as they share parameters at the unit level. This is the key benefit in comparison to the word units. Papers by [11, 45, 57, 66, 111] provide examples of the application of this concept to cluster hidden Markov models. For subphonetic modeling, we can treat the state in phonetic HMMs as the basic subphonetic unit. Hwang and Huang further generalized clustering to the state-dependent output distributions across different phonetic models [57]. Each cluster thus represents a set of similar Markov states and is called a

senone [56]. A subword model is thus composed of a sequence of senones after the clustering is finished. The optimal number of senones for a system is mainly determined by the available training corpus and can be tuned on a development set.

Each allophone model is an HMM made of states, transitions, and probability distributions. To improve the reliability of the statistical parameters of these models, some distributions can be tied. For example, distributions for the central portion of an allophone may be tied together to reflect the fact that they represent the stable (context-independent) physical realization of the central part of the phoneme, uttered with a stationary configuration of the vocal tract. Clustering at the granularity of the state rather than the entire model can keep the dissimilar states of two models apart while the other corresponding states are merged, thus leading to better parameter sharing.

Figure 9.5 illustrates how state-based clustering can lead to improved representations. These two HMMs come from the same phone class with a different right context, leading to very different output distributions in the last state. As the left contexts are identical, the first and second output distributions are almost identical. If we measure the overall model similarity based on the accumulative overall output distribution similarities of all states, these two models may be clustered, leading to a very inaccurate distribution for the last state. Instead, we cluster the first two output distributions while leaving the last one intact.

There are two key issues in creating trainable context-dependent phonetic or subphonetic units:

- We need to enable better parameter sharing and smoothing. As Figure 9.4 illustrates, many phones have similar effects on neighboring phones. If the acoustic realization is indeed identical, we tie them together to improve trainability and efficiency.
- Since the number of triphones in English is very large (over 100,000), there are many new or unseen triphones that are in the test set but not in the training set. It is important to map these unseen triphones into appropriately trained triphones.

As discussed in Chapter 4, a decision tree is a binary tree to classify target objects by asking binary questions in a hierarchical manner. Modeling unseen triphones is particularly important for *vocabulary independence*, since it is difficult to collect a training corpus which covers enough occurrences of every possible subword unit. We need to find models that are accurate, trainable, and especially generalizable. The senonic decision tree classifies Markov states of triphones represented in the training corpus by asking linguistic questions composed of conjunctions, disjunctions, and/or negations of a set of predetermined simple categorical linguistic questions. Examples of these simple categorical questions are: *Is the left-context phone a fricative? Is the right-context phone a front vowel?* The typical question set used in Whisper to generate the senone tree is shown in Table 9.3. So, for each node in the tree, we check whether its left or right phone belongs to one of the categories. As discussed in Chapter 4, we measure the corresponding entropy reduction or likelihood increase for each question and select the question that has the largest entropy decrease to split the node.

Thus, the tree can be automatically constructed by searching, for each node, the question that renders the maximum entropy decrease. Alternatively, complex questions can be formed for each node for improved splitting.

When we grow the tree, it needs to be pruned using cross-validation as discussed in Chapter 4. When the algorithm terminates, the leaf nodes of the tree represent the senones to be used. Figure 9.6 shows an example tree we built to classify the second state of all */k/* triphones seen in a training corpus. After the tree is built, it can be applied to the second state of *any* */k/* triphone, thanks to the generalizability of the binary tree and the general linguistic questions. Figure 9.6 indicates that the second state of the */k/* triphone in *welcome* is mapped to the second senone, no matter whether this triphone occurs in the training corpus or not.

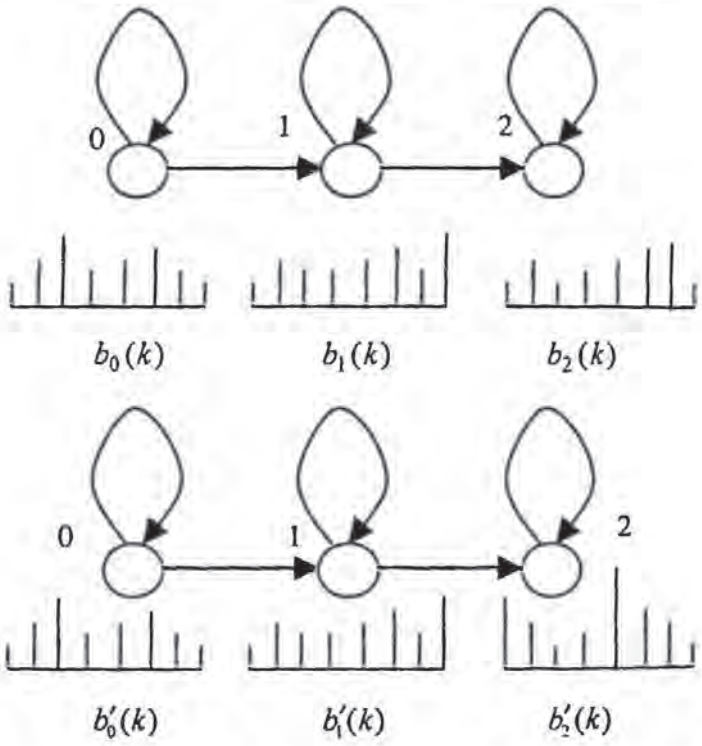


Figure 9.5 State-based vs. model-based clustering. These two models are very similar, as both the first and the second output distributions are almost identical. The key difference is the output distribution of the third state. If we measure the overall model similarity, which is often based on the accumulative output distribution similarities of all states, these two models may be clustered, leading to a very inaccurate distribution for the last state. If we cluster output distributions at state level, we can cluster the first two output distributions while leaving the last ones intact, leading to more accurate representations.

Table 9.3 Some example questions used in building senone trees.

Questions	Phones in Each Question Category
Aspseg	*hh*
Sil	*sil*
Alvstp	*d t*
Dental	*dh th*
Labstp	*b p*
Liquid	*l r*
Lw	*l w*
S/Sh	*s sh*
Sylbic	*er axr*
Velstp	*g k*
Affric	*ch jh*
Lqgl-B	*l r w*
Nasal	*m n ng*
Retro	*r er axr*
Schwa	*ax ix axr*
Velar	*ng g k*
Fric2	*th s sh f*
Fric3	*dh z zh v*
Lqgl	*l r w y*
S/Z/Sh/Zh	*s z sh zh*
Wglide	*uw aw ow w*
Labial	*w m b p v*
Palatl	*y ch jh sh zh*
Yglide	*iy ay ey oy y*
High	*ih ix iy uh uw y*
Lax	*eh ih ix uh ah ax*
Low	*ae aa ao aw ay oy*
Orstp2	*p t k*
Orstp3	*b d g*
Alvelr	*n d t s z*
Diph	*uw aw ay ey iy ow oy*
Fric1	*dh th s sh z zh v f*
Round	*uh ao uw ow oy w axr er*
Frnt-R	*ae eh ih ix iy ey ah ax y aw*
Tense	*iy ey ae uw ow aa ao ay oy aw*
Back-L	*uh ao uw ow aa er axr l r w aw*
Frnt-L	*ae eh ih ix iy ey ah ax y oy ay*
Back-R	*uh ao uw ow aa er axr oy l r w ay*
Orstp1	*b d g p t k ch jh*
Vowel	*ae eh ih ix iy uh ah ax aa ao uw aw ay ey ow oy er axr*
Son	*ae eh ih ix iy ey ah ax oy ay uh ao uw ow aa er axr aw l r w y*
Voiced	*ae eh ih ix iy uh ah ax aa ao uw aw ay ey ow oy l r w y er axr m n ng jh b d dh g v z zh*

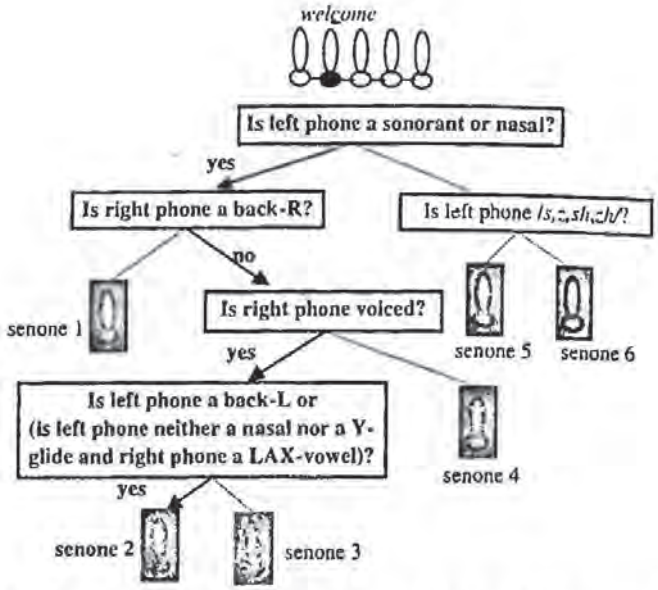


Figure 9.6 A decision tree for classifying the second state of K-triphone HMMs [48].

In practice, senone models significantly reduce the word recognition error rate in comparison with model-based clustered triphone models, as illustrated in Table 9.4. It is the senonic model's significant reduction of the overall system parameters that enables the continuous mixture HMMs to perform well for large-vocabulary speech recognition [56].

Table 9.4 Relative error reductions for different modeling units. The reduction is relative to that of the preceding row.

Units	Relative Error Reductions
Context-independent phone	Baseline
Context-dependent phone	+25%
Clustered triphone	+15%
Senone	+24%

9.4.4. Lexical Baseforms

When appropriate subword units are used, we must have the correct pronunciation for each word so that concatenation of subword units can accurately represent the word to be recognized. The dictionary represents the standard pronunciation used as a starting point for building a workable speech recognition system. We also need to provide alternative pronunciations for words such as *tomato* that may have very different pronunciations. For example, the COMLEX dictionary from LDC has about 90,000 baseforms that cover most words used in many years of *The Wall Street Journal*. The CMU Pronunciation Dictionary, which was optimized for continuous speech recognition, has about 100,000 baseforms.

In continuous speech recognition, we must also use phonologic rules to modify inter-word pronunciations or to have reduced sounds. Assimilation is a typical coarticulation phenomenon—a change in a segment to make it more like a neighboring segment. Typical examples include phrases such as *did you /d ih jh y ah/, set you /s eh ch er/, last year / l ae s ch iy r/, because you've /b iy k ah zh uw v/,* etc. Deletion is also common in continuous speech. For example, */t/* and */d/* are often deleted before a consonant. Thus, in conversational speech, you may find examples like *find him /f ay n ix m/, around this /ix r aw n ih s/,* and *Let me in /l eh m eh n/.*

Dictionaries often don't include proper names. For example, the 20,000 names included in the COMLEX dictionary are a small fraction of 1–2 million names in the USA. To deal with these new words, we often have to derive their pronunciation automatically. These new words have to be added on the fly, either by the user or through an interface from speech-aware applications. Unlike Spanish or Italian, rule-based letter-to-sound (LTS) conversion for English is often impractical, since so many words in English don't follow phonological rules. A trainable LTS converter is attractive, since its performance can be improved by constantly learning from examples so that it can generalize rules for the specific task. Trainable LTS converters can be based on neural networks, HMMs, or the CART described in Chapter 4. In practice, CART-based LTS has a very accurate performance [10, 61, 71, 89].

When CART is used, the basic YES-NO question for LTS conversion looks like: *Is the second right letter 'p'?* or: *Is the first left output phone /ay/?* The question for letters and phones can be on either the left or the right side. The range of question positions should be long enough to cover the most important phonological variations. Empirically, a 10-letter window (5 for left letter context and 5 for right letter context) and 3-phone window context is generally sufficient. A primitive set of questions can include all the singleton questions about each letter or phone identity. If we allow the node to have a complex question—that is, a combination of primitive questions—the depth of the tree can be greatly reduced and performance improved. For example, a complex question: *Is the second left letter 't' and the first left letter 'i' and the first right letter 'n'?* can capture *o* in the common suffix *tion* and convert it to the correct phone. Complex questions can also alleviate possible data-fragmentation problems caused by the greedy nature of the CART algorithm.

Categorical questions can be formed in both the letter and phone domains with our common linguistic knowledge. For example, the most often used set includes the letter or phone clusters for vowels, consonants, nasals, liquids, fricatives, and so on. In growing the decision tree, the context distance also plays a major role in the overall quality. It is very important to weight the entropy reduction according to the distance (either letter or phoneme) to avoid overgeneralization, which forces the tree to look more carefully at the *nearby* context than at the *far-away* context. Each leaf of the tree has a probability distribution for letter-to-phoneme mapping.

There are a number of ways to improve the effectiveness of the decision tree. First, pruning controls the tree's depth. For example, certain criteria have to be met for a node to be split. Typically splitting requires a minimum number of counts and a minimum entropy reduction. Second, the distribution at the leaves can be smoothed. For example, a leaf distri-

bution can be interpolated with the distributions of its ancestor nodes using deleted interpolation. Finally, we can partition the training data and build multiple trees with different prediction capabilities. These trees accommodate different phonological rules with different language origins.

When the decision tree is used to derive the phonetic pronunciation, the phonetic conversion error is about 8% for the *Wall Street Journal* newspaper text corpora [61]. These errors can be broadly classified into two categories. The first includes errors of proper nouns and foreign words. For example, *Pacino* can be mistakenly converted to */p ax s iy n ow/* instead of */p ax ch iy n ow/*. The second category includes generalization errors. For example, *shier* may be converted to */sh ih r/* instead of the correct pronunciation */sh ay r/* if the word *cashier /k ae sh ih r/* appears in the training data. The top three phone confusion pairs are */ix/ax/*, */dx/t/*, and */ae/ax/*. The most confusing pair is */ix/ax/*. This is not surprising, because */ix/ax/* is among the most inconsistent transcriptions in most of the published dictionaries. There is no consensus for */ix/ax/* transcription among phoneticians.

Although automatic LTS conversion has a reasonable accuracy, it is hardly practical if you don't use an exception dictionary. This is especially true for proper nouns. In practice, you can often ask the person who knows how to pronounce the word to either speak or write down the correct phonetic pronunciation, updating the exception dictionary if the correct one disagrees with what the LTS generates. When acoustic examples are available, you can use the decision tree to generate multiple results and use these results as a language model to perform phone recognition on the acoustic examples. The best overall acoustic and LTS probability can be used as the most likely candidate in the exception dictionary. Since there may be many ways to pronounce a word, you can keep multiple pronunciations in the dictionary with a probability for each possible one. If the pronunciation probability is inaccurate, an increase in multiple pronunciations essentially increases the size and confusion of the vocabulary, leading to increased speech recognition error rate.

