Chapter 1

How is speech processed in a cell phone conversation?

Every cell phone solves 10 linear equations
in 10 unknowns every 20 milliseconds

T. Dutoit(°), N. Moreau(*), P. Kroon(+)

(°) Faculté Polytechnique de Mons, Belgium
(*) Ecole Nationale Supérieure des Télécommunications, Paris, France
(+) LSI, Allentown, PA, USA

Although most people see the cell phone as an extension of conventional wired phone service or POTS (plain old telephone service) the truth is that cell phone technology is extremely complex and a marvel of technology. Very few people realize that these small devices perform hundreds of millions of operations per seconds to be able to maintain a phone conversation. If we take a closer look at the module that converts the electronic version of the speech signal into a sequence of bits, we see that for every 20 ms of input speech a set of speech model parameters is computed, and transmitted to the receiver. The receiver converts these parameters back into speech. In this Chapter, we will see how linear predictive (LP) analysis-synthesis lies at the very heart of mobile phone transmission of speech. We first start with an introduction to linear predictive speech modeling, and follow with a MATLAB-based proof of concept.

1.1 Background – Linear predictive processing of speech

Speech is produced by an excitation signal generated in our throat, which is modified by resonances produced by different shapes of our vocal, nasal, and pharyngeal tracts. This excitation signal can be the glottal pulses produced by the periodic opening and closing of our vocal folds (which creates *voiced* speech such as the vowels in "*voice"*), or just some continuous air flow pushed by our lungs (which creates *unvoiced* speech such as the last sound in "*voice"*), or even a combination of both at the same time (such as the first sound in "*voice"*).

The periodic component of the glottal excitation is characterized by its fundamental frequency F_0 [Hz], called *pitch* [1]. The resonant frequencies of the vocal, oral, and pharyngeal tracts are called *formants*. On a spectral plot of a speech frame, pitch appears as narrow peaks for fundamental and harmonics; formants appear as wide peaks of the envelope of the spectrum (Fig. 1.1).

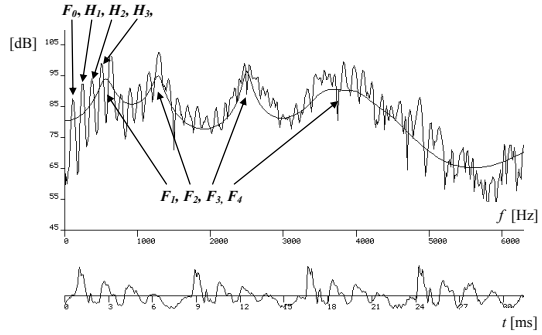

Fig. 1.1 A 30 ms frame of voiced speech (bottom) and its spectrum (shown here as the magnitude of its FFT). Harmonics are denoted as H_1, H_2, H_3, etc.; formants are denoted as F_1, F_2, F_3, etc. The spectral envelope is shown here for convenience; it only implicitly appears in the regular FFT.

[1] Strictly speaking, pitch is defined as the *perceived* fundamental frequency.

1.1.1 The LP model of speech

As early as 1960, Fant proposed a linear model of speech production (Fant, 1960), termed as the *source-filter model*, based on the hypothesis that the glottis and the vocal tract are fully uncoupled. This model led to the well-known *autoregressive (AR)* or *linear predictive (LP)*[2] model of speech production (Rabiner and Shafer 1978), which describes speech *s(n)* as the output $\tilde{s}(n)$ of an *all-pole* filter *1/A(z)*, resulting from some excitation $\tilde{e}(n)$:

$$\tilde{S}(z) = \tilde{E}(z)\frac{1}{\sum_{i=0}^{p} a_i z^{-i}} = \tilde{E}(z)\frac{1}{A_p(z)} \qquad (a_0 = 1) \tag{1.1}$$

where $\tilde{S}(z)$ and $\tilde{E}(z)$ are the Z transforms of the speech and excitation signals respectively, and *p* is the *prediction order*. The excitation of the LP model (Fig. 1.2) is assumed to be either a sequence of regularly spaced pulses (whose period T_0 and amplitude σ can be adjusted), or white Gaussian noise (whose variance σ^2 can be adjusted), thereby implicitly defining the so-called Voiced/Unvoiced (V/UV) decision. The filter $1/A_p(z)$ is termed as the *synthesis filter* and $A_p(z)$ is called the *inverse filter*.

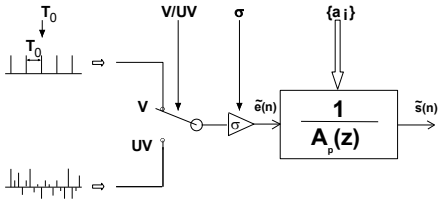

Fig. 1.2 The LP model of speech production

Equation (1.1) implicitly introduces the concept of linear predictability of speech (hence the name of the model), which states that each speech sample can be expressed as a weighted sum of the *p* previous samples, plus some excitation contribution:

[2] Sometimes it is denoted as the *LPC model (linear predictive coding)* because it has been widely used for speech coding

$$\tilde{s}(n) = \tilde{e}(n) - \sum_{i=1}^{p} a_i \tilde{s}(n-i) \tag{1.2}$$

1.1.2 The LP estimation algorithm

From a given signal, a practical problem is to find the best set of prediction coefficients – that is, the set that minimizes modeling errors – by trying to minimize audible differences between the original signal and the one that is produced by the model of Fig. 1.2. This implies to estimate the value of the LP parameters: pitch period T_0, gain σ, V/UV switch position, and prediction coefficients $\{a_i\}$.

Pitch and voicing (*V/UV*) determination is a difficult problem. Although speech seems periodic, it is never truly the case. Glottal cycle amplitude varies from period to period (*shimmer*) and its period itself is not constant (*jitter*). Moreover, the speech waveform only reveals filtered glottal pulses, rather than glottal pulses themselves. This makes a realistic measure of T_0 even more complex. In addition, speech is rarely completely voiced; its additive noise components make pitch determination even harder. Many techniques have been developed to estimate T_0 (see Hess, 1992; de la Cuadra, 2007).

The estimation of σ and of the prediction coefficients can be performed simultaneously and fortunately independently of the estimation of T_0.

For a given speech signal *s(n),* imposing the value of the $\{a_i\}$ coefficients in the model results in the *prediction residual* signal*, e(n)*:

$$e(n) = s(n) + \sum_{i=1}^{p} a_i s(n-i) \tag{1.3}$$

which is simply the output of the inverse filter excited by the speech signal (Fig. 1.3).

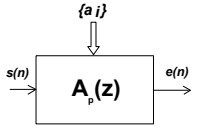

Fig. 1.3 Inverse filtering of speech

The principle of AR estimation is to choose the set $\{a_1,a_2,...a_p\}$ which minimizes the expectation $E(e^2(n))$ of the residual energy:

$$\{a_i\}^{opt} = \arg\min_{a_i}\left(E(e^2(n))\right) \tag{1.4}$$

As a matter of fact, it can be shown that, if *s(n)* is stationary, the synthetic speech $\tilde{s}(n)$ produced by the LP model (Fig. 1.2) using this specific set of prediction coefficients in (1.2) will exhibit the same spectral envelope as *s(n)*. Since the excitation of the LP model (pulses or white noise) has a flat spectral envelope, this means that the frequency response of the synthesis filter will approximately match the spectral envelope of *s(n),* and that the spectral envelope of the LP residual will be approximately flat. In a word: inverse filtering de-correlates speech.

Developing the LMSE (Least Mean Squared Error) criterion (1.4) easily leads to the so-called set of *p* Yule-Walker linear equations:

$$\begin{bmatrix} \phi_{xx}(0) & \phi_{xx}(1) & \dots & \phi_{xx}(p-1) \\ \phi_{xx}(1) & \phi_{xx}(0) & \dots & \phi_{xx}(p-2) \\ \dots & \dots & \dots & \dots \\ \phi_{xx}(p-1) & \phi_{xx}(p-2) & \dots & \phi_{xx}(0) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \dots \\ a_p \end{bmatrix} = - \begin{bmatrix} \phi_{xx}(1) \\ \phi_{xx}(2) \\ \dots \\ \phi_{xx}(p) \end{bmatrix} \tag{1.5}$$

in which $\phi_{xx}(k) \quad (k = 0 \dots p)$ are the *p*+1 first autocorrelation coefficients of *s(n)*. After solving this set of equations, the optimal value of σ is then given by:

$$\sigma^2 = \sum_{i=0}^{p} a_i \phi_{xx}(i) \tag{1.6}$$

It should be noticed that, since equations (1.5) are only based on the autocorrelation function of *s(n)*, the model does not try to imitate the exact speech waveform, but rather its spectral envelope (based on the idea that our ear is more sensitive to the amplitude spectrum than to the phase spectrum).

1.1.3 LP processing in practice

Since speech is non stationary, the LP model is applied on speech *frames* (typically 30 ms long, with an overlap of 20 ms; Fig. 1.4), in which the signal is assumed to be stationary given the inertia of the articulatory muscles[3].

Speech samples are usually weighted using a *weighting window* (typically a 30 ms-long Hamming window). This prevents the first samples of each frame, which cannot be correctly predicted, from having too much weight in (1.4) by producing higher values of $e^2(n)$.

The $\phi_{xx}(k) \quad (k = 0 \dots p)$ autocorrelation coefficients are then estimated on a limited number of samples (typically 240 samples, for 30 ms of speech with a sampling frequency of 8 kHz). The prediction order *p* (which is also the number of poles in the all-pole synthesis filter) is chosen such that the resulting synthesis filter has enough degrees of freedom to copy the spectral envelope of the input speech. Since there is approximately one formant per kHz of bandwidth of speech, at least 2**B* poles are required (where *B* is the signal bandwidth in kHz, i.e. half the sampling frequency). Two more poles are usually added for modeling the glottal cycle waveform (and also empirically, because the resulting LPC speech sounds better). For telephone-based applications, working with a sampling frequency of 8 kHz, this leads to *p*=10.

Although (1.5) can be solved with any classical matrix inversion algorithm, the so-called *Levinson-Durbin* algorithm is preferred for its speed, as it takes into account the special structure of the matrix (all elements on diagonals parallel to the principal diagonal are equal; this characterizes a *Toeplitz* matrix). See Rabiner and Schafer (1978) or Quatieri (2002) for details.

The *prediction coefficients* $\{a_i\}$ are finally computed for every frame (i.e., typically every 10 to 20 ms).

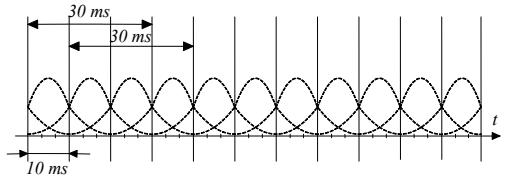

Fig. 1.4 Frame-based processing of speech (shown here with a frame length of 30 ms and a shift of 10 ms).

[3] In practice this is only an approximation, which tends to be very loose for plosives, for instance.

The complete block diagram of an LPC speech analysis-synthesis system is given in Fig. 1.5.

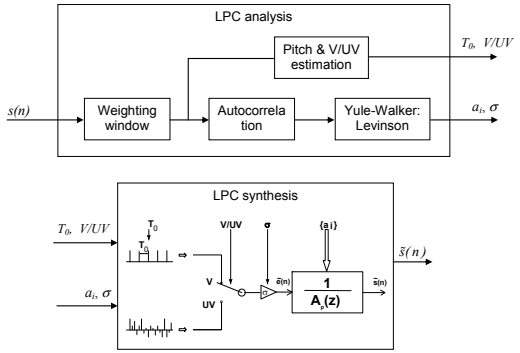

Fig. 1.5 A linear predictive speech analysis-synthesis system

1.1.4 Linear predictive coders

The LPC analysis-synthesis system which has been described above is not exactly the one embedded in cell phones.

It is however implemented in the so-called NATO LPC10 standard (NATO, 1984), which was used for satellite transmission of speech communications until 1996. This norm makes it possible to encode speech with a bit rate as low as 2400 bits/s (frames are 22.5 ms long, and each frame is coded with 54 bits: 7 bits for pitch and *V/UV* decision, 5 bits for the gain, 42 bits for the prediction coefficients [4]). In practice, prediction coefficients are actually not used as such; the related *reflection coefficients* or *log area ratios* are preferred, since they have better quantization properties. Quantization of prediction coefficients can result in unstable filters.

The number of bits in LPC10 was chosen such that it does not bring audible artifacts to the LPC speech. The example LPC speech produced in Section 2 is therefore a realistic example of typical LPC10 speech. Clearly

[4] Advanced LP coders, such as CELP, have enhanced prediction coefficients coding down to 30 bits.

this speech coder suffers from the limitations of the poor (and binary!) excitation model. Voiced fricatives, for instance, cannot be adequately modeled since they exhibit voiced *and* unvoiced features simultaneously. Moreover, the LPC10 coder is very sensitive to the efficiency of its voiced/unvoiced detection and F_0 estimation algorithms. Female voices, whose higher F_0 frequency sometimes results in a second harmonic at the center of the first formant, often lead to F_0 errors (the second harmonic being mistaken for F_0).

One way of enhancing the quality of LPC speech is obviously to reduce the constraints on the LPC excitation, so as to allow for a better modeling of the prediction residual *e(n)* by the excitation $\tilde{e}(n)$. As a matter of fact, passing this residual through the synthesis filter $1/A(z)$ produces the original speech (Fig. 1.6, which is the inverse of Fig. 1.3).

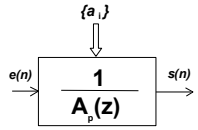

Fig. 1.6 Passing the prediction residual through the synthesis filter produces the original speech signal

The *Multi-Pulse Excited* (MPE; Atal and Remde 1982) was an important step in this direction, as it was the first approach to implement an analysis-by-synthesis process (i.e., a closed loop) for the estimation of the excitation features. The MPE excitation is characterized by the positions and amplitudes of a limited number of pulses per frame (typically 10 pulses per 10ms frame; Fig. 1.7). Pitch estimation and voiced/unvoiced decision are no longer required. Pulse positions and amplitudes are chosen iteratively (Fig. 1.8), so as to minimize the energy of the modeling error (the difference between the original speech and the synthetic speech). The error is filtered by a *perceptual filter* before its energy is computed:

$$P(z) = \frac{A(z)}{A(z/\gamma)} \tag{1.7}$$

The role of this filter, whose frequency response can be set to any intermediate between all pass response (γ =1) and the response of the inverse filter (γ =0), is to reduce the contributions of the formants to the estimation of the error. The value of γ is typically set to 0.8.

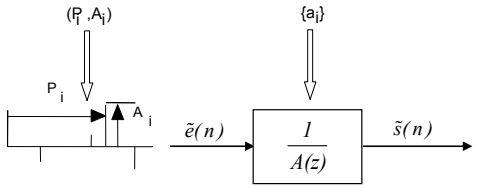

Fig. 1.7 The MPE decoder

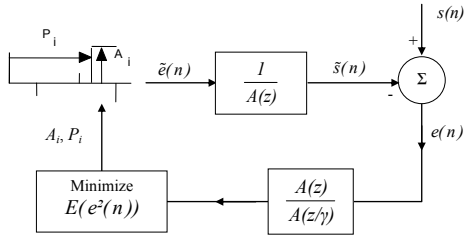

Fig. 1.8 Estimation of the MPE excitation by an analysis-by-synthesis loop in the MPE encoder

The *Code-Excited Linear Prediction* coder (CELP; Schroeder and Atal 1985) further extended the idea of analysis-by-synthesis speech coding, by using the concept of *vector quantization* (VQ) for the excitation sequence. In this approach, the encoder selects one excitation sequence from a predefined *stochastic codebook* of possible sequences (Fig. 1.9), and only sends the index of the selected sequence to the decoder, which has a similar codebook. While the lowest quantization rate for scalar quantization is 1 bit per sample, VQ allows fractional bit rates. For example, quantizing 2 samples simultaneously using a 1 bit codebook will result in 0.5 bits per sample. More typical values are a 10-bit codebook with codebook vectors of dimension 40, resulting in 0.25 bits/sample. Given the very high variability of speech frames, however, (due to changes in glottal excitation *and* vocal tract), vector quantized speech frames would only be possible with a very large codebook. The great idea of CELP is precisely to perform VQ on LP residual sequences: as we have seen in Section 1.1.2, the LP residual has a

flat spectral envelope, which makes it easier to produce a small but somehow exhaustive codebook of LP residual sequences. CELP can thus be seen as an *adaptive vector quantization* scheme of speech frames (adaptation being performed by the synthesis filter).

CELP additionally takes advantage of the periodicity of voiced sounds to further improve predictor efficiency. A so-called *long-term predictor* filter is cascaded with the synthesis filter, which enhances the efficiency of the codebook. The simplest long-term predictor consists of a simple variable delay with adjustable gain (Fig. 1.9).

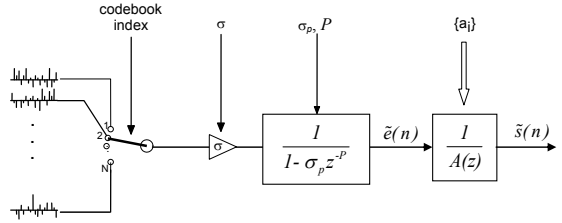

Fig. 1.9 The CELP decoder

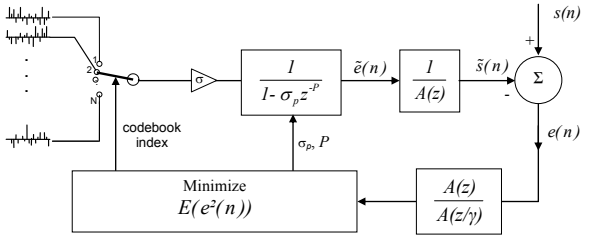

Fig. 1.10 Estimation of the CELP excitation by an analysis-by-synthesis loop in the CELP encoder

Various coders have been developed after MPE and CELP using the same analysis-by-synthesis principle, with the goal of enhancing CELP quality while further reducing bit-rate, among which the Mixed-Excitation

Linear Prediction (MELP; McCree and Barnwell 1995) and the Harmonic and Vector eXcitation coding (HVXC; Matsumoto *et al* 1997). In 1996, LPC-10 was replaced by MELP to be the United States Federal Standard for coding at 2.4 kbps.

From 1992 to 1996, GSM (Global System for Mobile Communication) phones embedded a particular form of MPE, the *RPE-LPC* (Regular Pulse Excited; Kroon *et. al* 1986) coder, with additional constraints on the positions of the pulses: the RPE pulses were evenly spaced (but their amplitude, as well as the position of the first pulse, is left open). Speech is divided into 20 millisecond frames, each of which is encoded as 260 bits, giving a total bit rate of 13 kbps. In 1996, this so-called *full-rate* (FR) codec was replaced by the *enhanced full-rate* (EFR) codec, implementing a variant of CELP termed as Algebraic-CELP (ACELP, Salami *et al.* 1998). The ACELP codebook structure allows efficient searching of the optimal codebook index thereby eliminating one of the main drawbacks of CELP which is its complexity. The EFR coder operates at 11.2 kb/s and produces better speech quality than the FR coder at 13 kb/s. A variant of the ACELP coder has been standardized by ITU-T as G.729 for operation at a bit rate of 8 kb/s. Newer generations of coders that are used in cell phones are all based on the CELP principle and can operate at bitrates as low as 4.75 kb/s to 11.2 kb/s.

1.2 MATLAB proof of concept : ASP_cell_phone.m

We will first examine the contents of a speech file (1.2.1) and perform LP analysis and synthesis on a voiced (1.2.2) and on an unvoiced frame (1.2.3). We will then generalize this approach to the complete speech file, by first synthesizing all frames as voiced and imposing a constant pitch (1.2.4), then by synthesizing all frames as unvoiced (1.2.5), and finally by using the original pitch[5] and voicing information as in LPC10 (1.2.6). We will conclude this Section by changing LPC10 into CELP (1.2.7).

[5] By "original pitch", we mean the pitch which can be measured on the original signal.

1.2.1 Examining a speech file

Let us load file 'speech.wav', listen to it, and plot its samples (Fig. 1.11). This file contains the sentence "Paint the circuits" sampled at 8 kHz, with 16 bits[6].

```
speech=wavread('speech.wav');
plot(speech)
xlabel('Time (samples)'); ylabel('Amplitude');
sound(speech,8000);
```

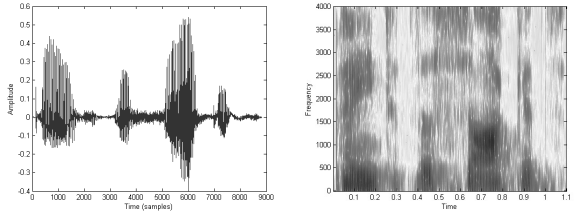

Fig. 1.11 Input speech: the speech.wav file (left: waveform; right: spectrogram)

The file is about 1.1 s long (9000 samples). One can easily spot the position of the four vowels appearing in this plot, since vowels usually have higher amplitude than other sounds. The vowel 'e' in "the", for instance, is approximately centered on sample 3500.

As such, however, the speech waveform is not "readable", even by an expert phonetician. Its information content is hidden. In order to reveal it to the eyes, let us plot a spectrogram of the signal (Fig. 1.11). We then choose a *wideband spectrogram*[7], by imposing the length of each frame to be approximately 5 ms long (40 samples) and a hamming weighting window.

```
specgram(speech,512,8000,hamming(40))
```

In this plot, pitch periods appear as vertical lines. As a matter of fact, since the length of analysis frames is very small, some frames fall on the

[6] This sentence was taken from the Open Speech Repository on the web.

[7] A wideband spectrogram uses a small amount of samples (typically less than the local pitch period), so as to better reveal formants.

peaks (resp., on the valleys) of pitch periods, and thus appear as a darker (resp., lighter) vertical lines.

In contrast, formants (resonant frequencies of the vocal tract) appear as dark (and rather wide) horizontal traces. Although their frequency is not easy to measure with precision, experts looking at such a spectrogram can actually often read it (i.e. guess the corresponding words). This clearly shows that formants are a good indicator of the underlying speech sounds.

1.2.2 Linear prediction synthesis of 30 ms of voiced speech

Let us extract a 30 ms frame from a voiced part (i.e. 240 samples) of the speech file, and plot its samples (Fig. 1.12).

```
input_frame=speech(3500:3739);
plot(input_frame);
```

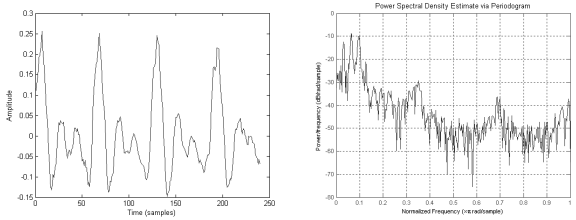

Fig. 1.12 A 30ms-long voiced speech frame taken from a vowel (left: waveform; right: periodogram)

As expected this sound is approximately periodic (period=65 samples, i.e. 80 ms; fundamental frequency = 125 Hz). Notice, though, that this is only apparent; in practice, no sequence of samples can be found more than once in the frame.

Now let us see the spectral content of this speech frame (Fig. 1.12), by plotting its *periodogram* on 512 points (using a normalized frequency axis; remember π corresponds to Fs/2, i.e. to 4000 Hz here).

```
periodogram(input_frame,[],512);
```


The fundamental frequency appears again at around 125 Hz. One can also roughly estimate the position of formants (peaks in the spectral envelope) at ± 300 Hz, 1400 Hz, 2700 Hz.

Let us now fit an LP model of order 10 to our voiced frame [8]. We obtain the prediction coefficients (`ai`) and the variance of the residual signal (`sigma_square`).

```
[ai, sigma_square]=lpc(input_frame,10);
sigma=sqrt(sigma_square);
```

The estimation parameter inside LPC is called the Levinson-Durbin algorithm. It chooses the coefficients of an FIR filter *A(z)* so that when passing the input frame into *A(z)*, the output, termed as the prediction residual, has minimum energy. It can be shown that this leads to a filter which has anti-resonances wherever the input frame has a formant. For this reason, the *A(z)* filter is termed as the "inverse" filter. Let us plot its frequency response (on 512 points), and superimpose it to that of the "synthesis" filter *1/A(z)* (Fig. 1.13).

```
[HI,WI]=freqz(ai, 1, 512);
[H,W]=freqz(1,ai, 512);
plot(W,20*log10(abs(H)),'-',WI,20*log10(abs(HI)),'--');
```

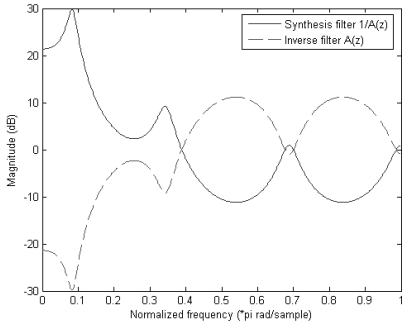

Fig. 1.13 Frequency responses of the inverse and synthesis filters

[8] We do not apply windowing prior to LP analysis now, as it has no tutorial benefit. We will add it in subsequent Sections.

In other words, the frequency response of the filter *1/A(z)* matches the spectral amplitude envelope of the frame. Let us superimpose this frequency response to the periodogram of the vowel (Fig. 1.14)[9].

```
periodogram(input_frame,[],512,2)
hold on;
plot(W/pi,20*log10(sigma*abs(H)));
hold off;
```

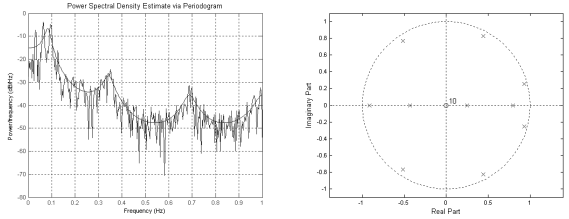

Fig. 1.14 left: Frequency response of the synthesis filter, superimposed with the periodogram of the frame; right: Poles and zeros of the filter

In other words, the LPC fit has automatically adjusted the poles of the synthesis filter close to the unit circle at angular positions chosen to imitate formant resonances (Fig. 1.14).

```
zplane(1,ai);
```

If we apply the inverse of this filter to the input frame, we obtain the prediction residual (Fig. 1.15).

```
LP_residual=filter(ai,1,input_frame);
plot(LP_residual)

periodogram(LP_residual,[],512);
```

[9] The `periodogram` function of MATLAB actually shows the so-called *one-sided periodogram*, which has twice the value of the two-sided periodogram in [0, Fs/2]. In order to force MATLAB to show the real value of the two-sided periodogram in [0, Fs/2], we claim Fs=2.

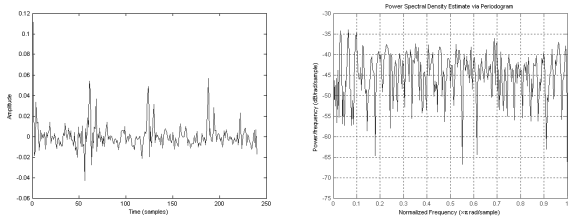

Fig. 1.15 The prediction residual (left: waveform; right: periodogram)

Let us compare the spectrum of this residual to the original spectrum. The new spectrum is approximately flat; its fine spectral details, however, are the same as those of the analysis frame. In particular, its pitch and harmonics are preserved.

For obvious reasons, applying the synthesis filter to this prediction residual results in the analysis frame itself (since the synthesis filter is the inverse of the inverse filter).

```
output_frame=filter(1, ai,LP_residual);
plot(output_frame);
```

The LPC model actually models the prediction residual of voiced speech as an impulse train with adjustable pitch period and amplitude. For the speech frame considered, for instance, the LPC ideal excitation is a sequence of pulses separated by 64 zeros (so as to impose a period of 65 samples; Fig. 1.16). Notice we multiply the excitation by some gain so that its variance matches that of the residual signal.

```
excitation = [1;zeros(64,1);1;zeros(64,1);1;zeros(64,1);…
                1;zeros(44,1)];
gain=sigma/sqrt(1/65);
plot(gain*excitation);

periodogram(gain*excitation,[],512);
```

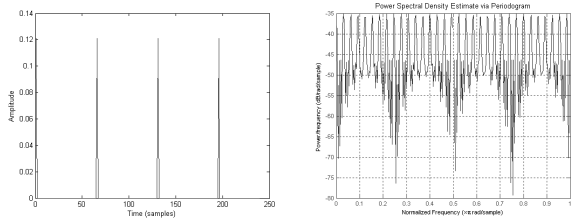

Fig. 1.16 The LPC excitation (left: waveform; right: periodogram)

Clearly, as far as the waveform is concerned, the LPC excitation is far from similar to the prediction residual. Its spectrum (Fig. 1.16), however, has the same broad features as that of the residual: flat envelope, and harmonic content corresponding to F_0. The main difference is that the excitation spectrum is "over-harmonic" compared to the residual spectrum.

Let us now use the synthesis filter to produce an artificial "e".

```
synt_frame=filter(gain,ai,excitation);
plot(synt_frame);

periodogram(synt_frame,[],512);
```

Although the resulting waveform is obviously different from the original one (this is due to the fact that the LP model does not account for the phase spectrum of the original signal), its spectral envelope is identical. Its fine harmonic details, though, also widely differ: the synthetic frame is actually "over-harmonic" compared to the analysis frame (Fig. 1.17).

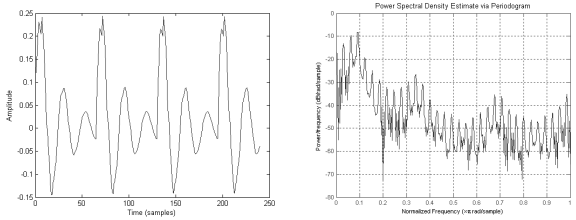

Fig. 1.17 Voiced LPC speech (left: waveform; right: periodogram)

1.2.3 Linear prediction synthesis of 30 ms of unvoiced speech

It is easy to apply the same process to an unvoiced frame, and compare the final spectra again. Let us first extract an unvoiced frame and plot it (Fig. 1.18).

```
input_frame=speech_HF(4500:4739);
plot(input_frame);
```

As expected, no clear periodicity appears.

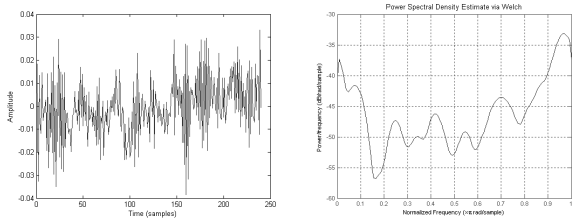

Fig. 1.18 A 30ms-long frame of unvoiced speech (left: waveform; right: power spectral density)

Now let us see the spectral content of this speech frame. Notice that, since we are dealing with noisy signals, we use the *averaged periodogram* to estimate power spectral densities, although with less frequency resolution than using a simple periodogram. The MATLAB `pwlech` function does this, with 8 sub-frames by default and 50% overlap.

```
pwelch(input_frame);
```

Let us now apply an LP model of order 10, and synthesize a new frame. Synthesis is performed by all-pole filtering a Gaussian white noise frame with standard deviation set to the prediction residual standard deviation, σ.

```
[ai, sigma_square]=lpc(input_frame,10);
sigma=sqrt(sigma_square);
excitation=randn(240,1);
synt_frame=filter(sigma,ai,excitation);
plot(synt_frame);

pwelch(synt_frame);
```

The synthetic waveform (Fig. 1.19) has no sample in common with the original waveform. The spectral envelope of this frame, however, is still similar to the original one, enough at least for both the original and synthetic signals to be perceived as the same colored noise [10].

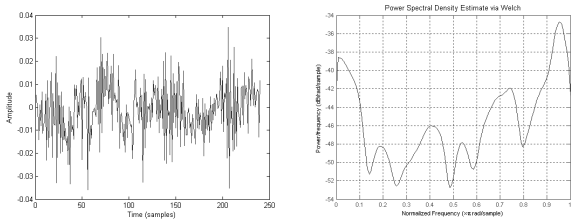

Fig. 1.19 Unvoiced LPC speech (left: waveform; right: psd)

1.2.4 Linear prediction synthesis of a speech file, with fixed F_0

We will now loop the previous operations for the complete speech file, using 30ms analysis frames overlapping by 20 ms. Frames are now weighted with a Hamming window. At synthesis time, we simply synthesize 10 ms of speech, and concatenate the resulting synthetic frames to obtain the output speech file. Let us choose 200 Hz as synthesis F0, for convenience: this way each 10ms excitation frame contains exactly two pulses.

```
for i=1:(length(speech)-160)/80; % number of frames

    % Extracting the analysis frame
    input_frame=speech_HF((i-1)*80+1:(i-1)*80+240);

    % Hamming window weighting and LPC analysis
    [ai, sigma_square]=lpc(input_frame.*hamming(240),10);
    sigma=sqrt(sigma_square);

    % Generating 10 ms of excitation
    % = 2 pitch periods at 200 Hz
    excitation=[1;zeros(39,1);1;zeros(39,1)];
    gain=sigma/sqrt(1/40);
```

[10] While both power spectral densities have identical spectral slopes, one should not expect them to exhibit a close match in terms of their details, since LPC modeling only reproduces the smooth spectral envelope of the original signal.


```
    % Applying the synthesis filter
    synt_frame=filter(gain, ai,excitation);

    % Concatenating synthesis frames
    synt_speech_HF=[synt_speech_HF;synt_frame];

end
```

The output waveform basically contains a sequence of LP filter impulse responses. Let us zoom on 30 ms of LPC speech (Fig. 1.20).

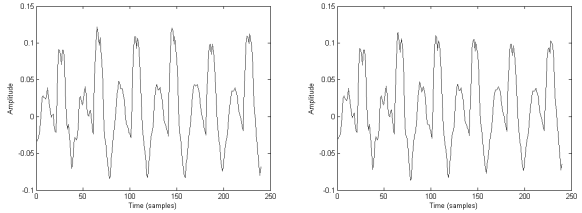

Fig. 1.20 Zoom on 30 ms of LPC speech (left: with internal variable reset; right: with internal variable memory)

It appears that in many cases the impulse responses have been cropped to the 10 ms synthetic frame size. As a matter of fact, since each synthesis frame was composed of two identical impulses, one should expect our LPC speech to exhibit pairs of identical pitch periods. This is not the case, due to the fact that for producing each new synthetic frame the internal variables of the synthesis filter are implicitly reset to zero. We can avoid this problem by maintaining the internal variables of the filter from the end of each frame to the beginning of the next one.

We initialize a vector `z` with ten zeros, and change the synthesis code into:

```
    % Applying the synthesis filter
    % Taking care of the internal variables of the filter
    gain=sigma/sqrt(1/40);
    [synt_frame,z]=filter(gain, ai, excitation, z);
```

This time the end of each impulse response is properly added to the beginning of the next one, which results in more smoothly evolving periods (Fig. 1.20).

If we want to synthesize speech with constant pitch period length different from a sub-multiple of 80 samples (say, `NO`=65 samples), we additionally need to take care of a possible pitch period `offset` in the excitation

signal. After initializing this `offset` to zero, we simply change the excitation code into:

```
% Generating 10 ms of excitation
% taking a possible offset into account

% if pitch period length > excitation frame length
if offset>=80
    excitation=zeros(80,1);
    offset=offset-80;
else
    % complete the previously unfinished pitch period
    excitation=zeros(offset,1);
    % for all pitch periods in the remaining of the frame
    for j=1:floor((80-offset)/N0)
        % add one excitation period
        excitation=[excitation;1;zeros(N0-1,1)];
    end;
    % number of samples left in the excitation frame
    flush=80-length(excitation);
    if flush~=0
        % fill the frame with a partial pitch period
        excitation=[excitation;1;zeros(flush-1,1)];
        % remember to fill the remaining of the period in
        % next frame
        offset=N0-flush;
    else offset=0;
    end
end
gain=sigma/sqrt(1/N0);
```

1.2.5 Unvoiced linear prediction synthesis of a speech file

Synthesizing the complete speech file as LPC unvoiced speech is easy. Periodic pulses are simply replaced by white noise, as in Section 1.2.3.

```
% Generating 10 ms of excitation
excitation=randn(80,1); % White Gaussian noise
gain=sigma;
```

As expected, the resulting speech sounds like whisper.

1.2.6 Linear prediction synthesis of a speech file, with original F_0

We will now synthesize the same speech, using the original F_0. We will thus have to deal with the additional problems of pitch estimation (on a frame-by-frame basis), including voiced/unvoiced decision. This approach is similar to that of the LPC10 coder (except we do not quantize coefficients here). We change the excitation generation code into:

```
% local synthesis pitch period (in samples)
N0=pitch(input_frame);
```



```
% Generating 10 ms of excitation
if N0~=0 % voiced frame
    % Generate 10 ms of voiced excitation
    % taking a possible offset into account

    (same code as in Section 1.2.4)

else
    % Generate 10 ms of unvoiced voiced excitation

    (same code as in Section 1.2.5)

    offset=0; % reset for subsequent voiced frames
end;
```

MATLAB function involved:

- `T0=pitch(speech_frame)`: returns the pitch period T_0 (in samples) of a speech frame (T_0 is set to zero when the frame is detected as unvoiced). T_0 is obtained from the maximum of the (estimated) autocorrelation of the LPC residual. Voiced/unvoiced decision is based on the ratio of this maximum by the variance of the residual. This simple algorithm is not optimal, but will do the job for this proof of concept.

The resulting synthetic speech (Fig. 1.21) is intelligible. It shows the same formants as the original speech. It is therefore acoustically similar to the original, except for the additional buzzyness which has been added by the LP model.

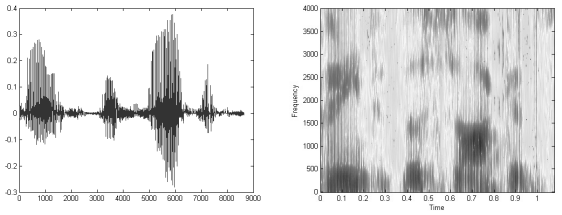

Fig. 1.21 LPC10 speech

1.2.7 CELP analysis-synthesis of a speech file

Our last step will be to replace the LPC10 excitation by a more realistic Code-Excited Linear Prediction (CELP) excitation, obtained by selecting the best linear combination of excitation components from a codebook. Component selection is performed in a closed loop, so as to minimize the difference between the synthetic and original signals.

We start with 30 ms LP analysis frames, shifted every 5 ms, and a codebook size of 512 vectors, from which 10 components are chosen for every 5 ms synthesis frame. [11]

MATLAB function involved:

- `[gains, indices] = find_Nbest_components(signal, ... codebook_vectors, codebook_norms , N)`

  This function finds the `N` best components of signal from the vectors in `codebook_vectors`, so that the residual error:

  `error = signal - codebook_vectors(indices)*gains`

  is minimized. Components are found one-by-one using a greedy algorithm. When components in `codebook_vectors` are not orthogonal, the search is therefore suboptimal.

```
frame_length=240; % length of the LPC analysis frame
frame_shift=40; % length of the synthesis frames
codebook_size = 512; % number of vectors in the codebook
N_components= 10;  % number of codebook components per frame

speech=wavread('speech.wav');

% Initializing internal variables
z_inv=zeros(10,1);  % inverse filter
z_synt=zeros(10,1); % synthesis filter
synt_speech_CELP=[];

% Generating the stochastic excitation codebook
codebook = randn(frame_shift,codebook_size);

for i=1:(length(speech)-frame_length+frame_shift)/frame_shift;

    input_frame=speech((i-1)*frame_shift+1:...
                       (i-1)*frame_shift+frame_length);

    % LPC analysis of order 10
    ai = lpc(input_frame.*hamming(frame_length), 10);

    % Extracting frame_shift samples from the LPC analysis frame
    speech_frame = input_frame((frame_length-frame_shift)/2+1:...
```

[11] These values actually correspond to a rather high bit-rate, but we will show in the next paragraphs how to lower the bit-rate while maintaining the quality of synthetic speech.


```
                       (frame_length-frame_shift)/2+frame_shift);

    % Filtering the codebook (all column vectors)
    codebook_filt = filter(1, ai, codebook);

    % Finding speech_frame components in the filtered codebook
    % taking into account the transient stored in the internal
    % variables of the synthesis filter
    ringing = filter(1, ai, zeros(frame_shift,1), z_synt);
    signal = speech_frame - ringing;
    [gains, indices] = find_Nbest_components(signal, ...
        codebook_filt, N_components);

    % Generating the corresponding excitation as a weighted sum
    % of codebook vectors
    excitation = codebook(:,indices)*gains;

    % Synthesizing CELP speech, and keeping track of the
    % synthesis filter internal variables
    [synt_frame, z_synt] = filter(1, ai, excitation, z_synt);
    synt_speech_CELP=[synt_speech_CELP;synt_frame];

end
```

Notice this anaylsis-synthesis simulation is implemented as mentioned in Section 1.2.4: as an adaptive vector quantization system. This is done by passing the whole codebook through the synthesis filter, for each new frame, and searching for the best linear decomposition of the speech frame in terms of filtered codebook sequences.

Notice also our use of `ringing`, which stores the natural response of the synthesis filter due to its non-zero internal variables. This response should not be taken into account in the adaptive VQ.

The resulting synthetic speech sounds more natural than in LPC10. Plosives are much better rendered, and voiced sounds are no longer buzzy, but speech sounds a bit noisy. Notice that pitch and V/UV estimation are no longer required.

One can see that the closed loop optimization leads to excitation frames which can somehow differ from the LP residual, while the resulting synthetic speech is similar to its original counterpart (Fig. 1.22).

In the above script, though, each new frame was processed independently of past frames. Since voiced speech is strongly self-correlated, it makes sense to incorporate a long-term prediction filter in cascade with the LPC (short-term) prediction filter. In the example below, we can reduce the number of stochastic components from 10 to 5, while still increasing speech quality thanks to long-term prediction.

```
N_components= 5;  % number of codebook components per frame
```

Since CELP excitation frames are only 5 ms long, we store them in a 256 samples circular buffer (i.e. a bit more than 30 ms of speech) for finding the best long-term prediction delay in the range [0-256] samples.

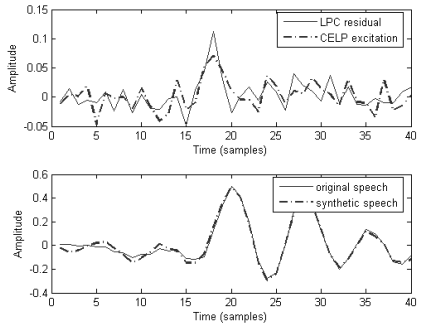

Fig. 1.22 CELP analysis-synthesis of frame #140. Top: CELP excitation compared to linear prediction residual. Bottom: CELP synthetic speech compared to original speech (frame 140)

```
LTP_max_delay=256; % maximum long-term prediction delay
excitation_buffer=zeros(LTP_max_delay+frame_shift,1);
```

Finding the delay itself (inside the frame-based loops) is achieved in a way which is very similar to finding the *N* best stochastic components in our previous example: we create a long-term prediction codebook, pass it through the synthesis filter, and search for *the* best excitation component in this filtered codebook.

```
% Building the long-term prediction codebook and filtering it
for j = 1:LTP_max_delay
    LTP_codebook(:,j) = excitation_buffer(j:j+frame_shift-1);
end
LTP_codebook_filt = filter(1, ai, LTP_codebook);

% Finding the best predictor in the LTP codebook
ringing = filter(1, ai, zeros(frame_shift,1), z_synt);
signal = speech_frame - ringing;
[LTP_gain, LTP_index] = find_Nbest_components(signal, ...
    LTP_codebook_filt, 1);

% Generating the corresponding prediction
LT_prediction= LTP_codebook(:,LTP_index)*LTP_gain;
```

Stochastic components are then searched *in the remaining signal* (i.e., the original signal minus the long-term predicted *signal*).

```
% Finding speech_frame components in the filtered codebook
% taking long term prediction into account
signal = signal - LTP_codebook_filt(:,LTP_index)*LTP_gain;
```



```
[gains, indices] = find_Nbest_components(signal, ...
    codebook_filt, N_components);
```

The final excitation is computed as the sum of the long-term predicted *excitation*.

```
excitation = LT_prediction + codebook(:,indices)*gains;
```

As can be seen on Fig. 1.23, the resulting synthetic speech is still similar to the original one, notwithstanding the reduction of the number of stochastic components.

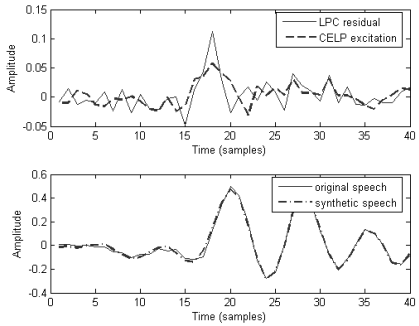

Fig. 1.23 CELP analysis-synthesis of frame #140, with long-term prediction and only 5 stochastic components. Top: CELP excitation compared to linear prediction residual. Bottom: CELP synthetic speech compared to original speech (frame 140)

While the search for the best components in the previous scripts aims at minimizing the energy of the difference between original and synthetic speech samples, it makes sense to use the fact that the ear will be more tolerant to this difference in parts of the spectrum that are louder and vice versa. This can be achieved by applying a perceptual filter to the error, which enhances spectral components of the error in frequency bands with less energy, and vice-versa (Fig. 1.24).

In the following example, we still decrease the number of components from 5 to 2, with the same overall synthetic speech quality.

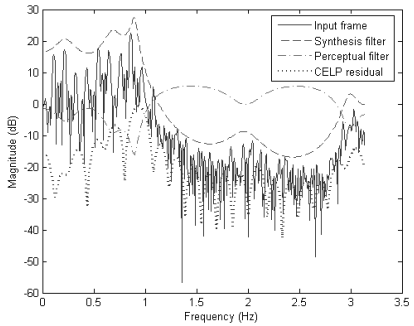

Fig. 1.24 CELP analysis-synthesis of frame #140: the frequency response of the perceptual filter approaches the inverse of that of the synthesis filter. As a result, the spectrum of the CELP residual somehow follows that of the speech frame.

```
N_components= 2;  % number of codebook components per frame
```

We will apply perceptual filter $A(z)/A(z/\gamma)$ to the input frame, and filter $1/A(z/\gamma)$ to the stochastic and long-term prediction codebook vectors [12]. We will therefore need to handle their internal variables.

```
gamma = 0.8;   % perceptual factor
z_inv=zeros(10,1);  % inverse filter
z_synt=zeros(10,1); % synthesis filter
z_gamma_s=zeros(10,1); % perceptual filter for speech
z_gamma_e=zeros(10,1); % perceptual filter for excitation
```

Finding the coefficients of $A(z/\gamma)$ is easy.

```
ai_perceptual = ai.*(gamma.^(0:(length(ai)-1)) );
```

One can then filter the input frame and each codebook.

```
% Passing the central 5ms of the input frame through
% A(z)/A(z/gamma)
[LP_residual, z_inv] = filter(ai, 1, speech_frame, z_inv);
[perceptual_speech, z_gamma_s] = filter(1, ...
                    ai_perceptual, LP_residual, z_gamma_s);

% Filtering both codebooks
LTP_codebook_filt = filter(1, ai_perceptual, LTP_codebook);
```

[12] In the previous examples, the input frame was not perceptually filtered, and codebooks were passed through the synthesis filter 1/A(z).


```
codebook_filt = filter(1, ai_perceptual, codebook);
```

The search for the best long-term predictor is performed as before, except that the perceptually filtered speech input is used as the reference from which to find codebook components.

```
% Finding the best predictor in the LTP codebook
ringing = filter(1, ai_perceptual, ...,
          zeros(frame_shift,1), z_gamma_e);
signal = perceptual_speech - ringing;
[LTP_gain, LTP_index] = find_Nbest_components(signal, ...
    LTP_codebook_filt, 1);

% Generating the corresponding prediction
LT_prediction= LTP_codebook(:,LTP_index)*LTP_gain;

% Finding speech_frame components in the filtered codebook
% taking long term prediction into account
signal = signal - LTP_codebook_filt(:,LTP_index)*LTP_gain;
[gains, indices] = find_Nbest_components(signal, ...
    codebook_filt, N_components);
```

Last but not least, one should not forget to update the internal variables of the perceptual filter applied to the excitation.

```
[ans, z_gamma_e] = filter(1, ai_perceptual, excitation, ...
    z_gamma_e);
```

While using less stochastic components than in the previous example, synthetic speech quality is maintained, as revealed by listening. The synthetic speech waveform also looks much more similar to original speech than its LPC10 counterpart (Fig. 1.25).

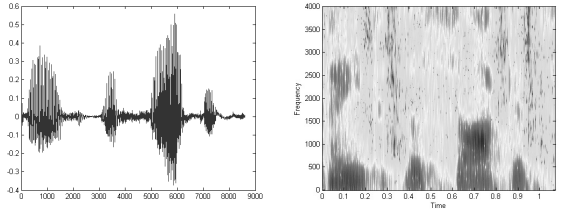

Fig. 1.25 CELP speech

One can roughly estimate the corresponding bit-rate. Assuming 30 bits are enough for the prediction coefficients and each gain factor is quantized on 5 bits, we have to send for each frame: 30 bits [ai] + 7 bits [LTP index]

+ 5 bits [LTP gain] + 2 [stochastic components] *(9 bits [index] + 5 bits [gain]) = 70 bits every 5 ms, i.e. 14 kbits/s.

Notice that G729 reaches a bit rate as low as 8 kbits/s by sending prediction coefficients only once every four frame.

1.3 Going further

Various tools and interactive tutorials on LP modeling of speech are available on the web (see Fellbaum 2007, for instance).

MATLAB code by A. Spanias for the LPC10e coder can be found on the web (Spanias and Painter 2002).

Another interesting MATLAB-based project on LPC coding, applied to wideband speech this time, can be found on the dspexperts.com website (Kahn and Kashif 2003).

D. Ellis provides interesting MATLAB-based audio processing examples on his web pages (Ellis, 2006), among which a sinewave speech analysis/synthesis demo (including LPC), and a spectral warping of LPC demo.

For a broader view of speech coding standards, one might refer to (Woodard 2007), or to the excellent book by Goldberg and Riek (2000).

1.4 Conclusion

We now understand how every cell phone solves a linear system of 10 equations in 10 unknowns every 20 milliseconds, which is the basis of the estimation of the LP model through Yule-Walker equations. The parameters that are actually sent from one cell phone to another are: vocal tract coefficients related to the frequency response of the vocal tract and source coefficients related to the residual signal.

The fact that the vocal tract coefficients are very much related to the geometric configuration of the vocal tract for each frame of 10 ms of speech call for an important conclusion: cell phones, in a way, transmit a picture of our vocal tract rather than the speech it produces.

In fact, the reach of LP speech modeling goes far beyond the development of cell phones. As shown by Gray (2006), its history is intermixed with that of Arpanet, the ancestor of Internet.

1.5 References

Atal BS, Remde JR (1982) A New Model LPC Excitation for Producing Natural Sounding Speech at Low Bit Rates. In: Proc. ICASSP'82, pp 614-617

de la Cuadra P (2007) Pitch detection methods review [online] Available: http://www-ccrma.stanford.edu/~pdelac/154/m154paper.htm [20/2/1007]

Ellis D (2006) Matlab audio processing examples [Online] Available: http://www.ee.columbia.edu/%7Edpwe/resources/matlab/ [20/2/2007]

Fant G (1970) Acoustic Theory of Speech Production. Mouton, The Hague

Fellbaum K (2007) Human Speech Production Based on a Linear Predictive Vocoder [online] Available: http://www.kt.tu-cottbus.de/speech-analysis/ [20/2/2007]

Gray RM (2006) Packet speech on the Arpanet: A history of early LPC speech and its accidental impact on the Internet Protocol. [Online] Available: http://www.ieee.org/organizations/society/sp/ Packet_Speech.pdf [20/2/2007]

Goldberg RG, Riek L (2000) Speech coders. CRC Press: Boca Raton, FL.

Hess W (1992) Pitch and Voicing Determination. In: Advances in Speech Signal Processing, S. Furui, M. Sondhi, eds., Dekker, New York, pp 3-48

Khan A, Kashif F (2003) Speech Coding with Linear Predictive Coding (LPC) [Online] Available: http://www.dspexperts.com/dsp/projects/lpc [20/2/2007]

Kroon P, Deprettere E, Sluyter R (1986) Regular-pulse excitation – A novel approach to effective and efficient multipulse coding of speech. IEEE Transactions on acoustics, speech, and signal processing 34-5:1054- 1063

McCree AV, Barnwell TP (1995) A mixed excitation LPC vocoder model for low bit rate speech coding. IEEE Transactions on Speech and Audio Processing, 3-4:242–250

Matsumoto J, Nishiguchi M, Iijima K (1997) Harmonic Vector Excitation Coding at 2.0 kbps. In: Procs of the IEEE Workshop on Speech Coding, pp 39–40

NATO (1984) Parameters and coding characteristics that must be common to assure interoperability of 2400 bps linear predictive encoded speech. NATO Standard STANAG-4198-Ed1

Quatieri T (2002) Discrete-Time Speech Signal Processing: Principles and Practice. Prentice-Hall, Inc.: Upper Saddle River, NJ

Rabiner LR, Schafer RW (1978) Digital Processing of Speech Signals. Prentice-Hall, Inc.: Englewood Cliffs, NJ

Salami R, Laflamme C, Adoul J-P, Kataoka A, Hayashi S, Moriya T, Lamblin C, Massaloux D, Proust S, Kroon P, Shoham, Y (1998) Design and description of CS-ACELP: a toll quality 8 kb/s speech coder, IEEE Transactions on Speech and Audio Processing 6–2, 116-130

Schroeder MR, Atal B (1985) Code-Excited Linear Prediction(CELP): High Quality Speech at Very Low Bit Rates. In: Proc. IEEE ICASSP-85, pp 937-940

Spanias A, Painter T (2002) Matlab simulation of LPC10e vocoder [Online] Available: http://www.cysip.net/lpc10e_FORM.htm [19/2/2007]

Woodard J (2007) Speech Coding [Online] Available: http://www-mobile.ecs.soton.ac.uk/speech_codecs/ [20/2/2007]

Chapter 2

How are bits played back from an audio CD?

An audio digital-to-analog converter adds noise to the signal, by requantizing 16-bit samples to one-bit. It does it…on purpose.

T. Dutoit (°), R. Schreier (*)

(°) Faculté Polytechnique de Mons, Belgium
(*) Analog Devices, Inc., Toronto, Canada

Loading a CD player with one's favorite CD has become an ordinary action. It is taken for granted that the stream of 16-bits digital information it contains can easily be made available to our ears, i.e., in the analog world in which we live. The essential tool for this is the Digital-to-Analog converter (DAC).

In this chapter we will see that, contrary to what might be expected, many audio DACs (including those used in CD and MP3 players, for instance, or in cell phones), first requantize the 16-bit stream into a *one-bit* stream[1] with very high sampling frequency, using a signal processing concept known as delta-sigma[2] (ΔΣ) modulation, and then convert the resulting bipolar signal back to an audio waveform.

The same technique is used in ADCs, for digitizing analog waveforms. It is also the heart of the Direct Stream Digital (DSD) encoding system, implemented in Super Audio CDs (SACDs).

[1] In practice, one-bit ΔΣ DACs have been superseded by multiple-bit DACs, but the principle remains the same.
[2] Sometimes also referred to as *sigma-delta*.

2.1 Background – Delta-sigma modulation

An *N*-bit DAC converts a stream of discrete-time linear PCM[3] samples of *N* bits at sample rate *Fs* to a continuous-time voltage. This can be achieved in many ways. Conventional DACs (2.1.2) directly produce an analog waveform from the input PCM samples. Oversampling DACs (2.1.3) start by increasing the sampling frequency using digital filters and then make use of a conventional DAC with reduced design constraints. Adding noise-shaping makes it possible to gain resolution (2.1.4). Delta-sigma DACs (2.1.5) oversample the input PCM samples, and then re-quantize them to a 1-bit data stream, whose low-frequency content is the expected audio signal.

Before examining these DACs, we start with a review of uniform quantization (2.1.1), as it will be used throughout the Chapter.

2.1.1 Uniform quantization: bits vs. SNR

Quantization lies at the heart of digital signal processing. An *N* bit uniform quantizer maps each sample $x(n)$ of a signal to one out of 2^N equally spaced values $X(n)$ in the interval $(-A,+A)$, separated by the quantization step $q=2A/2^N$. This operation (Fig. 2.1) introduces an error $e(n)$:

$$e(n) = X(n) - x(n) \qquad -q/2 \le e(n) \le q/2 \tag{2.1}$$

If the number of bits is high enough and the input signal is complex, the quantization error is equivalent to uniform white noise in the range[$(-q/2, +q/2)$]. It is easy to show that its variance is then given by:

$$\sigma^2_{ee} = \frac{q^2}{12} \tag{2.2}$$

The main result of uniform quantization theory, which can be found in most signal processing textbooks, is the standard "1bit = 6dB" law, which gives the expression of the signal-to-quantization-noise ratio:

$$SNR(dB) = 10 * \log_{10}\left(\frac{\sigma_{xx}{}^2}{\sigma_{ee}{}^2}\right) \tag{2.3}$$

as a function of *N* and *A,* in the absence of saturation:

[3] PCM stand for Pulse Code Modulated.

$$SNR(dB) = 6.02\ N + 4.77 + 20\ \log_{10}(\Gamma) \qquad (\Gamma = A/\sigma_{xx}) \qquad (2.4)$$

where Γ is the load factor defined as the saturation value of the quantizer normalized by the standard deviation of the input signal.

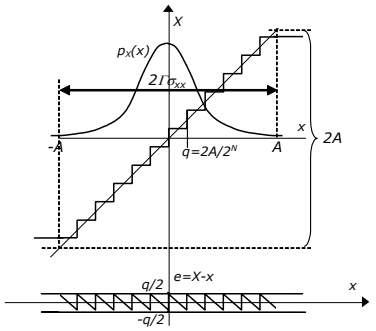

Fig. 2.1 Uniform quantization[4]

When the amplitude of the input signal approaches the quantization step, the quantization error becomes correlated with the signal. If the signal itself is not random, the quantization error can then be heard as non-linear audio distortion (rather than as additive noise).

This can be avoided by *dithering*, which consists of adding real noise to the signal before quantizing it. It has been shown that a triangular white noise (i.e. white noise with a triangular probability density function) in the range [$-q$,$+q$] is the best dither: it decorrelates the quantization noise with the signal (it makes the mean and variance of the quantization noise independent of the input signal; see Wannamaker 1997) while adding the least possible noise to the signal. Such a noise is easy to obtain by summing two independent, uniform white noise signals in the range [$-q/2$,$+q/2$], since the probability density function of the sum of two independent random variables is the convolution of their respective pdfs. As Wannamaker puts it:

[4] We actually show a *mid-rise* quantizer here, which quantizes a real number x into (floor(x/q)+0.5)*q. Mid-thread quantizers, which compute floor(x/q+0.5)*q, are also sometimes used (see Section 3.2.5 for instance).

"*Appropriate dithering prior to (re)quantization is as fitting as appropriate anti-aliasing prior to sampling – both serve to eliminate classes of signal-dependent errors*".

2.1.2 Conventional DACs

A conventional DAC uses analog circuitry (R/2R ladders, *thermometer* configuration, and others; see for instance Kester and Bryant 2003) to transform a stream of PCM codes $x(n)$ (at the sampling frequency F_s) into a staircase analog voltage $x^*(t)$ in which each stair lasts $T_s=1/F_s$ seconds and is a linear image of the underlying PCM code sequence. Staircase signal $x^*(t)$ is then smoothed with an analog lowpass filter $S(f)$ (Fig. 2.2), which suppresses the spectral images.

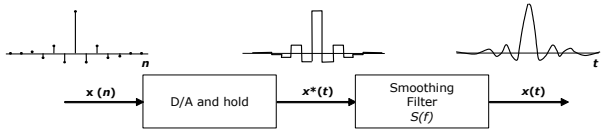

Fig. 2.2 Conventional DAC

The first operation can be seen as convolving a sequence of Dirac pulses $x^+(t)$ obtained from PCM code with a rectangular wave of length T_s, hence filtering the Dirac pulses with the corresponding lowpass filter (Fig. 2.3). The second operation completes the smoothing job.

