Chapter 1

Fundamentals of digital audio processing

Federico Avanzini and Giovanni De Poli

1.1 Introduction

The purpose of this chapter is to provide the reader with fundamental concepts of digital signal processing, which will be used extensively in the reminder of the book. Since the focus is on audio signals, all the examples deal with sound. Those who are already fluent in DSP may skip this chapter.

1.2 Discrete-time signals and systems

1.2.1 Discrete-time signals

Signals play an important role in our daily life. Examples of signals that we encounter frequently are speech, music, picture and video signals. A signal is a function of independent variables such as time, distance, position, temperature and pressure. For examples, speech and music signals represent air pressure as a function of time at a point in space.

Most signals we encounter are generated by natural means. However, a signal can also generated synthetically or by computer simulation. In this chapter we will focus our attention on a particulary class of signals: The so called *discrete-time signals*. This class of signals is the most important way to describe/model the sound signals with the aid of the computer.

1.2.1.1 Main definitions

We define a signal x as a function $x : \mathcal{D} \rightarrow \mathcal{C}$ from a domain $\mathcal{D}$ to a codomain $\mathcal{C}$. For our purposes the domain $\mathcal{D}$ represents a time variable, although it may have different meanings (e.g. it may represent spatial variables). A signal can be classified based on the nature of $\mathcal{D}$ and $\mathcal{C}$. In particular these sets can

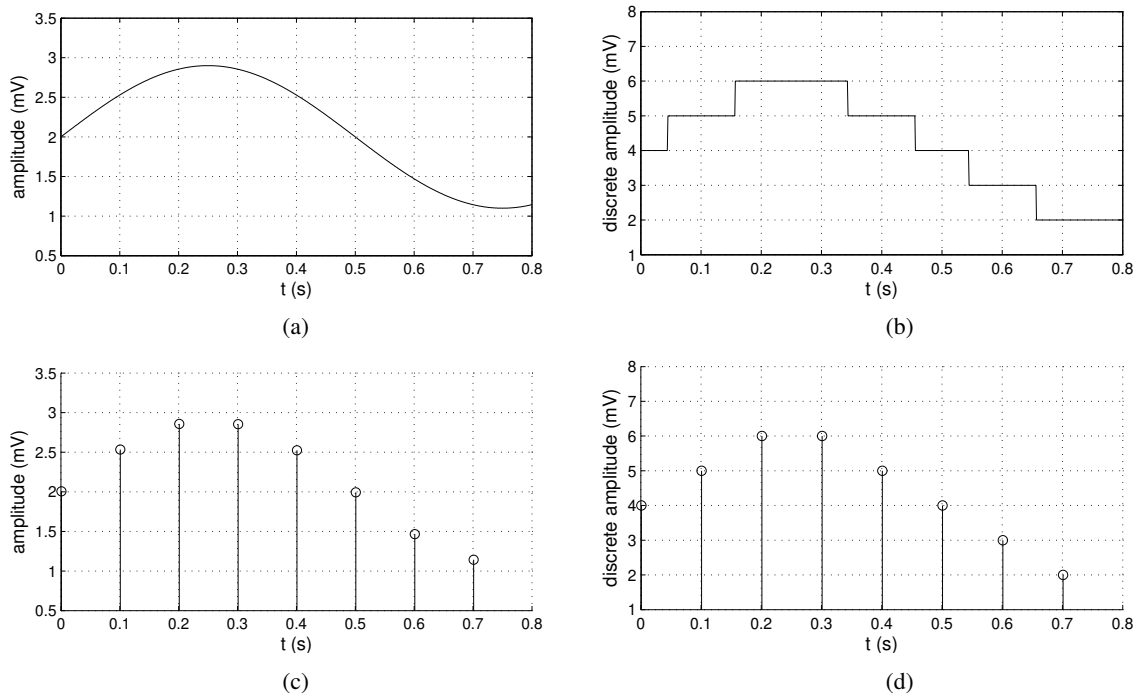


Figure 1.1: *(a) Analog, (b) quantized-analog, (c) discrete-time, and (d) numerical signals.*

be countable or non-countable. Moreover, $\mathcal{C}$ may be a subset of $\mathbb{R}$ or $\mathbb{C}$, i.e. the signal x may be either a real-valued or a complex-valued function.

When $\mathcal{D} = \mathbb{R}$ we talk of *continuous-time* signals $x(t)$, where $t \in \mathbb{R}$, while when $\mathcal{D} = \mathbb{Z}$ we talk of *discrete-time* signals $x[n]$. In this latter case $n \in \mathbb{Z}$ identifies discrete time instants t_n: the most common and important example is when $t_n = nT_s$, with T_s is a fixed quantity. In many practical applications a discrete-time signal x_d is obtained by *periodically sampling* a continuous-time signal x_c, as follows:

$$x_d[n] = x_c(nT_s) \quad -\infty < n < \infty, \tag{1.1}$$

The quantity T_s is called *sampling period*, measured in s. Its its reciprocal is the *sampling frequency*, measured in Hz, and is usually denoted as $F_s = 1/T_s$. Note also the use of square brackets in the notation for a discrete-time signal $x[n]$, which avoids ambiguity with the notation $x(t)$ used for a continuous-time signal.

When $\mathcal{C} = \mathbb{R}$ we talk of *continuous-amplitude* signals, while when $\mathcal{C} = \mathbb{Z}$ we talk of *discrete-amplitude* signals. Typically the range of a discrete-amplitude signal is a finite set of M values $\{x_k\}_{k=1}^{M}$, and the most common example is that of a *uniformly quantized* signal with $x_k = kq$ (where q is called quantization step).

By combining the above options we obtains the following classes of signals, depicted in Fig. 1.1:

1. $\mathcal{D} = \mathbb{R}$, $\mathcal{C} = \mathbb{R}$: *analog* signal.
2. $\mathcal{D} = \mathbb{R}$, $\mathcal{C} = \mathbb{Z}$: *quantized analog* signal.
3. $\mathcal{D} = \mathbb{Z}$, $\mathcal{C} = \mathbb{R}$: *sequence*, or *sampled* signal.
4. $\mathcal{D} = \mathbb{Z}$, $\mathcal{C} = \mathbb{Z}$: *numerical*, or *digital*, signal. This is the type of signal that can be processed with the aid of the computer.

In these sections we will focus on discrete-time signals, regardless of whether they are quantized or not. We will equivalently use the terms discrete-time signal and sequence. We will always refer to a single value $x[n]$ as the n-th *sample* of the sequence x, regardless of whether the sequence has been obtained by sampling a continuous-time signal or not.

1.2.1.2 Basic sequences and operations

Sequences are manipulated through various basic operations. The *product* and *sum* between two sequences are simply defined as the sample-by-sample product sequence and sum sequence, respectively. *Multiplication by a constant* is defined as the sequence obtained by multiplying each sample by that constant. Another important operation is *time shifting* or *translation*: we say that a sequence $y[n]$ is a shifted version of $x[n]$ if

$$y[n] = x[n - n_0], \tag{1.2}$$

with $n_0 \in \mathbb{Z}$. For $n_0 > 0$ this is a *delaying* operation while for $n_0 < 0$ it is an *advancing* operation.

Several basic sequences are relevant in discussing discrete-time signals and systems. The simplest and the most useful sequence is the *unit sample sequence* $\delta[n]$, often referred to as *unit impulse* or simply *impulse*:

$$\delta[n] = \begin{cases} 1, & n = 0, \\ 0, & n \neq 0. \end{cases} \tag{1.3}$$

The unit impulse is also the simplest example of a *finite-legth* sequence, defined as a sequence that is zero except for a finite interval $n_1 \leq n \leq n_2$. One trivial but fundamental property of the $\delta[n]$ sequence is that any sequence can be represented as a linear combination of delayed impulses:

$$x[n] = \sum_{k=-\infty}^{\infty} x[k]\delta[n - k]. \tag{1.4}$$

The *unit step sequence* is denoted by $u[n]$ and is defined as

$$u[n] = \begin{cases} 1, & n \geq 0, \\ 0, & n < 0. \end{cases} \tag{1.5}$$

The unit step is the simplest example of a *right-sided* sequence, defined as a sequence that is zero except for a right-infinite interval $n_1 \leq n < +\infty$. Similarly, *left-sided* sequences are defined as a sequences that are zero except for a left-infinite interval $-\infty < n \leq n_1$.

The unit step is related to the impulse by the following equalities:

$$u[n] = \sum_{k=0}^{\infty} \delta[n - k] = \sum_{k=-\infty}^{n} \delta[k]. \tag{1.6}$$

Conversely, the impulse can be written as the *first backward difference* of the unit step:

$$\delta[n] = u[n] - u[n - 1]. \tag{1.7}$$

The general form of the *real sinusoidal sequence* with constant amplitude is

$$x[n] = A\cos(\omega_0 n + \phi), \quad -\infty < n < \infty, \tag{1.8}$$

where A, ω_0 and ϕ are real numbers. By analogy with continuous-time functions, ω_0 is called *angular frequency* of the sinusoid, and ϕ is called the *phase*. Note however that, since n is dimensionless, the

dimension of ω_0 is radians. Very often we will say that the dimension of n is "samples" and therefore we will specify the units of ω_0 to be radians/sample. If $x[n]$ has been sampled from a continuous-time sinusoid with a given sampling rate F_s, we will also use the term *normalized angular frequency*, since in this case ω_0 is the continuous-time angular frequency normalized with respect to F_s.

Another relevant numerical signal is constructed as the sequence of powers of a real or complex number α. Such sequences are termed *exponential sequences* and their general form is

$$x[n] = A\alpha^n, \quad -\infty < n < \infty, \tag{1.9}$$

where A and α are real or complex constant. When α is complex, $x[n]$ has real and imaginary parts that are exponentially weighted sinusoid. Specifically, if $\alpha = |\alpha|e^{j\omega_0}$ and $A = |A|e^{j\phi}$, then $x[n]$ can be expressed as

$$x[n] = |\,A\,||\,\alpha\,|^n \cdot e^{j(\omega_0 n+\phi)} = |\,A\,||\,\alpha\,|^n \cdot (\cos(\omega_0 n + \phi) + j\sin(\omega_0 n + \phi))\,. \tag{1.10}$$

Therefore $x[n]$ can be expressed as $x[n] = x_{Re}[n]+jx_{Im}[n]$, with $x_{Re}[n] = |\,A\,||\,\alpha\,|^n \cos(\omega_0 n+\phi)$ and $x_{Im}[n] = |\,A\,||\,\alpha\,|^n \sin(\omega_0 n + \phi)$. These sequences oscillate with an exponentially growing magnitude if $|\,\alpha\,| > 1$, or with an exponentially decaying magnitude if $|\,\alpha\,| < 1$. When $|\,\alpha\,| = 1$, the sequences $x_{Re}[n]$ and $x_{Im}[n]$ are real sinusoidal sequences with constant amplitude and $x[n]$ is referred to as a the *complex exponential sequence*.

An important property of real sinusoidal and complex exponential sequences is that substituting the frequency ω_0 with $\omega_0 + 2\pi k$ (with k integer) results in sequences that are indistinguishable from each other. This can be easily verified and is ultimately due to the fact that n is integer. We will see the implications of this property when discussing the Sampling Theorem in Sec. 1.4.1, for now we will implicitly assume that ω_0 varies in an interval of length 2π, e.g. $(-\pi, \pi]$, or $[0, 2\pi)$.

Real sinusoidal sequences and complex exponential sequences are also examples of a *periodic sequence*: we define a sequence to be periodic with period $N \in \mathbb{N}$ if it satisfies the equality $x[n] = x[n + kN]$, for $-\infty < n < \infty$, and for any $k \in \mathbb{Z}$. The *fundamental period* N_0 of a periodic signal is the smallest value of N for which this equality holds. In the case of Eq. (1.8) the condition of periodicity implies that $\omega_0 N_0 = 2\pi k$. If $k = 1$ satisfies this equality we can say that the sinusoidal sequence is periodic with period $N_0 = 2\pi/\omega_0$, but this is not always true: the period may be longer or, depending on the value of ω_0, the sequence may not be periodic at all.

1.2.1.3 Measures of discrete-time signals

We now define a set of useful metrics and measures of signals, and focus exclusively on digital signals. The first important metrics is *energy*: in physics, energy is the ability to do work and is measured in N·m or Kg·m^2/s^2, while in digital signal processing physical units are typically discarded and signals are renormalized whenever convenient. The total *energy* of a sequence $x[n]$ is then defined as:

$$\mathcal{E}_x = \sum_{n=-\infty}^{\infty} |x[n]|^2. \tag{1.11}$$

Note that an infinite-length sequence with finite sample values may or not have finite energy. The rate of transporting energy is known as *power*. The average power of a sequence $x[n]$ is then defined as the average energy per sample:

$$\mathcal{P}_x = \frac{\mathcal{E}_x}{N} = \frac{1}{N}\sum_{n=0}^{N-1} |x[n]|^2. \tag{1.12}$$

Another common description of a signal is its *root mean square (RMS)* level. The RMS level of a signal $x[n]$ is simply $\sqrt{\mathcal{P}_x}$. In practice, especially in audio, the RMS level is typically computed after subtracting out any nonzero mean value, and is typically used to characterize periodic sequences in which $\mathcal{P}_x$ is computed over a cycle of oscillation: as an example, the RMS level of a sinusoidal sequence $x[n] = A\cos(\omega_0 n + \phi)$ is $A/\sqrt{2}$.

In the case of sound signals, $x[n]$ will typically represent a sampled *acoustic pressure* signal. As a pressure wave travels in a medium (e.g., air), the RMS power is distributed all along the surface of the wavefront so that the appropriate measure of the strength of the wave is power per unit area of wavefront, also known as *intensity*.

Intensity is still proportional to the RMS level of the acoustic pressure, and relates to the sound level perceived by a listener. However, the usual definition of *sound pressure level (SPL)* does not directly use intensity. Insted the SPL of a pressure signal is measured in *decibels (dB)*, and is defined as

$$SPL = 10\log_{10}(I/I_0) \quad \text{(dB)}, \tag{1.13}$$

where I and I_0 are the RMS intensity of the signal and a reference intensity, respectively. In particular, in an *absolute dB scale* I_0 is chosen to be the smallest sound intensity that can be heard (more on this in Chapter *Auditory based processing*). The function of the dB scale is to transform ratios into differences: if I_2 is twice I_1, then $SPL_2 - SPL_1 = 3$ dB, no matter what the actual value of I_1 might be.[1]

Because sound intensity is proportional to the square of the RMS pressure, it is easy to express level differences in terms of pressure ratios:

$$SPL_2 - SPL_1 = 10\log_{10}(p_2^2/p_1^2) = 20\log_{10}(p_2/p_1) \quad \text{(dB)}. \tag{1.14}$$

Therefore, depending on the physical quantity which is being used the prefactor 20 or 10 may be employed in a decibel calculation. To resolve the uncertainty of which is the correct one, note that there are two kinds of quantities for which a dB scale is appropriate: "energy-like" quantities and "dynamical" quantities. An energy-like quantity is real and never negative: examples of such quantities are acoustical energy, intensity or power, electrical energy or power, optical luminance, etc., and the appropriate prefactor for these quantities in a dB scale is 10. Dynamical quantities may be positive or negative, or even complex in some representations: examples of such quantities are mechanical displacement or velocity, acoustical pressure, velocity or volume velocity, electrical voltage or current, etc., and the appropriate prefactor for these quantities in a dB scale is 20 (since they have the property that their squares are energy-like quantities).

1.2.1.4 Random signals

1.2.2 Discrete-time systems

Signal processing systems can be classified along the same lines used in Sec. 1.2 to classify signals. Here we are interested in discrete-time systems, that act on sequences and produce sequences as output.

1.2.2.1 Basic systems and block schemes

We define a discrete-time system as a transformation $\mathcal{T}$ that maps an *input sequence* $x[n]$ into an *output sequence* $y[n]$:

$$y[n] = \mathcal{T}\{x\}[n]. \tag{1.15}$$

[1] This is a special case of the Weber-Fechner law, which attempts to describe the relationship between the physical magnitudes of stimuli and the perceived intensity of the stimuli: the law states that this relation is logaritmic: if a stimulus varies as a geometric progression (i.e. multiplied by a fixed factor), the corresponding perception is altered in an arithmetic progression (i.e. in additive constant amounts).

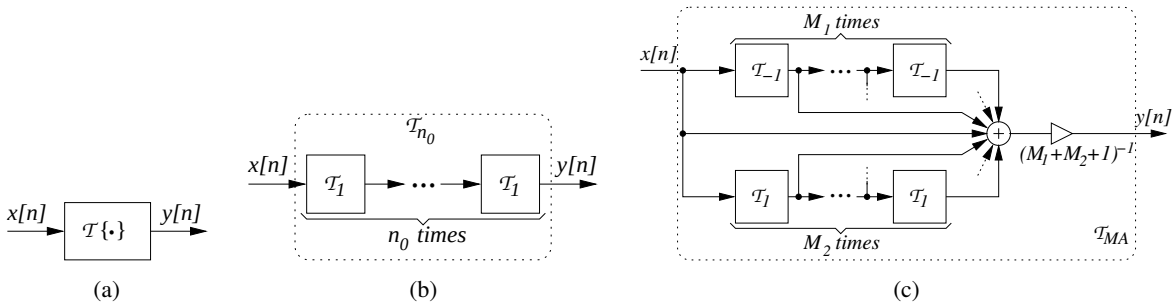


Figure 1.2: *Block schemes of discrete-time systems; (a) a generic system $\mathcal{T}\{\cdot\}$; (b) ideal delay system $\mathcal{T}_{n_0}$; (c) moving average system $\mathcal{T}_{MA}$. Note the symbols used to represent sums of signals and multiplication by a constant (these symbols will be formally introduced in Sec. 1.6.3).*

Discrete-time systems are typically represented pictorially through *block schemes* that depict the signal flow graph. As an example the block scheme of Fig.1.2(a) represents the generic discrete-time system of Eq. (1.15).

