Chapter 6

Discrete-Time Filters

Linear and time-invariant systems represent the fundamental class of signal processing devices. In this chapter we will study their many properties in detail, but we will always make sure to support and complement each theoretical derivation with practical examples. Our helpers in this journey are going to be two of the most useful and commonly used discrete-time filters, the moving average and the leaky integrator; they will be introduced informally at first, as intuitive solutions to a data processing toy problem, and they will reappear throughout the chapter in many concrete demonstrations of the properties that characterize all LTI systems.

The fundamental takeaway concept in this chapter is the *convolution* operation, which emerges naturally as the answer to two apparently unrelated questions:

- what are the consequences of requiring linearity and time invariance in a signal processing device? and
- how can we "reshape" the spectrum of a signal via multiplication by a desired "mask"?

The fact that both problems are solved by the same mathematical device is arguably the main reason behind the success of LTI systems. But of course the convolution appears in many other contexts as well (deep learning being probably the most famous) and, in general, we can expect it to be the answer whenever we want to use a signal to modify another signal in a meaningful way.

6.1 Linear Time-Invariant Systems

In its most general form, a discrete-time system is a processing unit that accepts one or more sequences as inputs and produces one or more sequences as outputs. This definition is clearly very broad and it's only by imposing additional constraints that we can arrive at a precise mathematical characterization of a system's properties. In this chapter we will thus focus on synchronous, single-input single-output, linear time-invariant (LTI)

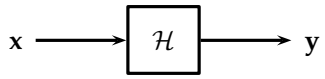

Figure 6.1: *Representation of a discrete-time filter.*

systems, commonly known as *discrete-time filters*, whose schematic representation is shown in Figure 6.1. A synchronous system produces an output sample for every incoming input sample and so a filter acts as an operator transforming an input sequence into an output sequence:

$$\mathbf{y} = \mathcal{H}\mathbf{x}; \tag{6.1}$$

the operation is said to be linear if, for any choice of $\mathbf{x}$ and $\mathbf{w}$ and for any scalar α, we have

$$\mathcal{H}(\alpha\mathbf{x}) = \alpha(\mathcal{H}\mathbf{x}) \tag{6.2}$$

$$\mathcal{H}(\mathbf{x} + \mathbf{w}) = \mathcal{H}\mathbf{x} + \mathcal{H}\mathbf{w} \tag{6.3}$$

and it is time-invariant if

$$\mathcal{H}(\mathcal{S}^k\mathbf{x}) = \mathcal{S}^k(\mathcal{H}\mathbf{x}) \quad \forall k \in \mathbb{Z} \tag{6.4}$$

where $\mathcal{S}$ is the time shift operator[1].

Linearity and time invariance describe in idealized form the kind of "reasonable" properties that we expect of a system. Consider for instance the case of an electric guitar connected to an amplifier: if we strum multiple strings at once we expect to hear a sound that's equal to the superposition of the sounds we would obtain by plucking each string independently, that is, we expect the amplifier to behave linearly. Similarly, it's reasonable to assume that the amplifier will operate in exactly the same way no matter *when* we start playing; that is, we expect it to be time-invariant. Nevertheless, even the best real-world systems are always only approximately LTI; linearity, in particular, is guaranteed only if the input amplitude is within the nominal operating range declared by the manufacturer. Time invariance is usually less of a problem but, over time, all phsyical devices will start to show signs of wear and ageing, and their characteristics may change significantly.

Finally, let's also note that LTI systems are not always what we want or need: the sound of a distorted electric guitar, for instance, is the result of carefully designed but deliberately nonlinear audio processing devices. Adaptive systems, which are at the heart of all modern telecommunication techniques, are inherently time-variant since their response changes a function of external circumstances independent of the input signal. However, as we are about to see, it's only with linearity and time invariance that a rich set of mathematical results can be developed to the analysis of discrete-time systems; and these results remain foundational even when, in the final design, a processing device also includes non-LTI components.

[1]See Section **??** for a recap on the notation used for signal operators.

6.1.1 Denoising filters

The best way to discover and understand the properties of LTI systems is to observe them "in action"; this is particularly easy to do in the case of discrete-time filters because they are numerical algorithms whose software implementation can be achieved in just a few lines of code.

A simple, intuitive, and yet realistic example is the problem of how to "clean up" a signal affected by additive random noise; typically, any set of discrete-time experimental measurements $\mathbf{x}$ can be modeled as the sum

$$\mathbf{x} = \mathbf{x}_c + \boldsymbol{\eta}$$

where $\mathbf{x}_c$ is the clean signal that we are trying to capture and $\boldsymbol{\eta}$ is a sequence of random noisy perturbations that affect each measured sample as a consequence of various unpredictable factors in the experimental setup (thermal noise, interferences, vibrations, etc.) With no additional information on the nature of the signals at play, it is obviously impossible to recover $\mathbf{x}_c$ and $\boldsymbol{\eta}$ from their observed sum $\mathbf{x}$; in general, however, the signal and the noise possess very different characteristics, something we can exploit to design a denoising strategy. For instance, we can almost always assume the clean signal to be smooth and slow-varying compared to the quickly-changing sequence of random noise samples; a typical example is shown in Figure 6.2, where the noise is visually apparent as small wiggles distorting the shape of a much slower, smooth signal.