Conventional DACs have several drawbacks. First, they require high precision analog components, and are very vulnerable to noise and interference. In a 16-bit DAC with 3V reference voltage, for instance, one half of the *least significant bit* corresponds to 2^{-17} 3V=23 μV. What is more, they impose hard constraints on the design of the analog smoothing filter, whose transition band must fit within [F_m, F_s-F_m] (where F_m is the maximum frequency of the signal; Fig. 2.3), so as to efficiently cancel spectral images.

2.1.3 Oversampling DACs

An oversampling DAC first performs digital K times oversampling of the input signal $x(n)$, by first inserting K-*1* zeros between each sample of $x(n)$,

and then applying digital lowpass filtering with passband equal to [0, F_s/K] (Fig. 2.4). The resulting signal, $x(n/K)$, is then possibly requantized on N' bits (with $N'<N$), by simply keeping the N' most significant bits from each interpolated sample, and the requantized signal $x'(n/K)$ is sent to a conventional DAC clocked at $K*F_s$ Hz, with N' bits of resolution.

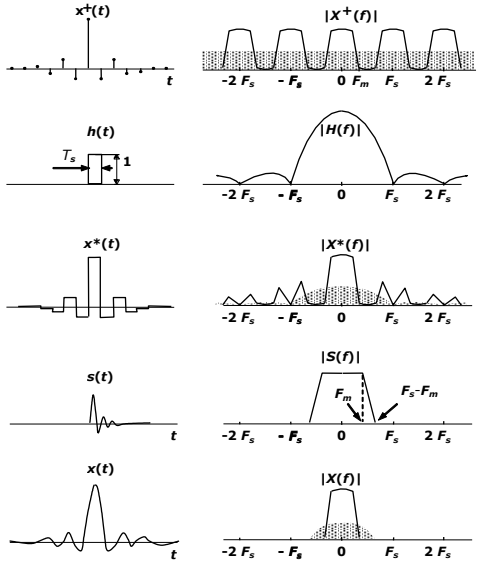

Fig. 2.3 Digital-to-Analog conversion seen as double filtering: $x^+(t)$ is convolved with $h(t)$ to produce $x^*(t)$, which is smoothed by $S(f)$. Quantization noise is shown as superimposed texture.

In principle, requantizing to less than the initial N bits lowers the SNR by 6 dB for each bit lost. However, although the variance of quantization noise $e'(n)$ generated by N'-bit requantization is higher than that of the initial N-bit quantization noise $e(n)$, it is now spread over a (K times) larger

frequency range. Its power spectral density (PSD, in V²/Hz) is thus given by:

$$S_{e'e'}(f) = \frac{\sigma_{e'e'}^2}{K\,F_s} \tag{2.5}$$

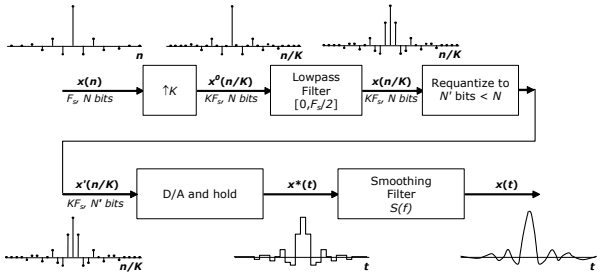

Fig. 2.4 Oversampling DAC (shown here with oversampling ratio set to 2, for convenience).

And, since only a fraction of this PSD (typically, $1/K$) will eventually appear in the analog output, thanks to the action of the smoothing filter, the effective SNR is higher than its theoretical value. In practice, each time a signal is oversampled by a factor 4, its least significant bit (LSB) can be ignored. As a matter of fact, the 6 dB increase in SNR is compensated by a 6dB decrease due to the fact that only one fourth of the variance of the new quantization noise is in the range [0, $F_s/2$][5].

Physically speaking, this is perfectly understandable: successive requantization noise samples $e'(n)$ produced at KF_s are independent random variables with variance equal to $q'^2/12 > q^2/12$. The lowpass smoothing filter performs a weighted average of neighboring samples, thereby reducing their variance.

This effect is very important in practice, as it allows a bad resolution DAC (N' bits<N) to produce high resolution signals.

As a result of oversampling, the smoothing filter is also allowed a larger transition bandwidth [0, KF_s-F_m] (Fig. 2.5). What is more, the same low-

[5] Strictly speaking, in [-$F_s/2$, $F_s/2$]. In this discussion, we only consider positive frequencies, for convenience.

pass filter can now be used for a large range or values for Fs (as required for the sound card of a computer, for instance).

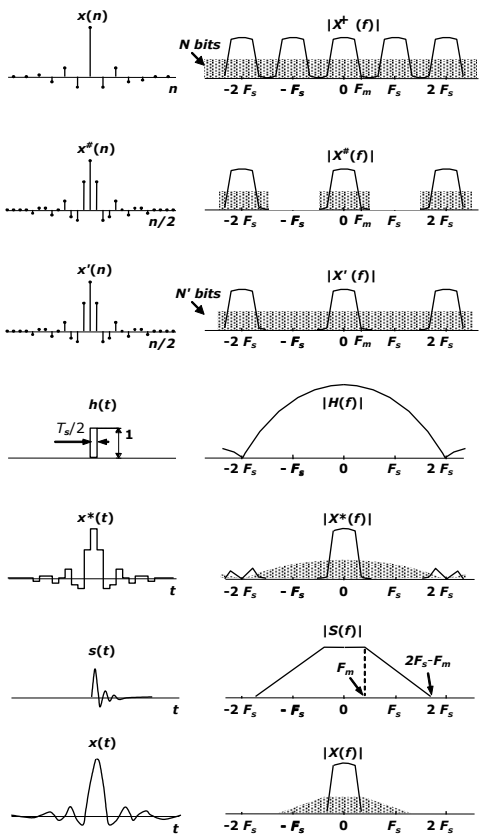

Fig. 2.5 Oversampling (shown here with an oversampling factor of 2) prior to Digital-to-Analog conversion. Quantization and requantization noise are shown as superimposed texture.

2.1.4 Oversampling DACs – Noise shaping

Oversampling alone is not an efficient way to obtain many extra bits of resolution: gaining B bits requires an oversampling ratio of 4^B, which quickly becomes impractical. An important improvement consists of performing *noise shaping* during requantization. Instead of keeping N' bits from each interpolated sample $x(n)$, a noise shaping quantizer implements a negative feedback loop between $x(n)$ and $x'(n)$ (Fig. 2.6),whose effect is to push the PSD of the requantization noise towards frequencies far above $F_s/2$ (up to K*$F_s/2$), while keeping the PSD of the signal untouched. As a result, the effective SNR is further increased (Hicks 1995).

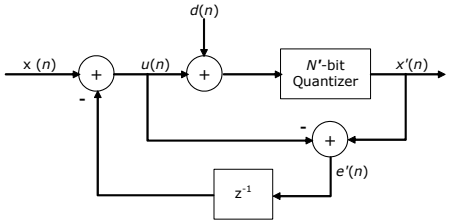

Fig. 2.6 First-order noise shaping (re)quantizer

As a matter of fact, we have:

$$\begin{aligned} U(z) &= X(z) - z^{-1}(X'(z) - U(z)) \\ &= \frac{X(z) - z^{-1}X'(z)}{1 - z^{-1}} \end{aligned} \tag{2.6}$$

and since the combined effect of dithering and quantization is to add some white quantization noise $e'(n)$ to $u(n)$:

$$\begin{aligned} X'(z) &= U(z) + E'(z) \\ &= \frac{X(z) - z^{-1}X'(z)}{1 - z^{-1}} + E'(z) \\ &= X(z) + (1 - z^{-1})E'(z) \end{aligned} \tag{2.7}$$

which shows that the output of the noise-shaping requantizer is the input signal plus the first derivative of the white quantization noise $e'(n)$ produced by requantization. This effectively results in colored quantization noise $c(n)=e'(n)-e'(n-1)$, with most of its PSD in the band $[F_s/2, KF_s/2]$ (see Fig. 2.7, to be compared to Fig. 2.5), where it will be filtered out by the smoothing filter. The noise shaping function $(1-z^{-1})$ being of first order, this configuration is termed as a first-order noise shaping cell.

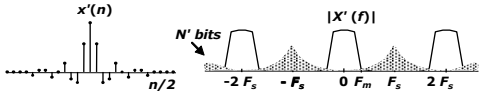

Fig. 2.7 The effect of noise shaping combined with oversampling (by a factor 2) on signal $x'(n)$ at the output of the quantizer

Noise shaping does increase the power of quantization noise, as the variance of colored noise $c(n)$ is given by:

$$\begin{aligned} \sigma_{cc'}^2 &= \frac{\sigma_{e'e'}^2}{K F_s} \int_{-KF_s/2}^{-KF_s/2} |1 - e^{-j2\pi f}|^2 \, df \\ &= \frac{\sigma_{e'e'}^2}{K F_s} \left(\int_{-KF_s/2}^{-KF_s/2} (1 + |e^{-j\theta}|^2) \, df \right. \\ &= 2\sigma_{e'e'}^2 \end{aligned} \tag{2.8}$$

But again, since this variance is mostly pushed in the $[F_s/2, KF_s/2]$ band, the effective SNR can be lowered (Fig. 2.8).

This technique makes it possible to gain 1 bit every time the signal is oversampled by a factor 2. It was used in early CD players, when only 14-bit hardware D/A converters were available at low cost. By combining oversampling and noise shaping (in the digital domain), a 14-bit D/A converter was made comparable to a 16-bit D/A converter.

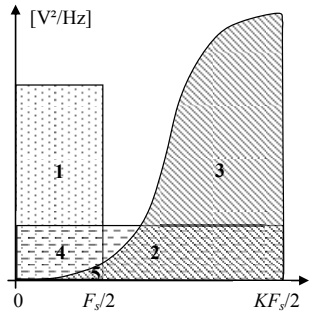

Fig. 2.8 A comparison of quantization noise power density functions for the same number of bits. 1: With a conventional DAC; effective noise variance = area 1; 2: With an oversampling DAC; effective noise variance = area 4; 3: With an oversampling DAC using noise shaping; effective noise variance = area 5.

2.1.5 Delta-sigma DACs

The delta-sigma architecture is the ultimate extension of the oversampling DAC and is used in most voiceband and audio signal processing applications requiring a D/A conversion. It makes use of a very high oversampling ratio, which makes it possible to requantize the digital signal to 1 bit only. This 1-bit signal is then converted to a purely bipolar analog signal by the DAC, whose output switches between equal positive and negative reference voltages (Fig. 2.9).

The bipolar signal is sometimes referred to as "pulse density modulated" (PDM), as the density of its binary transitions is a function of the amplitude of the original signal.

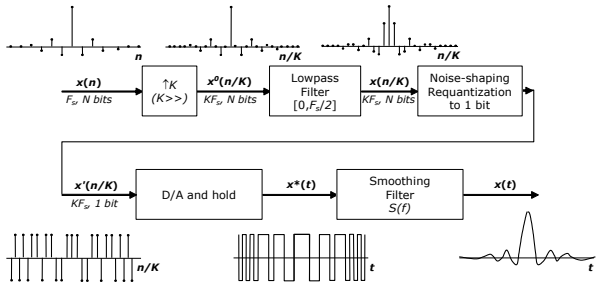

Fig. 2.9 Delta-Sigma DAC (shown here with oversampling ratio set to 2, for convenience; in practice much higher ratios are used).

In CD and MP3 players, this implies a gain of 15 bits of resolution. Improved noise shaping is therefore required, such as second order noise shaping cells (whose noise shaping function is $(1-z^{-1})^2$) or cascades of first-order noise shaping cells (termed as MASH: Multi-stage noise shaping; Matsuya *et al.* 1987). Deriving a general noise-shaping quantizer with noise shaping function $H(z)$ from that of Fig. 2.6 is easy: one simply needs to replace the delay by $1-H(z)$ (Fig. 2.10).

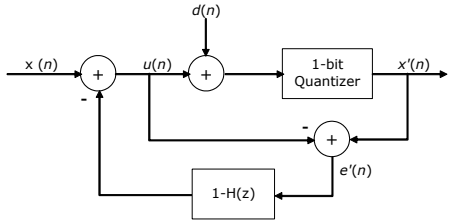

Fig. 2.10 General digital delta-sigma modulator

The influence of the oversampling ratio and of the order of the noise shaping filter on the noise power in the signal bandwidth is given in Fig. 2.11.

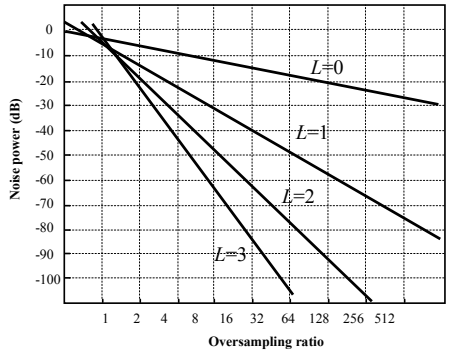

Fig. 2.11 Power of the noise in the signal bandwidth as a function of the oversampling ratio, for various noise shaping orders L. 0 dB of noise level corresponds to that of PCM sampled at the Nyquist rate. All quantizers use the same level spacing. (after Schreier and Temes 2005)

The very limited bandwidth allowed for the interpolation filter ([0, $F_s/2$] at a sampling frequency of KF_s) theoretically implies a very high order for its digital implementation (especially if it is synthesized as a linear phase FIR filter), i.e. many additions and multiplications per sample. Oversampling is therefore systematically performed in two steps: a first step which does most of the oversampling and uses a *comb filter* (or cascades thereof), for the interpolation, followed by a second step competing with lower oversampling ratio and a more efficient interpolation filter. A comb filter of order N is an FIR filter will all coefficients set to 1:

$$H_N(z) = \sum_{n=0}^{N-1} z^{-n} = \frac{1-z^{-N}}{1-z^{-1}} \tag{2.9}$$

This filter has linear phase, while being straightforwardly implemented as:

$$y(n) = x(n) - x(n-N) + y(n-1) \tag{2.10}$$

which implies no multiplication. Comb filters, however, cannot provide very efficient frequency responses (for an example, see Fig. 2.16-left). Their transition band is not sharp. This is counterbalanced by the second interpolation stage, which produces a sharp cut-off and additionally compensates for the frequency droop of the comb filter around $F_s/2$.

Summarizing: delta-sigma DACs are a wonderful example of mixed A-D signal processing constraints. Most of their load is in the digital domain, while the analog output smoothing filter usually reduces to a simple RC lowpass. These converters can thus be fabricated on a wide range of IC processes, which implies low cost and robustness to time and temperature drifts. This is achieved at the expense of speed: since the hardware has to operate at the high oversampling rate, sigma-delta converters are usually limited to sampling rates below 1 MHz.

From 1990 to 2000, the consumer audio industry has produced a large number of 1-bit, sigma-delta based converters, mainly because this technique leads to cheaper manufacturing. It was argued, however, that 1-bit sigma-delta converters were not suitable for high-quality audio applications (Lipshitz and Vanderkooy 2001), as they only allow partial dithering to be performed.[6] Most audio DACs made since 2000 implement *multi-bit* sigma-delta converters, which take the best of both the 1-bit and the 20+ -bit worlds (go to the Analog Devices web page, for instance, and search for "audio DAC").

2.2 MATLAB proof of concept: ASP_audio_cd.m

In this Section, we will first revise the basics of uniform quantization (2.2.1), including the important part played by dithering (2.2.2). We will then compare the internals of a conventional DAC (2.2.3) to those of more advanced DACs, using oversampling (2.2.4), noise-shaping (2.2.5), and delta-sigma modulation (2.2.6).

2.2.1 Uniform quantization

In order to perform D/A conversion, we first need to create a PCM signal. We will first check the main results of uniform quantization theory, which we will use later.

[6] This led to an interesting controversy in several papers published in conventions of the Audio Engineering Society.

Let us first generate 8000 samples of uniform white noise with zero mean, which we will assume to have been sampled at F_s=8000 Hz. For convenience, we set its variance to 1 (0 dB) by imposing its range to [−*peak*, +*peak*], with $peak = \sqrt{12}/2 \approx 1.73$. This is confirmed on the power spectral density of the signal (Fig. 2.12).

```
signal_std=1;
peak=sqrt(12*signal_std)/2;
signal=(rand(1,8000)-0.5)*2*peak;

pwelch(signal,[],[],[],2); [7]
```

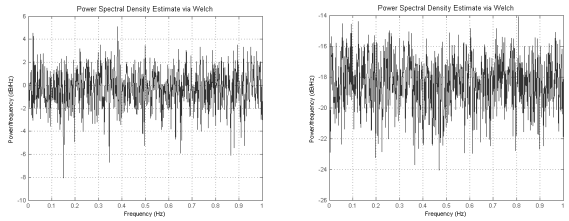

Fig. 2.12 Power spectral density of the input signal (left) and of the 6-bit quantized error (right)

Quantizing this signal uniformly on *N* bits is easy. Let us choose *N*=3 for convenience. We use mid-rise quantization (hence the `+q/2`), best suited for an even number of quantization steps.

```
N=3;
quantizer_saturation=peak;
q=(2*quantizer_saturation)/2^N;
signal_quantized=floor(signal/q)*q+q/2; % mid-rise
error=signal-signal_quantized;
```

The quantization noise is also white, as revealed by power spectral density estimation. The variance of the noise can thus be read (in dB) on the PSD plot (Fig. 2.12). From the load factor (i.e., the saturation value of the quantizer, normalized by the standard deviation of the signal), we can

[7] MATLAB's `pwelch` function shows the PSD in the range [0, F_s/2]. The PSD value it shows are such that their integral over [0, F_s/2] equals the variance of the signal. Claiming F_s=2 as an input argument therefore shows the variance as the average PSD over [0, 1].

compute the theoretical SNR due to quantization. It matches to the SNR computed from the samples, and corresponds to the variance of the noise (with opposite sign) since the variance of the signal is 0dB.

MATLAB function involved:

- `snr = snr(signal, signal_plus_noise, max_shift, showplot)` returns the signal-to-noise ratio computed from the input signals. `Max_shift` gives the maximum time-shift (in samples) between `signal` and `signal_plus_noise`. The actual time-shift (obtained from the maximum of the cross-correlation between both signals) is taken into account to estimate the noise. If `showplot` is specified, then the `signal`, `signal_plus_noise`, and `error` are plotted, and the SNR is printed on the plot.

```
pwelch(error,[],[],[],2);
load_factor=quantizer_saturation/signal_std;
snr_theory=6.02*N + 4.77 - 20*log10(load_factor)
snr_measured=snr(signal,signal_quantized,0)
```

```
snr_theory =  18.0588

snr_measured =  18.1119
```

2.2.2 Dithering

Let us apply this to a sine wave with fundamental frequency f_0=200 Hz, sampled at F_s=8000 Hz, and set its variance to 1 (0 dB) by imposing its peak to $\sqrt{2}$.

```
signal_std=1;
peak=sqrt(signal_std*2);
signal=peak*sin(2*pi*200*(0:1/8000:2));
var_signal_dB=10*log10(var(signal))
```

```
var_signal_dB = -2.8930e-015
```

When we quantize it uniformly to 3 bits in the range [-2,+2], we notice that the quantization error does not look like real noise (Fig. 2.13).

```
N=3;
quantizer_saturation=2;
q=2*quantizer_saturation/2^N;
signal_quantized=floor(signal/q)*q+q/2; % quantization
error=signal-signal_quantized;

plot(signal(1:50),'-.'); hold on;
plot(error(1:50));hold off
```

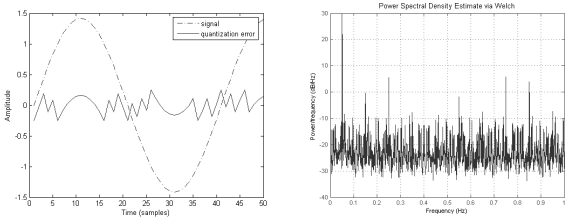

Fig. 2.13 Quantization error (left) and power spectral density of the quantized signal (right), without dithering. Frequencies are normalized with respect to $F_s/2$.

Harmonics of the original frequency appear in the spectrum of the quantized signal (Fig. 2.13) and sound very unpleasant. The SNR is not directly readable on the PSD plot, because of these harmonics

```
snr_quantized=snr(signal,signal_quantized,0)
pwelch(signal_quantized,[],[],[],2);
```

```
snr_quantized = 16.9212
```

In such a case it may be interesting to whiten the quantization error by adding real noise to it. This operation is termed as dithering.

Let us add a dithering noise with triangular pdf in the range [$-q$, $+q$], i.e. two times the quantization step. Such a noise is easily obtained by adding two uniform white noises in the range [$-q/2$, $q/2$].

```
dither=(rand(size(signal))-0.5)*q+(rand(size(signal))-0.5)*q;
signal_dithered=signal+dither;
```

The resulting quantization error looks indeed more noise-like[8].

```
signal_dithered_quantized=floor(signal_dithered/q)*q+q/2;
error_dithered_quantized=signal-signal_dithered_quantized;

plot(signal(1:50),'-.'); hold on;
plot(error_dithered_quantized(1:50));hold off
```

[8] Notice we do not account for quantizer saturation, which may occur when dither is added to a signal which approaches the saturation levels of the quantizer. This is note the case in our example.

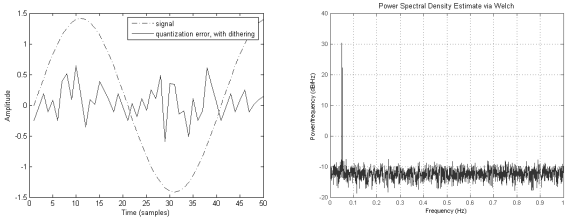

Fig. 2.14 Quantization error (left) and power spectral density of the quantized signal (right), with dithering.

The power spectral density of the quantized signal now appears as white noise added to the initial sine wave.

```
pwelch(signal_dithered_quantized,[],[],[],2);
```

Dithering clearly degrades the SNR (by about 4.8dB), but results in perceptually more acceptable quantization error.

```
load_factor=quantizer_saturation/signal_std;
snr_theory=6.02*N + 4.77 - 20*log10(load_factor)
snr_dithered_quantized=snr(signal,signal_dithered_quantized,0)
```

```
snr_theory =  16.8094

snr_dithered_quantized =  12.0562
```

2.2.3 Conventional DAC

A conventional DAC first creates a staircase signal from the sequence of samples, by converting each sample into an analog voltage and holding this voltage for $1/F_s$ seconds. This is called zero-order (analog) interpolation. This operation is critical: the higher the number of bits in the PCM code to convert, the higher the precision required for creating the staircase signal!

We will demonstrate this on a sine wave, with N=6 bits.

MATLAB function involved:

- `signal_quantized = uquantize(signal,N,saturation)` quantizes `signal` uniformly to `N` bits in the range [-saturation, +saturation], using a

mid-rise quantizer and triangular dither with a range of twice the quantization step. Clipping is performed when saturation is reached.

```
signal_std=1;
peak=sqrt(signal_std*2);
signal=peak*sin(2*pi*200*(0:1/8000:2));

N=3;
quantizer_saturation=2;
signal_quantized=uquantize(signal,N,quantizer_saturation);
snr_quantized=snr(signal,signal_quantized,0)
```

```
snr_quantized = 12.0809
```

We can *simulate* the analog zero-order interpolation in the digital domain by working with a much higher sampling frequency than F_s (say, F_s'=10 F_s, i.e. 80 kHz). Zero-order interpolation is then equivalent to inserting 9 zeros in between each sample, and convolving the resulting signal with a sequence of 10 unity samples (Fig. 2.15).[9]

```
signal_pulses=zeros(1,10*length(signal_quantized));
signal_pulses(1:10:10*length(signal))=…
                                    signal_quantized;
hold_impresp=ones(1,10);
signal_staircase=conv(signal_pulses,hold_impresp);

plot(1:10:391, signal_quantized(1:40),'o'); hold on;
plot(signal_staircase(1:400)); hold off;
```

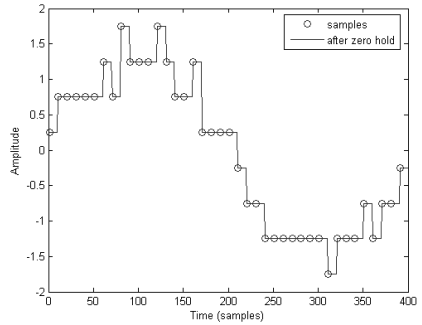

Fig. 2.15 Analog staircase signal after zero-order interpolation

[9] This is actually a very rudimentary form of numerical interpolation.

Notice that zero-order interpolation acts as a lowpass filter, performing a first attenuation of the spectral images of the signal at integer multiples of F_s. This directly affects the spectrum of the resulting "analog" signal. One can clearly see spectral images, weighted by the effect of the interpolation (Fig. 2.16).

```
freqz(hold_impresp,1, 256, 80000);
pwelch(signal_staircase,[],[],[],20);
```

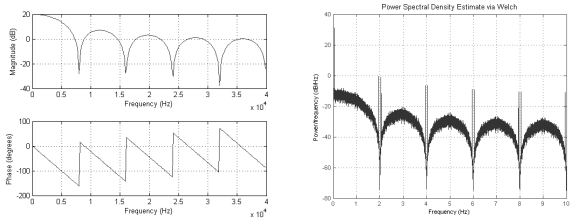

Fig. 2.16 Frequency response of the zero-order analog interpolation (left) and power spectral density of the resulting staircase signal (right)

The DAC then feeds the resulting staircase signal to an analog low-pass smoothing filter, which removes the spectral images due to sampling. The passband of this filter is limited by the maximum frequency of the signal, Fm, and its stopband must not be greater than F_s-F_m. Let us assume that the PCM signal we have is a telephone signal, with F_m=3400 Hz and F_s=8000 Hz, and perform the approximation of the filter with 0.1dB of ripple in the passband and -60 dB in the stopband. We use Chebyshev approximation, so as to keep the order of the filter low.

```
[order,Wn]=cheb1ord(2*pi*3400,2*pi*4600,0.1,60,'s');
[Num_LP,Den_LP]=cheby1(order,0.1,Wn,'s');
zplane(Num_LP,Den_LP);
```

The frequency response of this 12th order filter meets our requirements (Fig. 2.17).[10]

```
freqs(Num_LP,Den_LP);
```

[10] Notice its phase is non-linear, but for audio applications this is not a problem.

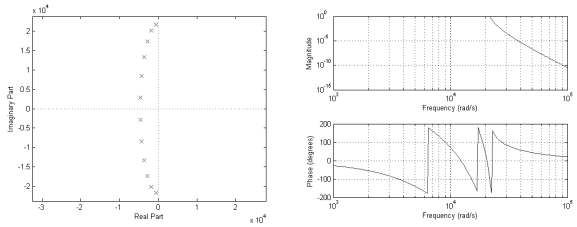

Fig. 2.17 Pole-zero plot and frequency response of the analog smoothing filter

We can now simulate analog filtering by convolving our highly oversampled staircase signal with a sampled version of the impulse response of the LP filter (Fig. 2.18).

MATLAB function involved:

- `y = filters(N,D,x,Fs)` simulates analog filtering of the data in vector `x` with the filter N(s)/D(s) described by vectors `N` and `D` to create the filtered data `y`. Fs is the sampling frequency of the input (and output). Filtering is performed by convolving the input with an estimate of the impulse response of the filter, obtained by partial fraction expansion.

```
analog_output=filters(Num_LP,Den_LP,signal_staircase,80000);
plot(analog_output(1000:2000));
```

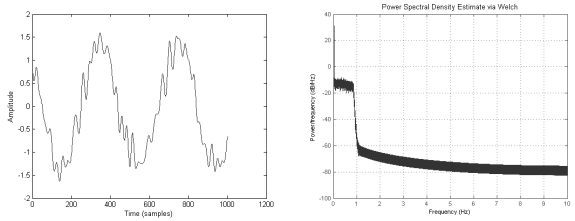

Fig. 2.18 Analog signal produced by the traditional DAC (left) and corresponding power spectral density (right)

The noise level has been reduced by the analog LP filter, as revealed by spectral analysis (Fig. 2.18). The final SNR is a bit higher than what we had before D/A conversion, but this is due to the fact that the zero-order interpolation attenuates the noise at the upper edge of its passband.

Notice, though, that the samples sent to the MATLAB's `soundsc` function are made available to your ears by ... another (real) DAC. One should therefore consider the final quality of this audio sample with care.

```
pwelch(analog_output,[],[],[],20);
signal_sampled_at_10Fs=peak*sin(2*pi*200*(0:1/80000:2));
snr_analog =snr(signal_sampled_at_10Fs,analog_output,400)
                                        % 400=max possible delay
soundsc(analog_output,80000);
```

```
snr_analog =  13.5954
```

2.2.4 Oversampling DAC

By oversampling the digital signal prior to sending it to the analog part of the DAC (which is responsible for creating a staircase analog signal before final analog low-pass filtering), we can broaden the transition band of the analog smoothing filter, thereby strongly decreasing its order (and hardware complexity).

What is more, if we requantize the signal on N-1 bits (1 bit less, i.e. 6 more dBs of total noise power) after multiplying F_s by 4, only one quarter of the resulting power spectral density will contribute to quantization noise in the [0, F_s/2] band; the audible part of the requantization noise power will thus be $10\log_{10}(1/4)$dB (i.e. 6 dB) lower than its total power. Hence, this N-1 requantization step will be felt as a new N-bit quantization.

In other words, dropping one bit from 4 times oversampled digital signals (or k bits for after oversampling by a ratio of k*4) does not do much harm to the signal, while it decreases the required precision of the hardware DAC

Let us check this on our sinusoidal test signal, quantized to 6 bits.

```
signal_std=1;
peak=sqrt(signal_std*2);
signal=peak*sin(2*pi*200*(0:1/8000:2));

N=6;
quantizer_saturation=2;
signal_quantized=uquantize(signal,N,quantizer_saturation);

pwelch(signal_quantized,[],[],[],2);
snr_quantized=snr(signal,signal_quantized,0)
```

```
snr_quantized =  30.1654
```

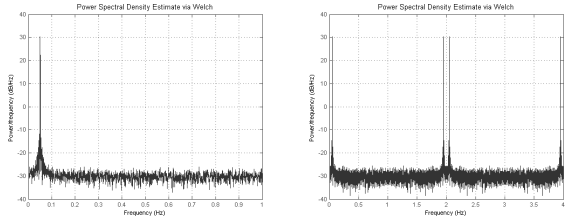

Fig. 2.19 Power spectral density of a sine wave quantized on 6 bits, before (left) and after four times upsampling (right).

We now upsample this quantized signal by a factor of 4, by adding 3 zeros in between its samples. The resulting signal has a sampling frequency F_s'=32000 Hz (Fig. 2.19). The amplification of samples by 4 is required for the final sine wave to have the same peak-to-peak level as the original.

```
signal_pulses=zeros(1,4*length(signal_quantized));
signal_pulses(1:4:4*length(signal_quantized))=4*signal_quantiz
ed;

pwelch(signal_pulses,[],[],[],8);
```

Interpolation is performed by filtering the impulses with a quarter-band digital LP filter (Fig. 2.20). Notice we use here a very sharp filter, which requires a length of 300 coefficients. In oversampling DACs, LP filtering is performed much more crudely, for meeting low computational load constraints.

```
lp_fir=firpm(300,[0 .24 .26 1],[1 1 0 0]);
              % Linear phase quarter-band FIR filter
signal_interpolated=filter(lp_fir,1,signal_pulses);

plot(signal_interpolated(1000:1400));
```

The variances of the signal and of the quantization noise (hence, the SNR) have not changed.

```
pwelch(signal_interpolated,[],[],[],8);
signal_sampled_at_4Fs=peak*sin(2*pi*200*(0:1/32000:2));
snr_interpolated=snr(signal_sampled_at_4Fs,…
                     signal_interpolated,300)
```

```
snr_interpolated = 30.1699
```


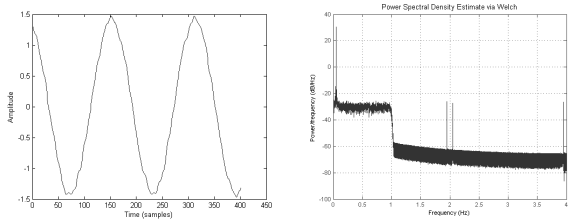

Fig. 2.20 Sine wave after upsamping and interpolation: time-domain (left) and power spectral density (right).

Here comes the first positive effect of oversampling on SNR: (re)quantizing the interpolated signal with N-1 bit (i.e., omitting the least significant bit of the underlying PCM codes) does not much affect the SNR[11].

Let us first perform 4-bit requantization of the interpolated signal.

```
signal_requantized=uquantize(signal_interpolated,4,…
                             quantizer_saturation);
plot(signal_sampled_at_4Fs(2000:2400)); hold on;
plot(signal_requantized(2151:2550)); hold off; [12]
```

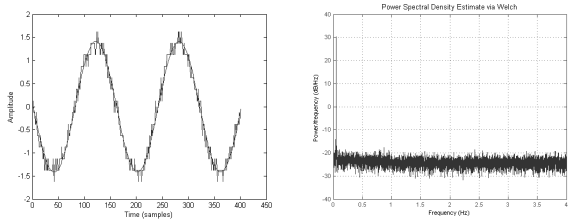

Fig. 2.21 Sine wave after requantization on 4 bits at $4F_s$: time-domain (left) and power spectral density (right).

[11] At worst it could degrade the SNR by 3 dB.

[12] Notice we have to shift the oversampled signal when comparing it to the requantized signal; this is due to the delay of the interpolation filter.

The PSD of the requantized signal shows noise with a variance of about 12dB higher than its initial value, but only 1/4th of that noise is in the band [0, F_s/2]. Therefore the measured SNR does not reflect the actual SNR. We will see below that the actual SNR has only decreased by about 6 dB.

```
pwelch(signal_requantized,[],[],[],8);
snr_requantized=snr(signal_sampled_at_4Fs,…
                    signal_requantized,300)
```

```
snr_requantized =  17.8152
```

Now we can again *simulate* the analog part of D/A conversion (as in Section 3), using an "analog sampling frequency" of 3*32=96 kHz (Fig. 2.22). Oversampling has a second positive effect on this part: it allows for a much wider transition band for the lowpass smoothing filter: [3400 Hz, 28600 Hz]. This results in a simpler filter, of order 4. [13]

```
[order,Wn]=cheb1ord(2*pi*3400,2*pi*28600,0.1,60,'s');
[Num_LP,Den_LP]=cheby1(order,0.1,Wn,'s'); % relaxed filter
signal_pulses=zeros(1,3*length(signal_requantized));
signal_pulses(1:3:3*length(signal_requantized))=…
                                    signal_requantized;
hold_impresp=ones(1,3);
signal_staircase=conv(signal_pulses,hold_impresp);
analog_output=filters(Num_LP,Den_LP,signal_staircase,96000);

plot(analog_output(2000:3300));
pwelch(analog_output,[],[],[],24);
```

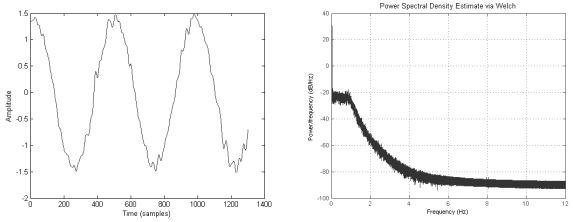

Fig. 2.22 Analog signal produced by the oversampling DAC: time domain (left) and power spectral density (right)

[13] We use Chebyshev approximation here. For some applications, a Bessel filter could be preferred, as it does not imply important phase non-linearity. For our sine-based proof-of-concept, it makes no difference.

As a result of the analog filtering, the apparent SNR is now about 7 dB lower than for our initial 6-bit quantization, i.e. only 1dB less than 5-bit quantization at 8 kHz. The lost dB comes from dithering, and from the fact that the LP filter specifications are loose.

```
signal_sampled_at_12Fs=peak*sin(2*pi*200*(0:1/96000:2));
snr_analog=snr(signal_sampled_at_12Fs,analog_output,4000)
```

```
snr_analog = 22.9805
```

Summarizing: a 6-bit DAC operating at F_s leads to the same SNR as a 5-bit DAC operating at the 4x oversampling rate 4*F_s.

2.2.5 Oversampling and noise shaping DAC

We will now show that noise shaping makes it possible to produce 4-bit quantization at 4*F_s with the same resolution as 6-bit quantization at F_s.

We start by quantizing to 6 bits and oversampling by 4.

```
signal_std=1;
peak=sqrt(signal_std*2);
signal=peak*sin(2*pi*200*(0:1/8000:2));
%quantization
signal_quantized=uquantize(signal,6,2);
% Oversampling by 4
signal_pulses=zeros(1,4*length(signal_quantized));
signal_pulses(1:4:4*length(signal_quantized))=…
                                    signal_quantized;

lp_fir=firpm(300,[0 .24 .26 1],[1 1 0 0]);
signal_interpolated=filter(lp_fir,1,4*signal_pulses);
```

Let us now perform 4-bit requantization of the interpolated signal, with noise shaping, as shown in Fig. 2.6.

```
N=4;
quantizer_saturation=2;
q=2*quantizer_saturation/2^N;

delay_memory=0;
signal_requantized=zeros(size(signal_interpolated));
for i=1:length(signal_interpolated)
    u=signal_interpolated(i)+delay_memory;
    % quantization, including dithering
    signal_requantized(i)=uquantize(u,N,quantizer_saturation);
    delay_memory=u-signal_requantized(i);
end;

signal_sampled_at_4Fs=peak*sin(2*pi*200*(0:1/32000:2));
plot(signal_sampled_at_4Fs(2000:2400)); hold on;
plot(signal_requantized(2151:2550)); hold off;
```

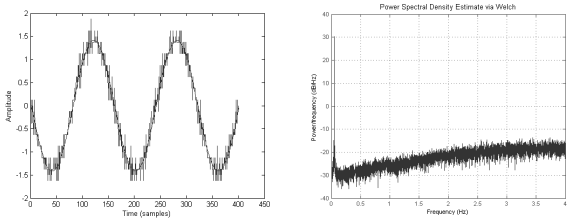

Fig. 2.23 Sine wave after requantization to 4 bits at $4F_s$ using noise shaping: time-domain (left) and power spectral density (right)

The PSD of the requantized signal shows colored quantization noise with most of its energy above $F_s/2$, as a result of noise shaping (Fig. 2.23). Therefore the measured SNR does not reflect the actual SNR.

```
pwelch(signal_requantized,[],[],[],8);
snr_requantized=snr(signal_sampled_at_4Fs,…
                    signal_requantized,300)
```

```
snr_requantized =  14.8699
```

Again, we can *simulate* the analog part of D/A conversion (Fig. 2.24).

```
signal_pulses=zeros(1,3*length(signal_requantized));
signal_pulses(1:3:3*length(signal_requantized))=…
                                        signal_requantized;
hold_impresp=ones(1,3);
signal_staircase=conv(signal_pulses,hold_impresp);

[order,Wn]=cheb1ord(2*pi*3400,2*pi*28600,0.1,60,'s');
[Num_LP,Den_LP]=cheby1(order,0.1,Wn,'s'); % relaxed filter

analog_output=filters(Num_LP,Den_LP,signal_staircase,96000);

plot(analog_output(2000:3300));
```

As a result of the analog filtering, the apparent SNR is now only 3 dB lower than with our initial 6-bit quantization, i.e. only 3dB less than 4-bit quantization at 8 kHz. Again, the lost dBs come from dithering, and from the fact that the LP filter specifications are loose.

```
pwelch(analog_output,[],[],[],24);
signal_sampled_at_12Fs=peak*sin(2*pi*200*(0:1/96000:2));
snr_analog=snr(signal_sampled_at_12Fs,analog_output,4000)
soundsc(analog_output,96000);
```


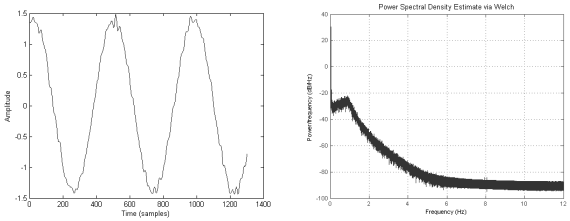

Fig. 2.24 Analog signal produced by the oversampling DAC with noise shaping: time-domain (left) and power spectral density (right)

```
snr_analog =      27.0433
```

2.2.6 Delta-sigma DAC

The principle of delta-sigma modulation is to apply noise shaping and oversampling in such a way that the signal ends up being quantized on 1 bit. This considerably alleviates the task of the hardware D/A converter (which reduces to a switch), and puts most of the load in the digital domain.

In this proof-of-concept, we will perform 6-to-1 bit requantization, using an oversampling ratio of 32 (2^5) and a first-order delta-sigma modulators. Real delta-sigma DACs use higher-order modulators, or cascades of first-order modulators, and thus do not have to implement oversampling of ratio 2^15!

We start by quantizing to 6 bits and oversampling by 32. [14]

```
signal_std=1;
peak=sqrt(signal_std*2);
signal=peak*sin(2*pi*200*(0:1/8000:1/8));

%quantization
N=6;
quantizer_saturation=3;
signal_quantized=uquantize(signal,N,quantizer_saturation);
snr_quantized=snr(signal,signal_quantized,0)

% Oversampling by 32
signal_pulses=zeros(1,32*length(signal_quantized));
```

[14] We do it here with a single interpolation filter, for convenience. In practice, it would be more computationally efficient to implement oversampling as a cascade of two intermediate oversampling blocks, as mentioned in 2.1.5.

```
signal_pulses(1:32:32*length(signal_quantized))=…
   signal_quantized*32;
lp_fir=firpm(300,[0 1/32-1/100 1/32+1/100 1],[1 1 0 0]);
signal_interpolated=filter(lp_fir,1,signal_pulses);
```

```
snr_quantized = 26.6606
```

Let us now perform 1-bit requantization with noise shaping. The resulting quantized signal is purely binary (Fig. 2.25).

```
N=1;
quantizer_saturation=3;
q=2*quantizer_saturation/2^N;

dither=(rand(size(signal_interpolated))-0.5)*q…
      +(rand(size(signal_interpolated))-0.5)*q;

signal_requantized=zeros(size(signal_interpolated));
delay_memory=0;
for i=1:length(signal_interpolated)
    u=signal_interpolated(i)-delay_memory;
    % quantization, including dithering
    signal_requantized(i)=uquantize(u,N,quantizer_saturation);
    % saving the internal variable
    delay_memory=signal_requantized(i)-u;
end;

signal_sampled_at_32Fs=peak*sin(2*pi*200*(0:1/256000:1/8));
plot(signal_sampled_at_32Fs(2000:2600)); hold on;
plot(signal_requantized(2151:2750)); hold off;
xlabel('Time (samples)'); ylabel('Amplitude');
pwelch(signal_requantized,[],[],[],64);
```

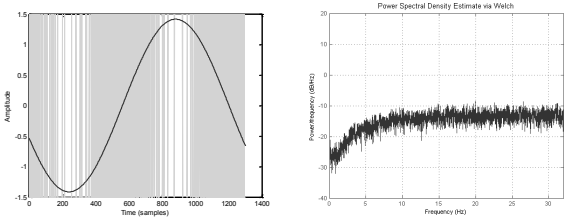

Fig. 2.25 Sine wave after requantization on 1 bit at $32F_s$ using noise shaping: time-domain (left, superimposed with the original sine wave) and power spectral density (right)

Again, we can *simulate* the analog part of D/A conversion (Fig. 2.26). We use an "analog sampling frequency" equal to the one we have reached after 32 times interpolation.


```
[order,Wn]=cheb1ord(2*pi*3400,2*pi*28600,0.1,60,'s');
[Num_LP,Den_LP]=cheby1(order,0.1,Wn,'s'); % relaxed filter
zplane(Num_LP,Den_LP);

analog_output=filters(Num_LP,Den_LP,signal_requantized,…
                      256000);

plot(analog_output(2000:3300));
```

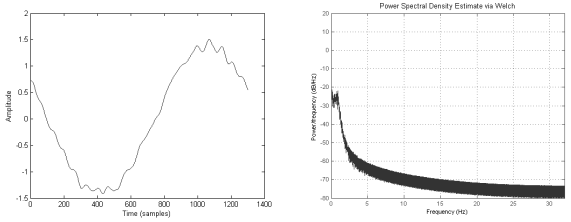

Fig. 2.26 Analog signal produced by the delta-sigma DAC: time-domain (left) and power spectral density (right)

As a result of the analog filtering, the apparent SNR is now very close to that of our initial 6-bit quantization! This is confirmed by listening.

```
pwelch(analog_output,[],[],[],64);
snr_analog=snr(signal_sampled_at_32Fs,analog_output,4000)
soundsc(analog_output,256000);
```

```
snr_analog =  25.4677
```

This technique is used in most CD players today[15], for requantizing 16-bits samples to one bit, hence with much higher oversampling ratios. Interpolation is therefore performed in several steps, for keeping the interpolation filters as simple as possible. Delta-sigma is also the heart of the DSD (direct stream digital) coding used in Super-Audio CDs (SACD), in which a 1-bit stream is created by the ADC, stored on the CD, and directly converted to sound by the DAC.

[15] Notice that all the filters used in this proof-of-concept use floating point arithmetics. Using fixed-point arithmetics is more complex.

2.3 Going further

A very simple interactive tutorial on delta-sigma analog-to-digital conversion is available from (Analog Devices 2007).

The unescapable reference in delta-sigma modulators is (Schreier 2005). This is a companion book to R. Schreier's MATLAB Delta-Sigma toolbox (Schreier 2003).

2.4 Conclusion

As announced in the introduction of this Chapter, efficient DACs add noise to the PCM signal, in three ways. First, they use dithering to avoid having quantization errors correlated with the input signal. Second, they requantize the signal at a higher sampling rate, which can be seen as adding a second quantization noise. Third, they use noise-shaping, which increases the overall quantization noise power but pushes most of it outside the useful band. As a result, and rather unexpectedly, 1-bit DACs are synonymous with high quality.

Besides, one-bit sigma-delta conversion is the basis of the Direct Stream Digital, a technology used to store audio signals in a digital format which is used in Super Audio CD (SACD), the high-resolution CD audio format trademarked by Philips and Sony (Reefman and Janssen 2004). In this technique (Fig. 2.27), the audio waveform for each channel is fed to an analog delta-sigma modulator which directly samples the signal with a sampling frequency 64 times higher than required and quantizes it to 1 bit. This 82 Mbits/s bit stream (to be compared to the 705 kbits/s bit stream of the classical CD) is recorded on the SACD (it is not low-pass filtered). The SACD reader therefore reduces to a 1-bit DAC producing a bipolar signal, followed by the smoothing filter of an oversampling DAC (i.e., with relaxed constraints).

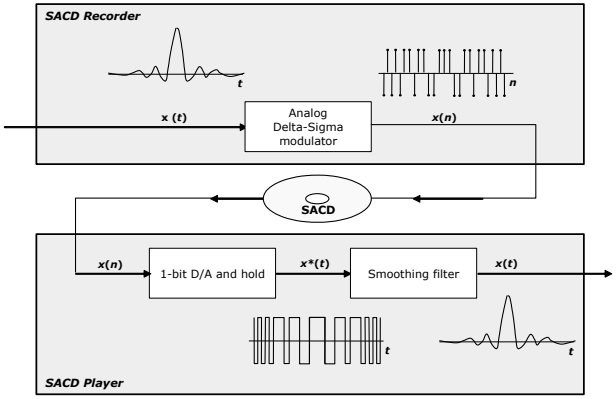

Fig. 2.27 The Super Audio CD concept

Last but not least, the pulse-density modulated (PDM) available after the 1-bit DAC in a DSD player ($x^*(t)$ in Fig. 2.27) can directly be amplified by transistors switching from full ON to full OFF, thereby implementing a fully digital *Class D* power amplifier (Gaalaas 2006).

2.5 References

Analog Devices (2007) Interactive Design Tools: Sigma-Delta Analog-to-Digital Converters : Sigma-Delta ADC Tutorial. [online] Available: http://www.analog.com/ [19/3/2007]

Candy JC (1997) An Overview of Basic Concepts. In: by Norsworthy, Schreier, and Temes (eds) Delta-Sigma Data Converters. IEEE Press, Piscataway, NJ: , pp 1–43

Gaalaas E (2006) Class D Audio Amplifiers: What, Why and How. Analog Dialogue 40-06 pp 1–7 [online] Available: http://www.analog.com/library/analogDialogue/archives/40-06/class_d.pdf [15/3/2007]

Hicks C (1995) The application of Dither and Noise-Shaping to Nyquist-Rate Digital Audio: an Introduction [online] Available: www.digitalsignallabs.com/noiseb.ps [15/3/2007]

Kester W, Bryant J (2003) DACs for DSP applications. In: Kester W (ed) Mixed Signal and DSP Design Techniques. Newnes, Amsterdam, pp 99–115.

Lipschitz SP, Vanderkooy J (2001) Why 1-Bit Sigma-Delta Conversion is Unsuitable for High-Quality Applications. 110th Audio Engineering Society Convention Paper 5395, Amsterdam.

Matsuya Y Uchimura K, Iwata A, Kobayashi T, Ishikawa M, and Yoshitome T, A (1987) 16-bit oversampling A-to-D conversion technology using triple-integration noise shaping. IEEE Journal of Solid-State Circuits 22-12, pp 921-929

Reefman D, Janssen E (2004) One-bit audio: an overview. Journal of the Audio Engineering Society 52-2

Schreier R (2003) The Delta-Sigma Toolbox. [online] Available: http://www.mathworks.com/matlabcentral/ (search for "delta-sigma") [19/3/2007]

Schreier R, Temes GC (2005) Understanding delta-sigma data converters. IEEE Press, Piscataway, NJ.

Wannamaker RA (1997) The Theory of Dithered Quantization. PhD Thesis, University of Waterloo, Canada

Chapter 3

How is sound processed in an MP3 player?

Audio was a bit loose in its PCM suit:
the suit could loose bits and turned into a lighter MP3 jacket

T. Dutoit (°), N. Moreau (*)

(°) Faculté Polytechnique de Mons, Belgium
(*) Ecole Nationale Supérieure des Télécommunications, Paris

In his 1929 painting "La trahison des images", Belgian painter René Magritte highlighted the power of illusions by painting a pipe and commenting it with "Ceci n'est pas une pipe" ("This is not a pipe"): as Magritte himself explained: "Try to stuff the painting with tobacco...".

Perceptual illusions have been used by engineers for designing many consumer products. As an example, one of the most extraordinary imperfections of our visual process, known today as the phi effect (and mistaken for a long time with the persistance of vision), has induced the choice for 24 images per second in the movie pictures industry. In this chapter we will see how the limitations of the human *auditory* process have been taken into account for the design of *lossy but transparent* perceptual audio coders. In particular, we will focus on the principles underlying the MPEG-1 Layer-I audio coding norm.

3.1 Background – Sub-band and transform coding

Sub-band signal processing is very useful in audio (as well as in video) coding applications. The sub-band encoder Fig. 3.1 decomposes an input

signal *x(n)* into *M* different frequency ranges called *sub-bands*, through the *analysis filter bank*. Sub-band signals $x_i(n)$ are then downsampled to $y_i(m) = x_i(Mm)$ and quantized. The decoder combines the quantized sub-band signals $\tilde{y}_i(m)$ into an output signal $\tilde{x}(n)$ by upsampling them to $\tilde{x}_i(n)$ (with $\tilde{x}_i(mM) = \tilde{y}_i(m)$ and $\tilde{x}_i(mM + l) = 0$ for $l = 0,...,M-1$) and passing them through the *synthesis filter bank* (Crochiere and Rabiner 1983, Vaidyanathan 1993). The number of sub-bands and the decimation-interpolation ratio is not independent: in a *critically sampled* filter bank, for which the number of sub-band signal samples is equal to the number of input samples, the number of sub-bands and the decimation-interpolation ratio are identical.

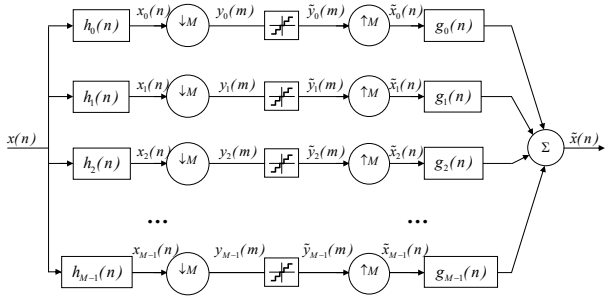

Fig. 3.1 A sub-band coder

The rationale for sub-band signal processing is the need for time- and frequency-dependent SNR control to allow transparent encoding of audio and video signals with very low bit-rates. As a matter of fact, when applying uniform quantization to the input signal using a given number of bits per sample, the resulting quantization noise is uniformly distributed in the frequency range of the signal (see Chapter 2). Since the power spectral density (PSD) of the input signal changes with time and frequency, uniform quantization results in time- and frequency-dependent signal-to-noise ratio (*local* SNR). The number of bits per sample (hence, the *overall SNR*) is therefore usually set to a very high value (the 16 bits-96 dB of the compact disk), in order for the worst-case local SNRs to be statistically acceptable to the human ear. In contrast, in sub-band signal processing, the number of bits allocated to each sub-band signal can be adapted to the specific SNR required in the corresponding frequency band. This, as we shall see, allows for drastic bit rate reduction.

In this Section, we will first examine the conditions required on the analysis-synthesis filters to satisfy the perfect reconstruction condition (3.1.1). We will show that the filter-bank approach can also be interpreted in terms of block-transform-based processing (3.1.2). The masking properties of the human ear will then be briefly reviewed and their use in the MPEG1- Layer I audio coding norm will be explained (3.1.3). We will conclude by mentioning current coding norms and bit-rates (3.1.4).

3.1.1 Perfect reconstruction filters

An important feature of the analysis and synthesis filters in a filter bank is that they should allow *perfect reconstruction* (PR) (or, in practice, *nearly* perfect reconstruction) when no quantization is performed, i.e.:

$$\tilde{x}(n) = x(n) \tag{3.1}$$

In order to examine this problem in a simple case, let us focus on a two-channel filter bank (Fig. 3.2), composed of a low-pass (upper) and a high-pass (lower) branch.

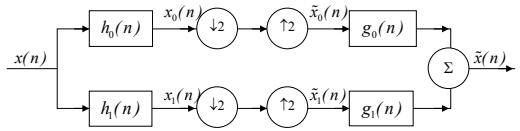

Fig. 3.2 A two-channel filter bank

The PR condition implies to cancel (or at least minimize) 3 types of distortion, respectively due to the phase and amplitude responses of the filters, and to the aliasing that is inevitably introduced by the imperfections of the decimation-interpolation steps. One of the great ideas of sub-band coding, precisely, is that the PR condition does not imply that aliasing should be minimized in each branch separately (which would require close-to-ideal filters), but that some aliasing distortion can be accepted for sub-band signals, *provided all sub-band distortion in adjacent sub-bands*

is cancelled by the final summation. As a consequence, analysis and synthesis filters are not chosen independently.

The combined decimation-interpolation steps, which transform each sub-band signal $x_i(n)$ into $\tilde{x}_i(n)$ replace every other sample in $x_i(n)$ ($i=0,1$) by zero. It is thus equivalent to adding to each sub-band signal $x_i(n)$ a copy of it in which every other sample is sign-reversed:

$$\tilde{x}_i(n) = \frac{1}{2}\left[x_i(n) + x_i^-(n)\right] \quad with \;\; x_i^-(n) = x_i(n)(-1)^n$$
$$\text{or } \tilde{X}_i(z) = \frac{1}{2}\left[X_i(z) + X_i^-(z)\right] \quad with \;\; X_i^-(z) = X_i(-z) \tag{3.2}$$

The added signal $x_i^-(n)$ can be seen as a modulation of $x_i(n)$ by $\exp(jn\pi)$, i.e. by a Nyquist rate carrier. Its discrete-time Fourier transform (DTFT) is therefore that of $x_i(n)$, shifted by the Nyquist frequency:

$$X_i^-(f) = X_i(f - \frac{1}{2}) \tag{3.3}$$

which implies that:

$$|X_i^-(f)| = |X_i(\frac{1}{2} - f)| \tag{3.4}$$

$|X_i^-(f)|$ is thus the quadrature mirror of $|X_i(f)|$, i.e. its mirror with respect to $f = ¼$.

Frequency aliasing occurs in the top branch of Fig. 3.2 when these DTFTs overlap, as can be seen in Fig. 3.3. A similar conclusion could be drawn for the lower branch.

The output of the filter bank can be written as:

$$\begin{aligned}\tilde{X}(z) &= \left[\tilde{X}_0(z)G_0(z) + \tilde{X}_1(z)G_1(z)\right] \\ &= \frac{1}{2}\left[X_0(z)G_0(z) + X_1(z)G_1(z)\right] \\ &+ \frac{1}{2}\left[X_0(-z)G_0(z) + X_1(-z)G_1(z)\right]\end{aligned} \tag{3.5}$$

in which the second term is due to the quadrature mirror images of signals $x_0(n)$ and $x_1(n)$, and is therefore responsible for the possible aliasing.

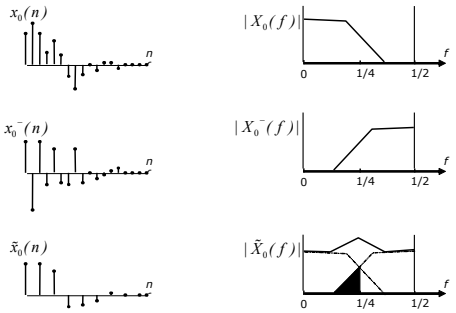

Fig. 3.3 Decimation-interpolation seen as the addition of $x_0^-(n)$ to the low sub-band signal. Due to the overlap of their DTFTs in the black triangle, aliasing occurs, which prevents $|\tilde{X}_0(f)|$ from being equal to $|X_0(f)|$ for f in [0,1/4].

Equation (3.5) can be further expanded as:

$$\begin{aligned}\tilde{X}(z) &= \frac{1}{2}X(z)\left[H_0(z)G_0(z) + H_1(z)G_1(z)\right] \\ &+ \frac{1}{2}X(-z)\left[H_0(-z)G_0(z) + H_1(-z)G_1(z)\right]\end{aligned} \tag{3.6}$$

which can be identified to (3.1) if:

$$\begin{cases} H_0(z)G_0(z) + H_1(z)G_1(z) = 2 \\ H_0(-z)G_0(z) + H_1(-z)G_1(z) = 0 \end{cases} \tag{3.7}$$

The second condition is easy to satisfy, with:

$$\begin{aligned} G_0(z) &= H_1(-z) \quad or \quad g_0(n) = h_1^-(n) \\ G_1(z) &= -H_0(-z) \quad or \quad g_1(n) = -h_0^-(n) \end{aligned} \tag{3.8}$$

The first condition in (3.7) then becomes:

$$H_0(z)H_1(-z) - H_0(-z)H_1(z) = 2 \tag{3.9}$$

QMF filters

One way of satisfying this constraint was introduced by Esteban and Galland (1977), who further imposed $h_1(n)$ to be the *quadrature mirror filter* (QMF) of $h_0(n)$:

$$h_1(n) = h_0^-(n) \quad \text{or} \quad H_1(z) = H_0(-z) \tag{3.10}$$

and found some prototype filter $H_0(z)$ such that:

$$\begin{aligned} & H_0^2(z) - H_0^2(-z) = 2 \\ \text{or} \quad & H_0^2(f) - H_0^2(f - \frac{1}{2}) = 2 \end{aligned} \tag{3.11}$$

The final filter bank is given in Fig. 3.4. The beauty of this idea lies in its simplicity.