The simplest concrete example of a discrete-time system is the *ideal delay* system $\mathcal{T}_{n_0}$, defined as

$$y[n] = \mathcal{T}_{n_0}\{x\}[n] = x[n - n_0], \tag{1.16}$$

where the integer n_0 is the delay of the system and can be both positive and negative: if $n_0 > 0$ then y corresponds to a time-delayed version of x, while if $n_0 < 0$ then the system operates a time advance. This system can be represented with the block-scheme of Fig. 1.2(b): this block-scheme also provides an example of *cascade connection* of systems, based on the trivial observation that the system $\mathcal{T}_{n_0}$ can be seen as cascaded of n_0 unit delay systems $\mathcal{T}_1$.

A slightly more complex example is the *moving average* system $\mathcal{T}_{MA}$, defined as

$$y[n] = \mathcal{T}_{MA}\{x\}[n] = \frac{1}{M_1 + M_2 + 1} \sum_{k=-M_1}^{M_2} x[n - k]. \tag{1.17}$$

The n-th sample of the sequence y is the average of $(M_1 + M_2 + 1)$ samples of the sequence x, centered around the sample $x[n]$, hence the name of the system. Fig. 1.2(c) depicts a block scheme of the moving average system: in this case all the branches carrying the shifted versions of $x[n]$ form a *parallel connection* in which they are all summed up and subsequently multiplied by the factor $1/(M_1+M_2+1)$.

1.2.2.2 Classes of discrete-time systems

Classes of systems are defined by placing constraints on the properties of the transformation $\mathcal{T}\{\cdot\}$. Doing so often leads to very general mathematical representations.

We define a system to be *memoryless* if the output sequence $y[n]$ at every value of n depends only on the value of the input sequence $x[n]$ at the same value of n. As an example, the system $y[n] = \sin(x[n])$ is a memoryless system. On the other hand, the ideal delay system and the moving average system described in the previous section are not memoryless: these systems are referred to as *having memory*, since they must "remember" past (or even future) values of the sequence x in order to compute the "present" output $y[n]$.

We define a system to be *linear* if it satisfies the principle of superposition. If $y_1[n]$, $y_2[n]$ are the responses of a system $\mathcal{T}$ to the inputs $x_1[n]$, $x_2[n]$, respectively, then $\mathcal{T}$ is linear if and only if

$$\mathcal{T}\{a_1x_1 + a_2x_2\}[n] = a_1\mathcal{T}\{x_1\}[n] + a_2\mathcal{T}\{x_2\}[n], \tag{1.18}$$

for any pair of arbitrary constants a_1 and a_2. Equivalently we say that a linear system possesses an additive property and a scaling property. As an example, the ideal delay system and the moving average system described in the previous section are linear systems. On the other hand, the memoryless system $y[n] = \sin(x[n])$ discussed above is clearly non-linear.

We define a system to be *time-invariant* (or *shift-invariant*) if a time shift of the input sequence causes a corresponding shift in the output sequence. Specifically, let $y = \mathcal{T}\{x\}$. Then $\mathcal{T}$ is time-invariant if and only if

$$\mathcal{T}\left\{\mathcal{T}_{n_0}\{x\}\right\}[n] = y[n - n_0] \quad \forall n_0, \tag{1.19}$$

where $\mathcal{T}_{n_0}$ is the ideal delay system defined previously. This relation between the input and the output must hold for any arbitrary input sequence x and its corresponding output. All the systems that we have examined so far are time-invariant. On the other hand, an example of non-time-invariant system is $y[n] = x[Mn]$ (with $M \in \mathbb{N}$). This system creates y by selecting one every M samples of x. One can easily see that $\mathcal{T}\left\{\mathcal{T}_{n_0}\{x\}\right\}[n] = x[Mn - n_0]$, which is in general different from $y[n - n_0] = x[M(n - n_0)]$.

We define a system to be *causal* if for every choice of n_0 the output sequence sample $y[n_0]$ depends only on the input sequence samples $x[n]$ with $n \leq n_0$. This implies that, if $y_1[n]$, $y_2[n]$ are the responses of a causal system to the inputs $x_1[n]$, $x_2[n]$, respectively, then

$$x_1[n] = x_2[n] \quad \forall n < n_0 \qquad \Rightarrow \qquad y_1[n] = y_2[n] \quad \forall n < n_0. \tag{1.20}$$

The moving average system discussed in the previous section is an example of a non-causal systems, since it needs to know M_1 "future" values of the input sequence in order to compute the current value $y[n]$. Apart from this, all the systems that we have examined so far are causal.

We define a system to be *stable* if and only if every bounded input sequence produces a bounded output sequence. A sequence $x[n]$ is said to be bounded if there exist a positive constant B_x such that

$$|\, x[n] \,| \leq B_x \quad \forall n. \tag{1.21}$$

Stability then requires that for such an input sequence there exists a positive constant B_y such that $|\, y[n] \,| \leq B_y \; \forall n$. This notion of stability is often referred to as *bounded-input bounded-output (BIBO)* stability. All the systems that we have examined so far are BIBO-stable. On the other hand, an example of unstable system is $y[n] = \sum_{k=-\infty}^{n} x[k]$. This is called the *accumulator* system, since $y[n]$ accumulates the sum of all past values of x. In order to see that the accumulator system is not stable it is sufficient to verify that $y[n]$ is not bounded when $x[n]$ is the step sequence.

1.2.3 Linear Time-Invariant Systems

Linear-time invariant (LTI) are a particularly relevant class of systems. A LTI system is any system that is both linear and time-invariant according to the definitions given in the previous section. As we will see in this section, LTI systems are mathematically easy to analyze and to characterize.

1.2.3.1 Impulse response and convolution

Let $\mathcal{T}$ be a LTI system, $y[n] = \mathcal{T}\{x\}[n]$ be the output sequence given a generic input x, and $h[n]$ the *impulse response* of the system, i.e. $h[n] = \mathcal{T}\{\delta\}[n]$. Now, recall that every sequence $x[n]$ can be

represented as a linear combination of delayed impulses (see Eq. (1.4)). If we use this representation and exploit the linearity and time-invariance properties, we can write:

$$y[n] = \mathcal{T}\left\{\sum_{k=-\infty}^{+\infty} x[k]\delta[n-k]\right\} = \sum_{k=-\infty}^{+\infty} x[k]\mathcal{T}\{\delta[n-k]\} = \sum_{k=-\infty}^{+\infty} x[k]h[n-k], \tag{1.22}$$

where in the first equality we have used the representation (1.4), in the second equality we have used the linearity property, and in the last equality we have used the time-invariance property.

Equation (1.22) states that a LTI system can be completely characterized by its impulse response $h[n]$, since the response to *any* imput sequence $x[n]$ can be written as $\sum_{k=-\infty}^{\infty} x[n]h[n-k]$. This can be interpreted as follows: the k-th input sample, seen as a single impulse $x[k]\delta[n-k]$, is transformed by the system into the sequence $x[k]h[n-k]$, and for each k these sequences are summed up to form the overall output sequence $y[n]$.

The sum on the right-hand side of Eq. (1.22) is called *convolution sum* of the sequences $x[n]$ and $h[n]$, and is usually denoted with the sign $*$. Therefore we have just proved that a LTI system $\mathcal{T}$ has the property

$$y[n] = \mathcal{T}\{x\}[n] = (x * h)[n]. \tag{1.23}$$

Let us consider again the systems defined in the previous sections: we can find their impulse responses through the definition, i.e. by computing their response to an ideal impulse $\delta[n]$. For the ideal delay system the impulse response is simply a shifted impulse:

$$h_{n_0}[n] = \delta[n-n_0]. \tag{1.24}$$

The impulse response of the moving average system is easily computed as

$$h[n] = \frac{1}{M_1+M_2+1}\sum_{k=-M_1}^{M_2}\delta[n-k] = \begin{cases} \frac{1}{M_1+M_2+1}, & -M_1 < n < M_2, \\ 0, & \text{elsewhere.} \end{cases} \tag{1.25}$$

Finally the accumulator system has the following impulse response:

$$h[n] = \sum_{k=-\infty}^{n}\delta[k] = \begin{cases} 1, & n \geq 0, \\ 0, & n < 0. \end{cases} \tag{1.26}$$

There is a fundamental difference between these impulse responses. The first two responses have a finite number of non-zero samples (1 and M_1+M_2+1, respectively): systems that possess this property are called *finite impulse response (FIR)* systems. On the other hand, the impulse response of the accumulator has an infinite number of non-zero samples: systems that possess this property are called *infinite impulse response (IIR)* systems.

1.2.3.2 Properties of LTI systems

Since the convolution sum of Eq. (1.23) completely characterizes a LTI system, the most relevant properties of this class of systems can be understood by inspecting properties of the convolution operator. Clearly convolution is *linear*, otherwise $\mathcal{T}$ would not be a linear system, which is by hypothesis. Convolution is also *associative*:

$$(x * (h_1 * h_2))[n] = ((x * h_1) * h_2)[n]. \tag{1.27}$$

Moreover convolution is *commutative*:

$$(x * h)[n] = \sum_{k=-\infty}^{\infty} x[n]h[n-k] = \sum_{m=-\infty}^{\infty} x[n-m]h[m] = (h * x)[n], \tag{1.28}$$

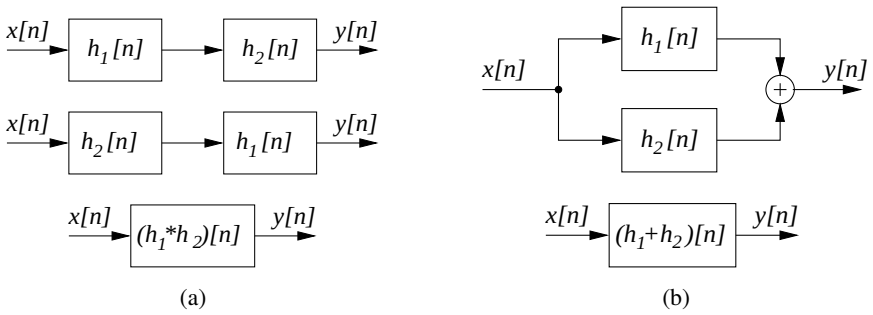


Figure 1.3: *Properties of LTI system connections, and equivalent systems; (a) cascade, and (b) parallel connections.*

where we have substituted the variable $m = n - k$ in the sum. This property implies that a LTI system with input $h[n]$ and impulse response $x[n]$ will have the same ouput of a LTI system with input $x[n]$ and impulse response $h[n]$. More importantly, associativity and commutativity have implications on the properties of *cascade connections* of systems. Consider the block scheme in Fig. 1.3(a) (upper panel): the output from the first block is $x * h_1$, therefore the final output is $(x * h_1) * h_2$, which equals both $(x * h_2) * h_1$ and $x * (h_1 * h_2)$. As a result the three block schemes in Fig. 1.3(a) represent three systems with the same impulse response.

Linearity and commutativity imply that the convolution is *distributive* over addition. From the definition (1.23) it is straightforward to prove that

$$(x * (h_1 + h_2))\,[n] = (x * h_1)[n] + (x * h_2)[n]. \tag{1.29}$$

Distributivity has implications on the properties of *parallel connections* of systems. Consider the block scheme in Fig. 1.3(b) (upper panel): the final output is $(x * h_1) + (x * h_2)$, which equals $x * (h_1 + h_2)$. As a result the two block schemes in Fig. 1.3(a) represent two systems with the same impulse response.

In the case of a LTI system, the notions of causality and stability given in the previous sections can also be related to properties of the impulse response. As for causality, it is a straightforward exercise to show that a LTI system is causal if and only if

$$h[n] = 0 \quad \forall n < 0. \tag{1.30}$$

For this reason, sequences that satisfy the above condition are usually termed *causal sequences*.

As for stability, recall that a system is BIBO-stable if any bounded input produces a bounded output. The response of a LTI system to a bounde input $x[n] \leq B_x$ is

$$|\,y[n]\,| = \left| \sum_{k=-\infty}^{+\infty} x[k]h[n-k] \right| \leq \sum_{k=-\infty}^{+\infty} |\,x[k]\,|\,|\,h[n-k]\,| \leq B_x \sum_{k=-\infty}^{+\infty} |\,h[n-k]\,|\,. \tag{1.31}$$

From this chain of inequalities we find that a sufficient condition for the stability of the system is

$$\sum_{k=-\infty}^{+\infty} |\,h[n-k]\,| = \sum_{k=-\infty}^{+\infty} |\,h[k]\,| < \infty. \tag{1.32}$$

One can prove that this is also a necessary condition for stability. Assume that Eq. (1.32) does not hold and define the input $x[n] = h^*[-n]/\,|\,h[n]\,|$ for $h[n] \neq 0$ ($x = 0$ elsewhere): this input is bounded by

unity, however one can immediately prove that $y[0] = \sum_{k=-\infty}^{+\infty} |\, h[k] \,| = +\infty$. In conclusion, a LTI system is stable if and only if h is absolutely summable, or $h \in \mathsf{L}^1(\mathbb{Z})$. A direct consequence of this property is that FIR systems are always stable, while IIR systems may not be stable.

Using the properties demonstrated in this section, we can look back at the impulse responses of Eqs. (1.24,1.25,1.26), and we can immediately immediately prove whether they are stable and causal.

1.2.3.3 Constant-coefficient difference equations

Consider the following *constant-coefficient difference equation*:

$$\sum_{k=0}^{N} a_k y[n-k] = \sum_{k=0}^{M} b_k x[n-k]. \tag{1.33}$$

Question: given a set of values for $\{a_k\}$ and $\{b_k\}$, does this equation define a LTI system? The answer is no, because a given input $x[n]$ does not univocally determine the output $y[n]$. In fact it is easy to see that, if $x[n], y[n]$ are two sequences satisfying Eq. (1.33), then the equation is satisfied also by the sequences $x[n], y[n] + y_h[n]$, where y_h is *any* sequence that satisfies the *homogeneous equation*:

$$\sum_{k=0}^{N} a_k y_h[n-k] = 0. \tag{1.34}$$

One could show that y_h has the general form $y_h[n] = \sum_{m=1}^{N} A_m z_m^n$, where the z_m's are roots of the polynomial $\sum_{k=0}^{N} a_k z^k$ (this can be verified by substituting the general form of y_h into Eq. (1.34)).

The situation is very much like that of linear constant-coefficient differential equations in continuous-time: since y_h has N undetermined coefficients A_m, we must specify N additional constraints in order for the equation to admit a unique solution. Typically we set some *initial conditions*. For Eq. (1.33), an initial condition is a set of N consecutive "initial" samples of $y[n]$. Suppose that the samples $y[-1], y[-2], \ldots y[-N]$ have been fixed: then all the infinite remaining samples of y can be recursively determined through the recurrence equations

$$y[n] = \begin{cases} -\sum_{k=1}^{N} \frac{a_k}{a_0} y[n-k] + \sum_{m=0}^{M} \frac{b_m}{a_0} x[n-m], & n \geq 0, \\ -\sum_{k=0}^{N-1} \frac{a_k}{a_0} y[n+N-k] + \sum_{m=0}^{M} \frac{b_m}{a_0} x[n+N-m], & n \leq -N-1. \end{cases} \tag{1.35}$$

In particular, $y[0]$ is determined with the above equation using the initial values $y[-1], y[-2], \ldots y[-N]$, then $y[1]$ is determined using the values $y[-0], y[-1], \ldots y[-N+1]$, and so on.

Another way of guaranteeing that Eq. (1.33) specifies a unique solution is requiring that the LTI system is also *causal*. If we look back at the definition of causality, we see that in this context it implies that for an input $x[n] = 0\ \forall n < n_0$, then $y[n] = 0\ \forall n < n_0$. Then again we have sufficient initial conditions to recursively compute $y[n]$ for $n \geq n_0$: in this case can speak of *initial rest conditions*.

All the LTI systems that we have examined so far can actually be written in the form (1.33). As an example, let us examine the accumulator system: we can write

$$y[n] = \sum_{k=-\infty}^{n} x[k] = x[n] + \sum_{k=-\infty}^{n-1} x[k] = x[n] + y[n-1], \tag{1.36}$$

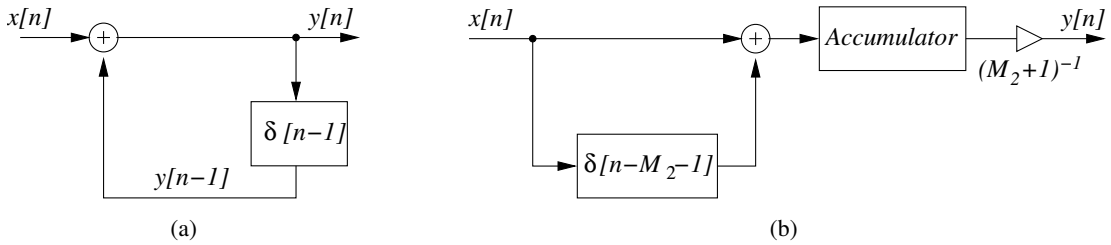


Figure 1.4: *Block schemes for constant-coefficient difference equations representing (a) the accumulator system, and (b) the moving average system.*

where in the second equality we have simply separated the term $x[n]$ from the sum, and in the third equality we have applied the definition of the accumulator system for the output sample $y[n-1]$. Therefore, input and output of the accumulator system satisfy the equation

$$y[n] - y[n-1] = x[n], \tag{1.37}$$

which is in form (1.33) with $N = 1,\ M = 0$ and with $a_0 = 1,\ a_1 = -1, b_0 = 1$. This also means that we can implement the accumulator with the block scheme of Fig. 1.4(a).