Even if you have accurate phonetic baseforms, pronunciations in spontaneous speech differ significantly from the standard baseform. Analysis of manual phonetic transcription of conversational speech reveals a large number (> 20%) of cases of genuine ambiguity: instances where human labelers disagree on the identity of the surface form [95]. For example, the word *because* has more than 15 different pronunciation variations, such as */b iy k ah z/, /b ix k ah z/, /k ah z/, /k ax z/, /b ix k ax z/, /b ax k ah z/, /b ih k ah z/, /k s/, /k ix z/, /k ih z/, /b iy k ah s/, /b iy k ah/, /b iy k ah zh/, /ax z/*, etc., in the context of conversational speech [39]. To characterize the acoustic evidence in the context of this ambiguity, you can partly resolve the ambiguity by deriving a suitable phonetic baseform from speech data [29, 95, 97]. This is because the widespread variation can be due either to a lexical fact (such as that the word *because* can be *'cause* in informal speech) or to the dialect differences. African American vernacular English has many vowels different from general American English.

To incorporate widespread pronunciations, we can use a probabilistic finite state machine to model each word's pronunciation variations, as shown in Figure 9.7. The probability with each arc indicates how likely that path is to be taken, with all the arcs that leave a node summing to 1. As with HMMs, these weights can be estimated from a real corpus for

improved speech recognition [20, 85, 102, 103, 110]. In practice, the relative error reduction of using probabilistic finite state machines is very modest (5–10%).

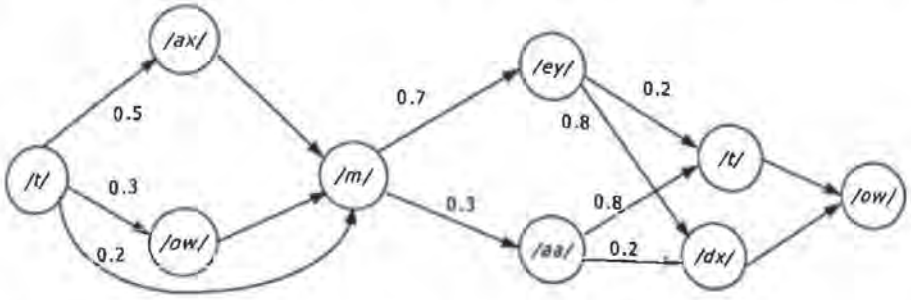


Figure 9.7 A possible pronunciation network for word *tomato*. The vowel */ey/* is more likely to flap, thereby having a higher transition probability into */dx/*.

9.5. ACOUSTIC MODELING—SCORING ACOUSTIC FEATURES

After feature extraction, we have a sequence of feature vectors, **X**, such as the MFCC vector, as our input data. We need to estimate the probability of these acoustic features, given the word or phonetic model, **W**, so that we can recognize the input data for the correct word. This probability is referred to as acoustic probability, $P(\mathbf{X} \mid \mathbf{W})$. In this section we focus our discussion on the HMM. As discussed in Chapter 8, it is the most successful method for acoustic modeling. Other emerging techniques are discussed in Section 9.8.

9.5.1. Choice of HMM Output Distributions

As discussed in Chapter 8, you can use discrete, continuous, or semicontinuous HMMs. When the amount of training data is sufficient, parameter tying becomes unnecessary. A continuous model with a large number of mixtures offers the best recognition accuracy, although its computational complexity also increases linearly with the number of mixtures. On the other hand, the discrete model is computationally efficient, but has the worst performance among the three models. The semicontinuous model provides a viable alternative between system robustness and trainability.

When either the discrete or the semicontinuous HMM is employed, it is helpful to use multiple codebooks for a number of features for significantly improved performance. Each codebook then represents a set of different speech parameters. One way to combine these multiple output observations is to assume that they are independent, computing the output probability as the product of the output probabilities of each codebook. For example, the semicontinuous HMM output probability of multiple codebooks can be computed as the product of each codebook:

$$b_i(\mathbf{x}) = \prod_m \sum_{k=1}^{L^m} f^m(\mathbf{x}^m \mid o_k^m) b_i^m(o_k^m) \tag{9.12}$$

where superscript m denotes the codebook-m related parameters. Each codebook consists of L^m-mixture continuous density functions.

Following our discussion in Chapter 8, the re-estimation algorithm for the multiple-codebook-based HMM could be extended. Since multiplication of the output probability density of each codebook leads to several independent terms in the Q-function, for codebook m, $\zeta_t(j,k^m)$ can be modified as follows:

$$\zeta_t(j,k^m) = \frac{\sum_i \alpha_{t-1}(i) a_{ij} b_j^m(k^m) f^m(\mathbf{x}_t \mid v_k^m) \prod_{n \neq m} \sum_k b_j^n(k^n) f^n(\mathbf{x}_t \mid v_k^n) \beta_t(j)}{\sum_k \alpha_T^m(k)} \tag{9.13}$$

Other intermediate probabilities can also be computed in a manner similar to what we discussed in Chapter 8.

Multiple codebooks can dramatically increase the representation power of the VQ codebook and can substantially improve speech recognition accuracy. You can typically build a codebook for $\mathbf{c}_k$, $\Delta\mathbf{c}_k$, and $\Delta\Delta\mathbf{c}_k$, respectively. As energy has a very different dynamic range, you can further improve the performance by building a separate codebook for $c_k[0]$, $\Delta c_k[0]$, and $\Delta\Delta c_k[0]$. In comparison to building a single codebook for $\mathbf{x}_k$ as illustrated in Eq. (9.6), the multiple-codebook system can reduce the error rate by more than 10%.

In practice, the most important parameter for the output probability distribution is the number of mixtures or the size of the codebooks. When there are sufficient training data, relative error reductions with respect to the discrete HMM are those shown in Figure 9.8.

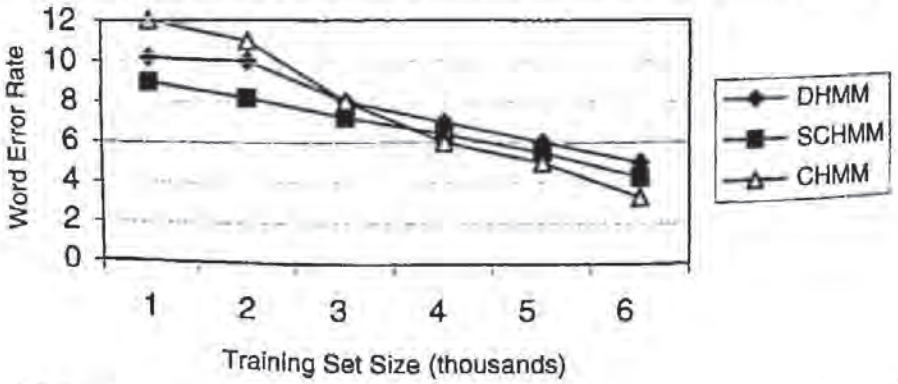


Figure 9.8 Continuous speaker-independent word recognition error rates of the discrete HMM (DHMM), SCHMM, and the continuous HMM (CHMM) with respect to the training set sizes (thousands of training sentences). Both the DHMM and SCHMM have multiple codebooks. The CHMM has 20 mixture diagonal Gaussian density functions.

As you can see from Figure 9.8, the SCHMM offers improved accuracy in comparison with the discrete HMM or the continuous HMM when the amount of training data is limited. When we increase the training data size, the continuous mixture density HMM starts to outperform both the discrete and the semicontinuous HMM, since the need to share model parameters becomes less critical.

Performance is also a function of the number of mixtures. With a small number of mixtures, the continuous HMM lacks the modeling power and it actually performs worse than the discrete HMM across the board. Only after we dramatically increase the number of mixtures does the continuous HMM start to offer improved recognition accuracy. The SCHMM can typically reduce the discrete HMM error rate by 10–15% across the board. The continuous HMM with 20 diagonal Gaussian density functions performed worse than either the discrete or the SCHMM when the size of training data was small. It outperformed either the discrete HMM or the SCHMM when sufficient amounts of training data became available. When the amount of training data is sufficiently large, it can reduce the error rate of the semicontinuous HMM by 15–20%.

9.5.2. Isolated vs. Continuous Speech Training

If we build a word HMM for each word in the vocabulary for isolated speech recognition, the training or recognition can be implemented directly, using the basic algorithms introduced in Chapter 8. To estimate model parameters, examples of each word in the vocabulary are collected. The model parameters are estimated from all these examples using the forward-backward algorithm and the reestimation formula. It is not necessary to have precise end-point detection, because the silence model automatically determines the boundary if we concatenate silence models with the word model in both ends.

If subword units,[3] such as phonetic models, are used, we need to share them across different words for large-vocabulary speech recognition. These subword units are concatenated to form a word model, possibly adding silence models at the beginning and end, as illustrated in Figure 9.9.

To concatenate subword units to form a word model, you can have a null transition from the final state of the previous subword HMM to the initial state of the next subword HMM, as indicated by the dotted line in Figure 9.9. As described in Chapter 8, you can estimate the parameters of the concatenated HMM accordingly. Please notice that the added null transition arc should satisfy the probability constraint with the transition probability of each phonetic HMM. The self-loop transition probability of the last state in each individual HMM has the topology illustrated in Figure 9.9. If we estimate these parameters with the concatenated model, the null arc transition probability, a_{ij}^{ε}, should satisfy the constraint $\sum_j (a_{ij} + a_{ij}^{\varepsilon}) = 1$ such that the self-loop transition probability of the last state is no longer equal to 1. For interword concatenation or concatenation involving multiple pronunciations, you can use multiple null arcs to concatenate individual models together.

[3] We have a detailed discussion on word models vs. subword models in Section 9.4.1.

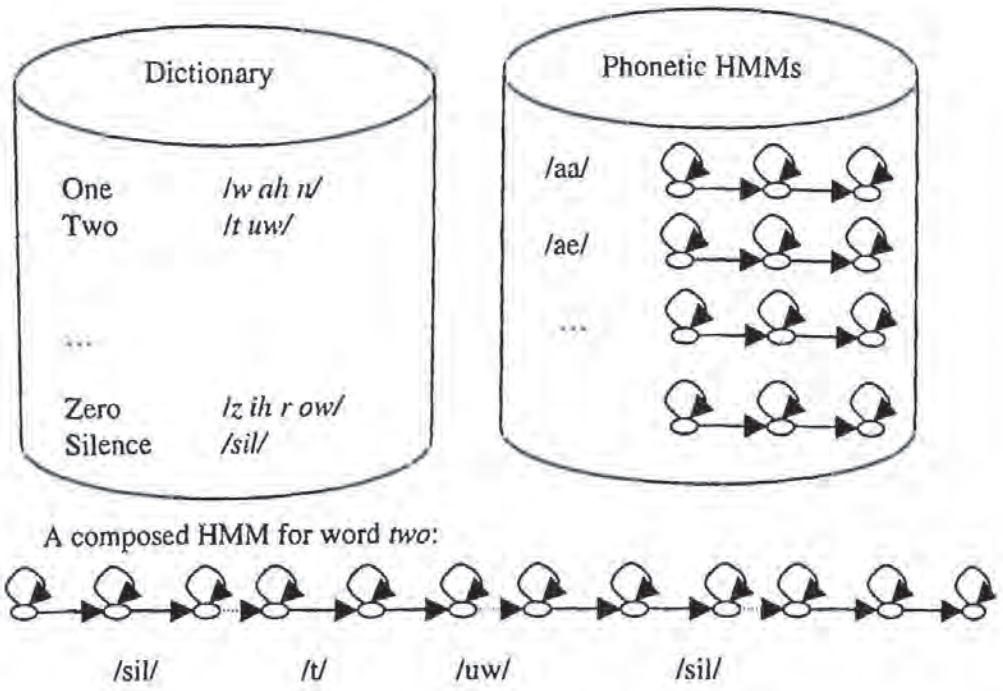


Figure 9.9 The construction of an isolated word model by concatenating multiple phonetic models based on the pronunciation dictionary.

In the example given in Figure 9.9, we have ten English digits in the vocabulary. We build an HMM for each English phone. The dictionary provides information on each word's pronunciation. We have a special word, *Silence*, that maps to a */sil/* HMM that has the same topology as the standard phonetic HMM. For each word in the vocabulary we first derive the phonetic sequence for each word from the dictionary. We link these phonetic models together to form a word HMM for each word in the vocabulary. The link between two phonetic models is shown in the figure as the dotted arrow.

For example, for word *two*, we create a word model based on the beginning silence */sil/*, phone */t/*, phone */uw/*, and ending silence */sil/*. The concatenated word model is then treated in the same manner as a standard large composite HMM. We use the standard forward-backward algorithm to estimate the parameters of the composite HMM from multiple sample utterances of the word *two*. After several iterations, we automatically get the HMM parameters for */sil/*, */t/*, and */uw/*. Since a phone can be shared across different words, the phonetic parameters may be estimated from acoustic data in different words.

The ability to automatically align each individual HMM to the corresponding unsegmented speech observation sequence is one of the most powerful features in the forward-backward algorithm. When the HMM concatenation method is used for continuous speech, you need to compose multiple words to form a sentence HMM based on the transcription of the utterance. In the same manner, the forward-backward algorithm absorbs a range of pos-

sible word boundary information of models automatically. There is no need to have a precise segmentation of the continuous speech.

In general, to estimate the parameters of the HMM, each word is instantiated with its concatenated word model (which may be a concatenation of subword models). The words in the sentence are concatenated with optional silence models between them. If there is a need to modify interword pronunciations due to interword pronunciation change, such as *want you*, you can add a different optional phonetic sequence for *t-y* in the concatenated sentence HMM.

In the digit recognition example, if we have a continuous training utterance *one three*, we compose a sentence HMM, as shown in Figure 9.10, where we have an optional silence HMM between the words *one* and *three*, linked with a null transition from the last state of the word model *one* to the first state of the word model *three*. There is also a direct null arc connection between the models *one* and *three* because a silence may not exist in the training example. These optional connections ensure that all the possible acoustic realizations of the natural continuous speech are considered, so that the forward-backward algorithm can automatically discover the correct path and accurately estimate the corresponding HMM from the given speech observation.

In general, the concatenated sentence HMM can be trained using the forward-backward algorithm with the corresponding observation sequence. Since the entire sentence HMM is trained on the entire observation sequence for the corresponding sentence, most possible word boundaries are inherently considered. Parameters of each model are based on those state-to-speech alignments. It does not matter where the word boundaries are. Such a training method allows complete freedom to align the sentence model against the observation, and no explicit effort is needed to find word boundaries.