In practice, $H_0(z)$ is a linear phase (i.e. $h_0(n)$ is symmetrical) FIR filter of length N (N being even[1]), and in order to implement it as a causal filter, its impulse response is delayed by $(N-1)/2$. This results in an overall delay of $N-1$ samples in both branches of the filter bank. Taking into account that:

$$H_0(f) = |H_0(f)| e^{j2\pi f(N-1)/2} \tag{3.12}$$

one finds easily from (3.11):

$$|H_0(f)|^2 + |H_0(f - \frac{1}{2})|^2 = 2 \tag{3.13}$$

It can be shown that designing $H_0(z)$ such that satisfies (3.13) while being frequency selective is impossible[2]. Johnston (1980) proposed an iterative approach, which results in frequency selective filter banks free of phase and aliasing distortion, but with some (controllable) amplitude distortion. Many authors have proposed other PR filters with various properties, such as conjugate quadrature filters (CQF). This scientific discussion also led to the parallel development of the *wavelet theory* (see Chapter 10).

[1] It can be shown that FIR QMF filters can be built for N odd, but require a modification of (3.10).

[2] Satisfying the QMF constraint while not being frequency selective is not very useful in sub-band coding, as it does not allow efficient frequency-dependent SNR control.

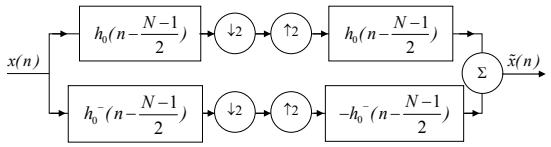

Fig. 3.4 A two-channel QMF filter bank implementing linear phase FIR filters of length N

Pseudo-QMF filters

Going from 2 to M sub-bands can be done by further splitting each sub-band in Fig. 3.4 into 2 bands and so on (the so-called *tree-structured* or *hierarchical filter bank* approach), but it is more straightforward to implement the filter bank as M parallel pass-band filters. In the Pseudo-QMF (PQMF) approach (Nussbaumer and Vetterli 1984, Vaidyanathan, 1993), the analysis filters $H_i(z)$ are obtained by modulating the impulse response $h(n)$ of a low-pass prototype filter with M carrier frequencies f_i uniformly distributed in the range [0,1/2]:

$$\begin{aligned} h_i(n) &= h(n)\cos(2\pi f_i n + \phi_i) \\ g_i(n) &= h(n)\cos(2\pi f_i n + \theta_i) \end{aligned} \quad with\ f_i = \frac{(2i+1)}{2M}\frac{1}{2} \tag{3.14}$$

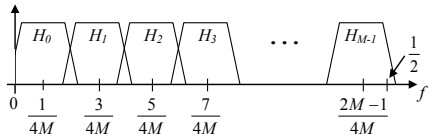

Fig. 3.5 A uniform M-channel filter bank (f denotes normalized frequency)

The values of ϕ_i and θ_i are chosen so that all significant aliased terms are canceled[3], which can be shown to happen if:

[3] The *pseudo* qualifier in PQMF comes from the fact that only adjacent bands are taken into account in the PR condition. Hence, PQMF filters do not strictly satisfy the PR condition.

$$\phi_i = -\frac{2i+1}{4}\pi = -\theta_i \tag{3.15}$$

The conditions imposed on the prototype low-pass filter are very similar to (3.13), except the normalized bandwidth of the filter must be 1/4*M* instead of 1/4. They also cause the analysis and synthesis filters to be the mirror image of each other:

$$h_i(n) = g_i(N-1-n) \tag{3.16}$$

which forces $H_i(z)G_i(z)$ in all sub-band channels to have linear phase.

PQMF filters are used in the MPEG-1 Layer-I audio coding norm.

MDCT filters

PQMF filters were later generalized to PR cosine modulated filter banks, with special emphasis on filters whose length *N* is twice the number of channels, known as the Modified Discrete Cosine Transform (MDCT) filter banks (Princen and Bradley 1986). The impulse responses of the analyis filters are given by:

$$\begin{aligned} h_i(n) &= h(n)\sqrt{\frac{2}{M}}\cos(2\pi f_i n + \phi_i) \\ g_i(n) &= h_i(2M-1-n) \end{aligned} \quad with \quad \begin{aligned} f_i &= \frac{(2i+1)}{2M}\frac{1}{2} \\ \phi_i &= \frac{(2i+1)(M+1)\pi}{4M} \end{aligned} \tag{3.17}$$

The PR conditions can be shown to lead to the following constraints on the prototype low-pass filter:

$$\begin{aligned} h^2(n) + h^2(n+M) &= 1 \\ h(n) &= h(2M-1-n) \end{aligned} \quad for\ n = 0,\ldots,M-1 \tag{3.18}$$

A special case, which is often mentioned, is the Modulated Lapped Transform filter bank (Malvar 1990), in which:

$$h(n) = \sin\left[\left(n+\frac{1}{2}\right)\frac{\pi}{2M}\right] \quad for\ n = 0,\ldots,2M-1 \tag{3.19}$$

MDCT filters are used in the MPEG-1 Layer-3 (MP3) as well as in the MPEG-2 Advanced Audio Coding (AAC) and MPEG-4 norms.

3.1.2 Filter banks and lapped transforms

If the length of the filters used in *M*-channel sub-band filters were exactly *M* samples, one could advantageously implement the operations performed simultaneously by the *M* analysis filters as a linear transform of successive non-overlapping *M* sample frames of the input signal.

Conversely, any linear transform can be seen as implementing a special filter bank (Fig. 3.6). If the linear transform is characterized by its *MxM* matrix **H**, and the input frame is stored in column vector **x**, then **y**=**Hx** computes the output of *M* analysis filters whose impulse responses are the time-reversed rows of **H**. As a matter of fact, Fig. 3.1 verifies:

$$\begin{bmatrix} y_0(m) \\ y_1(m) \\ \ldots \\ y_{M-1}(m) \end{bmatrix} = \begin{bmatrix} h_0(M-1) & \ldots & h_0(1) & h_0(0) \\ h_1(M-1) & \ldots & h_1(1) & h_1(0) \\ \ldots & \ldots & \ldots & \ldots \\ h_{M-1}(M-1) & \ldots & h_{M-1}(1) & h_{M-1}(0) \end{bmatrix} \begin{bmatrix} x(mM-M+1) \\ \ldots \\ x(mM-1) \\ x(mM) \end{bmatrix} \tag{3.20}$$

Downsampling is straightforward: since the output of the analysis filters must be downsampled by *M*, one only needs to compute one value of **y** every *M* samples (by using non-overlapping frames for **x**). Similarly, computing $\tilde{\mathbf{x}}=\mathbf{G}^T\mathbf{y}$, in which **G** is an *M*x*M* matrix whose rows are the time-reversed impulse responses of the analysis filters:

$$\begin{bmatrix} x(mM-M+1) \\ \ldots \\ x(mM-1) \\ x(mM) \end{bmatrix} = \begin{bmatrix} g_0(M-1) & g_1(M-1) & \ldots & g_{M-1}(M-1) \\ \ldots & \ldots & \ldots & \ldots \\ g_0(1) & g_1(1) & \ldots & \ldots \\ g_0(0) & g_1(0) & \ldots & g_{M-1}(0) \end{bmatrix} \begin{bmatrix} y_0(m) \\ y_1(m) \\ \ldots \\ y_{M-1}(m) \end{bmatrix} \tag{3.21}$$

computes the output of the synthesis filters, but it also sums all sub-band signals to produce the output frame. Obviously, PR is achieved in this case if $\mathbf{G}^T$ is the inverse of H, since $\mathbf{x}=\mathbf{H}^{-1}\mathbf{y}$.

Having input and output frames that do not overlap is a problem when sub-band signals are modified in a time-varying way (as it happens typically when they are quantized as a function of their instantaneous energy): the effect of each modification applies to one synthesis frame only, with no possibility of inter-frame smoothing. This effect is known as the *blocking*

effect[4]. Hence, practical filter banks make use of longer analysis and synthesis filters (as in the previous section on PQMF filters).

However, when the length N of the analysis and synthesis filters is longer than the number of channels, the filtering operations can still be implemented as the multiplication of N sample frames with NxM or MxN matrices. Analysis filtering is still achieved by $\mathbf{y}=\mathbf{H}\mathbf{x}$, in which $\mathbf{H}$ is an MxN matrix whose rows are the time-reversed impulse responses of the analysis filters, and $\mathbf{x}$ is an N sample frame. Downsampling is achieved by shifting successive input frames by M samples (i.e. they *overlap* by N-M samples).

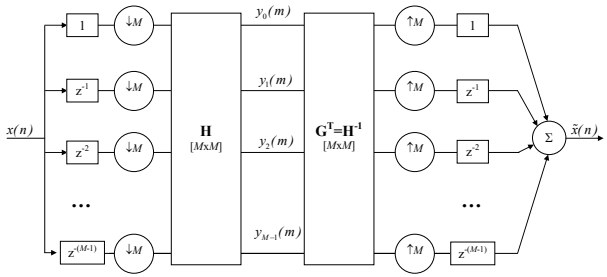

Fig. 3.6 An M-channel transform-based implementation of a filter bank using filters of length $N=M$

The upsampling, synthesis filtering, and summation operations are still performed as $\mathbf{G}^T\mathbf{y}$, in which $\mathbf{G}$ is an MxN matrix whose rows are the (not time-reversed) impulse responses of the analysis filters. This computes the sum of the responses of the synthesis filters excited each by the single sample stored in $\mathbf{y}$ for each sub-band. Just as the input frames, the output frames overlap in time (and should therefore be accumulated: see Fig. 3.7 for a simple example with N=2M). This leads to the more general implementation of Fig. 3.8, which only slightly differs from Fig. 3.6 and still clearly shows the underlying sub-band processing interpretation.

[4] The same concept will be for the case of image coding in Chapter 8.

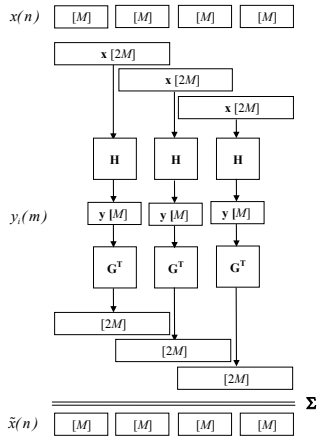

Fig. 3.7 Frame-based view of an M-channel transform-based implementation of a filter bank, using filters of length $N=2M$

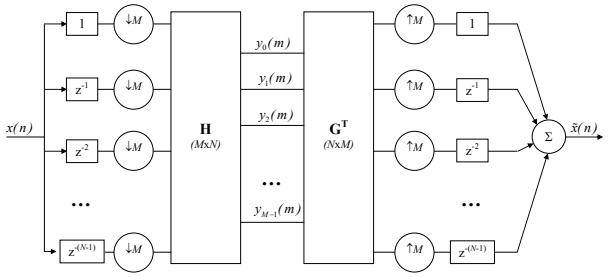

Fig. 3.8 An M-channel transform-based implementation of a filter bank using filters of length $N>M$

3.1.3 Using the masking properties of the human ear

From an engineering point of view, the human ear (not only the mechanical parts of it, but also the whole neural system that produces an acoustic sensation) is a complex device in many respects (Painter and Spanias 2000). First, its sensitivity depends on frequency: to be just heard, a narrow-band stimulus must have a sound pressure level higher than a threshold, called the *absolute auditory threshold,* which is a function of frequency (Fig. 3.9, left). Second, for a given frequency, the impression of loudness is not linearly related to sound pressure (this is obvious for the absolute auditory threshold, but can be generalized for all loudness values). Third, the sensitivity of the ear to a given stimulus is not the same if this stimulus is presented alone or if it is mixed with some masking stimulus. This is termed as the *masking effect*, and is approximated as a local modification of the auditory threshold in a frequency band surrounding that of the masking stimulus (Fig. 3.9, right). Even worse, this effect depends on the type of masking stimulus (pure tones and narrow-band noise do not have the same masking effect). Last but not least, not everyone is equal with respect to sound: the acoustic sensation is listener-dependent and varies with time.

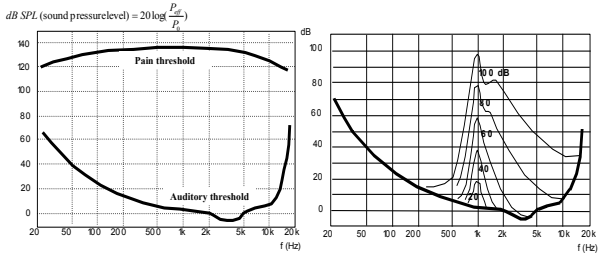

Fig. 3.9 Left: Auditory and pain thresholds as a function of frequency. Right: Masking effect due to a pure tone at 1000 Hz for various intensities. Sounds below the modified auditory threshold are "hidden" by the masking tone. A pure tone at 1000 Hz with an intensity of 80 dB SPL[5], for instance, will make another pure tone at 2000 Hz and 40 dB SPL inaudible.

[5] Sound pressure level (SPL) L_p is a logarithmic measure of the root mean square sound pressure of a sound relative to a reference value, usually measured in decibel:

$$L_p = 20\log_{10}\left(p_{rms} / p_{ref}\right)$$

where p_{ref} is a reference sound pressure, usually set to 20 μPa (rms), although 1 μPa (rms) is used for underwater acoustics.

One of the most practical features of the masking effect can be modeled as operating within a discrete set of 25 auditory "channels" termed as *critical bands* (with idealized band edges equal to 20, 100, 200, 300, 400, 510, 630, 770, 920, 1080, 1270, 1480, 1720, 2000, 2320, 2700, 3150, 3700, 4400, 5300, 6400, 7700, 9500, 12000, and 15500 Hz). As an example, if a signal and masker are presented simultaneously then only the masker frequencies falling within the critical bandwidth contribute to masking the signal. In other words, the ear acts as a sub-band processing system in which separate non-linear masking effects mainly appear in separate sub-bands.

The great interest of sub-band coding is to establish a master-slave relationship (Fig. 3.10), in each sub-band, between: (i) the local *masking threshold* computed by the so-called *psycho-acoustic model* from an estimation of the masking stimuli present in the auditory spectrum, and (ii) quantization noise, i.e. the number of bits used to quantize the corresponding sub-band signal.

Notice that although sub-band coders are based on the features of the auditory process, they do not completely mimic it. As an example, although the edges of the critical bands are not uniformly spaced in frequency, most sub-band coders use equal bandwidth channels. More importantly, since the masking effect depends on the actual loudness of the masking signal, while the coder is not aware of the actual volume level set by the listener, the psychoacoustic model used in sub-band coding *a priori* assumes that the user sets the volume level such that the least significant bit of the original 16-bit PCM code can just be heard.

3.1.4 Audio coders

MPEG-1 Audio

The MPEG-1 Audio norm actually defines three complexity layers of processing, each with its own sub-band coding scheme, psycho-acoustic model, and quantizer (see Pan 1995 or Painter and Spanias 2000 for a tutorial). Starting from 1.4 Mbits/s for a stereo audio signal, Layer-I achieves a 1:4 compression level (downto 384 kbits/s). With the addition of Layer-II (formerly known as Musicam: Masking Pattern adapted Universal Sub-band Coding And Multiplexing), a 1:6-8 ratio is reached (between 256 and 192 kbits/s). This coder is very much used in Digital Video Broadcasting

(DVB). Mpeg-1 Layer-III, commercially known as MP3, further compresses the data stream downto 128 to 112 kbits/s (i.e. a compression ratio on 1:10-12). In this Chapter, we only examine Layer-I, which is far enough for a proof of concept.

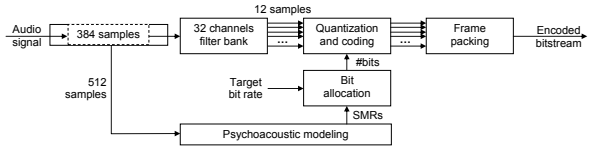

Fig. 3.10 Psychoacoustic model controlling sub-band quantizers in the MPEG-1 Layer-I audio coder

The MPEG-1 Layer-I coder (Fig. 3.10) implements a 32-channels sub-band coder. Samples are processed by frames of 384 samples (i.e. 12 samples at the downsampled rate). The psycho-acoustic model is based on a 512 samples FFT estimation of the local power spectral density of the signal, $S_{xx}(f)$. Local maxima with dominant peaks are detected as tonal maskers and extracted from the FFT. A single noise masker is then computed from the remaining energy in each critical band. Maskers are then combined, accounting again for the existence of critical bands (typically, several maskers in a single band are merged into the strongest of them). Individual masking thresholds are finally estimated for each tonal and noise masker, and summed to produce the overall masking threshold, $\Phi(f)$. This leads to the determination of a *Signal-to-Mask Ratio* (SMR) in each band:

$$SMR(k) = 10\log_{10}\left(\frac{\max\limits_{\text{band } k} S_{xx}(f)}{\min\limits_{\text{band } k} \Phi(f)}\right) \tag{3.22}$$

Sub-band signals are quantized uniformly, and the number of bits for each sub-band is chosen so as to make resulting signal-to-noise ratio lower than the SMR. Higher frequency sub-bands typically require less bits. In particular, sub-bands 27 to 32 are assigned 0 bits, by design. Quantization is made adaptive by normalizing sub-band signals before quantizing them. Normalization is performed by blocks of 12 sub-band samples (i.e. by blocks of 384 samples at the original sampling frequency), by estimating a local scale factor and dividing samples by its value (Fig. 3.11).

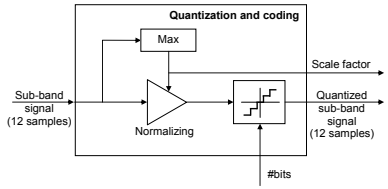

Fig. 3.11 Adaptive quantization in the MPEG-1 Layer-I audio coder

MPEG-1 Layer-II is based on the same principle, but uses 1024 samples for FFT computation in its psychoacoustic model and processes frames that are three times longer than in Layer-I[6]. It also implements non-simultaneous (sometimes called "temporal") masking, which denotes the fact that masking can occur both before masker onset and after masker removal.

MPEG-1 Layer-III further splits each of the 32 original sub-bands into 18 channels, by using a Modified Discrete Cosine Transform (MDCT, see Section 3.1.2), which provides a better frequency resolution[7] when required (i.e. in stationary parts of the audio signal). Its coding scheme is dynamic: more bits are allocated to frames that require it.

MPEG-2 Audio – Advanced Audio Coding

MPEG-2 was normalized in 1994. The MPEG-2 BC audio coder adds a series of features to the MPEG-1 audio coder: an extension to lower sampling frequencies (downto 8 kHz), and multi-channel processing for 5.1 surround sound. It remains *backward compatible* (BC) with the MPEG-1 coder, i.e. MPEG-1 audio coders can decode MPEG-2 BC, ignoring the additional information.

[6] More specifically, scale factors are computed as before, but quantized by groups of three.

[7] MPEG-1 Layers I and II have a good time resolution, since sub-band samples are updated every 32 audio samples, which enables them to track audio transients. They exhibit a bad frequency resolution though, as their sub-band filter bandwidth is $F_s/2/32 \approx 700$ Hz.

MPEG-2 AAC (Advanced Audio Coding[8]) appeared in 1997 and provides a great improvement over MPEG-2 BC (hence, over MP3), especially for low-bit rates (below 100 kbits/s). Contrary to the MPEG-1 Layer III coder, which added an MDCT filter bank for increasing the frequency resolution when needed, this coder uses a single filter bank, with adaptive blocksize: large 1024-sample frames (resp., short 128-sample frames) are used for stationary (resp., transient) sounds, which provides a good compromise between time and frequency resolution. This coder provides 5.1 transparency at 384 kbits/s. It is the most commonly used format for compressing audio CDs for Apple's iPod and iTunes.

MPEG-2 audio coders are also used in digital TV broadcasting.

MPEG-4 Audio

The AAC coder has been enhanced in the framework of MPEG-4, by including additional tools (such as long-term prediction) for intermediate bit rates. A low-delay version of the coder is also available for teleconference applications. The resulting MPEG-4 AAC coder, normalized in 1999, provides stereo transparency with a bit rate between 96 and 128 kbits/s and 5.1 transparency at 256 kbits/s.

In 2003, a technique known as *sub-band replication* (SBR) has been added to the MPEG-4 AAC coder, leading to the MPEG-4 HE AAC (High Efficiency). The idea is to use the strong correlation between low frequency (up to 4-8 KHz) and higher frequency spectral components of audio signals, and to avoid sending redundant information. HE AAC provides stereo transparency at 48 kbits/s and 5.1 transparency at 128 kbits/s.

A new version of it, HE AAC v2 (sometimes called AAC+), appeared in 2004, which further uses psycho-acoustic effects to better encode stereo channels as one main channel and side information for spatialization. Intermediate stereo quality is obtained at 24 kbits/s. This coder is used in the emerging Digital Radio Mondiale and in 3G mobile telephony (the 3GPP standard).

The latest development has been reached in July 2006 with the definition of the HE AAC surround (or MPEG surround), which adapts the spatialization coding principles of HE AAC v2 to multi-channel audio. As a result, 5.1 transparency is now available at 64 kbits/s.

[8] Not to be confused with AC3, the Dolby Digital 5.1 audio coding scheme developed by Dolby Labs.

3.2 MATLAB proof of concept: ASP_mp3.m

In this Section, we provide a proof of concept of the the MPEG-1Layer-I audio coding norm. We start by examining a two-channel filter bank using conventional filters (Section 3.2.1) and QMF filters (Section 3.2.2). We then extend our study to a 32-channel PQMF filter bank (Section 3.2.3) and show how it can be efficiently implemented using block-based lapped transforms (Section 3.2.4). We conclude by providing our filter bank with a perceptual quantizer for sub-band signals (Section 3.2.5).

3.2.1 Two-channel filter bank

Before examining a complete *M*-channel filter bank, we first discuss the 2-channel case, and the effect of a decimation and interpolation in each branch.

Let us first generate a 4-seconds chirp signal (from 0 to 4 kHz) with a sampling rate of 8 kHz, which we will use as a reference throughout this Section, and plot its spectrogram (Fig. 3.12, left).

```
Fs=8000;
input_signal=chirp((0:4*Fs)/Fs,0,4,4000);
specgram(input_signal,1024,Fs,256);
```

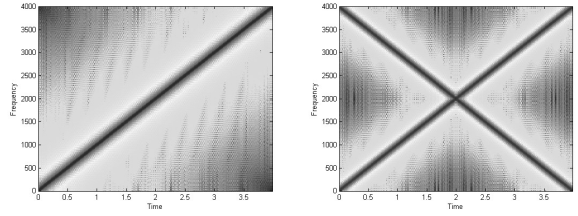

Fig. 3.12 Spectrogram of a chirp. Left: original signal; Right: after downsampling-upsampling without using analysis and synthesis filters.

Applying direct downsampling-upsampling by 2 results in replacing every other sample by zero. The result is that an artifact signal is added,

whose spectrum is the quadrature mirror image of that of the chirp. Two sinusoids can be heard at any time (Fig. 3.12, right). [9]

```
downsampled=input_signal(1:2:end);
upsampled(1:2:2*length(downsampled))=2*downsampled;
```

Adding a quarter-band filter after upsampling eliminates the image distortion, making sure that only one sinusoid appears at any time (Fig. 3.13). During the first half of the chirp, that sinusoid is the chirp itself; during the second half, the sinusoid is an alias, due to the quadrature mirror image of the chirp. We design the synthesis filter as a symmetric FIR, so as to avoid phase distortion, and we set its order to 1000, so as to obtain high stop-band rejection (close to 80 dB). Notice the first 1000 samples are a transient.

```
G0=fir1(1000,1/2);
freqz(G0,1);
G0_output=filter(G0,1,upsampled);

specgram(G0_output(1001:end),1024,Fs);
```

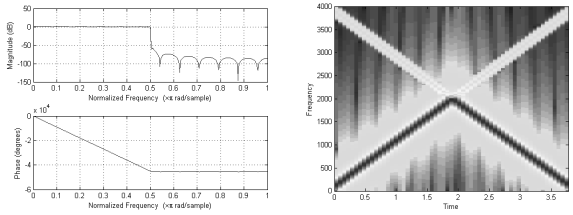

Fig. 3.13 Left: Frequency reponse of the quarter-band analysis filter; Right: Spectrogram of a chirp after downsampling-upsampling, using a low-pass synthesis filter only

Adding another quarter-band filter before downsampling removes the aliasing distortion, i.e. the alias in the second half of the chirp (Fig. 3.14, left). This synthesis filter can a priori be identical to the analysis filter.

```
H0=G0;
H0_output=filter(H0,1,input_signal);
```

[9] Notice we multiply the signal by 2 when upsampling, so that the power of the signal remains unchanged.


```
downsampled=H0_output(1:2:end);
upsampled(1:2:2*length(downsampled))=2*downsampled;
G0_output=filter(G0,1,upsampled);

specgram(G0_output(2001:end),1024,Fs, 256);
```

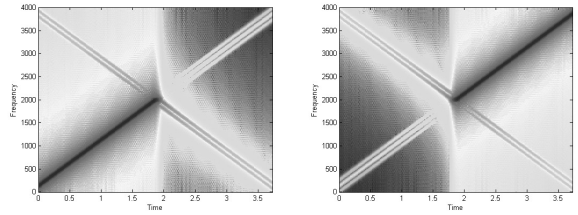

Fig. 3.14 Left: Spectrogram of a chirp after downsampling-upsampling, using low-pass analysis and synthesis filters; Right: same thing with high-pass analysis and synthesis filters

Using HF quarter-band filters instead of LF ones selects the second half of the chirp (Fig. 3.14, right).

```
G1=fir1(1000,1/2,'high');
H1=G1;
H1_output=filter(H1,1,input_signal);
downsampled=H1_output(1:2:end);
upsampled(1:2:2*length(downsampled))=2*downsampled;
G1_output=filter(G1,1,upsampled);
```

We now examine the output of the 2-channel sub-band filter bank, by adding the two signals obtained above (Fig. 3.15, left). Perfect reconstruction is not achieved, as shown in the previous spectrogram. Closer examination of the error waveform (Fig. 3.15, right) shows that most of the error lies in the center of the signal (this is not shown in the spectrogram), i.e., for frequencies for which the `H0` and `H1` filters overlap: more important aliasing cannot be avoided in each band for these frequencies.

```
synt_signal=G0_output+G1_output;
% Shift synt_signal to account for the filters delay.
% A symmetric FIR filter of order N (length N+1) brings
% a delay of N/2 samples.
error=synt_signal(1001:end)-input_signal(1:end-1000);
```

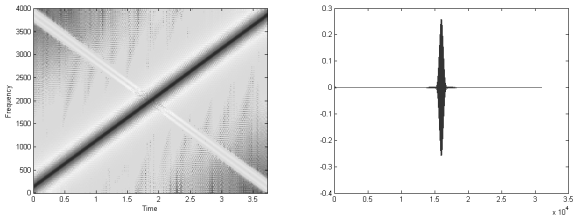

Fig. 3.15 Left: Spectrogram of the output of the 2-channel filter bank; Right: error signal waveform

3.2.2 Two channel QMF filter bank

Perfect reconstruction is possible even with non ideal filters (i.e. even with some aliasing in each band) provided the overall aliasing is canceled when adding the low-pass and high-pass sub-band signals. This, as we shall see, is mainly the responsibility of the analysis and synthesis filters. Johnson's type QMF filters provide a solution for nearly perfect reconstruction. In this example, we use Johnson's "B-12" QMF (Fig. 3.16).

```
H0_QMF=[-0.006443977 0.02745539 -0.00758164 ...
    -0.0913825  0.09808522 0.4807962];
H0_QMF=[H0_QMF fliplr(H0_QMF)];
[H0,W]=freqz(H0_QMF,1);

H1_QMF=H0_QMF.*[1 -1 1 -1 1 -1 1 -1 1 -1 1 -1];
[H1,W]=freqz(H1_QMF,1);
```

These filters are not very frequency selective (because their order is low), and will therefore allow important aliasing in each band. This is confirmed by passing our chirp signal through the corresponding filter bank (Fig. 3.17).

```
% LF band
H0_output=filter(H0_QMF,1,input_signal);
subband_0=H0_output(1:2:end);
upsampled(1:2:2*length(subband_0))=2*subband_0;
G0_QMF=H0_QMF;
G0_output=filter(G0_QMF,1,upsampled);

% HF band
H1_output=filter(H1_QMF,1,input_signal);
subband_1=H1_output(1:2:end);
upsampled(1:2:2*length(subband_0))=2*subband_1;
G1_QMF=-H1_QMF;
G1_output=filter(G1_QMF,1,upsampled);
```



```
synt_signal=G0_output+G1_output;
```

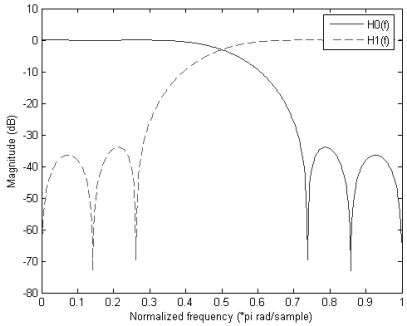

Fig. 3.16 Left: Spectrogram of the output of the 2-channel filter bank; Right: error signal waveform

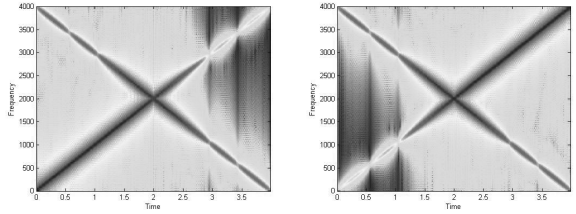

Fig. 3.17 Left: Spectrogram of the output of the LF channel of the QMF filter bank; Right: Spectrogram of the output of the HF channel. Important aliasing can be observed.

However, perfect reconstruction is now achieved, because the QMF analysis and synthesis filters are such that the aliasing in each band sum up to zero [10] (Fig. 3.18).

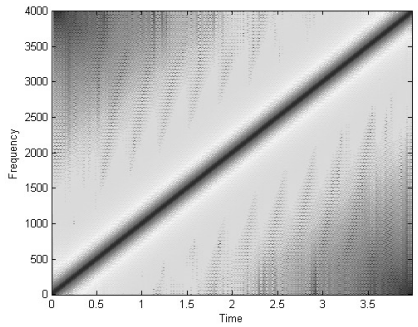

Fig. 3.18 Spectrogram of the output of the 2-channel QMF filter bank

3.2.3 32-Channel Pseudo-QMF filter bank

We now build a 32-channel PQMF filter bank, as implemented in the MPEG-1 Layer-I norm, and check its perfect reconstruction capability.

MATLAB function involved:

- `hn = PQMF32_prototype` returns in `hn` the impulse response of the prototype low-pass symmetric filter of length 512 for building a 32-channel PQMF filter bank. This filter is used in the MPEG-1 Layer-I coder. Its bandpass is Fs/2/32/2=344.5 Hz and it satisfies the PR condition.

We first build the filter bank from the prototype filter and examine the magnitude frequency responses of all filters (Fig. 3.19, left). Clearly, PQMF filters are not ideal band-pass filters with bandwidth=689 Hz, but

[10] More precisely, very close to 0.

their response only overlaps with their two neighboring filters (hence their name of "pseudo" QMF filters). The total bandwidth of each filter is indeed about two times 689 Hz. Notice the filter gain is 15 dB, i.e. 20*log10(32)/2. As a result, passing twice through the filter produces a gain of 32, which compensates for the decimation by 32. In the next lines, it is thus not necessary to multiply upsampled sub-band signals by 32.

```
% Load the prototype lowpass filter
hn=PQMF32_prototype;

% Build 32 cosine modulated filters centered on normalized
% frequencies Fi=(2*i+1)/64 *1/2
PQMF32_Gfilters = zeros(32, 512);
for i = 0:31
    t2 = ((2*i+1)*pi/(2*32))*((0:511)+16);
    PQMF32_Gfilters(i+1,:) = hn.*cos(t2);
end
PQMF32_Hfilters=fliplr(PQMF32_Gfilters);

for i = 1:32
    [H,W]=freqz(PQMF32_Hfilters(i,:),1,512,44100);
    plot(W,20*log10(abs(H))); hold on
end
xlabel('Frequency (Hz)'); ylabel('Magnitude (dB)');
set(gca,'xlim',[0 22050]);
ylabel('Magnitude (dB)');
hold off;
```

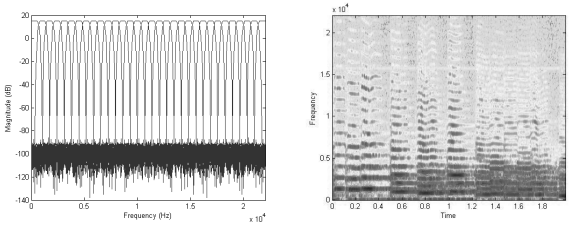

Fig. 3.19 Left: Magnitude frequency response of the 32 analysis filters of a PQMF filter bank; Right: Spectrogram of an audio signal containing a few notes played on the violin

Let us now check the output of the PQMF filter bank when fed with 2 seconds of violin monophonic signal sampled at 44.100 Hz (Fig. 3.19, right). As revealed by listening sub-band 3, isolated sub-band signals are very much aliased, because each PQMF filter is not ideal (Fig. 3.20, left).

```
[input_signal,Fs]=wavread('violin.wav');
```

```
output_signal=zeros(size(input_signal));

for i=1:32

    Hi_output=filter(PQMF32_Hfilters(i,:),1,input_signal);
    subband_i=Hi_output(1:32:end);
    upsampled_i(1:32:32*length(subband_i))=subband_i;

    % synthesis filters are the symmetric of the analysis
    % filters, which ensures linear phase in each sub-band
    Gi_output=filter(PQMF32_Gfilters(i,:),1,upsampled_i');
    output_signal=output_signal+Gi_output;

    if i==3
        G3_output=Gi_output;
    end;

end;
```

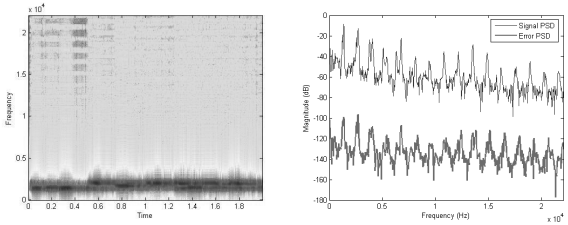

Fig. 3.20 Left: Spectrogram of sub-band 3; Right: Periodogram of the original and error signals, around sample 11000.

The PQMF filter bank, however, makes sure aliasing in adjacent bands cancels itself when sub-bands are added. The power of the reconstruction error is about 85 dB below that of the signal (Fig. 3.20, right). Notice the ouput is delayed by 511 samples (since analysis and synthesis filters have a delay of 511/2 samples).

MATLAB function involved:

- `snr = snr(signal,signal_plus_noise, max_shift, showplot)` returns the signal-to-noise ratio computed from the input signals. `Max_shift` gives the maximum time-shift (in samples) between `signal` and `signal_plus_noise`. The actual time-shift (obtained from the maximum of the cross-correlation between both signals) is taken into account

to estimate the noise. If `showplot` is specified, then the signal, signal_plus_noise, and error are plotted, and the SNR is printed on the plot.

```
error=output_signal(512:end)-input_signal(1:end-511);

[signal_psd,w]=periodogram(input_signal(11001:12024),…
                    hamming(1024));
[error_psd,w]=periodogram(error(11001:12024),…
                    hamming(1024));
plot(w/pi*22050,10*log10(signal_psd));
hold on;
plot(w/pi*22050,10*log10(error_psd),'r','linewidth',2);
hold off;

snr_PQMF=snr(input_signal(1:end-511),output_signal(512:end),0)
```

```
snr_PQMF =  84.2443
```

3.2.4 Filter banks and Lapped Transforms

We now show how analysis and synthesis filters can be implemented using block transforms.

When the length of the filters used in *M*-channel sub-band filters are exactly *M* samples, these operations reduce to a linear transform of successive non-overlapping *M* sample frames of the input signal. A 4-sample DFT, for instance, can implement a 4-channel filter bank whose sub-band filters impulse responses are the time-reversed of the lines of the 4x4 DFT matrix. Applying it to our chirp is straightforward.

```
Fs=8000;
input_signal=chirp((1:4*Fs)/Fs,0,4,4000);

for i=1:length(input_signal)/4

    % creating a column vector with 4 samples
    input_frame=input_signal(4*(i-1)+1:4*(i-1)+4)';

    % producing one sample in each downsampled sub-band, i.e.
    % band-pass filtering and downsampling all sub-bands in
    % one operation.
    subbands=fft(input_frame);

    % producing four samples of the filter bank output, i.e.
    % upsampling, band-pass filtering all sub-bands, and
    % summing them in one operation.
    output_frame=ifft(subbands);

    % storing the output column vector in the output signal
    output_signal(4*(i-1)+1:4*(i-1)+4)=output_frame';

end;

%soundsc(output_signal,Fs);
```

Obviously, this will return the original chirp. Since the underlying filters have complex coefficients, however, each sub-band signal is complex. Moreover, this type of filter bank is not very frequency selective, as shown in Fig. 3.21. The frequency overlap between adjacent bands is about half the main lobe band-pass (as in the previous section on PQMF), but the side lobes are very high. This does not make it a good candidate for sub-band coding.

```
tmp=[1 ; exp(-j*pi/2) ; exp(-j*pi) ; exp(-j*3*pi/2)];
DFT_matrix_4x4=vander(tmp);

for i=1:4
    [H,W]=freqz(fliplr(DFT_matrix_4x4(i,:)),1,'whole');
    plot(W/pi,max(20*log10(abs(H)),-50)); hold on
end
```

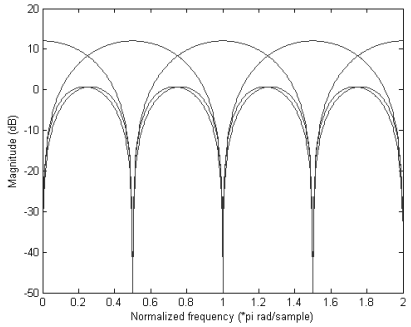

Fig. 3.21 Magnitude frequency response of the 4 channels of a DFT filter bank. Notice we show the responses in the full [0,1] frequency range, as these frequency responses are not two-sided.

In general, the length of the impulse responses of the analysis and synthesis filters used in sub-band coders is higher than the number *M* of channels. The filtering operations, however, can still be implemented as the multiplication of *L*-sample frames with *LxM* or *MxL* matrices. For example, the 32-channel PQMF filter-bank introduced in the previous Section, in which the length *N* of the impulse response of each filter is 512 samples, can be efficiently implemented as follows (which is very much similar to

the implementation of our previous DFT-based filter bank, with the addition of overlap). Obviously we get the same results as before (Fig. 3.22).

```
[input_signal,Fs]=wavread('violin.wav');

% Block-based sub-band filtering
input_frame=zeros(512,1);
output_signal=zeros(size(input_signal));

for i=1:(length(input_signal)-512+32)/32

    % Overlap input_frames (column vectors)
    input_frame=input_signal((i-1)*32+1:(i-1)*32+512);

    % Analysis filters and downsampling
    % Since PQMF H filters are the time-reversed G filters,
    % we use the G filters matrix to simulate analysis
    % filtering
    subbands_frame_i = PQMF32_Gfilters*input_frame;

    % Synthesis filters
    output_frame = PQMF32_Gfilters'*subbands_frame_i;

    % Overlap output_frames (with delay of 511 samples)
    output_signal((i-1)*32+1:(i-1)*32+512)= ...
        output_signal((i-1)*32+1:(i-1)*32+512)+output_frame;

end

error=output_signal(512:end)-input_signal(1:end-511);
```

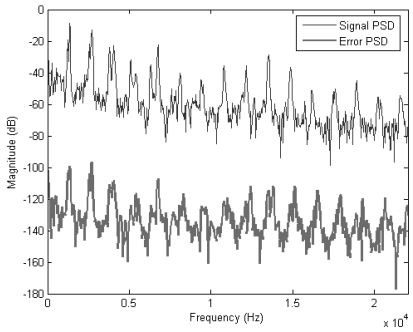

Fig. 3.22 Periodogram of the original and error signals, around sample 11000

3.2.5 Perceptual audio coding

The sub-band filtering process developed in the previous Sections transforms the original stream of samples at sampling frequency Fs into 32 parallel sub-bands sampled at Fs/32. It still needs a quantization and coding step to produce overall bit-rate compression.

Quantizing sub-band samples uniformly[11] does not allow much transparency at low bit rates, as shown in the 4-bits per sub-band sample trial below (compression factor = 4). The resulting output signal is degraded; its spectrogram is given in Fig. 3.23.

```
% Build the PQMF H and G filters
hn=PQMF32_prototype;
PQMF32_Gfilters = zeros(32, 512);
for i = 0:31
    t2 = ((2*i+1)*pi/(2*32))*((0:511)+16);
    PQMF32_Gfilters(i+1,:) = hn.*cos(t2);
end

[input_signal,Fs]=wavread('violin.wav');

% Block-based sub-band analysis filtering
input_frame=zeros(512,1);
output_signal=zeros(size(input_signal));

n_frames=(length(input_signal)-512+32)/32;
for i=1:n_frames

     % Overlap input_frames (column vectors)
     input_frame=input_signal((i-1)*32+1:(i-1)*32+512);

     % Analysis filters and downsampling
     % NB: we put sub-band signals in columns
     subbands(i,:) = (PQMF32_Gfilters*input_frame)';

     % Uniform quantization on 4 bits, using a mid-thread
       quantizer in [-1,+1]
     n_bits = 4;
     alpha = 2^(n_bits-1);
     quantized_subbands(i,:) =…
       (floor(alpha*subbands(i,:)+0.5))/alpha; % mid-thread

     % Synthesis filters
     output_frame = PQMF32_Gfilters'*quantized_subbands(i,:)';

     % Overlap output_frames (with delay of 511 samples)
     output_signal((i-1)*32+1:(i-1)*32+512)= ...
         output_signal((i-1)*32+1:(i-1)*32+512)+output_frame;
```

[11] We use a *mid-thread* quantizer here, as opposed to the more classical *mid-rise* quantizer (see Section 2.1.1), so as to encode low level signals to 0. Failing to do so quickly creates *musical noise*, caused by high-frequency changes in the LSB for low-level signals. Notice we do not use the `uencode` and `udecode` functions provided by MATLAB, which do not implement a true mid-thread quantizer.


```
end
```

The output signal is strongly distorted. The SNR falls down to 10.3 dB.

```
% NB: no delay compensation required, as the first sub-band
% samples produced by the lapped transform correspond to the
% 512th original samples.

error=output_signal-input_signal;

[signal_psd,w]=periodogram(input_signal(11001:12024),…
    hamming(1024));
[error_psd,w]=periodogram(error(11001:12024),…
    hamming(1024));
plot(w/pi*22050,10*log10(signal_psd));
hold on;
plot(w/pi*22050,10*log10(error_psd),'r','linewidth',2);
hold off;

snr_4bits=snr(input_signal(512:end-512),…
              output_signal(512:end-512),0)
```

```
snr_4bits = 10.3338
```

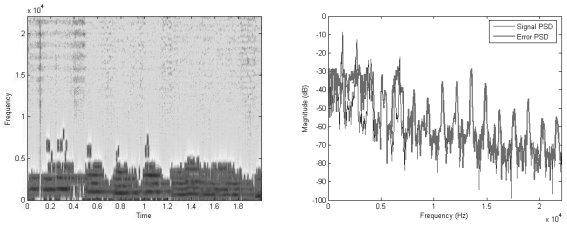

Fig. 3.23 Sub-band coding using fixed 4-bit uniform quantization. Left: Spectrogram; Right: The periodogram of the original and error signals (computed here around sample 11000) are almost identical, except for low frequencies.

One can see that this fixed [-1,+1] quantizer range does not adequately account for the variation of sub-band signal level across sub-bands, as well as in time (Fig. 3.24, left).

```
plot(quantized_subbands(100:200,2)); hold on;
plot(subbands(100:200,2),'--r'); hold off;
```

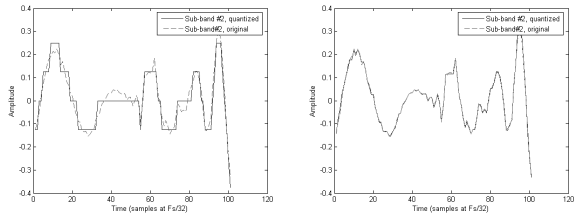

Fig. 3.24 Excerpt from sub-band #2 before and after quantization. Left: using fixed 4-bit uniform quantization; Right: using level-adaptive 4-bit uniform quantization.

An obvious means of enhancing its quality is therefore to apply a scale factor to each sub-band quantizer. As in the MPEG-1 Layer-I coder, we compute a new scale factor every 12 sub-band sample (i.e. every 32x12 = 384 sample at the original sample rate). The resulting quantization errors are much reduced ((Fig. 3.24, right).

```
% Adaptive quantization per blocks of 12 frames
n_frames=fix(n_frames/12)*12;
for k=1:12:n_frames

    % Computing scale factors in each 12 samples sub-band
    % chunk
    [scale_factors,tmp]=max(abs(subbands(k:k+11,:)));

    % Adaptive uniform quantization on 4 bits, using a
    % mid-thread quantizer in [-Max,+Max]
    for j=1:32 % for each sub-band

        n_bits = 4;
        alpha = 2^(n_bits-1)/scale_factors(j);
        quantized_subbands(k:k+11,j) = ... % mid-thread
            (floor(alpha*subbands(k:k+11,j)+0.5))/alpha;

    end;

end;
```

The resulting signal is now of much higher quality, as shown on Fig. 3.25. The overall SNR has increased to 25 dB.

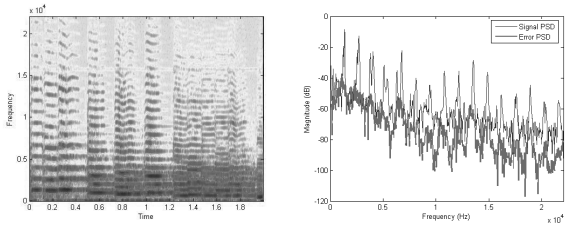

Fig. 3.25 Sub-band coding using adaptive 4-bit uniform quantization. Left: Spectrogram; Right: Periodogram of the original and error signals, around sample 11000.

```
snr_4bits_scaled=snr(input_signal(512:end-512),...
                     output_signal(512:end-512),0)
```

```
snr_4bits_scaled =   25.0629
```

The ultimate refinement, which is by far the most effective and results from years of audio research, consists in accepting more quantization noise (by allocating less bits, and thereby accepting a lower SNR) in frequency bands where it will not be heard, and using these extra bits for more perceptually prominent bands. The required perceptual information is provided by a psycho-acoustical model.

For any 512-sample frame taken from the input signal, the MPEG-1 Layer-I psycho-acoustical model computes a global masking threshold, obtained by first detecting prominent tonal and noise maskers, separately, and combining their individual thresholds. The maximum of this global threshold and the absolute auditory threshold is then taken as the final threshold (Fig. 3.26, top).

MATLAB function involved:

- `function [SMR, min_threshold_subband, masking_threshold] = MPEG1_psycho_acoustic_model1(frame)`

Computes the masking threshold (in dB) corresponding to psycho-acoustic model #1 used in MPEG-1 Audio (cf ISO/CEI norm 11172-3:1993 (F), pp. 122-128). Input `frame` length should be 512 samples.

`min_threshold_subband` returns the minimun of `masking threshold` in each of the 32 sub-bands. `SMR` returns 27 signal-to-mask ratios (in dB); `SMR`(28-32) are not used.

```
frame=input_signal(11001:11512);
[SMR, min_threshold,frame_psd_dBSPL]= ...
    MPEG1_psycho_acoustic_model1(frame);
% NB: the power levels returned by this function assume that a
full-scale signal (in [-1,+1]) corresponds to 96 DB SPL

f = (0:255)/512*44100;
auditory_threshold_dB = 3.64*((f/1000).^-0.8) - ...
    6.5*exp(-0.6.*((f/1000)-3.3).^2) + 0.001*((f/1000).^4);
plot(f, frame_psd_dBSPL, f, min_threshold, 'r', ...
     f, auditory_threshold_dB, 'k');
hold off;
axis([0 22050 -20 100]);
legend('Signal PSD', 'Min. threshold per sub-band',…
       'Absolute threshold');
xlabel('Frequency (Hz)'); ylabel('Magnitude (dB)');
```

Signal-to-Mask Ratios (SMR) are computed for each band, in a very conservative way: as the ratio of the maximum of the signal PSD to the minimum of the masking threshold in each band. Bit allocation is performed by an iterative algorithm which gives priority to sub-bands with higher SMR. The resulting SNR in each sub-band should be greater than or equal to the SMR, so as to push the noise level below the masking threshold (Fig. 3.26, bottom).

MATLAB function involved:

- `function [N_bits,SNR]= MPEG1_bit_allocation(SMR,bit_rate)`

Implements a simplified bit allocation greedy algorithm. `SMR` is the signal-to-mask ratios in each sub-band, as defined by the MPEG1 psycho-acoustic model. `bit_rate` is in kbits/s. `N_bits` is the number of bits in each sub-band. `SNR` is the maximum SNR in each sub-band after quantization, i.e. the SNR assuming each sub-band contains a full-range sinusoid. `N_bits` and `SNR` are set to zero for sub-bands 28 to 32.

```
% Allocating bits for a target bit rate of 192 kbits/s
% (compression ratio = 4)
[N_bits, SNR] = MPEG1_bit_allocation(SMR, 192000);

stairs((0:32)/32*22050,[SMR SMR(32)]);
hold on;
stairs((0:32)/32*22050,[SNR SNR(32)],'--');
axis([0 22050 -20 100]);
legend('SMR', 'SNR');
xlabel('Frequency (Hz)'); ylabel('Magnitude (dB)');
```


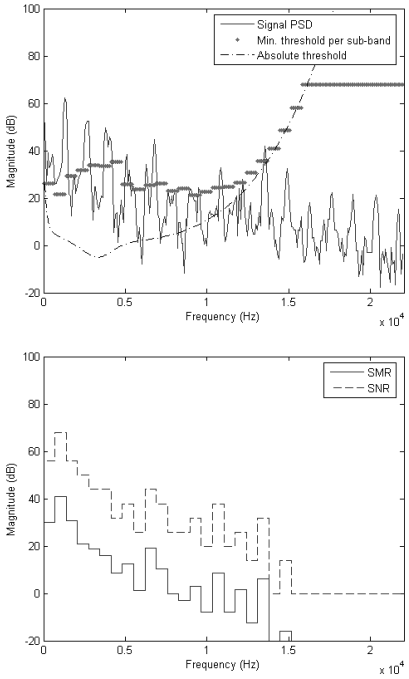

Fig. 3.26 The MEG-1 psychoacoustic model #1. Top: Masking threshold per sub-band; Bottom: Signal-to-Mask Ratios (SMR), and the maximum (theoretical) SNR obtained after bit allocation. SNR is systematically above SMR, so as to make quantization noise inaudible[12].

[12] Notice this is only made possible by the fact that we have allocated enough bits for quantizing frames. When the allowed bit rate decreases, SNR can be lower than SMR.

Let us finally test this on the complete signal, adding perceptual bit allocation to adaptive uniform quantization. Notice we simplify the quantization scheme here, compared to MPEG-1, by considering quantization with any number of bits in the range [0…16].

```
% Adaptive quantization per blocks of 12 frames
n_frames=fix(n_frames/12)*12;
for k=1:12:n_frames

    % Computing scale factors in each 12 samples sub-band
    % chunk
    [scale_factors,tmp]=max(abs(subbands(k:k+11,:)));

    % Computing SMRs [13]
    frame=input_signal(176+(k-1)*32:176+(k-1)*32+511);
    SMR = MPEG1_psycho_acoustic_model1(frame);

    N_bits = MPEG1_bit_allocation(SMR, 192000);

    % Adaptive perceptual uniform quantization, using a mid-
    % thread quantizer in [-Max,+Max]
    for j=1:32 % for each sub-band

        if N_bits(j)~=0
            codes= uencode(subbands(k:k+11,j),N_bits(j),...
                scale_factors(j),'signed');
            quantized_subbands(k:k+11,j) = ...
                udecode(codes,N_bits(j),scale_factors(j));
        else
            quantized_subbands(k:k+11,j) = 0;
        end;

    end;

end;
```

Quantization noise is now very small in some prominent sub-bands, like sub-band #2 (Fig. 3.27, left). It is also more important in other sub-bands, like sub-band #20 (Fig. 3.27, right).

[13] The input frame for the psycho-acoustic model is delayed by 176 samples, as it should be centred in the middle of the local block of 12 sub-band samples, and the first sub-band sample corresponds to original sample 256 (actually, to sample 512, but the analysis filter introduces a delay of 256 samples). So the centre of the first frame should be on original sample 256+11*32/2=432, and the beginning of this frame falls on sample 11*32/2=176.

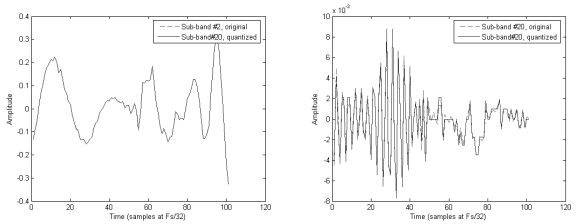

Fig. 3.27 Excerpts from sub-bands #2 and #20

The overall SNR has increased to 36.6 dB. The perceptual SNR is actually much higher, since most of the noise cannot be heard. The resulting signal is high-quality audio (Fig. 3.28).

```
snr_scaled_perceptual=snr(input_signal(512:end-512),...
    output_signal(512:end-512),0)
```

```
snr_scaled_perceptual =   36.6457
```

The price to pay is that the scale factors and the number of bits per sub-band must now be stored, every 12 frames (i.e. every 12*32 sub-band samples). In the MPEG-1 norm, scale factors are expressed in dB and quantized on 6 bits each. This comes from the fact that the ear perceives loudness as the log of the energy, and has about 96 dB of hearing dynamics with a sensitivity threshold of about 1 dB. With 6 bits, the quantization step is 96/64 dB and the error lies in [-96/64/2, +96/64/2], which is below the 1 dB threshold. Assuming 4 bits are required for encoding the number of bits used in each of the 27 first sub-bands (sub-bands 28 to 32 are ignored in MPEG1), this leads to 27*10=270 bits every 12 frames[14]. It is easy to obtain the number of bits used for the last block of 12 frames in our audio test, and the related bit-rate.

```
bits_per_block=sum(N_bits)*12+270
bit_rate=bits_per_block*44100/384
```

```
bits_per_block =   1674
```

[14] In practice, MPEG-1 does not even allow all integer values in [1,16] for the number of bits in each band, which makes it possible to quantize it to less then 4 bits. The total number of bits used for bit allocation can then be reduced to 88 instead of 4*27=108.

```
bit_rate =  1.9225e+005
```

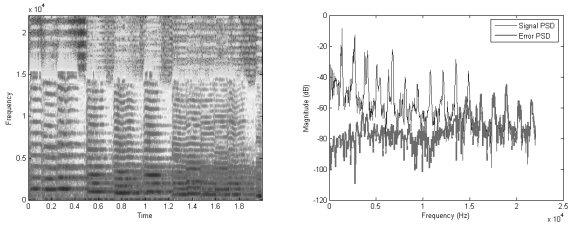

Fig. 3.28 Sub-band coding using adaptive perceptual uniform quantization. Left: Spectrogram; Right: Periodogram of the original and error signals, around sample 11000.

3.3 Going further

Unfortunately, MATLAB simulation code for audio coders is not widespread, with the remarkable exception of the toolbox designed by F. Petitcolas at Cambridge University (Petitcolas 2003).

Several other implementations (not in MATLAB) are available from the MPEG homepage (Chariglione, 2007) and the MPEG.ORG website (MPEG TV 1998).

3.4 Conclusion

Although a bit tricky to implement correctly, perceptual audio coders are based on simple principles: perfect reconstruction filter banks and sub-band quantization driven by the masking effect.

If audio compression is one of the most prominent applications of sub-band (or transform) coding, it is by far not the only one. As early as in the 1950s, the idea of using the frequency-dependent sensitivity of the human eye to color (they eye is mostly sensitive to luminance, much less to chrominance) led to the definition of the NTSC encoding system, in which most of the blue signal is discarded, keeping most of the green and only some of the red. More recently, the JPEG and MPEG/video coding standards were also based on transform coding, using the Discrete Cosine Transform (see Chapters 8 and 9).

3.5 References

Chariglione L (2007) The MPEG home page [online] Available: http://www.chiariglione.org/mpeg/ [17/04/07]

Crochiere RE, Rabiner LR (1983) Multirate Digital Signal Processing. Englewood Cliffs, NJ: Prentice Hall.

Esteban D, Galand C (1977) Application of quadrature mirror filters to split-band voice coding schemes. Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing. Hartford, Connecticut, pp 191–195

Johnston JD (1980) A filter family designed for use in quadrature mirror filter banks. Proc. IEEE International Conf. on Acoustics, Speech, and Signal Processing, pp. 291–294

Malvar H (1990) Lapped transforms for efficient transform/subband coding. IEEE Trans. Acoust., Speech, Signal Processing, 38, pp 969–978

MPEG TV (1998) MPEG Pointers and Ressources [online] Avaliable: http://www.mpeg.org [17/04/07]

Nussbaumer HJ, Vetterli M (1984) Pseudo quadrature mirror filters. Digital Signal Processing, pp 8–12

Painter T, Spanias A (2000) Perceptual Coding of Digital Audio. Proceedings of the IEEE 88-4, pp 451–512

Pan D (1995) A tutorial on MPEG audio compression. IEEE multimedia magazine 2–2: 60-74.

Petitcolas N (2003) MPEG for Matlab, [online] Available: http://www.petitcolas.net/fabien/software/mpeg/index.html [17/04/07]

Princen J, Bradley A (1986) Analysis/synthesis filter bank design based on time domain aliasing cancellation. IEEE Trans. Acoust. Speech, Signal Processing 34, pp 1153–1161

Vaidyanathan P.P (1993) Multirate Systems and Filter Banks. Englewood Cliffs, NJ: Prentice-Hall

Chapter 4

How does a dictation machine recognize speech?

This Chapter is not about how to wreck a nice beach[1]

T. Dutoit (°), L. Couvreur (°), H. Bourlard (*)

(°) Faculté Polytechnique de Mons, Belgium
(*) Ecole Polytechnique Fédérale de Lausanne, Switzerland

There is magic (or is it witchcraft?) in a speech recognizer that transcribes continuous radio speech into text with a word accuracy of even not more than 50%. The extreme difficulty of this task, tough, is usually not perceived by the general public. This is because we are almost deaf to the infinite acoustic variations that accompany the production of vocal sounds, which arise from physiological constraints (co-articulation), but also from the acoustic environment (additive or convolutional noise, Lombard effect), or from the emotional state of the speaker (voice quality, speaking rate, hesitations, etc.)[2]. Our consciousness of speech is indeed not stimulated until after it has been processed by our brain to make it appear as a sequence of meaningful units: phonemes and words.

In this Chapter we will see how statistical pattern recognition and statistical sequence recognition techniques are currently used for trying to mimic this extraordinary faculty of our mind (4.1). We will follow, in Section 4.2, with a MATLAB-based proof of concept of word-based automatic speech recognition (ASR) based on Hidden Markov Models (HMM), using a bigram model for modeling (syntactic-semantic) language constraints.

[1] It is, indeed, about *how to recognize speech*.
[2] Not to mention inter-speaker variability, nor regional dialects.

4.1 Background – Statistical Pattern Recognition

Most modern ASR systems have a pipe-line block architecture (see Fig. 4.1).

The acoustical wave is first digitized, usually with a sampling frequency of 8 kHz for telephone applications and 16 kHz for multimedia applications. A *speech detection* module then detects segments of speech activity in the digital signal: only these segments that compose the speech signal are transmitted to the following block. The purpose of speech detection is to reduce the computational cost and the probability of ASR error when unexpected acoustic events happen. Doing this automatically, however, is by itself a difficult problem. Speech detection is sometimes implemented manually: the speaker is asked to push a button while speaking in order to activate the ASR system (*push-to-talk* mode).

