Chapter 2

Sound modeling: signal-based approaches

Giovanni De Poli and Federico Avanzini

2.1 Introduction

The sound produced by acoustic musical instruments is caused by the physical vibration of a certain resonating structure. This vibration can be described by signals that correspond to the time-evolution of the acoustic pressure associated to it. The fact that the sound can be characterized by a set of signals suggests quite naturally that some computing equipment could be successfully employed for generating sounds, for either the imitation of acoustic instruments or the creation of new sounds with novel timbral properties.

A wide variety of sound synthesis algorithms is currently available either commercially or in the literature. Each one of them exhibits some peculiar characteristics that could make it preferable to others, depending on goals and needs. Technological progress has made enormous steps forward in the past few years as far as the computational power that can be made available at low cost is concerned. At the same time, sound synthesis methods have become more and more computationally efficient and the user interface has become friendlier and friendlier. As a consequence, musicians can nowadays access a wide collection of synthesis techniques (all available at low cost in their full functionality), and concentrate on their timbral properties.

Each sound synthesis algorithm can be thought of as a digital model for the sound itself. Though this observation may seem quite obvious, its meaning for sound synthesis is not so straightforward. As a matter of fact, modeling sounds is much more than just generating them, as a digital model can be used for representing and generating a whole class of sounds, depending on the choice of control parameters. The idea of associating a class of sounds to a digital sound model is in complete accordance with the way we tend to classify natural musical instruments according to their sound generation mechanism. For example, strings and woodwinds are normally seen as timbral classes of acoustic instruments characterized by their sound generation mechanism. It should be quite clear that the degree of compactness of a class

of sounds is determined, on one hand, by the sensitivity of the digital model to parameter variations and, on the other hand, on the amount of control that is necessary for obtaining a certain desired sound. As an extreme example we may think of a situation in which a musician is required to generate sounds sample by sample, while the task of the computing equipment is just that of playing the samples. In this case the control signal is represented by the sound itself, therefore the class of sounds that can be produced is unlimited but the instrument is impossible for a musician to control and play. An opposite extremal situation is that in which the synthesis technique is actually the model of an acoustic musical instrument. In this case the class of sounds that can be produced is much more limited (it is characteristic of the mechanism that is being modeled by the algorithm), but the degree of difficulty involved in generating the control parameters is quite modest, as it corresponds to physical parameters that have an intuitive counterpart in the experience of the musician.

An interesting conclusion that could be already drawn in the light of what stated above is that the compactness of the class of sounds associated to a sound synthesis algorithm is somehow in contrast with the "playability" of the algorithm itself. One should remember that the "playability" is of crucial importance for the success of a specific sound synthesis algorithm as, in order for a sound synthesis algorithm to be suitable for musical purposes, the musician needs an intuitive and easy access to its control parameters during both the sound design process and the performance. Such requirements often represents the reason why a certain synthesis technique is preferred to others.

Some considerations on control parameters are now in order. Varying the control parameters of a sound synthesis algorithm can serve several purposes, the first one of which is certainly that of exploring a sound space, i.e. producing all the different sounds that belong to the class characterized by the algorithm itself. This very traditional way of using control parameters would nowadays be largely insufficient by itself. As a matter of fact, with the progress in the computational devices that are currently being employed for musical purposes, the musician's needs have turned more and more toward problems of timbral dynamics. For example, timbral differences between soft (dark) and loud (brilliant) tones are usually obtained through appropriate parameter control. Timbral expression parameters tend to operate at a note-level time-scale. As such, they can be suitably treated as signals characterized by a rather slow rate.

Another reason for the importance of time-variations in the algorithm parameters is that the musician needs to control the musical expression while playing. For example, staccato, legato, vibrato etc. need to be obtained through parameter control. Such parameter variations operate at a phrase-level time-scale. Because of that, they can be suitably treated as sequences of symbols events characterized by a very slow rate.

In conclusion, control parameters are signals characterized by their own time-scales. Controls signals for timbral dynamics are best described as discrete-time signals with a slow sampling rate, while controls for musical expression are best described by streams of asynchronous symbols events. As a consequence, the generation of control signals can once again be seen as a problem of signal synthesis.

2.2 Time-segment based models

2.2.1 Wavetable synthesis

2.2.1.1 Definitions and applications

Finding a mathematical model that faithfully imitates a real sound is an extremely difficult task. If an existing reference sound is available, however, it is always possible to reproduce it through recording. Such a method, though simple in its principle, is widely adopted by digital sampling instruments or

samplers and is called wavetable synthesis or sampling.[1]

A samplers is a device which is able to store and play back a large quantity of recorded sounds: most typically these are musical, instrumental sounds, but they may also be non-musical, environmental sounds. In fact the most appealing quality of wavetable synthesis is the possibility of working with an unlimited variety of sounds. From the implementation viewpoint, computational simplicity is certainly an advantage of the technique, which contrasts with the need of huge memory capacities (as an example, a contemporary piano synthesizer based on sampling may require tens of Gigabytes to store the entire instrument).

The table look-up mechanism is in essence identical to that already outlined for the wavetable oscillator in Chapter *Fundamentals of digital audio processing*. The recorded sound is stored in a table of length L, and the synthesized sound $x[n]$ is the result of reading the table using a phase signal $\phi[n]$ with a certain length M :

$$x[n] = \text{tab}\{\phi[n]\}, \qquad \phi : [0, M-1] \rightarrow [0, L-1]. \tag{2.1}$$

Simple playback one sound of the stored repertoire is obtained using the signal $\phi[n] = n$ and with $M = L$. In this case all the original samples of the stored sound are played back at the original speed.

Using different signals $\phi[n]$ will result in modifications of the original sound. However the possibilities of modification are rather limited, as it would be for the sound recorded by a tape deck. The most common modification is obtained by varying the speed when reproducing the sound, which results in a pitch transposition. Other simple transformations include time-reversal (i.e. table read out starting from the last sample), looping (i.e. periodically scanning the wavetable or part of it), cutting and possibly rearranging portions of the original sound. In particular, it is possible to adopt looping techniques with almost any stationary portion of sounds. One method of improving the expressive possibilities of samplers is store multiple instances of a single instrumental note for different dynamics (e.g. from soft to loud key velocities in the case of the piano), or even different pitches, and switching or interpolating between these upon synthesis. This approach is called *multisampling*. During synthesis, linear interpolation between stored sounds is performed as function of the desired dynamics.

Historically, the use of samplers may be traced back to a time where digital sound was still to come and magnetic tapes had just made their appearance. So-called *musique concrète*, first developed in Paris in late 1940's mainly through the work of french composer Pierre Schaeffer and colleagues, stemmed from an aesthetic practice that was centered upon the use of sound as a primary compositional resource. The development of this music was facilitated by the emergence of then new technologies such as microphones and magnetic tape recorders. Compositions were based on (usually non-musical) sounds acquired through a microphone, recorded on tape, manipulated and recombined in variety of ways, and finally played back from tape.

Nowadays wavetable synthesis techniques are presented as a method for reproducing natural instrumental sounds that are comparable in quality with the original instruments. This is the main reason why the most popular commercial digital keyboards adopt this synthesis technique. Of course, sampling cannot feature all the expressive possibilities of the original instrument.

2.2.1.2 Tranformations: pitch shifting, looping

Pitch shifting, i.e. transposition of the original pitch of the recorded sound, is the simplest transformation obtainable with wavetable synthesis. Assume that the original sample rate and the playback sample rate are the same. If we read the entire table (of length L) with a signal ϕ of length M, then the overall sound duration is compressed or expanded by a factor M/L and the pitch is transposed by a factor L/M.

[1] In musical jargon it has become customary to use the term *sample* to denote an entire sound, recorded and stored in a table. To avoid confusion, in this book we only use the term *sample* only with the meaning explained in Chapter *Fundamentals of digital audio processing*, i.e. a single value of a digital signal.

An overall transposition of a factor L/M can be obtained by reading the table with the signal

$$\phi[n] = \left\lfloor n\frac{L}{M} \right\rfloor, \qquad n = 0, \ldots, M-1. \tag{2.2}$$

As an example, if $3M = 2L$ the sound will be transposed upwards by a perfect fifth (i.e. by a ratio of $3/2$). However, ϕ needs not to be a ramp that rises linearly in time. In the case where ϕ is still a monotonic signal, but changes its slope during time, the transposition will become time-dependent.

Although the pitch shifting outlined above is simple and straightforward to implement, it has to be noted that substantial pitch variations are generally not very satisfactory as a temporal waveform compression or expansion results in unnatural timbral modifications, which is exactly what happens when the playing speed is changed in a tape recorder. Satisfactory quality and timbral similarity between the original tone and the transposed one can be obtained only if small pitch variations (e.g. a few semitones) are performed. As an example, sampling a piano with a reasonable quality along the entire instrumental extension requires that many notes are stored (e.g. three for each octave). In this way, notes that have not been sampled can be obtained from the available ones through transposition of max. two semitones.

M-2.1

Import a .wav file of a single instrument tone. Scale it (compress and expand) to different extents and listen to the new sounds. Up to what scaling ratio are the results acceptable?

Often it is desired to vary the sound also in function of other parameters, the most important being the intensity. To this purpose it not sufficient to change the sound amplitude by a multiplication, by it is necessary to modify the timbre of the sound. In general louder sounds are characterized by a sharper attack and by a brighter spectrum. In this case a technique could be to use a unique sound prototype (e.g. a tone played fortissimo) and then obtaining the other intensity by simple spectral processing, as low pass filtering. A different and more effective solution, is to use a set of different sound prototype, recorder with different intensity (e.g. tones played fortissimo, mezzo forte, pianissimo) and then obtaining the other dynamic values by interpolations and further processing.

This technique is thus characterized by high computational efficiency and high imitation quality, but by low flexibility for sounds not initially included in the repertoire or not easily obtainable with simple transformations. There is a trade-off of memory size with sound fidelity.

In order to employ efficiently the memory, often the sustain part of the tone is not entirely stored but only a part (or few significant parts) and in the synthesis this part is repeated (*looping*). Naturally the repeated part should not be to short, to avoid a static character of the resulting sound. For example to lengthen the duration of a note, first the attack is reproduced without modification, then the sustain part is cyclically repeated, with possible cross interpolation among the different selected parts, and finally the sound release stored part is reproduced. Notice that if we want to avoid artefacts in cycling, particular care should be devoted to choosing the points of the beginning and ending of the loop. Normally an integer number of periods is used for looping starting with a null value, to avoid amplitude or phase discontinuities. In fact these discontinuities are very annoying. To this purpose it may be necessary to process the recorded samples by slightly changing the phases of the partials.

M-2.2

Import a .wav file of a single instrument tone. Find the stationary (sustain) part, isolate a section, and perform the looping operation. Listen to the results, and listen to the artifacts when the looped section does not start/end at zero-crossings.

If we want a less static sustain, it is possible to individuate some different and significant sound segments, and during the synthesis interpolate (*cross-fade*) among subsequent segments. In this case the temporal evolution of the tone can be more faithfully reproduced.

2.2.2 Overlap-Add (OLA) methods

The definition Overlap-Add (OLA) refers in general to a family of algorithms that produce a signal by properly assembling a number of signal segments. OLA methods are developed both in the time domain and in the frequency domain. Here we are interested in reviewing briefly time-domain approaches, while in Sec. 2.3 we will address time-frequency representations.

2.2.2.1 Basic time-domain overlap-add

Given a sound signal $x[n]$, a sequence of windowed segments $x_m[n]$ can be constructed as

$$x_m[n] = x[n]w_a[n - mS_a], \tag{2.3}$$

where $w_a[n]$ is an analysis window and S_a is the *analysis hop-size*, i.e. the time-lag (in samples) between one analysis frame and the following one. If the window w_a is N samples long, then the *block size*, i.e. the length of each frame $x_m[n]$, will be N. In order for the signal segments to actually overlap, the inequality $S_a \leq N$ must be verified. When $S_a = N$ the segments are exactly juxtaposed with no overlap.

Given the above signal segmentation, time-domain overlap-add (OLA) methods construct an output signal $y[n] = \sum_m y_m[n]$, where the segments y_m are modified versions of the input segments x_m. As an example, they can be obtained by modifying S_a in Eq. (2.3), or by repeating/removing some of the input segments x_m. In the absence of modifications, this procedure reduces to the identity ($y[n] \equiv x[n]$) if the overlapped and added analysis windows w_a sum to unity:

$$y[n] = \sum_m x_m[n] = \sum_m x[n]w_a[n - mS_a] = x[n] \quad \Leftrightarrow \quad A_{w_a}[n] \triangleq \sum_m w_a[n - mS_a] \equiv 1. \tag{2.4}$$

If this condition does not hold, then the function A_{w_a} acts on the reconstructed signal as a periodic *amplitude modulation envelope*, with period S_a. Depending on the application and on the window length N, this amplitude modulation can introduce audible artifacts. This kind of frame rate distortion can be seen in the frequency domain as a series of sidebands with spacing F_s/S_a in a spectrogram of the output signal. In fact, one may prove that the condition $A_{w_a} \equiv 1$ is equivalent to the condition $W(e^{j\omega_k}) = 0$ at all harmonics of the frame rate F_s/S_a. Figure 2.1 illustrates an example of signal reconstruction using a triangular window, which satisfies the condition of Eq. (2.4).

A widely studied effect is *time-stretching*, i.e. contraction or expansion of the duration of an audio signal. Time-stretching algorithms useful in a number of applications: think about wavetable synthesis, post-synchonization of audio and video, speech technology at large, and so on. A time-stretching algorithm should ideally shorten or lengthen a sound file composed of N_{tot} samples to a new desired length $N'_{tot} = \alpha N_{tot}$, where α is the *stretching* factor. Note that a mere resampling of the sound signal does not provide the desired result, since it has the side-effect of transposing the sound: in this context resampling is the digital equivalent of playing the tape at a different speed.

What one really wants is a scaling of the perceived timing attributes without affecting the perceived frequency attributes. More precisely, we want the time-scaled version of the audio signal to be perceived as the same sequence of *acoustic events* as the original signal, only distributed on a compressed/expanded time pattern. As an example, a time-stretching algorithm applied to a speech signal should change the speaking rate without altering the pitch.

Time-domain OLA techniques are one possible approach to time-stretching effects. The basic OLA algorithm described above can be adapted to this problem by defining an *analysis hop size*) S_a and a *synthesis hop size* $S_s = \alpha S_a$, scaled by the stretching factor, that will be applied to the output. An input signal $x[n]$ is then segmented into frames $x_k[n]$, each taken every S_a samples. The output signal $y[n]$

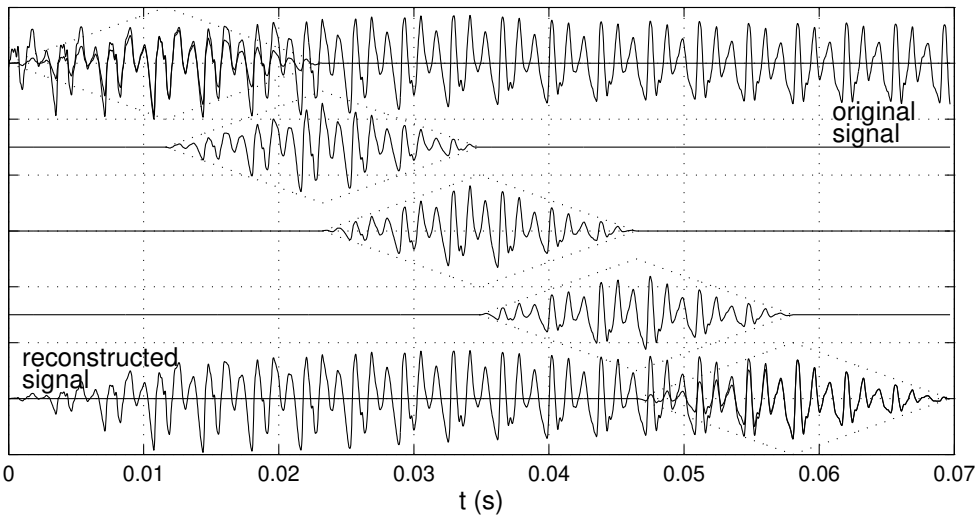


Figure 2.1: *An example of OLA signal reconstruction, with triangular windowing.*

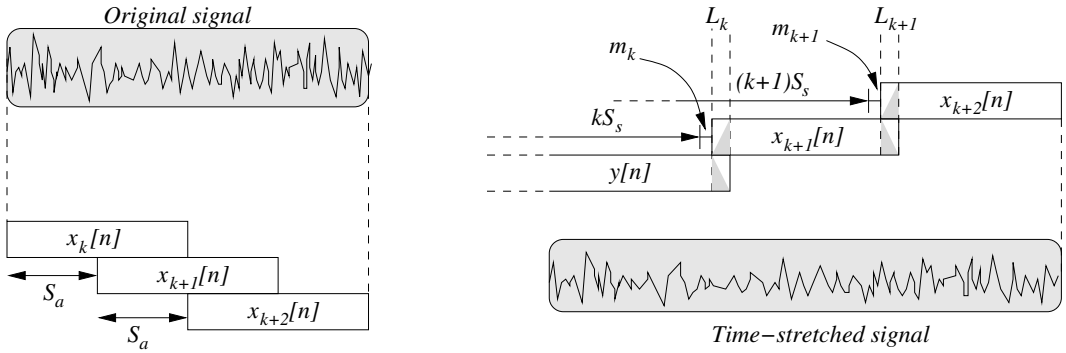


Figure 2.2: *The generic SOLA algorithmic step.*

is produced by reassembling the same frames $x_k[n]$, each added to the preceding one every S_s samples. However this repositioning of the input segments with respect to each other destroys the original phase relationships, and constructs the output signal by interpolating between these misaligned segments. This cause pitch period discontinuities and distortions that can produce heavily audible artifacts in the output signal.

2.2.2.2 Syncronous overlap-add

In order to avoid phase discontinuities at the boundaries between frames, a proper time alignment of the blocks has to be chosen. The *synchronous overlap-add (SOLA)* algorithm realizes such a proper alignement, and provides a good sound quality (at least for values of α not too far from 1) while remaining computationally simple, which makes it suitable even for real-time applications.

Let N be the analysis block length. In the initialization phase the SOLA algorithm copies the first N samples from $x_1[n]$ to the output $y[n]$, to obtain a minimum set of samples to work on:

$$y[j] = x_1[j] = x[j] \qquad \text{for } j = 0 \dots N-1. \tag{2.5}$$

Then, during the generic kth step the algorithm tries to find the optimal overlap between the last portion

of the output signal $y[n]$ and the incoming analysis frame $x_{k+1}[n]$. More precisely, $x_{k+1}[n]$ is pasted to the output $y[n]$ starting from sample $kS_s + m_k$, where m_k is a small discrete time-lag that optimizes the alignement between y and x_k (see Fig. 2.2). Note that m_k can in general be positive or negative, although for clarity we have used a positive m_k in Fig. 2.2.

When the optimal time-lag m_k is found, a *linear crossfade* is used within the overlap window, in order to obtain a gradual transition from the last portion of y to the first portion of x_k. Then the last samples of x_k are pasted into y. If we assume that the overlap window at the kth SOLA step is L_k samples long, then the algorithmic step computes the new frame of the input y as

$$y[kS_s + j] = \begin{cases} (1 - v[j])y[kS_s + j] + v[j]x_k[j] & \text{for } m_k \leq j \leq L_k \\ x_k[j] & \text{for } L_k + 1 \leq j \leq N \end{cases} \tag{2.6}$$

where $v[j]$ is a linear *smoothing function* that realizes the crossfade between the two segments. The effect of Eq. (2.6) is a local replication or suppression of waveform periods (depending on the value of α), that eventually results in an output signal $y[n]$ with approximately the same spectral properties of the input $x[n]$, and an altered temporal evolution.

At least three techniques are commonly used in order to find the optimal value for the discrete time lag m_k at each algorithmic step k:

1. Computation of the minimum vectorial inter-frame distance in an L_1 sense (cross-AMDF)

2. Computation of the maximum cross-correlation $r_k(m)$ in a neighborhood of the sample kS_s. Let M be the width of such neighborhood, and let $y_{M_k}[i] = y[kS_s + i]$ for $i = 1 \ldots M - 1$, and $x_{M_k}[i] = x_{k+1}[i]$ for $i = 1 \ldots M - 1$. Then the cross-correlation $r_k(m)$ is computed as

$$r_k[m] \triangleq \sum_{i=0}^{M-m-1} y_{M_k}[i] \cdot x_{M_k}[i + m], \qquad m = -M + 1, \ldots, M - 1. \tag{2.7}$$

   Then m_k is chosen to be the index of maximal cross-correlation: $r_k[m_k] = \max_m r_k[m]$.

3. Computation of the maximum *normalized* cross-correlation, where every value taken from the cross-correlation signal is normalized by dividing it by the product of the frame energies.

The latter technique is conceptually preferable, but the second one is often used for efficiency reasons.

M-2.3

Write a function that realizes the time-stretching SOLA algorithm through segment cross-correlation.

M-2.3 Solution

```
function y = sola_timestretch(x,N,Sa,alpha,L)
%N: block length; Sa: analysis hop-size; alpha: stretch factor; L: overlap int.

Ss = round(Sa*alpha); %synthesis hop-size
if ( (Sa > N) || (Ss >= N) || Ss > N-L) error('Wrong parameter values!'); end
if (rem(L,2)~= 0)   L = L+1; end

M = ceil(length(x)/Sa);   %number of frames
x(M*Sa+N)=0;              %now x is exactly M*Sa samples
y(1:N,1) =  x(1:N);         %first frame of x is written into y;

for m=1:M-1    %loop over frames
  frame=x(m*Sa+1:N+m*Sa);   %current analysis frame
  framecorr=xcorr(frame(1:L),y(m*Ss:m*Ss+(L-1)));
```

```
    [corrmax,imax]=max(framecorr);  %find point of max xcorrelation

    fade = (m*Ss-(L-1)+imax-1):length(y);     %points for crossfade
    fadein = (0:length(fade)-1)/length(fade); %from 0 to 1
    fadeout = 1 - fadein;                     %from 1 to 0
    y=[y(1:(fade(1)-1)); (y(fade).*fadeout' +frame(1:length(fade)).*fadein'); ...
       frame(length(fade)+1:length(frame))];
end
```

2.2.2.3 Pitch synchronous overlap-add

A variation of the SOLA algorithm for time stretching is the *pitch synchronous overlap-add* (PSOLA) algorithm, which is especially used for voice processing. PSOLA assumes that the input sound is pitched, and exploits the pitch information to correctly align the segments and avoid pitch discontinuities. The algorithm is composed of two phases: analysis/segmentation of the input sound, and resynthesis of the time-stretched output signal.

The analysis phase works in two main steps. The first *pitch estimation* step determines a set $\{n_i\}_i$ of time instants ("pitch marks"), such that signal portions between n_{i+1} and n_i contain an integer number of pitch periods (e.g., 2 or 4 periods), and the pitch can be considered constant within each of these portions. In this way a pitch function $P[n_i] = n_{i+1} - n_i$ is estimated. This function represents the time-varying period (in samples) of the original signal. As an example, in a voice signal pitch marks can be chosen to be points of maximum amplitude of the mouth pressure signal: they corresponds to instants of closure of the vocal folds, which occur periodically in a voiced signal. This step is clearly the most critical and computationally expensive one.

The second step in the analysis phase is a *segmentation* step. Segments $x_i[n]$ are created by windowing the input signal (typically a Hanning window is used): segments have a block length N of two pitch periods, and are centered at every pitch mark n_i. This procedure implies that the analysis hop-size is $S_a = N/2$.

In the synthesis phase, the segments $x_i[n]$ are realigned as follows. First, note that the time-stretched signal must have a pitch function $\tilde{P}$ that is a stretched version of P. More precisely, the relation $\tilde{P}[\alpha n_i] = P[n_i]$ must hold (where for simplicity we are assuming that the time instants αn_i are still integers). This relation is then used to determine a new set $\{\tilde{n}_k\}_k$ of pitch marks for the output signal. The $\tilde{n}_k$'s are iteratively determined as follows:

$$\begin{aligned} \tilde{n}_1 &= n_1, \\ \tilde{n}_{k+1} &= \tilde{n}_k + P\left(n_{i(k,\alpha)}\right), \quad k > 1, \end{aligned} \tag{2.8}$$

where $i(k, \alpha)$ is the index of the input pitch marks that minimizes the distance $|\, \alpha n_i - \tilde{n}_k \,|$. Intuitively, this means that at the time instant $\tilde{n}_k$ the time-stretched signal must have the same pitch possessed by the original signal at time n_i, with $\tilde{n}_k \sim \alpha n_i$.