Experimentally, when faced with noisy measurements, a standard approach is to apply some form of averaging to the data, which works as long as the noise is sufficiently random and its mean value is close to zero. We will now describe two discrete-time filters that implement such an averaging operation and that can be used to denoise a signal. In this section we will focus on proving their linearity and time invariance starting from their completely intuitive algorithmic design. In the rest of this chapter we will return to both filters to illustrate most of the fundamental properties of general LTI systems.

The Moving Average. If averaging reduces the effects of random noise, a simple idea to clean up a signal is to replace every sample with a *local average* of M neighboring samples; intuitively, if the clean signal is slow-varying, the local average will not be very far from the clean sample value but, if the noise is random, its local average will be close to zero. We can formalize this approach as a system whose input-output relationship is

$$y[n] = \frac{1}{M} \sum_{k=0}^{M-1} x[n-k]. \tag{6.5}$$

This algorithmic expression is easily implemented on a computing device and so we can already verify experimentally that it does smooth the input signal, and that the smoothing power increases with M, as shown in Figure 6.3. Note that the algorithm in (6.5) uses only current and past input samples, and thus it can work in real time (that is, it does not rely on future, unknown data).

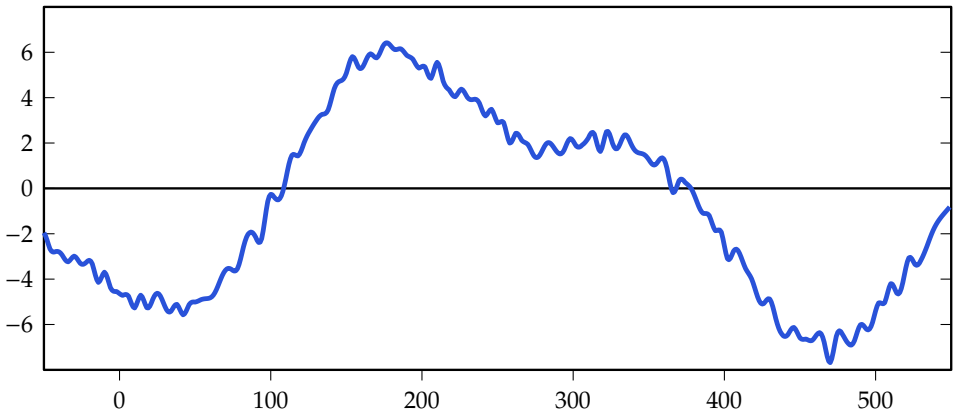

Figure 6.2: *A discrete-time noisy signal.*

The LTI system described by Equation (6.5) is called a *moving average* (MA) filter. One informal way to show its LTI nature is to rewrite Eq. (6.5) in operator notation as

$$\mathcal{H}\mathbf{x} = \frac{1}{M}\sum_{k=0}^{M-1} \mathcal{S}^{-k}\mathbf{x}; \tag{6.6}$$

and to notice that it involves only linear operations (sums and scalar multiplications) and time-invariant operators (discrete-time delays). Formally, as a useful exercise, we can prove both properties explicitly as follows: to have linearity it must be

$$\mathcal{H}(\alpha\mathbf{x}) = \alpha\,\mathcal{H}\mathbf{x}$$
$$\mathcal{H}(\mathbf{x}+\mathbf{w}) = \mathcal{H}\mathbf{x} + \mathcal{H}\mathbf{w}$$

and, indeed,

$$\frac{1}{M}\sum_{k=0}^{M-1} \alpha x[n-k] = \alpha\frac{1}{M}\sum_{k=0}^{M-1} x[n-k]$$
$$\frac{1}{M}\sum_{k=0}^{M-1} (x[n-k]+w[n-k]) = \frac{1}{M}\sum_{k=0}^{M-1} x[n-k] + \frac{1}{M}\sum_{k=0}^{M-1} w[n-k];$$

similarly, time invariance requires

$$\mathcal{H}(\mathcal{S}^m\mathbf{x}) = \mathcal{S}^m(\mathcal{H}\mathbf{x})$$

and, indeed,

$$\frac{1}{M}\sum_{k=0}^{M-1} x[(n+m)-k] = \frac{1}{M}\sum_{k=0}^{M-1} x[(n-k)+m]$$