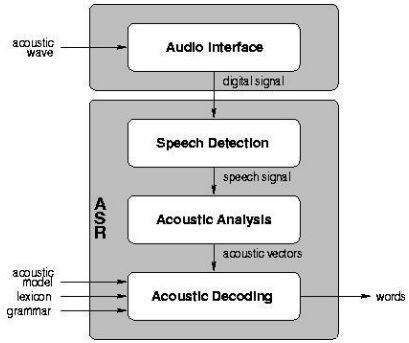

Fig. 4.1 Classical architecture of an automatic speech recognition system

The *acoustical analysis* module processes the speech signal in order to reduce its variability while preserving its linguistic information. A time-frequency analysis is typically performed (using frame-based analysis, with 30 ms frames shifted every 10 ms), which transforms the continuous input waveform into a sequence $X = [x(1), x(2), \ldots, x(N)]$ of *acoustic fea-*

ture vectors x(n)[3]. The performances of ASR systems (in particular, their *robustness*, i.e. their resistance to noise) are very much dependent on this formatting of the acoustic observations. Various types of feature vectors can be used, such as the LPC coefficients described in Chapter 1, although specific feature vectors, such as the Linear Prediction Cepstral Coefficients (LPCC) or the Mel Frequency Cepstral Coefficients (MFCC; Picone 1993), have been developed in practice for speech recognition, which are somehow related to LPC coefficients.

The *acoustic decoding* module is the heart of the ASR system. During a training phase, the ASR system is presented with several examples of every possible word, as defined by the *lexicon*. A statistical model (4.1.1) is then computed for every word such that it models the distribution of the acoustic vectors. Repeating the estimation for all the words, we finally obtain a set of statistical models, the so-called *acoustic model*, which is stored in the ASR system. At run-time, the acoustic decoding module searches the sequence of words whose corresponding sequence of models is the "closest" to the observed sequence of acoustic feature vectors. This search is complex since neither the number of words, nor their segmentation, are known in advance. Efficient decoding algorithms constrain the search for the best sequence of words by a *grammar*, which defines the authorized, or at least the most likely, sequence of words. It is usually described in terms of a statistical model: the *language model*.

In large vocabulary ASR systems, it is hard if not impossible to train separate statistical models for all words (and even to gather the speech data that would be required to properly train a word-based acoustic model). In such systems, words are described as sequences of phonemes in a *pronunciation lexicon*, and statistical modeling is applied to phonemic units. Word-based models are then obtained by concatenating the phoneme-based models. Small vocabulary systems (<50 words), on the contrary, usually consider words as basic acoustic units and therefore do not require a pronunciation lexicon.

4.1.1 The statistical formalism of ASR

The most common statistical formalism of ASR[4], which we will use throughout this Chapter, aims to produce the *most probable* word sequence

[3] Although *x(n)* is a vector, it will not be written with a bold font in this Chapter, to avoid overloading all equations.

[4] There are numerous textbooks that explain these notions in detail. See for instance (Gold and Morgan 2000), (Bourlard and Morgan 1994) or (Bourlard 2007). For a more general introduction to pattern recognition, see also (Polikar 2006) or the more complete (Duda *et al.* 2000).

W^* given the acoustic observation sequence X. This can be expressed mathematically by the so-called *Bayesian*, or *Maximum a Posteriori* (MAP) decision rule as:

$$W^* = \arg\max_{W_i} P(W_i \mid X, \Theta) \quad (4.1)^5$$

where W_i represents the i-th possible word sequence and the conditional probability is evaluated over all possible word sequences[6], and Θ represents the set of parameters used to estimate the probability distribution.

Since each word sequence W_i may be realized as an infinite number of possible acoustic realizations, it is represented by its model $M(W_i)$, also written M_i for the sake of simplicity, which is assumed to be able to produce all such possible acoustic realizations. This yields:

$$M^* = \arg\max_{M_i} P(M_i \mid X, \Theta) \quad (4.2)$$

where M^* is (the model of) the sequence of words representing the linguistic message in input speech X, M_i is (the model of) a possible word sequence W_i, $P(M_i \mid X,\Theta)$ is the posterior probability of (the model of) a word sequence given the acoustic input X, and the maximum is evaluated over all possible models (i.e., all possible word sequences).

Bayes' rule can be the applied to (4.2), yielding:

$$P(M_i \mid X, \Theta) = \frac{P(X \mid M_i, \Theta) P(M_i \mid \Theta)}{P(X \mid \Theta)} \quad (4.3)$$

[5] In equation (4.1), W_i and X are not random variables: they are values taken by their respective random variables. As a matter of fact, we will often use in this Chapter a shortcut notation for probabilities, when this does not bring confusion. The probability $P(A=a|B=b)$ that a discrete random variable A takes value a given the fact that random variable B takes value b will simply be written $P(a|b)$. What is more, we will use the same notation when A is a *continuous* random variable for referring to *probability density* $p_{A|B=b}(a)$).

[6] It is assumed here that the number of possible word sequences is finite, which is not true for natural languages. In practice, a specific component of the ASR, the *decoder*, takes care of this problem by restricting the computation of (4.1) for a limited number of most probable sequences.

where $P(X \mid M_i, \Theta)$ represents the contribution of the so-called *acoustic model* (i.e., the likelihood that a specific model M_i has produced the acoustic observation X), $P(M_i \mid \Theta)$ represents the contribution of the so-called *language model* (i.e., the a priori probability of the corresponding word sequence), and $P(X \mid \Theta)$ stands for the a priori probability of the acoustic observation. For the sake of simplicity (and tractability of the parameter estimation process), state-of-the-art ASR systems always assume independence between the acoustic model parameters, which will now be denoted Θ_A and the parameters of the language model, which will now be denoted Θ_L.

Based on the above, we thus have to address the following problems:

- *Decoding (recognition):* Given an unknown utterance X, find the most probable word sequence W^* (i.e., the most probable word sequence model M^*) such that:

$$M^* = \arg\max_{M_i} \frac{P(X \mid M_i, \Theta_A) P(M_i \mid \Theta_L)}{P(X \mid \Theta_A, \Theta_L)} \tag{4.4}$$

  Since during recognition all parameters Θ_A and Θ_L are frozen, probability $P(X \mid \Theta_A, \Theta_L)$ is constant for all hypotheses of word sequences (i.e., for all choices of i) and can thus be ignored, so that (4.4) simplifies to:

$$M^* = \arg\max_{M_i} P(X \mid M_i, \Theta_A) P(M_i \mid \Theta_L) \tag{4.5}$$

- *Acoustic modeling*: Given (the model of) a word sequence, M_i, estimate the probability $P(X \mid M_i, \Theta_A)$ of the unknown utterance X.

  This is typically carried out using Hidden Markov Models (HMM; see Section 4.1.3). It requires to estimate the acoustic model Θ_A. At training time, a large amount of training utterances X_j (j = 1, ... , J) with their associated models M_j are used to estimate the optimal acoustic parameter set Θ_A^*, such that:

$$\begin{aligned} \Theta_A^* &= \arg\max_{\Theta_A} \prod_{j=1}^{J} P(X \mid M_i, \Theta_A) \\ &= \arg\max_{\Theta_A} \sum_{j=1}^{J} \log(P(X \mid M_i, \Theta_A)) \end{aligned} \tag{4.6}$$

  which is referred to as the *Maximum Likelihood* (ML), or *Maximum Log Likelihood criterion*[7].

- *Language modeling*: The goal of the language model is to estimate prior probabilities of sentence models $P(M_i \mid \Theta_L)$.

  At training time, the language model parameters Θ_L are commonly estimated from large text corpora. The language model is most often formalized as word-based Markov models (See Section 4.1.2), in which case Θ_L is the set of transition probabilities of these chains, also known as *n-grams*.

4.1.2 Markov models

A *Markov model* is the simplest form of a *Stochastic Finite State Automaton* (SFSA). It describes a sequence of observations $X = [x(1), x(2), \ldots, x(N)]$ as the output of a finite state automaton (Fig. 4.2) whose internal states $\{q_1, q_2, \ldots, q_K\}$ are univocally associated with possible observations $\{x_1, x_2, \ldots, x_K\}$ and whose state-to-state transitions are associated with probabilities: a given state q_k always outputs the same observation x_k, except initial and final states (q_I and q_F, which output nothing); the *transition probabilities* from any state sum to one. The most important constraint imposed by a (first order) Markov model is known as the Markov property: the probability of a state (or that of the associated observation) only depends on the previous state (or that of the associated observation).

The probability of X given such a model reduces to the probability of the sequence of states $[q_I, q(1), q(2), \ldots, q(N), q_F]$ corresponding to X, i.e. to a product of transition probabilities (including transitions from state q_I and transitions to state q_F):

$$P(X) = P(q(1) \mid q_I) \left(\prod_{n=2}^{N} P(q(n) \mid q(n-1)) \right) P(q_F \mid q(N)) \tag{4.7}$$

[7] Although both criteria are equivalent, it is usually more convenient to work with the sum of *log* likelihoods. As a matter of fact, computing products of probabilities (which are often significantly lower than one) quickly exceeds the floating point arithmetic precision. Even the log of a sum of probabilities can be estimated, when needed, using log likelihoods (i.e., without having to compute likelihoods at any time), using:

$$\log(a+b) = \log(a) + \log\left(1 + e^{(\log b - \log a)}\right)$$

where *q(n)* stands for the state associated with observation *x(n)*. Given the one-to-one relationship between states and observations, this can also be written as:

$$P(X) = P(x(1)|I)\left(\prod_{n=2}^{N} P\big(x(n)\,|\,x(n-1)\big)\right)P\big(F\,|\,x(N)\big) \qquad (4.8)$$

where *I* and *F* stand for the symbolic beginning and end of *X*.

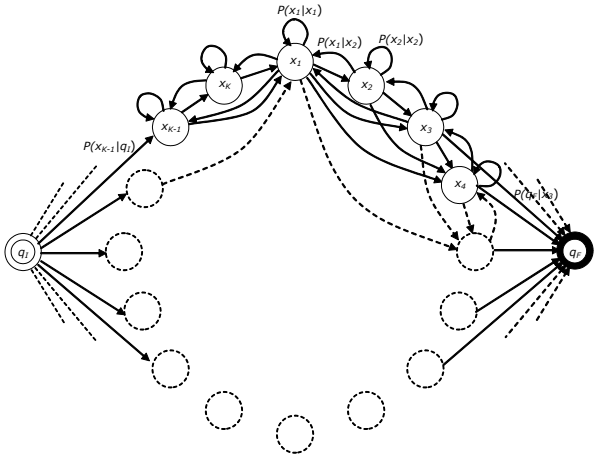

Fig. 4.2 A typical Markov model[8]. The leftmost and rightmost states are the initial and final states. Each internal state q_k in the center of the figure is associated to a specific observation x_k (and is labeled as such). Transition probabilities are associated to arcs (only a few transition probabilities are shown).

The set of parameters, represented by the (K×K)-transition probability matrix and the initial and final state probabilities:

[8] The model shown here is *ergodic*: transitions are possible from each state to all other states. In practical applications (such as in ASR, for language modeling), some transitions may be prohibited.

$$\Theta = \{P(q_k\,|\,q_l), P(q_k\,|\,q_l), P(q_F\,|\,q_l)\}, \quad \text{with } k,l \text{ in } (1,\ldots K) \qquad (4.9)$$

is directly estimated on a large amount of observation sequences (i.e., of state sequences, since states can be directly deduced from observations in a Markov model), such that:

$$\Theta^* = \arg\max_{\Theta} P(X\,|\,\Theta) \qquad (4.10)$$

This simply amounts to estimating the relative counts of observed transitions[9], i.e.:

$$P(q_k\,|\,q_l) = \frac{n_{lk}}{n_l} \qquad (4.11)$$

where n_{lk} stands for the number of times a transition from state q_l to state q_k occurred, while n_l represents the number of times state q_l was visited.

Markov models are intensively used in ASR for language modeling, in the form of *n-grams*, to estimate the probability of a word sequence W = [w(1)*,* w(2)*, . . ., w(L)*] as:

$$P(W) = P\big(w(1)\,|\,I\big)\left(\prod_{l=2}^{L} P\big(w(l)\,|\,w(l-1), w(l-2), \ldots w(l-n+1)\big)\right). \qquad (4.12)[10]$$
$$P\big(F\,|\,w(L)\big)$$

In particular, *bigrams* further reduce this estimation to:

$$P(W) = P\big(w(1)\,|\,I\big)\left(\prod_{l=2}^{L} P\big(w(l)\,|\,w(l-1)\big)\right)P\big(F\,|\,w(L)\big) \qquad (4.13)$$

In this case, each observation is a word from the input word sequence W, and each state of the model (except I and F) is characterized by a single word, which an observed word could possibly follow with a given probability.

As Jelinek (1991) pointed out: "That this simple approach is so successful is a source of considerable irritation to me and to some of my colleagues. We have evidence that better language models are obtainable, we think we

[9] This estimate is possibly smoothed in case there is not enough training data, so as to avoid forbidding state sequences not found in the data (those which are rare but not impossible).

[10] In this case, states are not associated to words, but rather to sequences of *n-1* words. Such models are called N^{th} order Markov models.

know many weaknesses of the trigram model, and yet, when we devise more or less subtle methods of improvement, we come up short."

Markov models cannot be used for acoustic modeling, as the number of possible observations is infinite.

4.1.3 Hidden Markov models

Modifying a Markov model by allowing several states (if not all) to output the same observations with state-dependent *emission probabilities* (Fig. 4.3), turns it into a hidden Markov model (HMM, Rabiner 1989). In such a model, the sequence of states cannot be univocally determined from the sequence of observations (such a SFSA is called *ambiguous*). The HMM is thus called "hidden" because there is an underlying stochastic process (i.e., the sequence of states) that is not observable, but affects the sequence of observations.

While Fig. 4.3 shows a *discrete* HMM, in which the number of possible observations is finite, *continuous* HMMs are also very much used, in which the output space is a continuous variable (often even multivariate). Emission probabilities are then estimated by assuming they follow a particular functional distribution: $P(x_m|q_k)$ is computed analytically (it can no longer be obtained by counting). In order to keep the number of HMM parameters as low as possible, this distribution often takes the classical form of a (multivariate, d-dimensional) Gaussian[11]:

$$P(x \mid q_k) = N(x, \mu_k, \Sigma_k) = \frac{1}{(2\pi)^{d/2}\left|\Sigma_k\right|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_k)^T \Sigma_k^{-1}(x-\mu_k)\right) \quad (4.14)$$

where μ_k and Σ_k respectively denote the mean vector and the covariance matrix associated with state q_k. When this model is not accurate enough, mixtures of (multivariate) Gaussians (Gaussian mixture model, GMM) are also used, which allow for multiple modes[12]:

[11] Gaussian PDFs have many practical advantages: they are entirely defined by their first two moments and are linear once derivated.

[12] It is also possible (and has proved very efficient in ASR) to use artificial neural networks (ANN) to estimate emission probabilities (Bourlard and Wellekens 1990, Bourlard and Morgan 1994). We do not examine this option here.

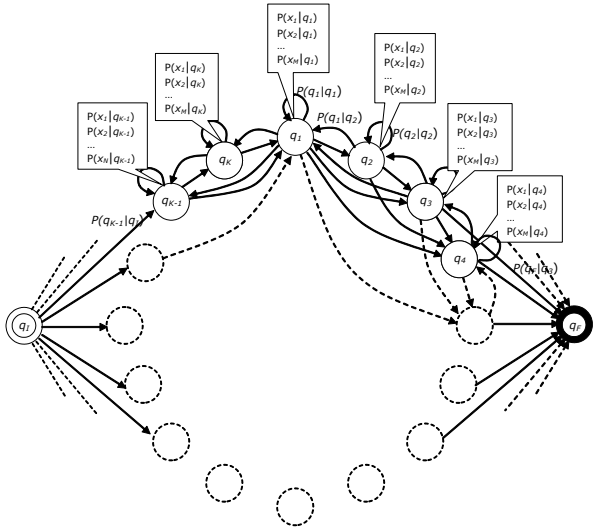

Fig. 4.3 A typical (discrete) hidden Markov model. The leftmost and rightmost states are the initial and final states. Each state q_k in the center of the figure is associated to several possible observations (here, to all observations $\{x_1, x_2, \ldots, x_M\}$) with the corresponding emission probability. Transition probabilities are associated to arcs (only a few transition probabilities are shown). The HMM is termed as *discrete* because the number of possible observations is finite.

$$P(x \mid q_k) = \sum_{g=1}^{G} c_{kg} N(x, \mu_{kg}, \Sigma_{kg}) \quad (4.15)$$

where G is the total number of Gaussian densities and c_{kg} are the mixture gain coefficients (thus representing the prior probabilities of Gaussian mixture components). These gains must verify the constraint:

$$\sum_{g=1}^{G} c_{kg} = 1 \quad \forall k = 1,...,K \tag{4.16}$$

Assuming the total number of states K is fixed, the set of parameters Θ of the model comprises all the Gaussian means and variances, gains, and transition probabilities.

Two approaches can be used for estimating $P(X \mid M, \Theta)$.

In the *full likelihood approach*, this probability is computed as a sum on all possible paths of length N. The probability of each path is itself computed as in (4.7):

$$P(X \mid M,\Theta) = \sum_{paths\ j} P\left(q_j(1) \mid q_1\right) P\left(x(1) \mid q_j(1)\right). \left(\prod_{n=2}^{N} P\left(q_j(n) \mid q_j(n-1)\right) P\left(x(n) \mid q_j(n)\right)\right) P\left(q_F \mid q_j(N)\right) \tag{4.17}$$

where $q_j(n)$ stands for the state in $\{q_1, q_2, \ldots, q_K\}$ which is associated with $x(n)$ in path j. In practice, estimating the likelihood according to (4.17) involves a very large number of computations, namely $O(NK^N)$, which can be avoided by the so-called *forward recurrence formula* with a lower complexity, namely $O(K^2N)$. This formula is based on the recursive estimation of an intermediate variable $\alpha_n(l)$:

$$\alpha_n(l) = P\left(x(1), x(2), ..., x(n), q(n) = q_l\right) \tag{4.18}$$

$\alpha_n(l)$ stands for the probability that a partial sequence $[x(1), x(2), \ldots, x(n)]$ is produced by the model in such a way that $x(n)$ is produced by state q_l. It can be obtained by using (Fig. 4.4):

$$\begin{aligned} &\alpha_1(l) = P(x(1) \mid q_l) P\left(q_l \mid q_1\right) \quad (l = 1,...,K) \\ &\text{for } n = 2,...,N \text{ (and } l = 1,...,K) \\ &\alpha_n(l) = P(x(n) \mid q_l) \sum_{k=1}^{K} \alpha_{n-1}(k) P\left(q_l \mid q_k\right) \\ &P(X \mid M,\Theta) = \alpha_{N+1}(F) = \sum_{k=1}^{K} \alpha_N(k) P\left(q_F \mid q_k\right) \end{aligned} \tag{4.19}$$

In the *Viterbi approximation approach,* the estimation of the data likelihood is restricted to the most probable path of length N generating the sequence X:

$$P(X \mid M,\Theta) \cong \max_{paths\ j} P\left(q_j(1) \mid 1\right) P(x_1 \mid q_j(1)). \left(\prod_{n=2}^{N} P\left(q_j(n) \mid q_j(n-1)\right) P\left(x_n \mid q_j(n)\right)\right) P\left(q_F \mid q_j(N)\right) \tag{4.20}$$

and the sums in (4.19) are replaced by the max operator. Notice it is also easy to memorize the most probable path given some input sequence by using (4.19) and additionally keeping in memory, for each $n = (1,...,N)$ and for each $l = (1,...,K)$, the value of k producing the highest term of $\alpha_{n+1}(l)$ in (4.19). Starting from the final state (i.e., the one leading to the highest term for $\alpha_{N+1}(F)$), it is then easy to trace back the best path, thereby associating one "best" state to each feature vector.

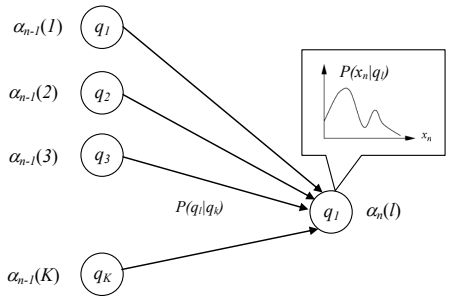

Fig. 4.4 Illustration of the sequence of operations required to compute the intermediate variable $\alpha_n(l)$

HMMs are intensively used in ASR acoustic models where every sentence model M_i is represented as a HMM. Since such a representation is not tractable due to the infinite number of possible sentences, sentence HMMs are obtained by compositing sub-sentence HMMs such as word HMMs, syllable HMMs or more generally phoneme HMMs. Words, syllables, or phonemes are then generally described using a specific HMM topology (i.e. allowed state connectivity) known as *left-to-right* HMMs (Fig. 4.5), as opposed to the general ergodic topology shown in Fig. 4.3. Although sequential signals, such as speech, are nonstationary processes, left-to-right HMMs assume that the sequence of observation vectors is a piecewise stationary process. That is, a sequence $X = [x(1), x(2), \ldots, x(N)]$

is modeled as a sequence of discrete stationary states with instantaneous transitions between these states.

4.1.4 Training HMMs

HMM training is classically based on the Maximum Likelihood criterion: the goal is to estimate the parameters of the model which maximize the likelihood of a large number of training sequences X_j (j = 1,... , J). For Gaussian HMMs (which we will examine here, as they are used in most ASR systems), the set of parameters to estimate comprises all the Gaussian means and variances, gains (if GMMs are used), and transition probabilities.

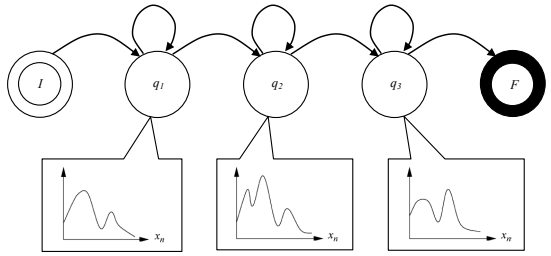

Fig. 4.5 A left-to-right continuous HMM, shown here with (univariate) continuous emission probabilities (which look like mixtures of Gaussians). In speech recognition, this could be the model of a word or of a phoneme which is assumed to be composed of three stationary parts.

Training algorithms

A solution to this problem is a particular case of the *Expectation-Maximization* (EM) algorithm (Moon 1996). Again, two approaches are possible.

In the *Viterbi approach* (Fig. 4.6), the following steps are taken:

1. Start from an initial set of parameters $\Theta^{(0)}$. With a left-to-right topology, one way of obtaining such a set is by estimating the parameters from a linear segmentation of feature vector sequences, i.e., by assuming that each training sequence X_j (j = 1,... , J) is produced by visiting each state of its associated model M_j the same amount of times. Then apply the expectation step to this initial linear segmentation.

2. (*Expectation* step) Compute transition probabilities as in (4.11). Obtain emission probabilities for state k by estimating the Gaussian parameters in (4.14) or the GMM parameters in (4.15) and (4.16) from all feature vectors associated to state k in the training sequences (see below).
2. (*Maximization* step) For all training utterances X_j and their associated models M_j find the maximum likelihood paths ("best" paths), maximizing $P(X_j|M_j)$ using the Viterbi recursion, thus yielding a new segmentation of the training data. This step is often referred to "forced alignment", since we are forcing the matching of utterances X_j on given models M_j.
3. Given this new segmentation, collect all the vectors (over all utterances X_j) associated with states q_k and reestimate emission and transition probabilities as in the expectation step. Iterate as long as the total likelihood of the training set increases or until the relative improvement falls below a pre-defined threshold.

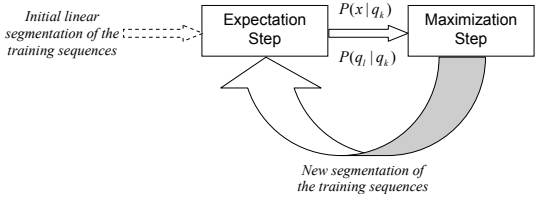

Fig. 4.6 The *Expectation-Maximization* (EM) algorithm, using the Viterbi approach.

In the *Forward-Backward*, or *Baum-Welch approach*, all paths are considered. Feature vectors are thus no longer univocally associated to states when reestimating the emission and transitions probabilities: each of them counts for some weight in the reestimation of the whole set of parameters.

The convergence of the iterative processes involved in both approaches can be proved to converge to a local optimum (whose quality will depend on the quality of the initialization).

Estimating emission probabilities

In the Viterbi approach, one needs to estimate the emission probabilities of each state q_k, given a number of feature vectors $\{x_{1k}, x_{2k}, \ldots, x_{Mk}\}$ associated to it. The same problem is encountered in the Baum-Welch ap-

proach, with feature vector partially associated to each state. We will explore the Viterbi case here, as it is easier to follow[13].

When a multivariate Gaussian distribution $N(\mu_k,\Sigma_k)$ is assumed for some state q_k, the classical estimation formulas for the mean and covariance matrix, given samples x_{ik} stored as column vectors, are:

$$\tilde{\mu}_k = \frac{1}{M}\sum_{i=1}^{M} x_{ik}$$
$$\tilde{\Sigma}_k = \frac{1}{M-1}\sum_{i=1}^{M}(x_{ik}-\tilde{\mu}_k)(x_{ik}-\tilde{\mu}_k)^T \quad (4.21)$$

It is easy to show that $\tilde{\mu}_k$ is the maximum likelihood estimate of μ_k. The ML estimator of Σ_k, though, is not exactly the one given by (4.21): the ML estimator normalizes by M instead of (M-1). However it is shown to be biased when the exact value of μ_k is not known, while (4.21) is unbiased.

When a multivariate GMM distribution is assumed for some state q_k, estimating its weights c_{kg}, means μ_{kg} and covariance matrices Σ_{kg} for g=1, ..., G as defined in (4.15), cannot be done analytically. The EM algorithm is used again for obtaining the maximum likelihood estimate of the parameters, although in a more straightforward way than above (there is not such a thing as transition probabilities in this problem). As before, two approaches are possible: the *Viterbi-EM approach*, in which each feature vectors is associated to one of the underlying Gaussians, and the *EM approach*, in which each vector is associated to all Gaussians, with some weight (for a tutorial on the EM algorithm, see Moon 1996, Bilmes 1998).

The Viterbi-EM and EM algorithms are very sensitive to the initial values chosen for their parameters. In order to maximize their chances to converge to a global maximum of the likelihood of the training data, the *k-means* algorithm is sometimes used for providing a first estimate of the parameters. Starting from an initial set of G prototype vectors, this algorithm iterates on the following steps:

1. For each feature vector x_{ik} (i=1, …, M), compute the squared Euclidian distance from the kth prototype, and assign x_{ik} to its closest prototype.
2. Replace each prototype with the mean of the feature vectors assigned to it in step 1.

Iterations are stopped when no further assignment changes occur.

[13] Details on the Baum-Welch algorithm can be found in (Bourlard, 2007).

4.2 MATLAB proof of concept: ASP_dictation_machine.m

Although speech is by essence a non-stationary signal, and therefore calls for dynamic modeling, it is convenient to start this script by examining static modeling and classification of signals, seen as a statistical pattern recognition problem. We do this by using Gaussian multivariate models in Section 4.2.1 and extend it to Gaussian Mixture Models (GMM) in Section 4.2.2. We then examine, in Section 4.2.3, the more general dynamic modeling, using Hidden Markov Models (HMM) for isolated word classification. We follow in Section 4.2.4 by adding a simple bigram-based language model, implemented as a Markov model, to obtain a connected word classification system. We end the Chapter in Section 4.2.5 by implementing a word-based speech recognition system[14], in which the system does not know in advance how many words each utterance contains.

4.2.1 Gaussian modeling and Bayesian classification of vowels

We will examine here how Gaussian multivariate models can be used for the classification of signals.

A good example is that of the classification of sustained vowels, i.e., of the classification of incoming acoustic feature vectors into the corresponding phonemic classes. Acoustic feature vectors are generally highly multidimensional (as we shall see later), but we will work in a 2D space, so as to be able to plot our results.

In this Chapter, we will work on a hypothetic language, whose phoneme set is only composed of four vowels {/a/, /e/, /i/, /u/}, and whose lexicon reduces to {"why" /uai/, "you" /iu/, "we" /ui/, "are" /ae/, "hear" /ie/, "here" /ie/}. Every speech frame can then be represented as a 2-dimensional vector of speech features in the form of pairs of formant values (the first and the second spectral formants, F_1 and F_2; see Chapter 1, Section 1.1).

Our first task will be to classify vowels, by using Gaussian probability density functions (PDF) for class models and Bayesian (MAP) decision. Let us load a database of features extracted from the vowels and words of

[14] Notice that we will not use the words *classification* and *recognition* indifferently. Recognition is indeed more complex than classification, as it involves the additional task of segmenting an input stream into segments for further classification.

this language[15]. Vowel samples are grouped in matrices of size N x 2, where each of the N rows is a training example and each example is characterized by a formant frequency pair $[F_1, F_2]$. Supposing that the whole database covers adequately our imaginary language, it is easy to compute the prior probability $P(q_k)$ of each class q_k (q_k in {/a/,/e/,/i/,/u/}). The most common phoneme in our hypothetic language is /e/.

```
load data; % vowels={a,e,i,u};

N_samples=0;
for j=1:4
    N_samples = N_samples+size(vowels{j}.training,1);
end;
for j=1:4
    prior(j) = size(vowels{j}.training,1)/N_samples;
end;
prior
```

```
prior =  0.1500    0.4000    0.1500    0.3000
```

As can be seen in Fig. 4.7 (left), our four vowel classes have serious overlap in the 2D vector space.

```
plot(vowels{1}.training(:,1),vowels{1}.training(:,2),'k+');
hold on;
plot(vowels{2}.training(:,1),vowels{2}.training(:,2),'r*');
plot(vowels{3}.training(:,1),vowels{3}.training(:,2),'gp');
plot(vowels{4}.training(:,1),vowels{4}.training(:,2),'bs');
```

Let us now assume that we are asked to identify an unknown vowel from its (F_1, F_2) features. One way of solving this problem is by performing multivariate Gaussian modeling of each class, i.e., finding the mean and covariance matrices of the data in each class.

MATLAB function involved:

- `plot_2Dgauss_pdf(mu,sigma)` plots the mean and standard deviation ellipsis of the 2D Gaussian process that has mean `mu` and covariance matrix `sigma`, in a 2D plot.

```
for j=1:4
    mu{j}=mean(vowels{j}.training)';
    sigma{j}=cov(vowels{j}.training);
    plot_gauss2D_pdf(mu{j},sigma{j})
end;
```

[15] These samples were actually generated from statistical models of the vowels, which we chose for tutorial purposes. See Appendix 1 in the **ASP_dictation_machine.m** script, and the **gendata.m** file.

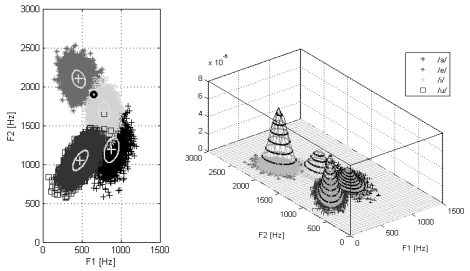

Fig. 4.7 Left: Samples of the four vowels {/a/, /e/, /i/, /u/} of our imaginary language in the (F_1,F_2) plane, superimposed with the standard deviation ellipsis of their 2D Gaussian model. Right: 2D Gaussian estimates of the PDF of these vowels in the (F_1,F_2) plane.

Fig. 4.7 shows that /i/, for instance, has its mean F_1 at 780 Hz and its mean F_2 at 1680 Hz [16]. The covariance matrix for the /i/ class is almost diagonal (the scatter plot for the class has its principal axes almost parallel to the coordinate axes, which implies that F_1 and F_2 are almost uncorrelated; see Appendix 1 of the **ASP_dictation_machine.m** file). Its diagonal elements are thus close to the square of the length of the halfmajor and halfminor axes of the standard deviation ellipsis: 76 Hz and 130 Hz, respectively.

```
mu{3}
sqrtm(sigma{3})
```

```
ans =   1.0e+003 *
    0.7814    1.6827

ans =
   75.3491   -4.4051
   -4.4051  125.5608
```

Let us estimate the likelihood of a test feature vector given the Gaussian model of class /e/, using the classical Gaussian PDF formula. The feature vector is shown as a black dot in Fig. 4.7.

```
sample=[650 1903];
```

[16] These values are the ones fixed in our imaginary language; they do not correspond to those of English vowels at all.

```
x = sample-mu{2};
likelihood = exp(-0.5* x* inv(sigma{2}) *x') / sqrt((2*pi)^2 …
             * det(sigma{2}))

plot(sample(1),sample(2),'ko','linewidth',3)
```

```
likelihood =  7.1333e-007
```

The likelihood of this vector is higher in class /i/ than in any other class (this is also intuitively obvious from the scatter plots shown previously), as shown below.

MATLAB function involved:

- `gauss_pdf(x,mu,sigma)` returns the likelihood of sample `x` (NxD) with respect to a Gaussian process with mean `mu` (1xD) and covariance `sigma` (DxD). When a set of samples is provided as input, a set of likelihoods is returned.

```
for j=1:4
    likelihood(j) = gauss_pdf(sample,mu{j},sigma{j});
end;
likelihood
```

```
likelihood =  1.0e-005 *

    0.0000    0.0713    0.1032    0.0000
```

Likelihood values are generally very small. Since we will use products of them in the next paragraphs, we will systematically prefer their log-likelihood estimates.

```
log(likelihood)
```

```
ans =  -29.3766  -14.1533  -13.7837  -36.9803
```

Since not all phonemes have the same prior probability, Bayesian (MAP) classification of our test sample is not equivalent to finding the class with maximum likelihood. Posterior probabilities *P(class|sample)* must be estimated by multiplying the likelihood of the sample by the prior of each class, and dividing by the marginal likelihood of the sample (obtained by summing its likelihood for all classes). Again, for convenience, we compute the log of posterior probabilities. The result is that our sample gets classified as /e/ ratter than as /i/, because the prior probability of /e/ is much higher than that of /i/ in our imaginary language.

```
marginal=sum(likelihood); % is a constant
log_posterior=log(likelihood)+log(prior)-log(marginal)
```



```
log_posterior =  -18.0153   -1.8112   -2.4224  -24.9258
```

Notice that the marginal likelihood of the sample is not required for classifying it, as it is a subtractive constant for all log posterior probabilities. We will not compute it in the sequel.

Multiplying likelihoods by priors can be seen as a weighting which accounts for the intrinsic frequency of occurence of each class. Plotting the posterior probability of classes in the (F_1, F_2) plane gives a rough idea of how classes are delimited (Fig. 4.7, right).

MATLAB function involved:

- `mesh_2Dgauss_pdf(mu,sigma,prior,gridx,gridy,ratioz)` plots the PDF of a 2D-Gaussian PDF in a 3D plot. `mu` (1x2) is the mean of the density, `sigma` (2x2) is the covariance matrix of the density. `prior` is a scalar used as a multiplicative factor on the value of the PSD. `gridx` and `gridy` must be vectors of the type (x:y:z) `ratioz` is the (scalar) aspect ratio on the Z axis.

```
hold on;
for j=1:4
    mesh_gauss2D_pdf(mu{j},sigma{j},prior(j),0:50:1500, ...
        0:50:3000, 7e-9);
    hold on;
end;
```

One can easily compare the performance of max likelihood vs. max posterior classifiers on test data sets taken from our four vowels (and having the same prior distribution as from the training set). The error rate is smaller for Bayesian classification: 2.4% vs. 2.2%.

MATLAB function involved:

- `gauss_classify(x,mus,sigmas,priors)` returns the class of the point `x` (1xD) with respect to Gaussian classes, using Bayesian classification. `mus` is a cell array of the (1xD) means, `sigmas` is a cell array of the (DxD) covariance matrices. `priors` is a vector of Gaussian priors. When a set of points (NxD) is provided as input, a set of classes is returned.

```
total=0;
errors_likelihood=0;
errors_bayesian=0;
for i=1:4
    n_test=size(vowels{i}.test,1);
    class_likelihood=gauss_classify(vowels{i}.test,mu,…
        sigma,[1 1 1 1]);
    errors_likelihood=errors_likelihood…
        +sum(class_likelihood'~=i);
    class_bayesian=gauss_classify(vowels{i}.test,mu,…
        sigma,prior);
```

```
        errors_bayesian=errors_bayesian…
            +sum(class_bayesian'~=i);
        total=total+n_test;
    end;
    likelihood_error_rate=errors_likelihood/total
    bayesian_error_rate=errors_bayesian/total
```

```
likelihood_error_rate = 0.0240
bayesian_error_rate   = 0.0220
```

4.2.2 Gaussian Mixture Models (GMM)

In the previous section, we have seen that Bayesian classification is based on the estimation of class PDFs. Up to now, we have modeled the PDF for each class /a/, /e/, /i/, /u/ as a Gaussian multivariate (one per class). This implicitly assumes that the feature vectors in each class have a (uni-modal) normal distribution, as we used the `mean` and `cov` functions, which return the estimates of the mean and covariance matrix of supposedly Gaussian multivariate data samples. It turns out that the vowel data we used had actually been sampled according to Gaussian distributions, so that this hypothesis was satisfied.

Let us now try to classify the words of our imaginary language, using the same kind of approach as above. We will use 100 samples of the six words {"why" /uai/, "you" /iu/, "we" /ui/, "are" /ae/, "hear" /ie/, "here" /ie/}[17] in our imaginary language, for which each speech frame is again characterized by an [F_1, F_2] feature vector. These samples (Fig. 4.8) are stored in the `words` variable.

```
    for i=1:6
        subplot(2,3,i)
        plot(words{i}.training_all(:,1),…
            words{i}.training_all(:,2),'+');
        title(words{i}.word);
    hold on;
    end;
```

Notice that "you" and "we" have the same statistical distribution, because of their phonemic content (in our imaginary language): /iu/ and /ui/. Notice also that "hear" and "here" also have the same distribution, because they have exactly the same phonemic transcription: /ie/. We will come back to this later.

[17] Again, the phonetic transcriptions of these words are not those of English (while they remain easy to remember for tutorial purposes).

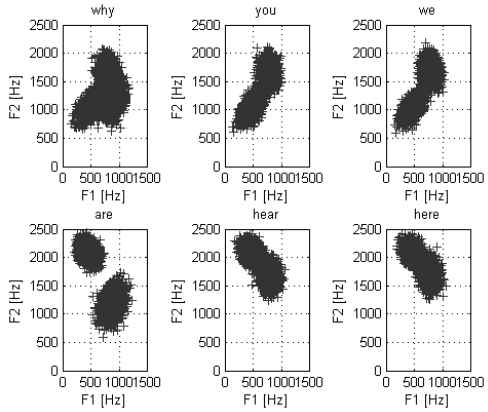

Fig. 4.8 Samples of the six words {"why" /uai/, "you" /iu/, "we" /ui/, "are" /ae/, "hear" /ie/, "here" /ie/} of our imaginary language in the (F_1,F_2) plane.

We are now facing a PDF estimation problem: the PDF of the data in each class in no longer Gaussian. This is typical of practical ASR systems: in word-based ASR, each class accounts for the realization of several phonemes and is thus better described as a multi-modal distribution, i.e. a distribution with several maxima. The same holds for phoneme-based ASR as well. As a matter of fact, speech is very much submitted to coarticulation, which often results in several modes for the acoustic realization of each phoneme, as a function of the phonetic context in which it appears.

If we apply a uni-modal Gaussian model to word "why", for instance, we get a gross estimation of the PDF (Fig. 4.9, left). This estimation does not correctly account for the fact that several areas in the (F_1, F_2) plane are more densely crowded. The maximum value of the Gaussian PDF is very low, since it spans more of the (F_1, F_2) space than it should (and the integral is constrained to one).

```
training_set=words{1}.training_all;
test_set=words{1}.test_all;
mu_all=mean(training_set);
sigma_all=cov(training_set);

plot(training_set(:,1),training_set(:,2),'+');
hold on;
```

```
mesh_gauss2D_pdf(mu_all,sigma_all,...
        1, 0:50:1500, 0:50:2500,7e-9);
```

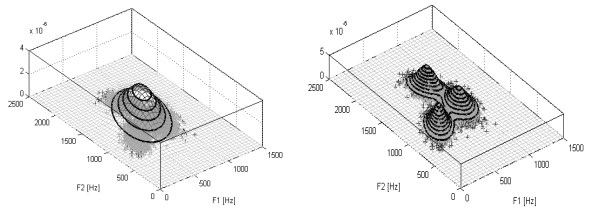

Fig. 4.9 Left: 2D Gaussian estimation of the PDF of the word "why" in the (F_1,F_2) plane. Right: The same PDF estimated from a GMM estimate (using a mixture of 3 Gaussians).

The total log likelihoods of the training and test data given this Gaussian model are obtained as sum of the log likelihoods of all feature vectors.

```
log_likelihood_training=…
   sum(log(gauss_pdf(training_set,mu_all,sigma_all)))
log_likelihood_test=…
   sum(log(gauss_pdf(test_set,mu_all,sigma_all)))
```

```
log_likelihood_training =  -7.4102e+004
log_likelihood_test = -8.3021e+004
```

One way of estimating a multi-modal PDF is by clustering data, and then estimate a uni-modal PDF in each cluster. An efficient way to do this (using a limited number of clusters) is by using *K*-means clustering. Starting with *k* prototype vectors or *centroids,* this algorithm first associates each feature vector in the training set to its closest centroid. It then replaces every centroid by the mean of all feature vectors that have been associated to it. The algorithm iterates by re-associating each feature vector to one of the newly found centroids, and so on until no further change occurs.

MATLAB function involved:

- `[new_means,new_covs,new_priors,distortion]= ... kmeans(data,n_iterations,n_clusters)` , where `data` is the matrix of observations (one observation per row) and `n_clusters` is the desired number of clusters, returns the mean vectors, covariance matrices, and priors of k-means clusters. `distortion` is an array of values (one per iteration) of sum of squared distances between the data and the mean of their cluster. The clusters are initialized with a heuristic that spreads them randomly around `mean(data)`. The algorithm iterates until convergence is reached or the number of iterations exceeds `n_iterations`. Using `kmeans(data,n_iterations,means)`, where `means` is a cell array containing initial mean vectors, makes it possible to initialize means.
- `plot_kmeans2D(data,means)` plots the clusters associated with `means` in `data` samples, using a Euclidian distance.


```
% Initializing prototypes "randomly" around the mean
initial_means{1} = [0,1] * sqrtm(sigma_all) + mu_all;
initial_means{2} = [0,0] * sqrtm(sigma_all) + mu_all;
initial_means{3} = [1,2] * sqrtm(sigma_all) + mu_all;

[k_means,k_covs,k_priors,totalDist]=kmeans(training_set,…
   1000,initial_means);

plot_kmeans2D(training_set, k_means);
```

The *K*-means algorithm converges monotonically, in 14 iterations, to a (local) minimum of the global distortion defined as the sum of all distances between feature vectors and their associated centroids (Fig. 4.10, left).

```
plot(totalDist,'.-');
xlabel('Iteration'); ylabel('Global LS criterion'); grid on;
```

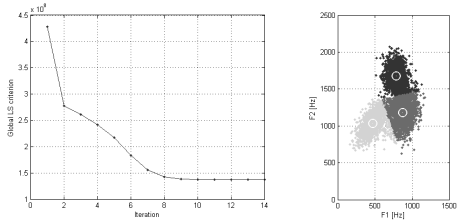

Fig. 4.10 Applying the *k*-means algorithm (with *k*=3) to the sample feature vectors for word "why". Left: Evolution of the total distortion; Right: Final clusters

The resulting sub-classes, though, do not strictly correspond to the phonemes of "why" (Fig. 4.10, right). This is because the global criterion that is minimized by the algorithm is purely geometric. It would actually be very astonishing in these conditions to find the initial vowel sub-classes.

This is not a problem, as what we are trying to do is to estimate the PDF of the data, not to classify it into "meaningful" sub-classes. Once clusters have been created, it is easy to compute the corresponding (supposedly uni-modal) Gaussian means and covariance matrices for each cluster (this is actually done inside our `kmeans` function), and to plot the sum of their PDFs, weighted by their priors. This produces an estimate of the PDF of our speech unit (Fig. 4.9, right).

MATLAB function involved:

- `mesh_GMM2D_pdf(mus,sigmas,weights,gridx,gridy,ratioz)` plots the PDF of a 2D Gaussian Mixture Model PDF in a 3D plot. `mus` is a cell array of the (1x2) means, `sigmas` is a cell array of the (2x2) covariance matrices. `weights` is a vector of Gaussian weights. `gridx` and `gridy` must be vectors of the type (x:y:z) `ratioz` is the (scalar) aspect ratio on the Z axis;

```
plot(training_set(:,1),training_set(:,2),'+');
hold on;
mesh_GMM2D_pdf(k_means,k_covs,k_priors, ...
        0:50:1500, 0:50:2500,2e-8);
hold off;
```

The total log likelihoods of the training and test data are obtained as above, except we now consider that each feature vector "belongs" to each cluster with some weight equal to the prior probability of the cluster. Its likelihood is thus computed as a weighted sum of likelihoods (one per Gaussian).

MATLAB function involved:

- `GMM_pdf(x,mus,sigmas,weights)` returns the likelihood of sample `x` (1xD) with respect to a Gaussian Mixture Model. `mus` is a cell array of the (1xD) means, `sigmas` is a cell array of the (DxD) covariance matrices. (1xD) `weight` is a vector of Gaussian weights. When a set of samples (NxD) is provided as input, a set of likelihoods is returned.

```
log_likelihood_training=…
   sum(log(GMM_pdf(training_set,k_means,k_covs,k_priors)))
log_likelihood_test=…
   sum(log(GMM_pdf(test_set,k_means, k_covs, k_priors)))
```

```
log_likelihood_training =
 -7.1310e+004

log_likelihood_test =
 -7.9917e+004
```


The *K*-means approach used above is not optimal, in the sense that it is based on a purely geometric convergence criterion. The central algorithm for training GMMs is based on the EM (Expectation-Maximization) algorithm. As opposed to *K*-means, EM truly maximizes the likelihood of the data given the GMM parameters (means, covariance matrices, and weights). Starting with *k* initial uni-modal Gaussians (one for each sub-class), it first estimates, for each feature vector, the probability of each sub-class given that vector. This is the Estimation step, which is based on soft classification: each feature vector belongs to all sub-classes, with some weights. In the Maximization step, the mean and covariance of each sub-class is updated, using all feature vectors and taking those weights into account. The algorithm iterates on the E and M steps, until the total likelihood increase for the training data falls under some threshold.

The final estimate obtained by EM, however, only corresponds to a local maximum of the total likelihood of the data, whose value may be very sensitive to the initial uni-modal Gaussian estimates provided as input. A frequently used value for these initial estimates is precisely the one provided by the K-means algorithm.

Applied to the sample feature vectors of "why", the EM algorithm converges monotonically, in 7 steps, from the *K*-means solution to a (local) maximum of the total likelihood of the sample data (Fig. 4.11).

MATLAB function involved:

- `[new_means,new_sigmas,new_priors,total_loglike]= ... GMM_train(data,n_iterations,n_gaussians)`, where `data` is the matrix of observations (one observation per row) and `n_gaussians` is the desired number of clusters, returns the mean vectors, covariance matrices, and priors of GMM Gaussian components. `total_loglike` is an array of values (one per iteration) of the total likelihood of the data given the GMM model. GMMs are initialized with a heuristic that spreads them randomly around `mean(data)`. The algorithm iterates until convergence is reached or the number of iterations exceeds `n_iterations`. `GMM_train(data,n_iterations,means,covs,priors)` makes it possible to initialize the means, covariance matrices, and priors of the GMM components.
- `plot_GMM2D(data, means, covs)` shows the standard deviation ellipsis of the Gaussian components of a GMM defined by `means` and `covs`, on a 2D plot, together with `data` samples.

```
[means,covs,priors,total_loglike]=GMM_train(training_set,…
   100,k_means,k_covs,k_priors);
```

```
plot(training_set(:,1),training_set(:,2),'+');
plot_GMM2D(training_set,means,covs);

plot(total_loglike,'.-');
xlabel('Iteration'); ylabel('Global Log Likelihood'); grid on;
```

The total log likelihoods of the training and test data given this GMM model are obtained as above. The increase compared to estimating the GMM parameters from *K*-means clustering is small, but this is due to the oversimplified PDF we are dealing with. GMMs are very much used in speech recognition for acoustic modeling.

```
log_likelihood_training=…
   sum(log(GMM_pdf(training_set,means,covs,priors)))
log_likelihood_test=…
   sum(log(GMM_pdf(test_set,means,covs,priors)))
```

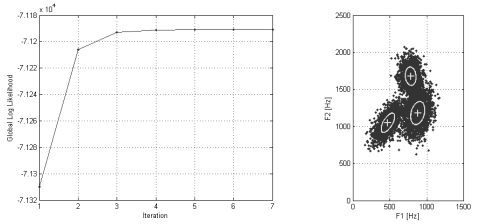

Fig. 4.11 Applying the EM algorithm (with 3 Gaussians) to the sample feature vectors for word "why". Left: Evolution of the total log likelihood of the data; Right: Standard deviation ellipses of the three Gaussian components. Notice we do not set colors to feature vectors, as EM precisely does not strictly assign Gaussians to feature vectors.

```
log_likelihood_training =
 -7.1191e+004

log_likelihood_test =
 -7.9822e+004
```

Now let us try to recognize sample words in {"why", "you", "we", "are", "hear", "here"}. We now use the sequence of feature vectors from our unknown signal (instead of a single vector as before), estimate the joint likelihood of all vectors in this sequence given each class, and obtain the posterior probabilities in the same way as above. If we assume that each sample in our sequence is independent from the others (which is in practice a rather bold claim, even for stationary signals; we will come back to this in the next section when introducing dynamic models), then the joint likelihood of the sequence is simply the product of the likelihoods of each sample.

We first estimate a GMM for each word, using 3 Gaussians per word.[18] The estimated GMMs are plotted in Fig. 4.12.

```
for i=1:6

   [GMMs{i}.means,GMMs{i}.covs,GMMs{i}.priors,total_loglike]=...
         GMM_train(words{i}.training_all,100,3);
end;

for i=1:6
   subplot(2,3,i)
   plot(words{i}.training_all(:,1),…
        words{i}.training_all(:,2),'+');
   title(words{i}.word);
   hold on;
   mesh_GMM2D_pdf(GMMs{i}.means,GMMs{i}.covs,GMMs{i}.priors, ...
            0:50:1500, 0:50:2500,8e-9);
end;
```

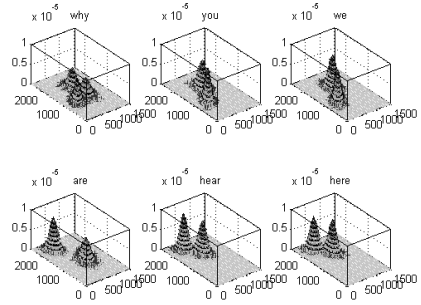

Fig. 4.12 GMMs estimated by the EM algorithm from the sample feature vectors of our six words: "why", "you", "we", "are", "hear", "here"

Let us first try to recognize the first test sequence taken from "why" (Fig. 4.13). Since we do not know the priors of words in our imaginary language, we will set them all to 1/6. As expected, the maximum log like-

[18] When 2 Gaussians are enough, one of the three ends up having very small weight.

lihood is encountered for word "why": our first test word is correctly recognized.

```
word_priors=ones(1,6)*1/6;
test_sequence=words{1}.test{1};
for i=1:6
    log_likelihood(i) = sum(log(GMM_pdf(test_sequence,...
        GMMs{i}.means,GMMs{i}.covs,GMMs{i}.priors)));
end;
log_posterior=log_likelihood+log(word_priors)
[maxlp,index]=max(log_posterior);
recognized=words{index}.word
```

```
log_posterior =
 -617.4004 -682.0656 -691.2229 -765.6281 -902.7732 -883.7884

recognized = why
```

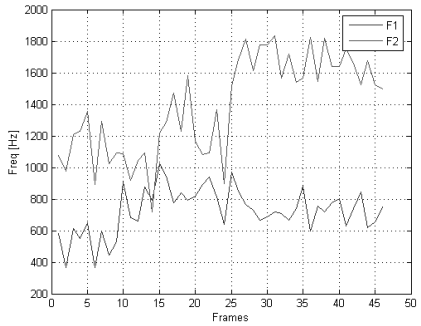

Fig. 4.13 Sequence of feature vectors of the first sample of "why". The three phonemes (each corresponding to a Gaussian in the GMM) are quite apparent.

Not all sequences are correctly classified, though. Sequence 2 is recognized as a "we".

```
test_sequence=words{1}.test{2};
for i=1:6
    log_likelihood(i) = sum(log(GMM_pdf(test_sequence,...
        GMMs{i}.means,GMMs{i}.covs,GMMs{i}.priors)));
end;
log_posterior=log_likelihood+log(word_priors)
[maxlp,index]=max(log_posterior);
recognized=words{index}.word
```

```
log_posterior =
  1.0e+003 *
```



```
   -0.6844   -0.6741   -0.6729   -0.9963   -1.1437   -1.1181

recognized = we
```

We may now compute the total word error rate on our test database.

MATLAB function involved:

- `GMM_classify(x,GMMs,priors)` returns the class of sample `x` with respect to GMM classes, using Bayesian classification. `x` {(NxD)} is a cell array of test sequences. `priors` is a vector of class priors. The function returns a vector of classes.

```
total=0;
errors=0;
for i=1:6
    n_test=length(words{i}.test);
    class=GMM_classify(words{i}.test,GMMs,word_priors);
    errors=errors+sum(class'~=i);
    class_error_rate(i)=sum(class'~=i)/n_test;
    total=total+n_test;

    subplot(2,3,i);
    hist(class,1:6);
    title(words{i}.word);
    set(gca,'xlim',[0 7]);

end;
overall_error_rate=errors/total
class_error_rate
```

```
overall_error_rate =
    0.3000

class_error_rate =
    0.0800    0.4300    0.4400    0.0100    0.3900    0.4500
```

Obviously, our static approach to word classification is not a success. Only 70% of the words are recognized. The rather high error rates we obtain are not astonishing. Except for "why" and "are", which have fairly specific distributions, "here" and "here" have identical PDFs, as well as "you" and "we". These pairs of words are thus frequently mistaken for one another (Fig. 4.14).

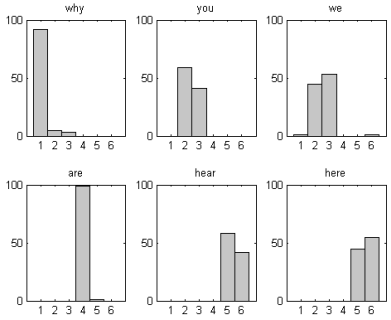

Fig. 4.14 Histograms of the outputs of the GMM-based word recognizer, for samples of each of the six possible input words. The integer values on the *x* axes refer to the index of the output word, in {"why", "you", "we", "are", "hear", "here"}.

4.2.3 Hidden Markov Models (HMM)

In the previous Sections, we have seen how to create a model, either Gaussian or GMM, for estimating the PDF of speech feature vectors, even with complicated distribution shapes, and have applied it to the classification of isolated words. The main drawback of such a static classification, as it stands, is that it does not take time into account. For instance, the posterior probability of a sequence of feature vectors does not change when the sequence is time-reversed, as in words "you" /iu/ and "we" /ui/. This is due to the fact that our Bayesian classifier implicitly assumed that successive feature vectors are statistically independent.

In this Section we will model each word in our imaginary language using a 2-state HMM (plus their initial and final states), except for "why", which will be modeled as a 3-state HMM. One should not conclude that word-based ASR systems set the number of internal HMM states for each word to the number of phonemes they contain. The number of states is usually higher than the number of phonemes, as phonemes are themselves produced in several articulatory steps which may each require a specific state. The reason for our choice is directly dictated by the fact that the test data we are using throughout this script was randomly generated by

HMMs (see appendix 1 in the MATLAB script) in which each phoneme was produced by one HMM state modeled as a multivariate Gaussian. As a result, our test data virtually exhibits no coarticulation, and hence does not require more than one state per phoneme.

We will make one more simplification here: that of having access to a corpus of pre-segmented sentences, from which many examples of our 6 words have been extracted. This will make it possible to train our word HMMs separately. In real ASR systems, segmentation (in words or phonemes) is not known. Sentence HMMs are thus created by concatenating word HMMs, and these sentence HMMs are trained. Words (or phoneme) segmentation is then obtained as a by-product of this training stage.

We start by loading our training data and creating initial values for the left-right HMM of each word in our lexicon. Each state is modeled using a Gaussian multivariate whose mean feature vector is set to a random value close to the mean of all feature vectors in the word. The elements $trans(i,j)$ of the transition matrix give the probability of going from state q_i to q_j (state 1 being the initial state). Transitions probabilities betweeen internal (emiting) states are set to a constant value of 0.8 for staying in the same state, and 0.2 for leaving to the next state.

```
% Initializing HMM parameters
% "why" is a special case: it has 3 states
mu=mean(words{1}.training_all);
sigma=cov(words{1}.training_all);
HMMs{1}.means = {[],mu,mu,mu,[]};
HMMs{1}.covs  = {[],sigma,sigma,sigma,[]};
HMMs{1}.trans = [ 0.0 1.0  0.0  0.0  0.0
                          0.0 0.8  0.2  0.0  0.0
                          0.0 0.0  0.8  0.2  0.0
                          0.0 0.0  0.0  0.8  0.2
                          0.0 0.0  0.0  0.0  1.0 ];
for i=2:6
    mu=mean(words{i}.training_all);
    sigma=cov(words{i}.training_all);
    HMMs{i}.means = {[],mu,mu,[]};
    HMMs{i}.covs  = {[],sigma,sigma,[]};
    HMMs{i}.trans = [0.0 1.0   0.0   0.0
                             0.0 0.8   0.2   0.0
                             0.0 0.0   0.8   0.2
                             0.0 0.0   0.0    1   ];
end
```

Let us train our HMM models using the Baum-Welch (or Forward-Backward) algorithm, which is a particular implementation of the EM algorithm we already used for training our GMMs in the previous Section. This algorithm will adapt the parameters of our word HMMs so as to maximize the likelihood of each training set given each HMM model.

MATLAB function involved:

- `new_hmm = HMM_train_FB(data,old_hmm,dmin,qmax)` returns the Maximum Likelihood re-estimation of a Gaussian Hidden Markov Model (i.e., a single, possibly multivariate, Gaussian probability density function per state) based on the forward-backward algorithm (aka. Baum-Welch re-estimation formulas). Note that most operations are performed in the log domain for accuracy[19]. `dmin` and `qmax` are respectively the minimum log-likelihood relative improvement and the maximum number of iterations until convergence.

```
for i=1:6
  HMMs{i}=HMM_gauss_train(words{i}.training,HMMs{i},0.001,50);
end;
```

The word "why" is now correctly modeled as a sequence of 3 states, each with a Gaussian multivariate PDF, which matches those of the underlying phonemes in the word: /uai/ (**Fig. 4.15**).

```
for i=2:4
    subplot(1,3,i-1)
    plot(words{i}.training_all(:,1),...
         words{i}.training_all(:,2),'+');
    title(['state ' num2str(i-1)]); % emiting states only
    hold on;
    mesh_gauss2D_pdf(HMMs{1}.means{i},HMMs{1}.covs{i},1, ...
         0:50:1500, 0:50:2500,1e-8);
end;
```

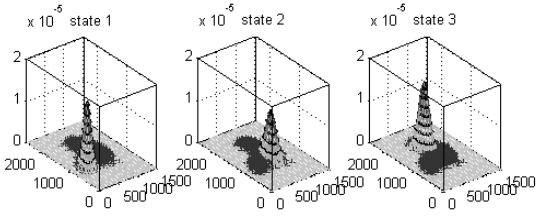

Fig. 4.15 PDF of the three Gaussian HMM states obtained from samples of "why".