Once the set $\{\tilde{n}_k\}_k$ has been determined in this way, for every k the segment $x_{i(k,\alpha)}[n]$ is overlapped and added at the point $\tilde{n}_k$. The algorithm is visualized in Fig. 2.3: note that with a stretching factor $\alpha > 1$ (time expansion, as in Fig. 2.3) some segments will be *repeated*, or equivalently the function $i(k, \alpha)$ will take identical values for some consecutive values of k. Similarly, when $\alpha < 1$ (time compression) some segments are discarded in the resynthesis.

The main advantage of the PSOLA algorithm with respect to SOLA is that it allows for a better alignment of segments, by exploiting information about pitch instead of using a simple cross-correlation. On

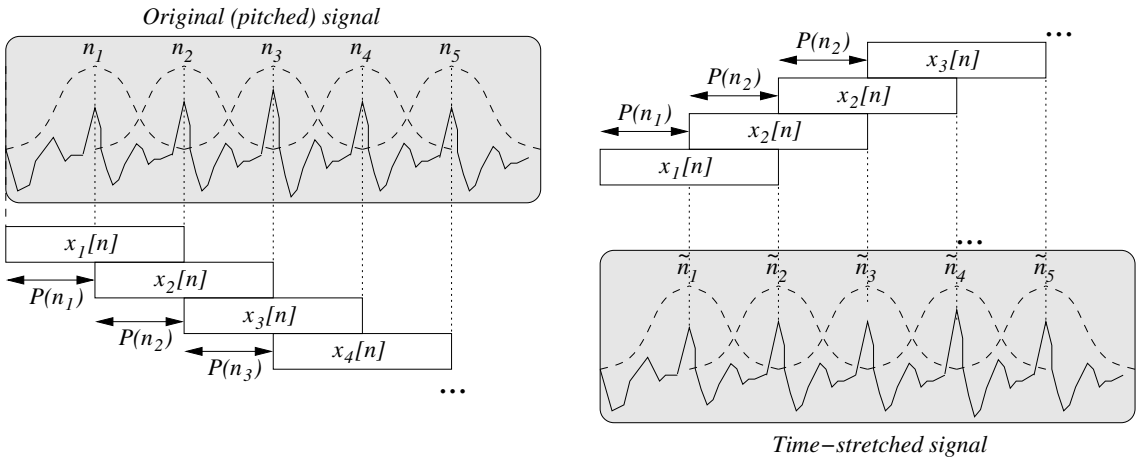


Figure 2.3: *The generic PSOLA algorithmic step.*

the other hand it has a higher complexity especially because of the pitch estimation procedure. Moreover, noticeable artifacts still appears for very small or large stretching factors. One problem is that when α is very large, identical segments will be repeated several times thus providing an unnatural character to the sound. A second more general problem is that the OLA algorithms examined here stretch an input signal uniformly, including possible trasients, which instead should be preserved.

M-2.4

Write a function that realizes the time-stretching PSOLA synthesis algorithm, given a vector of input pitch marks n_i.

M-2.4 Solution

```
function y=psola_timestretch(x,nis,alpha)
%N: block length; nis: pitch marks; alpha: stretch factor;

P = diff(nis);   %compute pitch periods
%%%%% remove first and last pitch marks %%%&
if( nis(1)<=P(1) )   nis=nis(2:length(nis));  P=P(2:length(P));   end
if( nis(length(nis))+P(length(P))>length(x) ) nis=nis(1:length(nis)-1);
else                                          P=[P P(length(P))]; end

y=zeros(1, ceil(length(x)*alpha +max(P)) ); %initialize output signal
nk = P(1)+1;                              %initialize output pitch mark
while round(nk)<length(y)
  [minimum i] = min( abs(alpha*nis - nk) );         %find analysis segment
  xi = x(nis(i)-P(i):nis(i)+P(i)).*hanning(2*P(i)+1); %take analysis frame
  %%%% overlap the frame to the output signal%%%
  y(round(nk)-P(i):round(nk)+P(i)) = y(round(nk)-P(i):round(nk)+P(i))+xi;
  nk=nk+P(i);
  display('done')
end
```

Clearly we have omitted the most difficult part of the PSOLA approach, i.e. an algorithm that determines the input pitch marks (the vector `nis` in our code).

2.2.3 Granular synthesis

The term "granular synthesis" can be used to define a family of sound synthesis approaches that share the basic idea of building complex sounds from simple ones. Granular synthesis assumes that a sound can be considered as a sequence, possibly with overlaps, of elementary acoustic elements called grains. Granular synthesis constructs complex and dynamic acoustic events starting from a large quantity of grains. The features of the grains and their temporal location determine the sound timbre.

One can make an analogy with cinema, where a rapid sequence of static images gives the impression of objects in movement. Another possible analogy is with the technique of mosaicing, where the grains are single small monochromatic plugs, and their juxtaposition produces a complex image. In music, the use of granular synthesis techniques arises from the experiences of taped electronic music. In the early years of electronic music, the tools that composers had at disposal (e.g., fixed waveform oscillators and filters) did not allow for substantial variations of sound timbres. However they were able to obtain dynamic sounds by cutting tapes into short sections and then putting them together again. The rapid alternation of acoustic elements provides a certain variety to the resulting sound.[2]

2.2.3.1 Gaborets

The initial idea of granular synthesis can be traced back to the work of the hungarian physicist Dennis Gabor, which was aimed at pinpointing the physical and mathematical ideas needed to understand what a time-frequency spectrum is. He considered sound as a sum of elementary Gaussian functions that have been shifted in time and frequency. Gabor considered these elementary functions as *acoustic quanta*, the basic constituents of a sound. These works have been rich in implications and have been the starting point for studying time-frequency representations and wavelet theory.

The usual Gabor expansion on a rectangular time-frequency lattice of a signal $x(t)$ can be expressed as a linear combination of "grains" $g_{mk}(t)$, that are shifted and modulated versions of a synthesis window $w(t)$

$$x(t) = \sum_m \sum_k a_{mk}\, g_{mk}(t), \qquad \text{with} \quad g_{mk}(t) = w(t - m\alpha T) e^{jk\beta\Omega t}. \tag{2.9}$$

Other names for these grains, or acoustic quanta, are *gaborets*, or *Gabor functions*, or *Gabor atoms*.

In Gabor formulation $w(t)$ is a gaussian window of the form

$$w(t) = \frac{1}{\sigma\sqrt{2\pi}} e^{-t/2\sigma^2}, \tag{2.10}$$

where σ is the standard deviation of the gaussian. An important property of this function is that it is possibly the only smooth, nonzero function, known in closed form, that is transformed to itself in the Fourier domain:

$$\mathcal{F}\{w\}(\omega) = e^{-t/2(1/\sigma)^2}. \tag{2.11}$$

Therefore the grain g_{mk} has a gaussian shape both in time and in frequency: it is a gaussian bell in the time-frequency domain. As a particular case of the uncertainty principle, the width of the time-domain gaussian is inversely proportional to that of the frequency-domain gaussian, as shown by Eq. (2.11).

Historically, the use of granular synthesis in musical applications has been classified into two main approaches. The first one is based on the use of sampled sounds to construct grains, while the second one is based on the use of abstract, entirely synthetic grains.

[2] The greek composer Iannis Xenakis is generally acknowledged to have provided the first formulation of granular synthesis in his compositions *Analogique A et B* (1958-59).

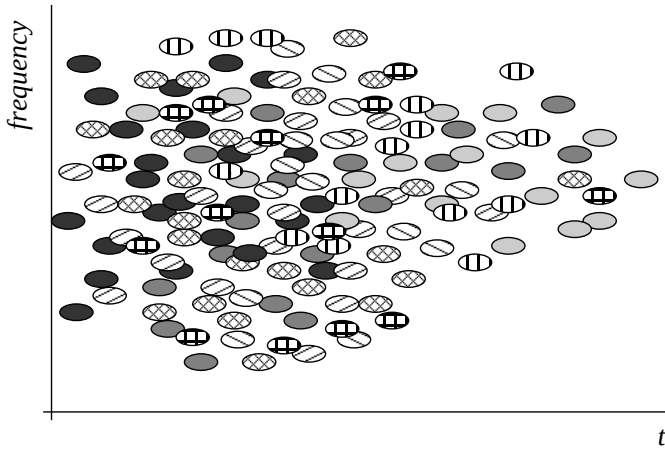


Figure 2.4: *Representation of granular synthesis where grains derived from different sources are randomly mixed.*

2.2.3.2 Sound granulation

The term *sound granulation* is typically used to identify the first of the two above mentioned approaches. Complex waveforms, extracted from real sounds or described by spectra, are organized in succession with partial overlap in time. In this way, it is possible both to reproduce accurately real sounds and modify then in their dynamic characteristics. The original sound $x[n]$ used to create the grains may have been previously recorded, or may be processed in real-time. In this respect granular synthesis can be viewed as an OLA technique in which segments $x_m[n]$ of a sound signal $x[n]$ represent the grains, and are processed both in time and frequency before being reassembled.

$$g_m[n] = x[n] \cdot w_m[n - n_m], \qquad \text{with} \quad n = n_m \dots n_m + L_m. \tag{2.12}$$

The time instant n_m indicates the point where the windowing starts and the segment is extracted; the length L_m of the window determines the amount of signal extracted; the shape of the window $w_m[n]$ should provide an amplitude envelope that ensures fade-in and fade-out at the border of the grain and affects the frequency content of the grain.

The length L_m is a critical parameter. Long grains tend to maintain the timbre identity of the portion of the input signal, while short ones acquire a pulse-like quality and frequency information is not perceivable any more. When the grain is long, the window has a flat top and it used only to fade-in and fade-out the borders of the segment. Typical grain length are in the range $5 - 100$ ms.

The organization of the choice of the frequencies is also very important, therefore in granular synthesis the proper timing organization of the grain is essential to avoid artifacts produced by discontinuities. This problem makes often the control quite difficult.

Notice that it is possible to extract grains from different sounds, to create hybrid textures, e.g. evolving from one texture to another. A schematic visualization of this approach is given in Fig. 2.4.

2.2.3.3 Synthetic grains

The second of the two above mentioned approaches is based on grains that consist of synthetic waveforms whose amplitude envelope is a short Gaussian function. Most typically, frequency modulated gaussian functions are used in order to localize the energy both in frequency and time domain, in the line of Gabor

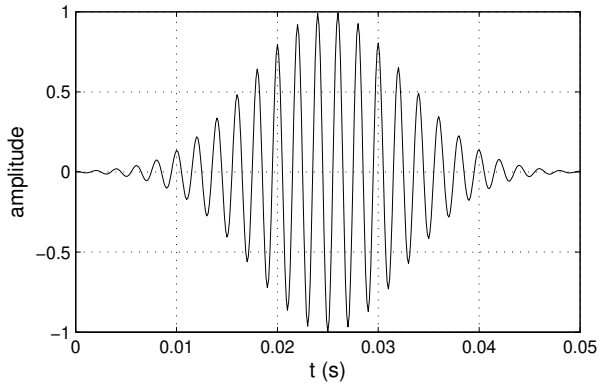


Figure 2.5: *Example of a synthetic grain waveform, with frequency* $\omega_k = 2\pi \cdot 500$ *rad/s and standard deviation* $\sigma = 0.2$.

work. In this case a single grain g_{mk} can be written as

$$g_{mk}[n] = w_{mk}[n - n_m] \cdot \cos\left(\omega_k \frac{n}{F_s} + \phi_k\right), \tag{2.13}$$

where the index m refers to the grain position in time, while the index k refers to its position in frequency. The window $w_{mk}[n]$ is a gaussian window shifted by n_m samples, and ω_k (in rad/s) is the frequency of the grain. A plot of a grain constructed in this way is provided in Fig. 2.5.

M-2.5

Write a function that computes a gaussian grain given its length, frequency, and standard deviation.

M-2.5 Solution

```
function yg = grain(L,f,sigma)
global Fs;

yg=(gausswin(L,1/sigma))'.*cos(2*pi*f*(1:L)/Fs);
```

The sound synthesized from these grains is again constructed according to Gabor formulation: it is an assemblage of grains scattered on the frequency-time plane in the form of "clouds". The general synthesis expression is given by

$$s[n] = \sum_m \sum_k a_{mk} \cdot g_{mk}[n], \tag{2.14}$$

where a_{mk} is the amplitude coefficient of the corresponding grain. Every grain contributes to the total sound energy around the point (n_m, ω_k) of the time frequency plane.

In order to implement the above synthesis equation, the simplest approach amounts to define a constant density of grains in time, so that there is a constant time-step between the generation of a grain and the following one. This approach is often termed *synchronous granular synthesis*.

M-2.6

Write a script that realizes a synchronous granular synthesis scheme.

M-2.6 Solution

```
global Fs; Fs=22050;

gfreq=800;    frange=800;  %grain freqs scattered around grainfreq
glength=0.05; lrange=0.05; %grain lengths scattered around glength
gsigma=0.2;  sigrange=0.1; %gaussian st.dev. scattered around gsigma

slength=2;                      %duration of the entire sound
gdens=100;                      % grain density (=no. of grains per second)
totgrains=round(slength*gdens); %total no. of grains to be generated

y=zeros(1,(slength+2*glength)*Fs);
for m=0:totgrains-1
    f = gfreq +(rand(1)-.5)*frange;              % grain frequency
    t = round(Fs*(glength+(rand(1)-.5)*lrange)); % grain length (samples)
    d = gsigma +(rand(1)-.5)*sigrange;           %grain st.dev.
    yg = grain(t,f,d);                           %construct grain
    frame = round(m*Fs/gdens) +(1:t); %frame to be written (shifts with m)
    y(frame) = y(frame) + yg;         %add current grain to sound
end
```

We have realized the synthesis in the time domain. Equivalent results could be obtained by working in the frequency domain.

However the most used type of granular synthesis is *asynchronous granular synthesis*, where grains are irregularly distributed in the time-frequency plane, e.g. they are scattered onto a mask that delimits specific portions of the time-frequency-amplitude space. This results in time-varying "clouds" of micro-sounds, or *sonic textures*, that can simulate even natural noisy sounds in which general statistical properties are more important than the exact sound evolution. Typical examples include the sound of numerous small objects (e.g., rice or sand) falling onto a resonating surface (e.g., a metal plate), or rain sounds composed by the accumulation of a large amount of water droplet micro-sounds, or even scratching/cracking sounds made by the accumulation of thousands of complex micro-sounds not necessarily deterministic. In general we can expect these types of sounds to occur in the real world when they are the result of multiple realizations of the same event or the same phenomenon.

Musical composers have tried to evaluate the effects of different control parameters from an aesthetic point of view. Grain duration affects the sonic texture: short duration (few samples) produces a noisy, particulate disintegration effect; medium duration (tens of ms) produces fluttering, warbling, gurgling; longer durations (hundreds of ms) produce aperiodic tremolo, jittering spatial position. When the grains are distributed on a large frequency region, the texture has a massive character, while when the band is quite narrow, it result a pitched sound. Sparse densities (e.g. 5 grains per second) give rise to a pointillistic texture.

2.2.3.4 Corpus-based concatenative synthesis

In recent years, synthesis methods conceptually similar to granular techniques have received a new impulse due to the availability of ever larger databases of sounds. Various definitions are used in the literature, including *concatenative synthesis*, *audio mosaicing*, and *musaicing* (neologism from music and mosaicing). All works in this direction share the general idea that a target sound can be approximated by samples taken from a pre-existing corpus of sounds.

Besides granular synthesis, the closest relative of this idea is in the the field of speech synthesis: *concatenative speech synthesis* started to develop in the early sixties and is currently the most used synthesis approach in text-to-speech systems. In short, written text is automatically segmented into

elementary phonetic units that are subsequently synthesized using a large database of sampled speech sounds. These components are pieced together to obtain a synthesis of the text.

Central point: how to properly describe both the target sound and the sounds in the database, in order to define measures of similarity. We need high-level *sound descriptors*. Sounds in the database can be segmented into units (e.g. an instrument sound can be subdivided into attack, sustain, and release portions), and some kind of *unit selection algorithm* has to be realized that finds the sequence of units that match best the target sound or phrase to be synthesized. The selection will be performed according to the descriptors of the units. The selected units can then be transformed to fully match the target specification, and are concatenated.

2.3 Time-frequency models

Consider a sound signal $x[n]$. A time-frequency representation of $x[n]$ can be seen as a series of overlapping DFTs, typically obtained by windowing x in the desired frame. More precisely, we have that a frame $X_m(e^{j\omega_k})$ of the STFT is the DFT of the signal windowed segment $x_m[n] = x[n]w_a[n - mS_a]$, where $w_a[n]$ is the chosen analysis window and S_a is the *analysis hop-size*, i.e. the time-lag (in samples) between one analysis frame and the following one. If the window w_a is N samples long, then the *block size*, i.e. the length of each frame X_m, will be N. In order for the signal segments to actually overlap, the inequality $S_a \leq N$ must be verified. When $S_a = N$ the segments are exactly juxtaposed with no overlap.

Given the above signal segmentation, OLA methods are typically used to modify and reconstruct the signal in two main steps:

1. Any desired modication is applied to the spectra (e.g. multiplying by a lter frequency response function), and modified frame spectra $Y_m(e^{j\omega_k})$ are obtained.

2. Windowed segments $y_m[n]$ of the modified signal $y[n]$ are obtained by computing the inverse DFT (IDFT) of the frames Y_m.

3. The output is reconstructed by overlapping-adding the windowed segments: $y[n] = \sum_m y_m[n]$.

In their most general formulation OLA methods utilize a *synthesis* window w_s that can in general be different from the analysis window w_a. In this case the second step of the procedure outlined above is modified as follows:

2. Windowed segments $y_m[n]$ of the modified signal $y[n]$ are obtained by ***(a)*** computing the inverse DFT (IDFT) of the frames Y_m, ***(b)*** dividing by the analysis window w_a (assuming that it is non-zero for all samples), and ***(c)*** multiplying by the synthesis window.

This approach provides greater flexibility than the previous one: the analysis window w_a can be chosen only on the basis of its time-frequency resolution properties, but needs not to satisfy the "sum-to-unity" condition $A_{w_a} \equiv 1$. On the other hand, the synthesis window w_s is only used to cross-fade between signal segments, therefore one should only ensure that $A_{w_s} \equiv 1$. We will see in section 2.4.1 an application of this technique to frequency-domain implementation of additive synthesis.

Many digital sound effects can be obtained by empolying OLA techniques. As an example, a *robotization* effect can be obtained by putting zero phase values on every FFT before reconstruction: the effect applies a fixed pitch onto a sound and moreover, as it forces the sound to be periodic, many erratic and random variations are converted into "robotic" sounds. Other effects are obtained by imposing a random phase on a time-frequency representation, with different behaviors depending on the block length N: if

N is large (e.g. $N = 2048$ with $F_s = 44.1$ kHz), the magnitude will represent the behavior of the partials quite well and changes in phase will produce an uncertainty over the frequency; if N is small (e.g. $N = 64$ with $F_s = 44.1$ kHz), the spectral envelope will be enhanced and this will lead to a whispering effect.

2.4 Spectral models

Since the human ear acts as a particular spectrum analyser, an important class of synthesis models aims at modeling and generating sound spectra. The Short Time Fourier Transform and other time-frequency representations provide powerful sound analysis tools for computing the time-varying spectrum of a given sound.

This section presents models that can be interpreted in the frequency domain. In computer music this models have ben traditionally called *additive synthesis*, since they regard a time-varying sound as a sum of sinusoidal components with time-varying amplitudes and frequencies. Note that the idea behind additive synthesis is not new. As a matter of fact, it has been used for centuries in some traditional instruments such as organs. Organ pipes, in fact, produce relatively simple sounds that, when combined together, contribute to the richer spectrum of some registers. Particularly rich registers are created by using many pipes of different pitch at the same time.

In Sec. 2.4.1 we discuss sinusoidal modeling of the deterministic sound signal and introduce the main concepts of additive synthesis. In Sec. 2.4.2 we discuss a *synthesis-by-analysis* procedure that allows to fit parameters of a sinusoidal model to a target sound. Finally. in Secs. 2.4.3 and 2.4.4 we address the problem of including stochastic components and fast transients into the framework of additive models.

2.4.1 Sinusoidal model

Spectral analysis of the sounds produced by musical instruments, or by any physical system, shows that the spectral energy of the sound signals can be interpreted as the sum of two main components: a *deterministic* component that is concentrated on a discrete set of frequencies, and a *stochastic* component that has a broadband characteristics.

The deterministic –or sinusoidal– component normally corresponds to the main modes of vibration of the system. The stochastic residual accounts for the energy produced by the excitation mechanism which is not turned into stationary vibrations by the system, and for any other energy component that is not sinusoidal.

As an example, consider the sound of a wind instrument: the deterministic signal results from self-sustained oscillations inside the bore, while the residual noisy signal is generated by the turbulent flow components due to air passing through narrow apertures inside the instrument. Similar considerations apply to other classes of instruments, as well as to voice sounds, and even to non-musical sounds.

2.4.1.1 Time-varying partials

The term *deterministic* signal means in general any signal that is not noise. The class of deterministic signals that we consider here is restricted to sums of sinusoidal components with varying amplitude and frequency. For pitched sounds in particular, spectral energy is mainly concentrated at a few discrete (slowly time-varying) frequencies f_k. These frequency lines correspond to different sinusoidal components called *partials*. The amplitude a_k of each partial is not constant and its time-variation is critical for timbre characterization. If there is a good degree of correlation among the frequency and amplitude variations of different partials, these are perceived as fused to give a unique sound with its timbre identity.

Amplitude and frequency variations can be noticed e.g. in sound attacks: some partials that are relevant in the attack can disappear in the stationary part. In general, the frequencies can have arbitrary distributions: for quasi-periodic sounds the frequencies are approximately harmonic components (integer multiples of a common fundamental frequency), while for non-harmonic sounds (such as that of a bell) they have non-integer ratios.

The deterministic part of a sound signal can be represented by the *sinusoidal model*, which assumes that the sound can be modeled as a sum of sinusoidal oscillators whose amplitude $a_k(t)$ and frequency $f_k(t)$ are slowly time-varying:

$$\begin{aligned} s_s(t) =& \sum_k a_k(t)\ \cos(\phi_k(t)), \\ \phi_k(t) =& 2\pi \int_0^t f_k(\tau) d\tau + \phi_k(0), \end{aligned} \tag{2.15}$$

or, in the discrete-time domain:

$$\begin{aligned} s_s[n] =& \sum_k a_k[n]\ \cos(\phi_k[n]), \\ \phi_k[n] =& 2\pi\ f_k[n] T_s + \phi_k[n-1], \end{aligned} \tag{2.16}$$

where T_s is the sampling period. Equation (2.16) can also be written as

$$\phi_k[n] = \phi_{0k} + 2\pi T_s \sum_{j=1}^{n} f_k[n], \tag{2.17}$$

where ϕ_{0k} represents an initial phase value. These equations have great generality and can be used to faithfully reproduce many types of sound, especially in a "synthesis-by-analysis" framework (that we discuss in Sec. 2.4.2 below). If the sound is almost periodic, the frequencies of partials are approximately multiples of the fundamental frequency f_0, i.e. $f_k(t) \simeq k\, f_0(t)$. In this sense Eqs. (2.16) are a generalization of the Fourier theorem to quasi-periodic sounds. Moreover the model is also capable of representing aperiodic and inharmonic sounds, as long as their spectral energy is concentrated near discrete frequencies (spectral lines).

As already noted, one limitation of the sinusoidal model is that it discards completely the noisy components that are always present in real signals. Another drawback of Eq. (2.16) is that it needs an extremely large number of control parameters: for each note that we want to reproduce, we need to provide the amplitude and frequency envelopes for all the partials. Moreover, the envelopes for a single note are not fixed, but depend in general on the intensity.

On the other hand, additive synthesis provides a very intuitive sound representation, and this is one of the reasons why it has been one of the earliest popular synthesis techniques in computer music.[3] Moreover, sound transformations performed on the parameters of the additive representation (e.g., time-scale modifications) are perceptually very robust.

2.4.1.2 Time- and frequency-domain implementations

Additive synthesis with equation (2.16) can be implemented either in the time domain or in the frequency domain. The more traditional time-domain implementation uses the digital sinusoidal oscillator in wavetable or recursive form, as discussed in Chapter *Fundamentals of digital audio processing*. The instantaneous

[3] Some composers have even used additive synthesis as a compositional metaphor, in which sound spectra are reinterpreted as harmonic structures.

