# Speech Signal Processing

In this chapter we treat of one of the most intricate and fascinating signals ever to be studied, human speech. The reader has already been exposed to the basic models of speech generation and perception in Chapter 11. In this chapter we apply our knowledge of these mechanisms to the practical problem of speech modeling.

Speech synthesis is the artificial generation of understandable, and (hopefully) natural-sounding speech. If coupled with a set of rules for reading text, rules that in some languages are simple but in others quite complex, we get *text-to-speech* conversion. We introduce the reader to speech modeling by means of a naive, but functional, speech synthesis system.

Speech recognition, also called *speech-to-text* conversion, seems at first to be a pattern recognition problem, but closer examination proves understanding speech to be much more complex due to time warping effects. Although a difficult task, the allure of a machine that converses with humans via natural speech is so great that much research has been and is still being devoted to this subject. There are also many other applications—speaker verification, emotional content extraction (voice polygraph), blind voice separation (cocktail party effect), speech enhancement, and language identification, to name just a few. While the list of applications is endless many of the basic principles tend to be the same. We will focus on the deriving of 'features', i.e., sets of parameters that are believed to contain the information needed for the various tasks.

Simplistic sampling and digitizing of speech requires a high information rate (in bits per second), meaning wide bandwidth and large storage requirements. More sophisticated methods have been developed that require a significantly lower information rate but introduce a tolerable amount of distortion to the original signal. These methods are called speech coding or speech compression techniques, and the main focus of this chapter is to follow the historical development of telephone-grade speech compression techniques that successively halved bit rates from 64 to below 8 Kb/s.

# **19.1 LPC Speech Synthesis**

We discussed the biology of speech production in Section 11.3, and the LPC method of finding the coefficients of an all-pole filter in Section 9.9. The time has come to put the pieces together and build a simple model that approximates that biology and can be efficiently computed. This model is often called the LPC speech model, for reasons that will become clear shortly, and is extremely popular in speech analysis and synthesis. Many of the methods used for speech compression and feature extraction are based on the LPC model and/or attempts to capture the deviations from it. Despite its popularity we must remember that the LPC speech model is an attempt to mimic the speech *production* apparatus, and does not directly relate to the way we *perceive* speech.

Recall the essential elements of the biological speech production system. For voiced speech the vocal chords produce a series of pulses at a frequency known as the pitch. This excitation enters the vocal tract, which resonates at certain frequencies known as formants, and hence amplifies the pitch harmonics that are near these frequencies. For unvoiced speech the vocal chords do not vibrate but the vocal tract remains unchanged. Since the vocal tract mainly emphasizes frequencies (we neglect zeros in the spectrum caused by the nasal tract) we can model it by an all-pole filter. The entire model system is depicted in Figure 19.1.



Figure 19.1: LPC speech model. The U/V switch selects one of two possible excitation signals, a pulse train created by the pitch generator, or white noise created by the noise generator. This excitation is input to an all-pole filter.

This extremely primitive model can already be used for speech synthesis systems, and indeed was the heart of a popular chip set as early as the 1970s. Let's assume that speech can be assumed to be approximately stationary for at least T seconds (T is usually assumed to be in the range from 10 to 100 milliseconds). Then in order to synthesize speech, we need to supply our model with the following information every T seconds. First, a single bit indicating whether the speech segment is voiced or unvoiced. If the speech is voiced we need to supply the pitch frequency as well (for convenience we sometimes combine the U/V bit with the pitch parameter, a zero pitch indicating unvoiced speech). Next, we need to specify the overall gain of the filter. Finally, we need to supply any set of parameters that completely specify the all-pole filter (e.g., pole locations, LPC coefficients, reflection coefficients, LSP frequencies). Since there are four to five formants, we expect the filter to have 8 to 10 complex poles.

How do we know what filter coefficients to use to make a desired sound? What we need to do is to prepare a list of the coefficients for the various phonemes needed. Happily this type of data is readily available. For example, in Figure 19.2 we show a scatter plot of the first two formants for vowels, based on the famous Peterson-Barney data.



Figure 19.2: First two formants from Peterson-Barney vowel data. The horizontal axis represents the frequency of the first formant between 200 and 1250 Hz, while the vertical axis is the frequency of the second formant, between 500 and 3500 Hz. The data consists of each of ten vowel sounds pronounced twice by each of 76 speakers. The two letter notations are the so-called ARPABET symbols. IY stands for the vowel in heat, IH for that in hid, and likewise EH head, AE had, AH hut, AA hot, AO fought, UH hood, UW hoot, ER heard.

Can we get a rough estimate of the information rate required to drive such a synthesis model? Taking T to be 32 milliseconds and quantizing the pitch, gain, and ten filter coefficients with eight bits apiece, we need 3 Kb/s. This may seem high compared to the information in the original text (even speaking at the rapid pace of three five-letter words per second, the text requires less than 150 b/s) but is amazingly frugal compared to the data rate required to transfer natural speech.

The LPC speech model is a gross oversimplification of the true speech production mechanism, and when used without embellishment produces synthetic sounding speech. However, by properly modulating the pitch and gain, and using models for the short time behavior of the filter coefficients, the sound can be improved somewhat.

### EXERCISES

- 19.1.1 The Peterson-Barney data is easily obtainable in computer-readable form. Generate vowels according to the formant parameters and listen to the result. Can you recognize the vowel?
- 19.1.2 Source code for the Klatt formant synthesizer is in the public domain. Learn its parameters and experiment with putting phonemes together to make words. Get the synthesizer to say 'digital signal processing'. How naturalsounding is it?
- 19.1.3 Is the LPC model valid for a flute? What model is sensible for a guitar? What is the difference between the excitation of a guitar and that of a violin?

### **19.2 LPC Speech Analysis**

The basic model of the previous section can be used for more than text-tospeech applications, and it can be used as the synthesis half of an LPC-based speech compression system. In order to build a complete compression system we need to solve the inverse problem, given samples of speech to determine whether the speech is voiced or not, if it is to find the pitch, to find the gain, and to find the filter coefficients that best match the input speech. This will allow us to build the analysis part of an LPC speech coding system.

Actually, there is a problem that should be solved even before all the above, namely deciding whether there is any speech present at all. In most conversations each conversant tends to speak only about half of the time, and there is no reason to try to model speech that doesn't exist. Simple devices that trigger on speech go under the name of VOX, for Voice Operated X (X being a graphic abbreviation for the word 'switch'), while the more sophisticated techniques are now called Voice Activity Detection. Simple VOXes may trigger just based on the appearance of energy, or may employ NRT mechanisms, or use gross spectral features to discriminate between speech and noise. The use of zero crossings is also popular as these can be computed with low complexity. Most VADs utilize parameters based on autocorrelation, and essentially perform the initial stages of a speech coder. When the decision has been made that no voice is present, older systems would simply not store or transfer any information, resulting in dead silence upon decoding. The modern approach is to extract some basic statistics of the noise (e.g., energy and bandwidth) in order to enable Comfort Noise Generation, (CNG).

Once the VAD has decided that speech is present, determination of the voicing (U/V) must be made; and assuming the speech is voiced the next step will be pitch determination. Pitch tracking and voicing determination will be treated in Section 19.5.

The finding of the filter coefficients is based on the principles of Section 9.9, but there are a few details we need to fill in. We know how to find LPC coefficients when there is no excitation, but here there *is* excitation. For voiced speech this excitation is nonzero only during the glottal pulse, and one strategy is to ignore it and live with the spikes of error. These spikes reinforce the pitch information and may be of no consequence in speech compression systems. In pitch synchronous systems we first identify the pitch pulse locations, and correctly evaluate the LPC coefficients for blocks starting with a pulse and ending before the next pulse. A more modern approach is to perform two separate LPC analyses. The one we have been discussing up to now, which models the vocal tract, is now called the short-term predictor. The new one, called the long-term predictor, estimates the pitch period and structure. It typically only has a few coefficients, but is updated at a higher rate.

There is one final parameter we have neglected until now, the gain G. Of course if we assume the excitation to be zero our formalism cannot be expected to supply G. However, since G simply controls the overall volume, it carries little information and its adjustment is not critical. In speech coding it is typically set by requiring the energy of the predicted signal to equal the energy in the original signal.

### EXERCISES

- 19.2.1 Multipulse LPC uses an excitation with several pulses per pitch period. Explain how this can improve LPC quality.
- 19.2.2 Mixed Excitation Linear Prediction (MELP) does switch between periodic and noise excitation, rather uses an additive combination of the two. Why can this produce better quality speech than LPC?
- 19.2.3 Record some speech and display its sonogram. Compute the LPC spectrum and find its major peaks. Overlay the peaks onto the sonogram. Can you recognize the formants? What about the pitch?
- 19.2.4 Synthesize some LPC data using a certain number of LPC coefficients and try to analyze it using a different number of coefficients. What happens? How does the reconstruction SNR depend on the order mismatch?

## 19.3 Cepstrum

The LPC model is not the only framework for describing speech. Although it is currently the basis for much of speech compression, cepstral coefficients have proven to be superior for speech recognition and speaker identification.

The first time you hear the word *cepstrum* you are convinced that the word was supposed to be *spectrum* and laugh at the speaker's spoonerism. However, there really is something pronounced 'cepstrum' instead of 'spectrum', as well as a 'quefrency' replacing 'frequency', and 'liftering' displacing 'filtering'. Several other purposefully distorted words have been suggested (e.g., 'alanysis' and 'saphe') but have not become as popular.

To motivate the use of cepstrum in speech analysis, recall that voiced speech can be viewed as a periodic excitation signal passed through an allpole filter. The excitation signal in the frequency domain is rich in harmonics, and can be modeled as a train of equally spaced discrete lines, separated by the pitch frequency. The amplitudes of these lines decreases rapidly with increasing frequency, with between 5 and 12 dB drop per octave being typical. The effect of the vocal tract filtering is to multiply this line spectrum by a window that has several pronounced peaks corresponding to the formants.