[19] This function uses a homemade `logsum.m` function, which computes the log of a sum of likelihoods from log-likelihoods, as mentioned in 4.1.1.

The transition probabilities between the states of "why" have been updated by the Baum-Welch algorithm.

```
HMMs{1}.trans

ans =
         0    1.0000         0         0         0
         0    0.9970    0.0030         0         0
         0         0    0.9951    0.0049         0
         0         0         0    0.9387    0.0613
         0         0         0         0    1.0000
```

As a result of this better modeling, the total likelihood of the data for word "why" is higher than with our previous static GMM model. The previous model can actually be seen as a single-state HMM, whose emission probabilities are modeled by a GMM.

```
log_likelihood_training=0;
for i=1:length(words{1}.training)
    training_sequence=words{1}.training{i};
    log_likelihood_training=log_likelihood_training+...
        HMM_gauss_loglikelihood(training_sequence,HMMs{1});
end;

log_likelihood_test=0;
for i=1:length(words{1}.test)
    test_sequence=words{1}.test{i};
    log_likelihood_test=log_likelihood_test+...
        HMM_gauss_loglikelihood(test_sequence,HMMs{1});
end;

log_likelihood_training
log_likelihood_test
```

```
log_likelihood_training = -6.7144e+004
log_likelihood_test = -7.5204e+004
```

HMM-based isolated word classification can now be achieved by finding the maximum of the posteriori probability of a sequence of feature vectors given all 6 HMM models. The 2nd test sequence for "why" (which was not correctly recognized using GMMs and a single state) now passes our classification test.

```
word_priors=ones(1,6)*1/6;

test_sequence=words{1}.test{2};
for i=1:6
   log_posterior(i) = HMM_gauss_loglikelihood(...
       test_sequence, HMMs{i})+log(word_priors(i));
end
log_posterior
[tmp,index]=max(log_posterior);
recognized=words{index}.word
```

```
log_posterior =
  1.0e+003 *
```

```
   -0.6425   -1.0390   -0.6471   -1.0427   -1.1199   -1.1057
recognized = why
```

The HMM model does not strictly assign states to feature vectors: each feature vector can be emitted by any state with a given probability. It is possible, though, to estimate the best path through the HMM given the data, by using the Viterbi algorithm (Fig. 4.16).

MATLAB function involved:

- `plot_HMM2D_timeseries(x,stateSeq)` plots a two-dimensional sequence `x` (one observation per row) as two separate figures, one per dimension. It superimposes the corresponding state sequence `stateSeq` as colored dots on the observations. `x` and `stateSeq` must have the same length.

```
best_path=HMM_gauss_viterbi(test_sequence,HMMs{index});
plot_HMM2D_timeseries(test_sequence,best_path);
```

We may now compute the total word error rate again.

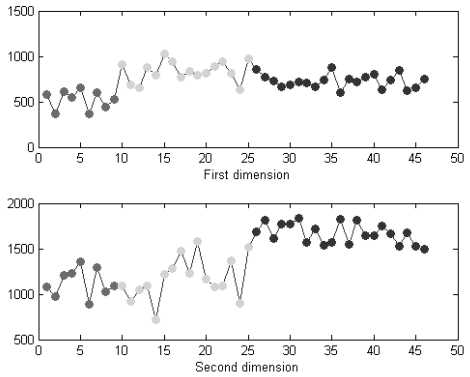

Fig. 4.16 Best path obtained by the Viterbi algorithm, from the sequence of feature vectors of "why" in Fig. 4.13.

MATLAB function involved:

- `HMM_gauss_classify(x,HMMs,priors)` returns the class of sample `x` with respect to HMM classes, using Bayesian classification. HMM states are modeled by a Gaussian multivariate. `x` (NxD) is a cell array of test sequences. `priors` is a vector of class priors. The function returns a vector of classes.

```
total=0;
errors=0;
for i=1:6
    n_test=length(words{i}.test);
    class=HMM_gauss_classify(words{i}.test,HMMs,word_priors);
    errors=errors+sum(class'~=i);
    class_error_rate(i)=sum(class'~=i)/n_test;
    total=total+n_test;

    subplot(2,3,i);
    hist(class,1:6);
    title(words{i}.word);
    set(gca,'xlim',[0 7]);

end;
overall_error_rate=errors/total
class_error_rate
```

```
overall_error_rate = 0.1600
class_error_rate =
         0         0         0         0    0.4400    0.5200
```

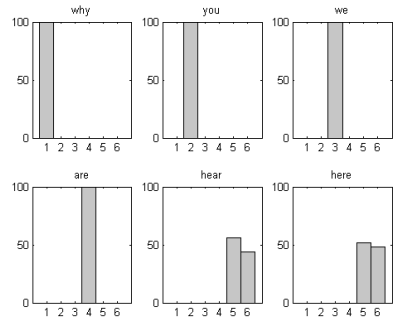

Fig. 4.17 Histograms of the outputs of the HMM-based word classifier, for samples of each of the six possible input words.

Notice the important improvement in the classification of "you" and "we" (Fig. 4.17), which are now modeled as HMMs with distinctive parameters. 84% of the (isolated) words are now recognized. The remaining errors are due to the confusion between "here" and "hear".

4.2.4 *N*-grams

In the previous Section, we have used HMM models for the words of our imaginary language, which led to a great improvement in isolated word classification. It remains that "hear" and "here", having strictly identical PDFs, cannot be adequately distinguished. This kind of ambiguity can only be resolved when words are embedded in a sentence, by using constraints imposed by the language on word sequences, i.e. by modeling the syntax of the language.

We will now examine the more general problem of connected word classification, in which words are embedded in sentences. This task requires adding a language model on top of our isolated word classification system. For convenience, we will assume that our imaginary language imposes the same syntactic constraints as English. A sentence like "you are hear" is therefore impossible and should force the recognition of "you are here" wherever a doubt is possible. In this first step, we will also assume that word segmentation is known (this could easily be achieved, for instance, by asking the speaker to insert silences between words and detecting silences based on energy levels).

Our data file contains a list of 150 such pre-segmented sentences. Let us plot the contents of the first one ("we hear why you are here", Fig. 4.18).

```
for i=1:length(sentences{1}.test)
    subplot(2,3,i);
    test_sequence=sentences{1}.test{i}; % ith word
    plot(test_sequence(:,1),'+-');
    hold on;
    plot(test_sequence(:,2),'r*-');
    title(['Word' num2str(i)]);
end;
```

We model the syntactic constraints of our language by a bigram model, based on the probability of pairs of successive words in the language. Such an approach reduces the language model to a simple Markov model. The component `bigram(i,j)` of its transition matrix gives P($word_i$|$word_j$): the probability that the jth word in the lexicon is followed by the ith word. Clearly, "You are hear" is made impossible by `bigrams(5,6)=0`.

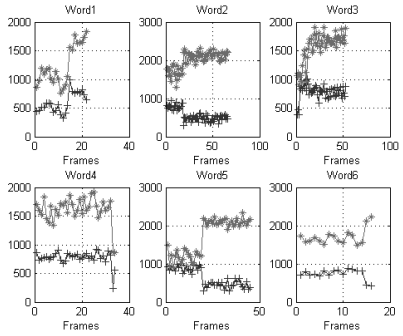

Fig. 4.18 Sequences of feature vectors for the six (pre-segmented) words in the first test sentence.

```
% states = I U {why,you,we,are,hear,here} U F
% where I and F stand for the begining and the end of a
sentence

bigrams = ...
    [0   1/6  1/6  1/6  1/6  1/6  0  ; % P(word|I)
     0   0    1/6  1/6  1/6  1/6  1/6  1/6; % P(word|"why")
     0   1/5  0    0    1/5  1/5  1/5  1/5; % P(word|"you")
     0   0    0    0    1/4  1/4  1/4  1/4; % P(word|"we")
     0   0    1/4  1/4  0    0    1/4  1/4; % P(word|"are")
     0   1/4  1/4  0    0    0    1/4  1/4; % P(word|"hear")
     0   0    1/4  1/4  1/4  0    0    1/4; % P(word|"here")
     0   0    0    0    0    0    0     1]; % P(word|F)
```

Let us now try to classify a sequence of words taken from the test set. We start by computing the log likelihood of each unknown word given the HMM model for each word in the lexicon. Each column of the log likelihood matrix stands for a word in the sequence; each line stands for a word in the lexicon {why,you,we,are,hear,here}.

```
n_words=length(sentences{1}.test);
log_likelihoods=zeros(6,n_words);

for j=1:n_words
    for k=1:6 % for each possible word HMM model
       unknown_word=sentences{1}.test{j};
       log_likelihoods(j,k) = HMM_gauss_loglikelihood(...
          unknown_word,HMMs{k});
    end;
end;

log_likelihoods
```

```
log_likelihoods =

  1.0e+003 *

   -0.2754   -0.3909   -0.2707   -0.4219   -0.6079   -0.5973
   -1.4351   -1.4067   -1.3986   -0.9186   -0.7952   -0.7977
   -0.6511   -0.8062   -0.7147   -0.9689   -0.8049   -0.8024
   -0.5203   -0.4208   -0.5043   -0.6925   -0.5306   -0.5284
   -0.9230   -1.0715   -1.0504   -0.5506   -0.6912   -0.6935
   -0.2510   -0.2772   -0.2400   -0.2851   -0.1952   -0.1953
```

With the approach we used in the previous Section, we would classify this sentence as "we hear why you are hear" (by choosing the max likelihood candidate for each word independently of its neighbors).

```
[tmp,indices]=max(log_likelihoods);
for j=1:n_words
    recognized_sequence{j}=words{indices(j)}.word;
end;
recognized_sequence
```

```
recognized_sequence =
    'we'    'hear'    'why'    'you'    'are'    'hear'
```

We implement our language model as a Markov model on top of our word HMMs. The resulting model for the sequence to recognize is a discrete HMM, in which there are as many internal states as the number of words in the lexicon (six in our case). Each state can emit any of the `n_words` input words (which we will label as '1', '2', ... '`n_words`'), with emission probabilities equal to the likelihoods computed above. Bigrams are used as transition probabilities. Finding the best sequence of words from the lexicon given the sequence of observations [1, 2, ..., `n_words`] is obtained by looking for the best path in this model, using the Viterbi algorithm again.

As shown below, we now correctly classify our test sequence as "we hear why you are here".

MATLAB function involved:

- `[state,likelihood] = HMM_viterbi(transition,emission)` performs the Viterbi search (log version) of the best state sequence for a discrete Hidden Markov Model.

  `transition`: (K+2)x(K+2) matrix of transition probabilities, first and last rows correspond to initial and final (non-emitting) states.
  `emission`: NxK matrix of state-conditional emission probabilities corresponding to a given sequence of observations of length N.
  `state`: (Nx1) vector of state-related indexes of best sequence.
  `likelihood`: best sequence likelihood.


```
best_path=HMM_viterbi(log(bigrams),log_likelihoods);
for j=1:n_words
    recognized_sequence{j}=words{best_path(j)}.word;
end;
recognized_sequence
```

```
recognized_sequence =
    'we'    'hear'    'why'    'you'    'are'    'here'
```

We may finally compute the word error rate on our complete test data.

```
n_sentences=length(sentences);

total=0;
errors=0;
class_error_rate=zeros(1,6);
class=cell(6); % empty cells

for i=1:n_sentences

    n_words=length(sentences{i}.test);
    log_likelihoods=zeros(6,n_words);

    for j=1:n_words
        unknown_word=sentences{i}.test{j};
        for k=1:6 % for each possible word HMM model
           log_likelihoods(j,k) = HMM_gauss_loglikelihood(...
              unknown_word,HMMs{k});
        end;
    end;

    best_path=HMM_viterbi(log(bigrams),log_likelihoods);

    for j=1:n_words
        recognized_word=best_path(j);
        actual_word=sentences{i}.wordindex{j};
        class{actual_word}= [class{actual_word}, …
                             recognized_word];

        if (recognized_word~=actual_word)
            errors=errors+1;
            class_error_rate(actual_word)=…
                class_error_rate(actual_word)+1;
        end;
    end;

    total=total+n_words;

end;

overall_error_rate=errors/total
class_error_rate
```

```
overall_error_rate = 0.1079

class_error_rate =
     0     0     0     0    37    31
```

We now have an efficient connected word classification system for our imaginary language. The final recognition rate is now 89.2%. Errors are still mainly due to "here" being confused with "hear" (Fig. 4.19). As a matter of fact, our bigram model is not constrictive enough. It still allows non admissible sentences, such as in sentence #3: "why are you hear". Bigrams cannot solve all "hear" vs. "here" ambiguities, because of the weaknesses of this poor language model. Trigrams could do a much better job ("are you hear", for instance, will be forbidden by a trigram language model), at the expense of additional complexity.

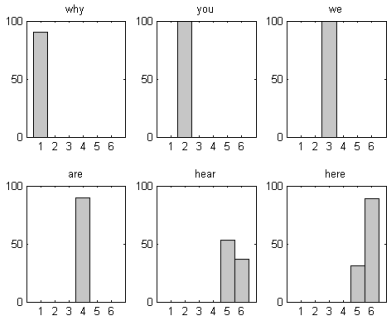

Fig. 4.19 Histograms of the outputs of the HMM-based word classifier, after adding a bigram language model.

4.2.5 Word-based continuous speech recognition

In this Section, we will relax the pre-segmentation constraint, which will turn our classification system into a true word-based speech recognition system (albeit still in our imaginary language).

The discrete sentence HMM we used previously implicitly imposed initial and final states of word HMMs to fall after some specific feature vectors[20]. When word segmentation is not known in advance, the initial and final states of all word HMMs must be erased, for the input feature vector sequence to be properly decoded into a sequence of words.

The resulting sentence HMM is a Gaussian HMM (as each word HMM state is modeled as a Gaussian) composed of all the word HMM states, connected in a left-right topology inside word HMMs, and connected in an ergodic topology between word HMMs. For the six words of our language, this makes 13 internal states, plus the sentence-initial and sentence-final states. The transition probabilities between word-internal states are taken from the previously trained word HMMs, while the transition probabilities between word-final and word-initial states are taken from our bigram model.

```
sentence_HMM.trans=zeros(15,15);

% word-initial states, including  sentence-final state;
word_i=[2 5 7 9 11 13 15];
word_f=[4 6 8 10 12 14]; % word-final states;

% P(word in sentence-initial position)
sentence_HMM.trans(1,word_i)=bigrams(1,2:8);

% copying trans. prob. for the 3 internal states of "why"
sentence_HMM.trans(2:4,2:4)=HMMs{1}.trans(2:4,2:4);

% distributing P(new word|state3,"why") to the first states of
% other word models, weighted by bigram probabilities.
sentence_HMM.trans(4,word_i)=...
    HMMs{1}.trans(4,5)*bigrams(2,2:8);

% same thing for the 2-state words
for i=2:6
   sentence_HMM.trans(word_i(i):word_f(i),word_i(i):word_f(i))=...
       HMMs{i}.trans(2:3,2:3);
   sentence_HMM.trans(word_f(i),word_i)=...
       HMMs{i}.trans(3,4)*bigrams(i+1,2:8);
end;
```

The emission probabilities of our sentence HMM are taken from the word-internal HMM states.

```
k=2;
sentence_HMM.means{1}=[]; % sentence-initial state
for i=1:6
    for j=2:length(HMMs{i}.means)-1
        sentence_HMM.means{k}=HMMs{i}.means{j};
        sentence_HMM.covs{k}=HMMs{i}.covs{j};
        k=k+1;
    end;
end;
sentence_HMM.means{k}=[]; % sentence-final state
```

[20] The sentence HMM therefore had to be changed for each new incoming sentence.

We search for the best path in our sentence HMM[21] given the sequence of feature vectors of our test sequence, with the Viterbi algorithm, and plot the resulting sequence of states (Fig. 4.20).

MATLAB function involved:

- `[states,log_likelihood] = HMM_gauss_viterbi(x,HMM)` returns the best state sequence and the associated log likelihood of the sequence of feature vectors `x` (one observation per row) with respect to a Markov model `HMM` defined by:
  `HMM.means`
  `HMM.covs`
  `HMM.trans`

This function implements the forward recursion to estimate the likelihood on the best path.

```
n_words=length(sentences{1}.test);
complete_sentence=[];
for i=1:n_words
    complete_sentence=[complete_sentence ; ...
        sentences{1}.test{i}];
end;

best_path=HMM_gauss_viterbi(complete_sentence,sentence_HMM);
plot_HMM2D_timeseries(complete_sentence,best_path);

state_sequence=best_path(diff([ 0 best_path])~=0)+1;
word_indices=state_sequence(ismember(state_sequence,word_i));
[tf,index]=ismember(word_indices,word_i);

recognized_sentence={};
for j=1:length(index)
    recognized_sentence{j}=words{index(j)}.word;
end;
recognized_sentence
```

```
recognized_sentence =
    'we'    'hear'    'why'    'you'    'are'    'here'
```

We may finally compute again the word error rate on our complete test data (this is done in the accompanying MATLAB script). The final error rate of our word-based continuous speech recognizer is about 86.8%. This is only 2.4% less than when using pre-segmented words, which shows the efficiency of our sentence HMM model for both segmenting and classifying words. In practice, non segmented data is used for both training and testing, which could still slightly increase the word error rate.

[21] The new sentence model is no longer sentence-dependent: the same HMM can be used to decode any incoming sequence of feature vectors into a sequence of words.

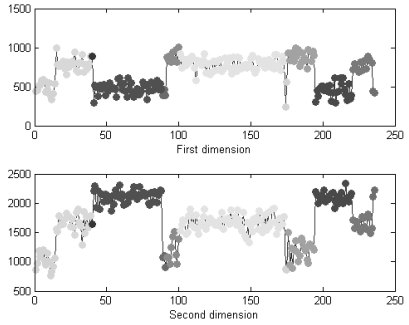

Fig. 4.20 Best path obtained by the Viterbi algorithm, from the sequence of feature vectors of the first test sentence.

4.3 Going further

Dictation machines still differ from this proof-of-concept in several ways. Mel Frequency Cepstral Coefficients (MFCCs) are used in place of our (F_1, F_2) formants for the acoustic model. Their first and second time-derivatives are added as features, as a simple way of accounting for the correlation between feature vectors *within* HMM states. Moreover, given the number of possible words in natural languages (several tens of thousands), ASR systems involve one additional layer in the statistical description of sentences: that of *phonemes*. The word HMMs we have trained above are replaced by phoneme HMMs. Word HMMs are themselves composed of phoneme HMMs (in the same way as we have built our sentence HMM from word HMMs), and additional pruning mechanisms are used in the decoder to constrain the search for the best sequence of words from the input feature vector sequence.

Several MATLAB-based HMM toolboxes are publicly available, such as Kevin Murphy's (Murphy 2005), Steinar Thorvaldsen's (Thorvaldsen 2005, applied to biology), or Olivier Cappé's (Cappé 2001). MATLAB also provides its own HMM toolbox under the Statistics Toolbox. The most famous HMM toolbox, originally developed for large vocabulary speech

recognition, is the HTK toolkit developed at Cambridge University (in ANSI C; Young *et al*. 2006).

4.4 Conclusion

In this Chapter, we have seen how GMMs are used for the classification of (supposedly stationary) signals, and how HMMs provide a means of modeling non-stationary signals as sequences of stationary states. We have also implemented a simple bigram model, whose coupling with our word HMMs has resulted in a unique sentence HMM, able to perform continuous speech recognition, i.e., to find how many words are present in an incoming stream of feature vectors, and which words they are.

4.5 References

Bilmes JA (1998) A gentle tutorial of the EM algorithm and its applications t parameter estimation for Gaussian mixture and hidden Markov models. Technical Report 97-021, ICSI, Berkeley, CA, USA.

Bourlard (2007) Automatic speech and speaker recognition. In: Speech and Language Engineering. M. Rajman, ed., EPFL Press, pp 267-335

Bourlard H, Morgan N (1994) Connectionist Speech Recognition – A Hybrid Approach. Kluwer Academic Publishers, Dordrecht.

Bourlard H, Wellekens C (1990) Links between Markov models and multilayer perceptrons. IEEE Trans on Pattern Analysis and Machine Intelligence, 12(12)

Cappé O (2001) H2M : A set of MATLAB/OCTAVE functions for the EM estimation of mixtures and hidden Markov models [on line] Available: http://www.tsi.enst.fr/~cappe/h2m/h2m.html [3/06/07]

Duda RO, Hart PE, Stork DG (2000) Pattern Classification. Wiley-Interscience

Gold B, Morgan N (2000) Speech and Audio Signal Processing, Processing and Perception of Speech and Music. Wiley, Chichester.

Jelinek F (1991) Up from Trigrams! Proceedings of Eurospeech 91, Genova, vol. 3, pp 1037–1040

Moon TK (1996) The Expectation-Maximization Algorithm. IEEE Signal Processing Magazine, 13 (6), pp 47–60

Murphy K (2005) Hidden Markov Model (HMM) Toolbox for Matlab [on line] Available: http://www.cs.ubc.ca/~murphyk/Software/HMM/hmm.html [20/05/07]

Picone JW (1993) Signal Modeling Techniques in Speech Recognition. Proceedings of the IEEE, 81 (2), pp 1214–1247

Polikar R (2006) Pattern Recognition. In: Wiley Encyclopedia of Biomedical Engineering, Akay M, Ed., New York, NY: Wiley

Rabiner LR (1989) A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. Proceedings of the IEEE, 77 (2), pp 257–286

Thorvaldsen S (2005) A tutorial on Markov models based on Mendel's classical experiments. Journal of Bioinformatics and Computational Biology, Vol. 3, No. 6, 1441-1460. [on line] Available: http://www.math.uit.no/bi/hmm/ [3/06/07]

Young S, Evermann G, Gales M, Hain T, Kershaw D, Liu X, Moore G, Odell J, Ollason D, Povey D, Valtchev V, Woodland P (2006) The HTK book (for HTK Version 3.4) [on line] Available: http://htk.eng.cam.ac.uk/prot-docs/htkbook.pdf [3/06/07]

Chapter 5

How does an audio effects processor perform pitch-shifting?

"Christmas, Christmas time is near, Time for toys and time for cheer, ..."
The chipmunk song (Bagdasarian 1958)

T. Dutoit (°), J. Laroche (*)

(°) Faculté Polytechnique de Mons, Belgium
(*) Creative Labs, Inc., Scotts Valley, California

The old-fashioned charm of modified high-pitched voices has recently been revived by the American movie Alvin and the Chipmunks, based on the popular group and animated series of the same name which dates back from the fifties. These voices were originally performed by recording speech spoken (or sung) slowlier (typically at half the normal speed), and then accelerating ("pitching-up") the playback (typically at double speed, thereby increasing the pitch by one octave). The same effect can now be created digitally and in real time; it is widely used today in computer music.

5.1 Background – The phase vocoder

It has been shown in Chapter 3 (3.1.2 and 3.2.4) that linear transforms and their inverse can be seen as sub-band analysis/synthesis filter banks. The main idea behind the *phase vocoder*, which originally appeared in (Flanagan and Golden 1966 [1]) for speech transmission, is precisely that of using a discrete Fourier transform (DFT) as a filter bank for sub-band processing of signals (5.1.1). This, as we shall see, is a particular case of the more general Short-Time Fourier Transform (STFT) processing (5.1.2) with special care to perfect reconstruction of the input signal (5.1.3), which opens the way to non-linear processing effects, such as time-scale modification (5.1.4), pitch-shifting (5.1.5), harmonization, etc.

5.1.1 DFT-based signal processing

The most straightforward embodiment for frequency-based processing of an audio signal $x(n)$ is given in Fig. 5.1 and Fig. 5.2. Input frames $\mathbf{x}$ containing N samples $[x(mN), x(mN+1), \ldots, x(mN+N-1)]$ are used as input for an N-bin DFT. The N resulting complex frequency-domain values $\mathbf{X}=[X_0(m), X_1(m), \ldots, X_{N-1}(m)]$ are given by:

$$X_k(m) = \sum_{n=0}^{N-1} x(n+mN) e^{-jn\Omega_k} \quad \text{with } \Omega_k = k\frac{2\pi}{N} \tag{5.1}$$

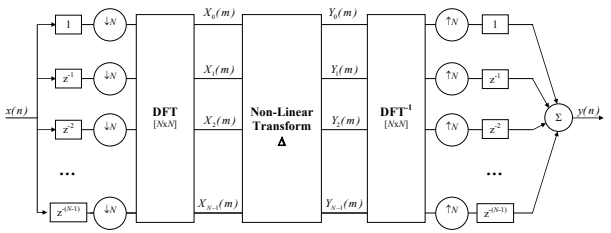

Fig. 5.1 *N*-bin DFT-based signal processing: block diagram

[1] Flanagan and Golden called this system the *phase vocoder* after H. Dudley's *channel vocoder*, which only transmitted the spectral envelope (i.e. a gross estimation of the amplitude spectrum) of speech, while spectral details were sent in a separate channel and modeled though a voiced/unvoiced generator. In contrast, in the phase vocoder, phase information was sent together with amplitude information, and brought a lot more information on the speech excitation.

They are changed into $\mathbf{Y}=[Y_0(m), Y_1(m),..., Y_{N-1}(m)]$ according to the target audio effect $\mathbf{\Delta}$, and an inverse DFT produces output audio samples $\mathbf{y}=[y(mM), y(mM+1), ..., y(mM+N-1)]$.

As already explained in Chapter 3, $X_k(m)$ can also be interpreted, for a fixed value of k, as the result of passing the input signal through a sub-band filter $H_k(z)$ whose frequency response $H_k(\varphi)$ is centered on Ω_k and whose impulse response is given by:

$$h_k(n) = \{1, e^{-j\Omega_k}, e^{-j2\Omega_k}, ..., e^{-j(N-1)\Omega_k}, 0, 0, ...\} \quad (5.2)$$

and further decimating its output by a factor N (Fig. 5.3). These sub-band processing and block-processing views are very complimentary (Fig. 5.4) and will be used often in this Chapter.

Notice that in this first simple embodiment, the input and output frames do not overlap: all sub-band signals $X_k(m)$ ($k=0,1,...N$-1) are implicitly downsampled by a factor N compared to the sampling rate of the input signal $x(n)$.

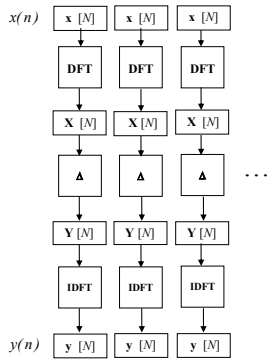

Fig. 5.2 N-bin DFT-based signal processing: frame-based view

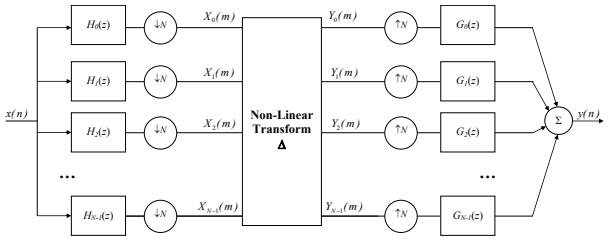

Fig. 5.3 N-channel DFT-based signal processing: sub-band processing view

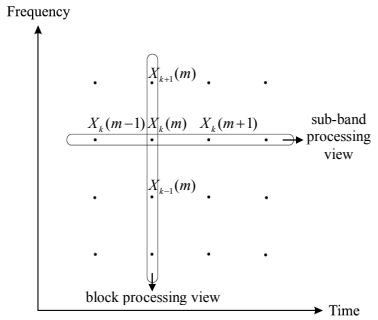

Fig. 5.4 Block processing vs. sub-band signal processing views, in the time-frequency domain (after Dolson, 1986). The former examines the outputs of all sub-band filters at a given instant, while the latter focuses on the output of a single sub-band filter over time.

5.1.2 STFT-based signal processing

The setting depicted in Fig. 5.1 has three apparent problems for the type of non-linear processing we will apply to sub-band signals in this Chapter. These problems are respectively related to frequency selectivity, frame

shift, and to the so-called *blocking effect*. Solving them implies, as we shall see, to use analysis weighting window, overlapping, and synthesis weighting windows.

Analysis weighting windows

DFT filter banks are not frequency selective, as already shown in Fig 3.19, which reveals important side lobes in the frequency response of each filter. This is a serious problem for the design of a phase vocoder: as we will see in Section 5.1.4, the processing applied to each sub-band signal in a phase vocoder is based on the hypothesis that, when the input signal is composed of sinusoidal components (called *partials*, in computer music), each sub-band signal will depend on at most one partial.

A solution is to use an analysis weighting window before the DFT[2]. As a matter of fact, the frequency response of the filter banks implemented by such a weighted DFT is the Fourier Transform of the weighting window itself. Using windows such as Hanning, Hamming, Blackman, or others, makes it possible to decrease the level of side lobes, at the expense of enlarging the bandwidth of each sub-band filter (Fig. 5.5). This drawback can easily be compensated by increasing N, the number of samples in each input frame.

Overlapping analysis frames

In Fig. 5.2, the sampling period of each sub-band signal, i.e. the shift M between successive DFT frames **y**, was explicitly set to N samples, while the normalized bandwidth of each sub-band signal $X_k(m)$ is approximately equal to $2/N$ (the width of the main lobe of the DFT of the weighting window, which is implicitly rectangular in Fig. 5.2). According to Shannon's theorem, we should have:

$$\frac{F_e}{M} \geq B \qquad M \leq \frac{F_e}{B} \tag{5.3}$$

where B/F_e is the normalized bandwidth of the weighting window. To avoid aliasing, sub-band signals should thus be sampled with a sampling period smaller than $N/2$ samples, i.e., input DFT frames should have an overlap greater than $N/2$ (Fig. 5.6).

[2] Notice that weighting windows are not *always* necessary in STFT-based processing. For FFT-based linear time-invariant filtering, for instance, overlapping rectangular windows do the job (Smith 2007).

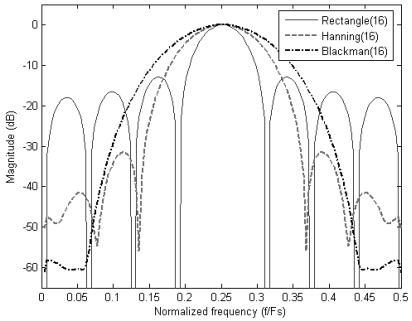

Fig. 5.5 Discrete time Fourier transform of various weighting windows of length N=16 samples, multiplied by a sine wave at 1/4th of the sampling frequency. The main lobe width of the rectangular, Hanning, and Blackman windows is respectively equal to $2/N$, $\approx 4/N$, and $\approx 6/N$; side lobes are respectively 13 dB, 31 dB, and 57 dB lower than the main lobe.

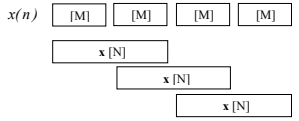

Fig. 5.6 Analysis frames with a frame shift $M=N/2$ (to be compared to the top of Fig. 5.2)

In practice though, we know that if we apply no modification to sub-band signals, i.e. if we set $Y_k(m)= X_k(m)$, the output signal $y(n)$ will be strictly equal to the input signal $x(n)$. This paradox is only apparent: in practice, each sub-band signal is indeed aliased, but the aliasing between adjacent sub-bands cancels itself, as shown in our study of perfect reconstruction filters (3.1.1). This, however, only strictly holds *provided no processing is applied to sub-band signals*.

In contrast, given the complex sub-band signal processing performed in the phase vocoder, it is safer to set the frame shift M in accordance with the Shannon theorem, i.e. to a maximum value of $N/2$ for a rectangular window, $N/4$ for a Hanning window, and $N/6$ for a Blackman window. Besides, we will see in 5.1.4 that the hypotheses underlying sub-band processing in the phase vocoder impose the same constraint.

This transforms our initial DFT (5.1) into the more general *short-time Fourier transform* (STFT):

$$X_k(m) = \sum_{n=0}^{N-1} x(n+mM)w(n)e^{-jn\Omega_k} \qquad (k=0, 1, \ldots, N\text{-}1) \tag{5.4}$$

Synthesis weighting windows

Since the analysis is based on overlapping input frames, synthesis is performed by summing overlapping output frames (as in Fig 3.7). However, when non-linear *block-adaptive* sub-band processing is performed, as in the phase vocoder, processing discontinuities appear at synthesis frame boundaries. This has already been mentioned in 3.1.2.

A partial[3] solution to this problem is to use *synthesis weighting windows* before adding overlapping output frames. The resulting processing scheme is termed as *Weighted OverLap-Add* (WOLA):

$$y(n) = \sum_{m=-\infty}^{\infty} y_m(n-mM)w(n-mM)$$
$$\text{with } y_m(n) = \frac{1}{N}\sum_{k=0}^{N-1} Y_k(m)e^{jn\Omega_k} \tag{5.5}$$

where in practice the number of terms in the first summation is limited by the finite length N of the weighting window $w(n)$. The association of IFFT and WOLA is sometimes referred to as *Inverse short-time Fourier transform* (ISTFT).

Fig. 5.7 shows the final block-processing view for STFT-based signal processing: an STFT-based analysis block, whose output (amplitudes and phases) is processed and sent to an ISTFT block. The corresponding sub-band processing view is not repeated here: it is still that of Fig. 5.3 provided the down- and upsampling ratios are set to M rather than N, and in which filters $H(z)$ and $G(z)$ now also account for the effect of the non-rectangular weighting window.

[3] The use of synthesis weighting windows only spreads discontinuities around the overlap area of successive frames, it does not cancel them. As a result, special care has to be taken to avoid introducing avoidable discontinuities in modified STFTs (see 5.1.4).

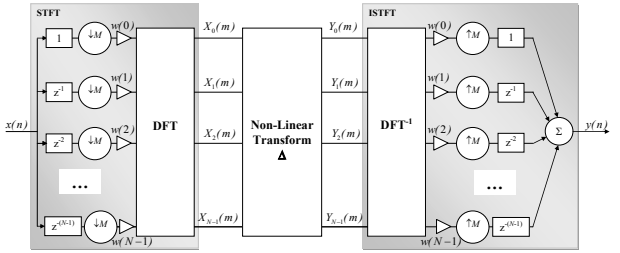

Fig. 5.7 STFT-based signal processing: block processing view

5.1.3 Perfect reconstruction

The results obtained with an STFT-based signal processing system obviously depend on the weighting window used, and on the shift between successive analysis windows.

Assuming the analysis and synthesis windows are identical, and making the further assumption that DFTs are not modified, WOLA's output is given by:

$$y(n) = \sum_{m=-\infty}^{\infty} x(n)w^2(n-mM) \tag{5.6}$$

Perfect reconstruction is therefore achieved when[4]:

$$\sum_{m=-\infty}^{\infty} w^2(n-mM) = 1 \quad \forall n \in \mathbb{Z} \tag{5.7}$$

[4] Notice the similarity between this condition and (3.18), in which only *m=0 and 1* are taken into account, since in the Modulated Lapped Transform the frame shift M is set to half the frame length N.

This condition is sometimes called the *Constant OverLap Add* (COLA) constraint, applied here on $w^2(n)$. It depends on the shape of the weighting window $w(n)$ and on the ratio of the frame shift M to the frame length N.

A simple example is that of the square root of the triangular (or *Bartlett*) window with $M=N/2$. The Hanning (or *Hann*) window with $M=N/4$ is usually preferred (Fig. 5.8), as its Fourier transform exhibits a better compromise between the level of its side lobes and the width of its pass-band. The square root of the Hanning window can also be used with $M=N/2$ [5].

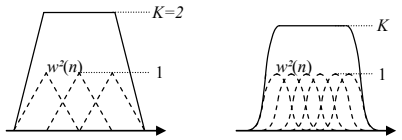

Fig. 5.8 Overlap-adding triangular windows with $M=N/2$ (left) and squared Hanning windows $M=N/4$ (right)

Usual weighting windows, when used in the COLA conditions, sum up to $K\neq 1$ (Fig. 5.8). Using the Poisson Summation Formula, it is easy to show that the value of K is given by (Smith 2007b):

$$K = \frac{\sum_{0}^{N-1} w^2(n)}{M} \tag{5.8}$$

It is then easy to multiply the weighting window by 1/sqrt(K) to enforce (5.7).

5.1.4 Time scale modification with the phase vocoder

The phase vocoder is a specific implementation of an STFT-based processing system in which:

1. The input signal is assumed to be composed of a sum of (not necessarily harmonic) sinusoïdal terms, named *partials*:

[5] The square root of the Hanning window is nothing else than the sinusoidal window of the Modulated Lapped Transform (MLT) used in perceptual audio coders (3.1.1), with a frame shift set to half the frame-length.

$$x(n) = \sum_{i=1}^{P} A_i \cos(n\varphi_i + \phi_i) \tag{5.9}$$

where P is the number of partials, A_i and ϕ_i are their amplitudes and initial phases, and φ_i are their normalized angular frequencies ($\varphi_i=\omega_i/F_s$ [rad/sample]). The value of $n\varphi_i+\phi_i$ gives the *instantaneous phase* of the ith partial at sample n.

2. The output of each DFT bin is assumed to be mainly influenced by a single partial. As already mentioned in 5.1.1, satisfying this condition implies to choose long input frames, since the width of the main spectral lobe of the analysis window depends on the number N of samples in the frame.

When these assumptions are verified[6], it is easy to show that the output of each DFT channel $X_k(m)$ (k=0,..,N-1) is a complex exponential function over m. For simplicity, let us assume that the input signal is a single imaginary exponential:

$$x(n) = Ae^{jn\varphi+\phi} \tag{5.10}$$

Then $X_k(m)$ is given by:

$$\begin{aligned} X_k(m) &= \sum_{n=0}^{N-1} Ae^{j(n+mM)\varphi+\phi} w(n) e^{-jn\Omega_k} \\ &= \left(A\sum_{n=0}^{N-1} e^{jn(\varphi-\Omega_k)} w(n) \right) e^{jmM\varphi+\phi} \qquad (k=0,\ 1,\ \ldots,\ N\text{-}1) \\ &= H_k(\varphi) e^{jmM\varphi+\phi} \end{aligned} \tag{5.11}$$

where $H_k(\varphi)$ is the frequency response of the kth analysis sub-band filter for frequency φ and therefore does not depend on m. The output of each DFT channel is thus an imaginary exponential function whose angular frequency $M\varphi$ only depends on the frequency φ of the underlying partial (and *not* on the central frequency Ω_k of the DFT bin). As a result, it is predictable from any of its past values:

$$|X_k(m+1)| = |X_k(m)| \tag{5.12}$$

[6] Clearly, these conditions are antagonistic. Since the partials of an audio signal are generally modulated in amplitude and frequency, condition 1 is only valid for short frames, while verifying condition 2 requires long input frames.

$$\text{and } \angle X_k(m+1) = \angle X_k(m) + M\varphi$$

where $\angle X_k(m)$ is the *instantaneous phase* of the output of the *k*th DFT channel (Fig. 5.10). Notice that only a *wrapped* version of this phase is available in practice, since it is computed as an arctangent and therefore obtained in the range [-π,π].

Time-scale modification, with horizontal phase locking

Modifying the duration of the input signal can then be achieved by setting an analysis frame shift M_a in (5.5) lower (time expansion) or higher (time compression) than the synthesis frame shift M_s [7], and adapting the DFT channel values $Y_k(m)$ accordingly:

$$\begin{aligned} |Y_k(m)| &= |X_k(m)| \\ \text{and } \angle Y_k(m+1) &= \angle Y_k(m) + M_s\varphi_k(m) \end{aligned} \tag{5.13}$$

in which $\varphi_k(m)$ is the frequency of the partial which mainly influences $X_k(m)$.

The initial synthesis phases $\angle Y_k(0)$ are simply set to the corresponding analysis phases $\angle X_k(0)$, and the subsequent values are obtained from (5.13). This operation is termed as *horizontal phase locking* (Fig. 5.9), as it ensures phase coherence across frames in each DFT channel independently of its neighbors. In other words, it guarantees that the short-time synthesis frames $y_m(n)$ in (5.5) will overlap coherently[8].

Obtaining $\varphi_k(m)$ from (5.12), though, is not straightforward. Since the values of $\angle X_k(m)$ and $\angle X_k(m+1)$ are only known modulo 2π, we have:

$$\angle X_k(m+1) - \angle X_k(m) = M\varphi_k(m) + l2\pi \tag{5.14}$$

where l is an a priori unknown integer constant. Let us rewrite (5.14) as:

$$\angle X_k(m+1) - \angle X_k(m) = M(\theta_k(m) + \Omega_k) + l2\pi \tag{5.15}$$

where $\theta_k(m)$ is the frequency offset between $\varphi_k(m)$ and Ω_k (Fig. 5.10).

[7] The synthesis frame shift M_s is left constant, rather than the analysis frame shift M_a, so as to take the COLA constraint into account.

[8] Remember that this is only valid, though, for constant frequency partials.

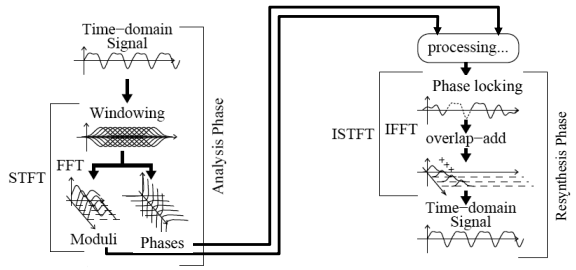

Fig. 5.9 The phase vocoder (after Arfib *et al.* 2002)

If we assume that $\theta_k(m)$ is small enough for $|M\theta_k(m)|$ to be less than π, and if $[]_{2\pi}$ denotes the reduction of the phase to its principal value in [-π,π], then:

$$\begin{aligned} \left[M\theta_k(m)\right]_{2\pi} &= \left[\angle X_k(m+1) - \angle X_k(m) - M\Omega_k - l2\pi\right]_{2\pi} \\ M\theta_k(m) &= \left[\angle X_k(m+1) - \angle X_k(m) - M\Omega_k\right]_{2\pi} \end{aligned} \tag{5.16}$$

which gives the expected value of $\theta_k(m)$.

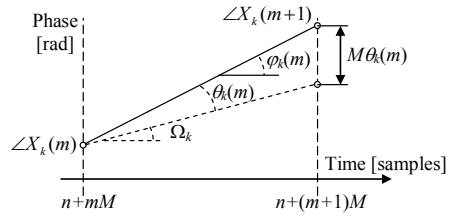

Fig. 5.10 Theoretical phase increment from frame *m* to frame *m*+1 in DFT channel *k* due to a main partial with frequency $\varphi_k(m)$ in its band-pass. In practice, $X_k(m)$ and $X_k(m+1)$ are only known modulo 2π (this is not shown here).

The assumption on the small value of $\theta_k(m)$ can be translated into an offset b in DFT bins whose central frequency is close enough to $\varphi_k(m)$ for (5.16) to be applicable:

$$\begin{aligned} &|M\left(b\frac{2\pi}{N}\right)|<\pi \\ &|b|<\frac{N}{2M} \end{aligned} \tag{5.17}$$

Since the half main spectral lobe width of a Hanning window covers approximately 2 bins, we conclude that, if a Hanning-based vocoder is designed in such a way that horizontal phase locking should be applied to the whole main lobe around each partial, then the frame shift M must be such that $M<N/4$. This condition happens to be identical as the one derived in 5.1.2.

Time-scale modification, with horizontal and vertical phase locking

The two most prominent time-scaling artifacts of the classical phase vocoder expose above are *transient smearing* and *phasiness*. Both occur even with modification factors close to one. Transient smearing is heard as a slight loss of percussiveness in the signal. Piano signals, for instance, may be perceived as having less "bite". Phasiness is heard as a characteristic coloration of the signal; in particular, time-expanded speech often sounds as if the speaker is much further from the microphone than in the original recording. Phasiness can be minimized by not only ensuring phase consistency *within* each DFT channel over time, but also as we shall see by ensuring phase coherency across the channels in a given synthesis frame. A system implementing both horizontal and vertical phase locking is called a *phase-locked vocoder*.

Coming back to the simple case of a single imaginary exponential, and further assuming that the weighting window is rectangular, $H_k(\varphi)$ in (5.11) becomes:

$$\begin{aligned} H_k(\varphi) &= \sum_{n=0}^{N-1} e^{jn(\varphi-\Omega_k)} = \frac{1-e^{jN(\varphi-\Omega_k)}}{1-e^{j(\varphi-\Omega_k)}} \\ &= e^{j\frac{N-1}{2}(\varphi-\Omega_k)}\left(\frac{\sin\frac{N(\varphi-\Omega_k)}{2}}{\sin\frac{(\varphi-\Omega_k)}{2}}\right) \quad (k=0,\ 1,\ \ldots,\ N\text{-}1) \end{aligned} \tag{5.18}$$

Since the bracketed expression in (5.18) is positive in its main lobe, the following cross-channel phase relationship exists in the main lobe:

$$\begin{aligned} H_{k+1}(\varphi) - H_k(\varphi) &= e^{j\frac{N-1}{2}(\Omega_k-\Omega_{k+1})} \\ &= e^{-j\frac{N-1}{N}\pi} \end{aligned} \tag{5.19}$$

This relationship is even simplified if the weighted samples are first circularly shifted by $(N\text{-}1)/2$ samples[9] before the DFT, as this introduces an additional linear phase factor which cancels the influence of Ω_k in $H_k(\varphi)$. As a result, (5.19) becomes:

$$H_{k+1}(\varphi) - H_k(\varphi) = 0 \tag{5.20}$$

This result can easily be generalized to any weighting window.

Puckette (1995) proposed a method called *loose phase locking,* in which the phase in each synthesis channel is implicitly influenced by the phases of the surrounding synthesis channels. The method first computes the synthesis phases in all channels $\angle Y_k(m+1)$ according to (5.13), and then modifies the results as:

$$\angle Y'_k(m+1) = \angle\left(Y_k(m-1) + Y_k(m) + Y_k(m+1)\right) \quad (k=0,\ 1,\ \ldots,\ N\text{-}1) \tag{5.21}$$

As a result, if a DFT channel is the maximum of the DFT magnitudes, its phase is basically unchanged, since $Y_k(m-1)$ and $Y_k(m+1)$ have a much lower amplitude. Conversely, the phase of a channel surrounding the maximum will roughly be set to that of the maximum: the channels around the peaks of the DFT are roughly *phase locked.*

The fundamental limitation (and also the attraction) of the loose phase locking scheme is that it avoids any explicit determination of the signal structure: the same calculation is performed in every channel, independently of its content. Laroche and Dolson (1999) have proposed an improved phase locking scheme, named *rigid phase locking* and based on the explicit identification of peaks in the spectrum. Their new phase-updating technique starts with a coarse peak-picking stage where vocoder channels are searched for local maxima. In the simplest implementation, a channel whose amplitude is larger than its four nearest neighbors is said to be a peak. The series of peaks divides the frequency axis into *regions of in-*

[9] In practice, since FFTs are used for implementing the phase vocoder, N is even, and samples are shifted by $N/2$. Consequently, (5.20) is only approximately verified.

flence located around each peak. The basic idea is then to update the phases for the peak channels only, according to (5.13): the phases of the remaining channels within each region are then locked in some way to the phase of the peak channel. In particular, in their so-called *identity phase locking* scheme, Laroche and Dolson constrain the synthesis phases around each peak to be related in the same way as the analyses phases. If p is the index of dominant peak channel, the phases of all channels k ($k \neq p$) in the peak's region of influence are set to:

$$\angle Y_k(m) = \angle Y_p(m) + \angle X_k(m) - \angle X_p(m) \qquad (k=0, 1, \ldots, N\text{-}1) \qquad (5.22)$$

Another improvement is also proposed by the authors in the *scaled phase locking* scheme, which accounts for the fact that peak channels may evolve with time.

5.1.5 Pitch shifting with the phase vocoder

The resampling-based approach

The standard pitch-scale modification technique combines time-scale modification and resampling. Assuming a pitch-scale modification by a factor β is desired (i.e., all frequencies must be multiplied by β), the first stage consists of using the phase-vocoder to perform a factor time-scale modification of the signal (its duration is multiplied by β). In the second stage, the resulting signal is resampled at a new sampling frequency F_s/β, where F_s is the original sampling period. The output signal ends up with the same duration as the original signal, but its frequency content has been expanded by a factor β during the resampling stage, which is the desired result (Fig. 5.11).

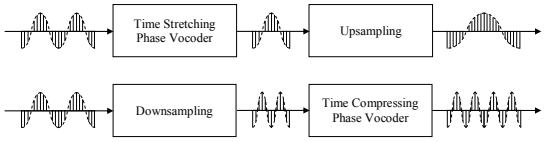

Fig. 5.11 A possible implementation of a pitch-shifter using a phase vocoder. Top: downward pitch-scaling ($\beta < 1$); Bottom: upward pitch scaling.

Note that it is possible to reverse the order of these two stages, which yields the same result if the window size is multiplied by β in the phase-

vocoder time-scaling stage. However, the cost of the algorithm is a function of the modification factor and of the order in which the two stages are performed. For example, for upward pitch-shifting ($\beta > 1$), it is more advantageous to resample first and then time-scale, because the resampling stage yields a shorter signal. For downward pitch-shifting, it is better to time-scale first and then resample, because the time-scaling stage yields a signal of smaller duration (as shown on Fig. 5.11).

This standard technique has several drawbacks. Its computational cost is a function of the modification factor. If the order in which the two stages are performed is fixed, the cost becomes increasingly large for larger upward or downward modifications. An algorithm with a fixed cost is usually preferable. Another drawback of the standard technique is that only *one* "linear" pitch-scale modification is allowed i.e., the frequencies of all the components are multiplied by the same factor . As a result, harmonizing a signal (i.e., adding several copies pitch-shifted with different factors) requires repeated processing at a prohibitive cost for real-time applications.

The STFT-based approach

A more flexible algorithm has been proposed by Laroche and Dolson (1999b), which allows non-linear frequency modifications, enabling the same kind of alterations that usually require non real-time sinusoidal analysis/synthesis techniques.

The underlying idea behind the new techniques consists of identifying peaks in the short-term Fourier transform, and then translating them to new arbitrary frequencies. If the relative amplitudes and phases of the bins around a sinusoidal peak are preserved during the translation, then the time-domain signal corresponding to the shifted peak is simply a sinusoid at a different frequency, modulated by the same analysis window.

For instance, in the case of a single imaginary exponential signal $x(n) = Ae^{jn\varphi+\phi}$, input frames will be of the form:

$$x_m(n) = A\, w(n+mM) e^{j(n+mM)\varphi+\phi} \qquad (5.23)$$

and (5.11) shows that $X_k(m)$ only depends on k via (φ - Ω_k). As a result, imposing[10]:

$$Y_k(m) = X^{(*)}_{(k+S) \bmod N}(m) \quad (k = 0,1,\ldots,N-1) \qquad (5.24)$$

[10] Notice the "mod N" in (5.24), which makes sure that, if a shifted region of influence spills out of the frequency range [0,2π], it is simply reflected back into [0,2π] after complex conjugation (indicated by our non standard use of the "(*)" notation) to account for the fact that the original signal is real.

in which S is an integer number of DFT bins, will result in output frames of the form:

$$y_m(n) = A\, w(n+mM) e^{j(n+mM)(\varphi+\Delta\varphi)+\phi} \tag{5.25}$$

with $\Delta\varphi = S\,\frac{2\pi}{N}$. These output frames, however, will not overlap-add coherently into:

$$y(n) = A e^{jn(\varphi+S\frac{2\pi}{N})+\phi} \tag{5.26}$$

since (5.24) does not ensure that the peak phases are consistent from one frame to the next. As a matter of fact, since the frequency has been changed from φ to $\varphi+\Delta\varphi$, phase coherency requires rotating all phases by $M\Delta\varphi$ and accumulating this rotation for each new frame, which results in the final pitch-scaling scheme:

$$Y_k(m) = X_{(k+S) \bmod N}(m)\, e^{jmM\Delta\varphi} \quad (k = 0,1,...,N-1) \tag{5.27}$$

Pitch-shifting a more complex signal is achieved by first detecting spectral peaks and regions of influence, as in the phase-locked vocoder, and then shifting each region of influence separately by β, using (5.27) with $\Delta\varphi=(\beta\text{-}1)\varphi$. Shifted areas of influence that overlap in frequency are simply added together.

When the value of the frequency shift $\Delta\varphi$ does not correspond to an integer number of bins S (which is the most general case), it can either be rounded to the closest integer value of S (which is very acceptable for large DFT sizes and low sampling rates, for which DFT channel are very narrow), or linear interpolation can be used to distribute each shifted bin into existing DFT bins (see Laroche and Dolson 1999b).

It is useful to note that because the channels around a given peak are rotated by the same angle, the differences between the phases of the channels around a peak in the input STFT are preserved in the output STFT short-term Fourier transform. This is similar to the "Identity Phase-Locking" scheme, which dramatically minimizes the phasiness artifact often encountered in phase-vocoder time or pitch-scale modifications.

Notice also, finally, that the exact value of the frequency φ of each partial is not required, since only $\Delta\varphi$ is used in (5.27). This is an important savings compared to the standard phase vocoder, and makes this approach more robust.

5.2 MATLAB proof of concept: ASP_audio_effects.m

In this section, we show that STFT-based signal processing provides very flexible tools for audio signal processing. We start by radically modifying the phases of the STFT, to create a robotization effect (5.2.1). Then we examine the MATLAB implementation of a phase vocoder, with and without vertical phase locking (5.2.2). We conclude the Chapter with two pitch-scale modification algorithms, which use the phase vocoder in various aspects (5.2.3).

5.2.1 STFT-based audio signal processing

We first implement a basic STFT-based processing scheme, using the same weighting window for both analysis and synthesis, and with very simple processing of the intermediate DFTs.

Weighting windows

Since weighting windows play an important part in STFT-based signal processing, it is interesting to check their features first. MATLAB proposes a handy tool for that: the `wvtool` function. Let us use it for comparing the rectangular, Hanning, and Blackman windows. Clearly, the spectral leakage of the Hanning and Blackman windows are lower than those of the rectangular and sqrt(Hanning) windows. By zooming on the the main spectral lobes, one can check that this is compensated by a higher lobe width (Fig. 5.12): the spectral width of the rectangular window is half that of the Hanning window, and a third of that of the Blackman window.

```
N=100;
wvtool(boxcar(N),hanning(N),blackman(N));
```

The Constant OverLap-Add (COLA) constraint

Let us now examine the operations involved in weighted overlap-add (WOLA). When no modification of the analysis frames is performed, one can to the least expect the output signal to be very close to the input signal. In particular, a constant input signal should result in a constant output signal. This condition is termed as Constant OverLap-Add (COLA) constraint. It is only strictly met for specific choices of the weighting window and of the window shift, as we shall see. We now check this on a chirp signal.

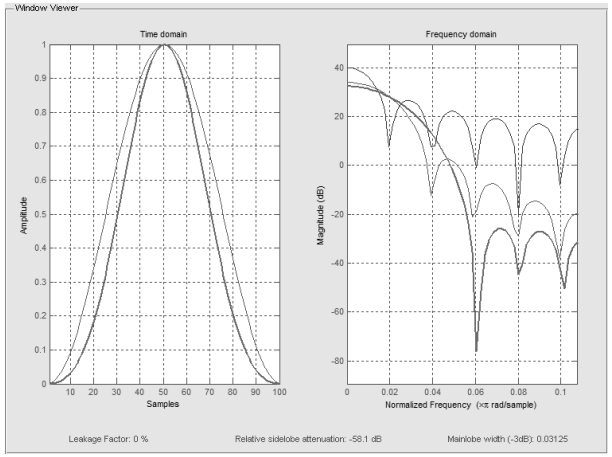

Fig. 5.12 The 100-samples rectangular, Hanning, and Blackman weighting windows (left) and their main spectral properties (right)

```
Fs=8000;
input_signal=chirp((0:Fs)/Fs,0,1,2000)';
specgram(input_signal,1024,Fs,256);
```

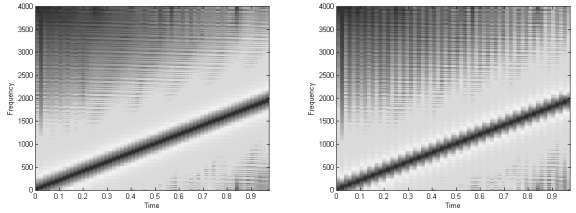

Fig. 5.13 Spectrogram of a chirp. Left: original signal; Right: after weighted overlap-add using a Hanning window and a frame shift set to half the frame length

We actually use the periodic version of the Hanning window, which better meets the COLA constraint for various values of the frame shift than the default symmetric version. It is easy to plot the SNR of the analysis-synthesis process, for a variable frame shift between 1 and 512 samples and a frame length N of 512 samples (Fig. 5.14, left). The resulting SNR is indeed very high for values of the frame shift M equal to $N/4$, $N/8$, $N/16$, ..., 1. These values are said to meet the COLA constraint for the squared Hanning window. In practice, values of the frame shift that do not meet the COLA constraint can still be used, provided the related SNR is higher than the local signal-to-mask ratio (SMR) due to psycho-acoustic effects (see Chapter 3). Using the highest possible value for the frame shift while still reaching a very high SNR minimizes the computational load of the system.

```
COLA_check=zeros(length(input_signal),2);

frame_length=512;

for frame_shift=1:frame_length

    % Amplitude normalization for imposing unity COLA
    window=hanning (frame_length,'periodic');
    COLA_ratio=sum(window.*window)/frame_shift;
    window=window/sqrt(COLA_ratio);

    output_signal=zeros(length(input_signal),1);
    pin=0;     % current position in the input signal
    pout=0;   % current position in the output signal

    while pin+frame_length<length(input_signal)

        % Creating analysis frames
        analysis_frame = ...
           input_signal(pin+1:pin+frame_length).* window;

        % Leaving analysis frames untouched
        synthesis_frame=analysis_frame;

        % Weighted OverLap-Add (WOLA)
        output_signal(pout+1:pout+frame_length) = ...
            output_signal(pout+1:pout+frame_length)+...
             synthesis_frame.*window;

        % Checking COLA for two values of the frame shift
        if (frame_shift==frame_length/2)
            COLA_check(pout+1:pout+frame_length,1)=...
               COLA_check(pout+1:pout+frame_length,1)...
                 +window.*window;
        elseif (frame_shift==frame_length/4)
            COLA_check(pout+1:pout+frame_length,2)=...
                COLA_check(pout+1:pout+frame_length,2)...
                  +window.*window;
        end;

        pin=pin+frame_shift;
        pout=pout+frame_shift;

    end;
```

```
    % Storing the output signal for frame shift = frame_length/2
    if (frame_shift==frame_length/2)
        output_half=output_signal;
    end;

    % Using the homemade snr function introduced in Chapter 2,
    % and dropping the first and last frames, which are only
    % partially overlap-added.
    snr_values(frame_shift)=snr(...
        input_signal(frame_length+1:end-frame_length),...
        output_signal(frame_length+1:end-frame_length),0);

end

plot((1:frame_length)/frame_length,snr_values);
```

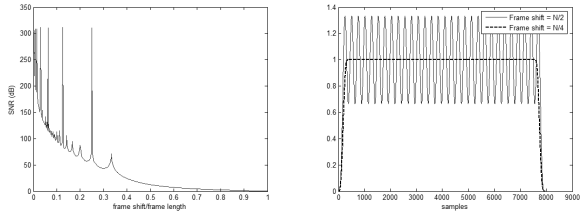

Fig. 5.14 Examining the COLA constraint for a Hanning window. Left: SNR for various values of the ratio frame shift/frame length; Right: result of the OLA with M=N/2 (non COLA) and N/4 (COLA)

When the input signal is a unity constant, the output signal can indeed be close to unity... or not, depending the value of the frame shift. For the Hanning window, obviously, a frame shift of N/4 meets the COLA constraint, while N/2 leads to an important amplitude ripple (Fig. 5.14, right). These values also depend on the weighting window. [11]

```
plot(COLA_check(:,1)); hold on;
plot(COLA_check(:,2),'k--','linewidth',2); hold off;
```

A bad choice of the frame shift may lead to important perceptual degradation of the output audio signal, as confirmed by listening to the output produced for M=N/2 (see Fig. 5.13, right for the related spectrogram).

```
soundsc(output_half, Fs);
```

[11] See Appendix 1 in the **audio_effects.m** file for a test on the square-root Hanning window.


```
specgram(output_half,1024,Fs,256);
```

STFT-based signal processing

We now add an STFT/ISTFT step. The choice of the frame length N, of the type of weighting window and of the frame shift, depends on the type of processing applied to frequency bands. For frequency-selective processing, a large value of N is required, as the bandwidth of the DFT channels is inversely proportional to N. In this Section, we will implement a simple robotization effect on a speech signal, which is not a frequency-selective modification. We therefore set N to 256 samples (but many other values would match our goal). We choose a Hanning window, with a frame shift of 80 samples. This value does not meet the COLA constraint for the Hanning window, but the effect we will apply here does not attempt to maintain the integrity of the input signal anyway.