In speech decoding, a word may begin and end anywhere within a given speech signal. As word boundaries cannot be detected accurately, all possible beginning and end points have to be accounted for. This converts a linear search (as for isolated word recognition) to a tree search, and a polynomial recognition algorithm to an exponential one. How to design an efficient decoder is discussed in Chapters 12 and 13.

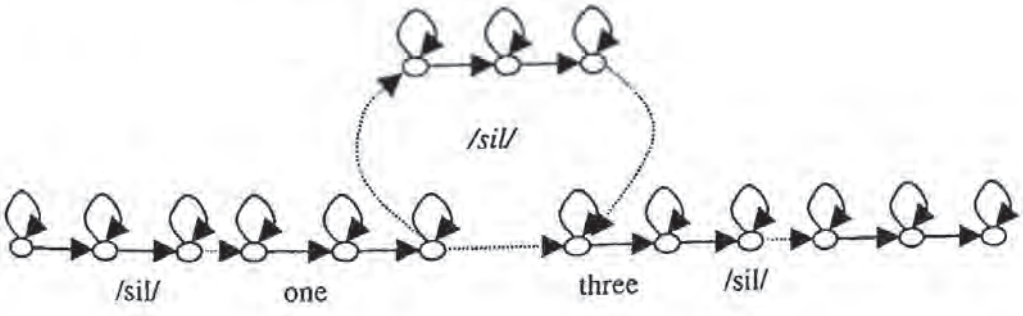


Figure 9.10 A composite sentence HMM. Each word can be a word HMM or a composite phonetic word HMM, as illustrated in Figure 9.9.

9.6. ADAPTIVE TECHNIQUES—MINIMIZING MISMATCHES

As Figure 1.2 illustrated, it is important to adapt both acoustic models and language models for new situations. A decent model can accommodate a wide range of variabilities. However, the mismatch between the model and operating conditions always exists. One of the most important factors in making a speech system usable is to minimize the possible mismatch dynamically with a small amount of *calibration data*. Adaptive techniques can be used to modify system parameters to better match variations in microphone, transmission channel, environment noise, speaker, style, and application contexts. As a concrete example, speaker-dependent systems can provide a significant word error-rate reduction in comparison to speaker-independent systems if a large amount of speaker-dependent training data exists [50]. Speaker-adaptive techniques can bridge the gap between these two configurations with a small fraction of the speaker-specific training data needed to build a full speaker-dependent system. These techniques can also be used incrementally as more speech is available from a particular speaker. When speaker-adaptive models are built, you can have not only improved accuracy but also improved speed and potentially reduced model parameter sizes because of accurate representations, which is particularly appealing for practical speech recognition.

There are a number of ways to use adaptive techniques to minimize mismatches. You can have a nonintrusive adaptation process that works in the background all the time. This is typically unsupervised, using only the outcome of the recognizer (with a high confidence score, as discussed in Section 9.7) to guide the model adaptation. This approach can continuously modify the model parameters so that any nonstationary mismatches can be eliminated. As discussed in Chapter 13, systems that are required to transcribe speech in a non-real-time fashion may use multiple recognition passes. You can use unsupervised adaptation on the test data to improve the models after each pass to improve performance for a subsequent recognition pass.

Since the use of recognition results may be imperfect, there is a possibility of divergence if the recognition error rate is high. If the error rate is low, the adaptation results may still not be as good as supervised adaptation in which the correct transcription is provided for the user to read, a process referred to as the *enrollment process*. In this process you can check a wide range of parameters as follows:

- Check the background noise by asking the user not to speak.
- Adjust the microphone gain by asking the user to speak normally.
- Adapt the acoustic parameters by asking the user to read several sentences.
- Change the decoder parameters for the best speed with no loss of accuracy.
- Compose dynamically new enrollment sentences based on the user-specific error patterns.

The challenge for model adaptation is that we can use only a small amount of observable data to modify model parameters. This constraint requires different modeling strategies

from the ones we discussed in building the baseline system, as the amount of training data is generally sufficient for offline training. In this section we focus on a number of adaptive techniques that can be applied to compensate either speaker or environment variations. Most of these techniques are model-based, since the acoustic model parameters rather than the acoustic feature vectors are adapted. We use speaker-adaptation examples to illustrate how these techniques can be used to improve system performance. We can generalize to environment adaptation by using environment-specific adaptation data and a noise-compensation model, which we discuss in Chapter 10. In a similar manner, we can modify the language model as discussed in Chapter 11.

9.6.1. Maximum a Posteriori (MAP)

Maximum a posteriori (MAP) estimation, as discussed in Chapter 4, can effectively deal with data-sparse problems, as we can take advantage of prior information about existing models. We can adjust the parameters of pretrained models in such a way that limited new training data would modify the model parameters guided by the prior knowledge to compensate for the adverse effect of a mismatch [35]. The prior density prevents large deviations of the parameters unless the new training data provide strong evidence.

More specifically, we assume that an HMM is characterized by a parameter vector Φ that is a random vector, and that prior knowledge about the random vector is available and characterized by a prior probability density function $p(\Phi)$, whose parameters are to be determined experimentally.

With the observation data $\mathbf{X}$, the MAP estimate is expressed as follows:

$$\hat{\Phi} = \arg\max_{\Phi}[p(\Phi \mid \mathbf{X})] = \arg\max_{\Phi}[p(\mathbf{X} \mid \Phi)p(\Phi)] \tag{9.14}$$

If we have no prior information, $p(\Phi)$ is the uniform distribution, and the MAP estimate becomes identical to the ML estimate. We can use the EM algorithm as the ML to estimate the parameters of HMMs. The corresponding Q-function can be defined as:

$$Q_{MAP}(\Phi, \hat{\Phi}) = \log p(\hat{\Phi}) + Q(\Phi, \hat{\Phi}) \tag{9.15}$$

The EM algorithm for the ML criterion can be applied here directly. The actual expression depends on the assumptions made about the prior density. For the widely used continuous Gaussian mixture HMM, there is no joint conjugate prior density. We can assume different components of the HMM to be mutually independent, so that the optimization can be split into different subproblems involving only a single component of the parameter set. For example, the prior density function for the mixture Gaussian can be as follows:

$$p_{b_i}(\mathbf{c}_i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) = p_{c_i}(\mathbf{c}_i)\prod_k p_{b_{ik}}(\boldsymbol{\mu}_{ik}, \boldsymbol{\Sigma}_{ik}) \tag{9.16}$$

where $p_{c_i}(\mathbf{c}_i)$ is a Dirichlet prior density for the mixing coefficient vector of all mixture components in the Markov state i, and $p_{b_{ik}}(\boldsymbol{\mu}_{ik}, \boldsymbol{\Sigma}_{ik})$ denotes the prior density for parameters

of the kth Gaussian component in the state i. The Dirichlet prior density $p_{c_i}(\mathbf{c}_i)$ is characterized by a vector v_i of positive hyperparameters such that:

$$p_{c_i}(\mathbf{c}_i) \propto \prod_k c_{ik}^{v_k - 1} \tag{9.17}$$

For full covariance D-dimensional Gaussian densities, the prior density can be a normal-Wishart density parameterized by two values $\eta > D-1, \tau > 0$, the vector μ_{mw}, and the symmetric positive definite matrix $\mathbf{S}$ as follows:

$$p_{b_{ik}}(\mu_{ik}, \Sigma_{ik}) \propto$$
$$\sqrt{\det(\Sigma_{ik})^{D-\eta}} \exp(-\frac{\tau}{2}(\mu_{ik} - \mu_{mw})\Sigma_{ik}^{-1}(\mu_{ik} - \mu_{mw})' - \frac{1}{2} tr(\mathbf{S}\Sigma_{ik}^{-1})) \tag{9.18}$$

We can apply the same procedure as the MLE Baum-Welch reestimation algorithm. For example, with the Q-function defined in Eq. (9.15), we can apply the Lagrange method to derive the mixture coefficients as follows:

$$\begin{cases} \dfrac{\partial}{\partial \hat{c}_{ik}}(\log p_{c_i}(\hat{\mathbf{c}}_i) + \sum_k \sum_t \xi_t(i,k) \log \hat{c}_{ik}) + \lambda = 0, \forall k \\ \sum_k \hat{c}_{ik} = 1 \end{cases} \tag{9.19}$$

Based on Eqs. (9.17) and (9.19), the solution is:

$$\hat{c}_{ik} = \frac{v_{ik} - 1 + \sum_t \xi_t(i,k)}{\sum_l (v_{il} - 1 + \sum_t \xi_t(i,l))} \tag{9.20}$$

A comparison between Eq. (9.20) and the ML estimate Eq. (8.58) shows that the MAP estimate is a weighted average between the mode of the prior density and the ML estimate with proportions given by $v_{ijk} - 1$ and $\sum \xi_t(i,k)$, respectively.

We can optimize Eq. (9.15) with respect to mean and covariance parameters in a similar fashion. For example, the solution of these estimates is:

$$\hat{\mu}_{ik} = \frac{\tau_{ik}\mu_{mw_{ik}} + \sum_{t=1}^{T} \zeta_t(i,k)\mathbf{x}_t}{\tau_{ik} + \sum_{t=1}^{T} \zeta_t(i,k)} \tag{9.21}$$

$$\hat{\Sigma}_{ik} = \frac{\mathbf{S}_{ik} + \tau_{ik}(\hat{\mu}_{ik} - \mu_{mw_{ik}})(\hat{\mu}_{ik} - \mu_{mw_{ik}})' + \sum_{t=1}^{T} \zeta_t(i,k)(\mathbf{x} - \hat{\mu}_{ik})(\mathbf{x} - \hat{\mu}_{ik})'}{\eta_{ik} - D + \sum_{t=1}^{T} \zeta_t(i,k)} \tag{9.22}$$

where τ_{ik} is the parameter in the normal-gamma density for the corresponding state i.

Thus, the reestimation formula for the Gaussian mean is a weighted sum of the prior mean with the ML mean estimate $\sum_{t=1}^{T}\zeta_t(i,k)\mathbf{x}_t / \sum_{t=1}^{T}\zeta_t(i,k)$. τ_{ik} is a balancing factor between prior mean and the ML mean estimate. When τ_{ik} is large, the value of the prior knowledge is small and the value of the mean μ_{imix} is assumed to have high certainty, leading to the dominance of the final estimate. When the amount of adaptation data increases, the MAP estimate approaches the ML estimate, as the adaptation data overwrite any important prior that may influence the final estimate. Similarly, the covariance estimation formula has the same interpretation of the balance between the prior and new data.

One major limitation of the MAP-based approach is that it requires an accurate initial guess for the prior $p(\Phi)$, which is often difficult to obtain. We can use the already trained initial models that embody some characteristics of the original training conditions. A typical way to generate an initial Gaussian prior is to cluster the initial training data based on speaker or environment similarity measures. We can derive a set of models based on the partition, which can be seen as a set of observations drawn from a distribution having $p(\Phi)$. We can, thus, estimate the prior based on the sample moments to derive the corresponding prior parameters.

Another major limitation is that the MAP-based approach is a local approach to updating the model parameters. Namely, only model parameters that are observed in the adaptation data can be modified from the prior value. When the system has a large number of free parameters, the adaptation can be very slow. Thus in practice we need to find correlations between the model parameters, so that the unobserved or poorly adapted parameters can be altered [3, 22]. Another possibility is to impose structural information so the model parameters can be shared for improved adaptation speed [96].

The MAP training can be iterative, too, which requires an initial estimate of model parameters. A careful initialization for the Gaussian densities is also very important. Unlike the discrete distributions, there is no such a thing as a uniform density for a total lack of information about the value of the parameters. We need to use the same initialization procedure as discussed in Chapter 8.

For speaker-adaptive speech recognition, it has been experimentally found that τ_{ik} can be a fixed constant value for all the Gaussian components across all the dimensions. Thus the MAP HMM can be regarded as an interpolated model between the speaker-independent and speaker-dependent HMM. Both are derived from the standard ML forward-backward algorithm. Experimental performance of MAP training is discussed in Section 9.6.3.

9.6.2. Maximum Likelihood Linear Regression (MLLR)

When the continuous HMM is used for acoustic modeling, the most important parameter set to adapt is the output Gaussian density parameters, i.e., the mean vector and the covariance matrix. We can use a set of linear regression transformation functions to map both means

and covariances in order to maximize the likelihood of the adaptation data [68]. The maximum likelihood linear regression (MLLR) mapping is consistent with the underlying criterion for building the HMM while keeping the number of free parameters under control. Since the transformation parameters can be estimated from a relatively small amount of adaptation data, it is very effective for rapid adaptation. MLLR has been widely used to obtain adapted models for either a new speaker or a new environment condition.

More specifically, in the mixture Gaussian density functions, the kth mean vector $\boldsymbol{\mu}_{ik}$ for each state i can be transformed using following equation:

$$\tilde{\boldsymbol{\mu}}_{ik} = \mathbf{A}_c \boldsymbol{\mu}_{ik} + \mathbf{b}_c \tag{9.23}$$

where $\mathbf{A}_c$ is a regression matrix and $\mathbf{b}_c$ is an additive bias vector associated with some broad class c, which can be either a broad phone class or a set of tied Markov states. The goal of Eq. (9.23) is to map the mean vector into a new space such that the mismatch can be eliminated. Because the amount of adaptation data is small, we need to make sure the number of broad classes c is small so we have only a small number of free parameters to estimate. Equation (9.23) can be simplified into:

$$\tilde{\boldsymbol{\mu}}_{ik} = \mathbf{W}_c \boldsymbol{\mu}_{ik} \tag{9.24}$$

where $\boldsymbol{\mu}_{ik}$ is extended as $[1, \boldsymbol{\mu}_{ik}^t]^t$ and $\mathbf{W}_c$ is the extended transform, $[\mathbf{b}_c, \mathbf{A}_c]$.

This mapping approach is based on the assumption that $\mathbf{W}_c$ can be tied for a wide range of broad phonetic classes so that the overall number of free parameters is significantly less than the number of mean vectors. Therefore, the same transformation can be used for several distributions if they represent similar acoustic characteristics.