Now if the spectrum is the product of the pitch train and the vocal tract window, then the logarithm of this spectrum is the sum of the logarithm of the pitch train and the logarithm of the vocal tract window. This logarithmic spectrum can be considered to be the spectrum of some new signal, and since the FT is a linear operation, this new signal is the sum of two signals, one deriving from the pitch train and one from the vocal tract filter. This new signal, derived by logarithmically compressing the spectrum, is called the *cepstrum* of the original signal. It is actually a signal in the time domain, but since it is derived by distorting the frequency components its axis is referred to as *quefrency*. Remember, however, that the units of quefrency are seconds (or perhaps they should be called 'cesonds').

We see that the cepstrum decouples the excitation signal from the vocal tract filter, changing a convolution into a sum. It can achieve this decoupling not only for speech but for any excitation signal and filter, and is thus a general tool for deconvolution. It has therefore been applied to various other fields in DSP, where it is sometimes referred to as *homomorphic deconvolution*. This term originates in the idea that although the cepstrum is not a linear transform of the signal (the cepstrum of a sum is not the sum of the cepstra), it is a generalization of the idea of a linear transform (the cepstrum of the convolution is the sum of the cepstra). Such parallels are called 'homomorphisms' in algebra.

The logarithmic spectrum of the excitation signal is an equally spaced train, but the logarithmic amplitudes are much less pronounced and decrease slowly and linearly while the lines themselves are much broader. Indeed the logarithmic spectrum of the excitation looks much more like a sinusoid than a train of impulses. Thus the pitch contribution is basically a line at a well defined quefrency corresponding to the basic pitch frequency. At lower quefrencies we find structure corresponding to the higher frequency formants, and in many cases high-pass liftering can thus furnish both a voiced/unvoiced indication and a pitch frequency estimate.

Up to now our discussion has been purposefully vague, mainly because the cepstrum comes in several different flavors. One type is based on the z transform S(z), which being complex valued, is composed of its absolute value R(z) and its angle  $\theta(z)$ . Now let's take the complex logarithm of S(z)(equation (A.14)) and call the resulting function  $\check{S}(z)$ .

$$\check{S}(z) = \log S(z) = \log R(z) + i\theta(z)$$

We assumed here the minimal phase value, although for some applications it may be more useful to unwrap the phase. Now  $\check{S}(z)$  can be considered to be the zT of some signal  $\check{s}_n$ , this signal being the *complex cepstrum* of  $s_n$ . To find the complex cepstrum in practice requires computation of the izT, a computationally arduous task; however, given the complex cepstrum the original signal may be recovered via the zT. The power cepstrum, or real cepstrum, is defined as the signal whose PSD is the logarithm of the PSD of  $s_n$ . The power cepstrum can be obtained as an iFT, or for digital signals an inverse DFT

$$\check{s}_n = rac{1}{2\pi} \int_{-\pi}^{\pi} \log |S(\omega)| e^{\mathrm{i}\omega n} \, d\omega$$

and is related to the complex cepstrum.

$$\check{s}_n = \frac{1}{2}(\check{s}_n + \check{s}_{-n}^*)$$

Although easier to compute, the power cepstrum doesn't take the phase of  $S(\omega)$  into account, and hence does not enable unique recovery of the original signal.

There is another variant of importance, called the *LPC cepstrum*. The LPC cepstrum, like the reflection coefficients, area ratios, and LSP coefficients, is a set of coefficients  $c_k$  that contains exactly the same information as the LPC coefficients. The LPC cepstral coefficients are defined as the coefficients of the zT expansion of the logarithm of the all-pole system function. From the definition of the LPC coefficients in equation (9.21), we see that this can be expressed as follows:

$$\log \frac{G}{1 - \sum_{m=1}^{M} b_m z^{-m}} = \sum_k c_k z^{-k}$$
(19.1)

Given the LPC coefficients, the LPC cepstral coefficients can be computed by a recursion that can be derived by series expansion of the left-hand side (using equations (A.47) and (A.15)) and equating like terms.

$$c_{0} = \log G$$

$$c_{1} = b_{1}$$

$$c_{k} = b_{k} + \frac{1}{k} \sum_{m=1}^{k-1} m c_{m} b_{k-m}$$
(19.2)

This recursion can even be used for  $c_k$  coefficients for which k > M by taking  $b_k = 0$  for such k. Of course, the recursion only works when the original LPC model was stable.

LPC cepstral coefficients derived from this recursion only represent the true cepstrum when the signal is exactly described by an LPC model. For real speech the LPC model is only an approximation, and hence the LPC cepstrum deviates from the true cepstrum. In particular, for phonemes that are not well represented by the LPC model (e.g., sounds like **f**, **s**, and **sh** that are produced at the lips with the vocal tract trapping energy and creating zeros), the LPC cepstrum bears little relationship to its namesakes. Nonetheless, numerous comparisons have shown the LPC cepstral coefficients to be among the best features for both speech and speaker recognition.

If the LPC cepstral coefficients contain precisely the same information as the LPC coefficients, how can it be that one set is superior to the other? The difference has to do with the other mechanisms used in a recognition system. It turns out that Euclidean distance in the space of LPC cepstral coefficients correlates well with the *Itakura-Saito distance*, a measure of how close sounds actually sound. This relationship means that the interpretation of closeness in LPC cepstrum space is similar to that our own hearing system uses, a fact that aids the pattern recognition machinery.

#### EXERCISES

- 19.3.1 The signal x(t) is corrupted by a single echo to become  $y(t) = x(t) + \alpha x(t-\tau)$ . Show that the log power spectrum of y is approximately that of x with an additional ripple. Find the parameters of this ripple.
- 19.3.2 Complete the proof of equation (19.2).
- 19.3.3 The reconstruction of a signal from its power cepstrum is not *unique*. When is it *correct*?
- 19.3.4 Record some speech and plot its power cepstrum. Are the pitch and formants easily separable?
- 19.3.5 Write a program to compute the LPC cepstrum. Produce artificial speech from an exact LPC model and compute its LPC cepstrum.

### **19.4** Other Features

The coefficients we have been discussing all describe the fine structure of the speech spectrum in some way. LPC coefficients are directly related to the all-pole spectrum by equation (13.24); the LSP frequencies are themselves frequencies; and the cepstrum was derived in the previous section as a type of spectrum of (log) spectrum. Not all speech processing is based on LPC coefficients; bank-of-filter parameters, wavelets, mel- or Bark-warped spectrum, auditory nerve representations, and many more representations are also used. It is obvious that all of these are spectral descriptions. The extensive use of these parameters is a strong indication of our belief that the information in speech is stored in its spectrum, more specifically in the position of the formants.

We can test this premise by filtering some speech in such a way as to considerably whiten its spectrum for some sound or sounds. For example, we can create an inverse filter to the spectrum of a common vowel, such as the **e** in the word 'feet'. The spectrum will be completely flat when this vowel sound is spoken, and will be considerably distorted during other vowel sounds. Yet this 'inverse-E' filtered speech turns out to be perfectly intelligible. Of course a speech recognition device based on one of the aforementioned parameter sets will utterly fail.

So where is the information if not in the spectrum? A well-known fact regarding our senses is that they respond mainly to change and not to steadystate phenomena. Strong odors become unnoticeable after a short while, our eyes twitch in order to keep objects moving on our retina (animals without the eye twitch only see moving objects) and even a relatively loud stationary background noise seems to fade away. Although our speech generation system is efficient at creating formants, our hearing system is mainly sensitive to *changes* in these formants.

One way this effect can be taken into account in speech recognition systems is to use derivative coefficients. For example, in addition to using LPC cepstral coefficients as features, some systems use the so-called delta cepstral coefficients, which capture the time variation of the cepstral coefficients. Some researchers have suggested using the delta-delta coefficients as well, in order to capture second derivative effects.

An alternative to this empirical addition of time-variant information is to use a set of parameters specifically built to emphasize the signal's time variation. One such set of parameters is called RASTA-PLP (Relative Spectra— Perceptual Linear Prediction). The basic PLP technique modifies the short time spectrum by several psychophysically motivated transformations, including resampling the spectrum into Bark segments, taking the logarithm of the spectral amplitude and weighting the spectrum by a simulation of the psychophysical equal-loudness curve, before fitting to an all-pole model. The RASTA technique suppresses steady state behavior by band-pass filtering each frequency channel, in this way removing DC and slowly varying terms. It has been found that RASTA parameters are less sensitive to artifacts; for example, LPC-based speech recognition systems trained on microphonequality speech do not work well when presented with telephone speech. The performance of a RASTA-based system degrades much less. Even more radical departures from LPC-type parameters are provided by cochlear models and auditory nerve parameters. Such parameter sets attempt to duplicate actual signals present in the biological hearing system (see Section 11.4). Although there is an obvious proof that such parameters can be effectively used for tasks such as speech recognition, their success to date has not been great.

Another set of speech parameters that has been successful in varied tasks is the so-called 'sinusoidal representation'. Rather than making a U/V decision and modeling the excitation as a set of pulses, the sinusoidal representation uses a sum of L sinusoids of arbitrary amplitudes, frequencies, and phases. This simplifies computations since the effect of the linear filter on sinusoids is elementary, the main problem being matching of the models at segment boundaries. A nice feature of the sinusoidal representation is that various transformations become relatively easy to perform. For example, changing the speed of articulation without varying the pitch, or conversely varying the pitch without changing rate of articulation, are easily accomplished since the effect of speeding up or slowing down time on sinusoids is straightforward to compute.

We finish off our discussion of speech features with a question. How many features are really needed? Many speech recognition systems use ten LPC or twelve LPC cepstrum coefficients, but to these we may need to add the delta coefficients as well. Even more common is the 'play it safe' approach where large numbers of features are used, in order not to discard any possibly relevant information. Yet these large feature sets contain a large amount of redundant information, and it would be useful, both theoretically and in practice, to have a minimal set of features. Such a set *might* be useful for speech compression as well, but not necessarily. Were these features to be of large range and very sensitive, each would require a large number of bits to accurately represent, and the total number of bits needed could exceed that of traditional methods.