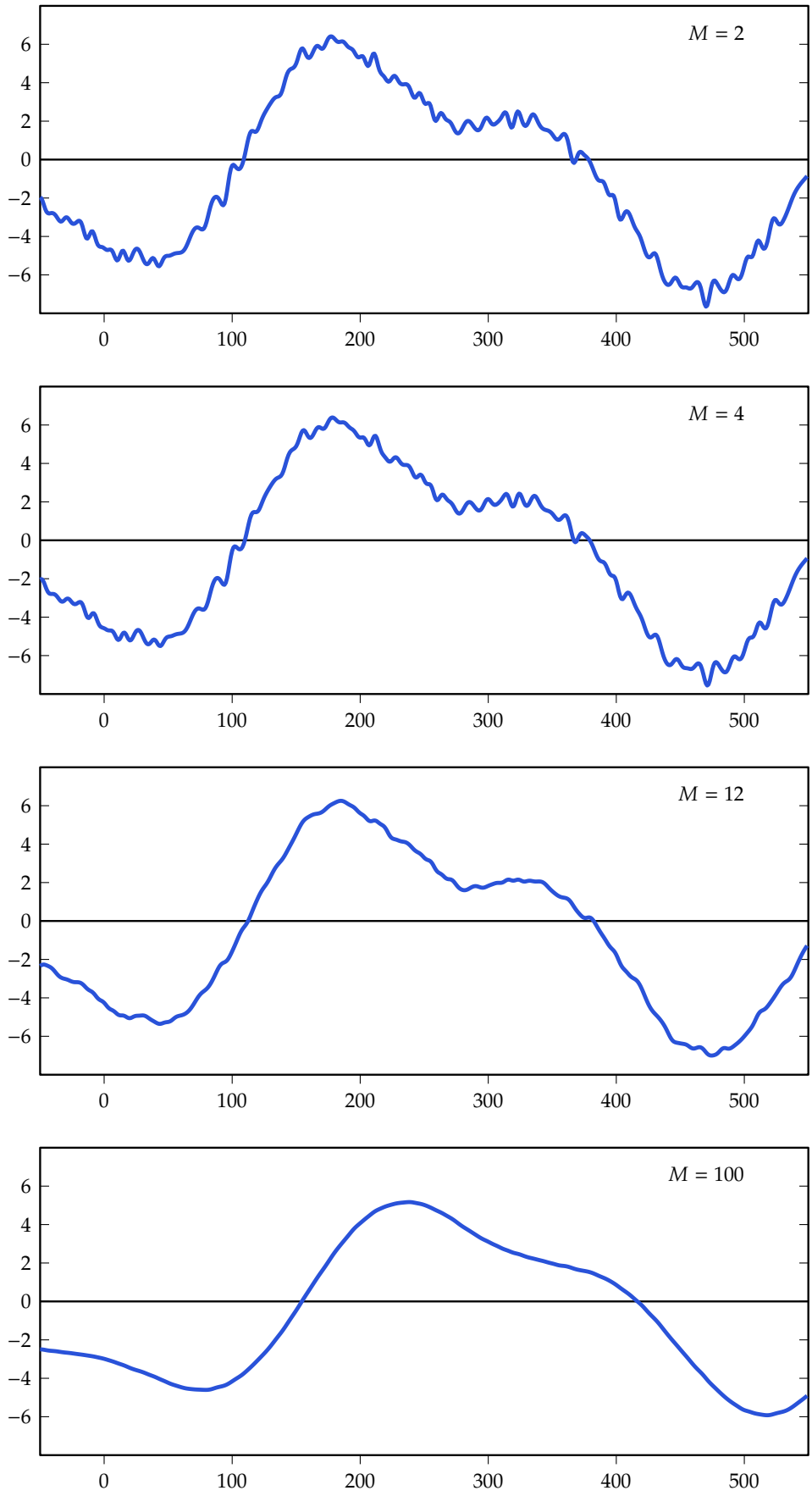

Figure 6.3: *Effects of the moving average filter for different values of M.*

The moving average algorithm requires $M-1$ additions and one multiplication per output sample, as well as a memory buffer of length $M-1$ to keep track of past input values; the computational requirements of the filter are thus proportional to M (see also Exercise **??**).

The Leaky Integrator. Many practical denoising applications require a significant amount of data smoothing and, with a moving average filter, this can only be attained with large values for M. Since the computational resources of a signal processing device can often be quite limited, it would be useful to have an equivalent averaging filter whose computational cost remained constant and independent of smoothing power.

Since the computational costs are proportional to M and since the available processing resources are often limited, we would like to design an averaging filter with b

As an alternative, consider the following ideas: first of all, instead of computing the average from scratch every time, which requires M additions, we could instead "update" the previous result like so: if

$$y[n] = \frac{1}{M}\left(x[n] + x[n-1] + \ldots + x[n-M+2] + x[n-M+1]\right)$$

and

$$y[n-1] = \frac{1}{M}\left(x[n-1] + x[n-2] + \ldots + x[n-M+1] + x[n-M]\right)$$

then the updated average can be obtained by "forgetting" the oldest sample and replacing it with the new:

$$\begin{aligned} y[n] &= \frac{1}{M}x[n] + \frac{1}{M}(x[n-1] + \ldots + x[n-M+1] + x[n-M]) - \frac{1}{M}x[n-M] \\ &= y[n-1] - \frac{1}{M}x[n-M] + \frac{1}{M}x[n]. \end{aligned}$$

Although this requires only two additions and one multiplication per output sample, the storage requirements remain the same since we need to remember the past M input samples. But what if, instead of removing the oldest sample exactly, we simply discard from the current mean a fraction $1/M$ of its own value? With the approximation

$$y[n-1] - \frac{1}{M}x[n-M] \approx y[n-1] - \frac{1}{M}y[n-1]$$

we can rewrite the *recursive* update as

$$y[n] = \lambda y[n-1] + (1-\lambda)x[n], \qquad \lambda = \frac{M-1}{M} \tag{6.7}$$

which requires only two multiplications and one addition per sample and uses just one memory cell to keep track of the previous output. Before analyzing the structure and