To estimate these transformation parameters in the MLE framework, we can use the same Q-function we discussed in Chapter 8. We need to optimize only

$$\sum_i \sum_{k=1}^{M} Q_{b_i}(\Phi, \hat{\mathbf{b}}_{ik}) \tag{9.25}$$

with respect to $\mathbf{W}_c$. Maximization of $Q_{b_i}(\Phi, \hat{\mathbf{b}}_{ik})$ with respect to $\mathbf{W}_c$ can be achieved by computing the partial derivatives. For the Gaussian mixture density function, the partial derivative with respect to $\mathbf{W}_c$ is:

$$\frac{\partial \hat{b}_{ik}(\mathbf{x})}{\partial \mathbf{W}_c} = N(\mathbf{x}, \tilde{\boldsymbol{\mu}}_{ik}, \hat{\boldsymbol{\Sigma}}_{ik}) \boldsymbol{\Sigma}_{ik}^{-1} (\mathbf{x} - \mathbf{W}_c \boldsymbol{\mu}_{ik}) \boldsymbol{\mu}_{ik}^t \tag{9.26}$$

Let us denote the set of Gaussian components forming the broad transformation classes as C; we use $b_{ik} \in C$ to denote that the k^{th} Gaussian density in state i belongs to the

class. We can expand the Q-function with the partial derivatives and set it to zero, leading to the following equation:

$$\sum_{t=1}^{T}\sum_{b_k \in C}\zeta_t(i,k)\Sigma_{ik}^{-1}\mathbf{x}_t\mu_{ik}' = \sum_{t=1}^{T}\sum_{b_k \in C}\zeta_t(i,k)\Sigma_{ik}^{-1}\mathbf{W}_c\mu_{ik}\mu_{ik}' \tag{9.27}$$

We can rewrite Eq. (9.27) as:

$$\mathbf{Z} = \sum_{b_k \in C}\mathbf{V}_{ik}\mathbf{W}_c\mathbf{D}_{ik} \tag{9.28}$$

where

$$\mathbf{Z} = \sum_{t=1}^{T}\sum_{b_k \in C}\zeta_t(i,k)\Sigma_{ik}^{-1}\mathbf{x}_t\mu_{ik}', \tag{9.29}$$

$$\mathbf{V}_{ik} = \sum_{t=1}^{T}\zeta_t(i,k)\Sigma_{ik}^{-1}, \tag{9.30}$$

and

$$\mathbf{D}_{ik} = \mu_{ik}\mu_{ik}'. \tag{9.31}$$

Estimating $\mathbf{W}_c$ for Eq. (9.28) is computationally expensive, as it requires solving simultaneous equations. Nevertheless, if we assume that the covariance matrix is diagonal, we can have a closed-form solution that is computationally efficient. Thus, we can define

$$\mathbf{G}_q = \sum_{b_{qk} \in C} v_{qq}\mathbf{D}_{ik} \tag{9.32}$$

where v_{qq} denotes the q^{th} diagonal element of matrix $\mathbf{V}_{ik}$. The transformation matrix can be computed row by row. So for the q^{th} row of the transformation matrix $\mathbf{W}_q$, we can derive it from the q^{th} row of $\mathbf{Z}_q$ [defined in Eq. (9.29)] as follows:

$$\mathbf{W}_q = \mathbf{Z}_q\mathbf{G}_q^{-1} \tag{9.33}$$

Since $\mathbf{G}_q$ may be a singular matrix, we need to make sure we have enough training data for the broad class. Thus, if the amount of training data is limited, we must tie a number of transformation classes together.

We can run several iterations to maximize the likelihood for the given adaptation data. At each iteration, transformation matrices can be initialized to identity transformations. We can iteratively repeat the process to update the means until convergence is achieved. We can

also incrementally adapt the mean vectors after each observation sequence or set of observation sequences while the required statistics are accumulated over time. Under the assumption that the alignment of each observation sequence against the model is reasonably accurate, we can accumulate these estimated counts over time and use them incrementally. In order to deal with the tradeoff between specificity and robust estimation, we can dynamically generate regression classes according to the senone tree. Thus, we can incrementally increase the number of regression classes when more and more data become available.

MLLR adaptation can be generalized to include the variances with the ML framework, although the additional gain after mean transformation is often less significant (less than relative 2% error reduction). When the user donates about 15 sentences for enrollment training, Table 9.5 illustrates how the MLLR adaptation technique can be used to further reduce the word recognition error rate for a typical dictation application. Here, there is only one context-independent phonetic class for all the context-dependent Gaussian densities. As we can see, most of the error reduction came from adapting the mean vectors.

We can further extend MLLR to *speaker-adaptive training* (SAT) [6, 74]. In conventional speaker-independent training, we simply use data from different speakers to build a speaker-independent model. An inherent difficulty in this approach is that spectral variations of different speakers give the speaker-independent acoustic model higher variance than the corresponding speaker-dependent model. We can include MLLR transformation in the process of training to derive the MLLR parameters for each individual speaker. Thus the training data are transformed to maximize the likelihood for the overall speaker-independent model. This process can be run iteratively to reduce mismatches of different speakers. By explicitly accounting for the interspeaker variations during training and decoding, SAT reduces the error rate by an additional 5–10%.

Table 9.5 Relative error reductions with MLLR methods. The reduction is relative to that of the preceding row.

Models	Relative Error Reduction
CHMM	Baseline
MLLR on mean only	+12%
MLLR on mean and variance	+2%
MLLR SAT	+8%

9.6.3. MLLR and MAP Comparison

The MLLR method can be combined with MAP. This guarantees that with the increased amount of training data, we can have not only a set of compact MLLR transformation functions for rapid adaptation, but also directly modified model parameters that converge to the ML estimates. We can use MAP to adapt the model parameters and then add MLLR to transform these adapted models. It is also possible to incorporate the MAP principle directly into MLLR [18, 19].

As an example, the result of a 60,000-word dictation application using various adaptation methods is shown in Figure 9.11.[4] The speaker-dependent model used 1000 utterances. Also included as a reference is the speaker-independent result, which is used as the starting point for adaptive training. When the speaker-independent model is adapted with about 200 utterances, the speaker-adaptive model has already outperformed both speaker-independent and speaker-dependent systems. The results clearly demonstrate that we have insufficient training data for speaker-dependent speech recognition, as MAP-based outperform ML-based models. This also illustrates that we can make effective use of speaker-independent data for speaker-dependent speech recognition. Also, notice that the MLLR method has a faster adaptation rate than the MAP method. The MLLR method has context-independent phonetic classes. So, when the amount of adaptation data is limited, the MLLR method offers better overall performance.

However, the MAP becomes more accurate when the amount of adaptation data increases to 600 per speaker. This is because we can modify all the model parameters with the MAP training, and the MLLR transformation can never have the same degrees of freedom as the MAP method. When the MLLR is combined with MAP, we can have not only rapid adaptation but also superior performance over either the MLLR or MAP method across a wide range of adaptation data points. There are a number of different ways to combine both MLLR and MAP for improved performance [4, 98].

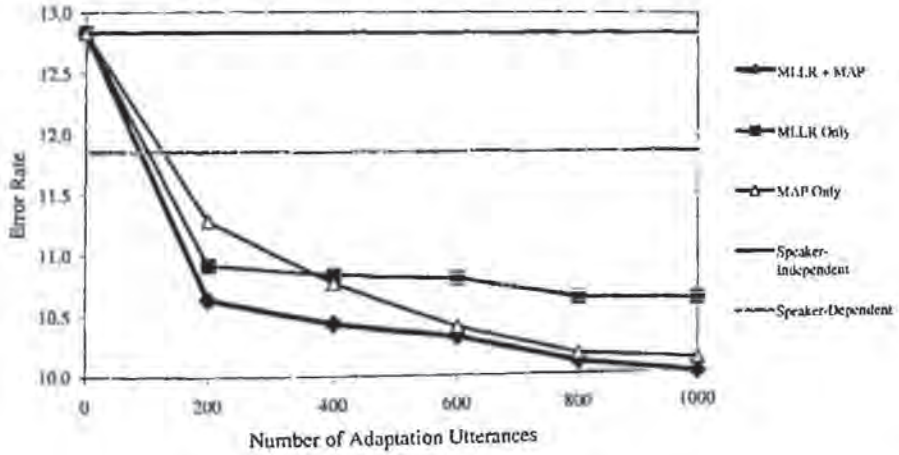


Figure 9.11 Comparison of Whisper with MLLR, MAP, and combined MLLR and MAP. The error rate is shown for a different amount of adaptation data. The speaker-dependent and speaker-independent models are also included. The speaker-dependent model was trained with 1000 sentences.

[4] In practice, if a large well-trained, speaker-independent model is used, the baseline performance may be very good, and hence, the relative error reduction from speaker adaptation may be smaller than for smaller and simpler models.

9.6.4. Clustered Models

Both MAP and MLLR techniques are based on using an appropriate initial model for adaptive modeling. How accurate we make the initial model directly affects the overall performance. An effective way to minimize the mismatch is, thus, to cluster similar speakers and environments in the training data, building a set of models for each cluster that has minimal mismatch for different conditions. When we have enough training data, and enough coverage for a wide range of conditions, this approach ensures significantly improved robustness.

For example, we often need a set of clustered models for different telephone channels, including different cellular phone standards. We also need to build gender-dependent models or speaker-clustered models for improved performance. In fact, when we construct speaker-clustered models, we can apply MLLR transformations or neural networks to minimize speaker variations such that different speakers can be mapped to the same golden speaker that is the representative of the cluster.

Speaker clusters can be created based on the information of each speaker-dependent HMM. The clustering procedure is similar to the decision-tree procedure discussed in Section 9.4.3. Using clustered models increases the amount of computational complexity. It also fragments the training data. Clustering is often needed to combine other smoothing techniques, such as deleted interpolation or MLLR transformation, in order to create clustered models from the pooled model. We can also represent a speaker as a weighted sum of individual speaker cluster models with the cluster adaptive training [33] or eigenvoice techniques [64].

When we select an appropriate model, we can compute the likelihood of the test speech against all the models and select the model that has the highest likelihood. Alternatively, we can compute likelihoods as part of the decoding process and prune away less promising models dynamically without significantly increasing the computational load. When multiple models are plausible, we can compute the weighted sum of the clustered models with pretrained mixing coefficients for different clusters, much as we train the deleted interpolation weights.

Traditionally speaker clustering is performed across different speakers without considering phonetic similarities across different speakers. In fact, clustered speaker groups may have very different degrees of variations for different phonetic classes. You can further generalize speaker clustering to the subword or subphonetic level [62]. With multiple instances derived from clustering for each subword or subphonetic unit, you can model speaker variation explicitly across different subword or subphonetic models.

In practice, gender-dependent models can reduce the word recognition error by 10%. More refined speaker-clustered models can further reduce the error rate, but not as much as the gain from gender-dependent models, unless we have a large number of clustered speakers. If the new user happens to be similar to one of these speaker clusters, we can approach speaker-dependent speech recognition without enrollment. For environment-dependent models, clustering is more critical. The challenge is to anticipate the kind of environment or channel distortions the system will have to deal with. Since this is often unpredictable, we

need to use adaptive techniques such as MAP and MLLR to minimize the mismatch. We discuss this in more detail in Chapter 10.

9.7. CONFIDENCE MEASURES: MEASURING THE RELIABILITY

One of the most critical components in a practical speech recognition system is a reliable *confidence measure*. With an accurate confidence measure for each recognized word, the conversational back end can repair potential speech recognition errors, can reject out-of-vocabulary words, and can identify key words (perform word spotting) that are relevant to the back end. In a speaker-dependent or speaker-adaptive system, the confidence measure can help user enrollment (to eliminate mispronounced words). It is also critical for unsupervised speaker adaptation, allowing selective use of recognition results so that transcriptions with lower confidence can be discarded for adaptation.

In theory, an accurate estimate of $P(\mathbf{W} \mid \mathbf{X})$, the posterior probability, is itself a good confidence measure for word $\mathbf{W}$ given the acoustic input $\mathbf{X}$. Most practical speech recognition systems simply ignore $P(\mathbf{X})$, as it is a constant in evaluating $P(\mathbf{W})P(\mathbf{X} \mid \mathbf{W}) / P(\mathbf{X})$ across different words. $P(\mathbf{W} \mid \mathbf{X})$ can be expressed:

$$P(\mathbf{W} \mid \mathbf{X}) = \frac{P(\mathbf{W})P(\mathbf{X} \mid \mathbf{W})}{P(\mathbf{X})} = \frac{P(\mathbf{W})P(\mathbf{X} \mid \mathbf{W})}{\sum_{\mathbf{W}} P(\mathbf{W})P(\mathbf{X} \mid \mathbf{W})} \tag{9.34}$$

Equation (9.34) essentially provides a solid framework for measuring confidence levels. It is the ratio between the score for the word hypothesis $P(\mathbf{W})P(\mathbf{X} \mid \mathbf{W})$ and the acoustic probability $\sum_{\mathbf{W}} P(\mathbf{W})P(\mathbf{X} \mid \mathbf{W})$. In the sections that follow we discuss a number of ways to model and use such a ratio in practical systems.

9.7.1. Filler Models

You can compute $P(\mathbf{X})$ in Eq. (9.34) with a general-purpose recognizer. It should be able to recognize anything such that it can *fill the holes* of the grammar in the normal speech recognizer. The filler model has various forms [7, 63]. One of the most widely used is the so-called all-phone network, in which all the possible phonetic and nonspeech HMMs are connected to each other, and with which any word sequence can be recognized.

In addition to evaluating $P(\mathbf{W})P(\mathbf{X} \mid \mathbf{W})$ as needed in normal speech recognition, a separate decoding process is used to evaluate $\sum_{\mathbf{W}} P(\mathbf{W})P(\mathbf{X} \mid \mathbf{W})$. Here $\mathbf{W}$ is either a phonetic or a word model. You can also apply phonetic n-gram probabilities that are derived from a lexicon targeted for possible new words. The best path from the all-phone network is compared with the best path from the normal decoder. The ratio between the two, as expressed in Eq. (9.34), is used to measure the confidence for either word or phone. In the decoding process (see Chapters 12 and 13), you can accumulate the phonetic ratio derived from Eq. (9.34) on a specific word. If the accumulative $P(\mathbf{W} \mid \mathbf{X})$ for the word is less than a predetermined threshold, the word is rejected as either a new word or a nonspeech event.

Both context-independent and context-dependent phonetic models can be used for the fully connected network. When context-dependent phonetic models are used, you need to make sure that only correct contextual phonetic connections are made. Although context-dependent models offer significant improvement for speech recognition, the filler phonetic network seems to be insensitive to context-dependency in empirical experiments.

There are *word-spotting* applications that need to spot just a small number of key words. You can use the filler models described here for word spotting. You can also build antiword models trained with all the data that are not associated with the key words of interest. Empirical experiments indicate that large-vocabulary speech recognition is the most suitable choice for word spotting. You can use a general-purpose n-gram (see Chapter 11) to generate recognition results and identify needed key words from the word lattice. This is because a large-vocabulary system provides a better estimate of $\sum_{\mathbf{W}} P(\mathbf{W})P(\mathbf{X}\,|\,\mathbf{W})$ with a more accurate language model probability. In practice, we don't need to use all the hypotheses to compute $\sum_{\mathbf{W}} P(\mathbf{W})P(\mathbf{X}\,|\,\mathbf{W})$. Instead, n-best lists and scores [40] can be used to provide an effective estimate of $\sum_{\mathbf{W}} P(\mathbf{W})P(\mathbf{X}\,|\,\mathbf{W})$.