One way to answer the question is by empirically measuring the dimensionality of speech sounds. We won't delve too deeply into the mechanics of how this is done, but it is possible to consider each set of N consecutive samples as a vector in N-dimensional space, and observe how this N-dimensional speech vector moves. We may find that the local movement is constrained to M < N dimensions, like the movement of a dot on a piece of paper viewed at some arbitrary angle in three-dimensional space. Were this the case we would conclude that only M features are required to describe the speech signal. Of course these M features will probably not be universal, like a piece of paper that twists and curves in three-dimensional space, its directions

changing from place to place. Yet as long as the paper is not crumpled into a three-dimensional ball, its local dimensionality remains two. Performing such experiments on vowel sounds has led several researchers to conclude that three to five local features are sufficient to describe speech.

Of course this demonstration is not constructive and leaves us totally in the dark as to how to find such a small set of features. Attempts are being made to search for these features using learning algorithms and neural networks, but it is too early to hazard a guess as to success and possible impact of this line of inquiry.

### EXERCISES

- 19.4.1 Speech has an overall spectral tilt of 5 to 12 dB per octave. Remove this tilt (a pre-emphasis filter of the form  $1 0.99z^{-1}$  is often used) and listen to the speech. Is the speech intelligible? Does it sound natural?
- 19.4.2 If speech information really lies in the changes, why don't we differentiate the signal and then perform the analysis?

# **19.5** Pitch Tracking and Voicing Determination

The process of determining the pitch of a segment of voiced speech is usually called pitch *tracking*, since the determination must be updated for every segment. Pitch determination would seem to be a simple process, yet no-one has ever discovered an entirely reliable pitch tracking algorithm. Moreover, even extremely sophisticated pitch tracking algorithms do not usually suffer from minor accuracy problems; rather they tend to make gross errors, such as isolated reporting of double the pitch period. For this reason postprocessing stages are often used.

The pitch is the fundamental frequency in voiced speech, and our ears are very sensitive to pitch changes, although in nontonal languages their content is limited to prosodic information. Filtering that removes the pitch frequency itself does not strongly impair our perception of pitch, although it would thwart any pitch tracking technique that relies on finding the pitch spectral line. Also, a single speaker's pitch may vary over several octaves, for example, from 50 to 800 Hz, while low-frequency formants also occupy this range and may masquerade as pitch lines. Moreover, speech is neither periodic nor even stationary over even moderately long times, so that limiting ourselves to times during which the signal is stationary would provide unacceptably large uncertainties in the pitch determination. Hoarse and high-pitched voices are particularly difficult in this regard.

All this said, there are many pitch tracking algorithms available. One major class of algorithms is based on finding peaks in the empirical autocorrelation. A typical algorithm from this class starts by low-pass filtering the speech signal to eliminate frequency components above 800 or 900 Hz. The pitch should correspond to a peak in the autocorrelation of this signal, but there are still many peaks from which to choose. Choosing the largest peak sometimes works, but may result in a multiple of the pitch or in a formant frequency. Instead of immediately computing the autocorrelation we first center clip (see equation (8.7)) the signal, a process that tends to flatten out vocal tract autocorrelation peaks. The idea is that the formant periodicity should be riding on that of the pitch, even if its consistency results in a larger spectral peak. Accordingly, after center clipping we expect only pitch-related phenomena to remain. Of course the exact threshold for the center clipping must be properly set for this preprocessing to work, and various schemes have been developed. Most schemes first determine the highest sample in the segment and eliminate the middle third of the dynamic range. Now autocorrelation lags that correspond to valid pitch periods are computed. Once again we might naively expect the largest peak to correspond to the pitch period, but if filtering of the original signal removed or attenuated the pitch frequency this may not be the case. A better strategy is to look for consistency in the observed autocorrelation peaks, choosing a period that has the most energy in the peak and its multiples. This technique tends to work even for noisy speech, but requires postprocessing to correct random errors in isolated segments.

A variant of the autocorrelation class computes the Average Magnitude Difference Function

$$AMDF(m) = \sum_{n} |x_n - x_{n+m}|$$

(AMDF) rather than the autocorrelation. The AMDF is a nonnegative function of the lag m that returns zero only when the speech is exactly periodic. For noisy nearly periodic signals the AMDF has a strong minimum at the best matching period. The nice thing about using a minimum rather than maximum is that we needn't worry as much about the signal remaining stationary. Indeed a single pitch period should be sufficient for AMDF-based pitch determination.

Another class of pitch trackers work in the frequency domain. It may not be possible to find the pitch line itself in the speech spectrum, but finding the frequency with maximal harmonic energy is viable. This may be accomplished in practice by compressing the power spectrum by factors of two, three, and four and adding these to the original PSD. The largest peak in the resulting 'compressed spectrum' is taken to be the pitch frequency.

In Section 19.3 we mentioned the use of power cepstrum in determining the pitch. Assuming that the formant and pitch information is truly separated in the cepstral domain, the task of finding the pitch is reduced to picking the strongest peak. While this technique may give the most accurate results for clean speech, and rarely outputs double pitch, it tends to deteriorate rapidly in noise.

The determination of whether a segment of speech is voiced or not is also much more difficult than it appears. Actually, the issue needn't even be clear cut; speech experts speak of the 'degree of voicing', meaning the percentage of the excitation energy in the pitch pulses as compared to the total excitation. The MELP and Multi-Band Exitation (MBE) speech compression methods abandon the whole idea of an unambiguous U/V decision, using mixtures or *per-frequency-band* decisions respectively.

Voicing determination algorithms lie somewhere between VADs and pitch trackers. Some algorithms search separately for indications of pitch and noise excitation, declaring voiced or unvoiced when either is found, 'silence' when neither is found, and 'mixed' when both are. Other algorithms are integrated into pitch trackers, as in the case of the cepstral pitch tracker that returns 'unvoiced' when no significant cepstral peak is found.

In theory one can distinguish between voiced and unvoiced speech based on amplitude constancy. Voiced speech is only excited by the pitch pulse, and during much of the pitch period behaves as a exponentially decaying sinusoid. Unvoiced speech should look like the output of a continuously exited filter. The difference in these behaviors may be observable by taking the Hilbert transform and plotting the time evolution in the I-Q plane. Voice speech will tend to look like a spiral while unvoiced sections will appear as filled discs. For this technique to work the speech has to be relatively clean, and highly oversampled.

The degree of periodicity of a signal should be measurable as the ratio of the maximum to minimum values of the autocorrelation (or AMDF). However, in practice this parameter too is overrated. Various techniques supplement this ratio with gross spectral features, zero crossing and delta zero crossing, and many other inputs. Together these features are input to a decision mechanism that may be hard-wired logic, or a trainable classifier.

### EXERCISES

- 19.5.1 In order to minimize time spent in computation of autocorrelation lags, one can replace the center clipping operation with a three-level slicing operation that only outputs -1, 0 or +1. How does this decrease complexity? Does this operation strongly affect the performance of the algorithm?
- 19.5.2 Create a signal that is the weighted sum of a few sinusoids interrupted every now and then by short durations of white noise. You can probably easily separate the two signal types by eye in either time or frequency domains. Now do the same using any of the methods discussed above, or any algorithm of your own devising.
- 19.5.3 Repeat the previous exercise with additive noise on the sinusoids and narrow band noise instead of white noise. How much noise can your algorithm tolerate? How narrow-band can the 'unvoiced' sections be and still be identifiable? Can you do better 'by eye' than your algorithm?

## **19.6** Speech Compression

It is often necessary or desirable to compress digital signals. By compression we mean the representation of N signal values, each of which is quantized to b bits, in less than Nb bits. Two common situations that may require compression are transmission and storage. Transmission of an uncompressed digital music signal (sampled at 48 KHz, 16 bits per sample) requires at least a 768 Kb/s transmission medium, far exceeding the rates usually available for users connected via phone lines. Storage of this same signal requires almost 94 KB per second, thus gobbling up disk space at about  $5\frac{1}{2}$  MB per minute. Even limiting the bandwidth to 4 KHz (commonly done to speech in the public telephone system) and sampling at 16 bits leads to 128 Kb/s, far exceeding our ability to send this same information over the same channel using a telephony-grade modem. This would lead us to believe that digital methods are less efficient than analog ones, yet there *are* methods of digitally sending multiple conversations over a single telephone line.

Since further reduction in bandwidth or the number of quantization bits rapidly leads to severe quality degradation we must find a more sophisticated compression method. What about general-purpose data compression techniques? These may be able to contribute another factor-of-two improvement, but that is as far as they go. This is mainly because these methods are *lossless*, meaning they are required to reproduce the original bit stream without error. Extending techniques that work on general bit streams to the lossy regime is fruitless. It does not really make sense to view the speech signal as a stream of bits and to minimize the number of bit errors in the reconstructed stream. This is because some bits are more significant than others—an error in the least significant bit is of much less effect than an error in a sign bit!

It is less obvious that it is also not optimal to view the speech signal as a stream of sample values and compress it in such a fashion as to minimize the energy of error signal (reconstructed signal minus original signal). This is because two completely different signals may sound the same since hearing involves complex physiological and psychophysical processes (see Section 11.4).

For example, by delaying the speech signal by two samples, we create a new signal completely indistinguishable to the ear but with a large 'error signal'. The ear is insensitive to absolute time and thus would not be able to differentiate between these two 'different' signals. Of course simple cross correlation would home-in on the proper delay and once corrected the error would be zero again. But consider delaying the digital signal by half a sample (using an appropriate interpolation technique), producing a signal with completely distinct sample values. Once again a knowledgeable signal processor would be able to discover this subterfuge and return a very small error. Similarly, the ear is insensitive to small changes in loudness and absolute phase. However, the ear is also insensitive to more exotic transformations such as small changes in pitch, formant location, and nonlinear warping of the time axis.

Reversing our point-of-view we can say that speech-specific compression techniques work well for two related reasons. First, speech compression techniques are *lossy* (i.e., they strive to reproduce a signal that is similar but not necessarily identical to the original); significantly lower information rates can be achieved by introducing tolerable amounts of distortion. Second, once we have abandoned the ideal of precise reconstruction of the original signal, we can go a step further. The reconstructed signal needn't really be similar to the original (e.g., have minimal mean square error); it should merely *sound* similar. Since the ear is insensitive to small changes in phase, timing, and pitch, much of the information in the original signal is unimportant and needn't be encoded at all.