Let us process the **speech.wav** file, which contains the sentence *Paint the circuits* sampled at 8 kHz. We first plot its spectrogram using the same frame length and frame shift as that of our STFT. The resulting plot is not especially pretty, but it shows exactly the data that we will process. The spectrogram reveals the harmonic structure of the signal, and shows that its pitch is a function of time (Fig. 5.15, left).

```
frame_length=256;
frame_shift=80;
window=hanning (frame_length,'periodic');
COLA_ratio=sum(window.*window)/frame_shift;
window=window/sqrt(COLA_ratio);

[input_signal,Fs]=wavread('speech.wav');
specgram(input_signal,frame_length,Fs,window);
```

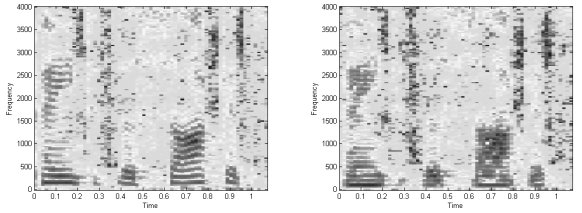

Fig. 5.15 Spectrogram of the sentence *Paint the circuits*. Left: original signal, as produced by the STFT; Right: robotized signal

When no processing of the intermediate DFTs is performed, the output signal is thus equal to the input signal. It is easy to modify the amplitudes or the phases of the STFT, to produce audio effects. Let us test a simple robotization effect, for instance, by setting all phases to zero. This produces a disruptive perceptual effect, at the periodicity of the frame shift (Fig. 5.16). When the shift is chosen small enough, and the effect is applied to speech, it is perceived as artificial constant pitch. This simple effect appears on the spectrogram as a horizontal reshaping of the harmonics (Fig. 5.15, right). It reveals the importance of phases modifications (and more specifically of phase discontinuities) in the perception of a sound.

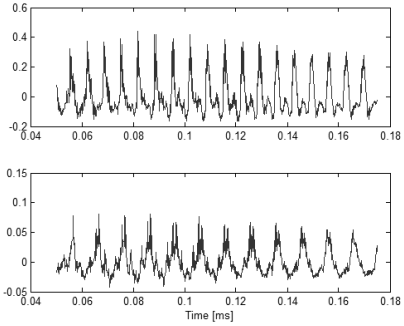

Fig. 5.16 Zoom on the first vowel of *Paint the circuits*. Top: original signal; Bottom: robotized signal with frame shift set to 10 ms (hence producing an artificial pitch of 100 Hz)

```
output_signal=zeros(length(input_signal),1);
pin=0;pout=0;

while pin+frame_length<length(input_signal)

    % STFT
    analysis_frame=input_signal(pin+1:pin+frame_length).* window;
    dft=fft(analysis_frame);

    % Setting all phases to zero
    dft=abs(dft);

    % ISTFT
    synthesis_frame=ifft(dft).*window;
     output_signal(pout+1:pout+frame_length) = ...
         output_signal(pout+1:pout+frame_length)+synthesis_frame;
```



```
    pin=pin+frame_shift;
    pout=pout+frame_shift;

end;

% Zoom on a few ms of signal, before and after robotization
clf;
range=(400:1400);
subplot(211);
plot(range/Fs,input_signal(range))
subplot(212);
plot(range/Fs,output_signal(range))

specgram(output_signal,frame_length,Fs,window);
```

5.2.2 Time scale modification

In this Section we examine methods for modifying the duration of an input signal without changing its audio spectral characteristics (e.g., without changing the pitch of any of the instruments). The test sentence we use here contains several chunks, sampled at 44100 Hz (Fig. 5.17, left). It starts with 1000 Hz sine wave, followed by a speech excerpt. The sound of a (single) violin comes next, followed by an excerpt of a more complex polyphonic musical piece (*Time* by Pink Floyd).

```
[input_signal,Fs]=wavread('time_scaling.wav');

frame_length=2048;
frame_shift=frame_length/4;
window=hanning (frame_length,'periodic');
COLA_ratio=sum(window.*window)/frame_shift;
window=window/sqrt(COLA_ratio);

specgram(input_signal,frame_length,Fs,window);
```

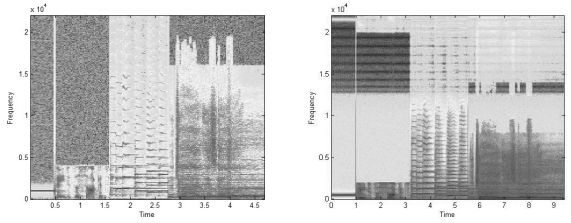

Fig. 5.17 Spectrogram of the test signal. Left: original signal; Right: after time-stretching by a factor of two

Interpolating the signal

Increasing the length of the input signal by a factor of two, for instance, is easily obtained by interpolating the signal, using the `resample` function provided by MATLAB, and not telling the digital-to-analog converter (DAC) about the sampling frequency change. This operation, however, also changes the frequency content of the signal: all frequencies are divided by two. Similarly, speeding-up the signal by a factor two would multiply all frequencies by two. This is sometimes referred to as the "chipmunk effect". Notice incidentally the aliasing introduced by the imperfect low-pass filter used by `resample` (Fig. 5.17, right).

```
resampled_signal=resample(input_signal,2,1);

specgram(resampled_signal,frame_length,Fs,window);
```

Time-scale modification with Weighted OverLap-Add (WOLA)

It is also possible to modify the time scale by decomposing it into ovelapping frames, changing the analysis frame shift into a different synthesis frame shift, and applying weighted overlap-add (WOLA) to the resulting synthesis frames. While this technique works well for unstructured signals, its application to harmonic signals produces unpleasant audio artifacts at the frequency of the synthesis frame rate, due to the loss of synchronicity between excerpts of the same partials in overlapping synthesis frames (Fig. 5.18). This somehow reminds us of the robotization effect obtained in Section 1.3.

```
frame_length=2048;
synthesis_frame_shift=frame_length/4;
window=hanning (frame_length,'periodic');
COLA_ratio=sum(window.*window)/synthesis_frame_shift;
window=window/sqrt(COLA_ratio);

time_scaling_ratio=2.85;
analysis_frame_shift=...
   round(synthesis_frame_shift/time_scaling_ratio);

pin=0;pout=0;
output_signal=zeros(time_scaling_ratio*length(input_signal),1);

while (pin+frame_length<length(input_signal)) ...
       && (pout+frame_length<length(output_signal))

    analysis_frame = input_signal(pin+1:pin+frame_length).*...
       window;
    synthesis_frame = analysis_frame.* window;
    output_signal(pout+1:pout+frame_length) = ...
       output_signal(pout+1:pout+frame_length)+synthesis_frame;

    pin=pin+analysis_frame_shift;
    pout=pout+synthesis_frame_shift;

    % Saving frames for later use
    if (pin==2*analysis_frame_shift) % 3rd frame
```



```
        frame_3=synthesis_frame;
    elseif (pin==3*analysis_frame_shift) % 4th frame
        frame_4=synthesis_frame;
    end;

end;

% Plot two overlapping synthesis frames and show the resulting
output signal
clf;
ax(1)=subplot(211);
range=(2*synthesis_frame_shift:2*synthesis_frame_shift+...
   frame_length-1);
plot(range/Fs,frame_3)
hold on;
range=(3*synthesis_frame_shift:3*synthesis_frame_shift+...
   frame_length-1);
plot(range/Fs,frame_4,'r')
ax(2)=subplot(212);
range=(2*synthesis_frame_shift:3*synthesis_frame_shift+...
   frame_length-1);
plot(range/Fs,output_signal(range))
xlabel('Time [ms]');
linkaxes(ax,'x');
set(gca,'xlim',[0.045 0.06]);

soundsc(output_signal,Fs);
```

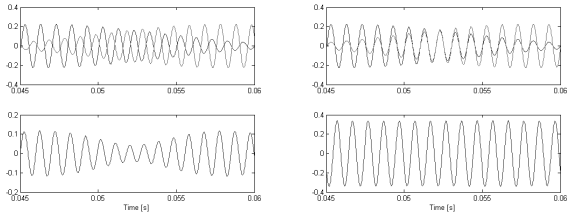

Fig. 5.18 Time-scale modification of a sinusoidal signal. Top: zoom on the overlap of two successive synthesis frames; Bottom: the resulting output signal; Left: using WOLA alone; Right: using the phase vocoder.

Time-scale modification with the phase vocoder

Let us now modify the time scale of the input signal without affecting its frequency content, by using a phase vocoder, which is a STFT-based signal processing system with specific hypotheses on the STFT. Time-scaling is again achieved by modifying the analysis frame shift without changing the synthesis frame shift, but the STFT has to be modified, so as to avoid creating phase mismatches between overlapped synthesis frames.

We use the periodic Hanning window, which provides a low spectral leakage while its main spectral lobe width is limited to $4*F_s/N$. The choice of the frame length N is dictated by a maximum bandwidth constraint for the DFT sub-band filters, since the phase vocoder is based on the hypothesis that each DFT bin is mainly influenced by a single partial, i.e. a sinusoidal component with constant frequency. For a Hanning window, each sinusoidal component will drive 4 DFT bins, the width of each is F_s/N. Clearly, the higher the value of N, the better the selectivity of the sub-band filters. On the other hand, a high value of N tends to break the stationarity hypothesis on the analysis frame (i.e., the frequency of partials will no longer be constant). We set N to 2048 samples here, which imposes the width of each DFT bin to 21 Hz, and the bandwidth of the DFT sub-band filters to 86 Hz. We set the frame shift to $N/4$=512 samples, which meets the COLA constraint for the Hanning window. What is more, this choice obeys the Shannon theorem for the output of each DTF bin, seen as the output of a sub-band filter. The result is much closer to what is expected (Fig. 5.19).

```
NFFT=2048;
frame_length=NFFT;
synthesis_frame_shift=frame_length/4;
window=hanning (frame_length,'periodic');
COLA_ratio=sum(window.*window)/synthesis_frame_shift;
window=window/sqrt(COLA_ratio);

time_scaling_ratio=2.65;
analysis_frame_shift=round(synthesis_frame_shift/...
   time_scaling_ratio);
% Central frequency of each DFT channel
DFT_bin_freqs=((0:NFFT/2-1)*2*pi/NFFT)';

pin=0;pout=0;

output_signal=zeros(time_scaling_ratio*length(input_signal),1);
last_analysis_phase=zeros(NFFT/2,1);
last_synthesis_phase=zeros(NFFT/2,1);

while (pin+frame_length<length(input_signal)) ...
       && (pout+frame_length<length(output_signal))

    % STFT
    analysis_frame=input_signal(pin+1:pin+frame_length).* window;
    dft=fft(fftshift(analysis_frame));
    dft=dft(1:NFFT/2);

    % PHASE MODIFICATION
    % Find phase for each bin, compute by how much it increased
    % since last frame.
    this_analysis_phase = angle(dft);
    delta_phase = this_analysis_phase - last_analysis_phase;
    phase_increment=delta_phase-analysis_frame_shift...
        *DFT_bin_freqs;

    % Estimate the frequency of the main partial for each bin
    principal_determination=mod(phase_increment+pi,2*pi)-pi;
```



```
    partials_freq=principal_determination/analysis_frame_shift+...
       DFT_bin_freqs;

    % Update the phase in each bin
    this_synthesis_phase=last_synthesis_phase+...
        synthesis_frame_shift*partials_freq;

    % Compute DFT of the synthesis frame
    dft= abs(dft).* exp(j*this_synthesis_phase);

    % Remember phases
    last_analysis_phase=this_analysis_phase;
    last_synthesis_phase=this_synthesis_phase;

    % ISTFT
    dft(NFFT/2+2:NFFT)=fliplr(dft(2:NFFT/2)');
    synthesis_frame = fftshift(real(ifft(dft))).* window;
    output_signal(pout+1:pout+frame_length) = ...
        output_signal(pout+1:pout+frame_length)+synthesis_frame;

    pin=pin+analysis_frame_shift;
    pout=pout+synthesis_frame_shift;

    % Saving the estimated frequency of partials for later use
    if (pin==2*analysis_frame_shift) % 3rd frame
        partials_freq_3=partials_freq;
    end;

end;

specgram(output_signal(1:end-5000),frame_length,Fs,window);
```

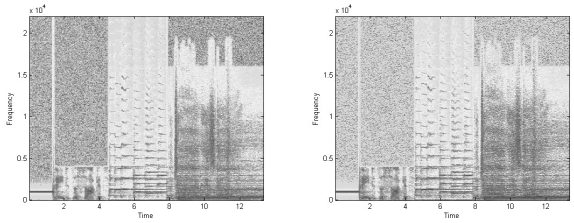

Fig. 5.19 Spectrogram of the test signal after time-stretching by a factor 2.85 using a phase vocoder with horizontal phase locking (left) and with both horizontal and vertical phase locking (right)

The phasing effect we had observed in the previous method has now disappeared (Fig. 5.18, right).

Let us check how the normalized frequency of the initial sinusoid (i.e. $1000*2*\pi/F_s$ = 0.142 rad/s) has been estimated. The first spectral line of

the sinusoid mainly influences DTF bins around index 1000/(44100/2048)=46.43. As observed on Fig. 5.20 (left), bins 42 to 51 (which correspond to the main spectral lobe) have indeed measured the correct frequency.

```
subplot(2,1,1);
plot((0:NFFT/2-1),partials_freq_3);
set(gca,'xlim',[25 67]);

subplot(2,1,2);
third_frame=input_signal(2*analysis_frame_shift:2*analysis_fra
me_shift+frame_length-1);
dft=fft(third_frame.* window);
dft=20*log10(abs(dft(1:NFFT/2)));
plot((0:NFFT/2-1),dft);
set(gca,'xlim',[25 67]);
```

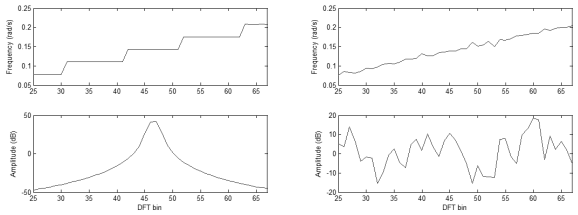

Fig. 5.20 Estimated (normalized) frequency of the partials in DFT bins 25 to 65 (top) and spectral amplitude for the same bins (bottom), in a phase vocoder. Left: in the sinusoid; Right: in more complex polyphonic music

However, for more complex sounds (as in the last frame of our test signal), the estimated partial frequency in neighboring bins vary a lot (Fig. 5.20, right).

```
subplot(2,1,1);
plot((0:NFFT/2-1),partials_freq);
set(gca,'xlim',[25 67]);

subplot(2,1,2);
dft=fft(analysis_frame);
dft=20*log10(abs(dft(1:NFFT/2)));
plot((0:NFFT/2-1),dft);
set(gca,'xlim',[25 67]);
```

As a result, although the spectrogram of the time-scaled signal looks similar to that of the original signal (except of course for the time axis), it exhibits significant phasiness and transient smearing.

The phase-locked vocoder

We have seen in the previous paragraphs that, in the case of a sinusoidal input signal, the estimated frequency of the main partial in several neighboring DFT bins (around the frequency of the sinusoid) is somehow constant. One of the reasons of the phasiness in the previous version of the phase vocoder comes from the fact that it does not enforce this effect. If other small amplitude partials are added to the sinusoid, each DFT bin computes its own partial frequency, so that these estimations in neighboring bins will only coincide by chance. The phase-locked vocoder changes this, by locking the estimation of partial frequencies in bins surrounding spectral peaks. This is called *vertical phase-locking*. In the following MATLAB lines, we give an implementation of the phase modification stage in the identity phase-locking scheme proposed in (Laroche and Dolson 1999).

MATLAB function involved:

- `function [maxpeaks,minpeaks] = findpeaks(x)` returns the indexes of the local maxima (`maxpeaks`) and of the local minima (`minpeaks`) in array `x`.

```
% PHASE MODIFICATION
% Find phase for each bin, compute by how much it increased
 % since last frame.
this_analysis_phase = angle(dft);
delta_phase = this_analysis_phase - last_analysis_phase;
phase_increment=delta_phase-analysis_frame_shift*...
    DFT_bin_freqs;

% Find peaks in DFT.
peaks = findpeaks(abs(dft));

% Estimate the frequency of the main partial for each bin
principal_determination(peaks)=...
    mod(phase_increment(peaks)+pi,2*pi)-pi;
partials_freq(peaks)=principal_determination(peaks)/...
    analysis_frame_shift+DFT_bin_freqs(peaks);

% Find regions of influence around peaks
regions = round(0.5*(peaks(1:end-1)+peaks(2:end)));
regions = [1;regions;NFFT/2];

% Set the frequency of partials in regions of influence to
% that of the peak (this is not strictly needed; it is used
 % for subsequent plots)
for i=1:length(peaks)
    partials_freq(regions(i):regions(i+1)) = ...
        partials_freq(peaks(i));
end

% Update the phase for each peak
this_synthesis_phase(peaks)=last_synthesis_phase(peaks)+...
```

```
    synthesis_frame_shift*partials_freq(peaks);

% force identity phase locking in regions of influence
for i=1:length(peaks)
    this_synthesis_phase(regions(i):regions(i+1)) = ...
        this_synthesis_phase(peaks(i)) +...
         this_analysis_phase(regions(i):regions(i+1))-...
          this_analysis_phase(peaks(i));
end
```

As a result of vertical phase-locking, a much larger number of bins around the main spectral line of our sinusoid have been assigned the same partial frequency. More importantly, this property has been enforced in more complex sounds, as shown in the last frame of our test signal.

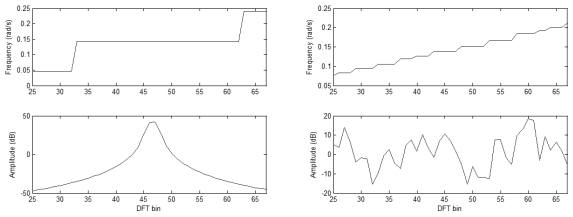

Fig. 5.21 Estimated (normalized) frequency of the partials in DFT bins 25 to 65 (top) and spectral amplitude for the same bins (bottom), in a phase-locked vocoder. Left: in the sinusoid; Right: in more complex polyphonic music

The result is improved, although it is still far from natural. It is possible to attenuate transient smearing by using separate analysis windows for stationary and transient sounds (as done in the MPEG audio coders). We do not examine this option here.

5.2.3 Pitch modification

The simplest way to produce a modification of the frequency axis of a signal is simply to resample it, as already seen in Section 2.1. This method, however, also produces time modification. In this Section we focus on methods for modifying the pitch of an input signal without changing its duration.

Time-scale modification and resampling
In order to avoid the time-scaling effect of a simple resampling-based approach, a compensatory time-scale modification can be applied, using a phase vocoder. Let us multiply the pitch by a factor 0.7, for instance. We first multiply the duration of the input signal by 0.7, and then resample it by 10/7, so as to recover the original number of samples.

MATLAB function involved:

`function [output_signal] = phase_locked_vocoder(input_signal, time_scaling_ratio)` is a simple implementation of the phase locked vocoder described in (Laroche and Dolson 1999).

```
[input_signal,Fs]=wavread('time_scaling.wav');

time_scaling_ratio=0.7;
resampling_ratio=2;

output_signal=phase_locked_vocoder(input_signal,...
     time_scaling_ratio);
pitch_modified_signal=resample(output_signal,10,7);

soundsc(input_signal,Fs);
soundsc(pitch_modified_signal,Fs);
```

The time scale of the output signal is now identical to that of the input signal. Notice again the aliasing introduced by the imperfect low-pass filter used by `resample` (Fig. 5.22, left).

```
specgram(pitch_modified_signal(1:end-5000),frame_length,...
     Fs,window);
```

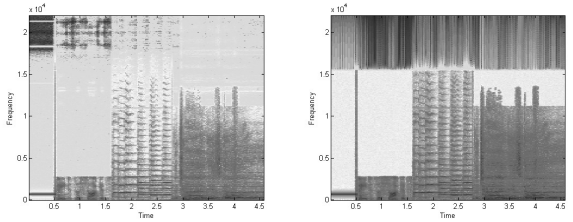

Fig. 5.22 Spectrogram of the test signal after pitch modification by a factor 0.7, using a phase vocoder followed by a resampling stage (left) and a fully integrated STFT-based approach (right)

The STFT-based approach

It is also possible, and much easier, to perform pitch modification in the frequency domain by translating partials to new frequencies, i.e. inside the phase vocoder. This technique is much more flexible, as it allows for non-linear modification of the frequency axis.

Notice that the implementation we give below is the simplest possible version: the peak frequencies are not interpolated for better precision, and the shifting is rounded to an integer number of bins. This will produce modulation artifacts on sweeping sinusoids, and phasing artifacts on audio signals such as speech. Moreover, when shifting the pitch down, peaks are truncated at DC, whereas they should be reflected with a conjugation sign.

```
NFFT=2048;
frame_length=NFFT;
frame_shift=frame_length/4;
window=hanning (frame_length,'periodic');
COLA_ratio=sum(window.*window)/frame_shift;
window=window/sqrt(COLA_ratio);

pitch_scaling_ratio=0.7;

% Central frequency of each DFT channel
DFT_bin_freqs=((0:NFFT/2-1)*2*pi/NFFT)';

pin=0;pout=0;

output_signal=zeros(length(input_signal),1);
accumulated_rotation_angles=zeros(1,NFFT/2);

while pin+frame_length<length(input_signal)

  % STFT
  analysis_frame = input_signal(pin+1:pin+frame_length)…
     .* window;
  dft=fft(fftshift(analysis_frame));
  dft=dft(1:NFFT/2);

  % Find peaks in DFT.
  peaks = findpeaks(abs(dft));

  % Find regions of influence around peaks
  regions = round(0.5*(peaks(1:end-1)+peaks(2:end)));
  regions = [1;regions;NFFT/2];

  % Move each peak in frequency according to modification
  % factor.

  modified_dft = zeros(size(dft));
  for u=1:length(peaks)

     % Locate old and new bin.
     old_bin = peaks(u)-1;
     new_bin = round(pitch_scaling_ratio*old_bin);

     % Be sure to  stay within 0-NFFT/2 when
     % shifting/copying the peak bins
     if(new_bin-old_bin+regions(u) >= NFFT/2) break; end;
     if(new_bin-old_bin+regions(u+1) >= NFFT/2)
         regions(u+1) = NFFT/2 - new_bin + old_bin;
     end;
     if(new_bin-old_bin+regions(u) <= 0)
         regions(u) = 1 - new_bin + old_bin;
     end;

     % Compute the rotation angle required, which has
     % to be cumulated from frame to frame
     rotation_angles=accumulated_rotation_angles(old_bin+1)...
         + 2*pi*frame_shift*(new_bin-old_bin)/NFFT;

     % Overlap/add the bins around the peak, changing the
     % phases accordingly
     modified_dft(new_bin-old_bin+(regions(u):regions(u+1)))= ...
       modified_dft(new_bin-old_bin+(regions(u):regions(u+1)))...
       + dft(regions(u):regions(u+1)) * exp(j*rotation_angles);
     accumulated_rotation_angles((regions(u):regions(u+1))) = ...
       rotation_angles;

  end


  % ISTFT
  modified_dft(NFFT/2+2:NFFT)=fliplr(modified_dft(2:NFFT/2)');
  synthesis_frame = fftshift(real(ifft(modified_dft))).* window;
  output_signal(pout+1:pout+frame_length) = ...
      output_signal(pout+1:pout+frame_length)+synthesis_frame;

  pin=pin+frame_shift;
  pout=pout+frame_shift;

end;
```


The time scale of the output signal is still identical to that of the input signal, and the frequency content above `Fs*time_scaling_ratio` is set to zero (Fig. 5.22, right).

```
specgram(output_signal(1:end-5000),frame_length,Fs,window);
```

5.3 Going further

For people interested in the use of transforms for audio processing, (Sethares 2007) is an excellent reference.

An important research area for phase vocoders is that of transient detection, for avoiding the transient smearing effect mentioned in this Chapter (see Röbel 2003 for instance).

One of the extensions of phase vocoders which have not been dealt with in this Chapter is the fact that not only phases but also amplitudes should be modified when performing time-scale modifications. In the frequency domain, a chirp sinusoid has a wider spectral lobe than a constant-frequency sinusoid. Exploiting this fact, however, would imply an important increase in computational and algorithmic complexity. Sinusoidal modeling techniques (McAulay and Quatieri 1986), which have been de-

veloped in parallel to the phase vocoder, are more suited to handle such effects.

For monophonic signals, such as speech, a number of time-domain techniques known as synchronized OLA (SOLA) have also been developed with great success for time scaling and pitch modification, given their very low computational cost compared to the phase vocoder. The wavefom Similarity SOLA (WSOLA), and pitch-synchronous (PSOLA) techniques are presented in a unified framework in Verhelst *et al.* (2000). Their limitations are exposed in (Laroche 1998).

The multi-band resynthesis OLA (MBROLA) technique (Dutoit 1993) attempts to take the best of both worlds: it performs off-line frequency-domain modifications of the signal to make it more amenable to time-domain time scaling and pitch modification. Recently, Laroche (2003) has also proposed a vocoder-based technique for monophonic sources, relying on a (possibly online) pitch detection, and able to perform very flexible pitch and formant modifications.

5.4 Conclusion

Modifying the duration or the pitch of a sound is not as easy as it might seem at first sight. Simply changing the sampling rate only partially does the job. In this Chapter, we have implemented a phase-locked vocoder and tested it for time scale and pitch scale modification. We have seen that processing the signal in the STFT domain provides better results, although not perfect yet. One of the reasons for the remaining artifacts lies in the interpretation of the input signal in terms of partials, which are neither easy to spot, nor even always clearly defined (in transients, for instance). For pitch shifting, we have also implemented an all-spectral approach, which is simpler than using a phase coder followed by a sampling rate converter, but still exhibits artifacts.

5.5 References

Arfib D, Keiler F, Zölzer U (2002) Time-frequency processing. In: DAFX: Digital Audio Effects. U. Zölzer, Ed. Hoboken (NJ): John Wiley & Sons

Bagdasarian R (1958) The Chipmunk song (Christmas don't be late). Jet Records, UK

Dolson M (1986) The phase vocoder: A tutorial. Computer Music Journal, 10-4, pp 14–27

Dutoit T (1997) Time-domain algorithms. In: An Introduction to Text-To-Speech Synthesis. Dutoit T. Dordrecht: Kluwer Academic Publishers

Flanagan L, Golden RM (1966) Phase Vocoder. Bell System Technical Journal pp 1493–1509 [online] Available: http://www.ee.columbia.edu/~dpwe/e6820/papers/FlanG66.pdf[07/06/2007]

Laroche J (1998) Time and pitch-scale modification of audio signals. In: Applications of Digital Signal Processing to Audio Signals. M. Kahrs and K. Brandenburg, Eds. Norwell, MA: Kluwer Academic Publishers

Laroche J, Dolson M (1999) Improved Phase Vocoder Time-Scale Modification of Audio. IEEE Trans. Speech and Audio Processing, 3, pp 323–332

Laroche J, Dolson M (1999b) New Phase Vocoder Technique for Pitch-Shifting, Harmonizing and Other Exotic Effects. IEEE Workshop on Applications of Signal Processing to Audio and Acoustics. Mohonk, New Paltz, NY. [online]Available: http://www.ee.columbia.edu/~dpwe/papers/LaroD99-pvoc.pdf [07/06/2007]

McAulay R, Quatieri T (1986) Speech analysis/Synthesis based on a sinusoidal representation. IEEE Transactions on Acoustics, Speech, and Signal Processing 34:4, pp 744–754

Puckette MS (1995) Phase-locked Vocoder. Proc. IEEE Conf. on Applications of Signal Processing to Audio and Acoustics, Mohonk

Röbel A (2003) A new approach to transient processing in the phase vocoder. Proc. 6th Int. Conference on Digital Audio Effects (DAFx-03), pp 344–349

Sethares WA (2007) Rhythm and Transforms. London: Springer Verlag [online] Available: http://eceserv0.ece.wisc.edu/~sethares/vocoders/Transforms.pdf [20/3/08]

Smith JO (2007) Example of Overlap-Add Convolution. In: Spectral Audio Signal Processing, Center for Computer Research in Music and Acoustics (CCRMA), Stanford University [online] Available: http://ccrma.stanford.edu/~jos/sasp/Example_Overlap_Add_Convolution.html [14/06/07]

Smith JO (2007b) Dual of Constant Overlap-Add. In: Spectral Audio Signal Processing, Center for Computer Research in Music and Acoustics (CCRMA), Stanford University [online] Available: http://ccrma.stanford.edu/~jos/sasp/ Example_Overlap_Add_Convolution.html [14/06/07]

Verhelst V, van Compernolle D, Wambacq P (2000). A unified view on synchronized overlap-add methods for prosodic modifications of speech. In Proc. ICSLP2000, Vol. 2, 63–66

Chapter 6

How can marine biologists track sperm whales in the oceans?

Hear they are!

T. Dutoit (°), V. Kandia(+), Y. Stylianou (*)

(°) Faculté Polytechnique de Mons, Belgium
(+) Foundation for Research and Technology-Hellas, Heraklion, Greece
(*) University of Crete, Heraklion, Greece

Whale watching has become a trendy occupation these last years. It is carried on in the waters of some 40 countries, plus Antarctica. Far beyond its touristic aspects, being able to spot the position of whales in real time has several important scientific applications, such as censusing (estimation of animal population), behavior studies and mitigation efforts concerning fatal collisions between marine mammals and ships and exposure of marine mammals to loud sounds of anthropogenic origin (seismic surveys, oil and gas exploitation and drilling, naval and other uses of sonar).

When whales do not appear on the surface of water, one way of detecting their position is to listen to the sounds they emit through several fixed *hydrophones* (i.e., microphones specially designed for underwater use).

Hydrophone signals are then processed with a SOund Navigation And Ranging (*sonar*) technique, which uses sound propagation under water to detect and spot objects. In this Chapter, we thus plunge deep into underwater acoustics, by examining a proof of concept for sperm whale tracking

using *passive acoustics*[1], and based on the estimation of the cross-correlation between hydrophone signals to spot whales on a 2D map.

6.1 Background – Source localization

We start by examining sperm whale sounds (Section 6.1.1), and show how the Teager-Kaiser operator efficiently increases the signal-to-noise ratio of hydrophone signals (Section 6.1.2).

Sperm whale sounds are actually received by hydrophones at slightly different times, due to the various distances between the animal and the hydrophones. Knowing the location of the hydrophones, the *time difference of arrival* (TDOA) between pairs of hydrophones can be obtained, using various techniques (Sections 6.1.3 and 6.1.4), and provided to a *multilateration* algorithm (Section 6.1.5) for determining the position of the animal.

6.1.1 Sperm whale sounds

Sperm whales are highly vocal active animals. An adult sperm whale produces some 25000 clicks a day. If we consider that the heart rate of such a large whale is around 15 beats per minute (i.e., 20000 beats a day), sperm whale produces more clicks than heart beats (Madsen 2002). Their repertoire is made up almost entirely of a number of click types with different properties.

Usual (or *regular*) *clicks* (Fig. 6.1) are the most commonly heard click type during deep foraging dives, and are therefore used to locate the animals using passive acoustic methods. They are impulsive broadband sounds of multi-pulse structure with inter-click interval (ICI) between 0.5s-1s. Usual clicks are highly directional sounds with source levels up to 235 dB rms re 1µPa[2]. It is believed that these properties represent adaptations for long-range echolocation.

[1] Sonars can be *active* or *passive*. Passive sonars only listen to their surrounding, while active sonars emit sounds and detect echoes from their surrounding. Whales and dolphins use echolocation systems similar to active sonars to locate predators and preys; marine biologists listen to the resulting sounds and echoes (hence, in a passive set-up) to detect cetaceans.

[2] For underwater sound, 1 µPa is the reference pressure level. Since the measured pressure level from a particular sound source decreases with distance from the source, the convention is to use one meter as a reference distance. Thus, the

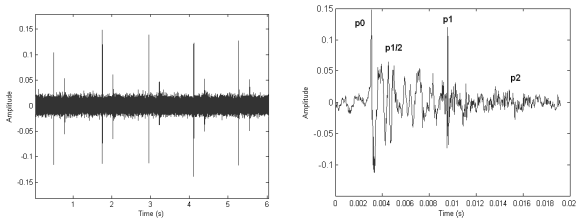

Fig. 6.1 Left: Sequence of usual sperm whale clicks. Each click is followed by an echo, due to surface reflection[3]. Right: zoom on a single click, showing the multi-pulse structure of the signal. The notation on pulses follows that of (Møhl et al. 2000, Zimmer et al. 2005)

Creak clicks (Fig. 6.2) are burst of mono-pulsed clicks with high repetition rate (up to 200 clicks per second). They are highly directional sounds with source level between 180-205 dB rms re 1μPa. They are produced during foraging dives and it is believed that they have a function analogous to the terminal buzzes produced by bats during echolocation [4].

Several other types of sounds are also encountered, such as *coda clicks*, *chirrup clicks*, *slow clicks*, *squeals*, and *trumpets,* most of which have a social communicative role. [5]

pressure level from a sound source is measured in "dB rms re 1μPa at 1m" (and the "at 1m" is usually omitted).

[3] Hydrophones are mounted at roughly 5m off the bottom, so that reflection from the bottom comes with a ~7ms delay, i.e. inside the direct path click itself. Furthermore, the hydrophones have an upward directed beam pattern.

[4] After detecting a potential prey using low rate echolocation calls, bats emit a characteristic series of calls at a high repetition rate (a *terminal buzz*) to localize the prey.

[5] For more information on sperm whale sounds, please refer to Madsen 2002, Drouot 2003, Teloni *et al.* 2005.

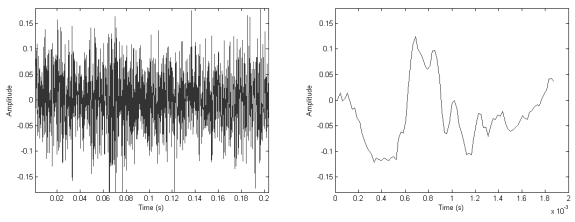

Fig. 6.2 Left: Segment of a creak, composed of several clicks (and their echoes) in high amplitude noise. Right: zoom on a single click of the segment, which shows the mono-pulse nature of the sound.

As shown in Fig. 6.1 and Fig. 6.2, hydrophone signals exhibit very low signal-to-noise ratios. In particular they are often contaminated by low-frequency noise, mostly due to human activity. Shipping indeed, which accounts for more of 75% of all human sound in the sea (ICES 2005), produces low-frequency sounds (rumble of engines, propellers). Commercial shipping traffic is growing as does the tonnage (cargo capacity) of ships, adding more noise in the low frequency band.

6.1.2 The Teager-Kaiser energy operator

The Teager–Kaiser (TK) energy operator is defined in the continuous domain as:

$$TK[x(t)] = \dot{x}(t)^2 - x(t)\ddot{x}(t) \tag{6.1}$$

where $\dot{x}$ and $\ddot{x}$ denote the first and second derivative over time, respectively. For a discrete time signal, it is shown in (Kaiser 1990) that the TK energy operator is given by:

$$TK[x(n)] = x^2(n) - x(n+1)x(n-1) \tag{6.2}$$

The TK operator is referred to as *energy operator* because it is related to the concept of energy in the generation of acoustic waves (Kaiser 1990).

This operator can be seen as a special case of *quadratic filters* defined by:

$$y(n) = \sum_{i_1=-\infty}^{\infty} \sum_{i_2=-\infty}^{\infty} h(i_1, i_2) x(n-i_1) x(n-i_2) \tag{6.3}$$

where $h(i_1,i_2)$ is known as the second order *Volterra kernel* [6]. As a matter of fact, the TK energy operator is obtained from (6.2) when:

$$h(i_1, i_2) = \begin{cases} 1 & i_1 = i_2 = 0 \\ -1/2 & (i_1, i_2) = (1,-1) \text{ or } (-1,1) \\ 0 & \text{otherwise} \end{cases} \tag{6.4}$$

The analysis of such non-linear filters is not trivial. Since their output to a sum of inputs is not the sum of their outputs to isolated inputs, the classical notions of impulse and frequency responses no longer have the usual reach. When the input is composed of a sum of sinusoids, for instance, cross-product terms appear in the output. In the case of the TK operator, it is easy to check from (6.2) that the response of this filter to $\delta(n)$ is $\delta(n)$, and that its response to an isolated cosine $A\cos(n\varphi_0)$ is a constant signal $A^2\sin^2(\varphi_0)$. This response, however, cannot be interpreted as a frequency response, and is not the Fourier transform of the impulse response.

In the specific case of noisy impulsive signals, the TK operator has interesting properties. Let us assume that the signal $x(n)$ recorded by a hydrophone is composed of the sum of a low frequency interference signal *i(n)*, of the impulsive click signal *s(n)* produced by a sperm whale, and of some background wideband noise *u(n)*:

$$x(n) = s(n) + i(n) + u(n) \tag{6.5}$$

[6] Quadratic filters are themselves a simple case of the more general *polynomial* or *Volterra filters* (Sicuranza 1992):

$$\begin{aligned} y(n) = & \sum_{i_1=-\infty}^{\infty} h(i_1) x(n-i_1) \\ & + \sum_{i_1=-\infty}^{\infty} \sum_{i_2=-\infty}^{\infty} h(i_1, i_2) x(n-i_1) x(n-i_2) \\ & + \ldots \\ & + \sum_{i_1=-\infty}^{\infty} \ldots \sum_{i_N=-\infty}^{\infty} h(i_1, \ldots, i_N) x(n-i_1) \ldots x(n-i_N) \end{aligned}$$

If the interference frequency is low and the signal-to-noise ratio (SNR) between the impulsive clicks and the background noise is sufficient, it is shown in (Kandia and Stylianou 2005) that the TK operator has the very interesting property of ignoring the interference component while increasing the SNR (Fig. 6.3). Moreover, it does not smear input pulses in time (Fang and Atlas 1995). As a matter of fact, if we neglect the influence of the noise $u(n)$ and identify $x(n)$ to a Dirac pulse $\delta(n)$ and $i(n)$ in the vicinity of this pulse to a constant value K, it is easy to show from (6.2) that the output $y(n)$ of the TK filter only has three non zero samples:

$$y(n) = \begin{cases} -K & n = -1 \\ 1-2K & n = 0 \\ -K & n = 1 \end{cases} \tag{6.6}$$

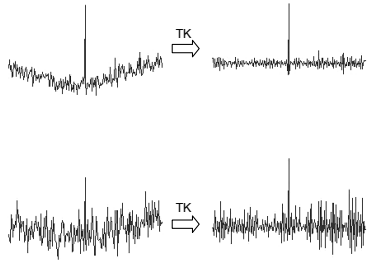

Fig. 6.3 Applying the Teager-Kaiser operator to simulated clicks corrupted by low-frequency interference and background noise with various SNR efficiently pre-processes the data for later click detection or TDOA estimation (Kandia and Stylianou 2007).

The application of the TK operator to sperm whale sounds therefore provides a good example of the efficient use of a non-linear filter for making the data more amenable to further TDOA estimation.

6.1.3 TDOA estimation based on the generalized cross correlation

Let us assume that signals $x_1(t)$ and $x_2(t)$ result from the propagation of signal $s(t)$ through different paths that are identified to a simple attenuation and a delay:

$$x_i(t) = a_i s(t-\tau_i) + b_i(t) \tag{6.7}$$

where $b_i(t)$ (i=1,2) are zero-mean, uncorrelated stationary random processes, which are also non-correlated with $s(t)$.

The *cross* power spectrum density (PSD) $S_{x_1x_2}(f)$ between x_1 and x_2 is given by:

$$\begin{aligned} S_{x_1x_2}(f) &= \int_{-\infty}^{\infty} \phi_{x_1x_2}(t) e^{-j2\pi ft} dt \\ &= a_1 a_2 e^{-j2\pi f\tau_{12}} \int_{-\infty}^{\infty} \phi_{ss}(t) e^{-j2\pi ft} dt \\ &= a_1 a_2 e^{-j2\pi f\tau_{12}} S_{ss}(f) \end{aligned} \tag{6.8}$$

in which $\phi_{x_1x_2}(t)$ is the cross correlation between $x_1(t)$ and $x_2(t)$, $\phi_{ss}(t)$ is the autocorrelation of $s(t)$, and $S_{ss}(f)$ is therefore the PSD of $s(t)$.

Knapp and Carter (1976) have proposed to estimate the *time difference of arrival* TDOA $\tau_{12} = \tau_1 - \tau_2$ between $x_1(t)$ and $x_2(t)$, as the position of the maximum of the *generalized cross correlation function*, defined as:

$$\psi_{x_1x_2}(\tau) = \int_{-\infty}^{+\infty} \Phi(f) S_{x_1x_2}(f) e^{j2\pi f\tau} df \tag{6.9}$$

in which $\Phi(f)$ is a weighting function.

In particular, when $\Phi(f)$ is set to 1, $\psi_{x_1x_2}(\tau)$ is the inverse Fourier Transform of the cross PSD, i.e., the standard cross-correlation function. For signals verifying (6.8), this leads to have:

$$\psi_{x_1x_2}(\tau) = a_1 a_2 \phi_{ss}(\tau - \tau_{12}) \tag{6.10}$$

in which $\phi_{ss}(\tau)$ is the autocorrelation function of $s(t)$. Since the maximum of $\phi_{ss}(\tau)$ is always found at $\tau = 0$, τ_{12} can be estimated as the position of the maximum of (6.10). However, if $b_1(t)$ and $b_2(t)$ exhibit some cross-correlation, (6.10) becomes:

$$\psi_{x_1x_2}(\tau) = a_1 a_2 \phi_{ss}(\tau - \tau_{12}) + \phi_{b_1b_2}(\tau) \tag{6.11}$$

whose maximum may not correspond to τ_{12}. In particular, if $b_i(t)$ are sinusoidal components with the same frequency, a sinusoidal term will appear in (6.11), due to a spectral line in $S_{x_1x_2}(f)$.

When $\Phi(f)$ is set to $1/|S_{x_1x_2}(f)|$, we obtain the so-called *phase transform,* which computes the generalized cross correlation from the phase of the cross PSD. For signals verifying (6.8), the phase transform still provides a perfect estimate of τ_{12}, since:

$$\Phi(f) S_{x_1x_2}(f) = \frac{S_{x_1x_2}(f)}{|S_{x_1x_2}(f)|} = e^{-j2\pi f\tau_{12}} \tag{6.12}$$

which leads to:

$$\psi_{x_1x_2}(\tau) = \delta(\tau - \tau_{12}) \tag{6.13}$$

Now if $b_i(t)$ are sinusoidal components with the same frequency, their contributions to $\psi_{x_1x_2}(\tau)$ will be much lower than in (6.11), since the spectral line in $S_{x_1x_2}(f)$ will be canceled in (6.12). This makes the phase transform an interesting estimator, provided $S_{x_1x_2}(f)$ can itself be correctly estimated.

In practice, $S_{x_1x_2}(f)$ is estimated in [0, F_s] from a finite number of samples of $x_1(n)$ and $x_2(n)$, as (Fig. 6.4):

$$S_{x_1x_2}(k\frac{F_s}{N}) \cong X_1(k) X_2^*(k) \quad (k = 0,\ldots,N-1) \tag{6.14}$$

where $X_i(k)$ (i=1,2) is the N-point Discrete Fourier Transform of the sequence $[x_i(n), x_i(n-1), \cdots, x_i(n-N+1)]$. This is known to give a biased but consistent estimation of $S_{x_1x_2}(f)$.

TDOA estimation from *real* signals is not so easy. As mentioned above, hydrophone signals are polluted with interference signals and additive noise due to surrounding sources other than the one we want to spot. More annoyingly, several propagation paths may exist to each hydrophone: typically a direct path with lowest attenuation plus many secondary paths,

which create echoes (one of which being due to surface reflection) and reverberations. As a result, a more realistic model for hydrophone signals x_1 and x_2 receiving a click $s(t)$ after propagation is given by:

$$x_i(t) = h_i(t) * s(t) + b_i(t) \tag{6.15}$$

where * denotes convolution, $h_i(t)$ is the acoustic impulse response of the channel between the whale and the ith hydrophone (including reverberation), and $b_i(t)$ is the additive noise received by this hydrophone. Consequently, applying the phase transform to real signals does not always lead to efficient estimates of TDOAs. Several other weighting functions have been proposed for the estimation of the general cross-correlation in (6.9), and proved to provide interesting results in specific cases (Knapp and Carter 1976).

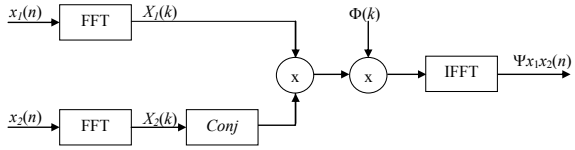

Fig. 6.4 Practical computation of the generalized cross correlation function

6.1.4 Adaptive TDOA estimation

A radically different approach was proposed in (Benesty 2000), which leads to a simple and elegant adaptive filtering implementation, in the framework of the Least Means Squares (LMS) algorithm. This approach is based on the estimation of the impulse responses from the source to the receivers.

If we neglect the noise components in (6.15), we have (Fig. 6.5):

$$\begin{aligned} x_1(n) * h_2(n) &= s(n) * h_1(n) * h_2(n) \\ &= s(n) * h_2(n) * h_1(n) \\ &= x_2(n) * h_1(n) \end{aligned} \tag{6.16}$$

which can be written in vector notation as:

$$\mathbf{x}_1^T(n)\mathbf{h_2} = \mathbf{x}_2^T(n)\mathbf{h_1} \qquad \text{for all } n \tag{6.17}$$

where

$$\mathbf{x}_i(n) = [x_i(n), x_i(n-1), \cdots, x_i(n-M+1)]^T \quad (i = 1,2) \tag{6.18}$$

and

$$\mathbf{h}_i = [h_i(0), h_i(1), \cdots, h_i(M-1)]^T \quad (i = 1,2) \tag{6.19}$$

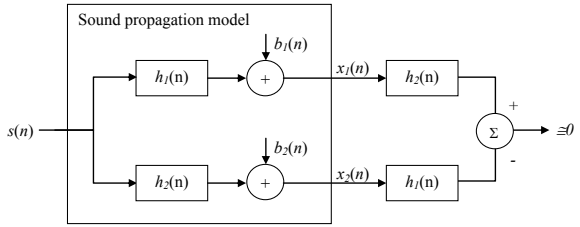

Fig. 6.5 If additive noise is neglected, passing $x_1(n)$ through h_2 gives the same result as passing $x_2(n)$ through h_1.

If we stack $\mathbf{x}_1(n)$ and $\mathbf{x}_2(n)$ as:

$$\mathbf{x}(n) = \begin{bmatrix} \mathbf{x}_1(n) \\ \mathbf{x}_2(n) \end{bmatrix} \tag{6.20}$$

then the covariance matrix of $\mathbf{x}(n)$ is given by:

$$\mathbf{R}_{\mathbf{xx}} = \begin{bmatrix} \mathbf{R}_{\mathbf{x}_1\mathbf{x}_1} & \mathbf{R}_{\mathbf{x}_1\mathbf{x}_2} \\ \mathbf{R}_{\mathbf{x}_2\mathbf{x}_1} & \mathbf{R}_{\mathbf{x}_2\mathbf{x}_2} \end{bmatrix} \tag{6.21}$$

with:

$$\mathbf{R}_{\mathbf{x}_i\mathbf{x}_j} = E\left[\mathbf{x_i}(n)\mathbf{x}_j^T(n)\right] \quad (i, j = 1,2) \tag{6.22}$$

It then follows from (6.17) and (6.21) that:

$$\mathbf{Ru} = 0 \quad \text{with } \mathbf{u} = \begin{bmatrix} \mathbf{h}_2 \\ -\mathbf{h}_1 \end{bmatrix} \tag{6.23}$$

This remarkably simple result provides a simple means of estimating the impulse responses $\mathbf{h}_1$ and $\mathbf{h}_2$ from the estimation of $\mathbf{R}_{\mathbf{xx}}$.

In practice, though, accurate estimation of **u** is not trivial, as the impulse responses may be long, and background noise may falsify (6.23). Benesty (2000) has therefore proposed an adaptive estimation of **u,** based on a Least Means Squares (LMS) principle. The idea is to find an error function $e(n)$ such that the expectation $E[e^2(n)]$ is minimized when (6.23) is verified, and such that the gradient of $E[e^2(n)]$ with respect to **u** has a simple analytical form**.** Clearly, $e(n)=\mathbf{u}^T\mathbf{x}$ meets these requirements, as $E(e^2(n))=\mathbf{u}^T\mathbf{Ru}$ and $\nabla_{\mathbf{u}}(\mathbf{u}^T\mathbf{Ru}) = 2\mathbf{Ru}$. Minimizing it on **u** can then be achieved by the *gradient descent* approach (Fig. 6.6): by starting from an initial guess and updating it in the direction of the steepest descent, i.e., in the direction of the negative gradient of the cost function with respect to the **u**:

$$\begin{aligned} \mathbf{u}^{new} &= \mathbf{u}^{old} - \frac{\mu}{2}\nabla_{\mathbf{u}}(\mathbf{u}^T\mathbf{Ru})\Big|_{\mathbf{u}=\mathbf{u}^{old}} \\ &= \mathbf{u}^{old} - \mu\mathbf{Ru}^{old} \end{aligned} \tag{6.24}$$

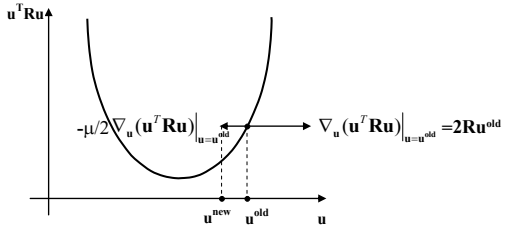

Fig. 6.6 A schematic 2D view of the gradient descent approach

The classical simplification of the LMS approach (Widrow and Stearns 1985) is then to replace **R** in (6.24) by its very short term estimate

$E(\mathbf{xx}^T)\approx\mathbf{xx}^T$, and to update **u** every sample, leading to the following update equation for $\mathbf{u}(n)$:

$$\begin{aligned} \mathbf{u}(n+1) &= \mathbf{u}(n) - \mu\mathbf{x}(n)\mathbf{x}^T(n)\mathbf{u}(n) \\ &= \mathbf{u}(n) - \mu\mathbf{x}(n)e(n) \end{aligned} \tag{6.25}$$

It is also suggested by Benesty (2000) to constrain $\mathbf{u}(n)$ to unitary norm, in order to avoid round-off error propagation, and to avoid selecting the obvious **u**=0 solution in (6.23). This is achieved by changing (6.25) into:

$$\mathbf{u}(n+1) = \frac{\mathbf{u}(n) - \mu\mathbf{x}(n)e(n)}{\| \mathbf{u}(n) - \mu\mathbf{x}(n)e(n) \|} \tag{6.26}$$

Additionally, as only the TDOA is required, one only needs to estimate the direct paths in $\mathbf{h}_1$ and $\mathbf{h}_2$ (as opposed to the complete impulse responses). A simple solution is to initialize $\mathbf{h}_2$ (the first half of **u**) to a single Dirac pulse. A "mirror" effect follows from (6.23) in the estimate of $\mathbf{h}_1$ (the second half of **u**): a negative dominant peak appears, which is an estimation of the direct path of $\mathbf{h}_1$. The expected TDOA can then be computed as time difference between these peaks within their respective impulse responses. Since the TDOA can *a priori* be positive or negative, the initial Dirac pulse is positioned at the center of $\mathbf{h}_2$, and the value of M is set to twice the maximum value of the TDOA.

Provided the adaptation step μ is correctly set (not too high to avoid instability, but not too low to be able to track the TDOA as a function of time), this simple algorithm provides remarkably stable results. Moreover, applied to sperm whale spotting, it does not require a preliminary detection of clicks.

6.1.5 Multilateration

Given two hydrophone locations (x_1,y_1) and (x_2,y_2) and a known TDOA τ_{12}, the loci of possible whale positions (x,y) on a 2D map is obtained easily. The travel time of a click to each hydrophone is given by:

$$\tau_i = \frac{1}{c}\left(\sqrt{(x-x_i)^2+(y-y_i)^2}\right) \tag{6.27}$$

where c is the propagation speed of sound in water (typically 1510 m/s). The locus is thus defined by:

$$\tau_{ij} = \tau_i - \tau_j = \frac{1}{c}\left(\sqrt{(x-x_i)^2+(y-y_i)^2} - \sqrt{(x-x_j)^2+(y-y_j)^2}\right) \quad (6.28)$$

This corresponds to a half-hyperbola whose foci are the hydrophones, and whose semi-major (or transverse) axis is given by $c\,\tau_{ij}/2$ (Fig. 6.7), where c is the speed of sound in water.

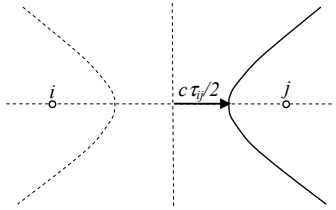

Fig. 6.7 The locus of whale positions for a given value of the TDOA τ_{ij} between hydrophones i and j

Obtaining the 2D position of the whale therefore requires two TDOAs (i.e. 3 hydrophones, which implicitly offers a third TDOA): a 2D *multilateration* system solves a system of two non-linear equations such as (6.28) to find the position of a whale in real time.

In practice, though, the half-hyperbola defined by the third TDOA may not have the same intersection as the first two ones, since TDOA estimates may not be very precise. It is then useful to increase the number of hydrophones. The location problem must then be restated as an optimization problem and solved by a least squares approach or with a Kalman filter (Huang *et al.* 2008). To keep things simple, we do not address this problem here.

6.2 MATLAB proof of concept: ASP_audio_effects.m

In this Section, we analyze sperm whale clicks received by several hydrophones (6.2.1), and use them to find the position of the cetacean. We start by processing the clicks with the (non-linear) Teager-Kaiser operator (6.2.2), so as to increase their signal-to-noise ratio. We then test cross-correlation estimation algorithms (6.2.3) to compute the time difference of

arrival (TDOA) between pairs of hydrophones, and compare them to an efficient adaptive estimation algorithm based on least means squares minimization (6.2.4). We conclude by a simple multilateration algorithm, which uses two TDOAs to estimate the position of the sperm whale (6.2.5).

6.2.1 Sperm whale sounds

Throughout Section 6.2, we will work on hydrophone signals collected at the Atlantic Undersea Test and Evaluation Center (AUTEC), Andros Island, Bahamas, and made available by the Naval Undersea War-fare Center (NUWC). These signals were provided to the 2nd International Workshop on Detection and Localization of Marine Mammals using Passive Acoustics, held in Monaco November 16–18, 2005 (Adam *et al.* 2006) and can be obtained from their website (Adam 2005). The Andros Island area has over five hundred square nautical miles of ocean that are simultaneously monitored via 68 broad-band hydrophones. The distance between most hydrophones is 5 nautical miles (about 7.5 km).

We will work with the sounds received by hydrophones I, G, and H (which we will rename as #1, #2, and #3 here), for about 30 seconds, recorded with a digital audio recorder at Fs=48 kHz.

Plotting the signals received by all three hydrophones (Fig. 6.8) shows variable attenuations and delays between clicks, as well as variable noise levels. Each direct path click is followed by an echo due to surface reflection. Obviously, clicks arrive at hydrophone #2 much earlier than at the other hydrophones. The amplitude of the signal on hydrophone #2 is also higher, and the noise level is lower. Moreover, the echo in signal #2 is much more delayed than in signals #1 and #3. All this suggests that the sperm whale is in the vicinity of hydrophone #2.

```
[hydrophone1,Fs]=wavread('hydrophone1.wav');
[hydrophone2,Fs]=wavread('hydrophone2.wav');
[hydrophone3,Fs]=wavread('hydrophone3.wav');

time=(0:length(hydrophone1)-1)/Fs;
ax(1)=subplot(3,1,1); plot(time,hydrophone1)
hold on;
ax(2)=subplot(3,1,2); plot(time,hydrophone2)
ax(3)=subplot(3,1,3); plot(time,hydrophone3)
linkaxes(ax,'x');
```

Since the delay between clicks is larger than the interclick interval, we will first apply a fixed time-shift between signals, so as to approximately align corresponding clicks. We shift the first signal by 7.72 ms to the left,

the second signal by 5.4 s to the left, third signal by 7.58 s to the left, and keep 25 s of each signal. We will later account for these artificial shifts.

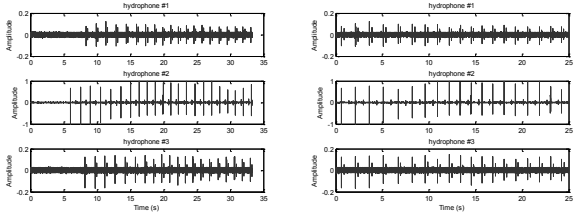

Fig. 6.8 Left: Clicks received by hydrophones #1, #2, and #3. Echoes are clearly visible between clicks (notice that the echo received at hydrophone #2 is more delayed than for the other hydrophones); Right: Same signals, time-shifted so as to approximately align corresponding clicks.

```
n_samples=25*Fs;
shift_1=fix(7.72*Fs);
shift_2=fix(5.40*Fs);
shift_3=fix(7.58*Fs);
[hydrophone1,Fs]=wavread('hydrophone1.wav',[1+shift_1 ...
   shift_1+n_samples]);
[hydrophone2,Fs]=wavread('hydrophone2.wav',[1+shift_2 ...
   shift_2+n_samples]);
[hydrophone3,Fs]=wavread('hydrophone3.wav',[1+shift_3 ...
   shift_3+n_samples]);
```

Zooming on a single click (Fig. 6.9) shows that whale clicks are not as simple as Dirac samples, which makes the visual estimation of TDOAs not very precise, and their automatic estimation not trivial.

Let us compute the PSD of the background noise in hydrophone #3 and compute its standard deviation, for later use. It appears (Fig. 6.10) that the noise is pink[7], with 40 dB more power around f=0Hz than around f=24000 Hz.

```
real_noise=hydrophone3(1.26e4:1.46e4);
real_noise_std=std(real_noise)

pwelch(real_noise,[],[],[],Fs);
```

[7] Strictly speaking, a *pink noise* is defined as a signal $b(t)$ whose power spectral density $S_{bb}(f)$ is proportional to the reciprocal of the frequency $1/f$. We use the term *pink* very loosely here, to indicate that the PSD of the noise decreases with frequency.


```
real_noise_std =   0.0073
```

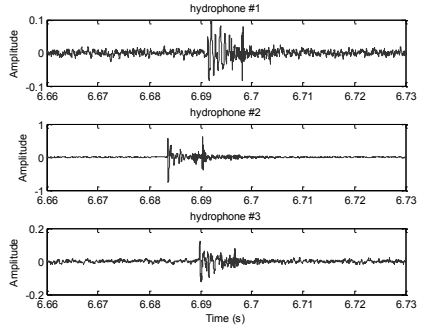

Fig. 6.9 Zoom on a single click

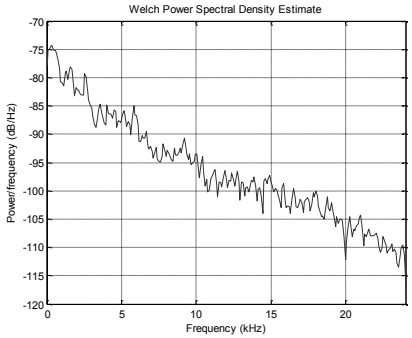

Fig. 6.10 Power spectral density of the background pink noise measured by hydrophone #3.