9.7.2. Transformation Models

To determine the confidence level for each word, subword confidence information is often helpful. Different phones have different impacts on our perception of words. The weight for each subword confidence score can be optimized from the real training data. If a word w has N phones, we can compute the confidence score of the word as follows:

$$CS(w) = \sum_{i=1}^{N} \wp_i(x_i)/N \tag{9.35}$$

where $CS(w)$ is the confidence score for word w, x_i is the confidence score for subword unit i in word w, and $\wp_i$ is the mapping function that may be tied across a number of subword units. The transformation function can be defined as:

$$\wp_i(x) = ax + b \tag{9.36}$$

We can use discriminative training to optimize the parameters a and b, respectively. A cost function can be defined as a sigmoid function of $CS(w)$. As shown in Figure 9.12, the optimal transformation parameters vary substantially across different phones. The weight for consonants is also typically larger than that of vowels.

The transformation function can be context dependent. Figure 9.13 illustrates the ROC curve of the context-dependent transformation model in comparison with the corresponding phonetic filler model. The filler model essentially has a uniform weight across all the phones in a given word. The estimated transformation model has 15–40% false-acceptance error reduction at various fixed false-rejection rates. The false-acceptance rate of the transformation model is consistently lower than that of the filler model [63].

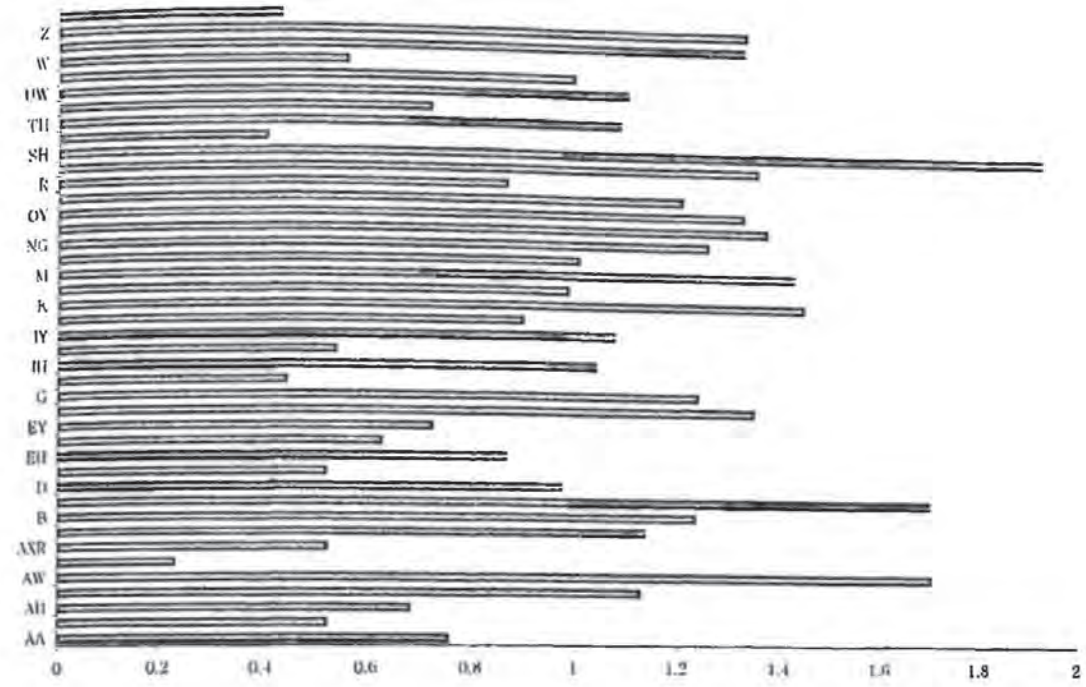


Figure 9.12 Transformation parameter a for each context-independent phone class. The weight of consonants is typically larger than that of vowels [63].

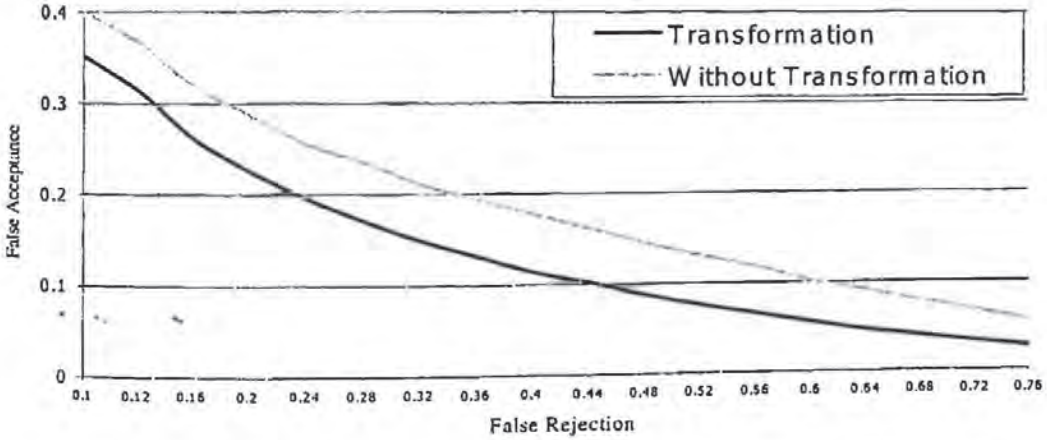


Figure 9.13 The ROC curve of phonetic filler models with and without optimal feature transformation [63].

9.7.3. Combination Models

In practical systems, there are a number of features you can use to improve the performance of confidence measures. For example, the following features are helpful:

- Word stability ratio from different language model weights (*WrdStabRatio*). This is obtained by applying different language weights to see how stably each word shows up in the recognition *n*-best list.
- Logarithm of the average number of active words around the ending frame of the word (*WrdCntEnd*).
- Acoustic score per frame within the word normalized by the corresponding active senone scores (*AscoreSen*).
- Logarithm of the average number of active words within the word (*WrdCntW*).
- Acoustic score per frame within the word normalized by the phonetic filler model (*AscoreFiller*).
- Language model score (*LMScore*).
- Language model back-off (trigram, bigram, or unigram hit) for the word (*LMBackOff*).
- Logarithm of the average number of active states within the word (*StateCnt*).
- Number of phones in the word (*Nphones*).
- Logarithm of the average number of active words around the beginning frame of the word (*WrdCntBeg*).
- Whether the word has multiple pronunciations (*Mpron*).
- Word duration (*WordDur*).

To clarify each feature's relative importance, Table 9.6 shows its linear correlation coefficient against the correct/incorrect tag for each word in the training set. Word stability ratio (*WrdStabRatio*) has the largest correlation value.

Several kinds of classifiers can be used to compute the confidence scores. Previous research has shown that the difference between classifiers, such as linear classifiers, generalized linear models, decision trees, and neural networks, is insignificant. The simplest linear classifier based on discriminative training performs well in practice. As some features are highly correlated, you can iteratively remove features to combat the curse of dimensionality. The combination model can have up to 40–80% false-acceptance error reduction at fixed false-rejection rate in comparison to the single-feature approach.

Table 9.6 Correlation coefficients of several features against correct/incorrect tag.

Feature	Correlation
WrdStabRatio	0.590
WrdCntW	−0.223
LMBackOff	0.171
AscoreSen	0.250
LMScore	0.175
Nphones	0.091
WordDur	0.012
WrdCntEnd	−0.321
AscoreFiller	0.219
StateCnt	−0.155
Mpron	0.057
WrdCntBeg	−0.067

9.8. OTHER TECHNIQUES

In addition to HMMs, a number of interesting alternative techniques are being actively investigated by researchers. We briefly review two promising methods here.

9.8.1. Neural Networks

You have seen both single-layer and multilayer neural nets in Chapter 4 for dealing with static patterns. In dealing with nonstationary signals, you need to address how to map an input sequence properly to an output sequence when two sequences are not synchronous, which should include proper alignment, segmentation, and classification. The basic neural networks are not well equipped to address these problems in a unified way.

Recurrent neural networks have an internal state that is a function of the current input and the previous internal state. A number of them use time-step delayed recurrent loops on the hidden or output units of a feedforward network, as discussed in earlier chapters. For sequences of finite numbers of delays, we can transform these networks into equivalent feedforward networks by unfolding them over the time period. They can be trained with the standard back propagation procedure, with the following modifications:

- The desired outputs are functions of time, and error functions have to be computed for every copy of the output layer. This requires the selection of an appropriate time-dependent target function, which is often difficult to define.
- All copies of the unfolded weights are constrained to be identical during the training. We can compute the correction terms separately for each weight and use the average to update the final estimate.

In most of these networks, you can have a partially recurrent network that has feedback of the hidden and output units to the input layer. For example, the feedforward network can be used in a set of local feedforward connections with one time-step delay. These networks are usually implemented by extending the input field with additional feedback units containing both the hidden and output values generated by the preceding input. You can encode the past nonstationary information that is often required to generate the correct output, given the current input, as illustrated in Figure 9.14.

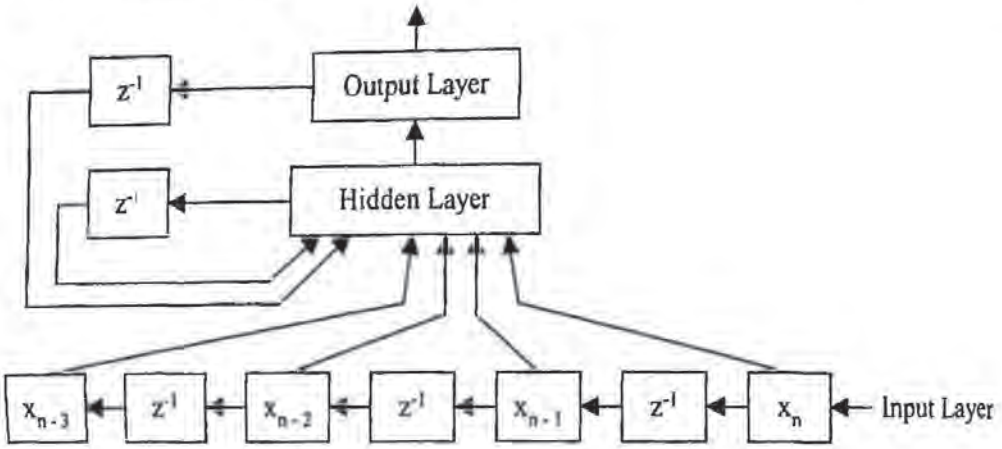


Figure 9.14 A recurrent network with contextual inputs, hidden vector feedback, and output vector feedback.

One of the popular neural networks is the *Time Delay Neural Network* (TDNN) [105]. Like static networks, the TDNN can be trained to recognize a sequence of predefined length (defined by the width of the input window). The activation in the hidden layer is computed from the current and multiple time-delayed values of the preceding layer, and the output units are activated only when a complete speech segment has been processed. A typical TDNN is illustrated in Figure 9.15. The TDNN has been successfully used to classify presegmented phonemes.

All neural networks have been shown to yield good performance for small-vocabulary speech recognition. Sometimes they are better than HMMs for short, isolated speech units. By recurrence and the use of temporal memory, they can perform some kind of integration over time. It remains a challenge for neural networks to demonstrate that they can be as effective as HMMs for dealing with nonstationary signals, as is often required for large-vocabulary speech recognition.

To deal with continuous speech, the most effective solution is to integrate neural nets with HMMs [91, 113]. The neural network can be used as the output probabilities to replace the Gaussian mixture densities. Comparable results can be obtained with the integrated approach. These HMM output probabilities could be estimated by applying the Bayes' rule to the output of neural networks that have been trained to classify HMM state categories. The neural networks can consist either of a single large trained network or of a group of separately trained small networks [21, 31, 75, 90].

A number of techniques have been developed to improve the performance of training these networks. Training can be embedded in an EM-style process. For example, dynamic programming can be used to segment the training data. The segmented data are then used to retrain the network. It is also possible to have Baum-Welch style training [14, 42].

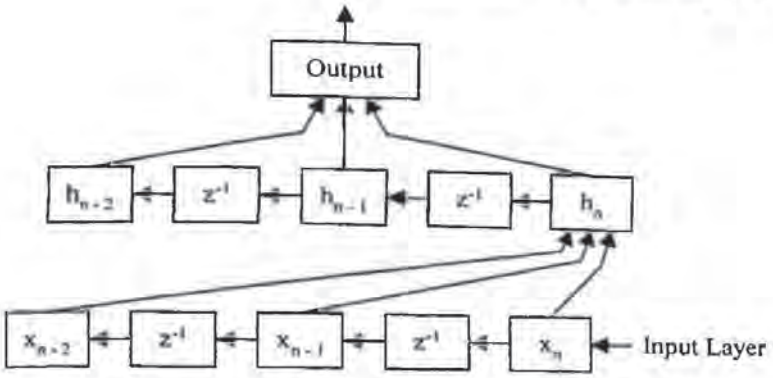


Figure 9.15 A time-delay neural network (TDNN), where the box h_t denotes the hidden vector at time t, the box x_t denotes the input vector at time t, and the box z^{-1} denotes a delay of one sample.

9.8.2. Segment Models

As discussed in Chapter 8, the HMM *output-independence assumption* results in a piecewise stationary process within an HMM state. Although the nonstationary speech may be modeled sufficiently with a large number of states, the states in which salient speech features are present are far from stationary [25, 99]. While the use of time-derivative features (e.g., delta and/or delta-delta features) alleviates these limitations, the use of such longer-time-span features may invalidate the conditional independence assumption.

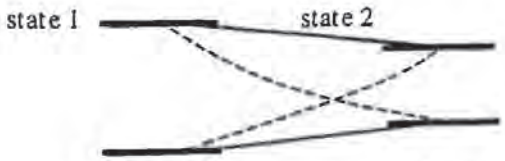


Figure 9.16 Diagram illustrating that HMM's output observation can hop between two unexpected quasi-stationary states [46].

The use of Gaussian mixtures for continuous or semicontinuous HMMs, as described in Chapter 8, could introduce another potential problem, where arbitrary transitions among the Gaussian mixture components between adjacent HMM states are allowed [59]. Figure 9.16 illustrates two HMM states with two mixture components. The solid lines denote the valid trajectories actually observed in the training data. However, in modeling these two trajectories, the Gaussian mixtures inadvertently allow two *phantom* trajectories, shown in

dashed lines in Figure 9.16, because no constraint is imposed on the mixture transitions across the state. It is possible that such phantom trajectories degrade recognition performance, because the models can overrepresent speech signals that should be modeled by other acoustic units. Segment models can alleviate such HMM modeling deficiencies [77, 79].