It was once common to differentiate between two types of speech coders. 'Waveform coders' exploit characteristics of the speech signal (e.g., energy concentration at low frequencies) to encode the speech samples in fewer bits than would be required for a completely random signal. The encoding is a lossy transformation and hence the reconstructed signal is not identical to the original one. However, the encoder algorithm is built to minimize some distortion measure, such as the squared difference between the original and reconstructed signals. 'Vocoders' utilize speech synthesis models (e.g., the speech model discussed in Section 9.9) to encode the speech signal. Such a model is capable of producing speech that sounds very similar to the speech that we desire to encode, but requires the proper parameters as a function of time. A vocoder-type algorithm attempts to find these parameters and usually results in reconstructed speech that sounds similar to the original but as a signal may look quite different. The distinction between waveform encoders and vocoders has become extremely fuzzy. For example, the distortion measure used in a waveform encoder may be perception-based and hence the reconstructed signal may be quite unlike the original. On the other hand, analysis by synthesis algorithms may find a vocoder's parameters by minimizing the squared error of the synthesized speech.

When comparing the many different speech compression methods that have been developed, there are four main parameters that should be taken into consideration, namely rate, quality, complexity, and delay. Obviously, there are trade-offs between these parameters, lowering of the bit rate requires higher computational complexity and/or lower perceived speech quality; and constraining the algorithm's delay while maintaining quality results in a considerable increase in complexity. For particular applications there may be further parameters of interest (e.g., the effect of background noise, degradation in the presence of bit errors).

The perceived quality of a speech signal involves not only how understandable it is, but other more elusive qualities such as how natural sounding the speech seems and how much of the speaker's identity is preserved. It is not surprising that the most reliable and widely accepted measures of speech quality involve humans listening rather than pure signal analysis. In order to minimize the bias of a single listener, a psychophysical measure of speech quality called the Mean Opinion Score (MOS) has been developed. It is determined by having a group of seasoned listeners listen to the speech in question. Each listener gives it an opinion score: 1 for 'bad' (not understandable), 2 for 'poor' (understandable only with considerable effort), 3 for 'fair' (understandable with moderate effort), 4 for 'good' (understandable with no apparent effort), and 5 for 'excellent'. The mean score of all the listeners is the MOS. A complete description of the experimental procedure is given in ITU-T standard P.830.

Speech heard directly from the speaker in a quiet room will receive a MOS ranking of 5.0, while good 4 KHz telephone-quality speech (termed

toll quality) is ranked 4.0. To the uninitiated telephone speech may seem almost the same as high-quality speech, however, this is in large part due to the brain compensating for the degradation in quality. In fact different phonemes may become acoustically indistinguishable after the band-pass filtering to 4 KHz (e.g. s and f), but this fact often goes unnoticed, just as the 'blind spots' in our eyes do. MOS ratings from 3.5 to 4 are sometimes called 'communications quality', and although lower than toll quality are acceptable for many applications.

Usually MOS tests are performed along with calibration runs of known MOS, but there still are consistent discrepancies between the various laboratories that perform these measurements. The effort and expense required to obtain an MOS rating for a coder are so great that objective tests that correlate well with empirical MOS ratings have been developed. Perceptual Speech Quality Measure (PSQM) and Perceptual Evaluation of Speech Quality (PESQ) are two such which have been standardized by the ITU.

### EXERCISES

- 19.6.1 Why can't general-purpose data compression techniques be lossy?
- 19.6.2 Assume a language with 64 different phonemes that can be spoken at the rate of eight phonemes per second. What is the minimal bit rate required?
- 19.6.3 Try to compress a speech file with a general-purpose lossless data (file) compression program. What compression ratio do you get?
- 19.6.4 Several lossy speech compression algorithms are readily available or in the public domain (e.g., LPC-10e, CELP, GSM full-rate). Compress a file of speech using one or more of these compressions. Now listen to the 'before' and 'after' files. Can you tell which is which? What artifacts are most noticeable in the compressed file? What happens when you compress a file that had been decompressed from a previous compression?
- 19.6.5 What happens when the input to a speech compression algorithm is not speech? Try single tones or DTMF tones. Try music. What about 'babble noise' (multiple background voices)?
- 19.6.6 Corrupt a file of linear 16-bit speech by randomly flipping a small percentage of the bits. What percentage is not noticed? What percentage is acceptable? Repeat the experiment by corrupting a file of compressed speech. What can you conclude about media for transmitting compressed speech?

# 19.7 PCM

In order to record and/or process speech digitally one needs first to acquire it by an A/D. The digital signal obtained in this fashion is usually called 'linear PCM' (recall the definition of PCM from Section 2.7). Speech contains significant frequency components up to about 20 KHz, and Nyquist would thus require a 40 KHz or higher sampling rate. From experimentation at that rate with various numbers of sample levels one can easily become convinced that using less than 12 to 14 bits per sample noticeably degrades the signal. Eight bits definitely delivers inferior quality, and since conventional hardware works in multiples of 8-bit bytes, we usually digitize speech using 16 bits per sample. Hence the simplistic approach to capturing speech digitally would be to sample at 40 KHz using 16 bits per sample for a total information rate of 640 Kb/s. Assuming a properly designed microphone, speaker, A/D, D/A, and filters, 640 Kb/s digital speech is indeed close to being indistinguishable from the original.

Our first step in reducing this bit rate is to sacrifice bandwidth by lowpass filtering the speech to 4 KHz, the bandwidth of a telephone channel. Although 4 KHz is not high fidelity it is sufficient to carry highly intelligible speech. At 4 KHz the Nyquist sampling rate is reduced to 8000 samples per second, or 128 Kb/s.

From now on we will use more and more specific features of the speech signal to further reduce the information rate. The first step exploits the psychophysical laws of Weber and Fechner (see Section 11.2). We stated above that 8 bits were not sufficient for proper digitizing of speech. What we really meant is that 256 equally spaced quantization levels produces speech of low perceived quality. Our perception of acoustic amplitude is, however, logarithmic, with small changes at lower amplitudes more consequential than equal changes at high amplitudes. It is thus sensible to try unevenly spaced quantization levels, with high density of levels at low amplitudes and much fewer levels used at high amplitudes. The optimal spacing function will be logarithmic, as depicted in Figure 19.3 (which replaces Figure 2.25 for this case). Using logarithmically spaced levels 8 bits is indeed adequate for toll quality speech, and since we now use only 8000 eight-bit samples per second, our new rate is 64 Kb/s, half that of linear PCM. In order for a speech compression scheme to be used in a communications system the sender and receiver, who may be using completely different equipment, must agree as to its details. For this reason precise standards must be established that ensure that different implementations can interoperate. The ITU has defined a number of speech compression schemes. The G.711 standard defines two



Figure 19.3: Quantization noise created by logarithmically digitizing an analog signal. In (A) we see the output of the logarithmic digitizer as a function of its input. In (B) the noise is the rounding error, (i.e., the output minus the input).

options for logarithmic quantization, known as  $\mu$ -law (pronounced mu-law) and A-law PCM respectively. Unqualified use of the term 'PCM' in the context of speech often refers to either of the options of this standard.

 $\mu$ -law is used in the North American digital telephone system, while Alaw serves the rest of the world. Both  $\mu$ -law and A-law are based on rational approximations to the logarithmic response of Figure 19.3, the idea being to minimize the computational complexity of the conversions from linear to logarithmic PCM and back.  $\mu$ -law is defined as

$$\check{s} = \operatorname{sgn}(s)\,\check{s}_{max}\,\frac{1+\mu\frac{|s|}{s_{max}}}{1+\frac{|s|}{s_{max}}}$$
(19.3)

where  $s_{max}$  is the largest value the signal may attain,  $\check{s}_{max}$  is the largest value we wish the compressed signal to attain, and  $\mu$  is a parameter that determines the nonlinearity of the transformation. The use of the absolute value and the sgn function allow a single expression to be utilized for both positive and negative x. Obviously,  $\mu = 1$  forces  $\check{s} = x$  while larger  $\mu$  causes the output to be larger than the input for small input values, but much smaller for large s. In this way small values of s are emphasized before quantization at the expense of large values. The actual telephony standard uses  $\mu = 255$  and further reduces computation by approximating the above expression using 16 staircase segments, eight for positive signal values and eight for negative. Each speech sample is encoded as a sign bit, three segment bits and four bits representing the position on the line segment. The theoretical A-law expression is given by

$$\check{s} = \operatorname{sgn}(s) \check{s}_{max} \begin{cases} \frac{A \frac{|s|}{s_{max}}}{1 + \ln(A)} & 0 < \frac{|s|}{s_{max}} < \frac{1}{A} \\ \frac{1 + \ln(A \frac{|s|}{s_{max}})}{1 + \ln(A)} & \frac{1}{A} < \frac{|s|}{s_{max}} < 1 \end{cases}$$
(19.4)

and although it is hard to see this from the expression, its behavior is very similar to that of  $\mu$ -law. By convention we take A = 87.56 and as in the  $\mu$ -law case approximate the true form with 16 staircase line segments. It is interesting that the A-law staircase has a rising segment at the origin and thus fluctuates for near-zero inputs, while the approximated  $\mu$ -law has a horizontal segment at the origin and is thus relatively constant for very small inputs.

### EXERCISES

- 19.7.1 Even 640 Kb/s does not capture the entire experience of listening to a speaker in the same room, since lip motion, facial expressions, hand gestures, and other body language are not recorded. How important is such auxiliary information? When do you expect this information to be most relevant? Estimate the information rates of these other signals.
- 19.7.2 Explain the general form of  $\mu$  and A laws. Start with general logarithmic compression, extend it to handle negative signal values, and finally force it to go through the origin.
- 19.7.3 Test the difference between high-quality and toll-quality speech by performing a *rhyme test*. In a rhyme test one person speaks out-of-context words and a second records what was heard. By using carefully chosen words, such as lift-list, lore-more-nor, jeep-cheep, etc., you should be able to both estimate the difference in accuracy between the two cases and determine which phonemes are being confused in the toll-quality case.
- 19.7.4 What does  $\mu$ -law (equation (19.3)) return for zero input? For maximal input? When does y = x? Plot  $\mu$ -law for 16-bit linear PCM, taking  $x_{max} = 2^{15} = 32768$ , for various  $\mu$  from 1 to 255. What is the qualitative difference between the small and large  $\mu$  cases?
- 19.7.5 Plot the  $\mu$ -law (with  $\mu = 255$ ) and A-law (with A = 87.56) responses on the same axes. By how much do they differ? Plot them together with true logarithmic response. How much error do they introduce? Research and plot the 16 line segment approximations. How much further error is introduced?