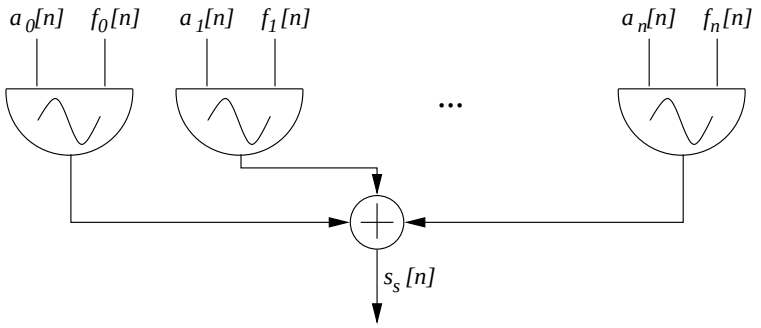


Figure 2.6: *Sum of sinusoidal oscillators with time-varying amplitudes and frequencies.*

amplitude and the instantaneous angular frequency of a particular partial are obtained by linear interpolation, as discussed there. Figure 2.6 provides a block diagram of such a time-domain implementation.

M-2.7

Use the sinusoidal oscillator realized in Chapter *Fundamentals of digital audio processing* to synthesize a sum of two sinusoids.

M-2.7 Solution

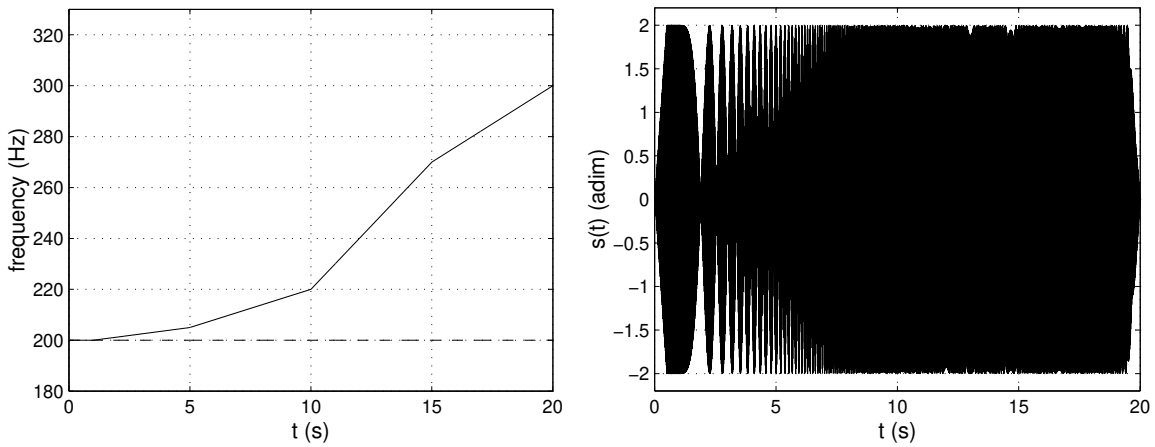
```
global Fs; global SpF; %global variables: sample rate, samples-per-frame

Fs=22050;
framelength=0.01;          %frame length (in s)
SpF=round(Fs*framelength); %samples per frame

%%%%%% define controls %%%%%%
a=envgen([0,.5,5,10,15,19.5,20],[0,1,1,1,1,1,0]); %fade in/out
f1=envgen([0,20],[200,200]);          %constant freq. envelope
f2=envgen([0,1,5,10,15,20],...        %increasing freq. envelope
         [200,200,205,220,270,300]);
%%%%%% compute sound %%%%%%
s=sinosc(0,a,f1,0)+sinosc(0,a,f2,0);
```

The sinusoidal oscillator controlled in frequency and amplitude is the fundamental building block for time-domain implementations of additive synthesis. Here we employ it to look at the beating phenomenon. We use two oscillators, of which one has constant frequency while the second is given a slowly increasing frequency envelope. Figure 2.7 shows the `f1, f2` control signals and the amplitude envelope of the resulting sound signal: note the beating effect.

In alternative to the time-domain approach, a very efficient implementation of additive synthesis can be developed in the frequency domain, using the inverse FFT. As we have seen in Chapter *Fundamentals of digital audio processing*, the DFT of a windowed sinusoid is the DFT of the window, centered at the frequency of the sinusoid, and multiplied by a complex number whose magnitude and phase are the magnitude and phase of the sine wave:

$$s[n] = a\cos(2\pi f_0 n/F_s + \phi) \quad \Rightarrow \quad \mathcal{F}[w \cdot s](f) = ae^{j\phi}W(f - f_0). \tag{2.18}$$

If the window $W(f)$ has a sufficiently high sidelobe attenuation, the sinusoid can be generated in the spectral domain by calculating the samples in the main lobe of the window transform, with the appropriate magnitude, frequency and phase values. One can then synthesize as many sinusoids as desired,



(a) (b)

Figure 2.7: *Beating effect: (a) frequency envelopes (*`f1` *dashed line,* `f2` *solid line) and (b) envelope of the resulting signal.*

by adding a corresponding number of main lobes in the Fourier domain and performing a single IFFT to obtain the corresponding time-domain signal in a frame.

By an overlap-add process one then obtains the time-varying characteristics of the sound. Note however that, in order for the signal reconstruction to be free of artifacts, the overlap-add procedure must be carried out using a window with the property that its shifted copies overlap and add to give a constant. A particularly simple and effective window that satisfies this property is the triangular window.

The FFT-based approach can be convenient with respect to time-domain techniques when a very high number of sinusoidal components must be reproduced: the reason is that the computational costs of this implementation are largely dominated by the cost of the IFFT, which does not depend on the number of components. On the other hand, this approach is less flexible than the traditional oscillator bank implementation, especially for the instantaneous control of frequency and magnitude. Note also that the instantaneous phases are not preserved using this method. A final remark concerns the FFT size: in general one wants to have a high frame rate, so that frequencies and magnitudes need not to be interpolated inside a frame. At the same time, large FFT sizes are desirable in order to achieve good frequency resolution and separation of the sinusoidal components. As in every short-time based processes, one has to find a trade-off between time and frequency resolution.

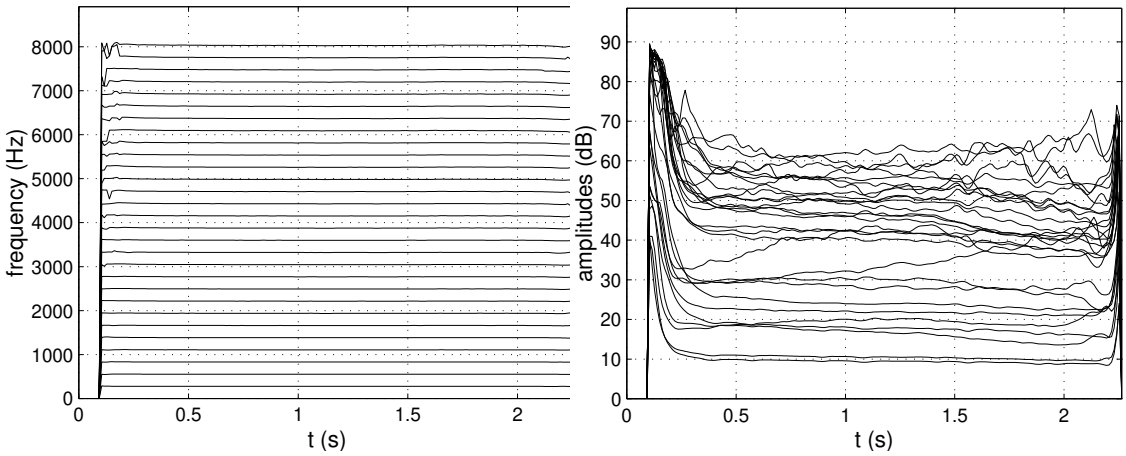
2.4.2 Synthesis by analysis

As already remarked, additive synthesis allows high quality sound reproduction if the amplitude and frequency control envelopes are extracted from Fourier analysis of real sounds. Figure 2.8 shows the result of this kind of analysis, in the case of a saxophone tone. Using these data, additive resynthesis is straightforward.

M-2.8

Assume that two matrices `sinan_freqs` and `sinan_amps` have been created from analysis of a real sound. These matrices contain frequency and amplitude envelopes of sinusoidal partials of the analyzed sound. Write a function that resynthesizes the sound.

M-2.8 Solution



(a) (b)

Figure 2.8: *Fourier analysis of a saxophone tone: (a) frequency envelopes and (b) amplitude envelopes of the sinusoidal partials, as functions of time.*

```
function s=sin_resynth(sinan_freqs, sinan_amps)

global Fs; global SpF; %global variables: sample rate, samples-per-frame

%%% define controls (analysis matrices sinan_freqs and
%%% sinan_amps have been created in the analysis phase)
npart=size(sinan_amps,1);  %number of analyzed partials
t0=0;  %initial time

%%% compute sound %%%
s=0; %initialize output signal
for (i=1:npart)                          %generate all partials and sum
   s=s+sinosc(t0,sinan_amps(i,:),sinan_freqs(i,:),0);
end
```

2.4.2.1 Magnitude and Phase Spectra Computation

The first step of any analysis procedure that tracks frequencies and amplitudes of the sinusoidal components is the frame-by-frame computation of the sound magnitude and phase spectra, through the STFT. The subsequent tracking procedure will be performed in the frequency domain. The control parameters for the STFT are the window-type and size, the FFT-size, and the frame-rate. These must be set depending on the sound to be processed.

Note that the analysis step is completely independent from the synthesis, therefore the observations made in Sec. 2.4.1 about FFT-based implementations (the window must overlap and add to a constant) do not apply here. Good resolution of the spectrum is needed in order to correctly resolve, identify, and track the peaks which correspond to the deterministic component.

If the analyzed sound is almost stationary, long windows (i.e. windows that cover several periods) that have good side-lobe rejection can be used, with a consequent good frequency resolution. Unfortunately most interesting sounds are not stationary and a trade-off has to be found. For harmonic sounds one can scale the actual window size as a function of pitch, thus achieving a constant time-frequency trade-off.

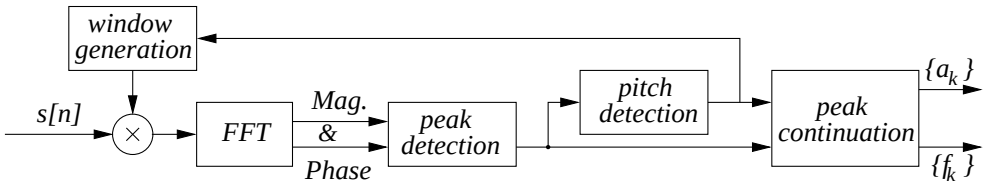


Figure 2.9: *Block diagram of the sinusoid tracking process, where $s[n]$ is the analyzed sound signal and a_k, f_k are the estimated amplitude and frequency of the kth partial in the current analysis frame.*

For inharmonic sounds the size should be set according to the minimum frequency difference that exists between partials.

The question is now how to perform automatic detection and tracking of the spectral peaks that correspond to sinusoidal components. In the next section we present the main guidelines of a general analysis framework, which is summarized in Fig. 2.9. First, the FFT of a sound frame is computed according to the above discussion. Next, the prominent spectral peaks are detected and incorporated into partial trajectories. If the sound is pseudo-harmonic, a pitch detection step can improve the analysis by providing information about the fundamental frequency information, and can also be used to choose the size of the analysis window.

Such a scheme is only one of the possible approaches that can be used to attack the problem. Hidden Markov Models (HMMs) are another one: a HMM can optimize groups of peaks trajectories according to given criteria, such as frequency continuity. This type of approach might be very valuable for tracking partials in polyphonic sounds and complex inharmonic tones.

2.4.2.2 A sinusoid tracking procedure

We now discuss in more detail the analysis steps depicted in Fig. 2.9.

The first step is *peak detection*. The most prominent frequency peaks (i.e., local maxima in the magnitude spectrum) in the current analysis frame are identified in this step. Real sounds are not periodic, do not have clearly spaced and defined spectral peaks, exhibit interactions between components. Therefore, the best that one can do at this point is to detect as many peaks as possible and postpone to later analysis steps the decision of which ones actually correspond to sinusoidal components. The peaks are then searched by only imposing two minimal constraints: they have to lie within a given frequency range, and above a given magnitude threshold. The detection of very soft peaks is hard: they have little resolution, and measurements are very sensitive to transformations because as soon as modifications are applied to the analysis data, parts of the sound that could not be heard in the original can become audible.

In ideal conditions, i.e. if the analyzed sound is very clean and has with the maximum dynamic range, the magnitude threshold can be set to the amplitude of the background noise floor. One possible strategy to decide whether a peak estimated in this way actually belongs to a sinusoidal partial or not is to measure how close the peak shape is to the ideal sinusoidal peak: the difference from the samples of the measured peak to the samples of the analysis window transform (centered at the measured frequency and scaled to the measured magnitude) provides a measure of how likely the peak is to belong to a sinusoidal component. Another refinement to the peak detection stage is to pre-process the sound in order to introduce *pre-emphasis* in the high frequency range: this allows to gain better resolution in the high frequency range. If applied, pre-emphasis will need to be compensated later on before the resynthesis.

The second step in Fig. 2.9 is *pitch detection*. Pitch is a valuable source of additional information in order to decide whether a particular peak belongs to a sinusoidal partial or not. If a fundamental

frequency is actually present, it can be exploited in two ways. First, it helps the tracking of partials. Second, the size of the analysis window can be set according to the estimated pitch in order to keep the number of periods-per-frame constant, therefore achieving the best possible time-frequency trade-off (this is an example of a *pitch-synchronous* analysis). There are many possible pitch detection strategies, which will be presented in Chapter *From audio to content*.

The third step in Fig. 2.9 is a *peak continuation* algorithm. The basic idea is that a set of "guides" advance in time and follow appropriate frequency peaks, forming trajectories out of them. A guide is therefore an abstract entity which is used by the algorithm to create the trajectories, and the trajectories are the actual result of the peak continuation process. The guides are turned on, advanced, and finally turned off during the continuation algorithm, and their instantaneous state (frequency and magnitude) is continuously updated during the process. If the analyzed sound is harmonic and a fundamental has been estimated, then the guides are created at the beginning of the analysis, with frequencies set according to the estimated harmonic series. When no harmonic structure can be estimated, each guide is created when the first available peak is found. In the successive analysis frames, the guides modify their status depending on the last peak values. This past information is particularly relevant when the sound is not harmonic, or when the harmonics are not locked to each other and we cannot rely on the fundamental as a strong reference for all the harmonics.

The main guidelines to construct a peak continuation algorithm can be summarized as follows. A peak is assigned to the guide that is closest to it and that is within an assigned frequency deviation. If a guide does not find a match, the corresponding trajectory can be turned off, and if a continuation peak is not found for a given amount of time the guide is killed. New guides and trajectories can be created starting from peaks of the current frame that have high magnitude and are not "claimed" by any of the existing trajectories. After a certain number of analysis frames, the algorithm can look at the trajectories created so far and adopt corrections: in particular, short trajectories can be deleted, and small gaps in longer trajectories can be filled by interpolating between the values of the gap edges.

One final refinement to this process can be added by noting that the sound attack is usually highly non-stationary and noisy, and the peak search is consequently difficult in this part. Therefore it is customary to perform the whole procedure backwards in time, starting from the end of the sound (which is usually a more stable part). When the attack is reached, a lot of relevant information has already been gained and non-relevant peaks can be evaluated and/or rejected.

2.4.3 "Sines-plus-noise" models

At the beginning of our discussion on additive modeling, we remarked that the spectral energy of the sound signals has a *deterministic* component that is concentrated on a discrete set of frequencies, and a *stochastic* component that has a broadband characteritics. So far we have discussed the problem of modeling the deterministic –or sinusoidal– component. Now we have to include the stochastic component into the model.

A sinusoidal representation may in principle be used also to simulate noise, since noise consists of sinusoids at every frequency within the band limits. It is clear however that such a representation would be computationally very demanding. Moreover it would not be a *flexible* sound representation. Therefore the most convenient sound model is of the form

$$s[n] = s_s[n] + e[n], \tag{2.19}$$

where $s_s[n]$ represents the deterministic part of the sound and has already been modeled with Eq. (2.16), while $e[n]$ represents the stochastic component and is modeled separately from $s_s[n]$.

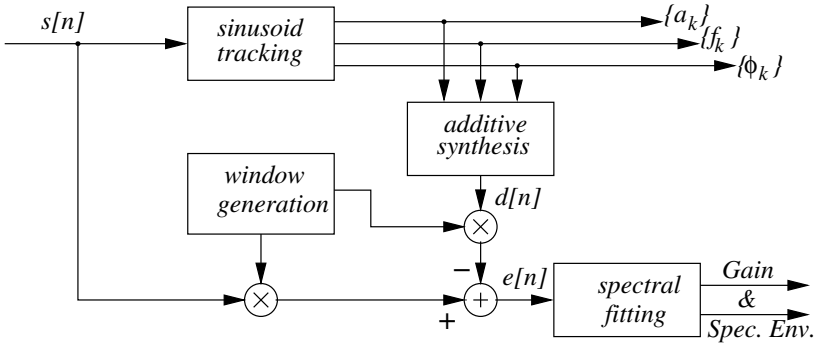


Figure 2.10: *Block diagram of the stochastic analysis and modeling process, where $s[n]$ is the analyzed sound signal and $a_k,\ f_k,\ \phi_k$ are the estimated amplitude, frequency, and phase of the kth partial in the current analysis frame.*

2.4.3.1 Stochastic analysis

The most straightforward approach to estimation of the stochastic component is through *subtraction* of the deterministic component from the original signal. Subtraction can be performed either in the time domain or in the frequency domain. Time domain subtraction must be done while preserving the phases of the original sound, and instantaneous phase preservation can be computationally very expensive. One the other hand, frequency-domain subtraction does not require phase preservation. However, time-domain subtraction provides much better results, and is usually favored despite the higher computational costs. For this reason we choose to examine time-domain subtraction in the remainder of this section. Figure 2.10 provides a block diagram.

Suppose that the deterministic component has been estimated in a given analysis frame, using for instance the general scheme described in section 2.4.2 (note however that in this case the analysis should be improved in order to provide estimates of the instantaneous phases as well). Then the first step in the subtraction process is the time-domain resynthesis of the deterministic component with the estimated parameters. This should be done by properly interpolating amplitude, frequency, and phase values in order to avoid artifacts in the resynthesized signal. The actual subtraction can be performed as

$$e[n] = w[n] \cdot [s[n] - d[n]]\,, \qquad n = 0, \ldots, N-1, \tag{2.20}$$

where $s[n]$ is the original sound signal and $d[n]$ is the re-synthesized deterministic part. The difference $(s - d)$ is multiplied by an analysis window w of size N, which deserves some discussion.

We have seen in 2.4.2 that high frequency resolution is needed for the deterministic part, and for this reason long analysis windows are used for its estimation. On the other hand, good time resolution is more important for the stochastic part of the signal, especially in sound attacks, while frequency resolution is not a major issue for noise analysis. A way to obtain good resolutions for both the components is to use two different analysis windows. Therefore w in equation (2.20) is not in general the same window used to estimate $d[n]$, and the size N is in general small.

Once the subtraction has been performed, there is one more step than can be used to improve the analysis, namely, test can be performed on the estimated residual in order to assess how good the analysis was. If the spectrum of the residual still contains some partials, then the analysis of the deterministic component has not been performed accurately and the sound should be re-analyzed until the residual is free of deterministic components. Ideally the residual should be as close as possible to a stochastic

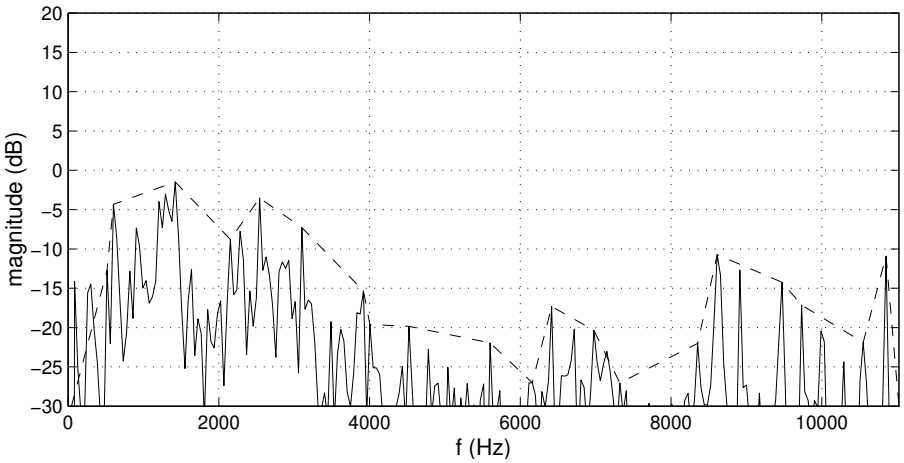


Figure 2.11: *Example of residual magnitude spectrum (solid line) and its line-segment approximation (dashed line), in an analysis frame. The analyzed sound signal is the same saxophone tone used in figure 2.8.*

signal, therefore one possible test is a measure of correlation of the residual samples.[4]

2.4.3.2 Stochastic modeling

The assumption that the residual is a stochastic signal implies that it is fully described by its amplitude and its spectral envelope characteristics. Information on the instantaneous phase is not necessary. Based on these considerations, a frame of the stochastic residual can be completely characterized by a filter that models the amplitude and general frequency characteristics of the residual. The representation of the residual for the overall sound will then be a time-varying filter.

Within a given frame we therefore assume that $e[n]$ can be modeled as

$$E[k] = H[k] \cdot U[k], \qquad k = 0, \ldots, N-1, \tag{2.21}$$

where $U[k]$ is the DFT of a white noise sequence and $H[k]$ represents the frequency response of a filter which varies on a frame-by-frame basis. The stochastic modeling step is summarized in the last block of figure 2.10.

The filter design problem can be solved using different strategies. One approach that is often adopted uses some sort of curve fitting (line-segment approximation, spline interpolation, least squares approximation, and so on) of the magnitude spectrum of e in an analysis frame. As an example, line-segment approximation can be obtained by stepping through the magnitude spectrum, finding local maxima at each step, and connecting the maxima with straight lines. This procedure can approximate the spectral envelope with reasonable accuracy, depending on the number of points, which in turn can be set depending on the sound complexity. See Fig. 2.11 for an example.

Another possible approach to the filter design problem is Linear Prediction (LP) analysis, which is a popular technique in speech processing. In this context, however, curve fitting on the noise spectrum (e.g., line-segment approximation) is usually considered to a be more flexible approach and is preferred to LP analysis. We will return on Linear Prediction techniques in section 2.5.3.

The next question is how to implement the estimated time-varying filter in the resynthesis step.

[4] Note that if the analyzed sound has not been recorded in silent and anechoic settings the residual will contain not only the stochastic part of the sound, but also reverberation and/or background noise.

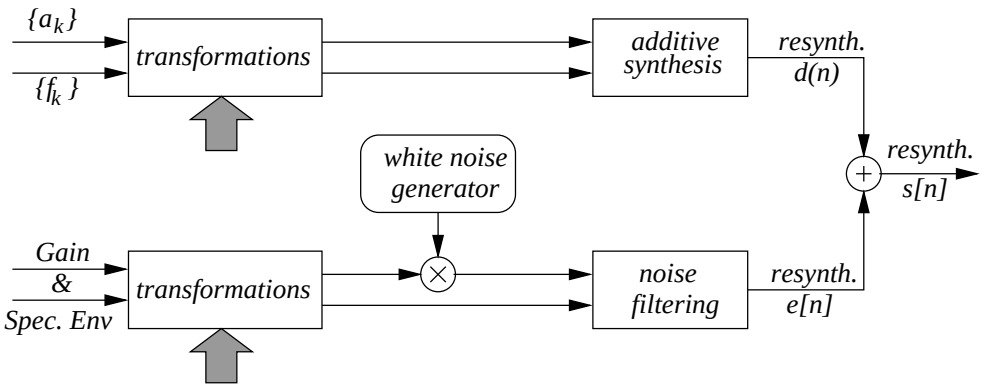


Figure 2.12: *Block diagram of the sines-plus-noise synthesis process.*

2.4.3.3 Resynthesis and modifications

Figure 2.12 shows the block diagram of the synthesis process. The deterministic signal, i.e., the sinusoidal component, results from the magnitude and frequency trajectories, or their transformation, by generating a sine wave for each trajectory (additive synthesis). As we have seen, this can either be implemented in the time domain with the traditional oscillator bank method or in the frequency domain using the inverse-FFT approach.