In the standard HMM, the output probability distribution is modeled by a quasi-stationary process, i.e.,

$$P(\mathbf{x}_1^L \mid s) = \prod_{l=1}^{L} b_s(\mathbf{x}_l) \tag{9.37}$$

For the segment model (SM), the output observation distribution is modeled by two stochastic processes:

$$P(\mathbf{x}_1^L \mid s) = P(\mathbf{x}_1^L \mid s, L)P(L \mid s) \tag{9.38}$$

The first term of Eq. (9.38) is no longer decomposable in the absence of the output-independence assumption. The second term is similar to the duration model described in Chapter 8. In contrast to the HMM whose quasi-stationary process for each state s generates one frame $\mathbf{x}_l$, a state in a segment model can generate a variable-length observation sequence $\{\mathbf{x}_1, \mathbf{x}_2, \cdots \mathbf{x}_L\}$ with random length L.

Since the likelihood evaluation of segment models cannot be decomposed, the computation of the evaluation is not shareable between different segments (even for the case where two segments differ only by one frame). This results in a significant increase in computation for both training and decoding [77]. In general, the search state space is increased by a factor of $L_{\max}$, the maximum segment duration. If segment models are used for phone segments, $L_{\max}$ could be as large as 60. On top of this large increase in search state space, the evaluation of segment models is usually an order of magnitude more expensive than for HMM, since the evaluation involves several frames. Thus, the segment model is often implemented in a multipass search framework, as described in Chapter 13.

Segment models have produced encouraging performance for small-vocabulary or isolated recognition tasks [25, 44, 79]. However, their effectiveness on large-vocabulary continuous speech recognition remains an open issue because of necessary compromises to reduce the complexity of implementation.

9.8.2.1. Parametric Trajectory Models

Parametric trajectory models [25, 37] were first proposed to model a speech segment with curve-fitting parameters. They approximate the D-dimensional acoustic observation vector $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_T)$ by a polynomial function. Specifically, the observation vector $\mathbf{x}_t$ can be represented as

$$\mathbf{x}_t = \mathbf{C} \times F_t + \mathbf{e}_t(\Sigma) = \begin{pmatrix} c_1^0 & c_1^1 & \cdots & c_1^N \\ c_2^0 & c_2^1 & \cdots & c_2^N \\ \vdots & \vdots & \vdots & \vdots \\ c_D^0 & c_D^1 & c_D^2 & c_D^N \end{pmatrix} \begin{pmatrix} f_0(t) \\ f_1(t) \\ \vdots \\ f_N(t) \end{pmatrix} + \mathbf{e}_t(\Sigma) \tag{9.39}$$

where the matrix $\mathbf{C}$ is the trajectory parameter matrix, F_t is the family of N^{th}-order polynomial functions, and $\mathbf{e}_t(\Sigma)$ is the residual fitting error. Equation (9.39) can be regarded as modeling the time-varying mean in the output distribution for an HMM state. To simplify computation, the distribution of the residual error is often assumed to be an independent and identically distributed random process with a normal distribution $N(0, \Sigma)$. To accommodate diverse durations for the same segment, the relative linear time sampling of the fixed trajectory is assumed [37].

Each segment M is characterized by a trajectory parameter matrix $\mathbf{C}_m$ and covariance matrix Σ_m. The likelihood for each frame can be specified [46] as

$$P(\mathbf{x}_t \mid \mathbf{C}_m, \Sigma_m) = \frac{\exp\left\{-tr\left[(\mathbf{x}_t - \mathbf{C}_m F_t)\Sigma_m^{-1}(\mathbf{x}_t - \mathbf{C}_m F_t)^t\right]/2\right\}}{(2\pi)^{D/2} \mid \Sigma_m \mid^{1/2}} \tag{9.40}$$

If we let $\mathbf{F}_T = (F_0, F_1, \cdots, F_{T-1})^t$, then the likelihood for the whole acoustic observation vector can be expressed as

$$P(\mathbf{X} \mid \mathbf{C}_m, \Sigma_m) = \frac{\exp\left\{-tr\left[(\mathbf{X} - \mathbf{C}_m \mathbf{F}_T)\Sigma_m^{-1}(\mathbf{X} - \mathbf{C}_m \mathbf{F}_T)^t\right]/2\right\}}{(2\pi)^{DT/2} \mid \Sigma_m \mid^{T/2}} \tag{9.41}$$

Multiple mixtures can also be applied to SM. Suppose segment M is modeled by K trajectory mixtures. The likelihood for the acoustic observation vector $\mathbf{X}$ becomes

$$\sum_{k=1}^{K} w_k p_k(\mathbf{X} \mid \mathbf{C}_k, \Sigma_k) \tag{9.42}$$

Hon et al. [47] showed that only a handful of target trajectories are needed for speaker-independent recognition, in contrast to the many mixtures required for continuous Gaussian HMMs. This should support the phantom-trajectory argument involved in Figure 9.16.

The estimation of segment parameters can be accomplished by the EM algorithm described in Chapter 4. Assume a sample of L observation segments $\mathbf{X}_1, \mathbf{X}_2, \cdots, \mathbf{X}_L$, with corresponding duration $T_1, T_2, \cdots, T_L$, are generated by the segment model M. The MLE formulae using the EM algorithm are:

$$\gamma_k^i = \frac{w_k p_k(\mathbf{X}_i \mid \mathbf{C}_k, \Sigma_k)}{P(\mathbf{X}_i \mid \Phi_m)} \tag{9.43}$$

$$\hat{w}_k = \frac{1}{L}\sum_{i=1}^{L}\frac{w_k p_k(\mathbf{X}_i \mid \mathbf{C}_k, \Sigma_k)}{P(\mathbf{X}_i \mid \Phi_m)} \tag{9.44}$$

$$\hat{\mathbf{C}}_k = \left[\sum_{i=1}^{L}\gamma_k^i \mathbf{X}_i \mathbf{F}_{T_i}^t\right] / \left[\sum_{i=1}^{L}\gamma_k^i \mathbf{F}_{T_i}\mathbf{F}_{T_i}^t\right] \tag{9.45}$$

$$\hat{\Sigma}_k = \sum_{i=1}^{L}\gamma_k^i (\mathbf{X}_i - \mathbf{C}_k\mathbf{F}_{T_i})(\mathbf{X}_i - \mathbf{C}_k\mathbf{F}_{T_i})^t \Big/ \sum_{i=1}^{L}\gamma_k^i T_i \tag{9.46}$$

Parametric trajectory models have been successfully applied to phone classification [25, 46] and word spotting [37], and offer a modestly improved performance over HMMs.

9.8.2.2. Unified Frame- and Segment-Based Models

The strengths of the HMM and the segment-model approaches are complementary. HMMs are very effective in modeling the subtle details of speech signals by using one state for each quasi-stationary region. However, the transitions between quasi-stationary regions are largely neglected by HMMs because of their short durations. In contrast, segment models are powerful in modeling the transitions and longer-range speech dynamics, but might need to give up the detailed modeling to assure trainability and tractability. It is possible to have a unified framework to combine both methods [47].

In the unified complementary framework, the acoustic model $p(\mathbf{X}_1^T \mid \mathbf{W})$ can be considered as two joint hidden processes, as in the following equation:

$$\begin{aligned} p(\mathbf{X} \mid \mathbf{W}) &= \sum_{\mathbf{q}^h}\sum_{\mathbf{q}^s} p(\mathbf{X}, \mathbf{q}^h, \mathbf{q}^s \mid \mathbf{W}) \\ &= \sum_{\mathbf{q}^h}\sum_{\mathbf{q}^s} p(\mathbf{X} \mid \mathbf{q}^h, \mathbf{q}^s) p(\mathbf{q}^s \mid \mathbf{q}^h) p(\mathbf{q}^h \mid \mathbf{W}) \end{aligned} \tag{9.47}$$

where $\mathbf{q}^h$ represents the hidden process of the HMM and $\mathbf{q}^s$, the segment model. The conditional probability of the acoustic signal $p(\mathbf{X} \mid \mathbf{q}^s, \mathbf{q}^h)$ can be further decomposed into two separate terms:

$$p(\mathbf{X} \mid \mathbf{q}^s, \mathbf{q}^h) = p(\mathbf{X} \mid \mathbf{q}^h) p(\mathbf{X} \mid \mathbf{q}^s)^a \tag{9.48}$$

where a is a constant that is called *segment-model weight*. The first term is the contribution from normal frame-based HMM evaluation. We further assume for the second term that recognition of segment units can be performed by detecting and decoding a sequence of salient events in the acoustic stream that are statistically independent. In other words,

$$p(\mathbf{X} \mid \mathbf{q}^s) = \prod_i p(\mathbf{X}_i \mid q_i^s) \tag{9.49}$$

where $\mathbf{X}_i$ denotes the i^{th} segment.

We assume that the phone sequence and the phone boundaries hypothesized by HMMs and segment models agree with each other. Based on the independent-segment assumption, this leads to a segment duration model as

$$p(\mathbf{q}^s \mid \mathbf{q}^h) = \prod_i p(t_i, t_{i+1} - 1 \mid \mathbf{X}_i) \tag{9.50}$$

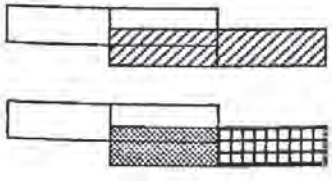
By treating the combination as a hidden-data problem, we can apply the decoding and iterative EM reestimation techniques here. This unified framework enables both frame- and segment-based models to *jointly* contribute to optimal segmentations, which leads to more efficient pruning during the search. The inclusion of the segment models does not require massive revisions in the decoder, because the segment model scores can be handled in the same manner as the language model scores; whereas the segment evaluation is performed at each segment boundary.

Since subphonetic units are often used in HMMs to model the detailed quasi-stationary speech region, the segment units should be used to model long-range transition. As studies have shown that phone transitions play an essential role in humans' perception, the phone-pair segment unit that spans over two adjacent phones can be used [47]. Let e_i and t_i denote the phone and the starting time of the i^{th} segment, respectively. For a phone-pair (e_{i-1}, e_i) segment between t_i and t_{i+1}, the segment likelihood can be computed as follows:

$$p(\mathbf{X}_i \mid q_i^s) = p(\mathbf{x}_{t_{i-1}}^{t_i-1} \mid e_{i-1}, e_i) \tag{9.51}$$

Rather than applying segment evaluation for every two phones, an *overlapped evaluation* scheme can be used, as shown in Figure 9.17 (a), where a phone-pair segment model evaluation is applied at each phone boundary. The overlapped evaluation implies that each phone is evaluated twice in the overall score. Most importantly, the overlapped evaluation places constraints on overlapped regions to assure consistent trajectory transitions. This is an important feature, as trajectory mixtures prohibit *phantom* trajectories within a segment unit, but there is still no mechanism to prevent arbitrary trajectory transitions between adjacent segment units.

Some phone-pairs might not have sufficient training data. Units containing silence might also have obscure trajectories due to the arbitrary duration of silence. As a result, a



(a) phone-pair segment models

(b) two phone (monophone or gen. triphone) segment models

Figure 9.17 Overlapped evaluation using (a) a phone-pair segment model, or (b) back-off to two phone units when the phone-pair (e_{i-1}, e_i) segment model does not exist [47].

phone-pair unit (e_{i-1}, e_i) can be backed off with two phone units as shown in Figure 9.17 (b). The phone units can be context independent or context dependent [46]. Thus, the back-off segment-model evaluation becomes:

$$p(\mathbf{X}_i \mid q_i^s) = \beta * p(\mathbf{x}_{i,-1}^{t_i} \mid e_{i-1}) p(\mathbf{x}_{i,}^{t_{i+1}} \mid e_i) \tag{9.52}$$

where β is the back-off weight, generally smaller than 1.0. The use of back-off weight has the effect of giving more preference to phone-pair segment models than to two-phone-based back-off segment models.

The phone-pair segment model outperformed the phone-pair HMM by more than 20% in a phone-pair classification experiment [46]. The unified framework achieved about 8% word-error-rate reduction on the WSJ dictation task in comparison to the HMM-based Whisper [47].

9.9. CASE STUDY: WHISPER

Microsoft's Whisper engine offers general-purpose speaker-independent continuous speech recognition [49]. Whisper can be used for command and control, dictation, and conversational applications. Whisper offers many features such as continuous speech recognition, speaker-independence with adaptation, and dynamic vocabulary. Whisper has a unified architecture that can be scaled to meet different application and platform requirements.

The Whisper system uses MFCC representations (see Chapter 6) and both first- and second-order delta MFCC coefficients. Two-mean cepstral normalization discussed in Chapter 10 is used to eliminate channel distortion for improved robustness.

The HMM topology is a three-state left-to-right model for each phone. Senone models discussed in Section 9.4.3 are derived from both inter- and intraword context-dependent phones. The generic shared density function architecture can support either semicontinuous or continuous density hidden Markov models.

The SCHMM has a multiple-feature front end. Independent codebooks are built for the MFCC, first-order delta MFCC, second-order delta MFCC, and power and first and second power, respectively. Deleted interpolation is used to interpolate output distributions of context-dependent and context-independent senones. All codebook means and covariance matrices are reestimated together with the output distributions except the power covariance matrices, which are fixed.

The overall senone models can reduce the error rate significantly in comparison to the triphone or clustered triphone model. The shared Markov state also makes it possible to use continuous-density HMMs efficiently for large-vocabulary speech recognition. When a sufficient amount of training data becomes available, the best performance is obtained with the continuous-density mixture HMM. Each senone has 20 mixtures, albeit such an error reduction came at the cost of significantly increased computational complexity.

We can further generalize sharing to the level of each individual Gaussian probability density function. Each Gaussian function is treated as the basic unit to be shared across any Markov state. At this extreme, there is no need to use senones or shared states any more, and

the shared probability density functions become the acoustic kernels that can be used to form any mixture function for any Markov state with appropriate mixture weights. Parameter sharing is, thus, advanced from a phone unit to a Markov state unit (senones) to a density component unit.

Regarding lexicon modeling, most words have one pronunciation in the lexicon. For words that are not in the dictionary, the LTS conversion is based on the decision tree that is trained from the existing lexicon. For the purpose of efficiency, the dictionary is used to store the most frequently used words. The LTS is only used for new words that need to be added on the fly.

For speaker adaptation, the diagonal variances and means are adapted using the MAP method. Whisper also uses MLLR to modify the mean vectors only. The MLLR classes are phone dependent. The transition probabilities are context independent and they are not modified during the adaptation stage.