As a second example, consider the moving average system, with $M_1 = 0$ (so that the system is causal). Then Eq. (1.17) is already a constant-coefficient difference equation. But we can also represent the system with a different equation by noting that

$$y[n] = \frac{1}{M_2+1}\sum_{k=0}^{M_2} x[n-k] = \frac{1}{M_2+1}\sum_{-\infty}^{n} \left(x[n] - x[n-M_2-1]\right). \tag{1.38}$$

Now we note that the sum on the right-hand side represents an accumulator applied to the signal $x_1[n] = (x[n] - x[n-M_2-1])$. Therefore we can apply Eq. (1.37) and write

$$y[n] - y[n-1] = \frac{1}{M_2+1}x_1[n] = \frac{1}{M_2+1}\left(x[n] - x[n-M_2-1]\right). \tag{1.39}$$

We have then found a totally different equation, which is still in the form (1.33) and still represents the moving average system. The corresponding block scheme is given in Fig. 1.4(b). This example also shows that different equations of the form (1.33) can represent the same LTI system.

1.3 Signal generators

In this section we describe methods and algorithms used to directly generate a discrete-time signal. Specifically we will examine periodic waveform generators and noise generators, which are both particularly relevant in audio applications.

1.3.1 Digital oscillators

Many relevant musical sounds are almost periodic in time. The most direct method for synthesizing a periodic signal is repeating a single period of the corresponding waveform. An algorithm that implements this method is called oscillator. The simplest algorithm consists in computing the appropriate value of the waveform for every sample, assuming that the waveform can be approximately described through a polynomial or rational truncated series. However this is definitely not the most efficient approach. More efficient algorithms are presented in the remainder of this section.

1.3.1.1 Table lookup oscillator

A very efficient approach is to pre-compute the samples of the waveform, store them in a table which is usually implemented as a circular buffer, and access them from the table whenever needed. If a copy of one period of the desired waveform is stored in such a *wavetable*, a periodic waveform can be generated by cycling over the wavetable with the aid of a circular pointer. When the pointer reaches the end of the table, it wraps around and points again at the beginning of the table.

Given a table of length L samples, the period T_0 of the generated waveform depends on the sampling period T_s at which samples are read. More precisely, the period is given by $T_0 = LT_s$, and consequently the fundamental frequency is $f_0 = F_s/L$. This implies that in order to change the frequency (while maintaing the sample sampling rate), we would need the same waveform to be stored in tables of different lengths.

A better solution is the following. Imagine that a single wavetable is stored, composed of a very large number L of equidistant samples of the waveform. Then for a given sampling rate F_s and a desired signal frequency f_0, the number of samples to be generated in a single cycle is F_s/f_0. From this, we can define the *sampling increment* (SI), which is the distance in the table between two samples at subsequent instants. The SI is given by the following equation:

$$SI = \frac{L}{F_s/f_0} = \frac{f_0 L}{F_s}. \tag{1.40}$$

Therefore the SI is proportional to f_0. Having defined the sampling increment, samples of the desired signal are generated by reading one every SI samples of the table. If the SI is not an integer, the closest sample of the table will be chosen (obviously, the largest L, the better the approximation). In this way, the oscillator resample the table to generate a waveform with different fundamental frequencies.

M-1.1

Implement in Matlab a circular look-up from a table of length L and with sampling increment SI.

M-1.1 Solution

```
phi=mod(phi +SI,L);
s=tab[round(phi)];
```

where `phi` is a state variable indicating the reading point in the table, `A` is a scaling parameter, `s` is the output signal sample. The function `mod(x,y)` computes the remainder of the division x/y and is used here to implement circular reading of the table. Notice that `phi` can be a non integer value. In order to use it as array index, it has to be truncated, or rounded to the nearest integer (as we did in the code above). A more accurate output can be obtained by linear interpolation between adjacent table values.

1.3.1.2 Recurrent sinusoidal signal generators

Sinusoidal signals can be generated also by recurrent methods. A first method is based on the following equation:

$$y[n+1] = 2\cos(\omega_0)y[n] - y[n-1] \tag{1.41}$$

where $\omega_0 = 2\pi f_0/F_s$ is the normalized angular frequency of the sinusoid. Then one can prove that given the initial values $y[0] = \cos\phi$ and $y[-1] = \cos(\phi - \omega_0)$ the generator produces the sequence

$$y[n] = \cos(\omega_0 + \phi). \tag{1.42}$$

In particular, with initial values $y[0] = 1$ and $y[-1] = \cos\omega_0$ the generator produces the sequence $y[n] = \cos(\omega_0 n)$, while with initial conditions $y[0] = 0$ and $y[-1] = -\sin\omega_0$ it produces the sequence $y[n] = \sin(\omega_0 n)$. This property can be justified by recalling the trigonometric relation $\cos\omega_0 \cdot \cos\phi = 0.5[\cos(\phi + \omega_0) + \cos(\phi - \omega_0)]$.

A second recursive method for generating sinusoidal sequence combines both the sinusoidal and cosinusoidal generators and is termed *coupled form*. It is described by the equations

$$\begin{aligned} x[n+1] &= \cos\omega_0 \cdot x[n] - \sin\omega_0 \cdot y[n], \\ y[n+1] &= \sin\omega_0 \cdot x[n] + \cos\omega_0 \cdot y[n]. \end{aligned} \tag{1.43}$$

With $x[0] = 1$ and $y[0] = 0$ the sequences $x[n] = \cos(\omega_0 n)$ and $y[n] = \sin(\omega_0 n)$ are generated. This property can be verified by noting that for the complex exponential sequence the trivial relation $e^{j\omega_0(n+1)} = e^{j\omega_0} e^{j\omega_0 n}$ holds. From this relation, the above equations are immediately proved by calling $x[n]$ and $y[n]$ the real and imaginary parts of the complex exponential sequence, respectively.

A major drawback of both these recursive methods is that they are not robust against quatization. Small quantization errors in the computation will cause the generated signals either to grow exponentially or to decay rapidly into silence. To avoid this problem, a periodic re-initialization is advisable. It is possible to use a slightly different set of coefficients to produce absolutley stable sinusoidal waveforms

$$\begin{aligned} x[n+1] &= x[n] - c \cdot y[n], \\ y[n+1] &= c \cdot x[n+1] + y[n], \end{aligned} \tag{1.44}$$

where $c = 2\sin(\omega_0/2)$. With $x[0] = 1$ and $y[0] = c/2$ we have $x[n] = \cos(\omega_0 n)$.

1.3.1.3 Control signals and envelope generators

Amplitude and frequency of a sound are usually required to be time-varying parameters. Amplitude control can be needed to define suitable sound envelopes, or to create effects such as *tremolo* (quasi-periodic amplitude variations around an average value). Frequency control can be needed to simulate continuous gliding between two tones (*portamento*, in musical terms), or to obtain subtle pitch variations in the sound attack/release, or to create effects such as *vibrato* (quasi-periodic pitch variations around an average value), and so on. We then want to construct a digital oscillator of the form

$$x[n] = a[n] \cdot \text{tab}\{\phi[n]\}, \tag{1.45}$$

where $a[n]$ scales the amplitude of the signal, while the phase $\phi[n]$ relates to the *instantaneous frequency* $f_0[n]$ of the signal: if $f_0[n]$ is not constant, then $\phi[n]$ does not increase linearly in time. Figure 1.5(a) shows the symbol usually adopted to depict an oscillator with fixed waveform and varying amplitude and frequency.

The signals $a[n]$, and $f_0[n]$ are usually referred to as *control signals*, as opposed to *audio signals*. The reason for this distinction is that control signals vary on a much slower time-scale than audio signals (as an example, a musical *vibrato* usually have a frequency of a no more than ~ 5 Hz). Accordingly, many sound synthesis languages define control signals at a different (smaller) rate than the audio sampling rate F_s. This second rate is called *control rate*, or *frame rate*: a frame is a time window with pre-defined length (e.g. 5 or 50 ms), in which the control signals can be reasonably assumed to have small variations. We will use the notation F_c for the control rate.

Suitable control signals can be synthesized using *envelope generators*. An envelope generator can be constructed through the table-lookup approach described previously. In this case however the table will be read only once since the signal to be generated is not periodic. Given a desired duration (in seconds) of the control signal, the appropriate sampling increment will be chosen accordingly.

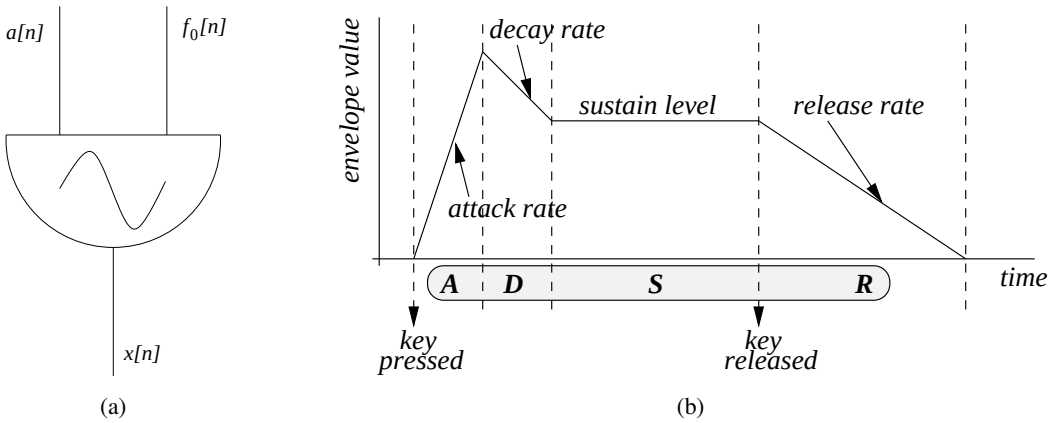


Figure 1.5: *Controlling a digital oscillator; (a) symbol of the digital controlled in amplitude and frequency; (b) example of an amplitude control signal generated with an ADSR envelope.*

Alternatively, envelope generators can be constructed by specifying values of control signals at a few control points and interpolating the signal in between them. In the simplest formulation, linear interpolation is used. In order to exemplify this approach, we discuss the so-called *Attack, Decay, Sustain, and Release (ADSR)* envelope typically used in sound synthesis applications to describe the time-varying amplitude $a[n]$. This envelope is shown in Fig. 1.5(b)): amplitude values are specified only at the boundaries between ADSR phases, and within each phase the signal varies linearly.

The attack and release phases mark the identity of the sound, while the central phases are associated with the steady-state portion of the sound. Therefore, in order to synthesize two sounds with the similar identity (or timbre) but different durations, it is advisable to only slightly modify the duration of attack and release, while the decay and especially sustain can be lengthened more freely.

M-1.2

Write a function that realizes a line-segment envelope generator. The input to the function are a vector of time instants and a corresponding vector of envelope values.

M-1.2 Solution

```
function env = envgen(t,a,method);      %t= vector of control time instants
                                        %a= vector of envelope vaues

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

if (nargin<3) method='linear'; end

frt=floor(t*Fs/SpF+1);                  %control time instants as frame numbers
nframes=frt(length(frt));               %total number of frames
env=interp1(frt,a,[1:nframes],method); %linear (or other method) interpolation
```

The envelope shape is specified by break-points, described as couples (time instant (sec) and amplitude). The function generates the envelope at frame rate. Notice that the interpolation function `interp1` allows to easily use cubic of $spline$ interpolations.

The use of waveform and envelope generators allows to generate quasi periodic sounds with very limited hardware and constitutes the building block of many more sophisticated algorithms.

M-1.3

Assume that a function `sinosc(t0,a,f,ph0)` realizes a sinusoidal oscillator controlled in frequency and amplitude, with `t0` initial time, `a,f` frame-rate amplitude and frequency vectors, and `ph0` initial phase (see example M-1.4). Then generate a sinusoid with varying amplitude and constant frequency.

M-1.3 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 %%%
slength=2;                                          %sound length (in s)
nframes=slength*Fs/SpF;                             %total no. of frames
f=50*ones(1,nframes);                               %constant frequency (Hz)
a=envgen([0,.2,3,3.5,4],[0,1,.8,.5,0],'linear'); %ADSR amp. envelope

s=sinosc(0,a,f,0);       % compute sound signal
```

Note the structure of this simple example: in the "headers" section some global parameters are defined, that need to be known also to auxiliary functions; a second section defines the control parameters, and finally the audio signal is computed.

1.3.1.4 Frequency controlled oscillators

While realizing an amplitude modulated oscillator is quite straightforward, realizing a frequency modulated oscillator requires some more work. First of all we have to understand what is the instantaneous frequency of such an oscillator and how it relates to the phase function ϕ. This can be better understood in the continuous time domain. When the oscillator frequency is constant the phase is a linear function of time, $\phi(t) = 2\pi f_0 t$. In the more general case in which the frequency varies at frame rate, the following equation holds:

$$f_0(t) = \frac{1}{2\pi}\frac{d\phi}{dt}(t), \tag{1.46}$$

which simply says that the instantaneous angular frequency $\omega_0(t) = 2\pi f_0(t)$ is the instantaneous angular velocity of the time-varying phase $\phi(t)$. If $f_0(t)$ is varying slowly enough (i.e. it is varying at frame rate), we can say that in the k-th frame the following first-order approximation holds:

$$\frac{1}{2\pi}\frac{d\phi}{dt}(t) = f_0(t) \sim f_0(t_k) + F_c\left[f_0(t_{k+1}) - f_0(t_k)\right] \cdot (t - t_k), \tag{1.47}$$

where t_k, t_{k+1} are the initial instants of frames k and $k+1$, respectively. The term $F_c\left[f_0(t_{k+1}) - f_0(t_k)\right]$ approximates the derivative df_0/dt inside the kth frame. We can then find the phase function by integrating equation (1.47):

$$\phi(t) = \phi(t_k) + 2\pi f_0(t_k)(t - t_k) + 2\pi F_c[f_0(t_{k+1}) - f_0(t_k)]\frac{(t - t_k)^2}{2}. \tag{1.48}$$

From this equation, the discrete-time signal $\phi[n]$ can be computed within the kth frame, i.e. for the time indexes $(k-1)\cdot \mathrm{SpF} + n$, with $n = 0 \ldots (\mathrm{SpF} - 1)$.

In summary, Eq. (1.48) allows to compute $\phi[n]$ at sample rate inside the kth frame, given the frame rate frequency values $f_0(t_k)$ and $f_0(t_{k+1})$. The key ingredient of this derivation is the linear interpolation (1.47).

M-1.4

Realize the `sinosc(t0,a,f,ph0)` function that we have used in M-1.3. Use equation (1.48) to compute the phase given the frame-rate frequency vector `f`.

M-1.4 Solution

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

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

nframes=length(a);      %total number of frames
if (length(f)==1)       f=f*ones(1,nframes);      end
if (length(f)~=nframes) error('wrong f length!'); end

s=zeros(1,nframes*SpF);              %initialize signal vector to 0
lasta=a(1); lastf=f(1); lastph=ph0;  %initialize amplitude, frequency, phase

for i=1:nframes                  %cycle on the frames
   naux=1:SpF;                   %count samples within frame
   %%%%%%%%%%%% compute amplitudes and phases within frame %%%%%%%%%%%%
   ampl=lasta + (a(i)-lasta)/SpF.*naux;
   phase=lastph +pi/Fs.*naux.*(2*lastf +(1/SpF)*(f(i)-lastf).*naux);
   %%%%%%%%%%%%%%%% read from table %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   s(((i-1)*SpF+1):i*SpF)=ampl.*cos(phase); %read from table
   %%%%%%%%% save last values of amplitude, frequency, phase
   lasta=a(i); lastf=f(i); lastph=phase(SpF);
end
s=[zeros(1,round(t0*Fs)) s]; %add initial silence of t0 sec.
```

Both the amplitude `a` and frequency `f` envelopes are defined at frame rate and are interpolated at sample rate inside the function body. Note in particular the computation of the `phase` vector within each frame.

We can finally listen to a sinudoidal oscillator controlled both in amplitude and in frequency.

M-1.5

Synthesize a sinusoid modulated both in amplitude and frequency, using the functions `sinosc` and `envgen`.

M-1.5 Solution

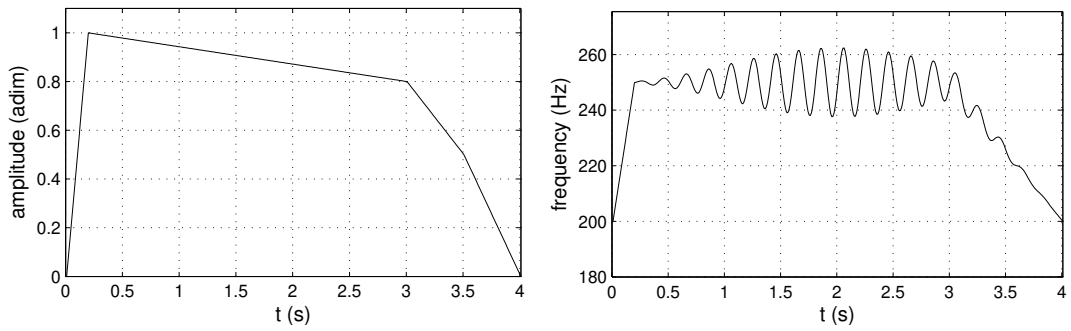
```
global Fs; global SpF; %global variables: sample rate, samples-per-frame

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

%%% define controls %%%
a=envgen([0,.2,3,3.5,4],[0,1,.8,.5,0],'linear');  %ADSR amp. envelope
f=envgen([0,.2,3,4],[200,250,250,200],'linear');   %pitch envelope
f=f+max(f)*0.05*...                %pitch envelope with vibrato added
      sin(2*pi*5*(SpF/Fs)*[0:length(f)-1]).*hanning(length(f))';

%%% compute sound %%%
s=sinosc(0,a,f,0);
```

Amplitude `a` and frequency `f` control signals are shown in Fig. 1.6.