Concerning the stochastic component, a frequency-domain implementation is usually preferred to a direct implementation of the time-domain convolution (2.21), due to its computational efficiency[5] and flexibility. In each frame, the stochastic signal is generated by an inverse-FFT of the spectral envelopes. Similarly to what we have seen for the deterministic synthesis in section 2.4.1, the time-varying characteristics of the stochastic signal is then obtained using an overlap-and-add process.

In order to perform the IFFT, a magnitude and a phase responses have to be generated starting from the estimated frequency envelope. Generation of the magnitude spectrum is straightforwadly obtained by first linearly interpolating the spectral envelope to a curve with half the length of the FFT-size, and then multiplying it by a gain that corresponds to the average magnitude extracted in the analysis. The estimated spectral envelope gives no information on the phase response. However, since the phase response of noise is noise, a phase response can be created from scratch using a random signal generator. In order to avoid periodicities at the frame rate, new random values should be generated at every frame.

The sines-plus-noise representation is well suited for modification purposes.

- By only working on the deterministic representation and modifying the amplitude-frequency pairs or the original sound partials, many kinds of frequency and magnitude transformations can be obtained. As an example, partials can be transposed in frequency. It is also possible to decouple the sinusoidal frequencies from their amplitude, obtaining pitch-shift effects that preserve the formant structure.

- Time-stretching transformations can obtained by resampling the analysis points in time, thus slowing down or speeding up the sound while maintaining pitch and formant structure. Given the stochastic model that we are using, the noise remains noise and faithful signal resynthesis is possible even with extreme stretching parameters.

[5] In fact, by using a frequency-domain implementation for both the deterministic and the stochastic synthesis one can add the two spectra and resynthesize both the components at the cost of a sigle IFFT per frame.

- By acting on the relative amplitude of the two components, interesting effects can be obtained in which either the deterministic or the stochastic parts are emphasized. As an example, the amount of "breathiness" of a voiced sound or a wind instrument tone can be adjusted in this way. One must keep in mind however that, when different transformations are applied to the two representations, the deterministic and stochastic components in the resulting signal may not be perceived as a single sound event anymore.

- Sound morphing (or *cross-synthesis* transformations can be obtained by interpolating data from two or more analysis files. This transformations are particularly effective in the case of quasi-harmonic sounds with smooth parameter curves.

2.4.4 Sinusoidal description of transients

So far we have seen how to extend the sinusoidal model by using a "sines-plus-noise" approach that explicitly describes the residual as slowly varying filtered white noise. Although this technique is very powerful, transients do not fit well into a filtered noise description, because they lose sharpness and are smeared. This consideration motivates us to handle transients separately.

One straightforward approach, that is sometimes used, is removing transient regions from the residual, performing the sines-plus-noise analysis, and adding the transients back into the signal. This approach obviously requires memory where the sampled transients must be stored, but since the transient residuals remain largely invariant throughout most of the range of an instrument, only a few residuals are needed in order to cover all the sounds of a single instrument. Although this approach works well, it is not flexible because there is no model for the transients. In addition, identifying transients as everything that is neither sinusoidal nor transient is not entirely correct. Therefore we look for a suitable transient model, that can be embedded in the additive description to obtain a "sines-plus-transients-plus-noise" representation.

2.4.4.1 The DCT domain

In the following we adopt a further modified version of the additive sound representation (2.16), in which the sound transients are explicitly modeled by an additional signal:

$$s[n] = s_s[n] + e_t[n] + e_r[n], \tag{2.22}$$

where $s_s[n]$ is the sinusoidal component, $e_t[n]$ is the signal associated to transients and $e_r[n]$ is the noisy residual. The transient model is based on a main undelying idea: we have seen that a slowly varying sinusoidal signal is impulsive in the frequency domain, and sinusoidal models perform short-time Fourier analysis in order to track slowly varying spectral peaks (the tips of the impulsive signals) over time. Transients are very much dual to sinusoidal components: they are impulsive in the time domain, and consequently they must be oscillatory in the frequency domain. Therefore, although transient cannot be tracked by a short-time analysis (because their STFT will not contain meaningful peaks), we can track them by performing sinusoidal modeling in a properly chosen frequency domain. The mapping that we choose to use is the one provided by the discrete cosine transform (DCT):

$$S[k] = \beta[k] \sum_{n=0}^{N-1} s[n] \cos\left[\frac{(2n+1)k\pi}{2N}\right], \quad \text{for } n,k = 0,1,\ldots,N-1, \tag{2.23}$$

where $\beta[0] = \sqrt{1/N}$ and $\beta[k] = \sqrt{2/N}$ otherwise. From equation (2.23) one can see that an ideal impulse $\delta[n-n_0]$ (i.e., a Kronecker delta function centered in n_0) is transformed into a cosine whose

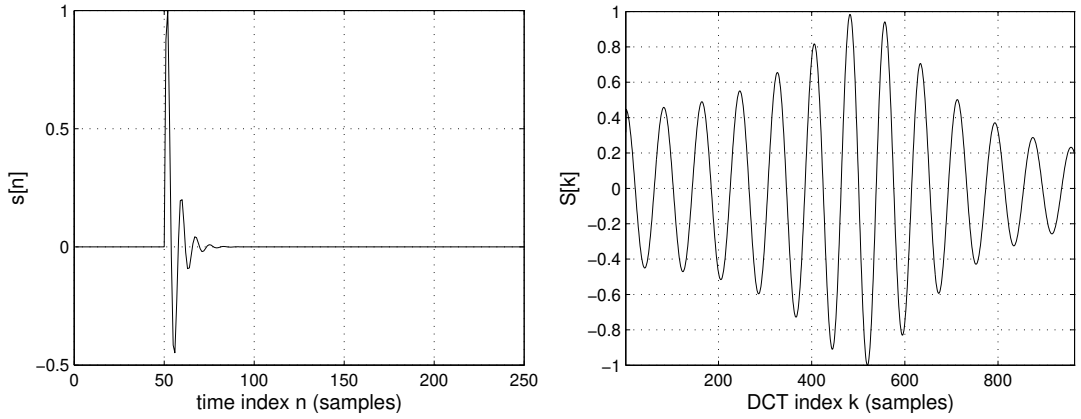


Figure 2.13: *Example of DCT mapping: (a) an impulsive transient (an exponentially decaying sinusoid) and (b) its DCT as a slowly varying sinusoid.*

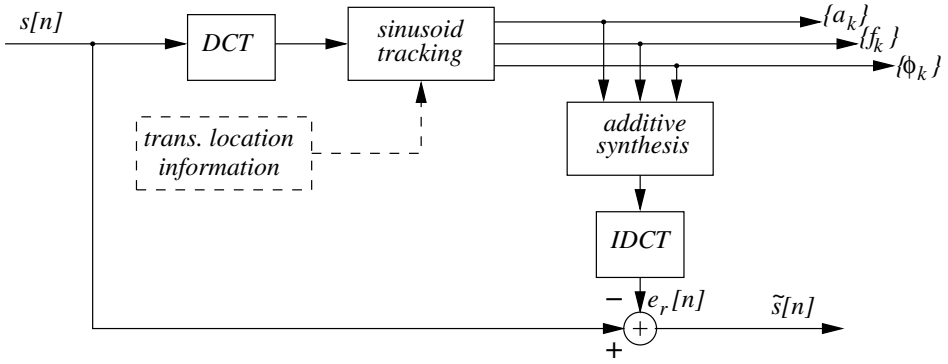


Figure 2.14: *Block diagram of the transient analysis and modeling process, where $s[n]$ is the analyzed sound signal and a_k, f_k, ϕ_k are the estimated amplitude, frequency, and phase of the kth DCT-transformed transient in the current analysis frame.*

frequency is monotonically related to n_0. Figure 2.13(a) shows a more realistic transient signal, a one-sided exponentially decaying sine wave. Figure 2.13(b) shows the DCT of the transient signal: a slowly varying sinusoid. This considerations suggest that the time-frequency duality can be exploited to develop a transient model: the same kind of parameters that characterize the sinusoidal components of a signal can also characterize the transient components of a signal, although in a different domain.

2.4.4.2 Transient analysis and modeling

Having transformed the transient into the DCT domain, the most natural way to proceed is performing sinusoidal modeling in this domain: STFT analysis of the DCT-domain signal can be used to find meaningful peaks, and then the signal can be resynthesized in the DCT domain and back-transformed to the time domain with an inverse DCT transform (IDCT). This process is shown in figure 2.14. We now discuss the main steps involved in this block diagram.

First the input signal $s[n]$ is divided into non-overlapping blocks in which DCT analysis will be per-

formed. The block length should be chosen so that a transient appears as "short", therefore large block sizes (e.g., 1 s) are usually chosen. The block DCT is followed by a sinusoidal analysis/modeling process which is identical to what we have seen is section 2.4.2. The analysis can optionally embed some information about transient location within the block: there are many possible transient detection strategies, which we do not want to discuss here. Also, the analysis can perform better if the sinusoid tracking procedure starts from the end of the DCT-domain signal and moves backwards toward the beginning, because the beginning of a DCT frame is usually spectrally rich and this can deteriorate the performance of the analysis (similar considerations were made in Sec. 2.4.2 when discussing sinusoid tracking in the time domain).

The analysis yields parameters that correspond to slowly varying sinusoids in the DCT domain: each transient is associated to a triplet $\{a_k, f_k, \phi_k\}$, amplitude, frequency, and phase of the kth "partial" in each STFT analysis frame within a DCT block. By recalling the properties of the DCT one can see that f_k correspond to onset locations, a_k is the amplitude of the time-domain signal also, and ϕ_k is related to the time direction (positive or negative) in which the transient evolves. Resynthesis of the transients is then performed using these parameters to reconstruct the sinusoids in the DCT domain. Finally an inverse discrete cosine transform (IDCT) on each of the reconstructed signals is used to obtain the transients in each time-domain block, and the blocks are concatenated to obtain the transients for the entire signal.

It is relatively straightforward to implement a "fast transient reconstruction" algorithm. Without entering the details, we just note that the whole procedure can be reformulated using FFT transformations only: in fact one could verify that the DCT can be implemented using an FFT block plus some post-processing (multiplication of the real and imaginary parts of the FFT by appropriate cosinusoidal and sinusoidal signals followed by a sum of the two parts). Furthermore, this kind of approach naturally leads to a FFT-based implementation of the additive synthesis step (see Sec. 2.4.1).

One nice property of this transient modeling approach is that it fits well within the sines-plus-noise analysis examined in the previous sections. The processing block depicted in Fig. 2.14 returns an output signal $\tilde{s}[n]$ in which the transient components e_t have been removed by subtraction: this signal can be used as the input to the sines-plus-noise analysis, in which the remaining components (deterministic and stochastic) will be analyzed and modeled. From the implementation viewpoint, one advantage is that the core components of the transient-modeling algorithm (sinusoid tracking and additive resynthesis) are identical to those used for the deterministic model. Therefore the same processing blocks can be used in the two stages, although working on different domains.

2.5 Source-filter models

Some sound signals can be effectively modeled through a feed-forward source-filter structure, in which the source is in general a spectrally rich excitation signal, and the filter is a linear system that acts as a resonator and shapes the spectrum of the excitation.

A typical example is *voice*, where the periodic pulses or random fluctuations produced by the vocal folds are filtered by the vocal tract, that shapes the spectral envelope. The vowel quality and the voice color greatly depends on the resonance regions of the filter, called *formants*. In computer music, source-filter models are traditionally grouped under the label *subtractive synthesis*. A number of analog voltage controlled synthesizers in the 1960's and 1970's made use of subtractive synthesis techniques in which audio filters were applied to spectrally rich waveforms.

Source-filter models are often used in an analysis-synthesis framework, in which both the source signal and the filter parameters are estimated from a target sound signal, that can be subsequently resynthesized through the identified model. Moreover, transformations can be applied to the filter and/or the excitation before the reconstruction (see Fig. 2.15). One of the most common analysis techniques is

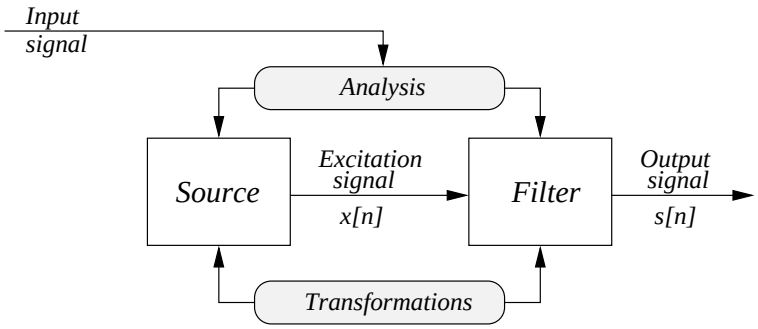


Figure 2.15: *Source-filter model.*

Linear Prediction, that we will address in Sec. 2.5.3.

2.5.1 Source signals and filter structures

We will assume the filter block to be linear and time-invariant (at least on a short-time scale), so that the excitation signal $x[n]$ and the output signal $s[n]$ are related through the difference equation

$$s[n] = \sum_{k=0}^{M} b_k x[n-k] - \sum_{k=1}^{N} a_k s[n-k] \ , \tag{2.24}$$

or, in the z-domain,

$$S(z) = H(z)X(z), \qquad \text{with} \quad H(z) = \frac{B(z)}{A(z)} = \frac{\sum_{k=0}^{M} b_k z^{-k}}{\sum_{k=0}^{N} a_k z^{-k}}. \tag{2.25}$$

Equation (2.25) shows how the features of source and filter are combined: the spectral fine structure of the excitation signal is multiplied by the spectral envelope of the filter, which has a *shaping* effect on the source spectrum. Therefore, it is possible to control and modify separately different features of the signal: as an example, the pitch of a speech sound depends on the excitation and can be controlled separately from the formant structure, which instead depends on the filter. When the filter coefficients are (slowly) varied over time, the frequency response $H\left(e^{j\omega_d}\right)$ changes. As a consequence, the output will be a combination of temporal variations of the input and of the filter (*cross-synthesis*).

2.5.1.1 Source signals

In order for the shaping effect of the filter to take place, the source signal must have a rich spectrum, that extends to a relevant portion of the audible frequency range. One important family of source signals are noise signals (see Chapter *Fundamentals of digital audio processing*), which have broadband spectral energy. One second important family are non-smooth periodic waveforms, whose spectral energy is concentrated in a (large) set of discrete spectral lines. This latter family includes square waves, sawtooth waves, and triangle waves, among others. Analog voltage-controlled synthetizers were typically equipped with a set of oscillators, which were able to synthesize these and possibly other waveforms.

An ideal *square wave* alternates periodically and instantaneously between two levels. An ideal *triangular wave* alternates periodically between a linearly rising portion and a linearly decreasing portion. An ideal *sawtooth wave* is a periodic series of linear ramps. Various formal definition of these signals (with

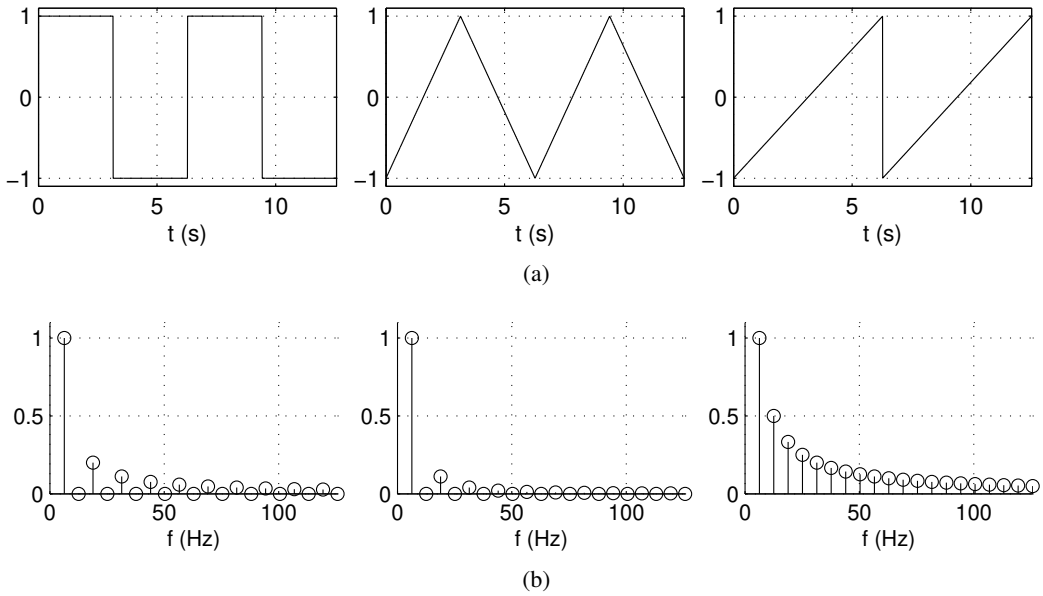


Figure 2.16: *Spectrally rich waveforms: (a) time-domain square, triangular, sawtooth, and impulse train waveforms; (b) corresponding spectra (first* 20 *partials).*

frequency f_0 Hz) can be conveniently given in the continuous-time domain.[6] A set of compact possible definitions is the following:

$$\begin{aligned} x_{\text{square}}(t) &= \operatorname{sgn}\left[\sin(2\pi f_0 t)\right], \quad x_{\text{triang}} = \frac{2}{\pi}\arcsin\left[\sin(2\pi f_0 t)\right] - 1, \\ x_{\text{saw}}(t) &= 2\left(f_0 t - \lfloor f_0 t \rfloor\right) - 1, \end{aligned} \tag{2.26}$$

where $\lfloor a \rfloor = \max\{n \in \mathbb{N} : n \leq a\}$ in the definition of the sawtooth wave indicates the floor function. These equations define waveforms that take values in the range $[-1, 1]$ and have zero average. The corresponding waveforms are depicted in Fig. 2.16(a).

These waveforms can be written in terms of the followin sinusoidal expansions:

$$\begin{aligned} x_{\text{square}}(t) &= \frac{4}{\pi}\sum_{k=1}^{+\infty}\frac{\sin\left[(2k-1)2\pi f_0 t\right]}{2k-1}, \quad x_{\text{triang}}(t) = \frac{8}{\pi^2}\sum_{k=1}^{+\infty}(-1)^{k-1}\frac{\sin\left[(2k-1)2\pi f_0 t\right]}{k^2}, \\ x_{\text{saw}}(t) &= -\frac{2}{\pi}\sum_{k=1}^{+\infty}\frac{\sin\left(2\pi k f_0 t\right)}{k}. \end{aligned} \tag{2.27}$$

Note that here we are using an expansion on real sinusoids, which can be straightforwardly derived from the usual Fourier expansion on complex sinusoids. The corresponding spectra are shown in Fig. 2.16(b). As expected, all the waveforms are spectrally rich. In particular, the square and triangular waves contain only odd harmonics, with higher harmonics rolling off faster in the triangular than in the square wave (this is in accordance with the triangular wave being –and sounding– smoother than the square wave). On the other hand, the sawtooth wave has energy on all harmonics.

One more relevant source signal is the ideal *impulse train waveform*, a sequence of unit impulses spaced by the desired fundamental period. It is used especially for the simulation of voiced speech

[6] In fact continuous-time domain definitions are appropriate since they were used in analog synthesizers.

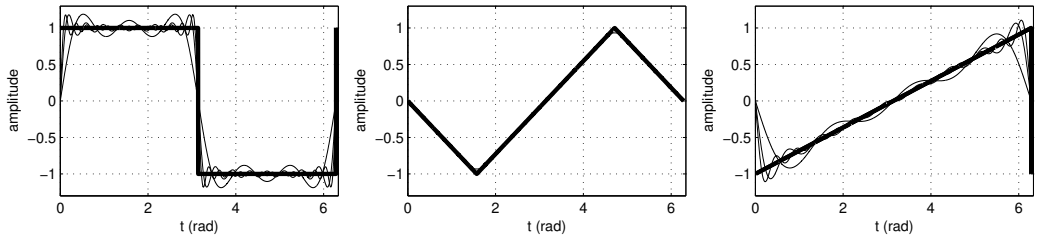


Figure 2.17: *Bandlimited synthesis of the square, triangular, and sawtooth waves, using* 3*,* 8*, and* 13 *sinusoidal components.*

sounds, and represents the periodic energy pulses provided by vocal fold movement to the vocal tract (we will return on this point in Sec. 2.5.2). It has a white spectrum.

2.5.1.2 Bandlimited digital oscillators

We have just seen that the spectra of square, triangular, and sawtooth waveforms are not bandlimited. For this reason, realizing digital oscillators able to synthesize these waveforms is not a completely trivial task because of potential aliasing problems. In fact, a simple wavetable implementation in which the waveforms are sampled from their theoretical definitions (2.26) would produce digital oscillators that are heavily affected by aliasing.

Spectral modifications introduced by aliasing deteriorate severely the sound quality, by introducing inharmonicity, beatings, and heterodyning. Inharmonicity is caused by the fact that aliased components will in general fall outside the harmonic series. In particular, some aliased components may have a smaller frequency than the fundamental frequency, thus altering the pitch. Some aliased components may instead fall in the vicinity of a harmonic component, thus generating beating. The phenomenon of heterodyning appears when the fundamental frequency of the oscillator is varied (e.g. in a vibrato): in this case the frequencies of some aliased components will move in the opposite direction with respect to the fundamental frequency, causing a characteristic timbral modulation.

It is therefore necessary to find aliasing-free implementations of such digital oscillator. The most straightforward approach amounts to exploiting the sinusoidal expansions given in Eqs. (2.27), in which only the partials with frequencies less than $F_s/2$ are employed. This is then a form of *bandlimited* additive synthesis, in which the number of sinusoidal components is chosen on the basis of the sampling rate.

Figure 2.17 shows an example of bandlimited additive synthesis in which different numbers of components have been used. Note that a low number of components already approximates closely the triangular wave, since this is a continuous function, while the square and sawtooth waves need higher number of components since they are discontinuous.

M-2.9

Write a function that realizes the generators for the square, triangular, sawtooth, and impulse train signals. The function will have parameters `(t0,a,f)`: initial time, amplitude envelope, and frequency envelope.

M-2.9 Solution

```
function s = waveosc(t0,a,f,ph0,wavetype,npart);
%%%% param wavetype can be 'cos' | 'square' | 'triang' | 'saw' | 'imp' %%%%

global Fs; global SpF;  %global variables: sample rate, samples-per-frame
```

```
npart = min(npart, floor(Fs/(2*max(f)))); % max. no of partials up to Nyquist
nframes=length(a); s=zeros(1,nframes*SpF); %initialize signal vector to 0
switch (wavetype)
  case {'cos'};
  s=sinosc(t0,a,f,ph0);
  case {'square'};
  for (k=1:npart) s=s+4/pi*sinosc(t0,a,(2*k-1)*f,ph0-pi/2)/(2*k-1); end
  case {'triang'};
  for (k=1:npart) s=s+8/pi^2*(-1)^k*sinosc(t0,a,(2*k-1)*f,ph0-pi/2)/(2*k-1)^2; end
  case {'saw'};
  for (k=1:npart) s=s-2/pi*sinosc(t0,a,k*f,ph0-pi/2)/k; end
  case {'imp'};
  for (k=1:npart) s= s + sinosc(t0,a,k*f,ph0)/(1+npart); end
end
```

This function utilizes the `sinosc` function written in Chapter *Fundamentals of digital audio processing*. We have used Eqs. (2.27) and for each waveform have summed up all the needed harmonic components up to the Nyquist frequency $F_s/2$.

2.5.1.3 Resonant filters

In Chapter *Fundamentals of digital audio processing* we have already examined some simple first-order low-pass and high-pass filters: these may be used in the "filter" block of Fig. 2.15. Another class of filters that is widely used in subtractive synthesis schemes is that of resonant filters. The second-order IIR filter is the simplest one, and is described by a transfer function with two complex conjugate poles:

$$H(z) = \frac{b_0}{1 + a_1 z^{-1} + a_2 z^{-2}} = \frac{b_0}{(1 - r \cdot e^{j\omega_c} z^{-1})(1 - r \cdot e^{-j\omega_c} z^{-1})}, \tag{2.28}$$

where r and $\pm\omega_c$ are the magnitude and phases of the poles, and the condition $r < 1$ must hold in order for the filter to be stable. If assume the filter to be causal then the impulse response is[7]

$$h[n] = \frac{b_0 r^n \sin[\omega_c(n+1)]}{\sin \omega_c} u[n], \tag{2.29}$$

where $u[n]$ is the unit step as usual. Therefore $h[n]$ is a right-sided, exponentially damped sinusoid, where r determines the decay time. An example of sequence $h[n]$ is shown in Fig. 2.18(a).