# 19.8 DPCM, DM, and ADPCM

The next factor-of-two reduction in information rate exploits the fact that long time averaged spectrum of speech does not look like white noise filtered to 4 KHz. In fact the spectrum is decidedly low-pass in character, due to voiced speech having pitch harmonics that decrease in amplitude as the frequency increases (see Section 11.3).

In Section 9.8 we studied the connection between correlation and prediction, here we wish to stress the connection between prediction and compression. Deterministic signals are completely predictable and thus maximally compressible; knowing the signal's description, (e.g., as a explicit formula or difference equation with given initial conditions) enables one to precisely predict any signal value without any further information required. White noise is completely unpredictable; even given the entire history from the beginning of time to now does not enable us to predict the next signal value with accuracy any better than random guessing. Hence pure white noise is incompressible; we can do no better than to treat each sample separately, and N samples quantized to b bits each will always require Nb bits.

Most signals encountered in practice are somewhere in between; based on observation of the signal we can construct a model that captures the predictable (and thus compressible) component. Using this model we can predict the next value, and then we need only store or transmit the residual error. The more accurate our prediction is, the smaller the error signal will be, and the fewer bits will be needed to represent it. For signals with most of their energy at low frequencies this predictability is especially simple in nature—the next sample will tend to be close to the present sample. Hence the difference between successive sample values tends to be smaller than the sample values themselves. Thus encoding these differences, a technique known as Ddelta-PCM (DPCM), will usually require fewer bits. This same term has come to be used in a more general way to mean encoding the difference between the sample value and a predicted version of it.

To see how this generalized DPCM works, let's use the previous value  $s_{n-1}$ , or the previous N values  $s_{n-N} \ldots s_{n-1}$ , to predict the signal value at time n.

$$\tilde{s_n} = p(s_{n-1}, s_{n-2}, \dots s_{n-N})$$
 (19.5)

If the predictor function p is a filter

$$\tilde{s_n} = \sum_{i=1}^{N} p_i s_{n-i}$$
(19.6)



Figure 19.4: Unquantized DPCM. The encoder predicts the next value, finds the prediction error  $\epsilon_n = s_n - \tilde{s}_n$ , and transmits this error through the communications channel to the receiver. The receiver, imitating the transmitter, predicts the next value based on all the values it has recovered so far. It then corrects this prediction based on the error  $\epsilon_n$  received.

we call the predictor a *linear predictor*. If the predictor works well, the prediction error

$$\epsilon_n = s_n - \tilde{s}_n \tag{19.7}$$

is both of lower energy and much whiter than the original signal  $s_n$ . The error is all we need to transmit for the receiver to be able to reconstruct the signal, since it too can predict the next signal value based on the past values. Of course this prediction  $\tilde{s}_n$  is not completely accurate, but the correction  $\epsilon_n$  is received, and the original value easily recovered by  $s_n = \tilde{s}_n + \epsilon_n$ . The entire system is depicted in Figure 19.4. We see that the encoder (linear predictor) is present in the decoder, but that there it runs as feedback, rather than feedforward as in the encoder itself.

The simplest DPCM system is Delta Modulation (DM). Delta modulation uses only a single bit to encode the error, this bit signifying whether the true value is above or below the predicted one. If the sampling frequency is so much higher than required that the previous value  $s_{n-1}$  itself is a good predictor of  $s_n$ , delta modulation becomes the sigma-delta converter of Section 2.11. In a more general setting a nontrivial predictor is used, but we still encode only the sign of the prediction error. Since delta modulation provides no option to encode zero prediction error the decoded signal tends to oscillate up and down where the original was relatively constant. This annoyance can be ameliorated by the use a *post-filter*, which low-pass filters the reconstructed signal.

There is a fundamental problem with the DPCM encoders we have just described. We assumed that the true value of the prediction error  $\epsilon_n$  is transferred over the channel, while in fact we can only transfer a quantized version  $\epsilon_n^Q$ . The very reason we perform the prediction is to save bits after quantization. Unfortunately, this quantization may have a devastating effect

on the decoder. The problem is not just that the correction of the present prediction is not completely accurate; the real problem is that because of this inaccuracy the receiver never has reliable  $s_n$  with which to continue predicting the next samples. To see this, define  $s_n^{Q}$  as the decoder's predicted value corrected by the quantized error. In general,  $s_n^{Q}$  does not quite equal  $s_n$ , but we predict the next sample values based on these incorrect corrected predictions! Due to the feedback nature of the decoder's predictor the errors start piling up and after a short time the encoder and decoder become 'out of sync'.

The prediction we have been using is known as *open-loop* prediction, by which we mean that we perform linear prediction of the input speech. In order to ensure that the encoder and decoder predictors stay in sync, we really should perform linear prediction on the speech as reconstructed by the decoder. Unfortunately, the decoder output is not available at the encoder, and so we need to calculate it. To perform *closed-loop* prediction we build an exact copy of the entire decoder into our encoder, and use its output, rather than the input speech, as input to the predictor. This process is diagrammed in Figure 19.5. By 'closing the loop' in this fashion, the decoded speech is precisely that expected, unless the channel introduces bit errors.

The international standard for 32 Kb/s toll quality digital speech is based on Adaptive Delta-PCM (ADPCM). The 'adaptive' is best explained by returning to the simple case of delta modulation. We saw above that the DM *encoder* compares the speech signal value with the predicted (or simply previous) value and reports whether this prediction is too high or too low. How does a DM *decoder* work? For each input bit it takes its present estimate for the speech signal value and either adds or subtracts some step size  $\Delta$ . Assuming  $\Delta$  is properly chosen this strategy works well for some range of input signal frequencies; but as seen in Figure 19.6 using a single step



Figure 19.5: Closed-loop prediction. In this figure, Q stands for quantizer, IQ inverse quantizer, PF prediction filter. Note that the encoder contains an exact replica of the decoder and predicts the next value based on the reconstructed speech.



Figure 19.6: The two types of errors in nonadaptive delta modulation. We superpose the reconstructed signal on the original. If the step size is too small the reconstructed signal can't keep up in areas of large slope and may even completely miss peaks (as in the higher-frequency area at the beginning of the figure). If the step size is too large the reconstructed signal will oscillate wildly in areas where the signal is relatively constant (as seen at the peaks of the lower-frequency area toward the end of the figure).

size cannot satisfy all frequencies. If  $\Delta$  is too small the reconstructed signal cannot keep up when the signal changes rapidly in one direction and may even completely miss peaks (as in the higher-frequency area at the beginning of the figure), a phenomenon called 'slope overload'. If  $\Delta$  is too large the reconstructed signal will oscillate wildly when the signal is relatively constant (as seen at the peaks of the lower-frequency area toward the end of the figure), which is known as 'granular noise'.

While we introduced the errors introduced by improper step size for DM, the same phenomena occur for general DPCM. In fact the problem is even worse. For DM the step size  $\Delta$  is only used at the decoder, since the encoder only needs to check the sign of the difference between the signal value and its prediction. For general delta-PCM the step size is needed at the encoder as well, since the difference must be quantized using levels spaced  $\Delta$  apart. Improper setting of the spacing between the quantization levels causes mismatch between the digitizer and the difference signal's dynamic range, leading to improper quantization (see Section 2.9).

The solution is to adapt the step size to match the signal's behavior. In order to minimize the error we increase  $\Delta$  when the signal is rapidly increasing or decreasing, and we decrease it when the signal is more constant. A simplistic way to implement this idea for DM is to use the bit stream itself to determine whether the step size is too small or too large. A commonly used version uses memory of the previous delta bit; if the present bit is the same as the previous we multiply  $\Delta$  by some constant K (K = 1.5 is a common choice), while if the bits differ we divide by K. In addition we constrain  $\Delta$  to remain within some prespecified range, and so stop adapting when it reaches its minimum or maximum value.

While efficient computationally, the above method for adapting  $\Delta$  is completely heuristic. A more general tactic is to set the step size for adaptive DPCM to be a given percentage of the signal's standard deviation. In this way  $\Delta$  would be small for signals that do vary much, minimizing granular noise, but large for wildly varying signals, minimizing slope overload. Were speech stationary over long times adaptation would not be needed, but since the statistics of the speech signal vary widely as the phonemes change, we need to continuously update our estimate of its variance. This can be accomplished by collecting N samples of the input speech signal in a buffer, computing the standard deviation, setting  $\Delta$  accordingly, and only then performing the quantization. N needs to be long enough for the variance computation to be accurate, but not so long that the signal statistics vary appreciably over the buffer. Values of 128 (corresponding to 16 milliseconds of speech at 8000 Hz) through 512 (64 milliseconds) are commonly used.

There are two drawbacks to this method of adaptively setting the scale of the quantizer. First, the collecting of N samples before quantization requires introducing buffer delay; in order to avoid excessive delay we can use an IIR filter to track the variance instead of computing it in a buffer. Second, the decoder needs to know  $\Delta$ , and so it must be sent as *side information*, increasing the amount of data transferred. The overhead can be avoided by having the decoder derive  $\Delta$ , but if  $\Delta$  is derived from the input signal, this is not possible. The decoder could try to use the reconstructed speech to find  $\Delta$ , but this would not exactly match the quantization step used by the encoder. After a while the encoder and decoder would no longer agree and the system would break down. As you may have guessed, the solution is to close the loop and have the encoder determine  $\Delta$  using its internal decoder, a technique called *backward adaptation*.