(a) (b)

Figure 1.6: *Amplitude (a) and frequency (b) control signals*

1.3.2 Noise generators

Up to now, we have considered signals whose behavior at any instant is supposed to be perfectly knowable. These signals are called deterministic signals. Besides these signals, *random signals* of unknown or only partly known behavior may be considered. For random signals, only some general characteristics, called statistical properties, are known or are of interest. The statistical properties are characteristic of an entire signal class rather than of a single signal. A set of random signals is represented by a random process. Particular numerical procedures simulate random processes, producing sequences of random (or more precisely, pseudorandom) numbers.

Random sequences can be used both as signals (i.e., to produce white or colored noise used as input to a filter) and a control functions to provide a variety in the synthesis parameters most perceptible by the listener. In the analysis of natural sounds, some characteristics vary in an unpredictable way; their mean statistical properties are perceptibly more significant than their exact behavior. Hence, the addition of a random component to the deterministic functions controlling the synthesis parameters is often desirable. In general, a combination of random processes is used because the temporal organization of the musical parameters often has a hierarchical aspect. It cannot be well described by a single random process, but rather by a combination of random processes evolving at different rates. For example this technique is employed to generate $1/f$ noise.

1.3.2.1 White noise generators

The spread part of the spectrum is perceived as random noise. In order to generate a random sequence, we need a random number generator. There are many algorithms that generate random numbers, typically uniformly distributed over the standardized interval $[0, 1)$. However it is hard to find good random number generators, i.e. that pass all or most criteria of randomness. The most common is the so called *linear congruential* generator. It can produce fairly long sequences of independent random numbers, typically of the order of two billion numbers before repeating periodically. Given an initial number (seed) $I[0]$ inn the interval $0 \leq I[0] < M$, the algorithm is described by the recursive equations

$$\begin{aligned} I[n] &= (\, aI[n-1] + c\,) \bmod M \\ u[n] &= I[n]/M \end{aligned} \tag{1.49}$$

where a and c are two constants that should be chosen very carefully in order to have a maximal length sequence, i.e. long M samples before repetition. The actual generated sequence depends on the initial

value $I[0]$; that is way the sequence is called pseudorandom. The numbers are uniformly distributed over the interval $0 \leq u[n] < 1$. The mean is $E[u] = 1/2$ and the variance is $\sigma_u^2 = 1/12$. The transformation $s[n] = 2u[n] - 1$ generates a zero-mean uniformly distributed random sequence over the interval $[-1, 1)$. This sequence corresponds to a white noise signal because the generated numbers are mutually independent. The power spectral density is given by $S(f) = \sigma_u^2$. Thus the sequence contains all the frequencies in equal proportion and exhibits equally slow and rapid variation in time.

With a suitable choice of the coefficients a and b, it produces pseudorandom sequences with flat spectral density magnitude (white noise). Different spectral shapes ca be obtained using white noise as input to a filter.

M-1.6

A method of generating a Gaussian distributed random sequence is based on the central limit theorem, which states that the sum of a large number of independent random variables is Gaussian. As exercise, implement a very good approximation of a Gaussian noise, by summing 12 independent uniform noise generators.

If we desire that the numbers vary at a slower rate, we can generate a new random number every d sampling instants and hold the previous value in the interval (*holder*) or interpolate between two successive random numbers (interpolator). In this case the power spectrum is given by

$$S(f) = |H(f)|^2 \frac{\sigma_u^2}{d}$$

with

$$|H(f)| = \left| \frac{\sin(\pi f d/F_s)}{\sin(\pi f/F_s)} \right|$$

for the holder and

$$|H(f)| = \frac{1}{d} \left[\frac{\sin(\pi f d/F_s)}{\sin(\pi f/F_s)} \right]^2$$

for linear interpolation.

1.3.2.2 Pink noise generators

$1/f$ noise generators A so-called *pink noise* is characterized by a power spectrum that fall in frequency like $1/f$:

$$S(f) = \frac{A}{f}. \tag{1.50}$$

For this reason pink noise is also called $1/f$ *noise*. To avoid the infinity at $f = 0$, this behaviour is assumed valid for $f \geq f_{min}$, where f_{min} is a desired minimum frequency. The spectrum is characterized by a 3 db per octave drop, i.e. $S(2f) = S(f)/2$. The amount of power contained within a frequency interval $[f_1, f_2]$ is

$$\int_{f_1}^{f_2} S(f) df = A \ln \left(\frac{f_1}{f_2} \right)$$

This implies that the amount of power in any octave is the same. $1/f$ noise is ubiquitous in nature and is related to fractal phenomena. In audio domain it is known as pink noise. It represents the psychoacoustic equivalent of the white noise because he approximately excites uniformly the critical bands. The physical interpretation is a phenomenon that depends on many processes that evolve on different time scales. So a $1/f$ signal can be generated by the sum of several white noise generators that are filtered through first¡order filters having the time constants that are successively larger and larger, forming a geometric progression.

M-1.7

In the Voss $1/f$ noise generation algorithm, the role of the low pass filters is played by the hold filter seen in the previous paragraph. The $1/f$ noise is generated by taking the average of several periodically held generators $y_i[n]$, with periods forming a geometric progression $d_i = 2^i$, i.e.

$$y[n] = \frac{1}{M} \sum_{i=1}^{M} y_i[n] \tag{1.51}$$

The power spectrum does not have an exact $1/f$ shape, but it is close to it for frequencies $f \geq F_s/2^M$. As exercise, implement a $1/f$ noise generator and use it to assign the pitches to a melody.

M-1.8

The music derived from the $1/f$ noise is closed to the human music: it does not have the unpredictability and randomness of white noise nor the predictability of brown noise. $1/f$ processes correlate logarithmically with the past. Thus the averaged activity of the last ten events has as much influence on the current value as the last hundred events, and the last thousand. Thus they have a relatively long-term memory.
$1/f$ noise is a fractal one; it exhibits self-similarity, one property of the fractal objects. In a self-similar sequence, the pattern of the small details matches the pattern of the larger forms, but on a different scale. In this case, is used to say that 1/f fractional noise exhibits statistical self-similarity. The pink noise algorithm for generating pitches has become a standard in algorithmic music. Use the $1/f$ generator developed in M-1.7 to produce a fractal melody.

1.4 Spectral analysis of discrete-time signals

Spectral analysis is one of the powerful analysis tool in several fields of engineering. The fact that we can decompose complex signals with the superposition of other simplex signals, commonly sinusoid or complex exponentials, highlights some signal features that sometimes are very hard to discover otherwise. Furthermore, the decomposition on simpler functions in the frequency domain is very useful when we want to perform modifications on a signal, since it gives the possibility to manipulate single spectral components, which is hard if not impossible to do on the time-domain waveform.

A rigorous and comprehensive tractation of spectral analysis is out the scope of this book. In this section we introduce the *Discrete-Time Fourier Transform (DTFT)*, which the discrete-time version of the classical Fourier Transform of continuous-time signals. Using the DTFT machinery, we then discuss briefly the main problems related to the process of sampling a continuous-time signal, namely frequency aliasing. This discussion leads us to the sampling theorem.

1.4.1 The discrete-time Fourier transform

1.4.1.1 Definition

Recall that for a continuous-time signal $x(t)$ the Fourier Transform is defined as:

$$\mathcal{F}\{x\}(\omega) = X(\omega) = \int_{-\infty}^{+\infty} x(t) e^{-j2\pi f t} dt = \int_{-\infty}^{+\infty} x(t) e^{-j\omega t} dt \tag{1.52}$$

where the variable f is *frequency* and is expressed in Hz, while the *angular frequency* ω has been defined as $\omega = 2\pi f$ and expressed in radians/s. Note that we are following the conventional notation by which time-domain signals are denoted using lowercase symbols (e.g., $x(n)$) while frequency-domain signals are denoted in uppercase (e.g., $X(\omega)$).

We can try to find an equivalent expression in the case of a discrete-time signal $x[n]$. If we think of $x[n]$ as the sampled version of a continuous-time signal $x(t)$ with a sampling interval $T_s = 1/F_s$,

i.e. $x[n] = x(nT_s)$, we can define the *discrete-time Fourier transform (DTFT)* starting from Eq. (1.52) where the integral is substituted by a summation:

$$\mathcal{F}\{x\}(\omega_d) = X(\omega_d) = \sum_{n=-\infty}^{+\infty} x(nT_s)e^{-j2\pi f \frac{n}{F_s}} = \sum_{n=-\infty}^{+\infty} x[n]e^{-j\omega_d n}. \tag{1.53}$$

There are two remarks to be made about this equation. First, we have omitted the scaling factor T_s in front of the summation, which would be needed to have a perfect correspondence with Eq. (1.52) but is irrelevant to our tractation. Second, we have defined a new variable $\omega_d = 2\pi f/F_s$: we call this the *normalized* (or *digital*) angular frequency. This is not to be confused with the angular frequency ω used in Eq. (1.52): ω_d is measured in radians/sample, and varies in the range $[-2\pi, 2\pi]$ when f varies in the range $[-F_s, F_s]$. In this book we use the notation ω to indicate the angular frequency in radians/s, and ω_d to indicate the normalized angular frequency in radians/sample.

As one can verify from Eq. (1.53), $X(\omega_d)$ is a periodic function in ω_d with a period 2π. Note that this periodicity of 2π in ω_d corresponds to a periodicity of F_s in the domain of the absolute-frequency f. Moreover $X(\omega_d)$ is in general a complex function, and can thus be written in terms of its real and imaginary parts, or alternatively in polar form as

$$X(\omega_d) = |\, X(\omega_d) \,|\, e^{\arg[X(\omega_d)]}, \tag{1.54}$$

where $|\, X(\omega_d) \,|$ is the *magnitude function* and $\arg[X(\omega_d)]$ is the *phase function*. Both are real-valued functions. Given the 2π periodicity of $X(\omega_d)$ we will arbitrarily assume that $-\pi < \arg[X(\omega_d)] < \pi$. We informally refer to $|\, X(\omega_d) \,|$ also as the *spectrum* of $x[n]$.

The *inverse discrete-time Fourier transform (IDTFT)* is found by observing that Eq. (1.53) represents the Fourier series of the periodic function $X(\omega_d)$. As a consequence, one can apply Fourier theory for periodic functions of continuous variables, and compute the Fourier coefficients $x[n]$ as

$$\mathcal{F}^{-1}\{X\}[n] = x[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(\omega_d)e^{j\omega_d n}d\omega_d. \tag{1.55}$$

Equations (1.53) and (1.55) together form a Fourier representation for the sequence $x[n]$. Equation (1.55) can be regarded as a *synthesis* formula, since it represents $x[n]$ as a superposition of infinitesimally small complex sinusoids, with $X(\omega_d)$ determining the relative amount of each sinusoidal component. Equation (1.53) can be regarded as an *analysis* formula, since it provides an expression for computing $X(\omega_d)$ from the sequence $x[n]$ and determining its sinusoidal components.

1.4.1.2 DTFT of common sequences

We can apply the DTFT definition to some of the sequences that we have examined. The DTFT of the unit impulse $\delta[n]$ is the constant 1:

$$\mathcal{F}\{\delta\}(\omega_d) = \sum_{n=-\infty}^{+\infty} \delta[n]e^{-j\omega_d n} = 1. \tag{1.56}$$

The unit step sequence $u[n]$ does not have a DTFT, because the sum in Eq. (1.53) takes infinite values. The exponential sequence (1.9) also does not admit a DTFT. However if we consider the *right sided exponential sequence* $x[n] = a^n u[n]$, in which the unit step is multiplied by an exponential with $|\, a \,| < 1$, then this admits a DTFT:

$$\mathcal{F}\{x\}(\omega_d) = \sum_{n=-\infty}^{+\infty} a^n u[n]e^{-j\omega_d n} = \sum_{n=0}^{+\infty} \left(ae^{-j\omega_d}\right)^n = \frac{1}{1-ae^{-j\omega_d}}. \tag{1.57}$$

Property	**Time-domain sequences**	**Frequency-domain DTFTs**
	$x[n],\ y[n]$	$X(\omega_d),\ Y(\omega_d)$
Linearity	$ax[n] + by[n]$	$aX(\omega_d) + bY(\omega_d)$
Time-shifting	$x[n - n_0]$	$e^{-j\omega_d n_0} X(\omega_d)$
Frequency-shifting	$e^{j\omega_0 n} x[n]$	$X(\omega_d - \omega_0)$
Frequency differentation	$nx[n]$	$j\frac{dX}{d\omega_d}(\omega_d)$
Convolution	$(x * y)[n]$	$X(\omega_d) \cdot Y(\omega_d)$
Multiplication	$x[n] \cdot y[n]$	$\frac{1}{2\pi}\int_{-\pi}^{\pi} X(\theta)Y(\omega_d - \theta)d\theta$
Parseval relation	$\sum_{n=-\infty}^{+\infty} x[n]y^*[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(\omega_d)Y^*(\omega_d)d\omega_d$	

Table 1.1: *General properties of the discrete-time Fourier transform.*

The complex exponential sequence $x[n] = e^{j\omega_0 n}$ or the real sinusoidal sequence $x[n] = \cos(\omega_0 n + \phi)$ are other examples of sequences that do not have a DTFT, because the sum in Eq. (1.53) takes infinite values. In general a sequences does not necessarily admit a Fourier representation, meaning with this that the series in Eq. (1.53) may not converge. One can show that $x[n]$ being absolutely summable (we have defined absolute summability in Eq. (1.32)) is a sufficient condition for the convergence of the series (recall the definition of absolute summability given in Eq. (1.32)). Note that an absolutely summable sequence has always finite energy, and that the opposite is not always true, since $\sum |\, x[n]\,|^2 \leq (\sum |\, x[n]\,|)^2$. Therefore a finite-energy sequence does not necessarily admit a Fourier representation.[2]

1.4.1.3 Properties

Table 1.1 lists a number of properties of the DTFT which are useful in digital signal processing applications. Time- and frequency-shifting are interesting properties in that they show that a shifting operation in either domain correspond to multiplication for an complex exponential function in the other domain. Proof of these properties is straightforward from the definition of DTFT.

The convolution property is extremely important: it says that a convolution in the time domain becomes a simple multiplication in the frequency domain. This can be demonstrated as follows:

$$\begin{aligned}\mathcal{F}\{x * y\}(\omega_d) &= \sum_{n=-\infty}^{+\infty}\left(\sum_{k=-\infty}^{+\infty} x[k]y[n-k]\right)e^{-j\omega_d n} = \sum_{m=-\infty}^{+\infty}\sum_{k=-\infty}^{+\infty} x[k]y[m]e^{-j\omega_d(k+m)} \\ &= \sum_{k=-\infty}^{+\infty} x[k]e^{-j\omega_d k} \cdot \sum_{m=-\infty}^{+\infty} y[m]e^{-j\omega_d m},\end{aligned} \tag{1.58}$$

where in the second equality we have substituted $m = n - k$. The multiplication property is dual to the convolution property: a multiplication in the time-domain becomes a convolution in the frequency domain.

The Parseval relation is also very useful: if we think of the sum on the left-hand side as an inner product between the sequences x and y, we can restate this property by saying that the DTFT preserves

[2]For non-absolutely summable sequences like the unit step or the sinusoidal sequence, the DTFT can still be defined if we resort to the Dirac delta $\delta_D(\omega_d - \omega_0)$. Since this is not a function but rather a distribution, extending the DTFT formalism to non-summable sequences requires to dive into the theory of distributions, which we are not willing to do.

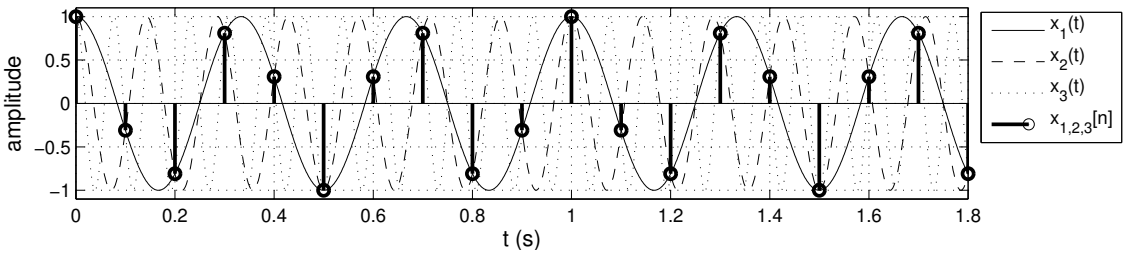


Figure 1.7: *Example of frequency aliasing occurring for three sinusoids.*

the inner product (apart from the scaling factor $1/2\pi$). In particular, when $x = y$, it preserves the energy of the signal x. The Parseval relation can be demonstrated by noting that the DTFT of the sequence $y^*[-n]$ is $Y^*(\omega_d)$. Then we can write:

$$\mathcal{F}^{-1}\{XY^*\}[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(\omega_d)Y^*(\omega_d)e^{j\omega_d n}d\omega_d = \sum_{k=-\infty}^{+\infty} x[k]y^*[k-n], \tag{1.59}$$

where in the first equality we have simply used the definition of the IDTFT, while in the second equality we have exploited the convolution property. Evaluating this expression for $n = 0$ proves the Parseval relation.

1.4.2 The sampling problem

1.4.2.1 Frequency aliasing

With the aid of the DTFT machinery, we can now go back to the concept of "sampling" and introduce some fundamental notions. Let us start with an example.

Consider three continuous-time sinusoids $x_i(t)$ $(i = 1, 2, 3)$ defined as

$$x_1(t) = \cos(6\pi t), \quad x_2(t) = \cos(14\pi t), \quad x_3(t) = \cos(26\pi t). \tag{1.60}$$

These sinusoids have frequencies 3, 7, and 13 Hz, respectively. Now we construct three sequences $x_i[n] = x_i(n/F_s)$ $(i = 1, 2, 3)$, each obtained by sampling one of the above signals, with a sampling frequency $F_s = 10$ Hz. We obtain the sequences

$$x_1[n] = \cos(0.6\pi n), \quad x_2[n] = \cos(1.4\pi n), \quad x_3[n] = \cos(2.6\pi n). \tag{1.61}$$

Figure 1.7 shows the plots of both the continuous-time sinusoids and the sampled sequences: note that all sequences have exactly the same sample values for all n, i.e. they actually *are* the same sequence. This phenomenon of a higher frequency sinusoid acquiring the identity of a lower frequency sinusoid after being sampled is called *frequency aliasing*.