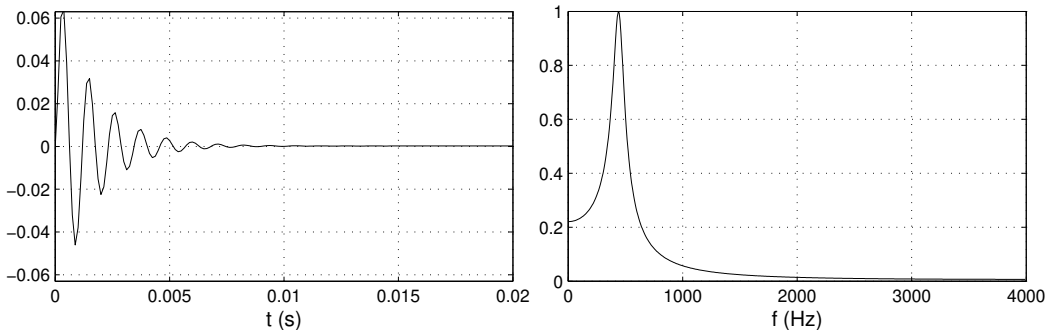
A pole $r \cdot e^{j\omega_c}$ causes a resonance to appear in the vicinity of ω_c, which becomes sharper and sharper as $r \to 1$. In order to choose appropriate parameter values for the filter, the first element to consider is the actual location ω_c of the *center frequency*, where the resonance occurs: it is close to ω_c but not exactly there. The resonance occurs when the first derivative of $\left| H\left(e^{j\omega_d}\right) \right|$ goes to zero. By writing explicitly this condition (or, more conveniently, the equivalent $d/d\omega_d[1/\mid H \mid^2] = 0$), some straightforward algebra leads to the result

$$\cos \omega_c = \frac{1 + r^2}{2r} \cos \omega_c. \tag{2.30}$$

This equation provides a means to choose ω_c given ω_c. It says that ω_c is close to ω_c when r is close to 1, but becomes significantly different for small r.

The second element to consider is the sharpness of the resonance. A quantitative measure is the *half-power bandwidth* (or simply bandwidth) B of the resonance, defined as the width of the magnitude

[7] This equation can be derived by applying the partial fraction expansion technique seen in Chapter *Fundamentals of digital audio processing*.



(a) (b)

Figure 2.18: *Example of a second-order resonator tuned on the center frequency $\omega_c = 2\pi 440/F_s$ and with bandwidth $B = 2\pi 100/F_s$; (a) impulse response; (b) magnitude response.*

response at the two half-power points (the points where the magnitude response is $1/\sqrt{2}$ times smaller than at the peak value). An equivalent measure is the *quality factor* $q = \omega_c/B$.

A half-power point ω_{hp} is such that $\left| H\left(e^{j\omega_{hp}}\right)/H\left(e^{j\omega_c}\right)\right|^2 = 1/2$, by definition. If we assume that in the vicinity of one pole the effect of the second pole is negligible, we can derive a simple relation between B and r:

$$\begin{aligned} \left| H\left(e^{j\omega_c}\right)\right|^2 &\sim \frac{b_o^2}{(1-r)^2}, \\ \left| H\left(e^{j\omega_{hp}}\right)\right|^2 &\sim \frac{b_o^2}{\left| e^{j\omega_{hp}} - re^{j\omega_c}\right|^2} = \ldots = \frac{b_o^2}{1+r^2-2r\cos\left(\omega_{hp}-\omega_c\right)}. \end{aligned} \tag{2.31}$$

By noting that $\omega_{hp} - \omega_c \sim B/2$, and by applying the definition of half-power point, we can write

$$\frac{(1-r)^2}{1+r^2-2r\cos\left(B/2\right)} = \frac{1}{2}, \quad \Rightarrow \quad \ldots \quad \Rightarrow \quad \cos\left(\frac{B}{2}\right) = 2 - \frac{1}{2}\left(r + \frac{1}{r}\right). \tag{2.32}$$

This latter equation provides a means to choose r given B. A further useful approximation can be obtained for very sharp resonances, i.e. when $r = 1-\epsilon$ with $\epsilon \ll 1$ and the poles are very close to the unit circle. Taylor expansions of the two sides of the equation give $\cos(B/2) \sim 1 - (B/2)^2$ and $2 - 1/2(r - 1/r)|_{r=(1-\epsilon)} \sim 1 - \epsilon^2/2$, respectively. Therefore in this limit one can write

$$B \sim 2\epsilon = 2(1-r). \tag{2.33}$$

In summary, given two values for ω_c and B, the poles can be determined using Eqs. (2.30) and (2.32) or (2.33). Then the coefficients can be written as functions of the parameters $r,\ \omega_c$ as

$$a_1 = -2r\cos(\omega_c), \quad a_2 = r^2, \quad b_0 = (1-r^2)\sin^2(\omega_c), \tag{2.34}$$

where b_0 has been determined by imposing that $\left| H\left(e^{\pm j\omega_c}\right)\right| = 1$. An example of magnitude response is shown in Fig. 2.18(b).

M-2.10

Write a function that computes the coefficients of a second-order resonant filter, given the normalized angular frequency ω_c (in radians) and the bandwidth B.

M-2.10 Solution

```
function [b,a]=reson2(omegac,B); %omegac and B are given in radians

r=(2-cos(B/2)) - sqrt( (2-cos(B/2))^2 -1); % <== cos(B/2)=2-1/2*(r-1/r)
omega0= acos( 2*r/(1+r^2)*cos(omegac) );

a0 =1;  a1=-2*r*cos(omega0);  a2=r^2;
b0 =(1-r^2)*( sin(omega0) );
a=[a0 a1 a2];
b=[b0 zeros(1,2)];
```

We have followed the Octave/Matlab convention in defining the coefficients `b, a`, but not the convention in defining normalized frequencies.

2.5.1.4 Subtractive synthesis of acoustic sounds

Subtractive synthesis has been mostly used for the generation of synthetic sounds which have no specific resemblance to acoustic instrumental sounds. However a few examples can be made.

We have already seen an example of subtractive synthesis when discussing spectral models: the stochastic component has been modeled as white noise passed through a time-varying filter. In this way noisy signals, like aeroacoustic noise produced by turbulent flow, can be effectively simulated.

Some of the source waveforms that we have examined previously in this section are already qualitatively similar to some instrumental sounds. As an example, the sawtooth wave has some similarity with the steady-state waveform produced by a bowed string, due to string-bow interaction mechanism. Consequently a simple subtractive synthesis scheme based on low-pass filtering of a sawtooth wave, and with the addition of suitable amplitude envelopes, can produce violin-like or cello-like sounds. Analogously, the spectrum of a clarinet sound is qualitatively similar to that of a square wave, since a cylindrical acoustic bore closed at one end and open at the other one supports all the odd harmonics of the fundamental frequency. Consequently the square wave is a good starting point for the subtractive synthesis of clarinet sounds.

One further example is modal synthesis of percussive sounds. A set of N second-order resonant filters R_i ($i = 1 \dots N$) of the form (2.28) can be grouped into a filterbank, where the same excitation signal x is fed to all the R_i's, as depicted in Fig. 2.19. This specific source-filter structure is well suited to simulate percussive sounds.

In this case the excitation signal has an impulsive characteristics and represents a "striker" that a hammer or a mallet impart to a resonating object. Suitable "striker" excitation signals are e.g. a square impulse or a noise burst. The filter block represents the resonating object hit by the hammer: the center frequencies f_{ci} ($i = 1 \dots N$) of the resonant filters can be chosen to match a desired spectrum. As an example, a string will have a harmonic spectrum in which partials are regularly spaced on the frequency axis, therefore $f_{ci} = if_{c1}$ ($i = 2 \dots N$), where f_{c1} acts as the fundamental frequency. On the other hand, the partial distribution of a bar, or a bell, or the circular membrane of a drum, will be inharmonic. As an example, it is known that the partials of an ideal bar clamped at one end are approximately $f_{c2} \sim (2.998/1.994)^2 f_{c1}$, $f_{ci} \sim [(2i+1)/1.994]^2 f_{c1}$ ($i = 3 \dots N$).

The bandwidths B_i of the R_i's determine the decay characteristics of each partial.A first possible choice is setting the same B (i.e. the same parameter r) for every filter. An alternative choice, that better describes the behavior of such resonant objects as strings, bars, and so on, amounts to setting the same *quality factor* $Q = B_i/f_{ci}$ for all the filters.

The structure depicted in Fig. 2.19 is also an example of *modal synthesis*. We will return on modal synthesis in Chapter *Sound modeling: source based approaches*, and will provide more physically sound foundations

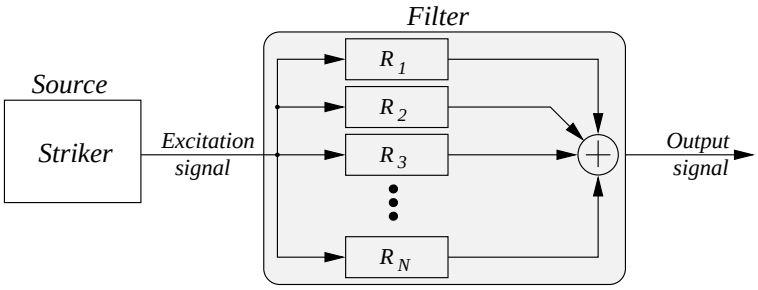


Figure 2.19: *Parallel structure of digital resonators for the simulation of struck objects – the R_i's have transfer functions of the form (2.28).*

to this sound synthesis technique.

M-2.11

Write a function `modalosc(t0,tau,a,omegac,B)` that realizes the structure of Fig. 2.19. The parameter `t0` is the total duration of the sound signal, `tau,a` define duration and max. amplitude of the striker signal (e.g. a noise burst), and the vectors `omegac,B` define center frequencies and bandwidths of a bank of second-order oscillators.

2.5.2 Voice modeling

2.5.2.1 Voice production mechanism and models

Voice is an acoustic pressure wave created when air is expelled from the lungs through the trachea and vocal tract (see Fig. 2.20). The vocal tract starts at vocal fold opening (the glottis), and includes throat, nose, mouth and lips. As the acoustic wave passes through the vocal tract, its spectrum is altered by the resonances of the vocal tract (the *formants*).

Two basic types of sounds characterize speech, namely *voiced* and *unvoiced* sounds. Voiced sounds (vowels or nasals) result from a quasi-periodic excitation of the vocal tract caused by oscillation of the vocal folds in a quasi-periodic fashion. On the other hand, unvoiced sounds do not involve vocal fold oscillations and are typically associated to turbulent flow generated when air passes through narrow restrictions of the vocal tract (e.g. fricatives). In light of this description it is reasonable to describe voiced and unvoiced sounds as the effect of an acoustic filter (the vocal tract) applied to a source signal (the acoustic flow).

During voiced signals, the vocal folds oscillate in a very non-sinusoidal fashion. The quasi-periodic nature of the oscillations gives rise to an harmonic signal, and the frequency associated with the first harmonic partial is commonly termed the pitch of the voiced signal. The range of potential pitch frequencies can vary approximately from 50 Hz to 250 Hz for adult males, and from 120 Hz to 500 Hz for adult females. The frequency varies from speaker to speaker as well: every speaker has a "preferred pitch", which is used naturally on the average. Additionally pitch is shifted up and down in speaking in response to factors relating to prosody and intonation (the pitch contour over time signals grammatical structure), but also stress and emotion.

Synthesized voice can be produced by several different approaches, all of which have some benefits and deficiencies. The methods are usually classified into three groups: *concatenative* synthesis, *formant* synthesis, and *articulatory* synthesis.

Concatenative synthesis uses pre-recorded samples of basic phonetic units, derived from natural speech. This techique is resemblant of the methods discussed in Sec. 2.2. Connecting pre-recorded

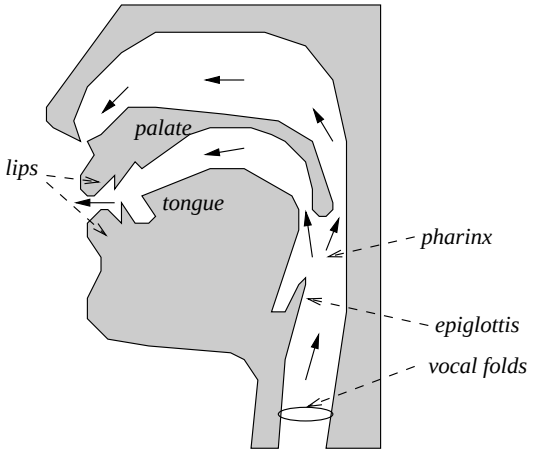


Figure 2.20: *A schematic view of the phonatory system. Solid arrows indicates the direction of the airflow generated by lung pressure.*

natural utterances is probably the easiest way and the most popular approach to produce intelligible and natural sounding synthetic speech. However, concatenative synthesizers are usually limited to one speaker and one voice and usually require more memory capacity than other methods.

Formant synthesis is based on the source-filter modeling approach described in Sec. 2.5 above: the transfer function of the vocal tract is typically represented as a series of resonant filters, each accounting for one formant. This was the most widely used synthesis method before the development of concatenative methods. Being based on a parametric model rather than on pre-recorded sounds, formant synthesis techniques are in principle more flexible than concatenative methods. We discuss formant synthesis in the next section.

Articulatory synthesis attempts to model the human voice production system directly and therefore belongs to the class of models discussed in Chapter *Sound modeling: source based approaches*. Articulatory synthesis typically involves models of the vocal folds, the vocal tract, and an associated set of *articulators* that define the area function between glottis and mouth. Articulators can be lip aperture, lip protrusion, tongue tip height and position, etc. Parameters associated to vocal folds can be glottal aperture, fold tension, lung pressure, etc. Although these methods promise high quality synthesis, computational costs are high and parametric control is arduous. At the time of writing no existing articulatory synthesizer can compare with a concatenative synthesizer.

2.5.2.2 Formant synthesis

Formant synthesis of voice realizes a source-filter model in which a broadband source signal undergoes multiple filtering transformations that are associated to the action of different elements of the phonatory system. Depending on whether voiced or unvoiced speech (see above) has to be simulated, two different models are used.

If $s[n]$ is a voiced signal, it can be expressed in the z-domain through the following cascaded spectral factors:

$$S(z) = g_v X(z) \cdot [G(z) \cdot V(z) \cdot R(z)], \tag{2.35}$$

x where the source signal $X(z)$ is a periodic pulse train whose period coincides with the pitch of the signal, g_v is a constant gain term, $G(z)$ is a filter associated to the response of the glottis (the vocal folds) to pitch pulses, $V(z)$ is the vocal tract filter, and $R(z)$ simulates the radiation effect of the lips.

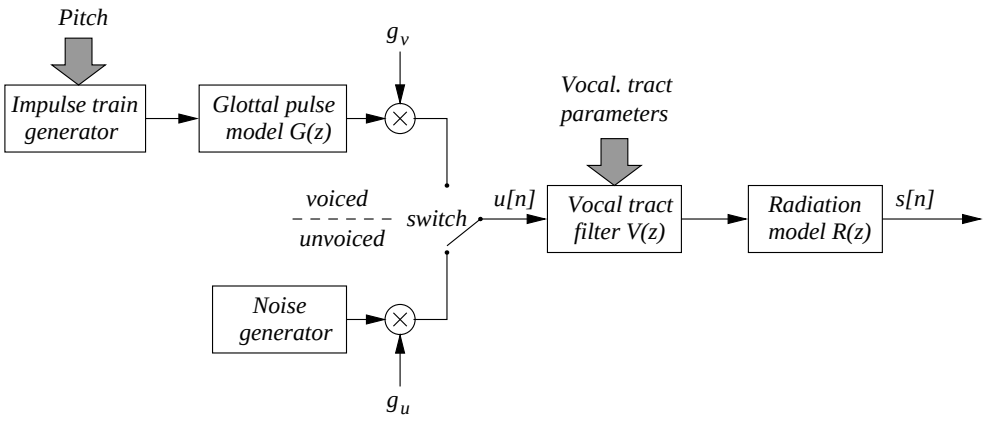


Figure 2.21: *A general model for formant synthesis of speech.*

If $s[n]$ is an unvoiced signal, vocal folds do not vibrate and turbulences is produced by the passage of air through a narrow constriction (such as the teeth). The turbulence can be modeled as white noise. In this case, the model is expressed in the Z-domain as

$$S(z) = g_u X(z) \cdot [V(z) \cdot R(z)], \tag{2.36}$$

where the source signal $X(z)$ is in this case a white noise sequence, while the gain term g_u is in general different from the voiced configuration gain, g_v. Note that the vocal fold response $G(z)$ is not included in the model in this case.

Any voiced or unvoiced sound is modeled by either Eq. (2.35) or (2.36). The complete transfer function $H(z) = S(z)/X(z)$ may or may not include vocal fold response $G(z)$ depending on whether the sound is voiced or unvoiced. The block structure of the resulting model is shown in Fig. 2.21.

The filter $G(z)$ shapes the glottal pulses. More specifically, since the input $x[n]$ is a pulse train, the output from this block is the impulse response $g[n]$ of this filter. We propose two historically relevant models. The first one is a FIR model with impulse response:

$$g_{\text{FIR}}[n] = \begin{cases} \frac{1}{2}\left[1 - \cos\left(\frac{\pi n}{N_1}\right)\right], & 0 \le n \le N_1, \\ \cos\left(\frac{\pi(n-N_1)}{2N_2}\right), & N_1 \le n \le N_1 + N_2, \\ 0 & \text{elsewhere.} \end{cases} \tag{2.37}$$

The second one is an IIR model with transfer function

$$G_{\text{IIR}}(z) = \frac{1}{[1 - \exp(-c/F_s)z^{-1}]^2}. \tag{2.38}$$

It represents a low-pass filter with one real pole of order 2, which integrates the pulse train.

The radiation filter $R(z)$ is a load that converts the airflow signal at the lips into an outgoing pressure wave (the signal $s[n]$). Under very idealized hypothesis, $R(z)$ can be approximated at least for low frequencies by a fixed differentiator:

$$R(z) = 1 - \rho z^{-1}, \tag{2.39}$$

where ρ is a lip radiation coefficient whose value is very close to 1.

The vocal tract filter $V(z)$ models vocal tract formants. A single formant can be modeled with a two-pole resonator (see Eq. (2.28)) which enables both the formant frequency and its bandwidth to be

specified. We denote the filter associated to the ith formant as $V_i(z)$, having center frequency f_i and bandwidth B_i. At least three vocal tract formants are generally required to produce intelligible speech and up to five formants are needed to produce high quality speech.

Two basic structures, parallel and cascade, can be used in general, but for better performance some kind of combination of these is usually adopted. A cascade formant synthesizer consists of band-pass resonators connected in series, i.e. the output of each formant resonator is applied to the input of the following one. A parallel formant synthesizer consists of resonators connected in parallel, i.e. the same input is applied to each formant filter and the outputs are summed. The corresponding vocal tract models are

$$V_{\text{casc}}(z) = g \prod_{i=1}^{K} V_i(z), \qquad V_{\text{par}}(z) = \sum_{i=1}^{K} a_i \cdot V_i(z). \tag{2.40}$$

The cascade structure needs only formant frequencies as control information. The main advantage of this structure is that the relative formant amplitudes for vowels do not need individual controls. A cascade model of the vocal tract is considered to provide good quality in the synthesis of vowels, but is less flexible than a parallel structure, which enables controlling of bandwidth and gain for each formant individually.

M-2.12

Using the functions `waveosc` and `reson2`, realize a parallel formant synthesizer. Use 3 second-order IIR cells, corresponding to the first 3 vowel formants.

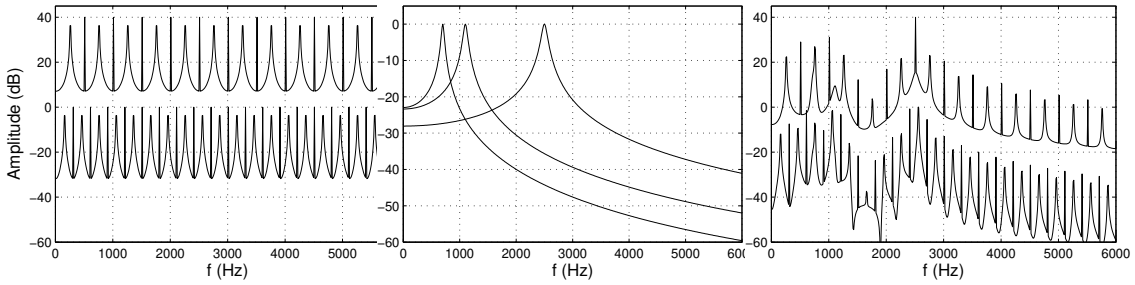
M-2.12 Solution

```
function s= formant_synth(a,f,vowel);

global Fs; global SpF;

%the vowel can be 'a' or 'e' or 'i'
if (vowel=='a')          fc=[700 1100 2500]; B=[50 75 100];
elseif (vowel=='e')      fc=[500 1850 2500]; B=[50 75 100];
elseif (vowel=='i')      fc=[300 2400 2500]; B=[50 75 100];
else error('Wrong vowel!'); end

%%% construct formant filters (numerators and denominators)
num=zeros(length(fc),3); den=zeros(length(fc),3);
for i=1:length(num)
    [num(i,:), den(i,:)] = reson2(2*pi/Fs*fc(i), 2*pi/Fs*B(i));
end

%%% compute sound %%%
x=waveosc(0,a,f,0,'imp',30);   %impulse train source
s=zeros(1,length(x));
for i=1:size(num,2) s= s + filter(num(i,:),den(i,:),x); end
```

Figure 2.22 shows an example of formant synthesis using a parallel formant filtering structure. In particular Fig. 2.22(c) shows that when the same vowel is uttered with two different pitches, only the fine spectral structure is affected, while the overall spectral envelope does not change its shape.

2.5.3 Linear prediction

The problem of extracting a spectral envelope from a signal spectrum is generally an ill-posed problem. If the signal contains harmonic partials only, one could state that the spectral envelope is the curve that



(a) (b) (c)

Figure 2.22: *Formant synthesis of voice: (a) spectra of two pulse trains with fundamental frequencies at* 150 *Hz and* 250 *Hz; (b) first three formants of the vowel /a/; (c) spectra of the two output signals obtained by filtering the pulse trains through a parallel combination of the three formants.*

passes through the partial peaks. This implies that *1)* the peak values have to be retrieved, and *2)* an interpolation scheme should be (arbitrarily) chosen for the completion of the curve in between the peaks. If the sound contains inharmonic partials or a noisy part, then the notion of a spectral envelope becomes completely dependent on the definition of what belongs to the "source" and what belongs to the "filter".

Three main techniques, with many variants, can be used for the estimation of the spectral envelope. The *channel vocoder* uses frequency bands and performs estimations of the amplitude of the signal inside these bands and thus the spectral envelope. *Linear prediction* estimates an all-pole filter that matches the spectral content of a sound. When the order of this filter is low, only the formants are taken, hence the spectral envelope. *Cepstrum* techniques perform smoothing of the logarithm of the FFT spectrum (in decibels) in order to separate this curve into its slow varying part (the spectral envelope) and its quickly varying part (the source signal). In this section we present the basics of Linear Prediction (LP) techniques. We will return on cepstral analysis in Chapter *Auditory based processing*.