#### EXERCISES

- 19.8.1 Obtain a copy of the G.726 ADPCM standard and study the main block diagrams for the encoder and decoder. Explain the function and connections of the adaptive predictor, adaptive quantizer, and inverse adaptive quantizer. Why is the standard so detailed?
- 19.8.2 Now study the expanded block diagram of the encoder. What is the purpose of the blocks marked 'adaptation speed control' and 'tone and transition detector'?
- 19.8.3 How does the MIPS complexity of the G.726 encoder compare with that of modern lower-rate encoders?
- 19.8.4 Show that the open-loop prediction results in large error because the quantization error is multiplied by the prediction gain. Show that with closed-loop prediction this does not occur.

### **19.9** Vector Quantization

For white noise we can do no better than to quantize each sample separately, but for other signals it may make sense to quantize groups of samples together. This is called Vector Quantization (VQ).

Before discussing *vector* quantization it is worthwhile to reflect on what we have accomplished so far in *scalar* quantization. The digitization of the A/D converters discussed in Section 2.7 was input independent and uniform. By this we mean that the positions of the quantization levels were preset and equidistant. In order to minimize the quantization noise we usually provide an amplifier that matches the analog signal to the predetermined dynamic range of the digitizer. A more sophisticated approach is to set the digitizer levels to match the signal, placing the levels close together for small amplitude signals, and further apart for stronger signals. When the range of the signal does not vary with time and is known ahead of time, it is enough to set this spacing once; but if the signal changes substantially with time we need to adapt the level spacing according to the signal. This leads to adaptive PCM, similar to but simpler than the ADPCM we studied in Section 19.8.

With adaptive PCM the quantization levels are not preset, but they are still equidistant. A more sophisticated technique is nonuniform quantization, such as the logarithmic PCM of Section 19.7. The idea behind logarithmic PCM was that low levels are more prevalent and their precision perceptually more important than higher ones; thus we can reduce the average (perceptual) error by placing the quantization levels closer together for small signal values, and further apart for large values.

We will return to the perceptual importance later; for now we assume all signal values to be equally important and just ask how to combine adaptivity with nonequidistant quantization thresholds. Our objective is to lower the average quantization error; and this can be accomplished by placing the levels closer together where the signal values are more probable.

Rather than adapting quantization thresholds, we can adapt the midpoints between these thresholds. We call these midpoints 'centers', and the quantization thresholds are now midway between adjacent centers. It is then obvious that classifying an input as belonging to the nearest 'center' is equivalent to quantizing according to these thresholds. The set of all values that are classified as closest to a given center (i.e., that lie between the two thresholds) is called its 'cluster'.

The reason we prefer to set centers is that there is an easily defined criterion that differentiates between good sets of centers and poor ones, namely mean square error (MSE). Accordingly, if we have observed N signal values  $\{x_n\}_{n=1}^N$ , we want to place M centers  $\{c_m\}_{m=1}^M$  in such a way that we minimize the mean square quantization error.

$$E = \frac{1}{N} \sum_{n=1}^{N} |x_n - c_n|^2$$
(19.8)

We have used here the short-hand notation  $c_n$  to mean that center closest to  $x_n$ .

Algorithms that perform this minimization given empirical data are called 'clustering' algorithms. In a moment we will present the simplest of these algorithms, but even it already contains many of the elements of the most complex of these algorithms.

There is another nomenclature worth introducing. Rather than thinking of minimal error clustering we can think of quantization as a form of *encoding*, whereby a real signal value is encoded by the index of the interval to which it belongs. When decoding, the index is replaced by the center's value, introducing a certain amount of error. Because of this perspective the center is usually called a *codeword* and the set of M centers  $\{c_j\}_{j=1}^M$  the *codebook*.

How do we find the codebook given empirical data? Our algorithm will be iterative. We first randomly place the M centers, and then move them in such a way that the average coding error is decreased. We continue to iterate until no further decrease in error is possible. The question that remains is how to move the centers in order to reduce the average error.



Figure 19.7: Quantization thresholds found by the scalar quantization algorithm for uniform and Gaussian distributed data. For both cases 1000 points were generated, and 16 centers found by running the basic scalar quantization algorithm until convergence.

Were we to know which inputs *should* belong to a certain cluster, then minimizing the sum of the squared errors would require positioning the center at the average of these input values. The idea behind the algorithm is to exploit this fact at each iteration. At each stage there is a particular set of M centers that has been found. The best guess for associating signal values to cluster centers is to classify each observed signal value as belonging to the closest center. For this set of classifications we can then position the centers optimally at the average. In general this correction of center positions will change the classifications, and thus we need to reclassify the signal values and recompute the averages. Our iterative algorithm for scalar quantization is therefore the following.

```
Given: signal values \{x_i\}_{i=1}^N,

the desired codebook size M

Initialize: randomly choose M cluster centers \{c_j\}_{j=1}^M

Loop:

Classification step:

for i \leftarrow 1 \dots N

for j \leftarrow 1 \dots M

compute d_{ij}^2 = |x_i - c_j|^2

classify x_i as belonging to C_j with minimal d_{ij}^2

Expectation step:

for j \leftarrow 1 \dots M

correct center c_j \leftarrow \frac{1}{N_i} \sum_{i \in C_j} x_i
```

Here  $N_j$  stands for the number of  $x_i$  that were classified as belonging to cluster  $C_j$ . If  $N_j = 0$  then no values are assigned to center j and we discard it.

Note that there are two steps in the loop, a *classification* step where we find the closest center  $c_n$ , and an *expectation* step where we compute the average of all values belonging to each center  $c_m$  and reposition it. We thus say

that this algorithm is in the class of *expectation-classification* algorithms. In the pattern recognition literature this algorithm is called 'k-means', while in speech coding it is called the LBG algorithm (after Linde, Buzo, and Gray). An example of two runs of LBG on scalar data is presented in Figure 19.7.

We now return to vector quantization. The problem is the same, only now we have N input vectors in D-dimensional space, and we are interested in placing M centers in such fashion that mean encoding error is minimized. The thresholds are more complex now, the clusters defining *Voronoy regions*, but precisely the same algorithm can be used. All that has to be done is to interpret the calculations as vector operations.

Now that we know how to perform VQ what do we do with it? It turns out *not* to be efficient to directly VQ blocks of speech samples, but sets of LPC coefficients (or any of the other alternative features) and the LPC residual can be successfully compressed using VQ. Not only does VQ encoding of speech parameters provide a compact representation for speech compression, it is also widely used in speech recognition.

#### EXERCISES

- 19.9.1 Prove the point closest to all points in a cluster is their average.
- 19.9.2 Generate bimodal random numbers, i.e., ones with a distribution with two separated peaks. Determine the error for the best standard quantization. Now run the LBG algorithm with the same number of levels and check the error again. How much improvement did you get?
- 19.9.3 Generate random vectors that are distributed according to a 'Gaussian mixture' distribution. This is done as follows. Choose *M* cluster centers in *N*dimensional space. For each number to be generated randomly select the cluster, and then add to it Gaussian noise (if the noise has the same variance for all elements then the clusters will be hyperspherical). Now run the LBG algorithm. Change the size of the codebook. How does the error decrease with codebook size?

### 19.10 SBC

The next factor-of-two can be achieved by noticing that the short time spectrum tends to have a only a few areas with significant energy. The SubBand Coding (SBC) technique takes advantage of this feature by dividing the spectrum into a number (typically 8 to 16) of subbands. Each subband signal, created by QMF band-pass filtering, is encoded separately. This in itself would not conserve bits, but adaptively deciding on the number of bits (if any) that should be devoted to each subband, does.

Typical SBC coders of this type divide the bandwidth from DC to 4 KHz into 16 bands of 250 Hz each, and often discard the lowest and highest of these, bands that carry little speech information. Each of the remaining subbands is decimated by a factor of 16, and divided into time segments, with 32 milliseconds a typical choice. 32 milliseconds corresponds to 256 samples of the original signal, but only 16 samples for each of the decimated subbands. In order to encode at 16 Kb/s the output of all the subbands together cannot exceed 512 bits, or an average of 32 bits per subband (assuming 16 subbands). Since we might be using only 14 subbands, and furthermore subbands with low energy may be discarded with little effect on the quality, the number of bits may be somewhat larger; but the bit allocation table and overall gain (usually separately encoded) also require bits. So the task is now to encode 16 decimated samples in about 32 bits.

After discarding the low-energy subbands the remaining ones are sorted in order of dynamic range and available bits awarded accordingly. Subbands with relatively constant signals can be replaced by scalar-quantized averages, while for more complex subbands vector quantization is commonly employed.

An alternative to equal division of the bandwidth is hierarchical logarithmic division, as described in Section 13.9. This division is both more efficient to compute (using the pyramid algorithm) and perceptually well motivated.

### EXERCISES

- 19.10.1 Can we always decimate subbands according to their bandwidth? (Hint: Recall the 'band-pass sampling theorem'.)
- 19.10.2 When dividing into equal-bandwidth bands, in which are more bits typically needed, those with lower or higher frequencies? Is this consistent with what happens with logarithmic division?
- 19.10.3 Will dividing the bandwidth into arbitrary bands adaptively matched to the signal produce better compression?

769

# 19.11 LPC Speech Compression

We now return to the LPC speech analysis and synthesis methods of sections 19.1 and 19.2 and discuss 'U.S. Standard 1015' more commonly known as LPC-10e. This standard compresses 8000 sample-per-second speech to 2.4 Kb/s using 10 LPC coefficients (hence its name).

LPC-10 starts by dividing the speech into 180-sample blocks, each of which will be converted into 53 bits to which one synchronization bit is added for a total of 54 bits. The 54 bits times 8000/180 results in precisely 2400 b/s. The U/V decision and pitch determination is performed using an AMDF technique and encoded in 7 bits. The gain is measured and quantized to 5 bits and then the block is normalized. If you have been counting, 41 bits are left to specify the LPC filter. LPC analysis is performed using the covariance method and ten reflection coefficients are derived. The first two are converted to log area ratios and all are quantized with between 3 and 6 bits per coefficient. Actually by augmenting LPC-10 with vector quantization we can coax down the data rate to less than 1 Kb/s.