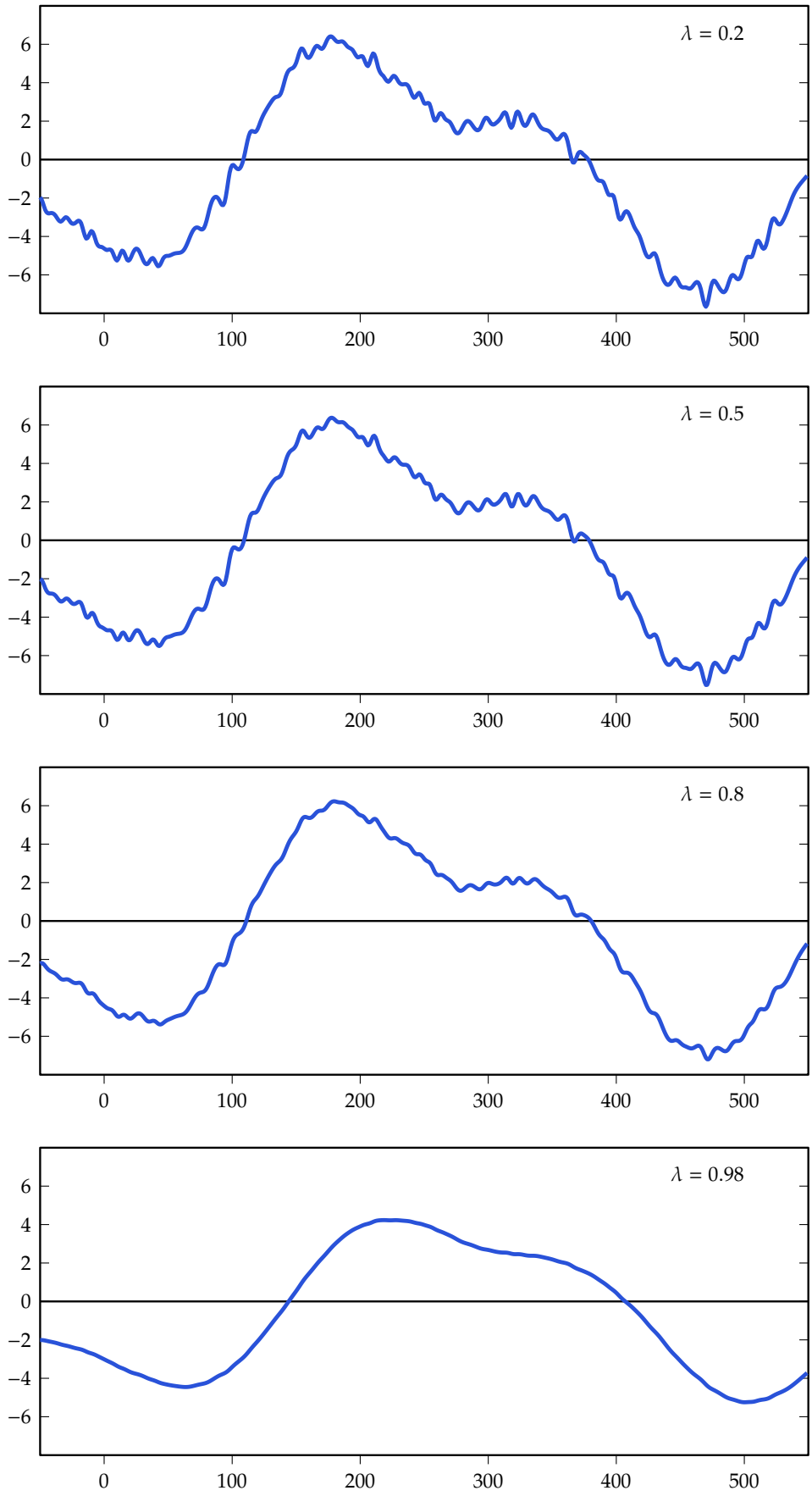

Figure 6.4: *Effects of the leaky integrator for different values of λ.*

the implications of this new approach, we can easily verify numerically that it works; Figure 6.4 shows the effects of the algorithm on the input sequence in Figure 6.2 for different values of λ and we can notice that the smoothing power increases as λ approaches unity.

The system described by Equation (6.7) is called a *leaky integrator* and it's one of the most common filters used in signal processing. But before we can call it a filter, we need to show that the system is linear and time invariant and this is a bit more complicated to do than in the case of the moving average because the algorithm is recursive: if every output value depends on its predecessor, there seems to be no starting point for a formal proof. The way around this "chicken and egg" situation is assuming that the system is intially off and that it is only switched on at a known time instant. This working hypothesis is called *zero initial conditions* and, formally, it states that there exists a time index n_0 such that:

- the input to the system is identically zero for $n < n_0$
- the output of the system is also identically zero for $n < n_0$
- at $n = n_0$ all memory elements in the systems are filled with zeros.

Assuming zero initial conditions, the response to an input sequence $\mathbf{x}$ is

$$y[n] = \begin{cases} 0 & n < n_0 \\ \lambda y[n-1] + (1-\lambda)x[n] & n \geq n_0 \end{cases}$$

and, very importantly, this response is guaranteed to be unique: namely, if $\mathcal{H}\mathbf{x} = \mathbf{y}$ then there cannot exist a sequence $\mathbf{w} \neq \mathbf{y}$ such that $\mathcal{H}\mathbf{x} = \mathbf{w}$. By contradiction, assume this were not the case and let $\mathbf{t} = \mathbf{y} - \mathbf{w}$:

$$\begin{aligned} t[n] &= \begin{cases} 0 & n < n_0 \\ \lambda y[n-1] + (1-\lambda)x[n] - \lambda w[n-1] - (1-\lambda)x[n] & n \geq n_0 \end{cases} \\ &= \begin{cases} 0 & n < n_0 \\ \lambda t[n-1] & n \geq n_0 \end{cases} \\ &= 0 \end{aligned}$$

but, if $\mathbf{t} = \mathbf{0}$ then necessarily $\mathbf{w} = \mathbf{y}$. Although this result may seem self-evident at first, please note that, for $n \geq n_0$, we proved $t[n] = 0$ by induction, and this is only valid under zero initial conditions. As a counterexample, consider a defective leaky integrator that fails to reset its internal storage when switched on, so that at time $n = n_0$ its memory cell contains a random nonzero value a; when applied repeatedly to the same input sequence, this system will return a different output signal every time:

$$y[n] = \begin{cases} 0 & n < n_0 \\ a + x[0] & n = n_0 \\ \lambda y[n-1] + (1-\lambda)x[n] & n > n_0 \end{cases}$$