2.5.3.1 Linear prediction equations

Consider a general linear system that describes a source-filter model:

$$S(z) = gH(z)X(z), \qquad \text{with } H(z) = \frac{1 + \sum_{k=1}^{q} b_k z^{-k}}{1 - \sum_{k=1}^{p} a_k z^{-k}}, \tag{2.41}$$

where g is a gain scaling factor and $X(z)$ and $S(z)$ are the Z-transforms of the source signal $x[n]$ and the output signal $s[n]$, respectively. This is often termed an *ARMA*(p, q) (Auto-Regressive Moving Average) model, in which the output is expressed as a linear combination of p past samples and $q+1$ input values. LP analysis works on an approximation of this system, namely on an all-pole model:

$$S(z) = gH_{LP}(z)X(z), \qquad \text{with } H_{LP}(z) = \frac{1}{1 - \sum_{k=1}^{p} a_k z^{-k}}. \tag{2.42}$$

The time-domain version of this equation reads $s[n] = gx[n] + \sum_{k=1}^{p} a_k s(n-k)$. Therefore the output $s[n]$ can be predicted using a linear combination of its p past values, plus a weighted input term. In statistical terminology, the output *regresses* on itself, therefore system (2.42) is often termed an *AR*(p) (Auto-Regressive) model

One justification of this approximation is that the input signal $x[n]$ is generally unknown together with the filter $H(z)$. A second more substantial reason is that any filter $H(z)$ of the form (2.41) can be

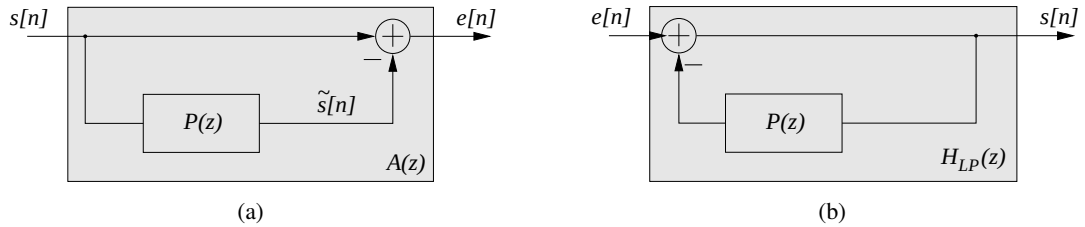


Figure 2.23: *LP analysis: (a) the inverse filter $A(z)$, and (b) the prediction error $e[n]$ interpreted as the unknown input $gx[n]$.*

written as $H(z) = h_0 H_{min}(z) H_{ap}(z)$, where h_0 is a constant gain, $H_{min}(z)$ is a minimum-phase filter, and H_{ap} is an all-pass filter (i.e. $\left| H_{ap}(e^{j\omega_d}) \right| = 1, \forall \omega_d$). Moreover, the minimum-phase filter $H_{min}(z)$ can be expressed as an all-pole system of the form (2.42). Therefore we can say that LP analysis ideally represents the all-pole minumum-phase portion of the general system (2.41), and therefore yelds at least a correct estimate of the magnitude spectrum.

Given an output signal $s[n]$, Linear Prediction analysis provides a method for deteriming the "best" estimate $\{\tilde{a}_i\}$ $(k = 1, \ldots, p)$ for the coefficients $\{a_i\}$ of the filter (2.42). The method can be interpreted and derived in many ways, here we propose the most straightforward one. Given an estimate $\{\tilde{a}_i\}$ of the filter coefficients, we define the *linear prediction* $\tilde{s}[n]$ of the output $s[n]$ as $\tilde{s}[n] = \sum_{k=1}^{p} \tilde{a}_k s(n-k)$. In the Z-domain we can write

$$\tilde{S}(z) = P(z)S(z), \qquad \text{with } P(z) = \sum_{k=1}^{p} a_k z^{-k}, \tag{2.43}$$

and we call the FIR filter $P(z)$ a *prediction filter*. We then define the *prediction error* or *residual* $e[n]$ as the difference between the output $s[n]$ and the linear prediction $\tilde{s}[n]$. In the z-domain, the prediction error $e[n]$ is expressed as

$$E(z) = A(z)S(z), \qquad \text{with } A(z) = 1 - \sum_{k=1}^{p} a_k z^{-k}. \tag{2.44}$$

Comparison of Eqs. (2.42) and (2.44) shows that, if the speech signal obeys the model (2.42) exactly, and if $\tilde{a}_k = a_k$, then the residual $e[n]$ coincides with the unknown input $x[n]$ times the gain factor g, and $A(z)$ is the inverse filter of $H_{LP}(z)$. Therefore LP analysis provides an estimate of the inverse system of (2.42):

$$e[n] = gx[n], \qquad A(z) = [H_{LP}(z)]^{-1}. \tag{2.45}$$

This interpretation is illustrated in Fig. 2.23. If we assume that the prediction error has white (flat) spectrum, then the all-pole filter $H_{LP}(z)$ completely characterizes the spectrum of $s[n]$. For this reason $A(z)$ is also called a *whitening* filter, because it produces a residual with a flat power spectrum. Two kinds of residuals, both having a flat spectrum, can be identified: the pulse train and the white noise. If LP is applied to speech signals, the pulse train represent the idealized vocal-fold excitation for voiced speech, while white noise represents the idealized excitation for unvoiced speech.

The roots of $A(z)$ (i.e., the poles of $H_{LP}(z)$) are representative of the formant frequencies. In other words, the phases of these poles, expressed in terms of analog frequencies, can be used as an estimate of the formant frequencies, while the magnitude of the poles relate to formant bandwidths according to the equations written in Sec. 2.5.1 when discussing resonant filters.

We now describe the heart of LP analysis and derive the equations that determine the "best" estimate $\{\tilde{a}_i\}$ $(k = 1, \ldots, p)$. In this context "best" means best in a least-square sense: we seek the $\{\tilde{a}_i\}$s that minimize the energy $E\{e\} = \sum_{m=-\infty}^{+\infty} e^2[m]$ of the residual, i.e. we set to zero the partial derivatives of $E\{e\}$ with respect to the a_is:

$$0 = \frac{\partial E\{e\}}{\partial a_i} = 2 \sum_{m=-\infty}^{+\infty} e[m] \frac{\partial e[m]}{\partial a_i} = -2 \sum_{m=-\infty}^{+\infty} \left\{ \left[s[m] - \sum_{k=1}^{p} a_k s(m-k) \right] s[m-i] \right\}, \quad (2.46)$$

for $i = 1 \ldots p$. If one defines the temporal autocorrelation of the signal $s[n]$ as the function $r_s[i] = \sum_{m=-\infty}^{+\infty} s[m]s[m-i]$, then the above equation can be written as

$$\sum_{k=1}^{p} a_k r_s[i-k] = r_s[i], \qquad \text{for } i = 1 \ldots p. \quad (2.47)$$

The system (2.47) is often referred to as the *normal equations*. Solving this system in the p unknowns a_i yelds the desired estimates $\tilde{a}_i$.

2.5.3.2 Short-time LP analysis

In many applications of interest, and in particular analysis and resynthesis of speech signals, the coefficients a_k are not constant but slowly time-varying. Therefore the description (2.42) is only valid in a short-time sense, i.e. the a_ks can be assumed constant during an analysis frame. Therefore short-time analysis has to be used, in which the coefficients and the residual are determined from windowed sections, or frames, of the signal.

There are various efficient methods to compute the filter coefficients, the most common ones being the autocorrelation method, the covariance method, and the Burg algorithm. In this section we briefly describe the autocorrelation method, that simply amounts to substitute the autocorrelation function r_s of Eq. (2.47) with its short-time approximation:

$$r_s[i] \sim \sum_{m=1}^{N} u[m]u[m-i], \qquad \text{where } u[m] = s[m]w[m] \quad (2.48)$$

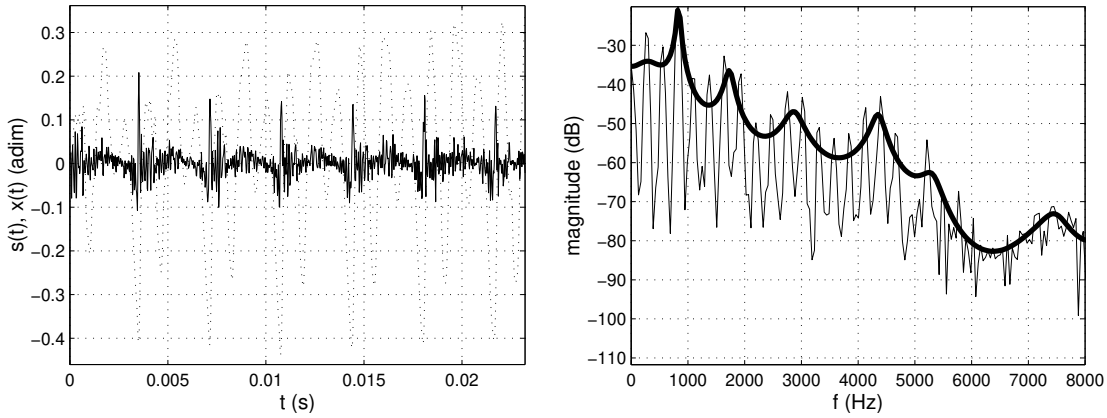
is a windowed version of $s[m]$ in the considered frame ($w[m]$ is typically a Hamming window), and N is the length of the frame. Then the system (2.47) is solved within each frame. An efficient solution is provided by the so-called *Levinson-Durbin recursion*, an algorithm for solving the problem $\boldsymbol{Ax} = \boldsymbol{b}$, with $\boldsymbol{A}$ Toepliz, symmetric, and positive definite, and $\boldsymbol{b}$ arbitrary. System (2.47) is an instance of this general problem.

M-2.13

Write a function `lp_coeffs` that computes the LP coefficients of a finite-length signal $s[n]$, given the desired prediction order p.

M-2.13 Solution

```
% Compute LP coeffs using the autocorrelation method
% s is the (finite-length) signal, p is the prediction order
% a are the computed LP coefficients, g is the gain (sqrt of residual variance)

function [a,g] = lp_coeffs(s,p)
```



(a) (b)

Figure 2.24: *Example of LP analysis/synthesis, with prediction order $p = 50$; (a) target signal $s[n]$ (dotted line) and unit variance residual $x[n]$ (solid line); (b) magnitude spectra $|S(f)|$ (thin line) and $|gH_{LP}(f)|$ (thick line).*

```
R=xcorr (s,p) ; % autocorrelation sequence R(k) with k=-p,..,p
R(1:p)=[];      % delete the first p samples
if norm(R) ~= 0
  [a,v] = levinson(R,p);  % Levinson-Durbin recursion
                          % a=[1,-a1,-a2,...,-ap], v = variance of the residual
else
  a=[1, zeros(1,p)];
end
g=sqrt(sum(a'.*R)); % gain factor (= sqrt(v))
```

Note that we are using the native function `levinson`, that computes the filter coefficients (as well as the variance of the residual) given the autocorrelation sequence and the prediction order.

Figure 2.24 shows an example of LP analysis and resynthesis of a single frame of a speech signal. As shown in Fig. 2.24(a), the analyzed frame is a portion of voiced speech and $s[n]$ is pseudo-periodic. Correspondingly, the estimated source signal $x[n]$ is a pulse train. Figure 2.24(b) shows the magnitude responses of the target signal and the estimated transfer function $gH_{LP}(z)$. A typical feature of LP spectral modeling can also be observed from this figure: the LP spectrum matches the signal spectrum much more closely in the region of large signal energy (i.e. near the spectrum peaks) than near the regions of low energy (spectrum valley).

M-2.14

Write an example script that analyzes frame-by-frame a voice signal using the LP model (2.42).

M-2.14 Solution

```
[s, Fs] = wavread('la.wav');   %%%% input file

%% analysis parameters
N=2048;                  %block length
Sa=256;                  %analysis hop-size
p=round(Fs/1000)+4 ;     %prediction order
```

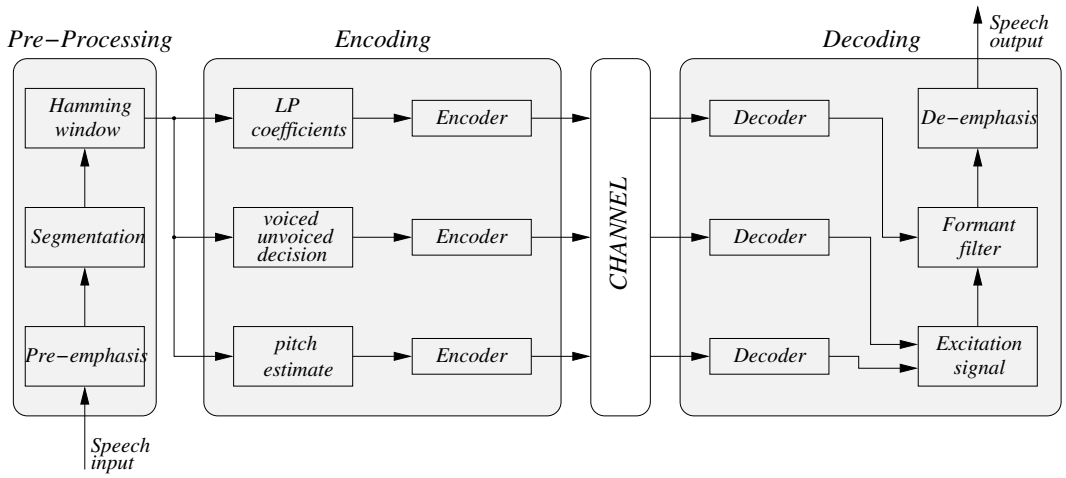


Figure 2.25: *General scheme of a simple LP based codec.*

```
win=hamming(N)';          %' window for input block

M = ceil(length(s)/Sa);     %number of frames in the signal
s(M*Sa+N)=0;                %now s is exactly M*Sa samples

for i=0:M-1   %%% frame-by-frame analysis
  frame=s( (i*Sa+1):(i*Sa+N) );
  [a,g]=lp_coeffs(frame, p); % compute LP coeffs
  x=filter(a,g,frame);       % X(z)=A(z)/g * S(z), i.e. x[n] = e[n]/g
  frameSpec= abs(fft(frame)); [H,F]=freqz(g,a,N,Fs);
  figure(1); clf; plot(frame); hold on; plot(x,'r'); grid on;
  figure(2); clf; plot(F,20*log10(frameSpec)); hold on; grid on;
   plot(F,20*log10(abs(H)),'r'); axis([0 5000 0 max(20*log10(frameSpec))]);
   pause;
end
```

Note that we have used the function `lp_coeffs` written in example M-2.13. The signals plotted in Fig. 2.24 have been computed from this script.

When formant parameters are extrected on a frame-by-frame basis, a lot of discontinuities and local estimation observation errors are found. Therefore, proper techniques have to be used in order to determine smooth formant trajectories over analysis frames. We have already encountered a conceptually similar problem in Sec. 2.4.2, when we have discussed a "sinusoid tracking" procedure.

M-2.15

Plot the formant frequencies as a function of the frame number, i.e., of time, in order to observe the time-evolution of the vocal tract filter. To this purpose, segment a speech signal $s[n]$ into M Hamming windowed frames $s_m[n]$, with a block length N and a hop-size $S_a = N/2$. Then, for each frame: a) compute the LP coefficients; b) find the filter poles and the corresponding formant frequencies; c) discard poles whose magnitude is less than 0.8, as these are unlikely to represent formants.

2.5.3.3 Linear Predictive Coding (LPC)

One of the most successful applications of LP analysis is in speech coding and synthesis, in particular for mobile phone communication. Figure 2.25 depicts a synthetic block diagram of a simple encoder-

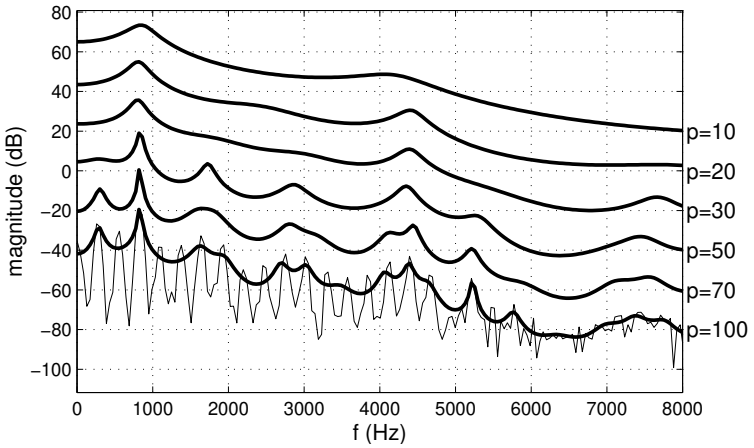


Figure 2.26: *Example of LP spectra for increasing prediction orders p (the target signal is a frame of voiced speech). For the sake of clarity each spectrum is plotted with a different offset.*

decoder system based on LP analysis. Speech is segmented in frames (typical frame lengths can range from 10 to 20 ms). In this phase a pre-emphasis processing can also be applied: since the lower formants contain more energy, they are preferentially modeled with respect to higher formants, and a pre-emphasis filter can compensate for this by boosting the higher frequencies (when reconstructing the signal the inverse filter should be used).

In its simplest formulation the encoder provides, for every frame, the coefficients a_k of the prediction filter, the gain g, a flag that indicates whether the frame corresponds to voiced or unvoiced speech, and the pitch (only in the case of voiced speech). The decoder uses this information to re-synthesize the speech signal. In the case of unvoiced speech, the excitation signal is simply white noise, while in the case of voiced speech the excitation signal is a pulse train whose period is determined by the encoded pitch information.

It is clear that most of the bits of the encoded signal are used for the a_k parameters. Therefore the degree of compression is strongly dependent on the order p of the LP analysis, which in turn has a strong influence on the degree of smoothness of the estimated spectral envelope, and consequently on the quality of the resynthesis (see Fig. 2.26). A commonly accepted operational rule for achieving reasonable intelligibility of the resynthesized speech is

$$p = \begin{cases} F_s + 4, & \text{for voiced speech,} \\ F_s, & \text{for unvoiced speech,} \end{cases} \tag{2.49}$$

where F_s is the sampling frequency in kHz, rounded to the nearest integer.

LPC-10 is an example of a standard that basically implements the scheme of Fig. 2.25. This standard uses an order $p = 10$ (hence the name), a sample rate $F_s = 8$ kHz (which is a common choice in telephone speech applications since most of the energy in a speech signal is the range $[300, 3400]$ Hz), and a frame length of about 22.5 ms. With these values, intelligible speech can be resynthesized.

Howerer LPC-10, and in general similar early codecs, produced speech with very poor quality due to many artifacts: "buzzy" noise through parameter updates, jitter in the excitation signal, wide formant bandwiths, and so on. More recent and commonly used codecs are able to provide natural sounding speech at relatively low bit rates, thanks to an improved description of the excitation signal. Instead of applying a simple two-state voiced/unvoiced model, these codecs estimate the excitation signal through an analysis-by-synthesis approach: excitation waveforms are passed through the formant filter, and the

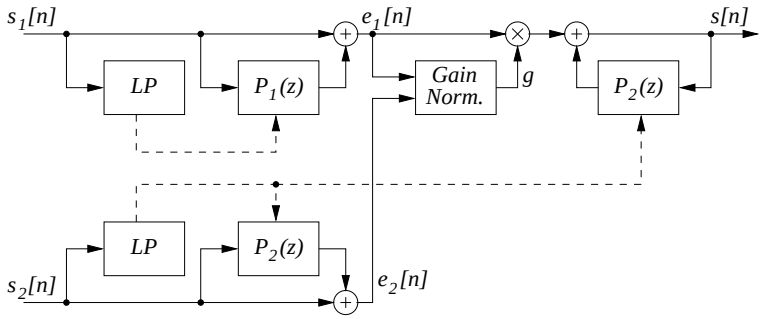


Figure 2.27: *Block scheme of a LP-based implementation of cross-synthesis (also known as vocoder effect) between two input sounds s_1 and s_2.*

excitation which gives the minimum weighted error between the original and the reconstructed speech is then chosen by the encoder and used to drive the synthesis filter at the decoder. It is this 'closed-loop' determination of the excitation which allows these codecs to produce good quality speech at low bit rates, at the expense of a much higher complexity of the coding stage.

Whitin this family of analysis-by-synthesis codecs, many different techniques have been developed for the estimation of the excitation signal. Historically, the first one is the Multi-Pulse Excited (MPE) codec. Later the Regular-Pulse Excited (RPE), and the Code-Excited Linear Predictive (CELP) codecs were introduced. The "Global System for Mobile communications" (GSM), a digital mobile radio system which is extensively used throughout Europe, and also in many other parts of the world, makes use of a RPE codec.

2.5.3.4 LP based audio effects

Several musical effects can be realized based on spectral envelope estimation techniques, particularly linear prediction. These effect exploit the source-filter decomposition obtained through LP analysis of an input sound: modifications are applied to either the excitation, or the synthesis filter, or both.

Thinking of vocal signals, one simple effect can be obtained by modifying the filter block and shifting the vocal tract formants towards higher frequencies, without modifying the source signal. This results in a so called "donald duck" effect, in which the vocal tract has been "shortened" without altering the pitch of the original voice.[8]

Time-stretching effects can be obtained by time-stretching the excitation signal and updating the synthesis filter at a correspondingly time-stretched rate, thus preserving the formant structure. Similarly, pitch-shifting can be obtained by modifying the pitch of the excitation signal: again, this approach allows to transpose pitch without affecting the formant structure of a signal.

A well known effect is *cross-synthesis* between two sound signals $s_1[n]$ and $s_2[n]$: linear prediction (or any other deconvolution technique) is applied to both signals and then the excitation signal of $s_1[n]$ is used in combination with the LP filter of $s_2[n]$. Figure 2.27 shows the block scheme of a possible implementation of a musical vocoder. The cross-synthesis gives particularly pleasant results if $x_2[n]$ is a speech signal and $x_1[n]$ is a musical sound signal: this produced in the popular "vocoder" musical effect, in which an instrumental sound becomes a "talking instrument" when filtered with a time-varying LP filter of a speaking voice. For musically interesting results, the two sounds should be synchronized.

[8] A similar result may be obtained by emitting voice after inhaling a light gas such as helium: the helium in the vocal tract changes its resonances, without affecting the oscillations of the vocal fold. Note that, because of the risk involved, it is safe not to try this experiment at home.

As an example, for the case of cross-synthesis between speech and music the played instrumental notes should fit to the rhythm of the syllables of the speech: this may be achieved if either speech or music is coming from a prerecorded source and the other sound is produced to match to the recording, or if a performer is both playing the instrument and speaking, thus producing both signals at the same time.

M-2.16

Realize the cross-synthesis effect depicted in Fig. 2.27.

2.6 Non-linear models

All the synthesis and processing models examined so far in this chapter are linear and time-invariant. A LTI system, which is completely characterized by its impulse response (or by its transfer function), cannot introduce spectral energy where it is not already present: a sinusoidal signal processed through a linear system remains a sinusoidal signal. When entering the domain of non-linear transformations, this picture changes radically: the spectrum of a signal processed through a non-linear system can be drastically modified, and spectral energy can be created even where it was not originally present. As a consequence, non-linear transformations can modify substantially the nature of the input sounds.

Spectral modifications introduced by non-linear transformations can be grouped into two main effects: spectrum enrichment and spectrum shift. The first effect is due to non-linear distortion of the signal, and reproduces to some extent the non-linearities and saturations found on real systems (e.g., analog amplifiers and electronic valves). The second effect is due to multiplication of the input signal by a sinusoidail carrier signal, which moves the spectrum to the vicinity of the carrier frequency. It derives from abstract mathematical properties of trigonometric functions as used in modulation theory applied to music signal. Therefore, it partially inherits – and simulates digitally – the processing blocks used in analog electronic music. Transformations that produce spectral shifts can produce very intriguing musical effects: complex harmonic and inharmonic spectra can be created starting from simple (sinusoidal) input sounds, and various harmonic relations among the partials can be established.

2.6.1 Memoryless non-linear processing

In general the output value of a non-linear system depends on present and past values of the input and the output, i.e. the system has *memory* (similarly to a generic LTI). However in many interesting audio signal processing applications *memoryless* non-linearities can be used, i.e. non-linear systems whose output depends only on the current input value and not on past values. In this cases the system can be described by a non-linear function $F : \mathbb{R} \to \mathbb{R}$, called *distortion function*, that maps the input into the output:

$$y[n] = F\,(\,x[n]\,)\,. \tag{2.50}$$

By means of such a distortion function, a sound that with a rich spectral content can be obtained by processing a simple sinusoidal input. A block diagram of such a *non-linear distortion synthesis* scheme (or *waveshaping* scheme) is provided in Fig. 2.28, for the particular case of a sinusoidal input signal $x[n]$.