Unfortunately, although highly compressed, straight LPC-encoded speech is of rather poor quality. The speech sounds synthetic and much of the speaker information is lost. The obvious remedy in such cases is to compute and send the error signal as well. In order to do this we need to add the complete decoder to the encoder, and require it to subtract the reconstructed signal from the original speech and to send the error signal through the channel. At the decoder side the process would then be to reconstruct the LPC-encoded signal and then to add back the error signal to obtain the original speech signal.

The problem with the above idea is that in general such error signals, sampled at the original sampling rate (8 KHz) may require the same number of bits to encode as the original speech. We can only gain if the error signal is itself significantly compressible. This was the idea we used in ADPCM where the difference (error) signal was of lower dynamic range than the original speech. The LPC error signal is definitely somewhat smaller than the original speech, but that is no longer enough. We have already used up quite a few bits per second on the LPC coefficients, and we need the error signal to be either an order-of-magnitude smaller or highly correlated in the time domain for sufficient compression to be possible.

Observing typical error signals is enlightening. The error is indeed smaller in magnitude than the speech signal, but not by an order-of-magnitude. It also has a very noticeable periodic component. This periodicity is at the pitch frequency and is due to the LPC analysis only being carried out for times longer than those of the pitch frequency. Our assumption that the pitch excitation could be modeled as a single pulse per pitch period and otherwise zero has apparently been pushed beyond its limits. If we remove the residual pitch period correlations the remaining error seems to be white noise. Hence, trying to efficiently compress the error signal would seem to be a useless exercise.

### EXERCISES

- 19.11.1 You can find code for LPC-10e in the public domain. Encode and then decode some recorded speech. How do you rate the quality? Can you always understand what is being said? Can you identify the speaker? Are some speakers consistently hard to understand?
- 19.11.2 In Residual Excited Linear Prediction (RELP) the residual is low-pass filtered to about 1 KHz and then decimated to lower its bit rate. Diagram the RELP encoder and decoder. For what bit rates do you expect RELP to function well?

# 19.12 CELP Coders

In the last section we saw that straight LPC using a single pulse per pitch period is an oversimplification. Rather than trying to encode the error signal, we can try to find an excitation signal that reduces the residual error. If this excitation can be efficiently encoded and transmitted, the decoder will be able to excite the remote predictor with it and reproduce the original speech to higher accuracy with tolerable increase in bit rate.

There are several different ways to encode the excitation. The most naive technique uses random codebooks. Here we can create, using VQ, a limited number  $2^m$  of random N-vectors that are as evenly distributed in N-dimensional space as possible. These vectors are known both to the encoder and to the decoder. After performing LPC analysis, we try each of these random excitations, and choose the one that produces the lowest prediction error. Since there are  $2^m$  possible excitations, sending the index of the best excitation requires only m bits. Surprisingly, this simple technique already provides a significant improvement in quality as compared to LPC, with only a modest increase in bit rate. The problem, of course, is the need to exhaustively search the entire set of  $2^m$  excitation vectors. For this reason CELP encoders are computationally demanding.

As an example of a simple CELP coder consider federal standard 1016. This coder operates at 4.8 Kb/s using a fixed random codebook and attains a MOS rating of about 3.2. The encoder computes a tenth-order LPC analysis on frames of 30 milliseconds (240 samples), and then bandwidth expansion of 15 Hz is performed. By bandwidth expansion we mean that the LPC poles are radially moved toward the origin by multiplication of LPC coefficient  $b_m$  by a factor of  $\gamma^m$  where  $\gamma = 0.994$ . This empirically improves speech quality, but is mainly used to increase stability. The LPC coefficients are converted to line spectral pairs and quantized using nonuniform scalar quantization. The 240-sample frame is then divided into four subframes, each of which is allowed a separate codeword from of a set of 256, so that eight bits are required to encode the excitation of each subframe, or 32 bits for the entire frame.

This same strategy of frames and subframes is used in all modern CELP coders. The codebook search is the major computational task of the encoder, and it is not practical to use a codebook that covers an entire frame. It is typical to divide each frame into four subframes, but the excitation search needn't be performed on the subframes that belong to the analysis frame. Forward prediction with lookahead uses an analysis window that stretches into the future, while backward analysis inputs excitation vectors into LPC coefficients calculated from past samples. For example, let's number the subframes 1, 2, 3, and 4. Backward prediction may use the LPC coefficients computed from subframes 1, 2, 3, and 4 when trying the excitations for subframes 5, 6, 7, and 8. Forward prediction with lookahead of 2 subframes would use coefficients computed from subframes 3, 4, 5, and 6 when searching for excitations on subframes 1, 2, 3, and 4. Note that lookahead introduces further delay, since the search cannot start until the LPC filter is defined. Not only do coders using backward prediction not add further delay, they needn't send the coefficients at all, since by using closed-loop prediction the decoder can reproduce the coefficients before they are needed.

If random codebooks work, maybe even simpler strategies will. It would be really nice if sparse codebooks (i.e., ones in which the vectors have most of their components zero) would work. Algebraic codebooks are sets of excitation vectors that can be produced when needed, and so needn't be stored. The codewords in popular algebraic codebooks contain mostly zeros, but with a few nonzero elements that are either +1 or -1. With algebraic codebooks we needn't search a random codebook; instead we systematically generate all the legal codewords and input each in turn to the LPC filter. It turns out that such codebooks perform reasonably well; toll-quality G.729 and the lower bit rate of G.723.1 both use them. Coders that search codebooks, choosing the excitation that minimizes the discrepancy between the speech to be coded and the output of the excitation-driven LPC synthesis filter, are called Analysis By Synthesis (ABS) coders. The rationale for this name is clear. Such coders analyze the best excitation by exhaustively synthesizing all possible outputs and empirically choosing the best. What do we mean by the best excitation? Up to now you may have assumed that the output of the LPC synthesis filter was rated by SNR or correlation. This is not optimal since these measures do not correlate well with subjective opinion as to minimal distortion.

The main effect that can be exploited is 'masking' (recall exercise 11.4.2). Due to masking we needn't worry too much about discrepancies that result from spectral differences close to formant frequencies, since these are masked by the acoustic energy there and not noticed. So rather than using an error that is equally weighted over the bandwidth, it is better perceptually to use the available degrees of freedom to match the spectrum well where error is most noticeable. In order to take this into account, ABS CELP encoders perform *perceptual weighting* of both the input speech and LPC filter output before subtracting to obtain the residual. However, since the perceptual weighting is performed by a filter, we can more easily subtract first and perform a single filtering operation on the difference.

The perceptual weighting filter should de-emphasize spectral regions where the LPC has peaks. This can be achieved by using a filter with the system function

$$H(z) = \frac{\sum \gamma_1^m \beta_m z^{-m}}{\sum \gamma_2^m \beta_m z^{-m}}$$
(19.9)

where  $0 < \gamma_2 < \gamma_1 \leq 1$ . Note that both the numerator and denominator are performing bandwidth expansion, with the denominator expanding more than the numerator. By properly choosing  $\gamma_1$  and  $\gamma_2$  this weighting can be made similar to the psychophysical effect of masking.

Something seems to have been lost in the ABS CELP coder as compared with the LPC model. If we excite the LPC filter with an entry from a random or algebraic codebook, where does the pitch come from? To a certain extent it comes automatically from the minimization procedure. The algebraic codewords can have nonzero elements at pitch onset, and random codewords will automatically be chosen for their proper spectral content. However, were we to build the CELP coder as we have described it so far, we would find that its residual error displays marked pitch periodicity, showing that the problem is not quite solved. Two different ways have been developed to put the pitch back into the CELP model, namely *long-term prediction* and *adaptive codebooks*.



Figure 19.8: ABS CELP encoder using short- and long-term prediction. Only the essential elements are shown; CB is the codebook, PP the pitch (short-term) predictor, LPC the long-term predictor, PW the perceptual weighting filter, and EC the error computation. The input is used directly to find LPC coefficients and estimate the pitch and gain. The error is then used in ABS fashion to fine tune the pitch and gain, and choose the optimal codebook entry.

We mentioned long-term prediction in Section 19.2 as conceptually having two separate LPC filters. The short-term predictor, also called the LPC filter, the formant predictor, or the spectral envelope predictor, tracks and introduces the vocal tract information. It only uses correlations of less than 2 milliseconds or so and thus leaves the pitch information intact. The longterm predictor, also called the pitch predictor or the fine structure predictor, tracks and introduces the pitch periodicity. It only has a few coefficients, but these are delayed by between 2 and 20 milliseconds, according to the pitch period. Were only a single coefficient used, the pitch predictor system function would be

$$H_{pp}(z) = \frac{1}{1 - \beta z^{-D}} \tag{19.10}$$

where D is the pitch period. D may be found open loop, but for high quality it should be found using analysis by synthesis. For unvoiced segments the pitch predictor can be bypassed, sending the excitation directly to the LPC predictor, or it can be retained and its delay set randomly. A rough block diagram of a complete CELP encoder that uses this scheme is given in Figure 19.8.

Adaptive codebooks reinforce the pitch period using a different method. Rather than actually filtering the excitation, we use an effective excitation composed of two contributions. One is simply the codebook, now called the *fixed codebook*. To this is added the contribution of the adaptive codebook, which is formed from the previous excitation by duplicating it at the pitch period. This contribution is thus periodic with the pitch period and supplies the needed pitch-rich input to the LPC synthesis filter.

One last trick used by many CELP encoders is 'post-filtering'. Just as for ADPCM, the post-filter is appended after the decoder to improve the subjective quality of the reconstructed speech. Here this is accomplished by further strengthening the formant structure (i.e., by emphasizing the peaks and attenuating the valleys of the LPC spectrum), using a filter like the perceptual weighting filter (19.9). This somewhat reduces the formant bandwidth, but also reduces the residual coding noise. In many coders the post-filter is considered optional, and can be used or not according to taste.