In fact we can understand the aliasing phenomenon in a more general way using the Fourier theory. Consider a continuous-time signal $x(t)$ and its sampled version $x_d[n] = x(nT_s)$. The we can prove that the Fourier Transform $X(\omega)$ of $x(t)$ and the DTFT $X_d(\omega_d)$ of $x_d[n]$ are related via the following equation:

$$X_d(\omega_d) = F_s \sum_{m=-\infty}^{+\infty} X(\omega_d F_s + 2m\pi F_s). \tag{1.62}$$

This equation tells a fundamental result: $X_d(\omega_d)$ is a periodization of $X(\omega)$, i.e. it is a periodic function (of period 2π) made of a sum of shifted and scaled replicas of $X(\omega)$. The terms of this sum for $m \neq 0$ are *aliasing terms* and are said to alias into the so-called *base band* $[-\pi F_s, \pi F_s]$. Therefore if two continuous-time signals $x_1(t),\ x_2(t)$ have Fourier transforms with the property $X_2(\omega) = X_1(\omega + 2m\pi F_s)$ for some $m \in \mathbb{Z}$, sampling these signals will produces identical DTFTs and therefore indentical sequences. This is the case of the sinusoids in Eq. (1.61).

In the remainder of this section we provide a proof of Eq. (1.62). We first write $x(t)$ and $x_d[n]$ in terms of their Fourier transforms:

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} X(\omega)e^{j\omega t}d\omega, \qquad x_d[n] = \frac{1}{2\pi}\int_{-\pi}^{+\pi} X_d(\omega_d)e^{j\omega_d n}d\omega_d. \tag{1.63}$$

The first integral can be broken up into an infinite sum of integrals computed on the disjoint intervals $[(2m-1)\pi F_s, (2m+1)\pi F_s]$, (with $m \in \mathbb{Z}$) each of length $2\pi F_s$. Then

$$\begin{aligned} x(t) &= \frac{1}{2\pi}\sum_{m=-\infty}^{+\infty}\int_{(2m-1)\pi F_s}^{(2m+1)\pi F_s} X(\omega)e^{j\omega t}d\omega \\ &= \frac{1}{2\pi}\int_{-\pi F_s}^{\pi F_s} e^{j\theta t}\sum_{m=-\infty}^{+\infty} X(\theta + 2m\pi F_s)e^{j2m\pi F_s t}d\theta, \end{aligned} \tag{1.64}$$

where in the second equality we have substituted $\omega = \theta + 2m\pi F_s$ in the integral, and we have swapped the integral and the series. If we sample this representation to obtain $x_d[n] = x(nT_s)$, we can write

$$\begin{aligned} x_d[n] = x(nT_s) &= \frac{1}{2\pi}\int_{-\pi F_s}^{\pi F_s} e^{j\theta nT_s}\sum_{m=-\infty}^{+\infty} X(\theta + 2m\pi F_s)e^{j2m\pi F_s nT_s}d\theta \\ &= \frac{1}{2\pi}\int_{-\pi F_s}^{\pi F_s} e^{j\theta nT_s}\left(\sum_{m=-\infty}^{+\infty} X(\theta + 2m\pi F_s)\right)d\theta, \end{aligned} \tag{1.65}$$

because the exponentials inside the sum are all equal to 1 ($e^{j2m\pi F_s nT_s} = e^{j2nm\pi} = 1$). If we finally substitute $\omega_d = \theta T_s$ we obtain

$$x_d[n] = \frac{F_s}{2\pi}\int_{-\pi}^{+\pi} e^{j\omega_d n}\left(\sum_{m=-\infty}^{+\infty} X(\omega_d F_s + 2m\pi F_s)\right)d\omega_d, \tag{1.66}$$

which proves Eq. (1.62).

1.4.2.2 The sampling theorem and the Nyquist frequency

Consider the three cases depicted in Fig. 1.8. The magnitude of the Fourier transform in Fig. 1.8(a) (upper panel) is zero everywhere outside the base band, and Eq. (1.62) tells us that the magnitude of the sampled signal looks like the plot in the lower panel. In Fig. 1.8(b) (upper panel) we have a similar situation except that the magnitude is non-zero in the band $[\pi F_s, 3\pi F_s]$. The magnitude of the corresponding sampled signal then looks like the plot in the lower panel, and is identical to the one in Fig. 1.8(a). Yet another situation is depicted in Fig. 1.8(c) (upper panel): in this case we are using a smaller sampling frequency F_s, so that the magnitude now extends to more than one band. As a result the shifted replicas of $|X|$ overlap and $|X_d|$ is consequently distorted.

These examples suggest that a "correct" sampling of a continuos signal $x(t)$ corresponds to the situation of Fig. 1.8(a), while for the cases depicted in Figs. 1.8(b) and 1.8(c) we loose information about

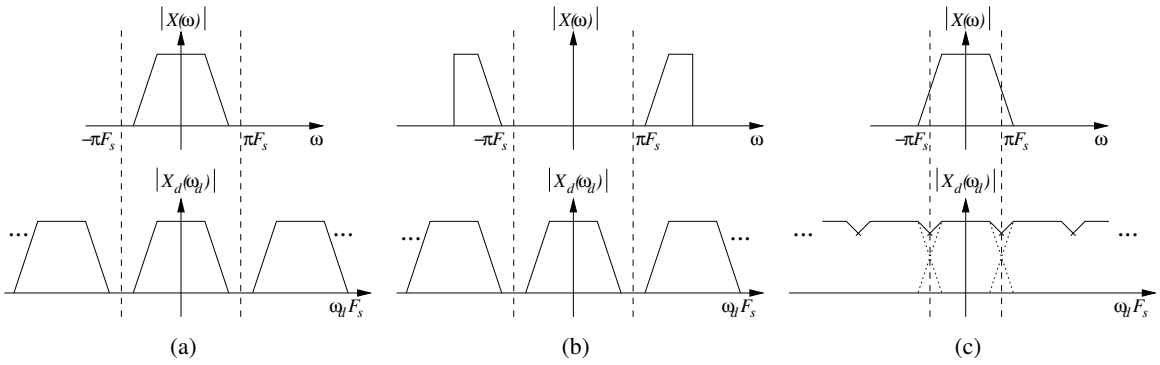


Figure 1.8: *Examples of sampling a continuous time signal: (a) spectrum limited to the base band; (b) the same spectrum shifted by 2π; (c) spectrum larger than the base band.*

the original signal. The *sampling theorem* formalizes this intuition by saying that $x(t)$ can be exactly reconstructed from its samples $x[n] = x(nT_s)$ if and only if $X(\omega) = 0$ outside the base band (i.e. for all $|\,\omega\,| \geq \pi/F_s$). The frequency $f_{\text{Ny}} = F_s/2$ Hz, corresponding to the upper limit of the base band, is called *Nyquist frequency*.

Based on what we have just said, when we sample a continuous-time signal we must choose F_s in such a way that the Nyquist frequency is above any frequency of interest, otherwise frequencies above f_{Ny} will be aliased. In the case of audio signals, we know from psychoacoustics that humans perceive audio frequencies up to $\sim$ 20 kHz: therefore in order to guarantee that no artifacts are introduced by the sampling procedure we must use $F_s > 40$ kHz, and in fact the most diffused standard is $F_s = 44.1$ kHz. In some specific cases we may use lower sampling frequencies: as an example it is known that the spectrum of a speech signal is limited to ~ 4 kHz, and accordingly the most diffused standard in telephony is $F_s = 8$ kHz.

In the remainder of this section we sketch the proof of the sampling theorem. If $X(\omega)) \neq 0$ only in the base band, then all the sum terms in Eq. (1.62) are 0 except for the one with $m = 0$. Therefore

$$X_d(\omega_d) = F_s X(\omega_d F_s) \qquad \text{for} \quad \omega_d \in (-\pi, \pi). \tag{1.67}$$

In order to reconstruct $x(t)$ we can take the inverse Fourier Transform:

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} X(\omega)e^{j\omega t}d\omega = \frac{1}{2\pi}\int_{-\pi F_s}^{\pi F_s} X(\omega)e^{j\omega t}d\omega = \frac{1}{2\pi F_s}\int_{\pi F_s}^{+\pi F_s} X_d\left(\frac{\omega}{F_s}\right)e^{j\omega t}d\omega. \tag{1.68}$$

where in the second equality we have exploited the hypothesis $X \equiv 0$ outside the base band and in the third one we have used Eq. (1.67). If we now reapply the definition of the DTFT we obtain

$$x(t) = \frac{1}{2\pi F_s}\int_{-\pi F_s}^{\pi F_s}\left[\sum_{n=-\infty}^{+\infty} x(nT_s))e^{-j\omega T_s n}\right]e^{j\omega t}d\omega = \sum_{n=-\infty}^{+\infty}\frac{x(nT_s)}{2\pi F_s}\int_{-\pi F_s}^{\pi F_s} e^{j\omega(t-nT_s)}d\omega, \tag{1.69}$$

where in the second equality we have swapped the sum with the integral. Now look at the integral on the right hand side. We can solve it explicitly and write

$$\begin{aligned}\frac{1}{2\pi F_s}\int_{-\pi F_s}^{\pi F_s} e^{j\omega(t-nT_s)}d\omega &= \frac{1}{2\pi F_s}\frac{2}{2j(t-nT_s)}\left[e^{j\pi F_s(t-nT_s)} - e^{-j\pi F_s(t-nT_s)}\right] = \\ &= \frac{\sin[\pi F_s(t-nT_s)]}{\pi F_s(t-nT_s)} = \text{sinc}[F_s(t-nT_s)].\end{aligned} \tag{1.70}$$

That is, the integral is a *cardinal sine* function, defined as $\mathrm{sinc}(t) \triangleq \sin(\pi t)/\pi t$ (the use of π in the definition has the effect that the sinc function has zero crossings on the non-zero integers). In conclusion, we can rewrite Eq. (1.68) as

$$x(t) = \sum_{n=-\infty}^{+\infty} x(nT_s)\,\mathrm{sinc}\left(\frac{t}{T_s} - n\right). \tag{1.71}$$

We have just proved that if $X \equiv 0$ outside the base band then $x(t)$ can be reconstructed from its samples through Eq. (1.71). The opposite implication is obvious: if $x(t)$ can be reconstructed through its samples it must be true that $X \equiv 0$ outside the base band, since a sampled signal only supports frequencies up to f_{Ny} by virtue of Eq. (1.62).

1.5 Short-time Fourier analysis

In these section we introduce the most common spectral analysis tool: the *Short Time Fourier Transform (STFT)*. Sounds are time-varying signals, therefore, it is important to develop analysis techniques to inspect some of their time-varying features. The STFT allows joint analysis of the temporal and frequency features of the sound signal, in other words it allows to follow the temporal evolution of the spectral parameters of a sound. The main building block of the STFT is the *Discrete Fourier Transform (DFT)*, which can be thought as a specialization of the DTFT for sequences of finite length.

1.5.1 The Discrete Fourier Transform

The *Discrete Fourier Transform (DFT)* is a special case of the DTFT applied to finite-length sequences. As such it is a useful tool for representing periodic sequences. as we said in the previous section, a periodic sequence does not have a DTFT in a strict sense. However periodic sequences are in one-to-one correspondence with finite-length sequences, meaning with this that a finite-length sequence can be taken to represent a period of a periodic sequence.

1.5.1.1 Definitions and properties

The *Discrete Fourier Transform (DFT)* is a special case of the DTFT applied to finite-length sequences $x[n]$ with $0 \leq n \leq N-1$. Let us consider one such sequence: we define the DFT of $x[n]$ as the sequence $X[k]$ obtained by uniformly sampling the DTFT $X(\omega_d)$ on the ω_d-axis between $0 \leq \omega_d < 2\pi$, at points at $\omega_k = 2\pi k/N$, $0 \leq k \leq N-1$. If $x[n]$ has been sampled from a continuous-time signal, i.e. $x[n] = x(nT_s)$, the points ω_k correspond to the frequency points $f_k = kF_s/N$ (in Hz).

From Eq. (1.53) one can then write

$$X[k] = X(\omega_d)|_{\omega_d = 2\pi k/N} = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}kn} = \sum_{n=0}^{N-1} x[n] W_N^{kn}, \quad 0 \leq k \leq N-1 \tag{1.72}$$

where we have used the notation $W_N = e^{-j2\pi/N}$. Note that the DFT is also a finite-length sequence in the frequency domain, with length N. The *inverse discrete Fourier Transform (IDFT)* is given by

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] W_N^{-kn}, \quad 0 \leq n \leq N-1. \tag{1.73}$$

This relation can be verified by multiplying both sides by W_N^{ln}, with l integer, and summing the result from $n = 0$ to $n = N - 1$:

$$\sum_{n=0}^{N-1} x[n]W_N^{ln} = \frac{1}{N}\sum_{n=0}^{N-1}\sum_{k=0}^{N-1} X[k]W_N^{-(k-l)n} = \frac{1}{N}\sum_{k=0}^{N-1} X[k]\left[\sum_{n=0}^{N-1} W_N^{-(k-l)n}\right], \tag{1.74}$$

where the last equality has been obtained by interchanging the order of summation. Now, the summation $\sum_{n=0}^{N-1} W_N^{-(k-l)n}$ has the interesting property that it takes the value N when $k - l = rN$ ($r \in \mathbb{Z}$), and takes the value 0 for any other value of k and l. Therefore Eq. (1.74) reduces to the definition of the DFT, and therefore Eq. (1.73) is verified.

We have just proved that, for a length-N sequence $x[n]$, the N values of its DTFT $X(\omega_d)$ at points $\omega_d = \omega_k$ are sufficient to determine $x[n]$, and hence $X(\omega_d)$, uniquely. This justifies our definition of Discrete Fourier Transform of finite-length sequences given in Eq. (1.72). The DFT is at the heart of digital signal processing, because it is a *computable* transformation.

Most of the DTFT properties listed in Table 1.1 have a direct translation for the DFT. Clearly the DFT is linear. The time- and frequency-shifting properties still correspond to a multiplication by a complex number, however these properties becomes periodic with period N. As an example, the time shifting properties for the DFT becomes

$$x_m[n] = x[n - m] \quad \Rightarrow \quad X_m[k] = W_N^{km} X[k]. \tag{1.75}$$

Clearly any shift of $m + lN$ samples cannot be distinguished from a shift by m samples, since $W_N^{km} = W_N^{k(m+lN)}$. In other words, the ambiguity of the shift in the time domain has a direct counterpart in the frequency domain.

The convolution property also holds for the DFT and is stated as follows:

$$z[n] = (x * y)[n] \triangleq \sum_{m=0}^{N-1} x[n]y[n - m] \quad \Rightarrow \quad Z[k] = (X \cdot Y)[k], \tag{1.76}$$

where in this case the symbol $*$ indicates the *periodic convolution*. The proof of this property is similar to the one given for the DTFT.

1.5.1.2 Resolution, leakage and zero-padding

Consider the complex exponential sequence $x[n] = e^{j\omega_0 n}$ over a finite number of points $0 \leq n < N$. In Sec. 1.2 we have already shown that this sequence is periodic over the interval $[0, N]$ only if $\omega_0 N = 2\pi k$ for some integer k. This implies that there are exactly N periodic complex exponential sequences representable with N samples, i.e. those for which $\omega_0 = 2\pi k_0/N$, with $k_0 = 0 \ldots N - 1$: these are the sequences $x[n] = e^{j2\pi k_0 n/N} = W_N^{-k_0 n}$. From the definitions of the DFT and the IDFT it follows immediately that $X[k] = \delta(k - k_0)$, i.e. the DFT associated to the sequence $W_N^{-k_0 n}$ takes the value 1 for $k = k_0$ and is zero everywhere else. As an example, Fig. 1.9(a) shows a 64-points DFT (computed numerically) for the sequence $x[n] = W_N^{-k_0 n}$for $k_0 = 20$.

Since the *resolution* of the DFT is limited by the number of DFT points, one may wonder how the DFT looks like for complex exponential sequences that are *not* periodic over the interval $[0, N]$. An example is shown in Fig. 1.9(b) for the complex exponential sequence with $\omega_0 = 2\pi k_0/N$, where we have used a non-integer value $k_0 = 20.5$ and $N = 64$. Although the value of ω_0 is very similar to the one in Fig. 1.9(a), the DFT looks very different. This happens because we have chosen a value for ω_0 that falls in the crack between two DFT points, and consequently the DFT does a poor job in resolving

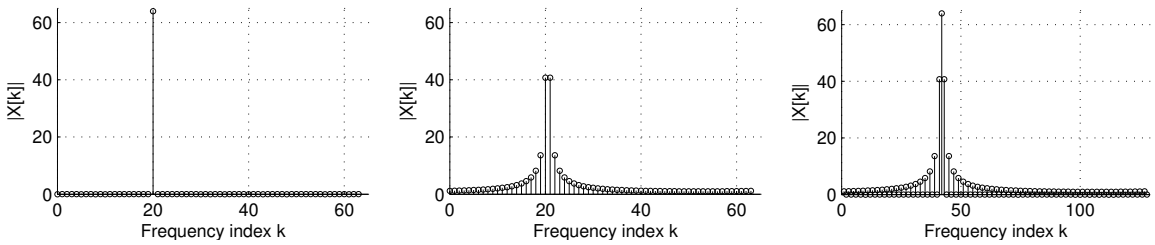


Figure 1.9: *Examples of DFT applied to complex exponential sequences: (a) $N = 64, k_0 = 20$, the sequence is periodic and the DFT is a delta-sequence in the frequency domain; (b)$N = 64, k_0 = 20.5$, the sequence is not periodic and the DFT exhibits leakage; (c) $N = 128, k_0 = 41$, the sequence is the windowed exponential of Eq. (1.77) and the DFT is a shifted version of the rectangular window DFT.*

the frequency of this particular signal. We call this effect *leakage*: since ω_0 does not line up with one of the "allowed" frequencies, the energy of the DFT leaks all over the base band.