This uniqueness result is the key to showing that the leaky integrator is indeed LTI. The starting point is the input-output relation $\mathbf{y} = \mathcal{H}\mathbf{x}$ with zero initial conditions at $n = n_0$. To prove linearity, we begin by showing that the sequences $\alpha\mathbf{x}$ and $\alpha\mathbf{y}$ satisfy the leaky integrator's input-output relation for all $\alpha \in \mathbb{C}$, which is obviously the case:

$$y[n] = \lambda y[n-1] + (1-\lambda)x[n] \Longrightarrow \alpha y[n] = \lambda \alpha y[n-1] + (1-\lambda)\alpha x[n];$$

since the response is unique, then $\mathcal{H}(\alpha\mathbf{x}) = \alpha\mathbf{y}$. The same argument can be used to show that $\mathcal{H}(\mathbf{x} + \mathbf{w}) = \mathcal{H}\mathbf{x} + \mathcal{H}\mathbf{w}$.

To prove time invariance, let $\mathbf{s} = \mathcal{S}^m\mathbf{x}$ be a time-shifted version of the input; note that, when shifting the input, the start time for zero initial conditions is also shifted by the same amount: if $x[n] = 0$ for $n < n_0$, then $s[n] = x[n+m] = 0$ for $n < n_0 - m$. Let $\mathbf{d} = \mathcal{H}\mathbf{s}$; we want to show that $\mathbf{d} = \mathcal{S}^m\mathbf{y}$. From the expression for the output

$$d[n] = \begin{cases} 0 & n < n_0 - m \\ \lambda d[n-1] + (1-\lambda)s[n] = \lambda d[n-1] + (1-\lambda)x[n+m] & n \geq n_0 - m \end{cases}$$

a first change of variable $i = n + m$ yields

$$d[i-m] = \begin{cases} 0 & i < n_0 \\ \lambda d[i-m-1] + (1-\lambda)x[i] & i \geq n_0 \end{cases}$$

and with a second change of variable $c[i] = d[i-m]$ we obtain

$$c[i] = \begin{cases} 0 & i < n_0 \\ \lambda c[i-1] + (1-\lambda)x[i] & i \geq n_0 \end{cases}$$

This shows that $\mathbf{c} = \mathcal{H}\mathbf{x}$ which, because of uniqueness, also means that $\mathbf{c} = \mathbf{y}$. But, by definition, $\mathbf{c} = \mathcal{S}^{-m}\mathbf{d}$ or, equivalently,

$$\mathbf{d} = \mathcal{S}^m\mathbf{c} = \mathcal{S}^m\mathbf{y}$$

which is what we wanted to prove. In the end,

$$\mathcal{H}(\mathcal{S}^m\mathbf{x}) = \mathcal{S}^m(\mathcal{H}\mathbf{x}).$$

To conclude our first encounter with the leaky integrator, a few words on the origins of its name. Consider a system that computes the running sum of a sequence up to time n:

$$s[n] = \sum_{k=-\infty}^{n} x[k];$$

this operation is the discrete-time equivalent of computing the integral of a function. The running sum can be clearly computed recursively as

$$s[n] = s[n-1] + x[n]$$

and so a leaky integrator is a discrete-time integrator where the running sum "leaks" a little bit at each step according to the value of λ (which is usually close to one); as we will see shortly, this leakage is what prevents the divergence of the filter's output.

6.2 The Impulse Response

For the filters introduced in the previous section, in both cases we started from an intuitive *algorithmic* procedure and then we checked if it indeed defined an LTI transformation. We will now flip our approach and, assuming linearity and time invariance, we will derive the general properties of an LTI system in the time domain. When designing or choosing a filter for a specific application, the first step will always be choosing one of the different filter "families" defined by these properties.

The *impulse response* is the output produced by a discrete-time system when the input is the delta sequence:

$$\mathbf{h} = \mathcal{H}\boldsymbol{\delta}. \tag{6.8}$$

If linear and time-invariant, a system is fully described by its impulse response; in particular, given an arbitrary input signal $\mathbf{x}$, the system's output can be computed algorithmically using only $\mathbf{h}$. To see this, remember that we can always express a signal as the sum of scaled atomic components (that is, as in Eq. (**??**), as a linear combination of the canonical basis vectors for $L_2(\mathbb{Z})$), namely:

$$\mathbf{x} = \sum_{k=-\infty}^{\infty} x[k]\,\boldsymbol{\delta}_k,$$

where $\boldsymbol{\delta}_k$ is the shifted delta sequence $\mathcal{S}^{-k}\boldsymbol{\delta}$ or, explicitly,

$$\delta_k[n] = \delta[n-k] = \begin{cases} 1 & n = k \\ 0 & n \neq k. \end{cases}$$