#### EXERCISES

- 19.12.1 Explain why replacing LPC coefficient  $b_m$  with  $\gamma b_m$  with  $0 < \gamma < 1$  is called bandwidth expansion. Show that 15 Hz expansion is equivalent to  $\gamma = 0.994$ .
- 19.12.2 The G.723.1 coder when operating at the 5.3 Kb/s rate uses an algebraic codebook that is specified by 17 bits. The codewords are of length 60 but have no more than four nonzero elements. These nonzero elements are either all in even positions or all in odd positions. If in even positions, their indexes modulo 8 are all either 0, 2, 4, or 6. Thus 1 bit is required to declare whether even or odd positions are used, the four pulse positions can be encoded using 3 bits, and their signs using a single bit. Write a routine that successively generates all the legal codewords.
- 19.12.3 Explain how to compute the delay of an ABS CELP coder. Take into account the buffer, lookahead, and processing delays. What are the total delays for G.728 (frame 20 samples, backward prediction), G.729 (frame 80 samples, forward prediction), and G.723.1 (frame 240 samples, forward prediction)?
- 19.12.4 Obtain a copy of the G.729 standard and study the main block diagram. Explain the function of each block.
- 19.12.5 Repeat the previous exercise with the G.723.1 standard. What is the difference between the two rates? How does G.723.1 differ from G.729?

# **19.13** Telephone-Grade Speech Coding

This section can be considered to be the converse of Section 18.20; the purpose of a telephone-grade modem is to enable the transfer of data over voice lines (*data over voice*), while the focus of speech compression is on the transfer of voice over digital media (*voice over data*). Data over voice is an important technology since the **Public Switched Telephone Network** (PSTN) is the most widespread communications medium in the world; yet

the PSTN is growing at a rate of about 5% per year, while digital communications use is growing at several hundred percent a year. The amount of data traffic exceeded that of voice sometime during the year 2000, and hence voice over data is rapidly becoming the more important of the two technologies.

The history of telephone-grade speech coding is a story of rate halving. Our theoretical rate of 128 Kb/s was never used, having been reduced to 64 Kb/s by the use of logarithmic PCM, as defined in ITU standard G.711. So the first true rate halving resulted in 32 Kb/s and was accomplished by ADPCM, originally designated G.721. In 1990, ADPCM at rates 40, 32, 24, and 16 Kb/s were merged into a single standard known as G.726. At the same time G.727 was standardized; this 'embedded' ADPCM covers these same rates, but is designed for use in packetized networks. It has the advantage that the bits transmitted for the lower rates are subsets of those of the higher rates; congestion that arises at intermediate nodes can be relieved by discarding least significant bits without the need for negotiation between the encoder and decoder.

Under 32 Kb/s the going gets harder. The G.726 standard defines 24 and 16 Kb/s rates as well, but at less than toll-quality. Various SBC coders were developed for 16 Kb/s, either dividing the frequency range equally and using adaptive numbers of bits per channel, or using hierarchical wavelet-type techniques to divide the range logarithmically. Although these techniques were extremely robust and of relatively high perceived quality for the computational complexity, no SBC system was standardized for telephone-grade speech. In 1988, a coder, dubbed G.722, was standardized that encoded wideband audio (7 KHz sampled at 16,000 samples per second, 14 bits per sample) at 64 Kb/s. This coder divides the bandwidth from DC to 8 KHz into two halves using QMFs and encodes each with ADPCM.

In the early 1990s, the ITU defined performance criteria for a 16 Kb/s coder that could replace standard 32 Kb/s ADPCM. Such a coder was required to be of comparable quality to ADPCM, and with delay of less than 5 milliseconds (preferably less than 2 milliseconds). The coder, selected in 1992 and dubbed G.728, is a CELP with backward prediction, with LPC order of 50. Such a high LPC order is permissible since with closed-loop prediction the coefficients need not be transmitted. Its delay is 5 samples (0.625 ms), but its computational complexity is considerably higher than ADPCM, on the order of 30 MIPS.

The next breakthrough was the G.729 8 Kb/s CELP coder. This was accepted simultaneously with another somewhat different CELP-based coder for 6.4 and 5.4 Kb/s. The latter was named G.723.1 (the notation G.723

having been freed up by the original merging into G.726). Why were two different coders needed? The G.729 specification was originally intended for toll-quality wireless applications. G.728 was rejected for this application because of its rate and high complexity. The frame size for G.729 was set at 10 ms. and its lookahead at 5 ms. Due to the wireless channel, robustness to various types of bit errors was required. The process of carefully evaluating the various competing technologies took several years. During that time the urgent need arose for a low-bit-rate coder for videophone applications. Here toll-quality was not an absolute must, and it was felt by many that G.729 would not be ready in the alloted time. Thus an alternative selection process, with more lax testing, was instigated. For this application it was decided that a long 30 millisecond frame was acceptable, that a lower bit rate was desirable, but that slightly lower quality could be accommodated. In the end both G.729 and G.723.1 were accepted as standards simultaneously, and turned out to be of similar complexity.

The G.729 coder was extremely high quality, but also required over 20 MIPS of processing power to run. For some applications, including 'voice over modem', this was considered excessive. A modified coder, called G.729 Annex A, was developed that required about half the complexity, with almost negligible MOS reduction. This annex was adopted using the quick standardization strategy of G.723.1. G.723.1 defined as an annex a standard VAD and CNG mechanism, and G.729 soon followed suit with a similar mechanism as its Annex B. More recently, G.729 has defined annexes for additional bit rates, including a 6.4 Kb/s one.

At this point in time there is considerable overlap (and rivalry) between the two standards families. G.723.1 is the default coder for the voice over IP standard H.323, but G.729 is allowed as an option. G.729 is the default for the 'frame relay' standard FRF.11, but G.723.1 is allowed there as an option. In retrospect it is difficult to see a real need for two different coders with similar performance.

For even lower bit rates one must decide between MIPS and MOS. On the low MIPS low MOS front the U.S. Department of Defense initiated an effort in 1992 to replace LPC-10e with a 2.4 Kb/s encoder with quality similar to that of the 4.8 Kb/s CELP. After comparing many alternatives, in 1997 a draft was published based on MELP. The excitation used in this encoder consists of a pulse train and a uniform-distributed random noise generator filtered by time-varying FIR filters. MELP's quality is higher than that of straight LPC-10 because it addresses the latter's main weaknesses, namely voicing determination errors and not treating partially-voiced speech.

For higher MOS but with significantly higher MIPS requirements there

are several alternatives, including the Sinusoidal Transform Coder (STC) and Waveform Interpolation (WI). Were we to plot the speech samples, or the LPC residual, of one pitch period of voiced speech we would obtain some characteristic waveform; plotting again for some subsequent pitch period would result in a somewhat different waveform. We can now think of this waveform as evolving over time, and of its shape at any instant between the two we have specified as being determinable by interpolation. To enforce this picture we can create two-dimensional graphs wherein at regular time intervals we plot characteristic waveforms perpendicular to the time axis.

Waveform interpolation encoders operate on equally spaced frames. For each voiced frame the pitch pulses located and aligned by circular shifting, the characteristic waveform is found, and the slowly evolving waveform is approximated as a Fourier series. Recently waveform interpolation techniques have been extended to unvoiced segments as well, although now the characteristic waveform evolves rapidly from frame to frame. The quantized pitch period and waveform description parameters typically require under 5 Kb/s. The decompression engine receives these parameters severely undersampled, but recreates the required output rate by interpolation as described above.

The ITU has launched a new effort to find a 4 Kb/s toll-quality coder. With advances in DSP processor technology, acceptable coders at this, and even lower bit rates, may soon be a reality.

### EXERCISES

19.13.1 Cellular telephony networks use a different set of coders, including RPE-LTP (GSM) and VSELP (IS-54). What are the principles behind these coders and what are their parameters?

# **Bibliographical Notes**

There is a plethora of books devoted to speech signal processing. The old standard references include [210, 211], and of the newer generation we mention [66]. A relatively up-to-date book on speech recognition is [204] while [176] is an interesting text that emphasizes neural network techniques for speech recognition.

The first artificial speech synthesis device was created by Wolfgang von Kempelen in 1791. The device had a bellows that supplied air to a reed, and a manually manipulated resonance chamber. Unfortunately, the machine was not taken seriously after von Kempelen's earlier invention of a chess-playing machine had been exposed as concealing a midget chess expert. In modern times Homer Dudley from Bell Labs [55] was an early researcher in the field of speech production mechanisms. Expanding on the work of Alexander Graham Bell, he analyzed the human speech production in analogy to electronic communications systems, and built the VODER (Voice Operation DEmonstratoR), an analog synthesizer that was demonstrated at the San Francisco and New York World's Fairs. An early digital vocoder is described in [80]. In the 1980s, Dennis Klatt presented a much improved formant synthesizer [130, 131].

The LPC model was introduced to speech processing by Atal [10] in the U.S. and Itakura [111] in Japan. Many people were initially exposed to it in the popular review [155] or in the chapter on LPC in [210]. The power cepstrum was introduced in [20]; the popular DSP text [186] devotes a chapter to homomorphic processing; and [37] is worth reading. We didn't mention that there is a nonrecursive connection between the LPC and LPC cepstrum coefficients [239].

Distance measures, such as the Itakura-Saito distance, are the subject of [112, 113, 110, 84]. The inverse-E filtering problem and RASTA-PLP are reviewed in [102, 101]. The sinusoidal representation has an extensive literature; you should start with [163, 201].

For questions of speech as a dynamical system and its fractal dimension consult [259, 156, 172, 226]. Unfortunately, there is as yet no reference that specifies for the optimal minimal set of features.

Pitch detectors and U/V decision mechanisms are the subject of [205, 206, 121]. Similar techniques for formant tracking are to be found in [164, 230].

Once, the standard text on coding was [116], but the field has advanced tremendously since then. Vector quantization is covered in a review article [85] and a text [69], while the LBG algorithm was introduced in [149].

Postfiltering is best learnt from [35]. The old standard coders are reviewed in [23] while the recent ones are described in [47]. For specific techniques and standards, LPC and LPC-10: [9, 261, 121]; MELP: [170]; basic CELP: [11]; federal standard 1016: [122]; G.729 and its annexes: [231, 228, 229, 15]; G.728: [34]; G.723.1: no comprehensive articles; waveform interpolation: [132].