In order to understand the leakage effect we now look at a third example. Figure 1.9(c) depicts the DFT of the sequence

$$x[n] = \begin{cases} e^{j2\pi k_0 n/N}, & 0 \le n < \frac{N}{2}, \\ 0, & \frac{N}{2} \le n < N, \end{cases} \tag{1.77}$$

with $k_0 = 41$ and $N = 128$. In other words, the sequence $x[n]$ is constructed by taking the complex exponential sequence $e^{j2\pi k_0 n/N}$ over 64 points, and by *zero-padding* this sequence over the remaining 64 points. Note that the complex exponential sequence of this example is clearly the same that we have considered in Fig. 1.9(b) (we have simply doubled the values of k_0 and N, so that ω_0 has the same value), and is clearly periodic over $N = 128$ points. Note also that the DFTs of Fig. 1.9(b) and 1.9(c) are identical, except that the one in Fig. 1.9(c) has twice the points and consequently a better resolution in frequency.

The sequence $x[n]$ of Eq. (1.77) can be also written as

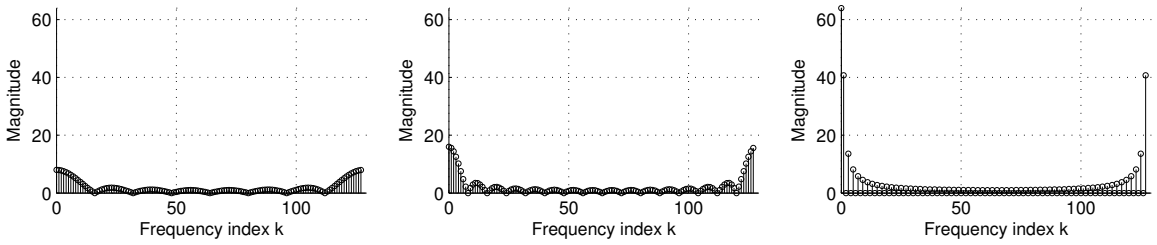
$$x[n] = e^{j2\pi k_0 n/N} \cdot w_{N/2}[n] = W_N^{-k_0 n} w_{N/2}[n], \qquad \text{where} \quad w_{N/2}[n] = \begin{cases} 1, & 0 \le n < \frac{N}{2}, \\ 0, & \frac{N}{2} \le n < N, \end{cases} \tag{1.78}$$

where $w_{N/2}[n]$ is a *rectangular window* of length $N/2$. More in general, we call $w_M[n]$ a rectangular window of length M.

The advantage of this representation is that we can now compute explicitely the DFT $X[k]$, since we know that multiplying by $W_N^{-k_0 n}$ in time correspoinds to shifting by k_0 samples in frequency. Therefore $X[k]$ equals the DFT of $w_{N/2}[h]$, shifted by k_0 samples. The DFT of the generic rectangular window $w_M[n]$ can be computed from the definition as

$$\mathcal{F}\{w_M\}[k] = \sum_{n=0}^{N-1} w_M[n] W_N^{kn} = \sum_{n=0}^{M-1} W_N^{kn} = \frac{1 - W_N^{kM}}{1 - W_N^k} = e^{-j\frac{\pi k}{N}(M-1)} \frac{\sin(\pi \frac{kM}{N})}{\sin(\pi \frac{k}{N})}, \tag{1.79}$$

where in the third equality we have used the property of the geometric series $\sum_{k=0}^{M-1} q^k = (1-q^M)/(1-q)$, and in the fourth equality we have applied some straigthforward trigonometry. Figure 1.10 shows three plots of this DFT, for different window lengths M. In particular Fig. 1.10(c) shows the case



(a) (b) (c)

Figure 1.10: *Examples of DFT ($N = 128$) applied to a rectangular window: (a) $M = N/16$, (b) $M = N/8$, (a) $M = N/2$.*

$M = N/2$: note that this plot coincides with the one in Fig. 1.9(c), except for a shift in frequency, which is what we expected.

To summarize, in this section we have shown that (a) the frequency resolution of the DFT is limited by the number N of DFT points, (b) the DFT of a complex exponential sequence which is not periodic over N points exhibits poor resolution and leakage, and (c) the leakage effect can be interpreted as the effect of a rectangular window of length N applied to the sequence.

M-1.9

Compute the DFT of complex exponential sequence $x[n] = e^{j2\pi k_0 n/N}$ for integer and non-integer values of k_0, in order to explore leakage effects. Then ompute the DFT of zero-padded complex exponential sequences, in order to see how frequency resolution and leakage are affected.

1.5.1.3 Fast computation of the DFT: the FFT algorithm

A brute-force approach to DFT computation has $\mathcal{O}(N^2)$ running time, since we need to perform $\mathcal{O}(N)$ multiplications and additions to compute each of the N DFT samples: we need more efficient solutions to the problem. The *Fast Fourier Transform (FFT)* algorithm provides the most efficient computation of the DFT, as it runs in $\mathcal{O}(N \log N)$ times. The algorithm is based on a divide-and-conquer strategy, that allows to compute the DFT of $x[n]$ using the DFTs of two half-length subsequences of $x[n]$. Additionally it exploits some nice properties of the N-th roots of unity W_N^k.

Assume for simplicity that the sequence $x[n]$ has length $N = 2^s$ for some $s \in \mathbb{N}$. Then we can write the DFT sequence $X[k]$ as

$$\begin{aligned} X[k] &= \sum_{n=0}^{\frac{N}{2}-1} x[2n] W_N^{2kn} + \sum_{n=0}^{\frac{N}{2}-1} x[2n+1] W_N^{k(2n+1)} \\ &= \sum_{n=0}^{\frac{N}{2}-1} x[2n] W_N^{2kn} + W_N^k \sum_{n=0}^{\frac{N}{2}-1} x[2n+1] W_N^{2kn} \\ &= \sum_{n=0}^{\frac{N}{2}-1} x[2n] W_{N/2}^{kn} + W_N^k \sum_{n=0}^{\frac{N}{2}-1} x[2n+1] W_{N/2}^{kn}, \end{aligned} \tag{1.80}$$

where in the first equality we have separated the terms involving even elements and odd elements, in the second one we have factorized a term W_N^k, and in the third one we have exploited the property

$W_{aN}^{ak} = W_N^k$ for any $a \in \mathbb{N}$. If we now look at the two sums in the last row, we see that they are the DFTs of the sequences $x_0[n] = \{x[0], x[2], \ldots x[N-2]\}$ and $x_1[n] = \{x[1], x[3], \ldots x[N-1]\}$, respectively. Both $x_0[n]$ and $x_1[n]$ have length $N/2$. Therefore the problem of computing $X[k]$ reduces to the problem of computing two half-length DFTs $X_0[k]$, $X_i[k]$, and then summing their values according to Eq. (1.80). The resulting computational procedure is detailed in Algorithm 1.1. It is quite obvious that the running time $T(N)$ of this algorithm is given by the recurrence equation $T(N) = 2T(N/2) + \mathcal{O}(N)$, therefore $T(N) = \mathcal{O}(N \log N)$.

Algorithm 1.1: RECURSIVE-FFT(x)

```
1  N ← length(x) // N is a power of 2
2
3  if n = 1 then return x
4  W_N ← e^{-2πj/N}
5  W ← 1
6  x_0[n] ← {x[0], x[2], ... x[N-2]}
7  x_1[n] ← {x[1], x[3], ... x[N-1]}
8  X_0[k] ←RECURSIVE-FFT(x_0)
9  X_1[k] ←RECURSIVE-FFT(x_1)
10 for k ← 0 to N/2 - 1 do
11     X[k] ← X_0[k] + W X_1[k]
12     X[k + N/2] ← X_0[k] - W X_1[k]
13     W ← W · W_N
14 return X
```

M-1.10

Realize Algorithm 1.1 and assess its functioning by comparing it with the `fft` function.

1.5.1.4 Iterative FFT algorithms and parallel realizations

Note that in writing the last cycle of Algorithm 1.1 we have exploited a relevant property of the W_N^k coefficients, namely $W_N^{(k+N/2)} = -W_N^k$. Thanks to this property the value $WX_1[k]$ is used twice (it is a *common subexpression*): first it is added to $X_0[k]$ to compute $X[k]$, then it is subtracted from $X_0[k]$ to compute $X[k + N/2]$. This is known as *butterfly operation*, and is the key element in the construction of a more efficient, iterative implementation of the FFT algorithm.

Figure 1.11(a) depicts the tree of calls in an invocation of the recursive algorithm, in the case $N = 8$. Looking at the tree we observe that, if we could arrange the elements in the order in which they appear in the leaves, we could compute the DFT as follows: first we take the elements in pairs and combine each pair with one butterfly operation, thus obtaining four 2-element DFTs; second, we take these DFTs in pairs and combine each pair with two butterfly operations, thus obtaining two 4-element DFTs; finally, we combine these two DFTs with four butterfly operations, thus obtaining the final 8-element DFT. The resulting scheme is an iterative FFT implementation.

The only problem left is how to arrange the elements in the order in which they appear in the leaves. Luckily the solution is straightforward: this order is a *bit-reversal permutation*, that is $x[n]$ is placed in the position obtained by reversing the bits of the binary representation of n. We can then write Algorithm 1.2. Clearly the algorithm is still $\mathcal{O}(N \log N)$, since the total number of butterfly operations is $\mathcal{O}(N \log N)$, and since the bit-reversal permutation is also a $\mathcal{O}(N \log N)$ procedure (we have to reverse N integers,

each made of $\log N$ bits). Figure 1.11(b) shows an efficient parallel implementation of this algorithm: it is made of $\log N$ stages, each performing $N/2$ butterfly operations.

Algorithm 1.2: ITERATIVE-FFT(x)

```
1  BIT-REVERSE-COPY(x, X)
2  N ← length(x)
3  for s ← 1 to log_2(N) do
4      m ← 2^s
5      W_m ← e^{-2πj/m}
6      for k ← 0 to N − 1 by m do
7          W ← 1
8          for l ← 0 to m/2 − 1 do
9              t ← W X[k + l + m/2]
10             u ← X[k + l]
11             X[k + l] ← u + t
12             X[k + l + m/2] ← u − t
13             W ← W · W_m
14 return X
```

M-1.11

Realize Algorithm 1.2 and assess its functioning by comparing it with the `fft` function.

1.5.2 The Short-Time Fourier Transform

1.5.2.1 Definition and examples

$$X_n(\omega_d) = \sum_{m=-\infty}^{+\infty} w[n-m]x[m]e^{-j\omega_d m} \tag{1.81}$$

If $w \equiv 1$, then this equation reduces to the DTFT of $x[n]$. However in practical applications we are interested in using finite-length windows, in order to analyze the spectral properties of $x[n]$ over a finite time interval. In such applications what we really do is computing, for each n, the DFT of the finite-length sequence $w[n-m]x[m]$, and what we obtain is a finite-length sequence $X_n[k] = X_n(\omega_d)|_{\omega_d=2\pi k/N}$.

$X_n(\omega_d)$ is the DTFT of the sequence $x_n[m] = w[n-m]x[m]$, therefore

$$w[n-m]x[n] = \frac{1}{2\pi}\int_{-\pi}^{+\pi} X_n(\omega_d)e^{j\omega_d m}d\omega_d, \tag{1.82}$$

from which, when $n = m$ and in the hypothesis of $x[0] \neq 0$

$$x[n] = \frac{1}{2\pi w[0]}\int_{-\pi}^{+\pi} X_n(\omega_d)e^{j\omega_d m}d\omega_d. \tag{1.83}$$

This equation shows that the sequence x can be reconstructed from its STFT.

The magnitude of STFT is called *spectrogram*. Since STFT is function of two variables (i.e., time n and frequency ω_d), the plot of the spectrogram lives in a 3-D space. A typical 2-D representation of

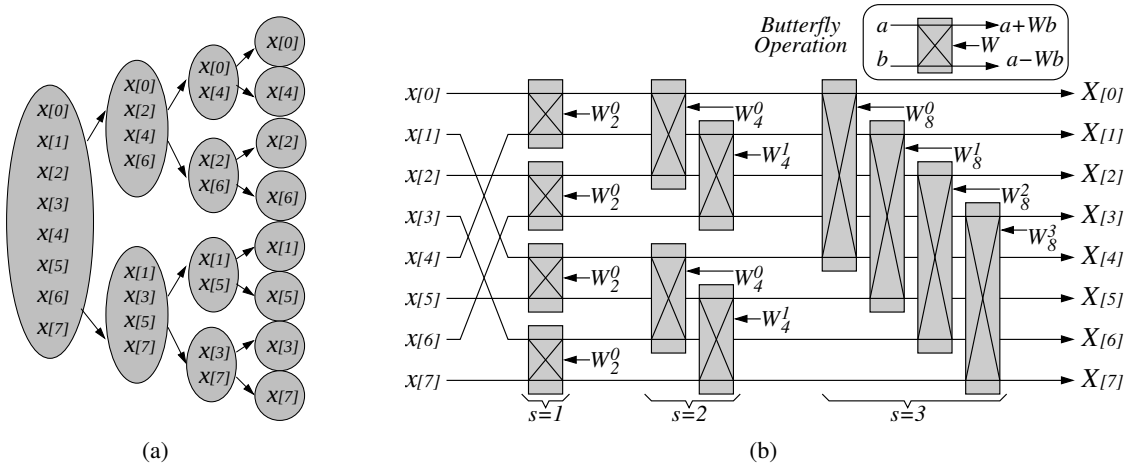


Figure 1.11: *(a) Tree of calls in an invocation of RECURSIVE-FFT: the leaves contain a bit-reversal permutation of $x[n]$; (b) parallel realization of ITERATIVE-FFT, where butterfly operations involve bit-reversed elements of $x[n]$.*

a spectrogram uses two axes for time and frequency, and magnitude values are represented with some color map (e.g. a greyscale map).

Figure 1.12 shows an example of short-time spectral analysis applied to a simple time-varying signal, the *chirp*. The chirp sequence is defined as

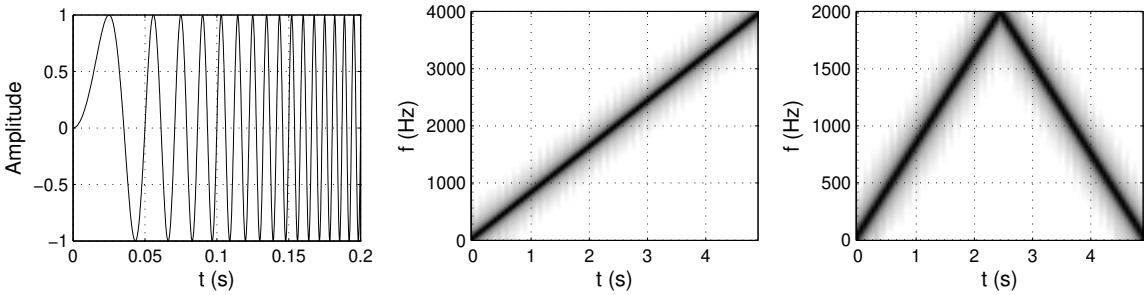
$$x[n] = A\cos\left[\frac{\omega_0}{2}(nT_s)^2\right]. \tag{1.84}$$

We have already seen in Sec. 1.3.1 that in general the instantaneous frequency of a signal $\cos[\phi(t)]$ is $d\phi/dt(t)$: therefore the instantaneuous frequency of this chirp signal is $\omega_0 n T_s$, i.e. it is not constant but increases linearly in time. A portion of a chirp signal with $\omega_0 = 2\pi \cdot 800$ rad/s^2 is shown in Fig. 1.12(a). Now we segment this signal into a set of subsequences with short finite length, e.g. using a rectangular window $w[n]$, as in Eq. (1.81). If the window is sufficiently short, we can reasonably assume that the frequency of the chirp is approximately constant in each subsequence. In fact the resulting spectrogram, shown in Fig. 1.12(b), shows that for a given time index n the STFT is essentially the DFT of a sinusoidal sequence: the STFT magnitude has large non-zero values around the instantaneous frequency, and much smaller non-zero values at other frequency points. Moreover the instantaneous frequency increases linearly and after 5 seconds it reaches the value 4000 Hz ($= 800 \cdot 5$ Hz), as expected.

So far in this example we have implicitely assumed that the sampling period T_s is sufficiently small, so that no aliasing occurs: in fact in drawing Fig. 1.12(b) we have used a sampling rate $F_s = 8$ kHz. However, aliasing will occur if we use smaller sampling rates. Figure 1.12(c) shows the spectrogram obtained using $F_s = 4$ kHz: in this case frequencies above $f_{\text{Ny}} = 2$ kHz are aliased, and this effect appears in the spectrogram when the black line starts moving down instead of increasing.

M-1.12

Synthesize a chirp signal and compute its STFT using different sampling rates, in order to verify the occurrence of aliasing.



(a) (b) (c)

Figure 1.12: *Short-time spectral analysis of a chirp signal: (a) initial portion of the chirp signal, with* $\omega_0 = 2\pi \cdot 800$ *rad/s*^{2}*; (b) spectrogram obtained with* $F_s = 8$ *kHz; (c) spectrogram obtained with* $F_s = 4$ *kHz.*

1.5.2.2 Windowing and the uncertainty principle

We have previously examined the resolution and leakage problems associated to the DFT: decreased frequency resolution and spectral energy leakage occur because the spectrum is convolved with that of a rectangular window. As the width of the rectangular window increases (see Fig. 1.10), the energy of its spectrum becomes more and more concentrated around the origin. In the limit, the spectrum of an infinite-width rectangular window is a $\delta[k]$ sequence, and no leakage occurs.