The language model used in Whisper can be either the trigram or the context-free grammar. The difference is largely related to the decoder algorithm, as discussed in Chapter 13. The trigram lexicon has the 60,000 most-frequent words extracted from a large text corpus. Word selection is based on both the frequency and the word's part-of-speech information. For example, verbs and the inflected forms have a higher weight than proper nouns in the selection process.

Whisper's overall word recognition error rate for speaker-independent continuous speech recognition is about 7% for the standard DARPA business-news dictation test set. For isolated dictation with similar materials, the error rate is less than 3%. If speaker-dependent data are available, it can further reduce the error rate by 15–30%, with less than 30 minutes' speech from each person. The performance can be obtained real-time on today's PC systems.

9.10. Historical Perspective and Further Reading

The first machine to recognize speech was a commercial toy named Radio Rex manufactured in the 1920s. Fueled by increased computing resources, acoustic-phonetic modeling has progressed significantly since then. Relative word error rates have been reduced by 10% every year, as illustrated in Figure 9.18, thanks to the use of HMMs, the availability of large speech and text corpora, the establishment of standards for performance evaluation, and advances in computer technology. Before the HMM was established as the standard, there were many competing techniques, which can be traced back to the 1950s. Gold and Morgan's book provides an excellent historical perspective [38].

The HMM is powerful in that, with the availability of training data, the parameters of the model can be estimated and adapted automatically to give optimal performance. There are many HMM-based state-of-the-art speech recognition systems [1, 12, 27, 34, 49, 55, 72, 73, 93, 108, 109, 112]. Alternatively, we can first identify speech segments, then classify the segments and use the segment scores to recognize words. This approach has produced competitive recognition performance that is similar to HMM-based systems in several small- to medium-vocabulary tasks [115].

Speech recognition systems attempt to model the sources of variability in several ways. At the level of signal representation, in addition to MFCC, researchers have developed representations that emphasize perceptually important speaker-independent features of the signal, and deemphasize speaker-dependent characteristics [43]. Other methods based on linear discriminant analysis to improve class separability [28, 54] and speaker normalization transformation to minimize speaker variations [51, 67, 86, 106, 107, 114] have achieved limited success. Linear discriminant analysis can be traced back to *Fisher's linear discriminant* [30], which projects a dimensional vector onto a single line that is oriented to maximize class separability. Its extension to speech recognition can be found in [65].

At the level of acoustic-phonetic modeling, we need to provide an accurate distance measure of the input feature vectors against the phonetic or word models from the signal-processing front end. Before the HMM was used, the most successful acoustic-phonetic model was based on the speech template where the feature vectors are stored as the model and dynamic-programming-based time warping was used to measure the distance between the input feature vectors and the word or phonetic templates [88, 94]. The biggest problem for template-based systems is that they are not as trainable as HMMs, since it is difficult to generate a template that is as representative as all the speech samples we have for the particular units of interest.

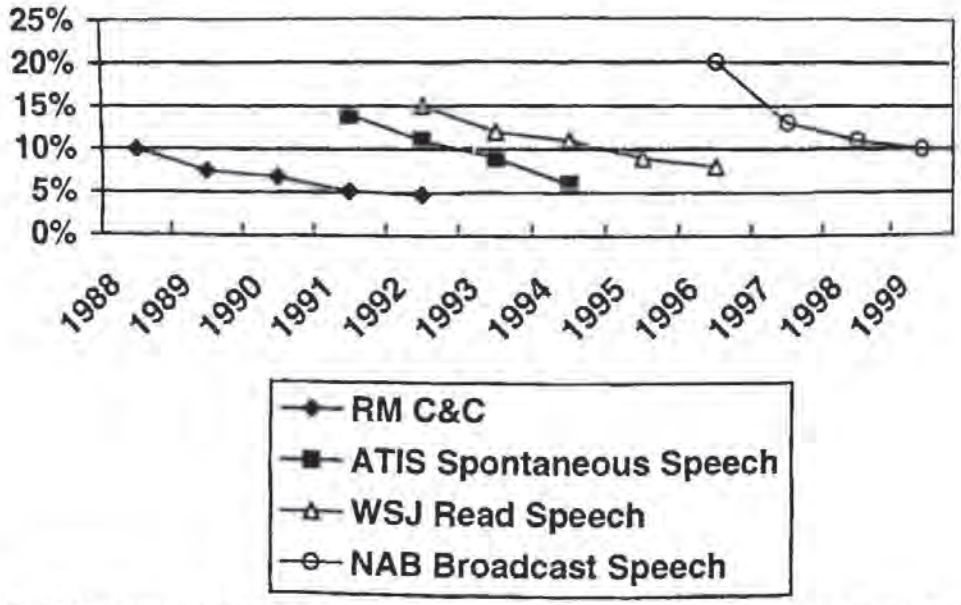


Figure 9.18 History of DARPA speech recognition word-error-rate benchmark evaluation results from 1988 to 1999. There are four major tasks: the Resource Management command and control task (RM C&C, 1000 words), the Air Travel Information System spontaneous speech understanding task (ATIS, 2000 words), the *Wall Street Journal* dictation task (WSJ, 20,000 words), and the Broadcast News Transcription Task (NAB, 60,000 words) [80-84].

Another approach that attracted many researchers is the knowledge-based one that originated from the Artificial Intelligence research community. This approach requires extensive knowledge engineering, which often led to many inconsistent rules. Due to the com-

plexity of speech recognition, rule-based approaches generally cannot match the performance of data-driven approaches such as HMMs, which can automatically extract salient rules from a large amount of training data [105].

Senones are now widely used in many state-of-the-art systems. Word models or allophone models can also be built by concatenation of basic structures made by states, transitions, and distributions such as *fenones* [8, 9] or *senones* [58].

Segment models, as proposed by Roucos and Ostendorf [79, 92], assume that each variable-length segment is mapped to a fixed number of representative frames. The resulting model is very similar to the HMM with a large number of states. Ostendorf published a comprehensive survey paper [77] on segment models. The parametric trajectory segment model was introduced by Deng et al. [25] and Gish et al. [37] independently. Gish's work is very similar to our description in Section 9.8.2.1, which is based on Hon et al. [46, 47], where the evaluation and estimation are more efficient because no individual polynomial fitting is required for likelihood computation. In addition to the phone-pair units described in this chapter, segment models have also been applied to phonetic units [25], subphonetic units [25], diphones [36], and syllables [78]. The dynamic model [24, 26] is probably the most aggressive attempt to impose a global transition constraint on the speech model. It uses the phonetic target theories on unobserved vocal-tract parameters, which are fed to an MLP to produce the observed acoustic data.

Today, it is not uncommon to have tens of thousands of sentences available for system training and testing. These corpora permit researchers to quantify the acoustic cues important for phonetic contrasts and to determine parameters of the recognizers in a statistically meaningful way. While many of these corpora were originally collected under the sponsorship of the U.S. Defense Advanced Research Projects Agency (ARPA) to spur human language technology development among its contractors [82], they have nevertheless gained international acceptance. Recognition of the need for shared resources led to the creation of the Linguistic Data Consortium (LDC)[5] in the United States in 1992 to promote and support the widespread development and sharing of resources for human language technology. The LDC supports various corpus development activities and distributes corpora obtained from a variety of sources. Currently, LDC distributes about twenty different speech corpora including those cited above, comprising many hundreds of hours of speech. The availability of a large body of data in the public domain, coupled with the specification of evaluation standards, has resulted in uniform documentation of test results, thus contributing to greater reliability in monitoring progress.

To further improve the performance of acoustic-phonetic models, we need a robust system so that performance degrades gracefully (rather than catastrophically) as conditions diverge from those under which it was trained. The best approach is likely to have systems continuously adapted to changing conditions (new speakers, microphone, task, etc.). Such adaptation can occur at many levels in systems, subword models, word pronunciations, language models, and so on. We also need to make the system portable, so that we can rapidly design, develop, and deploy systems for new applications. At present, systems tend to suffer

[5] http://www.cis.upenn.edu/ldc

significant degradation when moved to a new task. In order to retain peak performance, they must be trained on examples specific to the new task, which is time consuming and expensive. In the new task, system users may not know exactly which words are in the system vocabulary. This leads to a certain percentage of out-of-vocabulary words in natural conditions. Currently, systems lack a very robust method of detecting such out-of-vocabulary words. These words often are inaccurately mapped into the words in the system, causing unacceptable errors.

An introduction to all aspects of acoustic modeling can be found in *Spoken Dialogues with Computers* [76] and *Fundamentals of Speech Recognition* [87]. A good treatment of HMM-based speech recognition is given in [52, 60, 105]. Bourlard and Morgan's book [15] is a good introduction to speech recognition based on neural networks. There are a range of applications such as predictive networks that estimate each frame's acoustic vector, given the history [69, 104] and nonlinear transformation of observation vectors [13, 53, 101].

You can find tools to build acoustic models from Carnegie Mellon University's speech open source Web site.[6] This site contains the release of CMU's Sphinx acoustic modeling toolkit and documentation. A version of Microsoft's Whisper system can be found in the Microsoft Speech SDK.[7]

REFERENCES

[1] Abrash, V., *et al.*, "Acoustic Adaptation Using Nonlinear Transformations of HMM Parameters" in *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing* 1996, Atlanta, pp. 729-732.

[2] Acero, A., *Acoustical and Environmental Robustness in Automatic Speech Recognition*, 1993, Boston, Kluwer Academic Publishers.

[3] Ahadi-Sarkani, S.M., *Bayesian and Predictive Techniques for Speaker Adaptation*, Ph. D. Thesis, 1996, Cambridge University, .

[4] Ahn, S., S. Kang, and H. Ko, "Effective Speaker Adaptations for Speaker Verification," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 2000, Istanbul, Turkey, pp. 1081-1084.

[5] Alleva, F., *et al.*, "Can Continuous Speech Recognizers Handle Isolated Speech?," *Speech Communication*, 1998, **26**, pp. 183-189.

[6] Anastasakos, T., *et al.*, "A Compact Model for Speaker Adaptive Training," *Int. Conf. on Spoken Language Processing*, 1996, Philadelphia, pp. 1137-1140.

[7] Asadi, A., R. Schwartz, and J. Makhoul, "Automatic Modeling for Adding New Words to a Large-Vocabulary Continuous Speech Recognition System," in *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1991, Toronto, pp. 305-308.

[8] Bahl, L.R., *et al.*, "Multonic Markov Word Models for Large Vocabulary Continuous Speech Recognition," *IEEE Trans. on Speech and Audio Processing*, 1993, **1**(3), pp. 334-344.

[6] http://www.speech.cs.cmu.edu/sphinx/
[7] http://www.microsoft.com/speech

[9] Bahl, L.R., *et al.*, "A Method for the Construction of Acoustic Markov Models for Words," *IEEE Trans. on Speech and Audio Processing*, 1993, **1**(4), pp. 443-452.

[10] Bahl, L.R., *et al.*, "Automatic Phonetic Baseform Determination," *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1991, Toronto, pp. 173-176.

[11] Bahl, L.R., *et al.*, "Decision Trees for Phonological Rules in Continuous Speech," *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1991, Toronto, Canada, pp. 185-188.

[12] Bellegarda, J.R., *et al.*, "Experiments Using Data Augmentation for Speaker Adaptation," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1995, Detroit, pp. 692-695.

[13] Bengio, Y., *et al.*, "Global Optimization of a Neural Network-Hidden Markov Model Hybrid," *IEEE Trans. on Neural Networks*, 1992, **3**(2), pp. 252-259.

[14] Bourlard, H., "A New Training Algorithm for Statistical Sequence Recognition with Applications to Transition-Based Speech Recognition," *IEEE Signal Processing Letters*, 1996, **3**, pp. 203-205.

[15] Bourlard, H. and N. Morgan, *Connectionist Speech Recognition - A Hybrid Approach*, 1994, Boston, MA, Kluwer Academic Publishers.

[16] Brown, P.F., *The Acoustic-Modeling Problem in Automatic Speech Recognition*, PhD Thesis in Computer Science Department, 1987, Carnegie Mellon University, Pittsburgh, PA.

[17] Campbell, J., "Speaker Recognition: A Tutorial," *Proc. of the IEEE*, 1997, **85**(9), pp. 1437-1462.

[18] Chesta, C., O. Siohan, and C.H. Lee, "Maximum a Posteriori Linear Regression for Hidden Markov Model Adaptation," *Eurospeech*, 1999, Budapest, pp. 211-214.

[19] Chou, W., "Maximum a Posteriori Linear Regression with Elliptically Symmetric Matrix Priors," *Eurospeech*, 1999, Budapest, pp. 1-4.

[20] Cohen, M., *Phonological Structures for Speech Recognition*, Ph.D. Thesis 1989, University of California, Berkeley.

[21] Cook, G. and A. Robinson, "Transcribing Broadcast News with the 1997 Abbot System," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1998, Seattle, WA, pp. 917-920.

[22] Cox, S., "Predictive Speaker Adaptation in Speech Recognition," *Computer Speech and Language*, 1995, **9**, pp. 1-17.

[23] Davis, S. and P. Mermelstein, "Comparison of Parametric Representations for Monosyllable Word Recognition in Continuously Spoken Sentences," *IEEE Trans. on Acoustics, Speech and Signal Processing*, 1980, **28**(4), pp. 357-366.

[24] Deng, L., "A Dynamic, Feature-based Approach to the Interface Between Phonology and Phonetics for Speech Modeling and Recognition," *Speech Communication*, 1998, **24**(4), pp. 299-323.

[25] Deng, L., *et al.*, "Speech Recognition Using Hidden Markov Models with Polynomial Regression Functions as Nonstationary States," *IEEE Trans. on Speech and Audio Processing*, 1994, **2**(4), pp. 507-520.

[26] Digalakis, V., *Segment-based Stochastic Models of Spectral Dynamics for Continuous Speech Recognition*, Ph.D. Thesis in *Electrical Computer System Engineering*, 1992, Boston University.

[27] Digalakis, V. and H. Murveit, "Genones: Optimizing the Degree of Mixture Tying in a Large Vocabulary Hidden Markov Model Based Speech Recognizer" in *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1994, Adelaide, Australia, pp. 537-540.

[28] Doddington, G.R., "Phonetically Sensitive Discriminants for Improved Speech Recognition," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1989, Glasgow, Scotland, pp. 556-559.

[29] Eichner, M. and M. Wolff, "Data-Driven Generation of Pronunciation Dictionaries in the German Verbmobil Project—Discussion of Experimental Results," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 2000, Istanbul, Turkey, pp. 1687-1690.

[30] Fisher, R., "The Use of Multiple Measurements in Taxonomic Problems," *Annals of Eugenics*, 1936, 7(1), pp. 179-188.