Notice that the data used here does not reveal LF interferences, which may however occur in measurements and make the estimation of TDOA still more complex.

6.2.2 Teager-Kaiser filtering

In order to better understand the Teager-Kaiser non-linear filter, we first apply it to synthetic signals, composed of clicks (assimilated to Dirac impulses), background noise (white and pink noise), and an interference component (modelled as a sinusoid). We then show that, even though the effect of a general non-linear filter on a sum of signals is not the sum of its outputs to isolated input signals, the effect of the TK operator on synthetic clicks with background noise and sinusoidal interference can still somehow be analyzed in terms of its effect on isolated components.
We finally apply the TK operator on real sperm whale sounds, and compare it to a simple linear high-pass filter.

Dirac pulse input

We start with the response of the TK operator to a Dirac impulse $0.1\delta(n-4000)$, with the same order of magnitude as that of the clicks we found in the previous Section. The output is the input impulse, squared (Fig. 6.11, left).

```
click = [zeros(1,3999) 0.1 zeros(1,4000)];

% Applying TK filter
L = length(click);
click_response = click(2:L-1).^2-click(1:L-2).*click(3:L);
click_response = [click_response(1) click_response ...
    click_response(L-2)]; % lenght(output)=length(input)

subplot(2,1,1); plot(click);
subplot(2,1,2); plot(click_response);
```

Sinusoidal input

Sinusoidal interference signals of the form $A\cos(n\phi_0)$ produce a constant output $A^2\sin^2(\phi_0)$. To check this, we generate 8000 samples of a chirp $\cos(n\phi_0(n))$ with $\phi_0(n)=\phi_{max}n/8000$ and $\phi_{max}=0.1$ (which corresponds to frequencies from 0 to $F_s\phi_{max}/2\pi = 763$ Hz). The result is plotted on Fig. 6.12 (left).

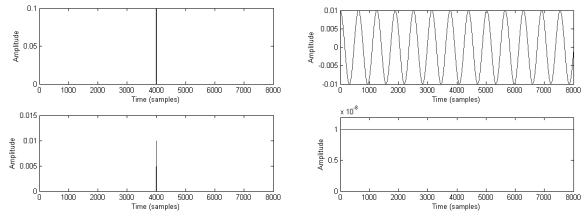

Fig. 6.11 Response of the TK filter to a Dirac pulse $0.1\delta(n-4000)$ (Left) and to a cosine $0.01\cos(0.01n)$ (Right)

MATLAB function involved:

- `y = teager_kaiser(x)` applies the Teager-Kaiser energy operator to input signal `x`.

```
Fs=48000;
phi_max=0.1;
test=chirp(0:1/Fs:7999/Fs,0,7999/Fs,Fs*phi_max/2/pi);
test_response=teager_kaiser(test);

subplot(2,1,1); plot(test);
subplot(2,1,2); plot(test_response);
```

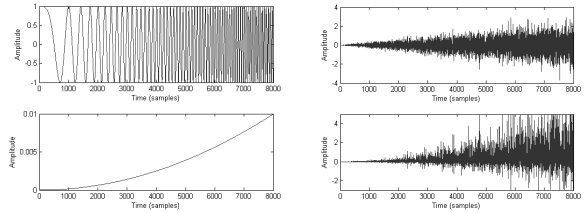

Fig. 6.12 Response of the TK filter to a chirp (Left) and to to white noise with linearly increasing standard deviation (Right)

The value of the output is close to zero when ϕ_0 is small, which will be the case for a typical LF interference component in sperm whale click measurements (Fig. 6.11, right).

```
interference = 0.01*cos (0.01*(0:7999)); % A²sin²(0.01)~1e-8
interference_response=teager_kaiser(interference);

subplot(2,1,1); plot(interference);
subplot(2,1,2); plot(interference_response);
```

White noise input

TK filtering of Gaussian white noise $N(0,\sigma)$ produces noise with mean of the order of σ^2 and standard deviation proportional to σ^2. We check this on white noise with linearly increasing standard deviation (see Fig. 6.12, right).

```
test = (1/8000:1/8000:1).*randn(1, 8000);
test_response = teager_kaiser(test);

subplot(2,1,1); plot(test);
subplot(2,1,2); plot(test_response);
```

The output noise is still white, but it is not Gaussian (Fig. 6.13).

```
noise = randn(1, 8000);
noise_response = teager_kaiser(noise);

subplot(2,1,1); pwelch(noise,[],[],[],Fs);
subplot(2,1,2); pwelch(noise_response,[],[],[],Fs);

subplot(2,1,1); hist(noise,30);
subplot(2,1,2); hist(noise_response,60);
```

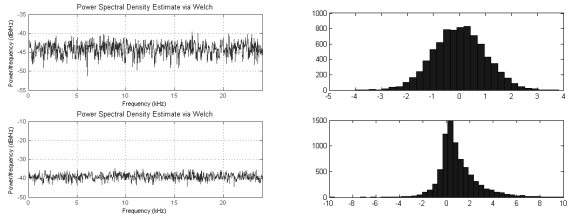

Fig. 6.13 Left: PSD of white noise input and its filtering thought the TK filter; Right: Histograms of the corresponding samples

Pink noise input

Let us now test the TK filter on pink noise similar to the one we found on real hydrophone signals in 0. We first create a filter with linear frequency response from -30 dB at f=0 to -65dB at f=F_s/2. Applying this filter to white noise $N(0,1)$ produces a realistic pink noise component (Fig. 6.14).

```
w=0:0.1:1;
a_dB=-35*w-30;
a=10.^(a_dB/20);
[B,A] = fir2(20,w,a);
freqz(B,A,512,48000);

pink_noise=filter(B,A,noise);
pwelch(pink_noise,[],[],[],Fs);
pink_noise_std=std(pink_noise)
```

```
pink_noise_std =   0.0108
```

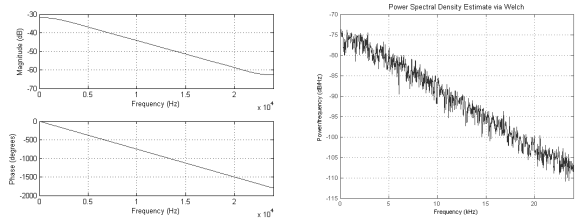

Fig. 6.14 Left: Frequency response of a noise shaping filter; Right: PSD of the resulting pink noise

TK filtering of pink noise produces pink noise, whose standard deviation is still proportional to the square of that of the original noise (although with a smaller proportionality factor than for white input noise; Fig. 6.15 Left).

```
test=100*(1/8000:1/8000:1).*pink_noise;
test_response = teager_kaiser(test);

subplot(2,1,1);
plot(test);
subplot(2,1,2);
plot(test_response);
```

The output noise is whiter; the PSD of the input noise has been decreased by about 50 dB in low frequencies and by about 30 dB in high frequencies (Fig. 6.15 Right).

```
pink_noise_response = teager_kaiser(pink_noise);

subplot(2,1,1);
pwelch(pink_noise,[],[],[],Fs);
subplot(2,1,2);
pwelch(pink_noise_response,[],[],[],Fs);
```

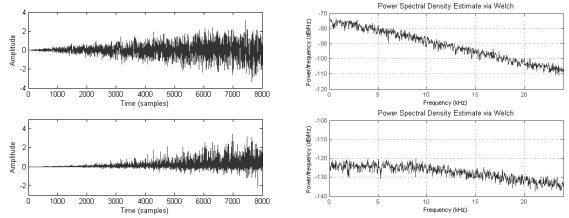

Fig. 6.15 Left: TK filtering of pink noise with linearly increasing standard deviation; Right: PSDs of input pink noise (with constant variance) and of its filtering through the TK filter.

Complex input

We now apply the TK operator to the complete synthetic signal, obtained by summing all three components: click, interference, and pink noise (Fig. 6.16, left). Thanks to the TK operator, the impulse is highlighted in the signal. The LF signal is removed, and the pink noise is very much attenuated relatively to the impulse. The SNR is therefore highly increased.

```
input=click+interference+pink_noise;
output_tk=teager_kaiser(input);

subplot(2,1,1);
plot(input);
subplot(2,1,2);
plot(output_tk);
```

Notice that the TK operator did not smear the impulsive part of the input waveform (Fig. 6.16, right).

```
set(gca,'xlim',[3980 4020]);
```


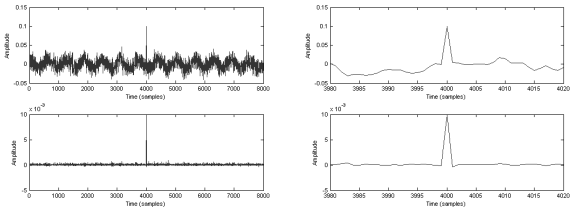

Fig. 6.16 Top left: Synthetic hydrophone signal composed of the sum of a Dirac pulse, sinusoidal interference, and background noise; Bottom left: Same signal, filtered by the TK operator; Right: Same signal as on the left, zoom around the Dirac pulse

Notice also that the effect of the TK filter on the SNR depends on the original value of the SNR. As a matter of fact, TK squares the amplitude A of the impulse and the standard deviation σ of the noise. Let us characterize the "visibility" of the pulse by the ratio $A/3\sigma$, since noise samples take most of their values in $[-3\sigma, 3\sigma]$. If this ratio is significantly higher than 1 (in the previous plot it was 0.1/0.033=3), the visibility of the pulse in the TK filtered signal, $A^2/9\sigma^2$, is higher than in the original signal. If $A/3\sigma$ approaches 1 but is still higher, the impulse may not be visible in the original signal, but more easily detected in the output signal. To check this, we multiply the standard deviation of the noise by two (Fig. 6.17).

```
input=click+2*pink_noise+interference;
output_tk=teager_kaiser(input);
```

Summarizing, the TK operator has a high-pass filtering effect on background pink noise (and it squares its standard deviation), a notch effect on possible low-frequency interference (while a DC component appears), and almost no effect on Dirac pulses (apart from squaring their amplitude): it acts as a signal-dependent filter and amplifier[8].

[8] In practice, it also produces cross terms when several components are summed at the input. We did not examine them here, as they are not essential for the kind of signal we deal with.

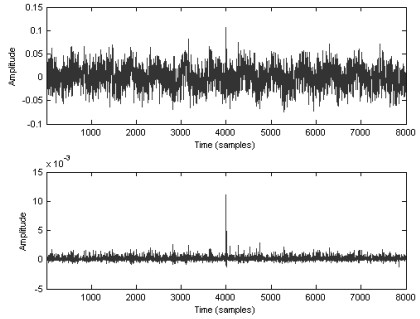

Fig. 6.17 Top: a Dirac pulse hidden in background noise; Bottom: same signal, after TK filtering

Hydrophone signal

Finally we apply the TK operator to a real click taken from a hydrophone signal, as examined in Section 1, and compare its effect to that of a linear filter.

We first create a linear-phase high-pass symmetric FIR filter whose frequency response has approximately the same effect on the pink background noise as the TK filter: a frequency-linear attenuation, from -50 dB close to f=0 to -30 dB close to f=F_s/2 (Fig. 6.18, Left). The length of its impulse response is 31 (its order is 30). It therefore has a delay of 15 samples.

```
clf;
w=0:0.1:1;
a_dB=20*w-50;
a=10.^(a_dB/20);
N=30;
[B,A] = fir2(N,w,a);
freqz(B,A,512,Fs);
```

Obviously the TK operator does a very good job at highlighting the click. While the linear filter has a similar effect on background noise, it also decreases the power of the click, thereby decreasing the overall SNR (Fig. 6.18, Right).

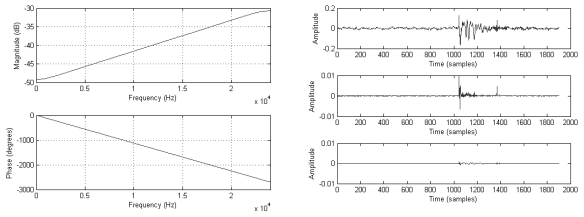

Fig. 6.18 Left: Frequency response of a linear filter having the same effect on background noise as the TK filter; Right: a real click (top), filtered by the TK operator (middle) and by the linear filter (bottom).

```
input=wavread('hydrophone3.wav',[464750 466650]);
output_tk=teager_kaiser(input);
% Taking the filter dealy into account
output_lin=[filter(B,A,input(16:end)) ; zeros(15,1)];

subplot(3,1,1); plot(input);
subplot(3,1,2); plot(output_tk);
subplot(3,1,3); plot(output_lin);
```

6.2.3 TDOA estimation using generalized cross-correlation

We will now show how generalized cross correlation (GCC) can be used for estimating TDOAs. We start again by applying it to simple synthetic signals, and end with real hydrophone signals.

Synthetic input

We first create a simple synthetic signal, as in Section 6.2.2, but with more severe interference and noise, and a second signal, with other noise samples, a phase-shifted interference signal, and an attenuated click, delayed by 60 samples (Fig. 6.19, Left).

```
click = [zeros(1,499) 0.1 zeros(1,500)];
interference = 0.03*cos (0.01*(0:999));
noise = 2*randn(1, 1000);
w=0:0.1:1;
a_dB=-35*w-30;
a=10.^(a_dB/20);
[B,A] = fir2(20,w,a);
pink_noise=filter(B,A,noise);

signal_1=click+interference+pink_noise;
```

```
click_2=circshift(click'*0.5,60)';
interference_2 = 0.02*cos (pi/4+0.01*(0:999));
noise_2 = randn(1, 1000); % new noise samples
pink_noise_2=filter(B,A,noise_2);

signal_2=click_2+interference_2+pink_noise_2;

subplot(2,1,1); plot(signal_1);
subplot(2,1,2); plot(signal_2);
```

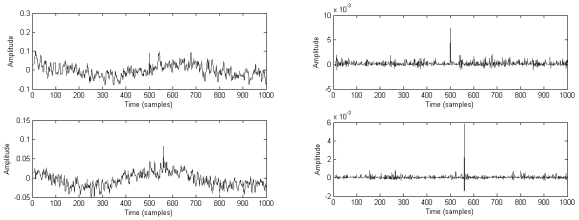

Fig. 6.19 Left: two synthetic hydrophone signals, with a delay of 60 samples between their respective Dirac pulses; Right: same signal, filtered by the TK operator.

Estimating the time difference of arrival (TDOA) between the filtered signals with the standard cross-correlation does not provide the expected value (-60 samples), as the total cross-correlation function is dominated by the correlation between the sinusoidal interference components. Notice that the (biased) cross-correlation is obtained via FFT-IFFT (Fig. 6.20), after making sure that the number of FFT samples is high enough for linear convolution to be identical to circular convolution (as obtained by multiplying FFTs).

```
% Computing the standard biased cross-correlation via FFT-IFFT
M = min(length(signal_1),length(signal_2));
NFFT = 2*M-1; % so that linear convolution = circular convolution
x1 = signal_1 - mean(signal_1);
x2 = signal_2 - mean(signal_2);
X1 = fft(x1,NFFT);
X2 = fft(x2,NFFT);
S_x1x2 = X1.*conj(X2);
phi_x1x2 = ifft(S_x1x2);
% re-arranging the IFFT
standard_xcorr = [phi_x1x2(NFFT-M+2:NFFT) phi_x1x2(1:M)];
[val,ind]=max(standard_xcorr);
TDOA_standard_xcorr = ind-M  % in samples
```

```
TDOA_standard_xcorr =   41
```



```
subplot(2,1,1);
plot((0:1/M:(M-1)/M), 20*log10(abs(S_x1x2(1:M))) );
subplot(2,1,2);
plot((0:1/M:(M-1)/M), unwrap(angle(S_x1x2(1:M))) );
```

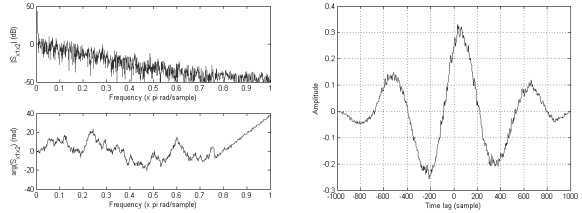

Fig. 6.20 Left: Cross PSD between the signals in (Fig. 6.19, Right); Right: Corresponding cross correlation function

```
plot((-M+1:M-1),standard_xcorr); grid;
```

Notice that the linear phase component in the HF part of (Fig. 6.20, left), where the pink noise is weak, reveals a delay. But since the amplitude of the noise and of the LF interference strongly dominate that of the HF part of the cross PSD, this delay cannot be observed in the cross correlation function.

Using the phase transform version of the generalized cross-correlation removes the LF interference and gives the same importance to all frequency bands in the phase spectrum. It produces a more prominent maximum, which leads to a correct estimate of the TDOA (Fig. 6.21).

```
phi_x1x2 = ifft(S_x1x2 ./ max(abs(S_x1x2),eps));
% re-arranging the IFFT
phase_transform = [phi_x1x2(NFFT-M+2:NFFT) phi_x1x2(1:M)];

[val,ind]=max(phase_transform);
TDOA_phase_transform = ind-M

subplot(2,1,1);
plot((0:1/M:(M-1)/M), 20*log10(abs(PT(1:M))));
subplot(2,1,2);
plot((0:1/M:(M-1)/M), unwrap(angle(PT(1:M))));

plot((-M+1:M-1),phase_transform); grid;
```

```
TDOA_phase_transform =  -60
```

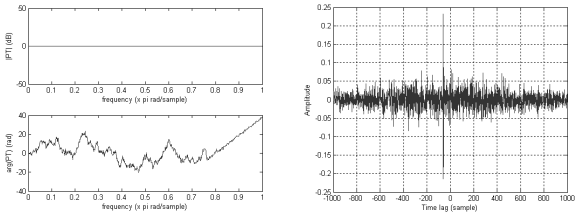

Fig. 6.21 Left: Discrete Fourier transform of the phase transform between the signals in (Fig. 6.19, Left); Right: Corresponding phase transform

Applying the TK operator obviously increases the SNR (Fig. 6.19, Right).

```
signal_1_tk=teager_kaiser(signal_1);
signal_2_tk=teager_kaiser(signal_2);

subplot(2,1,1);
plot(signal_1_tk);
subplot(2,1,2);
plot(signal_2_tk);
```

As a result of the suppression of the sinusoidal component, GCC produces an accurate result (Fig. 6.22, Left).

MATLAB function involved:

- `gcc(z1, z2, flag)` computes the generalized cross correlation (GCC) between signals z1 and z2, from FFT/IFFT, as specified in (Knapp & Carter 1976). `[flag]` makes it possible to choose the type of cross-correlation: standard cross correlation if `flag='cc'`; phase transform if `flag='phat'`.

```
standard_xcorr_tk = gcc(signal_1_tk, signal_2_tk, 'cc');
[val,ind]=max(standard_xcorr_tk);
TDOA_standard_xcorr_tk = ind-M  % in samples

plot((-M+1:M-1),standard_xcorr_tk); grid;
```

```
TDOA_standard_xcorr_tk =  -60
```

Using the phase transform on TK-filtered data still produces an even more prominent maximum (Fig. 6.22, Right).


```
phase_transform_tk = gcc(teager_kaiser(signal_1),
teager_kaiser(signal_2),  'phat');
[val,ind]=max(phase_transform_tk);
TDOA_phase_transform_tk = ind-M

plot((-M+1:M-1),phase_transform_tk); grid;
```

```
TDOA_phase_transform_tk =  -60
```

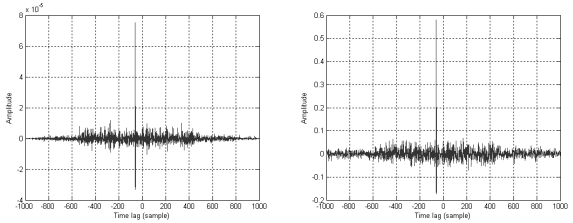

Fig. 6.22 Left: Cross correlation computed on TK-filtered signals; Right: Corresponding phase transform

Hydrophone signals

We now apply GCC to real hydrophone signals. Clicks are separated by about 400 samples (Fig. 6.23, Left).

```
Fs=48000;
shift_2=fix(5.40*Fs);
shift_3=fix(7.58*Fs);

[signal_1,Fs]=wavread('hydrophone2.wav',[460000+shift_2 ...
    460000+shift_2+2000]);
[signal_2,Fs]=wavread('hydrophone3.wav',[460000+shift_3 ...
    460000+shift_3+2000]);
```

Again, applying the TK operator has a positive effect on the SNR (Fig. 6.23, Right).

```
signal_1_tk=teager_kaiser(signal_1);
signal_2_tk=teager_kaiser(signal_2);

subplot(2,1,1); plot(signal_1_tk);
subplot(2,1,2); plot(signal_2_tk);
```

Applying the Phase Transform to these signals delivers a correct estimate of the TDOA: -404 samples (negative, since the click hits hydrophone #1 before hydrophone #2).

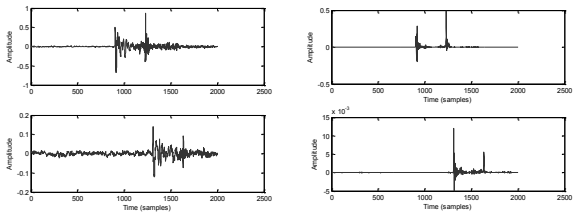

Fig. 6.23 Left: Two clicks, showing a delay of about 400 samples. Right: same, after TK filtering.

```
phase_transform_tk  = gcc(signal_1_tk, signal_2_tk,  'phat');
[val,ind]=max(phase_transform_tk );

M = min(length(signal_1),length(signal_2));
TDOA_phase_transform_tk= ind-M
```

```
TDOA_phase_transform_tk = -404
```

6.2.4 TDOA estimation using least-mean squares

In this section we will apply the Least Mean Squares (LMS) adaptive approach to TDOA estimation, again first on synthetic signals (with imposed TDOA), and then on real hydrophone signals.

Synthetic input

We start with the same synthetic signals as in Fig. 6.19, apply TK preprocessing, and run the Benesty's adaptive filtering algorithm for TDOA estimation within [-600,600] (samples) and step `mu`=0.01 (for full scale signals in [-1,+1]).

```
load synthetic_signals signal_1 signal_2
signal_1_tk=teager_kaiser(signal_1);
signal_2_tk=teager_kaiser(signal_2);

% Normalizing max signal amplitudes to +1
x1=signal_1_tk/max(signal_1_tk);
x2=signal_2_tk/max(signal_2_tk);

% LMS initialization
M = 600; % max value of the estimated TDOA
x1c = zeros(M,1);
x2c = zeros(M,1);
u = zeros(2*M,1);
```



```
u(M/2) = 1;
N = length(x1);
e = zeros(1,N);
tdoa = zeros(1,N);
peak = zeros(1,N);
mu = 0.01; % LMS step

% LMS loop
for n=1:N

    x1c = [x1(n);x1c(1:length(x1c)-1)];
    x2c = [x2(n);x2c(1:length(x2c)-1)];
    x = [x1c;x2c];

    e(n) = u'*x;
    u = u-mu*e(n)*x;
    u(M/2) = 1; %forcing g2 to an impulse response at M/2
    u = u/norm(u); %forcing ||u|| to 1

    [peak(n),ind] = min(u(M+1:end));
    peak(n)=-peak(n); % (positive) impulse in g1
    TDOA(n) = ind-M/2;

end

% Estimated TDOA(n), with values of the maximum peak in g1
subplot(2,1,1);
plot(TDOA);
xlabel('Time (samples)'); ylabel('TDOA (samples)');
subplot(2,1,2);
plot(peak);
xlabel('Time (samples)'); ylabel('peak');
```

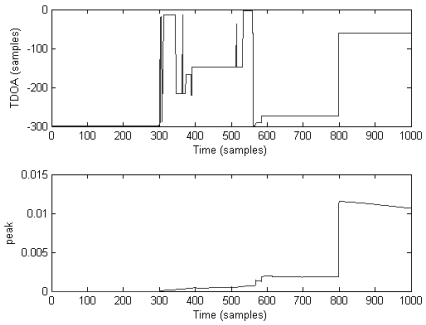

Fig. 6.24 Top: TDOA estimated by the LMS approach on the signals in Fig. 6.19, on an adaptive, sample-by-sample basis; Bottom: Value of the peak found in **h1** (the higher the peak, the more reliable the TDOA estimate)

It appears that the TDOA estimate is wrong at the beginning of the frame, as the adaptive filter must have started processing a click to start converging (Fig. 6.24). The value of the peak in **h1 (**the impulse response between the source and the first signal) is very low for these wrong TDOA values. The best TDOA estimate is the one that produced the most prominent peak in **h1**. This estimate is correct: -60 samples. After a click has been processed, the peak in **h1** starts decreasing again, but the value of TDOA remains correct.

```
[val,ind]=max(peak);
Best_estimate_TDOA=TDOA(ind)
```

```
Best_estimate_TDOA =  -60
```

Hydrophone signals

Applying the same algorithm to hydrophone signals #1 and #2 is straightforward.

```
Fs=48000;
n_samples=25*Fs;
shift_1=fix(7.72*Fs);
shift_2=fix(5.40*Fs);
[hydrophone1,Fs]=wavread('hydrophone1.wav',[1+shift_1 ...
   shift_1+n_samples]);
[hydrophone2,Fs]=wavread('hydrophone2.wav',[1+shift_2 ...
   shift_2+n_samples]);

% Subsampling by 6, for decreasing computational cost
signal_1=resample(hydrophone1,1,6);
signal_2=resample(hydrophone2,1,6);

% Applying Teager-Kaiser filter
signal_1_tk=teager_kaiser(signal_1);
signal_2_tk=teager_kaiser(signal_2);

subplot(2,1,1); plot(signal_1_tk);
subplot(2,1,2); plot(signal_2_tk);
```

The estimated TDOAs change by a few tens of milliseconds in our 25-seconds recording (Fig. 6.25)[9]. This shows that the sperm whale moved during the recording. Plotting our TODAs with the ones obtained by visual inspection of the signals proves that the LMS algorithm provided accurate results (the error is less than a millisecond).

[9] Again, the first TDOA estimates are not significant.

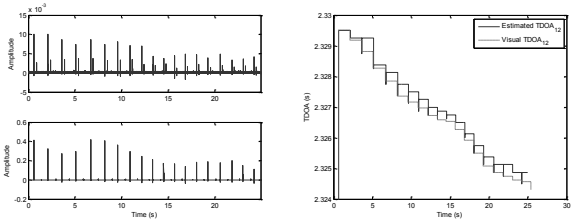

Fig. 6.25 Left: Hydrophone signals #1 and #2, after TK-filtering. Right; TDOA estimated from these signals, as a function of time, compared to the estimate obtained by visual inspection of the signals

MATLAB function involved:

- `[tdoa, peak] = TDOA_LMS(x1, x2, max_tdoa, mu)` is an implementation of (Benesty 2000) for source localization. It returns the time difference of arrival, in samples, between signals `x1` and `x2`, as a function of time, together with the value of the peak found in the estimate of the main propagation path for signal `x1`. This value can be used as a confidence value for the TDOA. `max_tdoa` is the maximum TDOA value returned and `mu` is the step weight.

```
TDOA_12 = TDOA_LMS(signal_1_tk,signal_2_tk,600,0.01);
TDOA_12 = TDOA_12*6; % taking downsampling into account
% Accounting for preliminary time-shifts
TDOA_12 = (TDOA_12 + shift_1-shift_2)/Fs;

load clicks; % clicks_1, clicks_2, and clicks_3 give the
             % positions of clicks in the respective
             % hydrophone signals, as estimated by visual
             % inspection.
TDOA_12_visual=(clicks_1-clicks_2)/Fs;

plot((0:length(TDOA_12)-1)*6/Fs,TDOA_12);
hold on;
stairs((clicks_2-shift_2)/Fs,TDOA_12_visual,'--r');
```

We repeat the operation for hydrophone signals #2 and #3 (Fig. 6.26).

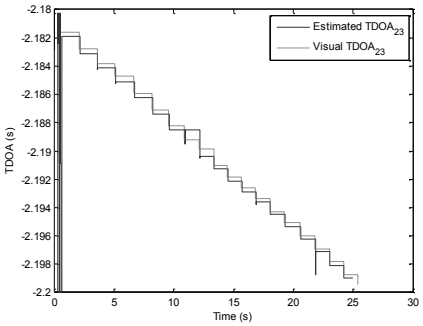

Fig. 6.26 TDOA estimated from hydrophone signals #2 and #3, as a function of time, compared to the estimate obtained by visual inspection of the signals

6.2.5 Multilateration

In this final Section, we use the TDOAs estimated in the previous Section from real hydrophones signals, for tracking a sperm whale on a 2D map. We develop a rudimentary multilateration system, on the basis of 2 TDOAs only.

The position of the hydrophones, taking some common point as a reference, is given in Table 6.1.

Table 6.1 Hydrophone relative locations and depth

Hydrophone	*X*	*Y*	*Z*
#1	14318,86m	-16189,18m	-1553,58m
#2	10658,04m	-14953,63m	-1530,55m
#3	12788,99m	-11897,12m	-1556,14m

We first show the position of the hydrophones on the map (Fig. 6.27), and then plot the hyperbolas corresponding to each pair of TDOA, assuming the speed of sound in water is 1510 m/s. Notice we drop the first TDOA estimates, which the LMS algorithm used for converging, and only use 10 estimates for our plot.


```
TDOA_12=TDOA_12(6000:length(TDOA_12)/10:end);
TDOA_23=TDOA_23(6000:length(TDOA_23)/10:end);

sound_speed=1510; % m/s
for i=1:length(TDOA_12)
   PlotHyp(hydrophone_pos(1,1), hydrophone_pos(1,2), ...
           hydrophone_pos(2,1), hydrophone_pos(2,2), ...
           -TDOA_12(i)*sound_speed/2,'b');
   PlotHyp(hydrophone_pos(2,1), hydrophone_pos(2,2), ...
           hydrophone_pos(3,1), hydrophone_pos(3,2), ...
           -TDOA_23(i)*sound_speed/2,'r');
end;
```

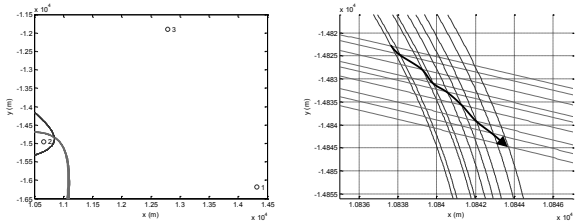

Fig. 6.27 Multilateration hyperbolas on the 2D hydrophone map. Left: general view; Right: zoom on the intersection area and observed movement

Since the maximum speed of a whale is close to 30 km/h (i.e. 8 m/s), it obviously could not move much around hydrophone #2 in the 25 s of data we have used for this proof-of-concept (Fig. 6.27, left).

Zooming around hydrophone #2 (Fig. 6.27, right) reveals its trajectory: it moved from (10837.5,-14822) to (10844,-14847), i.e. by about 26 m in 25 seconds, i.e. at a speed of about 1 m/s.

```
clf;
axis([10835, 10847, -14856, -14816]);
hold on;
for i=1:length(TDOA_12)-1
    PlotHyp(hydrophone_pos(1,1), hydrophone_pos(1,2), ...
            hydrophone_pos(2,1), hydrophone_pos(2,2), ...
            -TDOA_12(i)*sound_speed/2,'b');
    PlotHyp(hydrophone_pos(2,1), hydrophone_pos(2,2), ...
            hydrophone_pos(3,1), hydrophone_pos(3,2), ...
            -TDOA_23(i)*sound_speed/2,'r');
end;
distance = sqrt((10844-10837.5)^2+(14847-14822)^2)
```

```
distance = 25.8312
```

6.3 Going further

Readers interested in methods for estimating the position of a source from multiple TDOA estimations should report to (Huang *et al.* 2008).

MATLAB code for localization, including more elaborate multilateration algorithms, is available from the Beam Reach Marine Science and Sustainability School (Beam Reach 2007).

ISHMAEL (Integrated System for Holistic Multi-channel Acoustic Exploration and Localization) is free software for acoustic analysis, including localization , made freely available by Prof. Dave Mellinger at Oregon State University(Mellinger 2008). The same group also provides MobySound, a public a database for research in automatic recognition of marine animal calls (Mellinger 2007).

MATLAB-based UWB positioning software can also be found in Senad Canovic's Master thesis, from Norwegian University of Science and Technology (Canovic 2007).

6.4 Conclusion

In this Chapter, we have provided signal processing solutions for cleaning impulsive signals mixed in noise and low-frequency interferences, and for automatically estimating TDOAs based on either generalized cross-correlation or adaptive filtering.

Although we have examined these techniques in the specific context of whale spotting, they can easily be generalized to many positioning problems, such as the localization of speakers in a room, that of GSM mobile phones, of Global Positioning System (GPS) receivers, or to that of aircrafts or vehicles using either sonar or radar.

Last but not least, this Chapter also highlights one of the many "natural" abilities of human beings: that of using both ears to locate sound sources.

6.5 References

Adam O (2005) Website of the 2nd International Workshop on Detection and Localization of Marine Mammals using Passive Acoustics [online] Available: http://www.circe-asso.org/workshop/ [26/4/2008]

Adam O, Motsch JF, Desharnais F, DiMarzio N, Gillespie D, Gisiner RC (2006) Overview of the 2005 workshop on detection and localization of marine mammals using passive acoustics, Applied Acoustics 67:1061–1070

Beam Reach (2007) Acoustic Localization [online] Available: http://beamreach.org/soft/AcousticLocation/AcousticLocation-041001/ [23/6/2008]

Benesty J (2000) Adaptive eigenvalue decomposition algorithm for passive acoustic source localization. J. Acoust. Soc. Am. 107 (1): 384–391

Canovic S (2007) Application of UWB Technology for Positioning , a Feasibility Sudy. Master thesis, Norwegian University of Science and Technology [online] Available: http://www.diva-portal.org/ntnu/undergraduate/abstract.xsql?dbid=2091 [23/06/08]

Huang Y, Benesty J, Chen J (2007) Time delay estimation and source localization. In Springer Handbook of Speech Processing, Benesty J, Sondhi MM, Huang Y (eds), Springer:Berin, 51:1043–1063

Drouot V (2003) Ecology of sperm whale (Physeter macrocephalus) in the Mediterranean Sea. Ph.D. Thesis, Univ. of Whales, Bangor, UK

Fang J, Atlas LE (1995) Quadratic detectors for energy estimation. IEEE Transactions on Signal Processing 43–11: 2582–2594

Huang Y, Benesty J, Chen J (2008) Time delay estimation and source localization. In: Springer Handbook of Speech Processing, Benesty J, Sondhi MM, Huang Y (Eds.), Springer: Berlin. 1043–1063.

Kaiser JF (1990) On a simple algorithm to calculate the ''Energy'' of a signal. In: Proc. IEEE ICASSP, 381–384, Albuquerque, NM, USA

Kandia V, Stylianou Y (2005) Detection of creak clicks of sperm whales in low SNR conditions. In CD Proc. IEEE Oceans, Brest, France

Kandia V, Stylianou Y (2006) Detection of sperm whale clicks based on the Teager–Kaiser energy operator, Applied Acoustics 67:1144–1163

Kandia V, Stylianou Y (2006) Accurate TDOA (Time Difference of Arrival) using the Teager – Kaiser Energy Operator. In CD Proceedings of the 20th Annual Conference of the European Cetacean Society, Gdynia, Poland

Knapp CH and Carter GC (1976) The generalized correlation method for estimation of time delay. In: IEEE Transactions on Acoustic, Speech and Signal Processing 24, 320–327

ICES (International Council for the Exploration of the Sea - Advisory Committee on Ecosystems, ICES CM 2005/ACE:01) (2005) Report of the Ad-hoc Group on the Impact of Sonar on Cetaceans and Fish (AGISC).

Madsen PT (2002) Sperm whale sound production – in the acoustic realm of the biggest nose on record, In Madsen P.T., PhD. Dissertation, Sperm whale sound production., Dep. of Zoophysiology, University of Aarhus, Denmark

Mellinger D (2007) MobySound [online] Available: http://hmsc.oregonstate.edu/projects/MobySound/ [23/6/2008]

Mellinger D (2008) Ishmael Integrated System for Holistic Multi-channel Acoustic Exploration and Localization [online] Available: http://www.pmel.noaa.gov/vents/acoustics/whales/ishmael [23/6/2008]

Morrissey RP, Ward J, DiMarzio N, Jarvis S, Moretti DJ (2006) Passive acoustic detection and localization of sperm whales (Physeter macrocephalus) in the tongue of the ocean. Applied Acoustics 67:1091–1105

Møhl B, Wahlberg M, Madsen PT, Heerfordt A, Lund A (2000) Sperm whale clicks: Directionality and source level revisited, J. Acoust. Soc. Am. 107 (1): 638–648.

Sicuranza G (1992) Quadratic filters for signal processing. In: Proceedings of the IEEE, 80(8):1263–1285

Widrow B, Stearns SD (1985) Adaptive Signal Processing. Prentice-Hall,Inc., Upper Saddle River, NJ

Zimmer WMX, Madsen PT, Teloni V, Johnson MP, Tyack PL (2005) Off-axis effects on the multi-pulse structure of sperm whale usual clicks with implications for the sound production. J. Acoust. Soc. Am. 118: 3337–3345

Chapter 7

How could music contain hidden information?

*W*hy not *A*ttempt *T*o hid*E* the very p*R*esence
of a *M*ess*A*ge in an audio t*R*ac*K ?*

C. Baras (+), N. Moreau (*), T. Dutoit (°)

(+) GIPSA-lab (Grenoble Image Parole Signal et Automatique), Grenoble, France
(*) Ecole Nationale Supérieure des Télécommunications, Paris, France
(°) Faculté Polytechnique de Mons, Belgium

Audio watermarking started in the nineties as a modern and very technical version of playing cat and mouse[1]. The music industry, dominated by the "big four" record groups, also known as the "Majors" (Sony BMG, EMI, Universal and Warner), quickly realized that the availability of digital media for music recordings and the possibility to transfer it fast and degradation-free (thanks to the availability of high data-rate networks and of efficient data compression standards) would not only offer many benefits in terms of market expansion, but also expose their business to a great danger: that of piracy of intellectual property rights. Being able to insert proprietary marks in the media without affecting audio quality (i.e., in a

[1] The real origins of watermarking can be found in the old art of steganography (literally, "hidden writing", in ancient Greek). While the goal of cryptography is to render a message unintelligible, steganography attempts to hide the very presence of a message by embedding it into another information (Petitcolas *et al.* 1999)

"transparent way") was then recognized as a first step towards solving this issue. Additionally ensuring the robustness of the proprietary mark to not only usual media modifications (such as cropping, filtering, gain modification, or compression) but also to more severe piracy attacks quickly became a hot research topic worldwide.

Things changed radically in 2007 (i.e. before audio watermarking could be given a serious commercial trial). As a matter of fact, the Digital Rights Management (DRM) employed by the Majors to restrict the use of digital media (which did not make use of watermarking techniques) became less of a central issue after the last Majors stopped selling DRM-protected CDs in January 2007, and after S. Jobs' open letter calling the music industry to simply get rid of DRM, in February that same year (Jobs 2007).

Audio watermarking techniques are now targeted to more specific applications, related to the notions of "augmented content" or "hidden channel". One such application is audio fingerprinting[2], in which a content-specific watermark is inserted in the media, and can thus be retrieved for the automatic generation of the playlist of radio stations and music television channels for royalty tracking. Audio watermarking has also been proposed for the next generation of digital TV audience analysis systems: each TV channel would insert its own watermark in the digital audio and video streams it delivers, and a number of test TV viewers would receive a watermark detection system connected to a central computer[3].

In the next Section, we will first introduce the principles of spread spectrum and audio watermarking (7.1). This will be followed by a MATLAB-based proof of concept, showing how to insert a text message in an audio signal (7.2), and by a list of pointers to more advanced literature and software (7.3).

[2] The term *fingerprinting* is also used in the literature for specifying the technique which associates a short numeric sequence to an audio signal (i.e. some sort of *hashing;* Craver *et al*. 2001). We do not use it here with this meaning.

[3] With analog TV, audience analysis is much simpler: since each TV channel uses a specific carrier frequency, measuring this frequency through a sensor inserted in TV boxes gives the channel being watched at any time. This is no longer possible with digital TV, since several digital TV channels often share the same carrier frequency thanks to digital modulation techniques.

7.1 Background – Audio watermarking, seen as a digital communication problem

Watermarking an audio signal can be seen as a means of transmitting a sequence of bits ("1"s and "0"s) through a very particular noisy communication channel, whose "noise" contribution is the audio signal itself. With this view in mind, creating an audio watermarking system amounts to designing an emitter and a receiver adapted to the specificities of the channel: the emitter has to insert the watermark in the audio signal in such a way that the watermark cannot be heard, and the receiver must extract this hidden information from the watermarked audio signal[4] (Fig. 7.1).

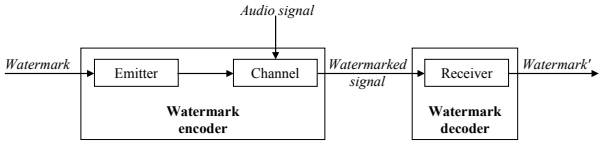

Fig. 7.1 Audio watermarking, seen as a communication problem

The problem therefore looks very similar to that of sending information bits through the internet with an ADSL connection: one has to maximize the bit rate while minimizing the bit error rate, even in the presence of communication noise and distortion[5]. In other words, watermarking directly benefits from fifty years of digital communication theory, after the pioneering work of Claude Shannon.

However, watermarking techniques are very specific in terms of bit rate, error rate, and signal-to-noise ratio. SNR is typically made very low, so as to make the watermark inaudible. This results in very low bit rates (several hundred bits per second at best) and high error rates (typically 10^{-3}), as opposed to the Megabits per second of ADSL modems and the associated low error rates (10^{-6}). In that respect, watermarking techniques are closer to those used for communication between interplanetary satellites and earth. In particular, spread spectrum techniques are used in both cases.

[4] This is often referred to as *blind* watermarking, in which the receiver has no *a priori* information on the audio signal.

[5] Watermark distortion is typically produced by filtering and encoding operations applied to the audio signal while being transmitted, such as MP3 encoding.

Spread spectrum signals will be reviewed in Section 7.1.1. In the next Sections, we will consider the emitter/receiver design issues in a blind (7.1.2), and informed watermarking system (7.1.3), respectively.

7.1.1 Spread spectrum signals

Spread spectrum communications (Peterson *et.al.* 1995) were originally used for military applications[6], given their robustness to narrow-band interference, their secrecy, and their security. As a matter of fact, spread spectrum signals are characterized by two unique features:

1. Their bandwidth is made much wider than the information bandwidth (typically 10 to 100 times for commercial applications; 1000 to 10^6 times for military applications). The spread of energy over a wide band results in lower power spectral density (PSD), which makes spread spectrum signals less likely to interfere with narrowband communications. Conversely, narrow band communications cause little to no interference to spread spectrum systems because their receiver integrates over a very wide bandwidth to recover a spread spectrum signal. As we shall see below, another advantage of such a reduced PSD is that spread spectrum signal PSD can be pushed on purpose below communication noise, thereby making these signals unnoticeable.
2. Some *spreading sequence* (also called *spreading code* or *pseudo-noise*) is used to create the wide-band spread spectrum signal from the information bits to send. This code must be shared by the emitter and the receiver, which makes spread spectrum signals hard to intercept.

The price to pay for these features is a reduced spectral efficiency, defined as the ratio between the bit rate and the bandwidth: the number of bits per second and per Hertz provided by spread spectrum techniques is typically less than 1/5, while "standard" techniques offer a spectral efficiency close to 1 (Massey 1994).

In the nineties, spread spectrum techniques have found a major application in Code Division Multiple Access (CDMA), now used in satellite positioning systems (GPS, hopefully in Galileo), and for wireless digital phone communications systems (UMTS).

Direct sequence spread spectrum (DSSS)

Common spread spectrum systems are of the *time hopping* (also called *direct sequence*) or *frequency hopping* type. In the latter case, as its name

[6] Spread spectrum communications dates back to World War II.

implies, the carrier of a frequency hopping system hops from frequency to frequency over a wide band, in an order defined by the spreading code. In contrast, direct sequence spread spectrum (DSSS) signals are obtained by multiplying each bit-sized period (of duration T_b) of the emitted signal (the watermark signal in our case) by a (unique) spreading sequence, composed of a random sequence of ±1 rectangular pulses (of duration T_c) (Fig. 7.2, left).

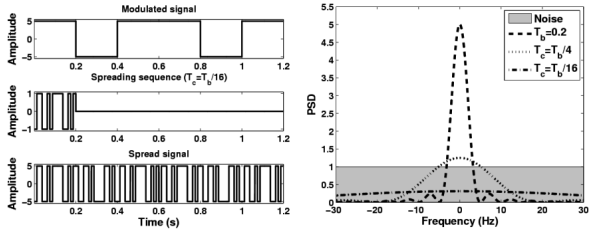

Fig. 7.2 Left: Emitted signal, spreading sequence, and spread signal waveforms, for T_b=0.2s and $T_c=T_b/16$. Right: PSD of the input signal, compared to that of the spread signal for $T_c=T_b/4$ and $T_c=T_b/16$.

It is easy to compute the PSD of such a spread spectrum signal if we assume that the emitted signal is a classical NRZ random binary code composed of rectangular pulses of duration T_b and of amplitude $\pm 1/T_b$. As a matter of fact, the PSD of such a signal is given by:

$$S_{NRZ}(f) = \frac{1}{T_b}\mathrm{sinc}^2(fT_b) \tag{7.1}$$

and since the spread signal is itself an NRZ signal with pulse duration T_c and pulse amplitude $\pm 1/T_b$, its PSD is given by:

$$S_{spread}(f) = \frac{T_c}{T_b{}^2}\mathrm{sinc}^2(fT_c) \tag{7.2}$$

Both PSDs are plotted in Fig. 7.2 (right), for two values of T_c. Clearly, when T_c decreases, the bandwidth of the spread signal increases, and its maximum PSD level decreases accordingly.

As announced above, the resulting spread spectrum signal offers an increased robustness to narrow-band interference and distortion: while a

low-pass filter with cut-off frequency of 5 Hz would almost completely delete the input NRZ signal, its effect on the spread signal would be mild. Spread spectrum techniques also make it possible to hide information in communication noise, as suggested in Fig. 7.2 (right) with a background noise of unitary variance.

7.1.2 Communication channel design

In this Section, we will discuss emitter and receiver design issues, and estimate the corresponding bit error rate.

Emitter

We will use the notations introduced in Fig. 7.3: the audio signal *x(n),* sampled at F_s Hz, is added to a spread spectrum watermark signal *v(n),* obtained by modulating a watermark sequence of M bits b_m (in {0,1}), with a spreading signal *c(n)* composed of N_b samples (and shown in Fig. 7.3 as a vector $\mathbf{c}=[c(0),c(1),\ldots,c(N_b\text{-}1)]^T$).

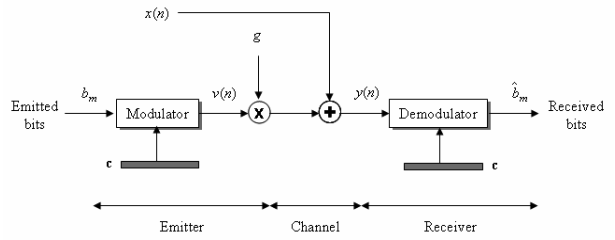

Fig. 7.3 A more detailed view of a watermarking system, seen as a communication system with additive channel noise

Vector **c** is synthesized by a pseudo-random sequence generator, as a Walsh-Hadamard sequence or a Gold sequence with values in {-1,+1}.

Following the time-hopping principle exposed in the previous Section, the spread spectrum signal *v(n)* is given by:

$$v(n)=\sum_{m=0}^{M-1} a_m c(n-mN_b) \tag{7.3}$$

where a_m is a symbol in {-1,+1} given by $a_m=2b_m\text{-}1$. One can also see this signal as a concatenation of vectors:

$$\mathbf{v_m}=a_m\mathbf{c}=\begin{cases}+\mathbf{c} & if\ a_m=1\\ -\mathbf{c} & if\ a_m=-1\end{cases} \tag{7.4}$$

It is finally amplified by some gain *g,* which makes it possible to control the signal-to-noise ratio (SNR) between the watermarked signal *v(n)* and the audio signal *x(n).*

Since one bit is emitted every N_b samples, the bit-rate is simply given by $R=F_s/N_b$ (bits/s), where F_s is the sampling frequency (typically 44.1 kHz).

Receiver

In case of the additive white Gaussian noise (AWGN) channel shown in Fig. 7.3, the signal $y(n)$ available at the receiver is a noisy version of the emitted signal *v(n)*, with channel noise PDF given by $N(0,\sigma_X^2)$.

We will further assume that the received samples *y(n)* are perfectly synchronized with the emitted samples *x(n)*, i.e., that the received samples can be organized in vectors $\underline{y}_m=[y(mN_b), y(mN_b+1), \ldots, y(mN_b+N_b\text{-}1)]^T$ related to vectors $\underline{v}_m$ by:

$$\mathbf{y_m}=g\mathbf{v_m}+\mathbf{x_m}=\begin{cases}+g\mathbf{c}+\mathbf{x_m} & if\ a_m=1\\ -g\mathbf{c}+\mathbf{x_m} & if\ a_m=-1\end{cases} \tag{7.5}$$

where $\mathbf{x_m}=[x(mN_b), x(mN_b+1), \ldots, x(mN_b+N_b\text{-}1)]^T$ is a vector of channel noise samples, and m is the frame index.

Since the modulation operation is assimilated to raw concatenation[7], and thereby produces no symbol interference, symbol detection can be performed on a symbol-by-symbol basis (no Viterbi algorithm is needed). The optimal detector in this case is the *Maximum a Posteriori* (MAP) detector (Proakis 2001)[8], which maximizes the probability of each received bit $\tilde{b}_m$ given the corresponding received vector $\mathbf{y_m}$:

[7] This is the simplest possible case of a memoryless linear modulation scheme.
[8] The same as the one used in Chapter 4 for vowel classification.

$$\tilde{b}_m = \arg\max_{b\in\{0,1\}} P(b \mid \mathbf{y_m}) \tag{7.6}$$[9]

In the next paragraphs, we will show that this is equivalent to estimating $\tilde{b}_m$ from the sign of the normalized scalar product α_m between the received vector $\mathbf{y_m}$ and the spreading code $\mathbf{c}$:

$$\alpha_m = \frac{<\mathbf{y_m},\mathbf{c}>}{\|\mathbf{c}\|^2} = \frac{<\mathbf{y_m},\mathbf{c}>}{N_b} \tag{7.7}$$

As a matter of fact, following the same development as in Chapter 4, Section 4.1, equation (7.6) is equivalent to:

$$\begin{aligned}\tilde{b}_m &= \arg\max_{b\in\{0,1\}} \frac{P(\mathbf{y_m} \mid b)P(b)}{P(\mathbf{y_m})} \\ &= \arg\max_{b\in\{0,1\}} P(\mathbf{y_m} \mid b)P(b)\end{aligned} \tag{7.8}$$

in which $P(\mathbf{y_m})$ is the *a priori* probability of $\mathbf{y_m}$, which does not influence the estimation of $\tilde{b}_m$, and $P(b)$ is the *a priori* probability of each possible value of b. Furthermore, if the symbols have the same probability of occurrence, (7.8) reduces to:

$$\tilde{b}_m = \arg\max_{b\in\{0,1\}} P(\mathbf{y_m} \mid b) \tag{7.9}$$

known as the *maximum likelihood* decision.

In the AWGN channel case, since $\mathbf{x_m}$ is an N_b-dimensional multivariate Gaussian random variable with zero mean and covariance matrix equal to $\sigma_x^2\mathbf{I}$ (where I is the N_b-dimensional unity matrix), equation (7.5) shows that $\mathbf{y_m}$ is also a multivariate Gaussian random variable, with mean equal to $a_m g\mathbf{c}$ and the same covariance matrix as $\mathbf{x_m}$. Hence:

$$P(\mathbf{y_m} \mid b) = \frac{1}{\sqrt{(2\pi\sigma_X^2)^{N_b}}} \exp\left(-\frac{\|\mathbf{y_m} - (2b-1)g\mathbf{c}\|^2}{2\sigma_X^2}\right) \tag{7.10}$$

[9] In the sequel, we use the same shortcut notation for probabilities as in Chapter 4: $P(a|b)$ denotes the probability $P(A=a|B=b)$ that a discrete random variable A is equal to a given the fact that random variable B is equal to b will simply be written, or the probability *density* $p_{A|B=b}$ (a)) when A is a continuous random variable.

It follows that:

$$\begin{aligned}\tilde{b}_m &= \arg\min_{b\in\{0,1\}} \|\mathbf{y_m} - (2b-1)g\mathbf{c}\|^2 \\ &= \arg\min_{b\in\{0,1\}} \left(\|\mathbf{y_m}\|^2 - 2<\mathbf{y_m},(2b-1)g\mathbf{c}> + (2b-1)^2\|\mathbf{c}\|^2\right) \\ &= \arg\max_{b\in\{0,1\}} <\mathbf{y_m},(2b-1)g\mathbf{c}>\end{aligned} \tag{7.11}$$

since $(2b\text{-}1)^2 = 1$ whatever the value of b in $\{0,1\}$. In other words, as announced, $\tilde{b}_m$ is obtained from the sign of the scalar product $<\mathbf{y_m}, \mathbf{c}>$, i.e., from the sign of α_m:

$$\tilde{b}_m = \begin{cases} 0 & if <\mathbf{y_m},\mathbf{c}> \le 0 \\ 1 & if <\mathbf{y_m},\mathbf{c}> > 0 \end{cases} \tag{7.12}$$

Fig. 7.4 shows a synthetic view of (7.9) in case b_m is "1". In the left plot, $\tilde{b}_m$ = "1", while the right plot will wrongly result in $\tilde{b}_m$ = "0".

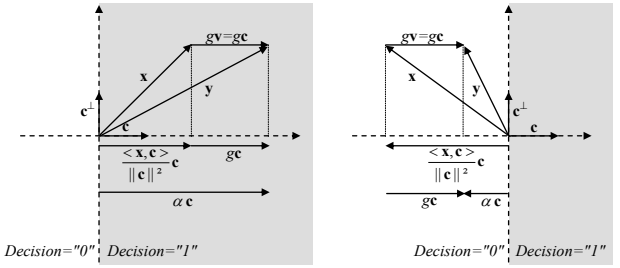

Fig. 7.4 Watermarking of audio vector $\mathbf{x}$ with symbol "1". Watermarked vector $\mathbf{y}$ is obtained by adding $g\mathbf{v}$ (i.e., $g\mathbf{c}$, given the symbol is "1") to $\mathbf{x}$. Symbol "1" will be detected if $\alpha \ge 0$. Left: since $<\mathbf{x},\mathbf{c}>$ is positive, any (positive) value of g will result in $\alpha \ge 0$. Right: since $<\mathbf{x},\mathbf{c}>$ is negative, only a value of $g \ge <\mathbf{x},\mathbf{c}>/\|\mathbf{c}\|^2$ will result in $\alpha \ge 0$ (this is not the case on the figure).

Error rate

The PDF of α_m for g set to 1 and σ_x^2 set to 20 dB is given in Fig. 7.5 for various vector lengths N_b, i.e. for various bit rates R.

Projecting Equation (7.5) on the spread sequence $\mathbf{c}$, the scalar product α_m appears as a one-dimensional Gaussian random variable with mean $(2b_m\text{-}1)g$ and variance $\sigma_x^2 / \|\mathbf{c}\|^2$. The latter is that of the normalized scalar product between the N_b-dimensional multivariate Gaussian random variable $\mathbf{x_m}$ and $\mathbf{c}$. Hence:

$$P(\alpha_m \mid b_m) = \frac{1}{\sqrt{2\pi}\sigma_X / \|\mathbf{c}\|} \exp\left(-\frac{\alpha_m - (2b_m - 1)g}{2\sigma_X^2 / \|\mathbf{c}\|^2} \right) \qquad (7.13)$$

As expected, the probability for α_m to be positive while b_m=0 and the related probability for α_m to be negative while b_m=1 are non-zero, and these values increase with R (i.e., when N_b decreases). The probability of error is given by:

$$P_e = P(b=1)P(\alpha \le 0 \mid b=1) + P(b=0)P(\alpha > 0 \mid b=0) \qquad (7.14)$$

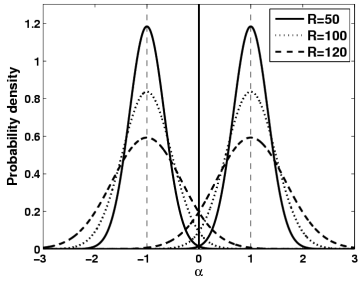

Fig. 7.5 Probability density functions for α when b=0 and when b=1, for various values of the bit rate R (and with F_s=44100, g=1, σ_x^2 =20dB)

When symbols have the same *a priori* probability, this probability of error can be estimated on Fig. 7.5 as the area under the intersection of $P(\alpha_m|b_m=0)$ and $P(\alpha_m|b_m=1)$. It is also possible to compute P_e analytically (Proakis 2001), which leads to:

$$P_e = Q\left(\sqrt{\frac{g^2 \|\mathbf{c}\|^2}{\sigma_X^2}}\right) = Q\left(\sqrt{\frac{F_s}{R}\frac{g^2\sigma_C^2}{\sigma_X^2}}\right) \quad with \quad Q(u) = \int_u^\infty \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{t^2}{2}\right) dt \qquad (7.15)$$

in which we have used $\sigma_C^2 = \|\mathbf{c}\|^2 / N_b$, an estimate of the power of the spreading sequence.

The probability of error is thus finally a function of the watermarking SNR (i.e., the ratio between the power of the emitted signal, $g^2\sigma_C^2$, and that of the audio signal, σ_X^2) and of the bit rate R. Fig. 7.6 shows a plot of $Q\left(\sqrt{u}\right)$ and the resulting plot of $P_e(R)$ for various values of the SNR. It appears that P_e=10^{-3} can be reached with a bit rate of 50 bits/s and a SNR of -20 dB. As we will see below, this SNR approximately corresponds to a watermark just low enough not to be heard (or, consequently, to the kind of SNR introduced by MPEG compression; see Chapter 3).

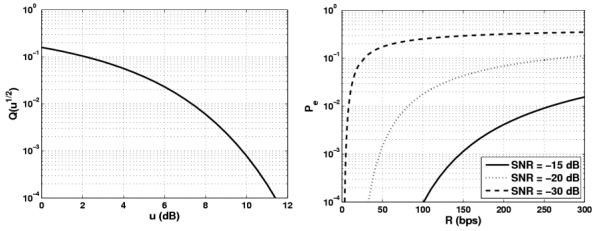

Fig. 7.6 Left: $Q\left(\sqrt{u}\right)$ with u in dB; Right: $P_e(R)$ for various values of the SNR and Fs=44100

7.1.3 Informed watermarking

While it is generally not possible to adapt the encoder to the features of the channel noise[10], such a strategy is perfectly possible in watermarking systems, as the communication noise itself is available at the encoder. In other

[10] In some communications systems, (like in ADSL modems), the communication SNR is estimated at the receiver, and sent back to the emitter through a communication backchannel.

words, watermarking is a good example of a communication channel with side information (Shannon 1958, Costa 1983). This opens the doors to the second generation of watermarking systems: that of *informed watermarking* (Cox *et al.* 1999, Chen and Wornwell 2001, Malvar and Florencio 2003) and its triple objectives: maximal bit-rate, minimal distorsion, maximal robustness. In the next paragraphs we will consider two steps in this direction: gain adaptation, and perceptual filtering.