Because of linearity and time invariance, the system's response to $\mathbf{x}$ can be expressed as:

$$\begin{aligned} \mathcal{H}\mathbf{x} &= \mathcal{H}\left(\sum_{k=-\infty}^{\infty} x[k]\,\mathcal{S}^{-k}\boldsymbol{\delta}\right) \\ &= \sum_{k=-\infty}^{\infty} x[k]\,\mathcal{H}\left(\mathcal{S}^{-k}\boldsymbol{\delta}\right) \\ &= \sum_{k=-\infty}^{\infty} x[k]\,\mathcal{S}^{-k}\mathbf{h}. \end{aligned} \tag{6.9}$$

that is, as the linear combination of time-shifted copies of the impulse response, each scaled by an input sample. This special linear combination is called the *convolution* of the sequences $\mathbf{x}$ and $\mathbf{h}$, and it is compactly notated as

$$\mathbf{x} * \mathbf{h}$$

Explicitly, if $\mathcal{H}\boldsymbol{\delta} = \mathbf{h}$ and $\mathcal{H}\mathbf{x} = \mathbf{y}$, the convolution can be expressed in algorithmic form as

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]; \tag{6.10}$$

and, with the change of variable $m = n - k$, commutativity is easily proven:

$$(\mathbf{x} * \mathbf{h})[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k] = \sum_{m=-\infty}^{\infty} x[n-m]h[m] = (\mathbf{h} * \mathbf{x})[n]. \tag{6.11}$$

We will return to the meaning and the properties of Equations (6.9) and (6.10) in Section 6.3.

6.2.1 Filter properties in the time domain

By analyzing its impulse response, a filter can be characterized in three fundamental ways.

FIR vs IIR. A filter's impulse response is always an infinite-length signal (since the delta sequence is infinte-length) but it may have only a finite number of nonzero samples[2]. Every discrete-time filter therefore belongs to one of two families:

- **FIR** filters, for which the impulse response is finite-support. For an FIR, the sum in (6.11) only contains a finite number of terms, ensuring a fixed upper bound on the amount of computation per output sample. Additionally, since the output of an FIR is a linear combination of input values only, these filters have no feedback paths and are always stable, as we will see momentarily.
- **IIR** filters, for which the impulse response has infinite support. The important subclass of *realizable filters* comprises the IIRs that can be implemented with a finite amount of computation per output sample; as we will show later, these filters have a one-sided impulse response and, algorithmically, they contain one or more internal feedback paths.

Causality. A system is called *causal* if its output at time n does not depend on future input values; intuitively, if we want a filter to work in real-time, the filter must necessarily be causal. An LTI system with impulse response $\mathbf{h}$ is causal if and only if $h[n] = 0$ for $n < n_0$ for some $n_0 \geq 0$; indeed, in this case, the sum in (6.10) becomes

$$y[n] = \sum_{k=-\infty}^{n-n_0} x[k]h[n-k],$$

[2]In signal processing parlance, the nonzero values in the impulse response are often called *taps*.

which only involves past input values up to $n - n_0$. By extension, we call any right-sided sequence a strictly causal sequence.

When **h** is not strictly causal there are three possibilities:

1. the impulse response is nonzero only for a finite number of negative indexes, i.e., $h[n] = 0$ for $n < n_0$ with n_0 a negative integer. In this case we call the system *causal up to a delay* since it can be made causal via a finite time shift. Clearly, an FIR filter is always causal up to a delay.
2. the impulse response is a left-handed sequence, i.e. $h[n] = 0$ for $n > 0$, in which case we call the system *anticausal*. Anticausal filtes appear mostly in textbook exercises but can be of some practical interest in applications that work "offline" (that is, not in real time).
3. the impulse response has infinite, two-sided support. Filters in this family cannot be implemented in practice since each output sample depends on an infinite amount of future input samples. We call these filters *ideal* because, with their abstract nature, they provide the theoretical baseline against which the performance of all realizable systems will be compared.

Stability. A natural requirement for a processing device is that the output shouldn't "blow up" if the input is reasonable, namely, if the input is a signal with a finite amplitude range. In technical terms this is known as BIBO stability: a Bounded Input should produce a Bounded Output.

In LTI systems, a necessary and sufficient condition for BIBO stability is the absolute summabilty of the impulse response. To prove that the condition is sufficient, assume that the input is bounded, that is, we have $|x[n]| \leq L \in \mathbb{R}^+$ for all values of n; with this

$$\begin{aligned} \big|y[n]\big| &= \left|\sum_{k=-\infty}^{\infty} h[k]x[n-k]\right| \\ &\leq \sum_{k=-\infty}^{\infty} \big|h[k]x[n-k]\big| \\ &\leq L \sum_{k=-\infty}^{\infty} \big|h[k]\big| \end{aligned}$$

and the last term is indeed finite when **h** is absolutely summable. Conversely, if the impulse response is *not* absolutely summable, we can find at least one bounded input that makes the output diverge. Indeed, consider the sequence

$$x[n] = \begin{cases} 1 & \text{if } h[-n] \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