[31] Fritsch, J. and M. Finke, "ACID/HNN: Clustering Hierarchies of Neural Networks for Context-Dependent Connectionist Acoustic Modeling," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1998, Seattle, WA, pp. 505-508.

[32] Fukunaga, K., *Introduction to Statistical Pattern Recognition*, 2nd ed, 1990, Orlando, FL, Academic Press.

[33] Gales, M., "Cluster Adaptive Training for Speech Recognition," *Int. Conf. on Spoken Language Processing*, 1998, Sydney, Australia, pp. 1783-1786.

[34] Gauvain, J.L., L. Lamel, and M. Adda-Decker, "Developments in Continuous Speech Dictation using the ARPA WSJ Task," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1995, Detroit, MI, pp. 65-68.

[35] Gauvain, J.L. and C.H. Lee, "Bayesian Learning of Gaussian Mixture Densities for Hidden Markov Models," *Proc. of the DARPA Speech and Natural Language Workshop*, 1991, Palo Alto, CA, pp. 272-277.

[36] Ghitza, O. and M.M. Sondhi, "Hidden Markov Models with Templates as Non-Stationary States: An Application to Speech Recognition," *Computer Speech and Language*, 1993, 7(2), pp. 101-120.

[37] Gish, H. and K. Ng, "A Segmental Speech Model with Applications to Word Spotting," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1993, Minneapolis, MN, pp. 447-450.

[38] Gold, B. and N. Morgan, *Speech and Audio Signal Processing: Processing and Perception of Speech and Music*, 2000, New York, John Wiley.

[39] Greenberg, S., D. Ellis, and J. Hollenback, "Insights into Spoken Language Gleaned from Phonetic Transcription of the Switchboard Corpus," *Int. Conf. on Spoken Language Processing*, 1996, Philadelphia, PA, pp. addendum 24-27.

[40] Gunawardana, A., H.W. Hon, and L. Jiang, "Word-Based Acoustic Confidence Measures for Large-Vocabulary Speech Recognition," *Int. Conf. on Spoken Language Processing*, 1998, Sydney, Australia, pp. 791-794.

[41] Haeb-Umbach, R., "Investigations on Inter-Speaker Variability in the Feature Space," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1999, Phoenix, AZ.

[42] Hennebert, J., *et al.*, "Estimation of Global Posteriors and Forward-Backward Training of Hybrid HMM/ANN Systems," *Proc. of the Eurospeech Conf.*, 1997, Rhodes, Greece, pp. 1951-1954.

[43] Hermansky, H., "Perceptual Linear Predictive (PLP) Analysis of Speech," *Journal of the Acoustical Society of America*, 1990, 87(4), pp. 1738-1752.

[44] Holmes, W. and M. Russell, "Probabilistic-Trajectory Segmental HMMs," *Computer Speech and Language*, 1999, **13**, pp. 3-37.

[45] Hon, H.W. and K.F. Lee, "CMU Robust Vocabulary-Independent Speech Recognition System," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1991, Toronto, pp. 889-892.

[46] Hon, H.W. and K. Wang, "Combining Frame and Segment Based Models for Large Vocabulary Continuous Speech Recognition," *IEEE Workshop on Automatic Speech Recognition and Understanding*, 1999, Keystone, CO.

[47] Hon, H.-W. and K. Wang, "Unified Frame and Segment Based Models for Automatic Speech Recognition," *Int. Conf. on Acoustic, Signal and Speech Processing*, 2000, Istanbul, Turkey, IEEE, pp. 1017-1020.

[48] Huang, X., *et al.*, "From Sphinx II to Whisper: Making Speech Recognition Usable" in *Automatic Speech and Speaker Recognition*, C.H. Lee, F.K. Soong, and K.K. Paliwal, eds. 1996, Norwell, MA, pp. 481-508, Kluwer Academic Publishers.

[49] Huang, X., *et al.*, "From Sphinx-II to Whisper - Make Speech Recognition Usable" in *Automatic Speech and Speaker Recognition*, C.H. Lee, F.K. Soong, and K.K. Paliwal, eds. 1996, Norwell, MA, Kluwer Academic Publishers.

[50] Huang, X. and K.-F. Lee, "On Speaker-Independent, Speaker-Dependent, and Speaker-Adaptive Speech Recognition," *IEEE Trans. on Speech and Audio Processing*, 1993, 1(2), pp. 150-157.

[51] Huang, X.D., "Speaker Normalization for Speech Recognition," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1992, San Francisco pp. 465-468.

[52] Huang, X.D., Y. Ariki, and M.A. Jack, *Hidden Markov Models for Speech Recognition*, 1990, Edinburgh, U.K., Edinburgh University Press.

[53] Huang, X.D., K. Lee, and A. Waibel, "Connectionist Speaker Normalization and its Applications to Speech Recognition," *IEEE Workshop on Neural Networks for Signal Processing*, 1991, New York, pp. 357-366.

[54] Hunt, M.J., *et al.*, "An Investigation of PLP and IMELDA Acoustic Representations and of Their Potential for Combination," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1991, Toronto, pp. 881-884.

[55] Huo, Q., C. Chan, and C.-H. Lee, "On-Line Adaptation of the SCHMM Parameters Based on the Segmental Quasi-Bayes Learning for Speech Recognition," *IEEE Trans. on Speech and Audio Processing*, 1996, 4(2), pp. 141-144.

[56] Hwang, M.Y., X. Huang, and F. Alleva, "Predicting Unseen Triphones with Senones," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1993, Minneapolis, pp. 311-314.

[57] Hwang, M.Y. and X.D. Huang, "Acoustic Classification of Phonetic Hidden Markov Models" in *Proc. of Eurospeech* 1991.

[58] Hwang, M.Y. and X.D. Huang, "Shared-Distribution Hidden Markov Models for Speech Recognition," *IEEE Trans. on Speech and Audio Processing*, 1993, 1(4), pp. 414-420.

[59] Iyer, R., *et al.*, "Hidden Markov Models for Trajectory Modeling," *Int. Conf. on Spoken Language Processing*, 1998, Sydney, Australia.

[60] Jelinek, F., *Statistical Methods for Speech Recognition*, 1998, Cambridge, MA, MIT Press.

[61] Jiang, L., H.W. Hon, and X. Huang, "Improvements on a Trainable Letter-to-Sound Converter," *Proc. of Eurospeech*, 1997, Rhodes, Greece, pp. 605-608.

[62] Jiang, L. and X. Huang, "Subword-Dependent Speaker Clustering for Improved Speech Recognition," *Int Conf. on Spoken Language Processing*, 2000, Beijing, China.

[63] Jiang, L. and X.D. Huang, "Vocabulary-Independent Word Confidence Measure Using Subword Features," *Int. Conf. on Spoken Language Processing*, 1998, Syndey, Australia.

[64] Kuhn, R., *et al.*, "Eigenvoices for Speaker Adaptation," *Int. Conf. on Spoken Language Processing*, 1998, Sydney, Australia, pp. 1771-1774.

[65] Kumar, N. and A. Andreou, "Heteroscedastic Discriminant Analysis and Reduced Rank HMMs for Improved Speech Recognition," *Speech Communication*, 1998, **26**, pp. 283-297.

[66] Lee, K.F., *Large-Vocabulary Speaker-Independent Continuous Speech Recognition: The SPHINX System*, Ph.D. Thesis in *Computer Science Dept.* 1988, Carnegie Mellon University, Pittsburgh.

[67] Lee, L. and R. Rose, "Speaker Normalization Using Efficient Frequency Warping Procedures," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1996, Atlanta, GA, pp. 353-356.

[68] Leggetter, C.J. and P.C. Woodland, "Maximum Likelihood Linear Regression for Speaker Adaptation of Continuous Density Hidden Markov Models," *Computer Speech and Language*, 1995, **9**, pp. 171-185.

[69] Levin, E., "Word Recognition Using Hidden Control Neural Architecture," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1990, Albuquerque, NM, pp. 433-436.

[70] Lippmann, R.P., E.A. Martin, and D.P. Paul, "Multi-Style Training for Robust Isolated-Word Speech Recognition," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1987, Dallas, TX, pp. 709-712.

[71] Lucassen, J.M. and R.L. Mercer, "An Information-Theoretic Approach to the Automatic Determination of Phonemic Baseforms," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1984, San Diego, pp. 42.5.1-42.5.4.

[72] Lyu, R.Y., *et al.*, "Golden Mandarin (III) - A User-Adaptive Prosodic Segment-Based Mandarin Dictation Machine for Chinese Language with Very Large Vocabulary" in *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1995, Detroit, MI, pp. 57-60.

[73] Matsui, T., T. Matsuoka, and S. Furui, "Smoothed N-best Based Speaker Adaptation for Speech Recognition" in *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1997, Munich, Germany, pp. 1015-1018.

[74] McDonough, J., *et al.*, "Speaker-Adapted Training on the Switchboard Corpus" in *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing* 1997, Munich, Germany, pp. 1059-1062.

[75] Morgan, N. and H. Bourlard, *Continuous Speech Recognition: An Introduction to Hybrid HMM/Connectionist Approach*, in *IEEE Signal Processing Magazine*, 1995, pp. 25-42.

[76] Mori, R.D., *Spoken Dialogues with Computers*, 1998, London, Academic Press.

[77] Ostendorf, M., V.V. Digalakis, and O.A. Kimball, "From HMM's to Segment Models: a Unified View of Stochastic Modeling for Speech Recognition," *IEEE Trans. on Speech and Audio Processing*, 1996, 4(5), pp. 360-378.

[78] Ostendorf, M. and K. Ross, "A Multi-Level Model for Recognition of Intonation Labels" in *Computing Prosody*, Y. Sagisaka, W.N. Campell, and N. Higuchi, eds., 1997, New York, pp. 291-308, Springer Verlag.

[79] Ostendorf, M. and S. Roukos, "A Stochastic Segment Model for Phoneme-Based Continuous Speech Recognition," *IEEE Trans. on Acoustics, Speech and Signal Processing*, 1989, 37(1), pp. 1857-1869.

[80] Pallett, D., J.G. Fiscus, and J.S. Garofolo, "DARPA Resource Management Benchmark Test Results June 1990," in *Proc. of the DARPA Speech and Natural Language Workshop* 1990, Hidden Valley, PA, pp. 298-305, Pallett.

[81] Pallett, D., J.G. Fiscus, and J.S. Garofolo, "DARPA Resource Management Benchmark Test Results," *Proc. of the DARPA Speech and Natural Language Workshop*, 1991, Morgan Kaufmann Publishers, pp. 49-58.

[82] Pallett, D.S., *et al.*, "The 1994 Benchmark Tests for the ARPA Spoken Language Program" in *Proc. of the ARPA Spoken Language Technology Workshop*, 1995, Austin, TX, pp. 5-38.

[83] Pallett, D.S., *et al.*, "1997 Broadcast News Benchmark Test Results: English and Non-English," *Proc. of the Broadcast News Transcription and Understanding Workshop*, 1998, Landsdowne, Virginia, Morgan Kaufmann Publishers.

[84] Pallett, D.S., J.G. Fiscus, and M.A. Przybocki, "1996 Preliminary Broadcast News Benchmark Tests," *Proc. of the DARPA Speech Recognition Workshop*, 1997, Chantilly, VA, Morgan Kaufmann Publishers.

[85] Pereira, F., M. Riley, and R. Sproat, "Weighted Rational Transductions and Their Application to Human Language Processing," *Proc. of the ARPA Human Language Technology Workshop*, 1994, Plainsboro, NJ, pp. 249-254.

[86] Pye, D. and P.C. Woodland, "Experiments in Speaker Normalization and Adaptation for Large Vocabulary Speech Recognition," in *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing* 1997, Munich, Germany, pp. 1047-1050.

[87] Rabiner, L.R. and B.H. Juang, *Fundamentals of Speech Recognition*, May, 1993, Prentice-Hall.

[88] Rabiner, L.R. and S.E. Levinson, "Isolated and Connected Word Recognition - Theory and Selected Applications," *IEEE Trans. on Communication*, 1981, COM-**29**(5), pp. 621-659.

[89] Riley, M. and A. Ljolje, eds. *Automatic Generation of Detailed Pronunciation Lexicons*, in Automatic Speech and Speaker Recognition, ed. C. Lee, F. Soong, and K. Paliwal, 1996, Kluwer Academic Publishers.

[90] Robinson, A., "An Application of Recurrent Nets to Phone Probability Estimation," *IEEE Trans. on Neural Networks*, 1994, **5**, pp. 298-305.

[91] Robinson, A.J., *et al.*, "A Neural Network Based, Speaker Independent, Large Vocabulary," *Proc. of the European Conf. on Speech Communication and Technology*, 1999, Berlin pp. 1941-1944.

[92] Roucos, S., *et al.*, "Stochastic Segment Modeling Using the Estimate-Maximize Algorithm," *Int. Conf. on Acoustic, Speech and Signal Processing*, 1988, New York, pp. 127-130.

[93] Sagisaka, Y. and L.S. Lee, "Speech Recognition of Asian Languages" in *Proc. IEEE Automatic Speech Recognition Workshop*, 1995, Snowbird, UT, pp. 55-57.

[94] Sakoe, H. and S. Chiba, "Dynamic Programming Algorithm Optimization for Spoken Word Recognition," *IEEE Trans. on Acoustics, Speech and Signal Processing*, 1978, **26**(1), pp. 43-49.

[95] Saraclar, M. and S. Khudanpur, "Pronunciation Ambiguity vs. Pronunciation Variability in Speech Recognition," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 2000, Istanbul, Turkey, pp. 1679-1682.

[96] Shinoda, K. and C. Lee, "Structural MAP Speaker Adaptation Using Hierarchical Priors," *IEEE Workshop on Automatic Speech Recognition and Understanding*, 1997, Santa Barbara, CA, pp. 381-388.

[97] Singh, R., B. Raj, and R. Stern, "Automatic Generation of Phone Sets and Lexical Transcriptions," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 2000, Istanbul, Turkey, pp. 1691-1694.

[98] Siohan, O., C. Chesta, and C. Lee, "Joint Maximum a Posteriori Estimation of Transformation and Hidden Markov Model Parameters," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 2000, Istanbul, Turkey, pp. 965-968.

[99] Siu, M., *et al.*, "Parametric Trajectory Mixtures for LVCSR," *Int. Conf. on Spoken Language Processing*, 1998, Sydney, Australia.

[100] Sixtus, A., *et al.*, "Recent Improvements of the RWTH Large Vocabulary Speech Recognition System on Spontaneous Speech," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 2000, Istanbul, Turkey, pp. 1671-1674.

[101] Sorenson, H., "A Cepstral Noise Reduction Multi-Layer Network," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1991, Toronto, pp. 933-936.