Gain adjustment for error-free detection

The most obvious way to turn the encoder into an informed encoder is to adapt the gain g as a function of time so as to compensate for the sudden power variations in the audio signal and keep the SNR constant. This can be achieved by estimating σ_X^2 on audio frames (typically 20 ms long, in which music is assumed to be stationary) and imposing g to be proportional to σ_X.

It is also possible, and more interesting, to adjust g to impose *error-free* detection (Malvar and Florencio 2003, LoboGuerrero *et al.* 2003). As we have seen in the previous Section indeed, the probability of error can be made as small as we want, by shifting apart the Gaussians in Fig. 7.5, i.e., by increasing g. It is even possible to obtain error-free detection by adapting the gain to each frame (hence we will denote it as g_m) so as to yield:

$$\alpha_m = \frac{<\mathbf{y_m},\mathbf{c}>}{\|\mathbf{c}\|^2} = a_m g_m + \underbrace{\frac{<\mathbf{x_m},\mathbf{c}>}{\|\mathbf{c}\|^2}}_{\beta_m} = \begin{cases} \geq 0 & if\ a_m = 1 \\ < 0 & if\ a_m = -1 \end{cases} \tag{7.16}$$

As seen in Fig. 7.4, this condition is automatically verified for any (positive) value of g_m, if β_m has the same sign as a_m. In the opposite case, it is verified by imposing $g_m = -\beta_m / a_m$.

In practice though, the transmission channel itself will add noise to the watermarked vector $\mathbf{y_m}$, leading to received vectors equal to $\mathbf{y_m}+\mathbf{p_m}$. Condition (7.16) becomes:

$$\begin{aligned} \alpha_m &= <\mathbf{y_m}+\mathbf{p_m},\mathbf{c}> \\ &= a_m g_m + \underbrace{\frac{<\mathbf{x_m},\mathbf{c}>}{\|\mathbf{c}\|^2}}_{\beta_m} + \underbrace{\frac{<\mathbf{p_m},\mathbf{c}>}{\|\mathbf{c}\|^2}}_{\gamma_m} = \begin{cases} \geq 0 & if\ a_m = 1 \\ < 0 & if\ a_m = -1 \end{cases} \end{aligned} \tag{7.17}$$

Some security margin Δ_g is then required to counterbalance the projection of $\mathbf{p_m}$ on $\mathbf{c}$ (Fig. 7.7):

$$\begin{cases} g_m + \beta_m \geq \Delta_g & if\ a_m = 1 \\ -g_m + \beta_m \leq -\Delta_g & if\ a_m = -1 \end{cases} \tag{7.18}$$

The value of Δ_g is set according to the estimated variance of the transmission channel noise.

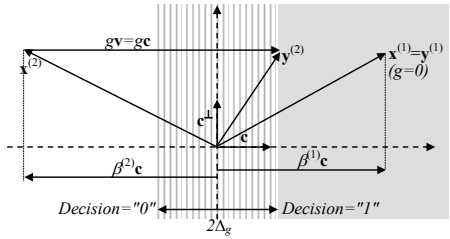

Fig. 7.7 Informed watermarking of audio vector $\mathbf{x}$ with symbol "1". Watermarked vector $\mathbf{y}$ is obtained by adding $g\mathbf{v}$ (i.e., $g\mathbf{c}$, given the symbol is "1") to $\mathbf{x}$, with g chosen such that $\mathbf{y}$ falls in the gray region, where decision will be "1". Two cases are shown: for $\mathbf{x}^{(1)}$, g can be set to zero; for $\mathbf{x}^{(2)}$, g must have a high value. The gray region has been shifted from the white region by a $2\Delta_g$-wide gap, to account for possible transmission channel noise.

The efficiency of this watermarking strategy stems from the fact that the gain g is directly proportional to the correlation between the audio signal $\mathbf{x}$ and the spread sequence $\mathbf{c}$. However, it does not ensure the perceptual transparency of the watermark: in case $\mathbf{x}$ is very much in the opposite direction of $\mathbf{c}$ while the emitted symbol is "1" (as it is the case of $\mathbf{x}^2$ in Fig. 7.7), g must be set to a high value. Chances are in such a case that the watermark may significantly distort the audio signal.

Perceptual shaping filter

A solution to the perceptual transparency requirement is to use properties of the human auditory system to ensure inaudible watermarking. As we have shown in Chapter 3, psychoacoustic modeling (PAM) has established the existence of a masking threshold, assimilated to a PSD $\Phi(f)$ (see Chapter 3, Sections 3.1.3 and 3.1.4) below which the PSD of the watermark $S_W(f)$ must lie for making it inaudible. This threshold is signal-

dependent and must therefore be updated frequently, typically every 20 ms.

Since the efficiency of a watermarking system is a function of the SNR ratio (Fig. 7.6), the best performance is reached when:

$$S_W(f) = \Phi(f) \tag{7.19}$$

This can be achieved, as shown in Fig. 7.8, by replacing the scalar gain *g* by a perceptual shaping filter *G*(*f*), which is conveniently constrained to be an all-pole filter :

$$G(z) = \frac{b_0}{1 + a_1 z^{-1} + a_2 z^{-2} + ... + a_p z^{-p}} \tag{7.20}$$

If the input *v*(*n*) of this filter is assimilated to white noise (with zero mean and unity variance), adjusting the coefficients of this filter so that its output *w*(*n*) will have a given PSD $S_W(f)$ is a typical filter synthesis problem, which has already been examined in Chapter 1. It is based on solving a set of *p* linear equations, the so-called *Yule-Walker* equations (1.5), whose coefficients are the first *p* values of the autocorrelation function $\phi_w(k)$ of *w*(*n*). These values are related to the PSD of *w*(*n*) by the inverse discrete-time Fourier transform formula:

$$\phi_w(k) = \int_{-1/2}^{1/2} \Phi(f) e^{j2\pi fk} df \tag{7.21}$$

which can be approximated by an inverse discrete Fourier transform using *N* samples of Φ(*f*):

$$\phi_w(k) \simeq \frac{1}{N} \sum_{n=0}^{N-1} \Phi\left(\frac{n}{N}\right) e^{j\frac{2\pi}{N}nk} \tag{7.22}$$

The receiver has to be modified accordingly (Fig. 7.8), by inserting an inverse, zero-forcing equalizer $1/G(f)$ before the demodulator (Larbi *et al,* 2004). However, since the audio signal *x*(*n*) used to derive *G*(*f*) is not available at the receiver, an estimate $1/\tilde{G}(f)$ is computed by psychoacoustic modeling of the watermarked signal *y*(*n*). The accuracy of this estimation depends on the robustness of the PAM to the watermark *and* to possible transmission channel noise, such as MPEG compression. This specific constraint on the PAM used in watermarking distinguishes it from the PAMs used in audio compression (such as in MPEG).

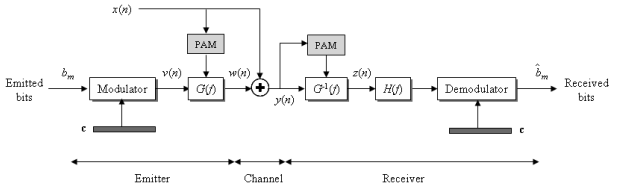

Fig. 7.8 Informed watermarking using a psychoacoustic model (PAM), a perceptual shaping filter *G*(*f*), and a Wiener equalizer *H*(*f*).

Once perceptual shaping has been inverted, detection is achieved as previously, except it is based on estimating the sign of $<\mathbf{z_m},\mathbf{c}>$, where *z(n)* is the output of the inverse filter. Assuming $1/\tilde{G}(f)$ is close enough to $1/G(f)$, $\mathbf{z_m}$ is given by:

$$\mathbf{z_m} = \mathbf{v_m} + \mathbf{r_m} \tag{7.23}$$

where *r*(*n*) is the audio signal *x*(*n*) filtered by $1/\tilde{G}(f)$. Equation (7.23) simply replaces equation (7.5).

Wiener filtering

As we will see in Section 7.2.3, the SNR at the output of the zero-forcing equalizer is very low: the power of the spread spectrum signal *v(n)* is small compared to that of the equalized audio signal *r(n)*. Since the PSDs of *v(n)* and *r(n)* are different, it is possible to filter *z(n)* so as to increase the SNR, by enhancing the spectral components of *z(n)* dominated by *v(n)* and attenuating those dominated by *r(n)*. This can even be done optimally by a symmetric FIR *Wiener filter H(z)* at the output of the zero-forcing equalizer (Fig. 7.8):

$$H(z) = \sum_{i=-p}^{p} h(i) z^{-i} \tag{7.24}$$

The output of the FIR Wiener filter *H(z)* is made maximally similar to *v(n),* in the minimum mean square error (MMSE) sense (Fig. 7.9): its coefficients are computed so as to minimize the power of the error signal

e(n)=z(n)-v(n). It can be shown (Haykin, 1996) that this condition is met if the coefficients *h(i)* of the filter are the solution of the so-called *Wiener-Hopf equations*:

$$\begin{bmatrix} \phi_{zz}(0) & \phi_{zz}(1) & \dots & \phi_{zz}(2p) \\ \phi_{zz}(-1) & \phi_{zz}(0) & \dots & \phi_{zz}(2p-1) \\ \dots & \dots & \dots & \dots \\ \phi_{zz}(-2p) & \phi_{zz}(-2p+1) & \dots & \phi_{zz}(0) \end{bmatrix} \begin{bmatrix} h_{-p} \\ h_{-p+1} \\ \dots \\ h_p \end{bmatrix} = \begin{bmatrix} \phi_{vv}(-p) \\ \phi_{vv}(-p+1) \\ \dots \\ \phi_{vv}(p) \end{bmatrix} \quad (7.25)$$

where $\phi_{zz}(k)$ and $\phi_{vv}(k)$ are the autocorrelation functions of *z(n)* and *v(n),* respectively.

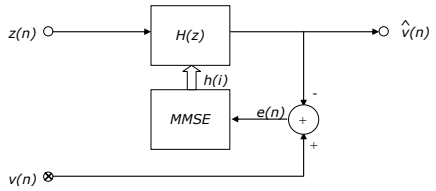

Fig. 7.9 Conceptual view of a Wiener filter [11]

In practice, since *r(n)* is not stationary, the Wiener filter is regularly adapted.

7.2 MATLAB proof of concept: ASP_watermarking.m

In this Section, we develop watermarking systems whose design is based on classical communication systems using spread spectrum modulation. We first examine the implementation of a watermarking system and give a brief overview of its performance in the simple and theoretical configuration where the audio signal is a white Gaussian noise (7.2.1). Then, we show how this system can be adapted to account for the audio signal specificities, while still satisfying the major properties and requirements of watermarking applications (namely inaudibility, correct detection and robustness of the watermark): we extend the initial system by focusing on the correct detection of the watermark (7.2.2), on the inaudibility constraint (7.2.3), and on the robustness of the system to MP3 compression (7.2.4).

[11] In practice, *v(n)* is not available. Nevertheless, only its autocorrelation is required to solve the Wiener-Hopf equations.

7.2.1 Audio watermarking, seen as a digital communication problem

The following digital communication system is the direct implementation of the theoretical results detailed in Section 7.1. The watermark message is embedded in the audio signal at the binary rate `R`=100 bps. To establish the analogy between watermarking system and communication channel, the audio signal is viewed as the channel noise. In this section, it will therefore be modeled as a white Gaussian noise with zero mean and variance `sigma2_x`. `sigma2_x` is computed so that the Signal-to-Noise Ratio (SNR), i.e. the ratio between the watermark and the audio signal powers, is equal to -20 dB.

Emitter/Embedder

Let us first generate a watermark message, and its corresponding bit sequence.

```
message = 'Audio watermarking: this message will be embedded
in an audio signal, using spread spectrum modulation, and
later retrieved from the modulated signal.'

bits = dec2bin(message); % converting ascii to 7-bit string
bits = bits(:)'; % reshaping to a single string of 1's and 0's
bits = double(bits) - 48; % converting to a 0's and 1's
symbols = bits*2-1;  % a (1x1050) array of -1's and +1's
N_bits = length(bits);
```

```
message =

Audio watermarking: this message will be embedded in an audio
signal, using spread spectrum modulation, and later retrieved from
the modulated signal.
```

We then generate the audio signal, the spread spectrum sequence, and the watermarked signal. The latter is obtained by first deriving a symbol sequence (-1's and +1's) from the input bit sequence (0's and +1's), and then by modulating the spread spectrum signal by the symbol sequence (Fig. 7.10, left). This is achieved by concatenating spread spectrum waveforms weighted by the symbol values.

```
% Random sequences generators initialization
rand('seed', 12345);
randn('seed', 12345);

% Sampling frequency |Fs|, binary rate |R|, samples per bit
% |N_samples| and |SNR|
Fs = 44100;
R = 100;
N_samples = round(Fs/R);
SNR_dB = -20;

% Spread waveform: a random sequence of -1's and +1's
spread_waveform = 2*round( rand(N_samples, 1) ) - 1;

% Audio signal: a Gaussian white noise with fixed variance
sigma2_x = 10^(-SNR_dB/10);
audio_signal = sqrt(sigma2_x)*randn(N_bits*N_samples, 1);

% Modulated signal and watermark signal
modulated_signal = zeros(N_bits*N_samples, 1);
for m = 0:N_bits-1
    modulated_signal(m*N_samples+1:m*N_samples+N_samples)= ...
       symbols(m+1)*spread_waveform;
end
watermark_signal = modulated_signal; % no gain applied

% Plotting the baseband signal (i.e., the emitted signal
% corresponding to the input bit sequence) and the watermark
% signal

emitted_signal = ones(N_samples, 1)*symbols;
emitted_signal = emitted_signal(:);

axe_ech = (4*N_samples:7*N_samples+1);
subplot(2,1,1);
plot(axe_ech/Fs, emitted_signal(axe_ech));
subplot(2,1,2);
plot(axe_ech/Fs, watermark_signal(axe_ech));
```

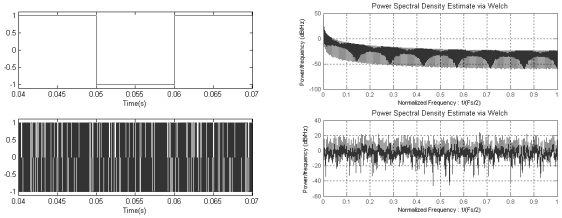

Fig. 7.10 Left: three symbols in {-1,+1} and the resulting spread spectrum waveforms, or watermark signal; Right : the corresponding PSDs

As expected, the power spectral density (PSD) of the spread spectrum sequence (i.e., the watermark signal) is much flatter than that of the baseband signal (i.e., the emitted signal) corresponding to the input bit sequence (Fig. 7.10, right), while their powers are identical: 0dB.


```
power_baseband_dB=10*log10(var(emitted_signal))
power_spread_spectrum_dB=10*log10(var(watermark_signal))

subplot(2,1,1)
pwelch(emitted_signal,[],[],[],2); [12]
subplot(2,1,2)
pwelch(watermark_signal,[],[],[],2);
```

```
power_baseband_dB =  -0.0152
power_spread_spectrum_dB =  5.5639e-006
```

The embedding process finally consists in adding the result to the audio signal, yielding the audio watermarked signal. Plotting the first 100 samples of the watermark, audio, and watermarked signals shows that the watermarked signal is only slightly different from the audio signal, given the SNR we have imposed (Fig. 7.11).

```
watermarked_signal = watermark_signal + audio_signal;

subplot(3,1,1); plot((1:100)/Fs,watermark_signal(1:100),'k-');
subplot(3,1,2);plot((1:100)/Fs,audio_signal(1:100));
subplot(3,1,3);plot((1:100)/Fs,watermarked_signal(1:100),'r');
```

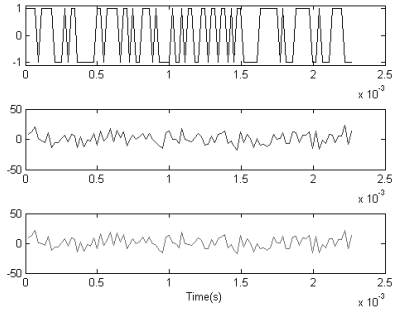

Fig. 7.11 Zoom on 100 samples. Top: watermark signal; Center: audio signal; Bottom: watermarked signal

[12] Notice that we claim F_s=2, so as to make **pwelch** return |FFT²/*N*|, in which the variance of a white noise is directly readable. See the appendix in **ASP_audio_cd.m** for more details.

Receiver

The watermark receiver is a correlation demodulator. It computes the normalized scalar product `alpha` between frames of the received signal and the spread spectrum waveform, and decides on the received bits, based on the sign of `alpha`. One can see on the plot (for bits 81 to 130) that the watermark sometimes imposes a wrong sign to `alpha`, leading to erroneous bit detection (Fig. 7.12).

```
alpha = zeros(N_bits, 1);
received_bits = zeros(1,N_bits);
for m = 0:N_bits-1
    alpha(m+1) =
(watermarked_signal(m*N_samples+1:m*N_samples+N_samples)' ...
        * spread_waveform)/N_samples;
    if alpha(m+1) <= 0
        received_bits(m+1) = 0;
    else
        received_bits(m+1) = 1;
    end
end

% Plotting the input bits, alpha, and the received bits.
range=(81:130);
subplot(3,1,1);
stairs(range, bits(range));
subplot(3,1,2);
stairs(range, alpha(range));
subplot(3,1,3);
stairs(range, received_bits(range));
```

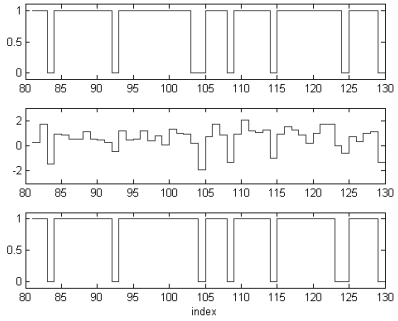

Fig. 7.12 Zoom on bits 80 to 130. Top: input bits; Center: **`alpha`**; Bottom: received bits

The performance of the receiver can be estimated by computing the Bit Error Rate (BER) and by decoding the received message. As can be observed on the message, a bit error rate of 0.02 is disastrous in terms of message understandability.

```
number_of_erroneous_bits= sum(bits ~= received_bits)
total_number_of_bits=N_bits
BER = number_of_erroneous_bits/N_bits
received_chars = reshape(received_bits(1:N_bits),N_bits/7, 7);
received_message = char( bin2dec(num2str(received_chars)))'
```

```
number_of_erroneous_bits =  21
total_number_of_bits =  1050
BER =    0.0200

received_message =Cudio □ateriarking: this messaGe will be embeeded
iN An audio signal( ushnG spread spectrum modulation1 and later
qetrimve$ from 4he mo uleteD!siGnal.
```

System performance overview

A good overview of the system performance can be drawn from the statistical observation of `alpha` values, through a histogram plot. As expected, a bimodal distribution is found, with non zero overlap (Fig. 7.13). Notice that the histogram only gives a rough idea of the underlying distribution, since the number of emitted bits is small.

```
hist(alpha,50);
```

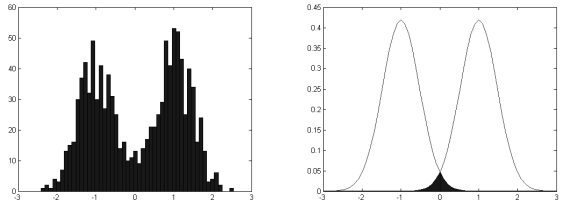

Fig. 7.13 Left: histogram of `alpha` values. Right: theoretical probability density function

In our particular configuration, the spread waveform is a realization of a random sequence with values in {+1,-1}, and the audio-noise vectors are modeled by `N_samples` independent Gaussian random variables with zero

mean and variance sigma2_x. In this case, it can be shown that the Probability Density Function (PDF) of alpha is the average of two normal distributions with mean a*g (a=+1 or -1) and variance sigma2_x/N_samples. Plotting the theoretical normal distribution shows a better view of the overlap area observed on the histogram. In particular, the area of the overlap between the two Gaussian modes corresponds to the theoretical BER (the exact expression of this BER is given in Section 7.1).

```
alpha_range = -3:.1:3;
gauss0 = 1/2*exp(-((alpha_range-1).^2)/ ...
   (2*sigma2_x/N_samples)) / sqrt(2*pi*sigma2_x/N_samples);
gauss1 = 1/2*exp(-((alpha_range+1).^2)/ ...
   (2*sigma2_x/N_samples)) / sqrt(2*pi*sigma2_x/N_samples);
theoretical_PDF=gauss0+gauss1;

plot(alpha_range, gauss0); hold on
plot(alpha_range, gauss1);

ind0 = find(alpha_range<=0);
ind1 = find(alpha_range>=0);
ae1 = area(alpha_range(ind1), gauss1(ind1), ...
  'FaceColor', [0.39 0.47 0.64]);   hold on;
ae2 = area(alpha_range(ind0), gauss0(ind0), ...
  'FaceColor', [0.39 0.47 0.64]);

theoretical_BER1=2*sum(gauss1(ind1))*(alpha_range(2)-...
   alpha_range(1))
```

```
theoretical_BER1 =  0.0228
```

7.2.2 Informed watermarking with error-free detection

The previous system did not take the audio signal specificities into account (since audio was modeled as white Gaussian noise). We now examine how to modify the system design to reach one major requirement for a watermarking application: that of detecting the watermark message without error. For this purpose, the watermark gain (also called *embedding strength*) has to be adjusted to the local audio variations. This is referred to as informed watermarking.

From now on, the audio signal will be a violin signal sampled at F_s=44.100 Hz, of which only the first N_bits*N_samples samples will be watermarked. This signal is normalized in [-1,+1].

```
audio_signal = wavread('violin.wav', [1 N_bits*N_samples]);
```

Informed emitter

To reach a zero BER, the watermark gain is adapted so that the correlation between the watermarked audio signal and the spread sequence is at least

equal to some security margin Delta_g (if am = +1) and -Delta_g (if am = -1). Delta_g sets up a robustness margin against additive perturbation (such as the additive noise introduced by MPEG compression of the watermarked audio signal). In the following MATLAB implementation, Delta_g is empirically set to 0.005.

```
Delta_g = 0.005;
for m=0:N_bits-1
  beta = audio_signal(m*N_samples+1:m*N_samples+N_samples)'*...
        spread_waveform /N_samples;
  if symbols(m+1) == 1
        if  beta >= Delta_g
            gain(m+1) = 0;
        else
            gain(m+1) = Delta_g - beta;
        end
  else % if symbols(m+1) == -1
        if  beta <= -Delta_g
            gain(m+1) = 0;
        else
            gain(m+1) = Delta_g + beta;
        end
  end
  watermark_signal(m*N_samples+1:m*N_samples+N_samples,1)= ...

gain(m+1)*modulated_signal(m*N_samples+1:m*N_samples+N_samples);
end

watermarked_signal = watermark_signal + audio_signal;
```

Plotting the gain for one second of signal, together with the resulting watermark signal and the audio signal, shows that the encoder has to make important adjustments as a function of the audio signal (Fig. 7.14).

```
subplot(2,1,1);
plot((0:99)*N_samples/Fs, gain(1:100));
subplot(2,1,2);
plot((0:Fs-1)/Fs, watermark_signal(1:Fs));
```

The watermark, though, is small compared to the audio signal (Fig. 7.15, left).

```
plot((0:Fs-1)/Fs, audio_signal(1:Fs)); hold on;
plot((0:Fs-1)/Fs, watermark_signal(1:Fs), 'r-');
```

Plotting again a few samples of the watermark, audio, and watermarked signals shows that the watermarked signal sometimes differs significantly from the audio signal (Fig. 7.15, right). This is confirmed by a listening test: the watermark is audible.

```
range=(34739:34938);
subplot(3,1,1); plot(range/Fs, watermark_signal(range), 'k-');
subplot(3,1,2); plot(range/Fs, audio_signal(range));
subplot(3,1,3); plot(range/Fs, watermarked_signal(range),'r');
```

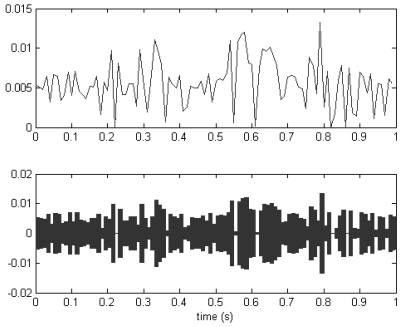

Fig. 7.14 Top: watermark gain; Bottom: watermark signal

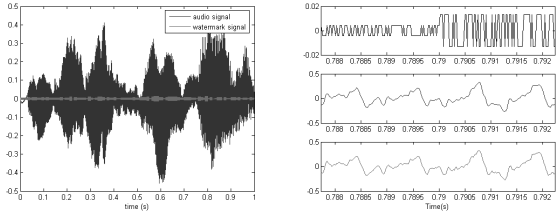

Fig. 7.15 Left: a comparison between the audio and watermark signals; Right: zoom on the watermark signal (top), the audio signal (center), and the watermarked signal (bottom)

Receiver

Using the same correlation demodulator as in Section 7.1, we conclude that the transmission is now effectively error-free (Fig. 7.16), as confirmed by the resulting BER.

```
number_of_erroneous_bits= sum(bits ~= received_bits)
BER = number_of_erroneous_bits/total_number_of_bits
```



```
received_chars = reshape(received_bits(1:N_bits),N_bits/7, 7);
received_message = char( bin2dec(num2str(received_chars)))'
```

```
number_of_erroneous_bits =  0
BER =  0

received_message = Audio watermarking: this message will be embedded
in an audio signal, using spread spectrum modulation, and later
retrieved from the modulated signal.
```

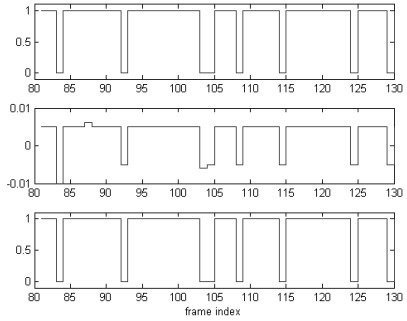

Fig. 7.16 Zoom on bits 80 to 130. Top: input bits; Center: `alpha`; Bottom: received bits

7.2.3 Informed watermarking made inaudible

The informed watermarking system exposed in Section 7.2.2 proves that prior knowledge of the audio signal can be efficiently used in the watermarking process: error-free transmission is reached thanks to an adaptive embedding gain. However, the resulting watermark is audible, since no perceptual condition is imposed on the embedding gain. In this Section, we examine how psychoacoustics can be put to profit to ensure the inaudibility constraint. A psychoacoustic model is used, which provides a signal-dependent masking threshold used as an upper bound for the PSD of the watermark. The watermark gain is therefore replaced by an all-pole perceptual shaping filter and the reception process is composed of a zero-forcing equalizer, followed by a linear-phase Wiener filter.

Emitter, based on perceptual shaping filtering

The perceptual shaping filter is an auto-regressive (all-pole) filter with 50 coefficients `ai` and gain `b0`. It is designed so that the PSD of the watermark (obtained by filtering the `modulated_signal`) equals the `masking_threshold`. The coefficients of the filter are obtained as in Chapter 1, via the Levinson algorithm. Both the masking threshold and the shaping filter have to be updated each time the statistical properties of the `audio_signal` change (here each `N_windows`=512 samples).

Let us first compute the masking threshold and apply the associated shaping filter to one audio frame (the 11-th frame, for instance: Fig. 7.17 left).

```
N_coef = 50;
N_samples_PAM = 512;
PAM_frame = (10*N_samples_PAM+1 :
10*N_samples_PAM+N_samples_PAM);

plot(PAM_frame/Fs,audio_signal(PAM_frame) );
```

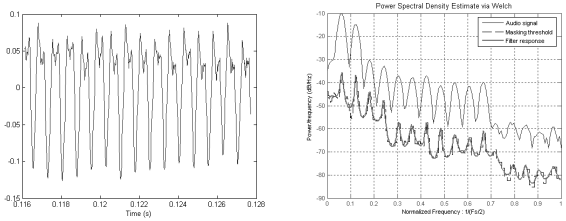

Fig. 7.17 Left: audio signal of frame #11; Right: PSD of frame #11, and the associated masking threshold and shaping filter response

Comparing the periodogram of the audio signal, the masking threshold and the frequency response of the filter shows that the filter closely matches the masking threshold (Fig. 7.17 right). Even the absolute amplitude level of the filter is the same as that of the masking threshold. As a matter of fact, since the audio signal is normalized in [-1,+1], its nominal PSD level is 0 dB.

MATLAB functions involved:

- `masking_threshold=psychoacoustical_model(audio_signal)`

returns the masking threshold deduced from a psychoacoustical analysis of the audio vector. This implementation is derived from the psycho-acoustic model #1 used in MPEG-1 Audio (see ISO/CEI norm 11172-3:1993 (F), pp. 122-128 or MATLAB function MPEG1_psycho_acoustic_model1.m from Chapter 3). It is based on the same principles as those used in the MPEG model, but it is further adapted here so as to make it robust to additive noise (which is a specific constraint of watermarking and is not found in MPEG).

- `[b0,ai]=shaping_filter_design(desired_frequency_response_dB,N_coef)` computes the coefficients of an auto-regressive filter:

$$G(z) = \frac{b0}{ai(1) + ai(2)z^{-1} + \ldots + ai(N_coef)z^{-N_coef+1}}$$

(with ai(1)=1) from the modulus of its `desired_frequency_response` (in dB) and the order `N_coef`. The coefficients are obtained as follows: if zero-mean and unity variance noise is provided at the input of the filter, the PSD of its ouput is given by `desired_frequency_response`. Setting the coefficients so that this PSD best matches `desired_frequency_response` is thus obtained by applying the Levinson algorithm to the autocorrelation coefficients of the output signal (computed itself from the IFFT of the `desired_frequency_response`).

```
masking_threshold = ...
   psychoacoustical_model(audio_signal(PAM_frame));
shaping_filter_response = masking_threshold;
[b0,ai]=shaping_filter_design(shaping_filter_response,N_coef);

% Plotting results. For details on how we use |pwelch|, see
% the appendix of ASP_audio_cd.m, the companion script file of
% Chapter 2.

pwelch(audio_signal(PAM_frame),[],[],[],2);
hold on;
[H,W]=freqz(b0,ai,256);
stairs(W/pi,masking_threshold,'k--');
plot(W/pi,20*log10(abs(H)),'r','linewidth',2);
hold off;
```

Applying this perceptual shaping procedure to the whole watermark signal requires to process the audio signal block per block. Notice that the filtering continuity from one block to another is ensured by the `state` vector, which stores the final state of the filter at the end of one block and applies it as initial conditions for the next block.

```
state = zeros(N_coef, 1);
for m = 0:fix(N_bits*N_samples/N_samples_PAM)-1
    PAM_frame = ...
        (m*N_samples_PAM+1:m*N_samples_PAM+N_samples_PAM);

    % Shaping filter design
    masking_threshold = psychoacoustical_model(...
       audio_signal(PAM_frame) );
    shaping_filter_response = masking_threshold;
    [b0, ai]=shaping_filter_design(shaping_filter_response,...
       N_coef);

    % Filtering stage
    [watermark_signal(PAM_frame,1), state] = ...
        filter(b0, ai, modulated_signal(PAM_frame), state);
end

% Filtering the last, incomplete frame in |watermark_signal|
PAM_frame = ...
    (m*N_samples_PAM+N_samples_PAM+1:N_bits*N_samples);
watermark_signal(PAM_frame,1) = ...
    filter(b0, ai, modulated_signal(PAM_frame), state);

plot((0:Fs-1)/Fs, audio_signal(1:Fs));
hold on;
plot((0:Fs-1)/Fs, watermark_signal(1:Fs), 'r-');
```

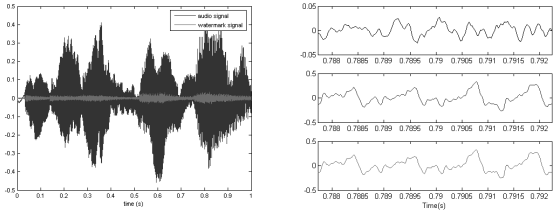

Fig. 7.18 Left: audio signal and watermark signal; Right: a few samples of the audio, watermark, and watermarked signals

The watermarked audio signal is still obtained by adding the audio signal and the watermark signal (Fig. 7.18 left). Plotting again few samples of the watermark, audio, and watermarked signals shows that the watermark signal has now been filtered (Fig. 7.18 right). As a result, the watermark is quite inaudible, as confirmed by a listening test. Its level, though, is similar to (if not higher than) that of the watermark signal in Section 7.2.2.

```
watermarked_signal = watermark_signal + audio_signal;

range=(34739:34938);
subplot(3,1,1); plot(range/Fs, watermark_signal(range), 'k');
subplot(3,1,2); plot(range/Fs, audio_signal(range));
subplot(3,1,3); plot(range/Fs, watermarked_signal(range),'r');
```


Receiver, based on zero-forcing equalization and detection

The zero-forcing equalization aims at reversing the watermark perceptual shaping, before extracting the embedded message. The shaping filter with frequency response `shaping_filter_response` is therefore recomputed from the audio `watermarked_signal`, since the original `audio_signal` is not available at the receiver. This process follows the same block processing as the watermark synthesis, but involves a moving average filtering stage: filtering the `watermarked_signal` by the filter whose frequency response is the inverse of the `shaping_filter_response` yields the filtered received signal denoted by `equalized_signal`.

Plotting the PSD of the `equalized_signal` shows its flat spectral envelope, hence its name (Fig. 7.19).

```
state = zeros(N_coef, 1);
for m = 0:fix(N_bits*N_samples/N_samples_PAM)-1

    PAM_frame = ...
        m*N_samples_PAM+1:m*N_samples_PAM+N_samples_PAM;

    % Shaping filter design, based the watermarked signal
    masking_threshold =
        psychoacoustical_model(watermarked_signal(PAM_frame));
    shaping_filter_response = masking_threshold;
    [b0, ai]=shaping_filter_design(shaping_filter_response,...
       N_coef);

    % Filtering stage
    [equalized_signal(PAM_frame), state] = ...
        filter(ai./b0,1,watermarked_signal(PAM_frame),state);

    if m==10
    % Showing the frequency response of the equalizer and PSD
    % of the |equalized_signal|, for frame 10.
        pwelch(equalized_signal(PAM_frame),[],[],[],2);
        hold on;
        [H,W]=freqz(ai./b0,1,256);
        plot(W/pi,20*log10(abs(H)),'r','linewidth',2);
    end;

end
```

The `equalized_signal` is theoretically the sum of the original watermark (i.e. the `modulated_signal`, which has values in {-1,+1}) and the `equalized_audio_signal`, which is itself the original `audio_signal` filtered by the Zero-Forcing equalizer. In other words, from the receiver point of view, everything looks as if the watermark had been added to the `equalized_audio_signal` rather than to the `audio_signal` itself.

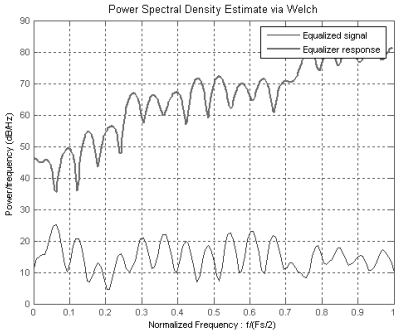

Fig. 7.19 Frequency response of the equalizer at frame #10, and PSD of the resulting equalized watermarked signal

Although the `equalized_audio_signal` is not available to the receiver, it is interesting to compute it and compare it to the original `audio_signal` (Fig. 7.20, left). Notice that the level of the `equalized_audio_signal` is much higher than that of the original `audio_signal`, mostly because its HF content has been enhanced by the equalizer.

```
equalized_audio_signal = zeros(N_bits*N_samples, 1);
state = zeros(N_coef, 1);

for m = 0:fix(N_bits*N_samples/N_samples_PAM)-1

    PAM_frame = ...
       m*N_samples_PAM+1:m*N_samples_PAM+N_samples_PAM;

    % Shaping filter design, based the watermarked signal
    masking_threshold = ...
       psychoacoustical_model(watermarked_signal(PAM_frame));
    shaping_filter_response = masking_threshold;
    [b0,ai]=shaping_filter_design(shaping_filter_response, ...
       N_coef);

    % Filtering stage
    [equalized_audio_signal(PAM_frame), state] = ...
        filter(ai./b0, 1, audio_signal(PAM_frame), state);
end

% Plotting 200 samples of the |equalized_audio_signal|, and
% |modulated_signal|
range=(34739:34938);
subplot(2,1,1); plot(range/Fs, equalized_audio_signal(range));
subplot(2,1,2); plot(range/Fs, modulated_signal(range));
```


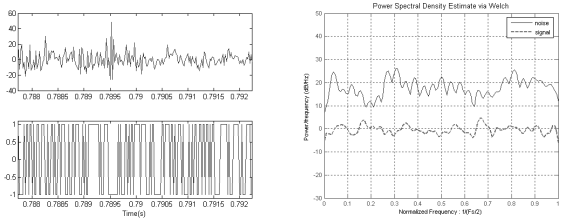

Fig. 7.20 Left: zoom on the equalized audio signal and on the modulated signal; Right: corresponding PSDs

It is also possible to estimate the SNR by visual inspection of the PSDs of the equalized audio signal and the modulated signal (Fig. 7.20, right) or to compute it from the samples of these signals.

```
range=(34398:34938);
pwelch(equalized_audio_signal(range),[],[],[],2);
[h,w]=pwelch(modulated_signal(range),[],[],[],2);
hold on;
plot(w,10*log10(h),'--r ');

snr_equalized=10*log10(var(modulated_signal(range))/...
   var(equalized_audio_signal(range)))
```

```
snr_equalized =  -19.0610
```

We can now proceed to the watermark extraction by applying the correlation demodulator to the `equalized_signal`, and compute the BER. As expected, the obtained BER is quite high, since the SNR is low: the level of the `equalized_audio_signal` is high compared to that of the embedded `modulated_signal` (Fig. 7.20).

```
for m = 0:N_bits-1
    alpha(m+1) = (equalized_signal(m*N_samples+1: ...
        m*N_samples+N_samples)' * spread_waveform)/N_samples;
    if alpha(m+1) <= 0
        received_bits(m+1) = 0;
    else
        received_bits(m+1) = 1;
    end
end

% Plotting the input bits, alpha, and the received bits.
range=(81:130);
subplot(3,1,1);
stairs(range, bits(range));
subplot(3,1,2);
stairs(range, alpha(range));
subplot(3,1,3);
```

```
stairs(range, received_bits(range));

number_of_erroneous_bits= sum(bits ~= received_bits)
total_number_of_bits=N_bits
BER = number_of_erroneous_bits/total_number_of_bits

received_chars = reshape(received_bits(1:N_bits),N_bits/7, 7);
received_message = char( bin2dec(num2str(received_chars)))'
```

```
number_of_erroneous_bits =    30
total_number_of_bits =   1050
BER =  0.0286

received_message = audio watdrmarkyng:Oplks mussage will be embedded
in e~Oaudik siwnql, }wi~g spread SpeCtruM(modul tion, and
later0retrieved from the modudated qienal.
```

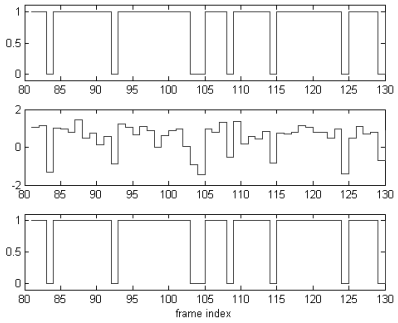

Fig. 7.21 Zoom on bits 80 to 130. Top: input bits; Center: **alpha**; Bottom: received bits

Wiener filtering

The Wiener filtering stage aims at enhancing the SNR between the `modulated_signal` and the `equalized_audio_signal`. This is achieved here by filtering the `equalized_signal` by a symmetric (non causal) FIR filter with `N_coef`=50 coefficients. Its coefficients `hi` are computed so that the output of the filter (when fed with the `equalized_signal`) becomes maximally similar to the `modulated_signal` in the RMSE sense. They are the solution of the so-called Wiener-Hopf equations:

```
hi= equalized_signal_cov_mat-1 * modulated_signal_autocor_vect
```


where `equalized_signal_cov_mat` and `modulated_signal_autocor_vect` are the covariance matrix of the `equalized_signal` and the autocorrelation vector of the `modulated_signal`, respectively.

Since the `modulated_signal` is unknown from the receiver, its autocorrelation is estimated from an arbitrary modulated signal and can be computed once. To make it simple, we use our previously computed `modulated_signal` here. On the contrary, the covariance matrix of the equalized signal, and therefore the coefficients `hi`, has to be updated each time the properties of `estimated_signal` change, that is every `N_samples_PAM`=512 samples. Wiener filtering is then carried out, by computing the covariance matrix of the `equalized_signal` and the impulse response of the Wiener filter for each PAM frame.

```
modulated_signal_autocor_vect=...
   xcorr(modulated_signal,N_coef,'biased');

state = zeros(2*N_coef, 1);

for m = 0:fix(N_bits*N_samples/N_samples_PAM)-1
    PAM_frame = ...
       (m*N_samples_PAM+1:m*N_samples_PAM+N_samples_PAM);

    % Estimating the covariance matrix of |equalized signal|,
    % as a Toeplitz matrix with first row given by the
    % autocorrelation vector of the signal.
    equalized_signal_autocor_vect= ...
        xcorr(equalized_signal(PAM_frame),2*N_coef,'biased');
    equalized_signal_cov_mat= ...
      toeplitz(equalized_signal_autocor_vect(2*N_coef+1:end));

    % Estimating the impulse response of the Wiener filter as
    % the solution of the Wiener-Hopf equations.
    hi=equalized_signal_cov_mat\modulated_signal_autocor_vect;

    % Filtering stage
    [Wiener_output_signal(PAM_frame,1), state] = ...
        filter(hi, 1, equalized_signal(PAM_frame), state);
    power = ...
       norm(Wiener_output_signal(PAM_frame))^2/N_samples_PAM;
    if (power ~= 0)
        Wiener_output_signal(PAM_frame) = ...
            Wiener_output_signal(PAM_frame)/ sqrt(power);
    end

    % Saving the Wiener filter for frame #78
    if m==78
        hi_78=hi/sqrt(power);
    end;

end

% Since the Wiener filter is non-causal (with |N_coeff|=50
% coefficients for the non-causal part), the resulting
% |Wiener_output_signal| is delayed with |N_coef| samples.
Wiener_output_signal = ...
   [Wiener_output_signal(N_coef+1:end);  zeros(N_coef, 1)];
```

It is interesting to check how the equalized audio signal and the modulated signal have been modified by the Wiener filter (Fig. 7.22 left) and how their PSDs have evolved (Fig. 7.22 right). Obviously, the Wiener filter has enhanced the frequency bands dominated by the modulated signal in Fig. 7.20 (right), thereby increasing the SNR by more than 5 dBs.

```
range=(34398:34938);
Wiener_output_audio=filter(hi,1,equalized_audio_signal(range));
Wiener_output_modulated=filter(hi,1,modulated_signal(range));
range=(34739:34938);
subplot(2,1,1); plot(range/Fs,Wiener_output_audio(341:540));
subplot(2,1,2); plot(range/Fs,Wiener_output_modulated(341:540));

pwelch(Wiener_output_audio,[],[],[],2);
[h,w]=pwelch(Wiener_output_modulated,[],[],[],2);
hold on;
plot(w,10*log10(h),'--r ', 'linewidth',2);

snr_Wiener=10*log10(var(Wiener_output_modulated(341:530)) ...
      /var(Wiener_output_audio(341:530)))
```

```
snr_Wiener =  -19.2023
```

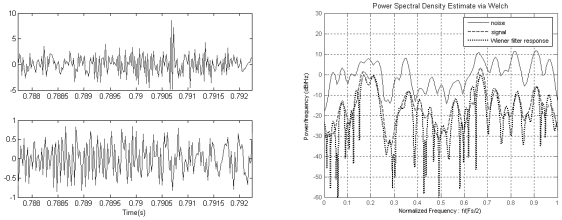

Fig. 7.22 Left: zoom on the equalized audio signal and on the modulated signal, passed through the Wiener filter; Right: corresponding PSDs, and frequency response of the Wiener filter.

We can finally apply the correlation demodulator to the estimated modulated signal. The resulting BER is lower (Fig. 7.23), thanks to Wiener filtering: a BER of the order of 10^{-3} is reached, while the watermark is now inaudible.

```
for m = 0:N_bits-1
    alpha(m+1) = (Wiener_output_signal(m*N_samples+1: ...
        m*N_samples+N_samples)'*spread_waveform )/N_samples;
    if alpha(m+1) <= 0
        received_bits(m+1) = 0;
    else
        received_bits(m+1) = 1;
    end
end

number_of_erroneous_bits= sum(bits ~= received_bits)
total_number_of_bits=N_bits
BER = number_of_erroneous_bits/total_number_of_bits

received_chars = reshape(received_bits(1:N_bits),N_bits/7, 7);
received_message = char( bin2dec(num2str(received_chars)))'
```

```
number_of_erroneous_bits =  2
total_number_of_bits =   1050
BER =   0.0019

received_message = Audio watermarking: this message will$be embedded
in an audIo signal, using spread spectrum modulation, and later
retrieved from the modulated signal.
```


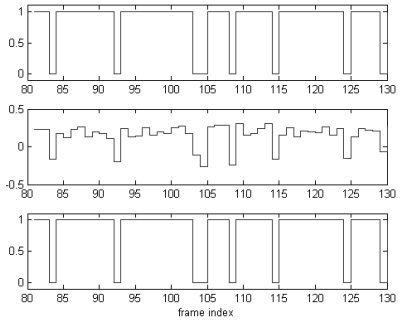

Fig. 7.23 Zoom on bits 80 to 130. Top: input bits; Center: `alpha`; Bottom: received bits

Robustness to MPEG compression

We finally focus on the robustness of our system to MPEG compression. Using the MATLAB functions developed in Chapter 3, we can easily apply an mp3 coding/decoding operation to the audio watermarked signal, yielding the distorted watermarked audio signal `compressed_watermarked_signal`.

MATLAB function involved:

- `output_signal = codec_mp3( input_signal, Fs)` returns the signal `output_signal` resulting from a mp3 coding/decoding operation of the input signal `input_signal` sampled at the `Fs` frequency sampling (see chapter 3).

```
compressed_watermarked_signal=mp3_codec(watermarked_signal,Fs);
```

We then pass the compressed signal through the Zero-Forcing equalizer and the Wiener filter, as before, and compute the resulting BER [13] and received message.

```
BER =   0.1048
received_message = )-%im$gate□mapcHng□"txIr }!UsagM
wiLm&bD$embed`de□hN"cn yUdho skenan, Usino`sppdaD sp}btrum
mOd5latio~( a^E□Lat%□02gtrievgd fPoMda*u0m/$uxetet4cggna}
```

Unfortunately, the obtained BER has significantly increased: although it has been shown above that the psychoacoustic model we use in our perceptual watermarking system is robust to the watermark (i.e., the masking threshold does not change significantly when the watermark is added to the audio signal), it is clearly not robust *yet* to MPEG compression.

To show this, let us compare the PSD of the watermarked signal and the associated masking threshold, before and after the MPEG compression, on the 11th frame for instance (Fig. 7.24). Clearly, the masking threshold is very sensitive to MPEG compression for frequencies over 11 kHz. As a result, the perceptual shaping filter and the equalizer no longer cancel each other, hence our high BER.

```
PAM_frame = (10*N_samples_PAM+1 : ...
   10*N_samples_PAM+N_samples_PAM);

subplot(1,2,1)
pwelch(watermarked_signal(PAM_frame),[],[],[],2);
hold on;
masking_threshold = psychoacoustical_model(...
   watermarked_signal(PAM_frame) );
stairs((1:N_samples_PAM/2)*2/N_samples_PAM,masking_threshold,...
   'r','linewidth',2)

subplot(1,2,2)
pwelch(compressed_watermarked_signal(PAM_frame),[],[],[],2);
hold on;
masking_threshold = psychoacoustical_model(...
   compressed_watermarked_signal(PAM_frame) );
stairs((1:N_samples_PAM/2)*2/N_samples_PAM,masking_threshold,...
   'r','linewidth',2)
```

[13] The corresponding MATLAB code can be found in the `ASP_watermarking.m` file. We do not repeat it here.

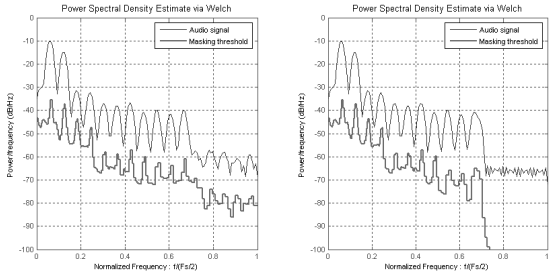

Fig. 7.24 PSD of frame #11, and the associated masking threshold. Left: before MPEG compression; Right: after MPEG compression

7.2.4 Informed watermarking, robust to MPEG compression

In order to improve the robustness of the system exposed in the previous Section, the watermark information should obviously be spread in the [0 Hz, 11 kHz] frequency range of the audio signal. This can be achieved with a low pass filter with cutoff frequency set to 11 kHz (Baras *et al.* 2006). As we shall see below, this filter will interfere with our watermarking system in three stages.

Designing the low pass filter

First, we design a symmetric FIR low pass filter with `Fc=11` kHz cutoff frequency, using the Parks-McClellan algorithm. Such a symmetric filter (Fig. 7.25, left) will have linear phase, and therefore will not change the shape of the modulated signal more than required. We set the order of the filter to `N_coef`=50. Notice that this filter is non-causal and introduces a `N_coef`/2-samples delay.

```
Fc = 11000;
low_pass_filter = firpm(N_coef, [0 Fc-1000 Fc+1000 Fs/2]*2/Fs,
   [1 1 1E-9 1E-9]);
%low_pass_filter = ...
   low_pass_filter/sqrt(sum(low_pass_filter)^2));

% Let us plot the impulse response of this filter.
plot(low_pass_filter);
```

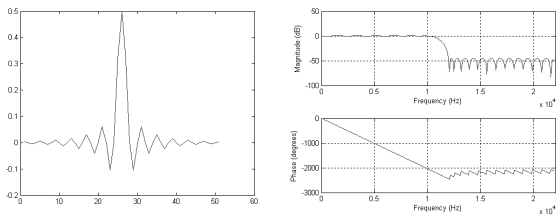

Fig. 7.25 Left: Impulse response of a symmetric FIR low-pass filter; Right: Its frequency response

It is easy to check that its frequency response matches our requirements (Fig. 7.25, right).

```
freqz(low_pass_filter,1,256);
```

Modifying the emitter

We first use this low-pass filter to create a spread sequence with spectral content restricted to the [0 Hz, 11 kHz] range. The modulated signal can then be designed as previously.

```
spread_waveform = ...
    filter(low_pass_filter,1,2*round(rand(N_samples+N_coef,1))-1);
% Getting rid of the transient
spread_waveform = spread_waveform(N_coef+1:end);
% Making sure the norm of the spread waveform is set to
% sqrt(N_samples), as in the previous Sections.
spread_waveform = spread_waveform/ ...
    sqrt(norm(spread_waveform).^2/N_samples);

for m = 0:N_bits-1
    modulated_signal(m*N_samples+1:m*N_samples+N_samples) = ...
        symbols(m+1)*spread_waveform;
end
```

To prevent the shaping filter from amplifying the residual frequency component of the modulated signal over 11 kHz, the `shaping_filter_response`, initially chosen to be equal to the masking threshold, is now designed to be equal to the masking threshold in the [0, 11] kHz frequency band and to zero anywhere else.


```
state = zeros(N_coef, 1);
N_cutoff = ceil(Fc/(Fs/2)*(N_samples_PAM/2));

for m = 0:fix(N_bits*N_samples/N_samples_PAM)-1
    PAM_frame = ...
       (m*N_samples_PAM+1:m*N_samples_PAM+N_samples_PAM);

    % Shaping filter design
    masking_threshold = ...
        psychoacoustical_model( audio_signal(PAM_frame) );
    shaping_filter_response = [masking_threshold(1:N_cutoff); ...
        -100*ones(N_samples_PAM/2-N_cutoff, 1) ];
   [b0,ai]=shaping_filter_design(shaping_filter_response,N_coef);

    % Filtering stage
    [watermark_signal(PAM_frame), state] = ...
        filter(b0, ai, modulated_signal(PAM_frame), state);
end

watermarked_signal = audio_signal + watermark_signal;
```

MPEG compression is then applied to the new `watermarked_signal`.

```
compressed_watermarked_signal=mp3_codec( watermarked_signal,Fs);
```

Modifying the receiver

Before starting the watermark extraction, the `compressed_watermarked_signal` is low-passed, to avoid residual components over 11 kHz.

```
compressed_watermarked_signal = ...
   filter(low_pass_filter, 1, [compressed_watermarked_signal; ...
      zeros(N_coef/2, 1)]);
% Compensating tof the |N_coef/2| delay
compressed_watermarked_signal = ...
    compressed_watermarked_signal(N_coef/2+1:end);
```

Watermark extraction is finally obtained as in Section 7.2.3, except the LP filter is again taken into account in the zero-forcing equalization[14]. The resulting BER is similar to the one we obtained with no MPEG compression: our watermarking system is thus now robust to MPEG compression.

```
BER =   0.00095

received_message = A5dio watermarking: this message will be embedded
in an audio signal, using spread spectrum modulation, and later
retrieved from the modulated signal.
```

[14] The corresponding MATLAB code can be easily found in the `ASP_watermarking.m` file. We do not repeat it here.

7.3 Going further

Readers interested in a more in-depth study of spread spectrum communications will find many details, as well as MATLAB examples, in (Proakis *et al.* 2000).

Several books are available on digital watermarking techniques. (Cox *et al,* 2001) provides an intuitive approach, and many examples in C. Source code can also be found in (Pan *et al.* 2004).

Notice also that this Chapter has only mentioned *additive watermarking* (yet only in the time-domain), as opposed to the other main approach, known as *substitutive watermarking* (Arnold *et al.* 2003). Moreover, additive watermarking can be performed in various domains, leading to various bit-rate vs. distorsion vs. robustness trade-offs (Cox *et al.* 2001): in the time-domain, in the frequency-domain, in the amplitude or phase domain (such as the one corresponding to the output of a Modulated Lapped Transform, for robustness to small de-synchronization; see Malvar and Florencio, 2003), in the cepstral domain (often used in speech processing for its relation to the source/filter model; see Lee and Ho, 2000), in other parametric domains (such as the MPEG compressed domain; see Siebenhaar *et al.* 2001). Last but not least, we have not examined synchronization problems here, which require special attention in the context of spread spectrum communications, since even a one-sample delay will have a disastrous effect on the resulting BER (Baras *et al.* 2006).

Security is one of the emerging challenges in watermarking (Furon *et al.* 2007). For more challenges, see for instance the BOWS challenges organized by the within European Network of Excellence ECRYPT (Bows2, 2007).

7.4 Conclusion

Hiding bits of information in an audio stream is not as hard as one might think. It can be achieved by modulating the transmitted bits by a spread spectrum sequence, and adding the resulting watermark signal to the audio stream. Making the watermark inaudible is a matter of dynamically shaping its spectrum so that it falls below the local masking threshold provided by a psycho-acoustical model. Last but not least, ensuring that the system is robust to MPEG compression simply requires the addition of a low-pass filter in the spectrum-shaping stage.

Provided the receiver is able to invert the spectrum-shaping operation using the same psycho-acoustical model as the emitter, and with the additional help of a Wiener filter to increase the SNR, a bit error rate of the order of 0.001 can be obtained.

7.5 References

Arnold M, Wolthusen S, Schmucker M (2003) Techniques and applications of digital water-marking and content protection. Artech House Publishers, Norwood, MA

Bows2 (2007) Break our Watermarking system, 2nd edition [online] Available: http://bows2.gipsa-lab.inpg.fr [02/1/2008]

Baras C, Moreau N, Dymarski P (2006) Controlling the inaudibility and maximizing the robustness in an audio annotation watermarking system. IEEE Transactions on Audio, Speech and Language Processing, 14-5:1772–1782

Chen B, Wornell G (2001) Quantization index modulation: a class of provably good methods for digital watermarking and information embedding. IEEE Transactions on Information Theory, 47:1423–1443

Costa M (1983) Writing on dirty paper. IEEE Transactions on Information Theory, 29:439–441

Cox I, Miller M, Bloom J (2001) Digital Watermarking: Principles and Practice Morgan Kaufmann, San Francisco, CA

Cox I, Miller M, McKellips A (1999) Watermarking as communications with side information. Proceedings of the IEEE, 87–7: 1127–1141

Craver SA, Wu M, Liu B (2001) What Can We Reasonably Expect From Watermarks? Proceedings of the IEEE Workshop on Applications of Signal Processing to Audio and Acoustics, New Paltz, NY, 223–226

Furon T, Cayre F, Fontaine C (2007) Watermarking Security. In: Cvejic and Seppanen (eds) Digital Audio Watermarking Techniques and Technologies: Applications and Benchmarks. Information Science Reference, Hershey, PA (USA)

Haykin S (1996) Adaptive filter theory, 3rd ed. Prentice-Hall, Upper Saddle River,NJ (USA)

Jobs S (2007) Thoughts on Music [online] Available: http://www.apple.com/hotnews/thoughtsonmusic/ [26/9/2007]

Kirovski D, Malvar H (2003) Spread-spectrum watermarking of audio signals. IEEE Transactions on Signal Processing, 51-4:1020–1033

Larbi S, Jaïdane M, Moreau N (2004). A new Wiener filtering based detection scheme for time domain perceptual audio watermarking. In IEEE Interna tional Conference on Acoustics, Speech and Signal Processing (ICASSP), Montreal, Canada, 5:949–952

Lee SK, Ho YS (2000) Digital audio watermarking in the cepstrum domain. IEEE Transactions on Consumer Electronics, 46-3: 744–750.

LoboGuerrero A, Bas P, Lienard J (2003) Iterative informed audio data hiding scheme using optimal filter," in Proc. IEEE Int. Conf. on Communication Technology, Beijing, China, 1408–1411

Malvar H, Florencio D (2003) "Improved spread spectrum: a new modulation technique for robust watermarking" IEEE Transactions on Signal Processing, 51-4:898–905

Massey JL (1994) Information theory aspects of spread-spectrum communications. IEEE Third International Symposium on Spread Spectrum Techniques and Applications (ISSSTA), 1:16–21

Pan JS, Huang HC, Jain LC (2004) Intelligent Watermarking Techniques (Innovative Intelligence). World Scientific Publishing Company, Singapore

Petitcolas FAP, Anderson RJ, Kuhn MG (1999) Information hiding – a survey. Proceedings IEEE, special issue on protection of multimedia content, 87(7):1062–1078

Peterson RL, Ziemer RA, Borth DE (1995) Introduction to spread spectrum communications. Prentice Hall, Upper Saddle River, NJ (USA)

Proakis JG, Salehi M, Bauch G (2004) Contemporary Communication Systems Using MATLAB and SIMULINK. Brooks/Cole-Thomson Learning, Belmont, CA (USA)

Shannon C (1958) Channel with side information at the transmitter. IBM Journal of Research and Development, 2: 222–293

Siebenhaar F, Neubauer C, Herre J, Kulessa R (2001) New results on combined audio compression/watermarking. In: 111th Convention of Audio Engineering Society (AES), New York, USA, preprint 5442.

Proakis J (2001) Digital communications, 4th ed. New York, USA: McGraw-Hill