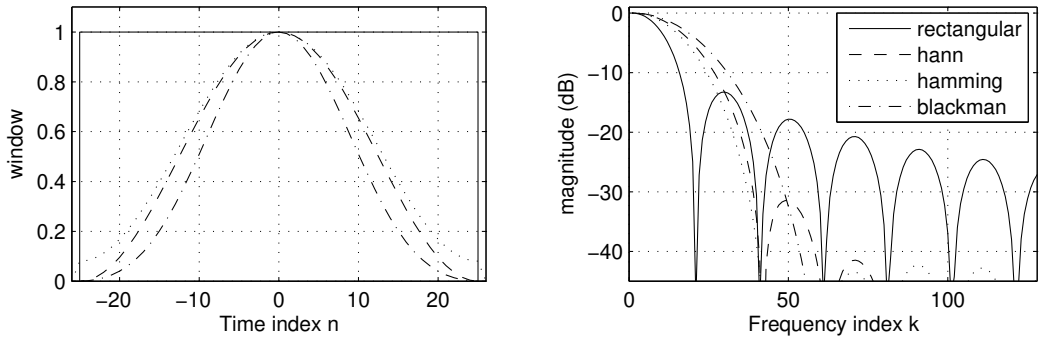
This qualitative analysis provides an example of the so-called *uncertainty principle*. Increasing resolution in frequency diminishes resolution in time, and viceversa. Although a certain trade-off between time resolution and frequency resolution is inevitable (and determined by the window length), one may wonder if such a trade-off can be improved by using windows with different shapes (and thus different spectra) from the rectangular window.

In fact the answer is yes. Some of the most commonly used window functions are listed below:

$$\begin{aligned} &\text{(Hann)} \quad w[n] = \frac{1}{2}\left[1 + \cos\left(\frac{2\pi n}{2M+1}\right)\right] \\ &\text{(Hamming)} \quad w[n] = 0.54 + 0.46\cos\left(\frac{2\pi n}{2M+1}\right) \\ &\text{(Blackman)} \quad w[n] = 0.42 + 0.5\cos\left(\frac{2\pi n}{2M+1}\right) + 0.08\cos\left(\frac{4\pi n}{2M+1}\right) \end{aligned} \tag{1.85}$$

Figure 1.13(a) depicts the plots of these windows in the time-domain, while a portion of the corresponding spectra is shown in Fig. 1.13(b). Note that these spectra share some common characteristics: all have large main "lobe" at 0 frequency, plus side lobes with decreasing amplitude. More precisely, two main spectral features have to be taken into account when analyzing the properties of these windows: first, the ability to resolve two nearby spectral components of a windowed signal depend mostly on the *main lobe width*, i.e. the nearest zero crossings on both sides of the main lobe; second, the amount of leakage from one frequency component to neighbouring bands depends on the amplitude of the side lobes, and primarily on *relative side lobe level*, i.e. the difference in dB between the amplitudes of the main lobe and the largest side lobe.

Figure 1.13(b) shows that the rectangular window has the smallest main lobe width, therefore it is better than other windows in resolving nearby sinusoids. On the other hand, it has the largest relative side lobe level, therefore it causes considerable leakage. Other windows have different performances in terms



(a) (b)

Figure 1.13: *Comparison of different windows; (a) time-domain window sequences, symmetrical with respect to $n = 0$; (b) window spectra in the vicinity of $k = 0$. Differences in terms of main lobe width and relative side lobe level can be appreciated.*

of main lobe width and relative side lobe level. The hamming window is often considered to provide the best trade-off for short-term spectral analysis of generic signals, however the choice of the window to use depends in general on many factors, including the characteristics of the signal to be analyzed.

M-1.13

Compute the DFT of complex exponential sequences $x[n] = e^{j2\pi k_0 n/N}$, windowed with rectangular, Hann, Hamming, and Blackman windows. Verify the performance of each window with respect to resolution and leakage effects, for integer and non-integer values of k_0.

1.6 Digital filters

1.6.1 The z-Transform

1.6.1.1 Definitions

The z-Transform is an operator that maps a sequence of $x[n]$ into a function $X : \mathbb{C} \to \mathbb{C}$. Definition:

$$\mathcal{Z}\{x\}(z) = X(z) = \sum_{n=-\infty}^{+\infty} x[n] z^{-n}, \tag{1.86}$$

with $z \in \mathbb{C}$ complex variable.

Close relationship with the DTFT: if $z = e^{j\omega_d}$, i.e. if we restrict the variable z to the *complex unit circle*, then the z-Transform reduces to the DTFT. In particular the point $z = 1$ corresponds to frequency $\omega_d = 0$, and $z = -1$ corresponds to $\omega_d = \pi$. Therefore evaluation of the z-Transform on the upper half of the complex unit circle gives the DTFT up to the (normalized angular) Nyquist frequency. In general we can write:

$$\mathcal{F}\{x\}(\omega_d) = \mathcal{Z}\{x\}(z)|_{z=e^{j\omega_d}}. \tag{1.87}$$

For this reason we sometimes write $X\left(e^{j\omega_d}\right)$ to indicate the DTFT of the sequence x.

The series in Eq. (1.86) does not converge for all values of z. For any sequence x, the set of values z for which the series converges is called *region of convergence (ROC)*. Since $|\, X(z)\,| \leq \sum_{n=-\infty}^{+\infty} |\, x[n]\,|\,|\, z\,|^{-n}$, if a point z_0 belongs to the ROC, then all points z that are on the complex circle with radius $|\, z_0\,|$ also

Property	**Sequences**	z**-Transforms**	**ROCs**
	$x[n],\ y[n]$	$X(z),\ Y(z)$	R_x, R_y
Linearity	$ax[n] + by[n]$	$aX(z) + bY(z)$	contains $R_x \cap R_y$
Time-shifting	$x[n - n_0]$	$z^{-n_0}X(z)$	R_x
z-scaling	$z_0^n x[n]$	$X(z/z_0)$	$\mid z_0 \mid R_x$
z- differentation	$nx[n]$	$-z\frac{dX}{dz}(z)$	R_x
Conjugation	$x^*[n]$	$X^*(z^*)$	R_x
Time-reversal	$x^*[-n]$	$X^*(1/z^*)$	$1/R_x$
Convolution	$(x * y)[n]$	$X(z) \cdot Y(z)$	contains $R_x \cap R_y$
Initial value theorem	If $x[n]$ causal (i.e. $x[n] = 0\ \forall n < 0$), then $\lim_{z\to\infty} X(z) = x[0]$.		

Table 1.2: *General properties of the z-Transform.*

belong to the ROC. Therefore the ROC is in general a *ring* in the complex plane. Such ring may or may not include the unit circle, in other words the z-Transform of $x[n]$ may exist in a certain region of the complex plane even if the DTFT of $x[n]$ does not exist.

Table 1.2 lists a set of relevant properties of the z-Transform, that are particularly useful in the study of discrete-time signals and digital filters. Most of them have direct counterparts in DTFT properties (see Table 1.1), and can be proved from the definition (1.86).

The inverse z-transform is formally defined as

$$x[n] = \frac{1}{2\pi j} \oint_{\mathcal{C}} X(z) z^{n-1} dz, \tag{1.88}$$

where the integral is evaluated over the contour $\mathcal{C}$, which can be any closed contour that belongs to the ROC of $X(z)$ and encircles $z = 0$. Without entering into details, we remark that for the kinds of sequences and z-transforms typically encountered in digital signal processing applications, less formal procedures are sufficient and preferable. We propose some examples in Sec. 1.6.2.

1.6.1.2 z-transforms of common sequences

We illustrate the z-transform with some notable examples.

The unit impulse $\delta[n]$ has z-transform 1 and the ROC is the entire complex plane. The ideal delay $\delta[n - n_0]$ has z-transform z^{-n_0} and the ROC is the entire complex plane.

Consider the *finite-length exponential sequence*

$$x[n] = \begin{cases} a^n, & 0 \le n < N \\ 0, & \text{elsewhere.} \end{cases} \tag{1.89}$$

The z-transform is

$$X(z) = \sum_{n=0}^{N-1} a^n z^{-n} = \sum_{n=0}^{N-1} \left(az^{-1}\right)^{-n} = \frac{1 - a^N z^{-N}}{1 - az^{-1}}. \tag{1.90}$$

In particular since the series has only a finite number of terms the ROC is the entire complex plane. We can generalize and state that any finite-length sequence admits a z-transform whose ROC is the entire complex plane.

Slightly more complex example: the right-sided exponential sequence $x[n] = a^n u[n]$ already examined in Sec. 1.4.1. In this case

$$X(z) = \sum_{n=-\infty}^{+\infty} a^n u[n] z^{-n} = \sum_{n=0}^{+\infty} \left(az^{-1}\right)^{-n} = \frac{1}{1 - az^{-1}}, \tag{1.91}$$

where the last equality can be written only if $\left| az^{-1} \right| < 1$, otherwise the series does not converge. In other words the ROC is the region $|\, z \,| > |\, a \,|$. It is easy to verify that the left-sided exponential sequence $x[n] = -a^n u[-n]$ also has a z-transform, identical to the one in (1.91), but with a different ROC (the region $|\, z \,| < |\, a \,|$).

This example shows that the algebraic expression of the z-transform does not completely specify the corresponding sequence, and that the ROC must also be specified. The example also shows a case of a sequence that has a z-transform but does not have a DTFT: for $a \geq 1$ the right-sided exponential sequence still admits the z-transform $1/(1 - az^{-1})$ in the region $|\, z \,| > a > 1$ although it increases exponentially in time and does not have a DTFT.

The *right-sided real sinusoidal sequence* $x[n] = \cos(\omega_0 n) u[n]$. Note that it can be written as a sum of two exponential sequences: $x[n] = \left(e^{j\omega_0 n} u[n] + e^{-j\omega_0 n} u[n]\right)$. Therefore

$$X(n) = \frac{1}{2}\left[\frac{1}{1 - e^{j\omega_0} z^{-1}} + \frac{1}{1 - e^{-j\omega_0} z^{-1}}\right] = \ldots = \frac{1 - [\cos\omega_0] z^{-1}}{1 - 2[\cos\omega_0] z^{-1} + z^{-2}}. \tag{1.92}$$

Since $|\, \cos\omega_0 \,| \leq 1$, the ROC is clearly the region $|\, z \,| > 1$. This is a second example of a sequence that does not admit a DTFT but admits a z-transform.

A final important example is the *exponentially damped* right-sided sinusoidal sequence, defined as $x[n] = r^{-n} \cos(\omega_0 n) u[n]$, with $0 < r < 1$. In this case

$$X(n) = \frac{1 - [r\cos\omega_0] z^{-1}}{1 - 2[r\cos\omega_0] z^{-1} + r^2 z^{-2}}. \tag{1.93}$$

The ROC is the region $|\, z \,| > r$. Note that in this case the ROC includes the unit circle, therefore the sequence $x[n]$ also admits a DTFT. In fact as we will see this sequence represents the impulse response of a second-order resonating filter.

1.6.1.3 Rational z-transforms

A very important and useful family of z-transforms is that of *rational* transforms, i.e. those that can be written as

$$X(z) = \frac{P(z)}{Q(z)}, \qquad \text{with} \quad P(z), Q(z) \quad \text{polynomials.} \tag{1.94}$$

In fact in the previous section we have just examined a few examples of sequences with rational transforms, in particular the right-sided exponential sequence (1.91). Recalling that the z-transform is linear, we can say that any sequence that can be expressed as a linear combination of right-sided exponentials has a rational z-transform.

$$x[n] = \sum_{k=1}^{N} A_k a_k^n u[n] \quad \Rightarrow \quad X(z) = \frac{\sum_{k=1}^{N} A_k \prod_{m \neq k} (1 - a_m z^{-1})}{\prod_{k=1}^{N} (1 - a_k z^{-1})} \tag{1.95}$$

The values z for which the $P(z) = 0$ (and therefore $X(z) = 0$) are called *zeros* of $X(z)$. The values z for which $Q(z) = 0$ are called *poles* of $X(z)$. A number of important relations exist between the poles of a rational transform and its ROC.

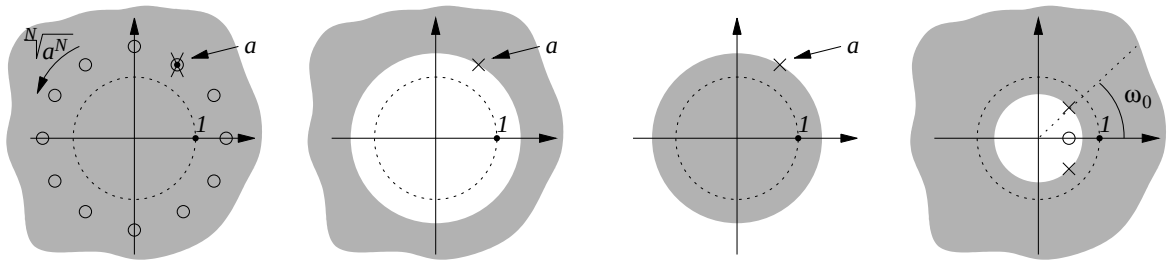


Figure 1.14: *Pole-zero plots and ROCs of some simple transforms: (a) finite-length exponential sequence; (b) right-sided exponential sequence with* $|\,a\,| > 1$*; (c) left-sided exponential sequence with* $|\,a\,| > 1$*; (d) exponentially damped, right-sided sinusoidal sequence. In all plots the dotted circle is the unit circle and the gray-shaded region is the ROC of the corresponding transform.*

First, the ROC cannot contain poles by definition, since $X(z)$ is not defined on the poles. It follows immediately that a finite-length sequence cannot have any poles. As an example, looking at Eq. (1.90) one notices that the pole at $z = a$ cancels with one of the zeros of the numerator $(1 - a^N z^{-N})$, therefore there are no poles. In a similar line of reasoning one can prove that for any right-sided sequence the ROC extends outwards from the pole with largest absolute value towards $|\,z\,| \to +\infty$, and that for any left-sided sequence the ROC extends inwards from the pole with smallest absolute value towards $z \to 0$. For a generic sequence that extends infinitely on both sides, the ROC consists of a ring bounded by a pole on the interior and exterior.

So-called *pole-zero plots* are typically used to represent z-transforms and their associated ROCs. Conventionally a zero is denoted with a "$\circ$" symbol and a pole is denoted with a "$\times$" symbol. As an example. figure 1.14 shows the pole-zero plots for some of the transforms discussed in the previous section. Note in particular that the right-sided and left-sided exponential sequences have identical pole-zero patterns, but have different ROCs.

Since the pole-zero pattern does not completely define the corresponding sequence, it is sometimes convenient to specify some time-domain properties of the sequences, that implicitly define the ROC. As an example, consider the pole-zero plot of either Fig. 1.14(b) or Fig. 1.14(c) and assume that the ROC is not known. If one states that the corresponding sequence is absolutely summable, then this implies that it admits a DTFT and consequently implies that the ROC must be that of Fig. 1.14(c). Alternatively one may state that the corresponding sequence is causal: this implies that the ROC must extend towards $|\,z\,| \to +\infty$ and consequently implies that the ROC must be that of Fig. 1.14(b).

1.6.2 Transfer function and frequency response of a LTI system

1.6.2.1 Definitions

In Sec. 1.2.3 we have seen that a LTI system is completely characterized by its impulse response $h[n]$, since the response to any imput sequence $x[n]$ can be written as the convolution between x and h (see Eqs. (1.22, 1.23)). Using the convolution property given in table 1.2, one can restate this by saying that for an LTI system an input sequence x is related to the corresponding output sequence y through the equation

$$Y(z) = H(z)X(z), \tag{1.96}$$

where $X(z), Y(z), H(z)$ are the z-transforms of $x[n], y[n], h[n]$, respectively. We call $H(z)$ the *transfer function* of the LTI system. Assuming that an appropriate ROC is specified for H, we can say that the LTI system is completely characterized by its transfer function.

If the ROC includes the unit circle, then $h[n]$ admits a Fourier representation. In this case we can also write

$$Y\left(e^{j\omega_d}\right) = H\left(e^{j\omega_d}\right) X\left(e^{j\omega_d}\right), \tag{1.97}$$

where $X\left(e^{j\omega_d}\right), Y\left(e^{j\omega_d}\right), H\left(e^{j\omega_d}\right)$ are the DTFTs of $x[n], y[n], h[n]$, respectively. We call $H\left(e^{j\omega_d}\right)$ the *frequency response* of the system. Assuming that the DTFTs are expressed in polar form (see Eq. (1.54)), we call $\left|H\left(e^{j\omega_d}\right)\right|$ and $\arg\left[H\left(e^{j\omega_d}\right)\right]$ the *magnitude response* and the *phase response* of the system, respectively.

If the LTI system under consideration has been expressed through a constant-coefficient difference equation (see Sec. 1.2.3), then one can immediately write the corresponding transfer function as a rational function:

$$\sum_{k=0}^{N} a_k y[n-k] = \sum_{k=0}^{M} b_k x[n-k] \quad \Leftrightarrow \quad \sum_{k=0}^{N} a_k z^{-k} Y(z) = \sum_{k=0}^{M} b_k z^{-k} X(z), \tag{1.98}$$

from which it follows immediately that

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{M} b_k z^{-k}}{\sum_{k=0}^{N} a_k z^{-k}}. \tag{1.99}$$

Such z-transforms arise frequently in digital signal processing applications. By looking at a transform of this kind, one can easily find the corresponding sequence (this is an example of an informal procedure for determining the z-transform). First one can note that $H(z)$ can be written as

$$H(z) = \frac{b_0}{a_0} \cdot \frac{\prod_{k=1}^{M}\left(1 - c_k z^{-1}\right)}{\prod_{k=1}^{N}\left(1 - d_k z^{-1}\right)}, \tag{1.100}$$

where the c_k's and the d_k's are the zeros and the poles of H, respectively. If $M \leq N$ and the poles are all first-order, then by applying a *partial fraction expansion* the above equation can be rewritten as

$$H(z) = \sum_{k=1}^{N} \frac{A_k}{1 - d_k z^{-1}}, \quad \text{with} \quad A_k = \left(1 - d_k z^{-1}\right) H(z)|_{z=d_k}. \tag{1.101}$$

Looking back at Eq. (1.91), we can conclude that $h[n]$ is a linear combination of right-sided exponential sequences (or left-sided exponential sequences, depending on the ROC).