2.6.1.1 Waveshaping and harmonic distortion

In Chapter *Fundamentals of digital audio processing* we have seen that a sinusoidal input $x[n] = a\cos(\omega_0 n)$ which passes through a LTI system (a filter) produces an output signal $y[n]$ which is still a sinusoid with the same frequency ω_0 and amplitude and phase modified according to the transfer function (see

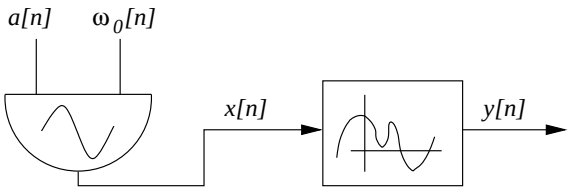


Figure 2.28: *Sound synthesis by non-linear distortion (or waveshaping).*

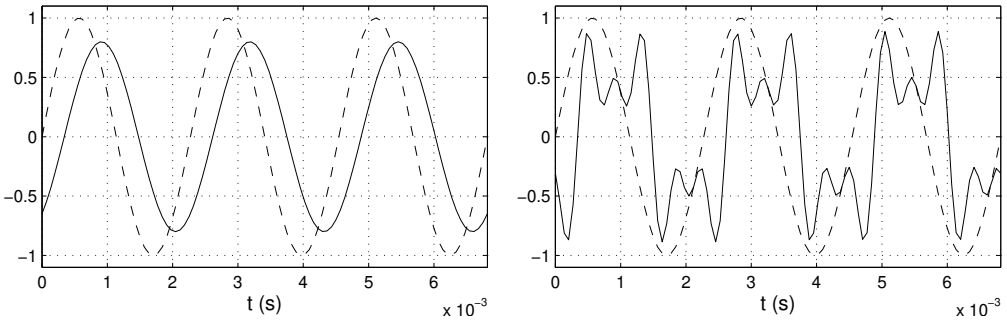


Figure 2.29: *Example of output signals from a linear and from a non-linear system, in response to a sinusoidal input; (a) in a linear system the input and output differ in amplitude and phase only; (b) in a non-linear system they have different spectra.*

Fig. 2.29(a)). On the other hand, if the signal is processed through a non-linear system of the form (2.50), more substantial modifications of the spectrum occur: the output has in general the form

$$y[n] = \sum_{k=0}^{N} a_k \cos(k\omega_0 n), \tag{2.51}$$

and therefore the spectrum of y possesses energy at higher harmonics of ω_0 (see Fig. 2.29(b)). This effect is termed *harmonic distortion*, and can be quantified through the *total harmonic distortion* (THD) parameter:

$$THD = \sqrt{\frac{\sum_{k=2}^{N} a_k^2}{\sum_{k=1}^{N} a_k^2}}. \tag{2.52}$$

In many cases one wants to minimize the THD in non-linear processing, but in other cases distortion is exactly what we want in order to enrich an input sound. An example is the effect of valves, as in amplifiers for electric guitars. There is no way to interpret harmonic distortion in terms of some transfer function, because the concept of transfer function itself cannot be defined for a non-linear system (equivalently, the impulse response of a non-linear system does not tell anything about its response to a generic input).

For a memoryless non-linear system, the THD has a straightforward interpretation if one rewrites the distortion function in terms of its (truncated) Taylor expansion around the origin:

$$y[n] = F\,(\,x[n]\,) = \sum_{i=0}^{N} a_i x^i[n]. \tag{2.53}$$

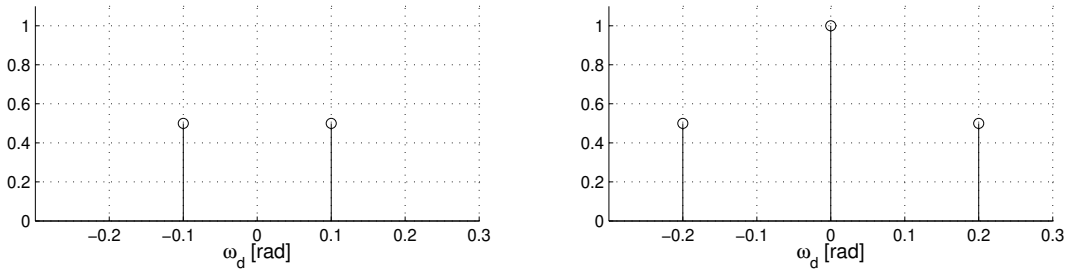


Figure 2.30: *Example of quadratic distortion; (a) spectrum of a sinusoid $x[n]$ and (b) spectrum of the squared sinusoid $x^2[n]$.*

Consider the effect of the quadratic term in this summation. The spectrum of $x^2[n]$ is the convolution of the spectrum of $x[n]$ with itself. Therefore, when $x[n] = a\cos(\omega_0 n)$, i.e. x is a sinusoidal signal with frequency ω_0, the spectrum of $x^2[n]$ is $[X * X](\omega_d)$, and thus contains the frequency $2\omega_0$ as well as the 0 frequency. The same result may be derived looking at the time-domain signal: straightforward trigonometry shows that the squaring operation on a sinusoidal input signal produces the output signal

$$y[n] = x^2[n] = \frac{a^2}{2}\left[1 + \cos(2\omega_0 n)\right]. \tag{2.54}$$

Again, one can see that the output signal contains a DC component and the frequency $2\omega_0$. For a generic input $x[n]$, the squaring operation will in general double the bandwidth of the spectrum. In particular, for an input signal $x[n] = \sum_k a_k \cos(\omega_k n)$ the squaring operation produces an output signal that contains all the frequencies $2\omega_k$ and moreover all the frequencies $\omega_{k_1} \pm \omega_{k_2}$ (the so-called intermodulation frequencies, which arise from the cross terms in the square of the sum).

Similar considerations apply to higher-order terms of the Taylor expansion: raising a sinusoidal signal with frequency ω_0 to the i-th power will produce a spectrum that contains every other frequency up to $i\omega_0$. In particular, if the Taylor expansion of F contains only odd (or only even) powers, then the resulting spectrum will contains only odd (or only even) partials.

One specific application of waveshaping is in the generation of pure harmonics of a sinusoid. As an example suppose that, given the input $x[n] = \cos(\omega_0 n)$ one wants to generate the output $y[n] = \cos(5\omega_0 n)$, i.e. the fifth harmonic of the input. It is easily verified that waveshaping the input through the polynomial distortion function $F(x) = 16x^5 - 20x^3 + 5x$ provides the desired result. More in general, the polynomial that transforms the sinusoid $\cos(\omega_0 n)$ into the sinusoid $\cos(i\omega_0 n)$ is the i-th order *Chebyshev polynomial*.[9] By combining Chebyshev polynomials, one can then produce any desired superposition of harmonic components in the output signal.

2.6.1.2 Aliasing and oversampling

Implementation of a memoryless non-linear system is apparently straightforward in the discrete-time domain: given a distortion function $F(x)$, either in analytical form or estimated from measurements, this can be pre-computed and stored in a look-up table. In order to process an incoming input sample $x[n]$, all that is needed is looking up the corresponding value $F(x[n])$ in the table, and possibly interpolating between adjacent points of the table in order to increase accuracy.

[9] Chebyshev polynomials are in general a sequence of orthogonal polynomials which can be defined recursively, and are widely used especially in approximation theory and polynomial interpolation.

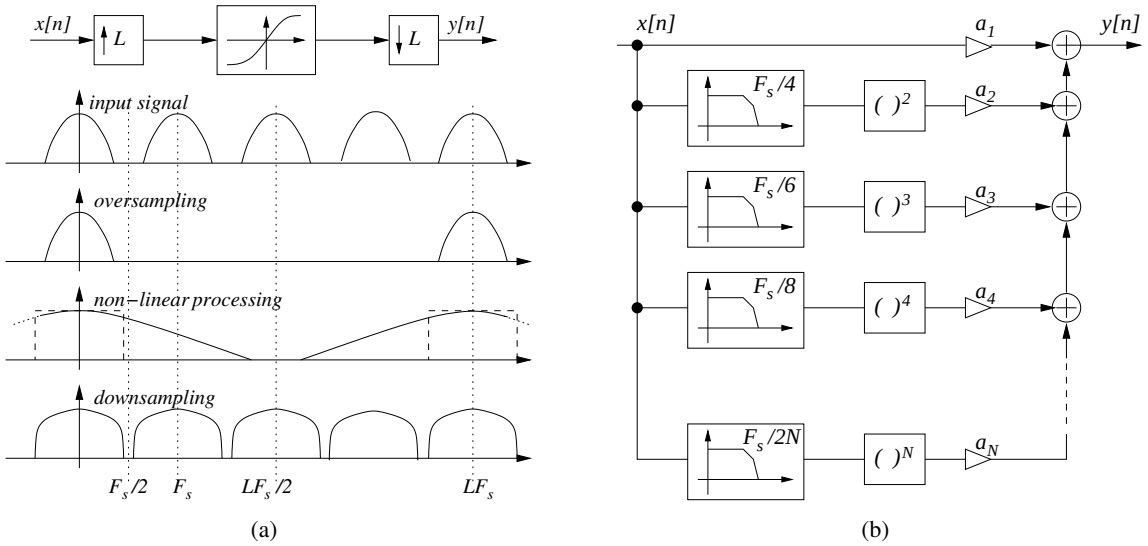


Figure 2.31: *Two implementations of a memoryless non-linear system; (a) non-linear processing inserted between oversampling and downsampling; (b) non-linear processing on band-limited versions of the input.*

From the discussion above we know that if $F(x)$ is a polynomial, or if its Taylor series expansion can be truncated, the bandwidth of the output spectrum induced by harmonic distortion remains limited. Nontheless it can easily extend beyond f_{Ny}, and consequently cause aliasing in the output signal. Even though in some musical effects such an additional aliasing distortion can be tolerated,[10] in general it has to be avoided as much as possible.

Solution: oversampling (see Fig. 2.31(a)). The procedure is illustrated in Fig. 2.31(a). The input is oversampled and interpolated to a higher sampling freqency, say LF_s with some $L > 1$. The distortion function is applied to this oversampled signal. The resulting output can have spectral energy up to the new Nyquist frequency $LF_s/2$. Finally the signal has to be converted back to the original sampling frequency: in order to avoid aliasing at this stage the signal is low-pass filtered back to the original Nyquist frequency.

If F is a polynomial or can be reasonably approximated by a polynomial through its truncated Taylor expansion, an alternative procedure can be designed, that avoids oversampling. This is illustrated in Fig. 2.31(b). The input signal is split into several low-pass versions, and each of them is processed through one term of the polynomial. In this stage no aliasing is generated by construction. Finally the output signal is constructed as the sum of the processed low-pass versions. This procedure is equivalent to the preceding one.

2.6.1.3 Clipping, overdrive and distortion effects

Clipping is a very common type of non-linear distortion. An ideal symmetric clipping distortion function is constructed as follows: it is the identity function, i.e. $F(x) = x$, as long as $|\,x\,| \leq x_{\max}$, and it is a constant function $F(x) = x_{\max}$ when $|\,x\,| > x_{\max}$. Therefore the input amplitude affects the output waveform in that the clipping function passes the input unchanged as long as its amplitude is small enough, while the output remains constant when the input amplitude grows beyond the limit. In

[10] It may be even considered to be helpul, e.g. for extreme metal guitar distortions.

particular, when the input has a decaying amplitude envelope (as in note played on a guitar) the output evolves from a nearly square waveform at the beginning to an almost pure sinusoid at the end.

This kind of effect will be well known to almost any guitarist or anyone who has played an instrument through an overdriven amplifier. In musical terms, *overdrive* refers to a nearly linear audio effect device which can be driven into the non-linear region of its distortion curve only by high input levels. The transition from the operating linear region to the non-linear region is smooth. *Distortion* instead refers to a similar effect, with the difference that the device operates mainly in the non-linear region of the distortion curve.

The sound of a valve amplifier is based on a combination of various factors: the main processing features of valves themselves are important, but the amplifier circuit as well as the chassis and loudspeaker combination have their influence on the final sound. Foot-operated pedal effects have simpler circuitry but always include a non-linear stage that introduces harmonic distortion on the input signal, in a faster way and at lower sound levels than valve amplifiers. The simplest digital emulations of overdrive and distortion effects can be obtained by using a static non-linearity that simulates some form of saturation and clipping.

M-2.17

Write a function that operates a distortion on a guitar input sound using a static non-linearity.

M-2.17 Solution

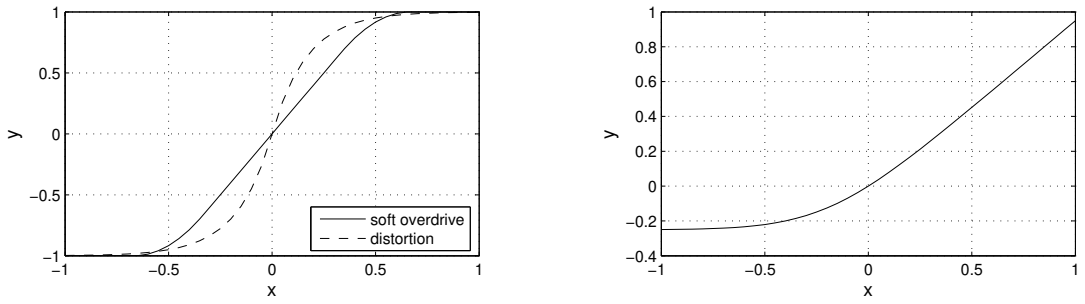
```
function y=distortion(x,dtype,params);

y=zeros(1,length(x));
for i=1:length(x)
    if dtype=='symm' y(i)=symm_overdrive(x(i));
    elseif dtype=='asymm' y(i)=asymm_overdrive(x(i),params(1),params(2));
    elseif dtype=='exp' y(i)=exp_distortion(x(i));
    end
end
```

Note that this is a naive implementation, that can potentially introduce aliasing in the output signal.

Let us now examine some specific non-linear functions that can be used to realize these effects. *Symmetric* distortion is based on static non-linearities that are odd with respect to the origin, are approximately linear for low input values, and saturate (i.e. progressively decrease their slope) with increasing input signals. As a consequence, these non-linearities produce a symmetric (with respect to positive and negative input values) clipping of the signal. A couple of possible parametrizations which have been proposed in the literature are the following:

$$F(x) = \begin{cases} 2x, & 0 \leq |\,x\,| \leq 1/3, \\ \mathrm{sgn}(x)\frac{3-(2-3|\,x\,|)^2}{3}, & \frac{1}{3} < |\,x\,| \leq \frac{2}{3}, \\ \mathrm{sgn}(x), & \frac{2}{3} < |\,x\,| \leq 1, \end{cases} \qquad F(x) = \mathrm{sgn}(x)\left(1 - e^{-q|\,x\,|}\right), \tag{2.55}$$

where the parameter q in the second equation controls the amount of clipping (higher values provide faster saturation). Both functions are shown in Fig. 2.32(a). The first one is claimed to be well suited for implementing an soft overdrive effect, since it realizes a smooth transition of the linear behaviour for low level signal to saturation for high level sounds, resulting in a warm and smooth sound. The second one realizes a stronger clipping and is claimed to be more effective for implementing a distortion effect. Note that, since these functions are odd, their Taylor expansions only contain odd terms and consequently only odd harmonics are generated.



(a) (b)

Figure 2.32: *Simulation of overdrive and distortion; (a) soft overdrive and exponential distortion (with $q = 6$); (b) asymmetric clipping (with $q = -0.2$ and $d = 8$).*

Asymmetric overdrive effects (that are resemblant of the effect of triode valves in the analog domain) are based on distortion curves that clip positive and negative input values in different ways. Since the distortion curve is no longer odd, also even harmonics are generated in this case. A proposal for a function that simulates asymmetric clipping is

$$F(x) = \frac{x - q}{1 - e^{-d(x-q)}} + \frac{q}{1 - e^{dq}}. \tag{2.56}$$

Note that this function is still linear for small input values ($f'(x) \to 1$ and $f(x) \to 0$ for $x \to 0$). The parameter q scales the range of linear behavior (more negative values increase the linear region of operation) and d controls the smoothness of the transition to clipping (higher values provide stronger distortions). A plot of this function is shown in Fig. 2.32(b).

M-2.18

Implement the three functions used in the previous example. For each of them, study the output spectrum when sinusoidal inputs with various amplitudes are provided.

M-2.18 Solution

```
function y=symm_overdrive(x);

if abs(x)<1/3 y=2*x;
elseif abs(x)<2/3 y=sign(x)*(3-(2-3*abs(x))^2)/3;
else y=sign(x);
end
```

2.6.1.4 Non-linear systems with memory

So far in this section we have only examined memoryless non-linear systems. However real non-linear systems are usually systems with memory. As an example, vacuum tubes and solid-state devices that realize analog guitar effects have their internal dynamics, although we have approximated them with simple non-linear instantaneous input-output relations.

In Chapter *Sound modeling: source based approaches* we will see that faithful simulations of non-linear systems can be obtained by writing a set of non-linear differential equations that describe the system dynamics

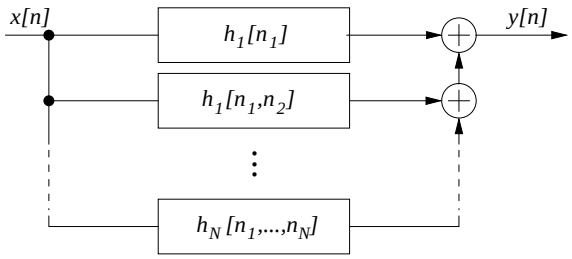


Figure 2.33: *Realization of a non-linear system with memory using the Volterra series (truncated at the order N).*

(e.g. current-voltage relations at each stage of the circuit) and then solving the system numerically. Here insted we stick to a signal based approach and discuss a generalization of the concepts examined so far.

The Volterra series is a model for non-linear systems, that can be seen on one hand as a generalization of the Taylor series for a non-linear function, and on the other hand as a generalization of the impulse response concept for a LTI system. Given an input signal $x[n]$, the output $y[n]$ of a discrete-time non-linear time-invariant system can be expanded in Volterra series as

$$\begin{aligned} y[n] = &\sum_{n_1=0}^{+\infty} h_1[n_1]x[n-n_1] + \sum_{n_1=0}^{+\infty}\sum_{n_2=0}^{+\infty} h_2[n_1,n_2]x[n-n_1]x[n-n_2] + \ldots + \\ &\ldots + \sum_{n_1=0}^{+\infty} \ldots \sum_{n_k=0}^{+\infty} h_k[n_1,\ldots,n_k]x[n-n_1]\ldots x[n-n_k] + \ldots \end{aligned} \tag{2.57}$$

The first term of the series corresponds to usual convolution of an impulse response with the input. However now higher terms are also present, all of which perform multiple convolutions and therefore depend in principle on the input at all past instants. Note also that if the multidimensional impulse responses h_k reduce to unit impulses, then the Volterra expansion reduces to a Taylor expansion.

The main advantage in representing a non-linear system through Eq. (2.57) is that various methods exist for estimating the responses h_k from measurements on real systems. If estimates for these responses are available, then Eq. (2.57) also suggests an implementation scheme, depicted in Fig. 2.33.

On the other hand, these kind of representations are useful only for representing systems with mild non-linearities, while for highly non-linear systems the Volterra series does not converge quickly enough and even with many terms it does not provide a sufficiently accurate representation of system behavior.

2.6.2 Multiplicative synthesis

In this section and in the next one we discuss sound synthesis techniques based on amplitude and frequency modulation, that are not derived from models of sound signals or sound production, and are instead based on abstract mathematical descriptions. These techniques provide versatile methods for producing many types of sounds, with great timbral variability, by using a very limited number of control parameters and with low computational costs.

The main drawback of these techniques is that they cannot be embedded in an analysis-synthesis scheme in which parameters of the synthesis model are derived from analysis of real sounds. No intuitive interpretation can be given to the parameter choice as this synthesis technique does not evoke any previous musical experience of the performer. For these reasons these techniques have progressively lost popularity over the years. However they still retain the attractiveness of their own peculiar timbral spaces

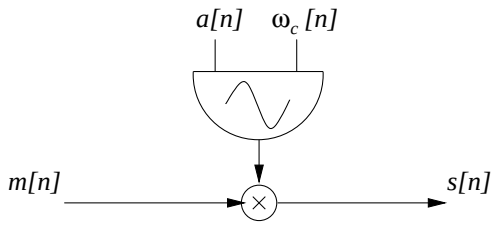


Figure 2.34: *Ring modulation with a sinusoidal carrier.*

and, besides their historical relevance, they still offer a wide range of original synthesis and processing schemes.

2.6.2.1 Ring modulation

The simplest modulation consists of the multiplication of two signals. In the analog domain, synthesis techniques based on this scheme have been called *ring modulation*. Although precise multiplication of two audio signals is not simple to obtain in the analog domain, it becomes a straightforward task for digital signals. Let $x_1[n]$ and $x_2[n]$ be two input signals, the resulting signal is

$$s[n] = x_1[n] \cdot x_2[n] \tag{2.58}$$

and its spectrum is the convolution of the two input signal spectra, i.e. $S(\omega_d) = [X_1 * X_2](\omega_d)$.

Most typically one of the two signals is a sinusoid with frequency ω_c, and is called the *carrier* signal $c[n]$, while the second signal is the input that will be transformed by the ring modulation and is called the *modulating* signal $m[n]$:

$$x_1[n] = c[n] = \cos(\omega_c n + \phi_c), \qquad x_2[n] = m[n]. \tag{2.59}$$

This modulation scheme is shown in Fig. 2.34. Note that the resulting modulated signal $s[n]$ is formally identical to signals with time-varying amplitude examined in other occasions (e.g. sinusoidal oscillators controlled in amplitude). The only but fundamental difference is that in this case the amplitude signal is not a "slow" control signal, but varies at audio rate, and consequently it is perceived in a different way: due to the limited resolution of the human ear, a modulation slower than ~ 20 Hz will be perceived in the time-domain as a time-varying amplitude envelope, whereas a modulation faster than that will be perceived as distinct spectral components. More precisely, the spectrum of $s[n] = c[n]m[n]$ is

$$S(\omega_d) = \frac{1}{2}\left[M(\omega_d - \omega_c)e^{j\phi_c} + M(\omega_d + \omega_c)e^{-j\phi_c}\right], \tag{2.60}$$

i.e. $S(\omega_d)$ is composed of two copies of the spectrum of $M(\omega_d)$, symmetric around ω_c: a lower sideband (LSB), reversed in frequency, and an upper sideband (USB). When the bandwidth of $M(\omega_d)$ extends beyond ω_c, part of the LSB extends to the negative region of the frequency axis, and this part is aliased.

A variant of ring modulation is *amplitude modulation*:

$$s[n] = \{1 + \alpha m[n]\}c[n], \qquad S(\omega_d) = C(\omega_d) + \frac{\alpha}{2}\left[M(\omega_d - \omega_c)e^{j\phi_c} + M(\omega_d + \omega_c)e^{-j\phi_c}\right], \tag{2.61}$$

where α is the amplitude modulation index. In this case the spectrum $S(\omega_d)$ contains also the carrier spectral line, plus side-bands of the form (2.60). From the expression for $S(\omega_d)$ one can see that α controls the amplitude of the sidebands.

2.6.2.2 $|\,\omega_c \pm k\omega_m\,|$ spectra

Let us consider an example of ring modulation composed by a sinusoidal carrier of the form (2.59) and a periodic modulating signal, $m[n] = \sum_{k=1}^{N} b_k \cos(k\omega_m n + \phi_k)$, with fundamental frequency ω_m and N harmonic partials with frequencies $k\omega_m$ ($k = 1, \ldots, N$). In this case multiplicative synthesis causes every spectral line $k\omega_m$ to be replaced by two spectral lines, one in the LSB and the other one in the USB, with frequencies $\omega_c - k\omega_m$ and $\omega_c + k\omega_m$:

$$s[n] = \sum_{k=1}^{N} \frac{b_k}{2} \left\{ \cos\left[(\omega_c + k\omega_m)n + \phi_k\right] - \cos\left[(\omega_c - k\omega_m)n + \phi_k\right] \right\}. \tag{2.62}$$

If $\omega_c - k\omega_m < 0$ for some k, then the corresponding spectral line will be aliased around zero. The resulting spectrum has partials at frequencies $|\,\omega_c \pm k\omega_m\,|$ with $k = 1, \ldots, N$, where the absolute value is used to take into account the possible aliasing around the origin.