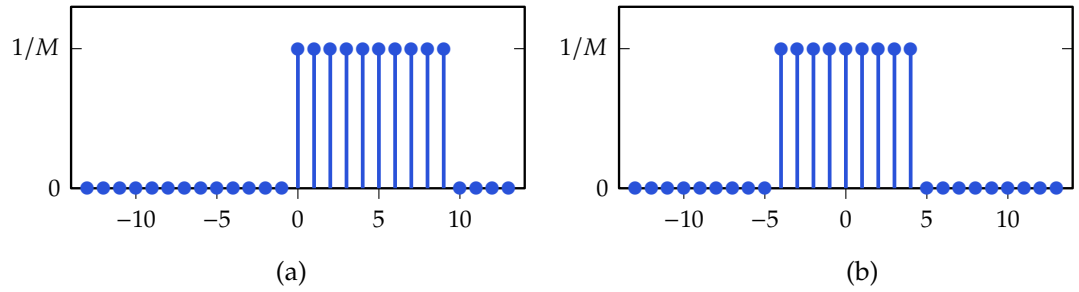

Figure 6.5: *Impulse response for a moving average filter of length $M = 9$; (a) causal and (b) noncausal, zero-centered implementation.*

which is clearly bounded since $|x[n]| \leq 1$ and let's compute the system's response to $\mathbf{x}$ for $n = 0$: $\mathbf{x}$ is clearly bounded since it takes values only over the set $\{-1, 0, +1\}$, and yet

$$y[0] = \sum_{k=-\infty}^{\infty} h[k]x[-k] = \sum_{k=-\infty}^{\infty} \big|h[k]\big| = \infty.$$

An important corollary is that **FIR filters are always BIBO stable**, since their impulse response contains only a finite number of nonzero samples.

6.2.2 Examples revisited: impulse response

Moving Average. The impulse response of the moving average is

$$h[n] = \frac{1}{M} \sum_{k=0}^{M-1} \delta[n-k] = \begin{cases} \dfrac{1}{M} & 0 \leq n < M \\ 0 & \text{otherwise.} \end{cases} \tag{6.12}$$

Since this is a right-sided, finite-support sequence, the moving average is a causal FIR, which is BIBO stable by definition.

The causal nature of the impulse response provides an explanation for the processing delay introduced by the filter. Consider the problem of computing the local average of a signal at a specific time n; the intuitive approach would be to select a set of samples *symmetrically centered* around $x[n]$, namely

$$y[n] = \frac{1}{2L+1} \sum_{k=-L}^{L} x[n-k].$$

This defines a noncausal moving average whose impulse response is centered in zero and Figure 6.6 shows how this filter eliminates the lag between input and output that was observed in Figure 6.3. From the practical point of view, the zero-centered moving

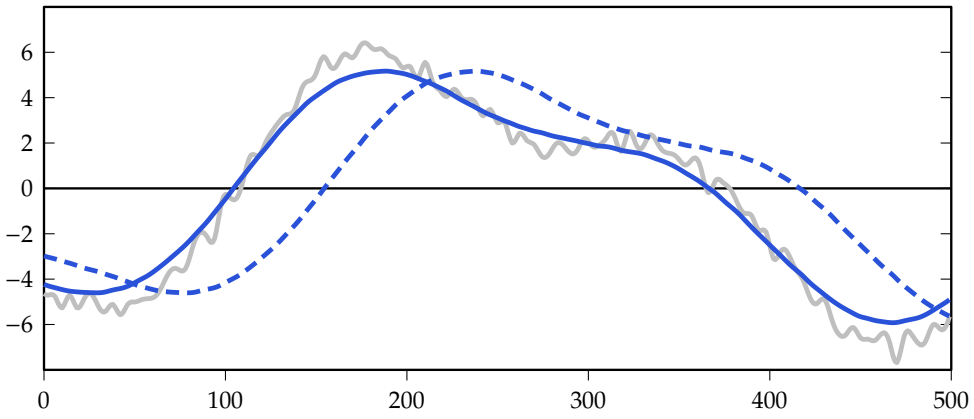

Figure 6.6: *Ouptuts of a causal (dashed line) and noncausal, zero-centered 101-tap moving average filter. Input signal is shown in gray.*

average is causal up to a delay, so it can be implemented using its causal version; in order to re-align input and output, the output needs to be delayed by L samples.

Leaky Integrator. In order to compute the impulse response of a leaky integrator we can assume zero initial conditions at $n_0 = 0$ since the delta sequence is zero for all negative values of the index. With this, we can proceed iteratively until we detect a pattern:

$$
\begin{aligned}
h[n] &= 0 \qquad \text{for } n < 0 \\
h[0] &= 1 - \lambda \\
h[1] &= (1 - \lambda)\lambda \\
h[2] &= (1 - \lambda)\lambda^2 \\
&\vdots \\
h[n] &= (1 - \lambda)\lambda^n .
\end{aligned}
$$

The impulse response is therefore a causal, infinite-support exponential sequence, a portion of which is shown in Figure 6.7-(a) for $\lambda = 0.8$. The leaky integrator is thus an IIR filter and therefore we must explicitly check if it is stable. One way to do so is verifying that the impulse response is absolutely summable, as discussed in Section 6.2.1; we obtain the geometric sum

$$\sum_{n=-\infty}^{\infty} \big|h[n]\big| = |1 - \lambda| \sum_{n=0}^{\infty} \big|\lambda^n\big| \tag{6.13}$$

which is finite if and only if $|\lambda| < 1$. We have seen that the smoothing power of the Leaky Integrator increases as λ gets closer to one and now the condition for BIBO stability explains why λ should always remain smaller than one in magnitude.