1.6.2.2 The concept of filtering

The magnitude and phase responses of a LTI system describe how the system transforms a sinusoidal input $x[n] = A\cos(\omega_0 n) = A\left(e^{j\omega_0} + e^{-j\omega_0}\right)/2$: the corresponding output is

$$y[n] = (h * x)[n] = A\left|H\left(e^{j\omega_0}\right)\right| \cdot \cos\left(\omega_0 n + \arg\left[H\left(e^{j\omega_0}\right)\right]\right), \tag{1.102}$$

i.e. the magnitude response at ω_0 defines the *gain* and the *phase delay* of the system at the frequency $\omega_d = \omega_0$. Similar considerations apply to a generic input $x[n]$: the corresponding output can be described in terms of system magnitude and phase response as

$$\begin{aligned} \left|Y\left(e^{j\omega_d}\right)\right| &= \left|H\left(e^{j\omega_d}\right)\right| \cdot \left|X\left(e^{j\omega_d}\right)\right|, \\ \arg\left[Y\left(e^{j\omega_d}\right)\right] &= \arg\left[H\left(e^{j\omega_d}\right)\right] + \arg\left[X\left(e^{j\omega_d}\right)\right]. \end{aligned} \tag{1.103}$$

The first equation in (1.103) says that frequency components of the input are emphasized or attenuated (or even suppressed) depending on the values of $\left| H\left(e^{j\omega_d}\right) \right|$ at those frequencies. For this reason we typically refer to an LTI system as a *frequency selective filter*, or simply a *filter*. Thinking of audio, equalization is an obvious example of filtering an input sound by emphasizing certain frequency ranges and attenuating other ranges.

The second equation in (1.103) says that frequency components of the input are delayed in a frequency-dependent manner. The amount and type of tolerable *phase distortion* depends on the application. Often phase responses are disregarded in audio applications because phase distortions are to a large extent inaudible. However taking into account the phase response can be important in certain cases, e.g. when one wants to preserve the shape of the time-domain waveform.

A generally tolerable type of phase distortion is *linear distortion*. A filter with a linear phase response produces the same phase delay for all frequencies: as an example, the ideal delay system $h_{n_0} = \delta(n-n_0)$ has a linear phase response $\arg\left[H_{n_0}\left(e^{j\omega_d}\right)\right] = \omega_d n_0$. A convenient measure of the linearity of the phase response of a filter is the *group delay*, defined as[3]

$$\tau(\omega_d) = -\frac{d}{d\omega_d}\left\{\arg\left[H\left(e^{j\omega_d}\right)\right]\right\}. \tag{1.104}$$

The deviation of τ from a constant indicates the degree of phase non-linearity. Figure 1.15(a) provides a graphical comparison of the phase delay and the group delay.

The reason why τ is termed group delay is that this quantity relates to the effect of the phase on a quasi-sinusoidal signal. More precisely, consider the signal $x[n] = a[n]e^{\omega_0 n}$, and assume that $a[n]$ is varying slowly (equivalently, assume that the spectrum $A(e^{j\omega})$ is concentrated near $\omega = 0$). Then x will look like the signal in Fig. 1.15(b) (upper panel). The signal can also be rewritten as

$$x[n] = a[n]e^{\omega_0 n} = \left[\frac{1}{2\pi}\int_{-\epsilon}^{+\epsilon} A(e^{j\omega})e^{j\omega n}d\omega\right]e^{\omega_0 n}, \tag{1.105}$$

where $\epsilon \ll \pi$ is the upper limit of the band of A. Therefore x can be viewed as a superposition of neighboring sinusoidal components, or a *group* around ω_0.

Since $X\left(e^{j\omega_d}\right) \neq 0$ only in the vicinity of ω_0), the phase response can be approximated in that neighborhood as a line with slope $-\tau(\omega_0)$. With this approximation it is then quite straightforward to show that the output is the filter output $y[n]$ looks like in Fig. 1.15(b) (lower panel), i.e. $\tau(\omega_0)$ represents the delay applied to the slowly-varying amplitude $a[n]$.

1.6.2.3 Stability, causality, and transfer functions

We have defined in Sec. 1.2.3 the notion of BIBO stability, and we have proved that an LTI system is stable if and only if its impulse response is absolutely summable. This latter condition can be rewritten as

$$\sum_{n=-\infty}^{+\infty} \left| h[n]z^{-n} \right| < \infty, \tag{1.106}$$

for $|z| = 1$. Therefore the condition of stability is equivalent to the condition that the ROC of the transfer function includes the unit circle. The examples depicted in Fig. 1.14 confirm this finding.

Consider the relevant particular case of rational transfer functions associated to causal systems: for these transfer functions the condition of stability is equivalent to the condition that all the poles are inside the unit circle, because the ROC extends outwards from the pole with the largest absolute value.

[3] Our definition uses ω_d, so that τ is adimensional. Usual definitions employ ω, so that τ is in seconds.

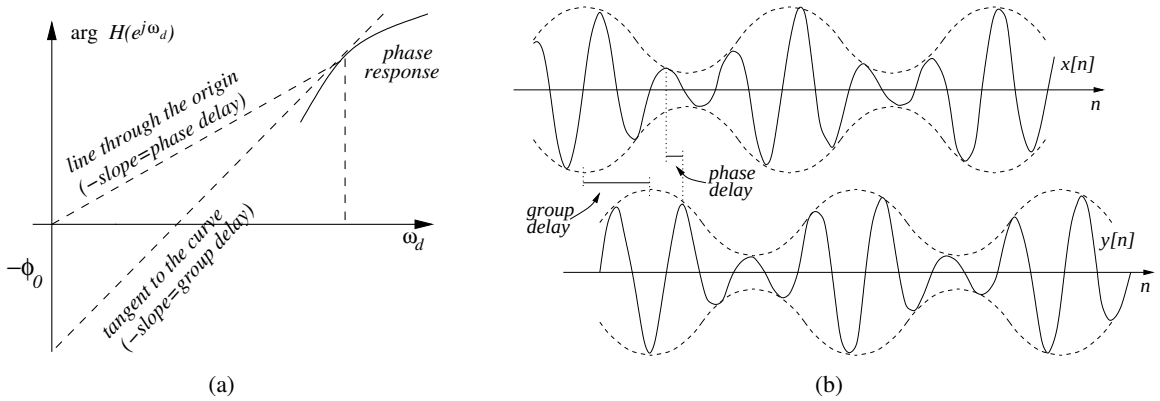


Figure 1.15: *Comparing phase delay and group delay: (a) evaluation of phase delay and group delay for a generic non-linear phase response; (b) illustration of phase delay and group delay for a narrowband signal.*

Another relevant particular case are FIR systems. We have alredy seen that a FIR system is always stable since its impulse response is always absolutely summable. This property can be "seen" in the z domain by noting that the transfer function of a FIR system does not have poles, therefore the ROC is always the entire z plane and includes the unit circle.

1.6.3 Digital filter structures and design approaches

In the following we only give a quick overview. In the next chapters we will discuss many specific filters. In particular we will examine the second-order resonant filter in Chapter *Sound modeling: signal based approaches*; comb and all-pass in Chapter *Sound modeling: source based approaches*; more comb and all-pass structures in Chapter *Sound in space*. We do not discusss filter structures (direct forms etc.), just a few words. Same with design techniques. In Chapter *Sound modeling: source based approaches* we talk about bilinear transform, s-to-z mappings, impulse response invariance, all techniques used to design a digital filter from an analog one. In Chapter *Sound in space* we will see pole-zero approximations.

1.6.3.1 Block diagram representations

Most the LTI systems that we are going to realize are represented as linear constant coefficient equations. As we have seen, the output of a LTI system represented in this way can be computed recursively provided that past values of input and output are available. This values will undergo two operations in the computation of the filter output: multiplication with coefficients, and addition.

Therefore the three basic elements for the implementation of a filter are memory for storing, past values, adders, and multipliers. A filter structure is created by properly interconnecting these elements. Figure 1.16 shows the pictorial symbols that are typically used for representing them. With these elements, a general filter

$$y[n] - \sum_{k=1}^{N} a_k y[n-k] = \sum_{k=0}^{M} b_k x[n-k] \quad \Rightarrow \quad H(z) = \frac{\sum_{k=0}^{M} b_k z^{-k}}{\sum_{k=1}^{N} a_k z^{-k}} \tag{1.107}$$

can be represented with the block diagram of Fig. 1.16(b).

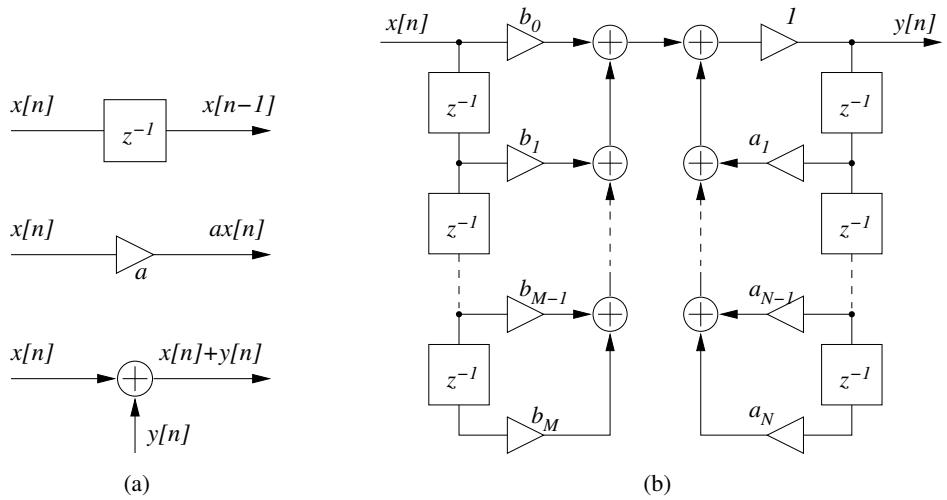


Figure 1.16: *Block diagram representation of filters: (a) conventional pictorial symbols for delay, adder, and multiplier; (b) a general filter structure.*

1.6.3.2 Filter classification

Filters can be classified according to most salient characteristics of their frequency responses, most typically their magnitude. Basic filter categories are represented in Fig. 1.17. Other types of filters can in general be described as a combination of these basic elements.

Low-pass filters (see Fig. 1.17(a), upper panel) select low frequencies up to a given *cut-off* frequency ω_c, annd attenuate higher frequencies. *High-pass* filters (see Fig. 1.17(a), lower panel) have the opposite behavior: they select frequencies above ω_c, and attenuate lower frequencies.

Band-pass filters (see Fig. 1.17(b), upper panel) select frequencies within a frequency band, specified by two cut-off frequencies ω_{c1} and ω_{c2}, while frequencies outside this band are attenuated. *Band-reject* filters (see Fig. 1.17(b), lower panel) have the opposite behavior: they select frequencies outside the band $[\omega_{c1}, \omega_{c2}]$, and attenuate frequencies within the band.

Resonator filters (see Fig. 1.17(c), upper panel) amplify frequencies in a narrow band around a cut-off frequency ω_c. Conversely, *notch* filters (see Fig. 1.17(c), lower panel) attenuate frequencies in a narrow band around ω_c. Finally, when the magnitude response is perfectly flat the filter is called an *all-pass* filter, since all frequencies are passed. Note however that an all-pass filter modifies the phase of the input signal. In the next chapter we will see some important uses of all-pass filters.

In order to optimize their frequency selective properties, ideal filters should have magnitude responses exhibiting vertical transition between selected frequencies and rejected ones, Moreover they should have null or linear phase response in order not to introduce phase distortion. As an example, the *ideal low-pass filter* has the frequency response

$$H_{lp}\left(e^{j\omega}\right) = \begin{cases} 1, & |\,\omega\,| \leq \omega_c, \\ 0, & \omega_c < |\,\omega\,| \leq \pi. \end{cases} \tag{1.108}$$

However the corresponding impulse response is

$$h_{lp}[n] = \frac{\sin \omega_c n}{\pi n}, \quad -\infty < n < +\infty, \tag{1.109}$$

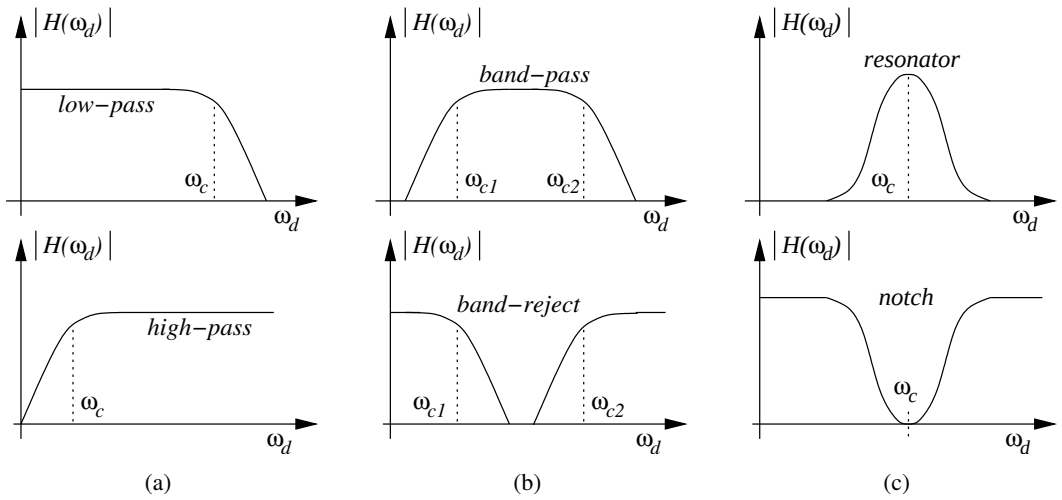


Figure 1.17: *Classification of filters into basic categories, depending on their magnitude response* $|H(\omega_d)|$*: (a) low-pass and high-pass filter; (b) band-pass and band-reject filter; (c) resonator and notch filter.*

i.e. the filter is non-causal and has a two-sided infinite impulse response. Therefore it is not possible to compute its output either recursively or non-recursively. In other words, the filter is *not realizable*. Similar considerations apply to other types of ideal filters.

The simplest examples of realizable filters are first-order filters (i.e. filters with no more than one pole and/or one zero). First-order FIR low-pass and high-pass filters are defined as follows:

$$H_{lp}(z) = \frac{1}{2}\left(1 + z^{-1}\right), \qquad H_{hp}(z) = \frac{1}{2}\left(1 - z^{-1}\right). \tag{1.110}$$

They have a zero in $z = 1$ and $z = -1$, respectively. Therefore the magnitude responses decrease and increase monotically, respectively. The low-pass filter in particular can be recognized to be a moving average filter that averages two contiguous samples.

First-order IIR low-pass and high-pass filters are defined as follows:

$$H_{lp}(z) = \frac{1-\alpha}{2}\frac{1+z^{-1}}{1-\alpha z^{-1}}, \qquad H_{hp}(z) = \frac{1+\alpha}{2}\frac{1-z^{-1}}{1-\alpha z^{-1}}. \tag{1.111}$$

Both have a pole in $z = \alpha$, therefore $|\alpha| < 1$ for stability, and $\alpha \in \mathbb{R}$ in order for the impulse responses to be real-valued. They have a zero in $z = 1$ and $z = -1$, respectively. For $\alpha = 0$ they reduce to the FIR filters above, while for $\alpha > 0$ they have steeper responses.

M-1.14

Study the frequency responses of the first-order low-pass and high-pass filters of Eqs. (1.110, 1.111). Apply them to a broad-band audio signal (e.g. a snaredrum), and study their effect.

1.7 Commented bibliography

A general reference for digital signal processing is[Oppenheim et al., 1999]. Another classic is Mitra [2005]: more implementation oriented, with Matlab examples.

Primers on digital audio processing: Rocchesso [2003] and Steiglitz [1996]. Examples focused on audio are found also in [Zölzer, 2002, Chapter 1]

A useful reading about Fourier analysis for discrete-time signals is provided in [Smith, 2008]. Our discussion of FFT algorithms is based on [Cormen et al., 2001].

A discussion on recursive generators of sinusoidal signals is found e.g. in [Orfanidis, 1996]. Models for fractal signals are also partially discussed in [Orfanidis, 1996].

References

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2001.

Sanjit K Mitra. *Digital Signal Processing*. McGraw-Hill, third edition, 2005.

Alan V. Oppenheim, Ronald W. Schafer, and John R. Buck. *Discrete-Time Signal Processing*. Prentice Hall, Upper Saddle River, NJ, 1999.

Sophocles J. Orfanidis. *Introduction to Signal Processing*. Prentice Hall, 1996.

Davide Rocchesso. *Introduction to Sound Processing*. Mondo Estremo, Firenze, 2003. http://profs.sci.univr.it/˜rocchess/SP.

Julius O. Smith. *Mathematics of the Discrete Fourier Transform (DFT)*. http://ccrma.stanford.edu/ jos/mdft/, 2008. Online book, accessed Oct. 2008.

Kenneth Steiglitz. *A Digital Signal Processing Primer - With Applications to Digital Audio and Computer Music*. Prentice Hall, 1996.

Udo Zölzer, editor. *DAFX – Digital Audio Effects*. John Wiley & Sons, 2002.

Contents