Spectra of this kind can be characterized through the ratio ω_c/ω_m, sometimes also called c/m ratio. When this ratio is rational (i.e. $\omega_c/\omega_m = N_1/N_2$ with $N_1, N_2 \in \mathbb{N}$ and mutually prime), the resulting sound is periodic: more precisely all partials are multiples of the fundamental frequency $\omega_0 = \omega_c/N_1 = \omega_m/N_2$ and ω_c, ω_m coincide with the N_1th and N_2th harmonic partial, respectively. As a special case, if $N_2 = 1$ all the harmonics are present and the components with $k < -N_1$, i.e. with negative frequency, overlap some components with positive k. In general the N_1/N_2 ratio can be considered as an index of the harmonicity of the spectrum. The sound spectrum is more resemblant as a complete harmonic spectrum when the N_1/N_2 ratio is simple. The simplest possible c/m ratio is $1/2$: in this case the effect of ring modulation is simply that of producing a sound whose fundamental frequency is half that of the modulating sound, with a limited distortion of the overall spectral envelope. This is a kind of *octave divider* effect.

The c/m ratios can be grouped in families. All ratios of the type $|\,\omega_c \pm k\omega_m\,|\,/\omega_m$ produce the same components. As an example, the ratios $2/3$, $5/3$, $1/3$, $4/3$, $7/3$ and so on produce the same set of partials, in which only those that are multiples of 3 are missing. As a consequence, one family of ratio can be identifed through a *normal form* ratio, i.e. the smallest ratio (the normal form ratio in the previous example is $1/3$).

When the ω_c/ω_m ratio is irrational, the resulting sound is inharmonic. This configuration can be used to create inharmonic sounds, such as bells. As an example if $\omega_c/\omega_m = 1/\sqrt{2}$, the sound contains partials with frequency $\omega_c \pm k\sqrt{2}$ and no implied fundamental pitch is audible. Of particular interest is the case of an ω_c/ω_m ratio approximating a simple rational value, that is,

$$\frac{\omega_c}{\omega_m} = \frac{N_1}{N_2} + \epsilon, \qquad \text{with} \quad \epsilon \ll 1. \tag{2.63}$$

In this case the fundamental frequency is still $\omega_0 = \omega_m/N_2$, but partials are shifted from the harmonic series by $\pm\epsilon\omega_m$, so that the spectrum becomes slightly inharmonic. A small shift of ω_c does not change the pitch, but it slightly spread the partials and makes the sound more lively.

2.6.3 Frequency and phase modulation

The definition of *synthesis by frequency modulation (FM)* encompasses an entire family of techniques in which the instantaneous frequency of a *carrier* signal is itself a *modulating* signal that varies at audio rate. We have already discussed oscillators whose frequency varies slowly in time, and is consequently perceived as a varying pitch. In this case we are considering audio-rate frequency changes, which produce radically different effects.

2.6.3.1 A frequency-modulated sinusoidal oscillator

We have already seen in Chapter *Fundamentals of digital audio processing* how to compute the signal phase $\phi[n]$ when the instantaneous frequency $f_0[n]$ is varying at frame rate. We now face the problem of computing $\phi[n]$ when the instantaneous frequency varies at audio rate. A way of approximating $\phi[n]$ is through a first-order expansion.

Recalling that, in continuous time, phase and instantaneous frequency are related through $2\pi f_0(t) = d\phi/dt(t)$ (see Chapter *Fundamentals of digital audio processing*), we can approximate this relation over two consecutive discrete time instants as

$$\frac{d\phi}{dt}\left((n-1)T_s\right) \sim 2\pi\left[\frac{f_0(nT_s)+f_0((n-1)T_s)}{2}\right], \tag{2.64}$$

i.e. the phase derivative is approximated as the average of the instantaneous frequency at two consecutive instants. Using this approximation, a first-order expansion of the phase can be approximated in discrete time as

$$\phi[n] = \phi[n-1] + \frac{\pi}{F_s}(f_0[n] + f_0[n-1]). \tag{2.65}$$

M-2.19

Write a function that realizes a frequency-modulated sinusoidal oscillator, with input parameters `t0` (initial time), `a` (frame-rate amplitude vector) `f` (audio signal representing the instantaneous frequency vector), and `ph0` (initial phase).

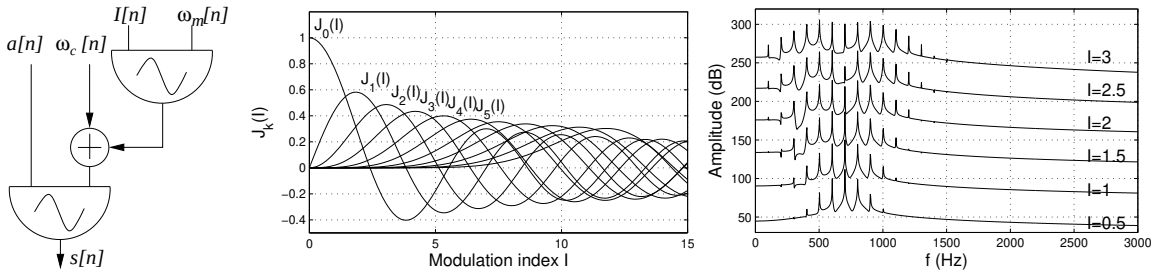
M-2.19 Solution

```
function s=fm_osc(t0,a,f,ph0);

global Fs; global SpF; %global variables: sample rate, samples-per-frame

nframes=length(a);          %total number of frames
s=zeros(1,nframes*SpF);     %initialize signal vector to 0
lastf=f(1); lastph=ph0;     %initialize frequency, phase

for (i=1:nframes)
   phase=zeros(1,SpF);       %phase vector in a frame
   for(k=1:SpF)  % work at sample rate
      phase(k)=lastph + pi/Fs*(f((i-1)*SpF+k)+lastf); %compute phase
      lastph=phase(k); lastf=f((i-1)*SpF+k);          %save last values
   end
   s(((i-1)*SpF+1):i*SpF)=a(i).*sin(phase);
end
s=[zeros(1,round(t0*Fs)) s]; %add initial silence of t0 sec.
```

Compare this function with the `sinosc` function discussed in Chapter *Fundamentals of digital audio processing*. The only difference is that in this case the frequency is given at audio rate. Consequently the phase computation differs.

Although early realizations of FM synthesis were implemented in this fashion, in the remainder of this section we will follow an equivalent "phase-modulation" formulation, according to which the FM oscillator is written as:

$$s[n] = a[n] \cdot \sin\left(\omega_c[n]n + \phi[n]\right), \tag{2.66}$$



(a) (b) (c)

Figure 2.35: *Simple modulation: (a) block scheme; (b) the first 10 Bessel functions; (c) spectra produced by simple modulation with $\omega_c = 2\pi 700$ Hz, $f_m = 2\pi 100$ Hz, and I varying from 0.5 to 3 (for the sake of clarity each spectrum is plotted with a different offset).*

where $a[n]$ is the (frame rate) amplitude signal, $\omega_c[n]$ is the (frame rate) *carrier* frequency, and $\phi[n]$ is the (audio rate) modulating signal. In this case the iterative computation used in the example M-2.19 can be substituted by the following:

$$\begin{aligned} \varphi[n] =& \varphi[n-1] + \omega_c[n] + \phi[n], \\ y[n] =& a[n] \cdot \sin(\varphi[n]), \end{aligned} \tag{2.67}$$

where $\varphi[n]$ is a state variable representing the instantaneous phase of the oscillator.

2.6.3.2 Simple modulation

The simplest FM scheme (appropriately termed *simple modulation*) employs in Eq. (2.66) a sinusoidal modulating signal $\phi[n]$ with amplitude $I[n]$ (called *modulation index*) and frequency $\omega_m[n]$:

$$\phi[n] = I[n] \sin(\omega_m[n] n), \tag{2.68}$$

where $I[n]$, $\omega_m[n]$ vary at frame rate. This modulation (shown in Fig. 2.35(a)) produce the signal

$$s[n] = a[n] \sin\left[\omega_c[n] n + I[n] \sin(\omega_m[n] n)\right] = a[n] \sum_{k=-\infty}^{+\infty} J_k(I[n]) \sin\left[(\omega_c[n] + k\omega_m[n])n\right], \tag{2.69}$$

where $J_k(I[n])$ is the k-th order Bessel function of the first kind, evaluated in the point $I[n]$. From Eq. (2.69) we can see that the spectrum has partials at frequencies $|\, \omega_c \pm k\omega_m \,|$ (as already discussed for ring modulation, negative frequencies are aliased around the origin). Each partial has amplitude $J_k(I)$: a plot of the first Bessel functions is shown in Fig. 2.35(b), from which one can see that partial amplitudes are modulated in a very complex fashion when the modulation index I is varied.

Note that an infinite number of partials is generated, so that the signal bandwidth is not limited. In practice however only a few low-order Bessel functions take significantly non-null values for small values of I. As I increases, the number of significantly non-null Bessel functions increases too. A way of characterizing the bandwidth of $s[n]$ is by saying that the number M of lateral spectral lines $|\, \omega_c \pm k\omega_m \,|$ that are greater than $1/100$ of the nonmodulated signal is given by $M(I) = I + 2.4 \cdot I^{0.27}$: therefore $M(I) \sim I$ for non small I values, and the bandwidth around ω_c is approximately $2I$. Manipulation of the modulation index produces an effect similar to low-pass filtering with varying cut-off frequency, and with smooth variation of the amplitude of partials. Figure 2.35(c) show the spectra produced by simple modulation, with varying modulation index values: as the index increases the energy of the carrier frequency is progressively transferred to the lateral bands, according to the predicted behaviour.

2.6.3.3 Other basic FM schemes

There are many variation of the simple modulation scheme examined above. If the modulating signal is composed of N sinusoids, $\phi[n] = \sum_{i=1}^{N} I_i[n] \sin(\omega_{m,i}[n]n)$, the corresponding FM scheme is termed *compound* (or *complex*) *modulation* (shown in Fig. 2.36(a)) and the following relation holds:

$$\begin{aligned} s[n] &= a[n] \sin\left[\omega_c[n]n + \sum_{i=1}^{N} I_i[n] \sin(\omega_{m,i}[n]n)\right] \\ &= a[n] \sum_{k_1,\ldots,k_N} \prod_{i=1}^{N} J_{k_i}(I_i[n]) \sin\left[\left(\omega_c[n] + \sum_{i=1}^{N} k_i \omega_{m,i}[n]\right) n\right], \end{aligned} \tag{2.70}$$

where the integers $k_1, \ldots, k_N$ all vary between $-\infty$ and $+\infty$. Therefore $s[n]$ possesses all the partials with frequencies $|\, \omega_c \pm k_1\omega_{m,1} \pm \cdots \pm k_N\omega_{m,N} \,|$ with amplitudes given by the product of N Bessel functions. If the ratios between the $\omega_{m,i}$s are sufficiently simple, then the spectrum is again of the type $|\, \omega_c \pm k\omega_m \,|$. Otherwise the spectrum is highly inharmonic (and takes a noisy character for high index values).

M-2.20

Synthesize a frequency modulated sinusoid in the case of compound modulation, and study the signal spectra when control parameters are varied.

A more complex FM scheme is *nested modulation* (shown in Fig. 2.36(b)) , in which a sinusoidal modulator is itself modulated by a second one, i.e. $\phi[n] = I_1[n] \sin[\omega_{m,1}[n]n + I_2[n] \sin(\omega_{m,2}[n]n)]$. In this case the resulting signal is

$$\begin{aligned} s[n] =& a[n] \sin\{\omega_c[n]n + I_1[n] \sin[\omega_{m,1}[n]n + I_2[n] \sin(\omega_{m,2}[n]n)]\} = \\ =& a[n] \sum_{k=-\infty}^{+\infty} \sum_{n=-\infty}^{+\infty} J_k(I_1[n]) J_n(kI_2[n]) \sin\{(\omega_c[n] + k\omega_{m,1}[n] + n\omega_{m,2}[n])n\}. \end{aligned} \tag{2.71}$$

The result can be interpreted as if each partial produced by the modulating frequency $\omega_{m,1}$ were modulated by $\omega_{m,2}$ with modulation index kI_2. The spectral structure is similar to that produced by two sinusoidal modulators, but with larger bandwidth.

The last FM scheme that we examine is *feedback modulation* (shown in Fig. 2.36(c)), in which past values of the output signal are used as a modulating signal, i.e. $\phi[n] = \beta s[n - n_0]$. If $n_0 = 1$, the modulated signal is

$$s[n] = a[n] \sin(\omega_c[n]n + \beta s[n-1]) = a[n] \sum_{k=-\infty}^{+\infty} \frac{2}{k\beta} J_k(k\beta) \sin(k\omega_c[n]n), \tag{2.72}$$

and β (called the *feedback factor*) acts as a scale factor or feedback modulation index. For increasing values of β the resulting signal is periodic of frequency ω_c and changes smoothly from a sinusoid to a sawtooth waveform. Moreover one may vary the delay n_0 in the feedback, and observe emergence of chaotic behaviors for suitable combinations of the parameters n_0 and β.

2.6.3.4 FM synthesis of instrumental sounds

As already mentioned earlier in this chapter, one of the main drawbacks of non-linear modulation synthesis approaches (ring modulation, frequency modulation) is that they cannot be embedded into a synthesis-by-analysis framework in which parameters of the synthesis models are derived from analysis of real

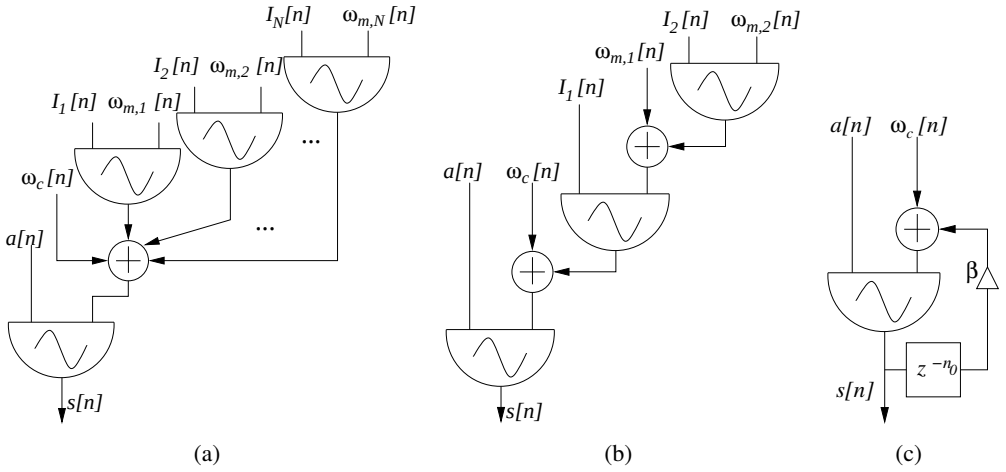


Figure 2.36: *Basic FM schemes; (a) compound modulation, (b) nested modulation, and (c) feedback modulation.*

sounds. In particular, tuning a FM synthesizer to simulate an acoustic instrumental sound is by no means a trivial task, which can only be accomplished through empirical exploration of the parameter space, intuition, and a lengthy trial and error process.

Nonetheless, since the early FM formulations researchers and musician have been experimenting with FM schemes, parameter values, and envelopes, with the goal of imitating familiar instrumental sounds. Several "FM instruments" have been proposed over the years. A simple example can be given using the basic FM schemes analyzed earlier in this section: specifically, the compound modulation scheme of Fig. 2.36(a) can be used to synthesize a FM piano. Taking $N = 2$ modulating signals, with $\omega_{m,1} \simeq \omega_c$ and $\omega_{m,2} \simeq 4\omega_c$, then the greatest common divisor is $\omega_{m,1}$ and the frequency modulated sound has a fundamental frequency around $\omega_{m,1}$ and a slightly inharmonic spectrum. If suitable ADSR envelopes are added to this scheme, piano tones can be simulated, using higher modulation indexes for lower tones (inharmonicity is generated especially in low piano strings).

During the 1980's FM synthesis became a major industry with the introduction of several commercial synthesizers. In particular, it was implemented in the (possibly) most successful synthesizer of all times: the Yamaha DX7. This synthesizer, along with many others based on the same principles, utilized a set of so-called "operators" (6 in the DX7), i.e. amplitude controlled FM oscillators. These operators were connected in a number of different combinations (including simple, compound, nested, and feedback schemes), to produce several so-called "algorithms". The user had control on all the synthesis parameters, particularly in terms of temporal envelopes, and could produce his/her own sounds besides the instrumental presets. Through the DX7 and its successors, FM synthesis became in a way a trademark of the sound of pop music in the 1980's.

2.7 Commented bibliography

Among the plethora of available sound synthesis languages, one of the most widely used (and one of the most important historically) is Csound, developed by Barry Vercoe at the Massachusetts Institute of Technology. Csound descends from the family of Music-N languages created by Max Mathews at Bell Laboratories. See [Vercoe, 1993].

A second influential sound synthesis programming paradigm was developed starting from the early

1980's, mainly by Miller Puckette, and is today represented in three main software implementation: Max/MSP, jmax, and Pd. The "Max paradigm" (so named in honor of Max Mathews) is described by Puckette [Puckette, 2002] as a way of combining pre-designed building blocks into sound-processing "patches", to be used in real-time settings. This includes a scheduling protocol for both control- and audio-rate computations, modularization and component intercommunication, and a graphical environment to represent and edit patches.

About sampling and wavetable synthesis. Contemporary music synthesizer are still based on these techniques, and allow ever increasing quality thanks to the ever increasing availability of storage capacity. A multi-sampled instrument can occupy several Gb. From the point of view of music history these techniques are rooted in several works from the '50s, especially by composer Pierre Schaefer and coworkers, who experimented with the use of recorded environmental sound as sonic material in their compositions. This approach to musical composition has been termed *musique concrete*.

About granular synthesis. The scientific foundations of these approaches can be found in the work of hungarian physicist Dennis Gabor (see [Gabor, 1947]). The composer Iannis Xenakis developed this method in the field of analog electronic music. Starting from Gabor theory, Xenakis suggested a compositional method based on the organization of the grains by means of screen sequences, which specify frequency and amplitude parameters of the grains at discrete points in time. In this way a common conceptual approach is used both for micro and macro musical structure: "All sound, even continuous musical variation, is conceived as an assemblage of a large number of elementary sounds adequately disposed in time. In the attack, body and decline of a complex sound, thousands of pure sounds appear in a more or less short time interval of time Δt" [Xenakis, 1992].

The most widely treated case is (*asynchronous granular synthesis*), where simple grains are distributed irregularly. A classic introduction to the topic is [Roads, 1991]. In particular, figure 2.4 in this chapter is based on an analogous figure in [Roads, 1991]. In another classic work, Truax [Truax, 1988] describe the granulation of recorded waveforms.

About recent *corpus-based* concatenative synthesis techniques. A review is provided in [Schwarz, 2007]

About overlap-add techniques. The pitch-synchronous overlap-add algorithm for time-stretching was introduced by Moulines and Charpentier [1990] in the context of speech processing applications.

Additive synthesis was one of the first sound modeling techniques adopted in computer music and has been extensively used in speech applications as well. The main ideas of the synthesis by analysis techniques that we have reviewed date back to the work by McAulay and Quatieri [McAulay and Quatieri, 1986]. In the same period, Smith and Serra started working on "sines-plus-noise" representations, usually termed *SMS* (Spectral Modeling Synthesis) by Serra. A very complete coverage of the topic is provided in [Serra, 1997]. The extension of the additive approach to a "sines-plus-transients-plus-noise" representation is more recent, and has been proposed by Verma and Meng [Verma and Meng, 2000].

Subtractive synthesis techniques became extremely popular in the 1960's and 1970's, with the advent of analog voltage controlled synthesizers. The Moog synthesizers were especially successful and were based on a range of signal generators, filters, and control modules, which could be easily interconnected to each other. The central component was the voltage-controlled oscillator (VCO), which could produce a variety of waveforms and could be connected to other modules such as voltage-controlled amplifiers (VCA), voltage-controlled filters (VCF), envelope generators, and other devices. Moog's innovations were first presented in [Moog, 1965]. Several techniques for antialiasing digital oscillators, to be used in digital emulation of analog subtractive synthesis, are discussed in [Välimäki and Huovilainen, 2007].

A tutorial about filter design techniques, including normalization approaches that use L^1, L^2, and L^∞ norms of the amplitude response, is [Dutilleux, 1998]. Introductions to formant speech synthesis and linear prediction techniques and their applications in speech technology can be found in many textbook. See e.g. [Rabiner and Schafer, 1978] (our Fig. 2.21 is based on a similar figure in this book). Another

useful reference on the topic is [Deller et al., 1993]. One technique that is alternative to linear prediction and widely used is *digital all-pole modeling (DAP)* [El-Jaroudi and Makhoul, 1991].

The use of frequency modulation as a sound synthesis algorithm was first experimented by Chowning [1973] (later reprinted in [Roads and Strawn, 1985]), although these techniques had already been used for decades in electrical communications. While performing experiments on different extents of vibrato applied to simple oscillators, Chowning realized that when vibrato rates entered the audio range, dramatic timbral changes were produced. Soon after FM became very popular and were applied also to the simulation of real sounds: see [Schottstaedt, 1977] (also reprinted in [Roads and Strawn, 1985]). Our example of a synthetic piano tone at the end of Sec. 2.6.3 is taken from this latter work. The FM algorithms used for the DX7 synth are discussed at length in [Chowning and Bristow, 1986].

References

John Chowning. The synthesis of complex audio spectra by means of Frequency Modulation. *J. Audio Engin. Soc.*, 21(7), 1973.

John Chowning and David Bristow. *FM Theory and applications*. Yamaha Music Foundation, Tokio, 1986.

John R Deller, John G. Proakis, and John. H.L. Hansen. *Discrete-Time Processing of Speech Signals*. Macmillan, New York, 1993.

P. Dutilleux. Filters, Delays, Modulations and Demodulations: A Tutorial. In *Proc. COST-G6 Conf. Digital Audio Effects (DAFx-98)*, pages 4–11, Barcelona, 1998.

Amro El-Jaroudi and John Makhoul. Discrete all-pole modeling. *IEEE Trans. Sig. Process.*, 39:411–423, Feb. 1991.

Dennis Gabor. Acoustical quanta and the theory of hearing. *Nature*, 159(4044):591–594, 1947.

R. McAulay and T. F. Quatieri. Speech Analysis/Synthesis Based on a Sinusoidal Speech Model. *IEEE Trans. Acoust., Speech, and Sig. Process.*, 34:744–754, 1986.

Robert A. Moog. Voltage-controlled electronic music modules. *J. Audio Eng. Soc.*, 13(3):200–206, July 1965.

E..... Moulines and F..... Charpentier. Pitch synchronous waveform processing techniques for text to speech synthesis using diphones. *Speech Communication*, 9(5/6):453–467, 1990.

M. Puckette. Max at seventeen. *Computer Music J.*, 26(4):31–43, 2002.

L. R. Rabiner and R. W. Schafer. *Digital Processing of Speech Signals*. Prentice-Hall, Englewood Cliffs, NJ, 1978.

C. Roads. Asynchronous granular synthesis. In G. De Poli, A. Piccialli, and C. Roads, editors, *Representations of Musical Signals*, pages 143–186. MIT Press, 1991.

C. Roads and J. Strawn, editors. *Foundations of Computer Music*. MIT Press, 1985.

William Schottstaedt. The simulation of natural instrument tones using frequency modulation with a complex modulating wave. *Computer Music J.*, 1(4):46–50, 1977.

Diemo Schwarz. Corpus-based concatenative synthesis. *IEEE Signal Processing Magazine*, pages 92–104–, Mar. 2007.

X. Serra. Musical sound modeling with sinusoids plus noise. In C. Roads, S. Pope, A. Piccialli, and G. De Poli, editors, *Musical Signal Processing*, pages 91–122. Swets & Zeitlinger, 1997. http://www.iua.upf.es/∼xserra/articles/msm/.

B. Truax. Real-time granular synthesis with a digital signal processor. *Computer Music J.*, 12(2):14–26, 1988.

Vesa Välimäki and Antti Huovilainen. Antialiasing oscillators in subtractive synthesis. *IEEE Signal Processing Magazine*, 24 (2):116–125, Mar. 2007.

B. Vercoe. Csound: A manual for the audio processing system and supporting programs with tutorials. Technical report, Media Lab, M.I.T., Cambridge, Massachusetts, 1993. Software and Manuals available from ftp://ftp.maths.bath.ac.uk/pub/dream/.

T. S. Verma and T. H. Y. Meng. Extending Spectral Modeling Synthesis with Transient Modeling Synthesis. *Computer Music J.*, 24(2):47–59, 2000.

Iannis Xenakis. *Formalized music: Thought and Mathematics in Composition*. Pendragon Press, Stuyvesant, NY, 1992.

Contents