Interestingly, although its impulse response has infinite length, the leaky integrator can be implemented with a finite number of operations per output sample and it is thus a realizable IIR filter. Like the moving average, the processing delay increases with the smoothing power of the filter, but in order to quantify this delay we need to wait until we can analyze filters in the frequency domain.

6.3 Convolution

Convolution is a mathematical operation that emerges naturally in the study of many unrelated problems, from algebra to probability theory, from physics to data science. This surprising diversity makes it difficult to choose a formal definition that's both generic and intuitive and perhaps the best approach is simply to say that "convolution is what convolution does". In the present context of discrete-time filters, what convolution does is use a sequence (the impulse response) to modify another (the system's input).

6.3.1 Mathematical Properties

The convolution of two sequences $\mathbf{x}$ and $\mathbf{y}$ is well-defined if and only if the sum

$$\sum_{k=-\infty}^{\infty} x[k]y[n-k].$$

is finite for all values of n. This is always the case if both sequences are absolutely summable but that's not a necessary condition so that, for example, absolutely summable sequences can be safely convolved with any sequence that's simply bounded, as guaranteed by the BIBO stability theorem we saw in Section 6.2.1.

Convolution is linear and time-invariant, as expected from an operator that captures the inner workings of an LTI system:

$$\mathbf{x} * (\alpha\mathbf{y} + \beta\mathbf{w}) = \alpha(\mathbf{x} * \mathbf{y}) + \beta(\mathbf{x} * \mathbf{w}) \tag{6.14}$$

$$\mathbf{x} * (\mathcal{S}^k\mathbf{y}) = (\mathcal{S}^k\mathbf{x}) * \mathbf{y} = \mathcal{S}^k(\mathbf{x} * \mathbf{y}). \tag{6.15}$$

With a simple change of variable in (6.10) it is easy to prove commutativity

$$\mathbf{x} * \mathbf{y} = \mathbf{y} * \mathbf{x} \tag{6.16}$$

as well as associativity

$$(\mathbf{x} * \mathbf{h}) * \mathbf{w} = \mathbf{x} * (\mathbf{h} * \mathbf{w}) \tag{6.17}$$

but note that associativity only holds for absolutely summable sequences (see also Exercise **??**). If we think of $\mathbf{h}$ and $\mathbf{w}$ as the impulse responses of stable filters, Equation (6.17) has two important corollaries:

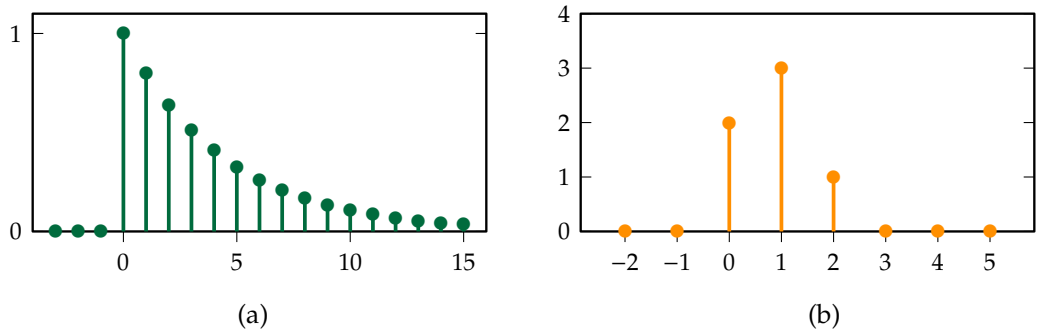

Figure 6.7: *Impulse response and finite-support signal used in Figures 6.8 and 6.9.*

1. a cascade of two (or more) stable filters is equivalent to a single filter whose impulse response is the convolution of the individual impulse responses
2. by applying the commutative property, we have that

   $$(\mathbf{x} * \mathbf{h}) * \mathbf{w} = \mathbf{x} * (\mathbf{h} * \mathbf{w}) = \mathbf{x} * (\mathbf{w} * \mathbf{h}) = (\mathbf{x} * \mathbf{w}) * \mathbf{h}$$

   meaning that in a cascade of two (or more) stable filters the order of the filters is irrelevant.

6.3.2 Convolution and Impulse Response

Informally speaking, the relationship between a filter and its impulse response is quite straightforward to visualize, in particular when the filter is causal[3]: starting from zero initial conditions, we "kick" the filter on using an input signal whose energy is fully concentrated in $n = 0$ and we record the resulting output. In discrete time, the maximally-compact signal is the well-defined delta sequence and, since every possible signal can be interpreted as a series of successive "kicks" with varying intensity, the output of an LTI system will always be a sum of delayed replicas of the impulse response, scaled by the intensity of each kick:

$$\mathcal{H}\mathbf{x} = \sum_{k=-\infty}^{\infty} x[k]\,\mathcal{S}^{-k}\mathbf{h}. \tag{6.18}$$

In this formulation, illustrated in Figure 6.8, the output signal is built up *globally*, that is, as the sum is evaluated incrementally, each successive term can affect the values of potentially all output samples.

In practical implementations, by contrast, filters are expected to work synchronously, that is, generate an output sample every time a new input sample arrives. If we restrict (6.18)

[3]In this section we will focus on how to interpret the impulse response and on the way it appears in the algorithm for convolution. In order to proceed as intuitively as possible, all examples will use strictly causal filters but, needless to say, the general ideas remain perfectly valid for noncausal systems as well.

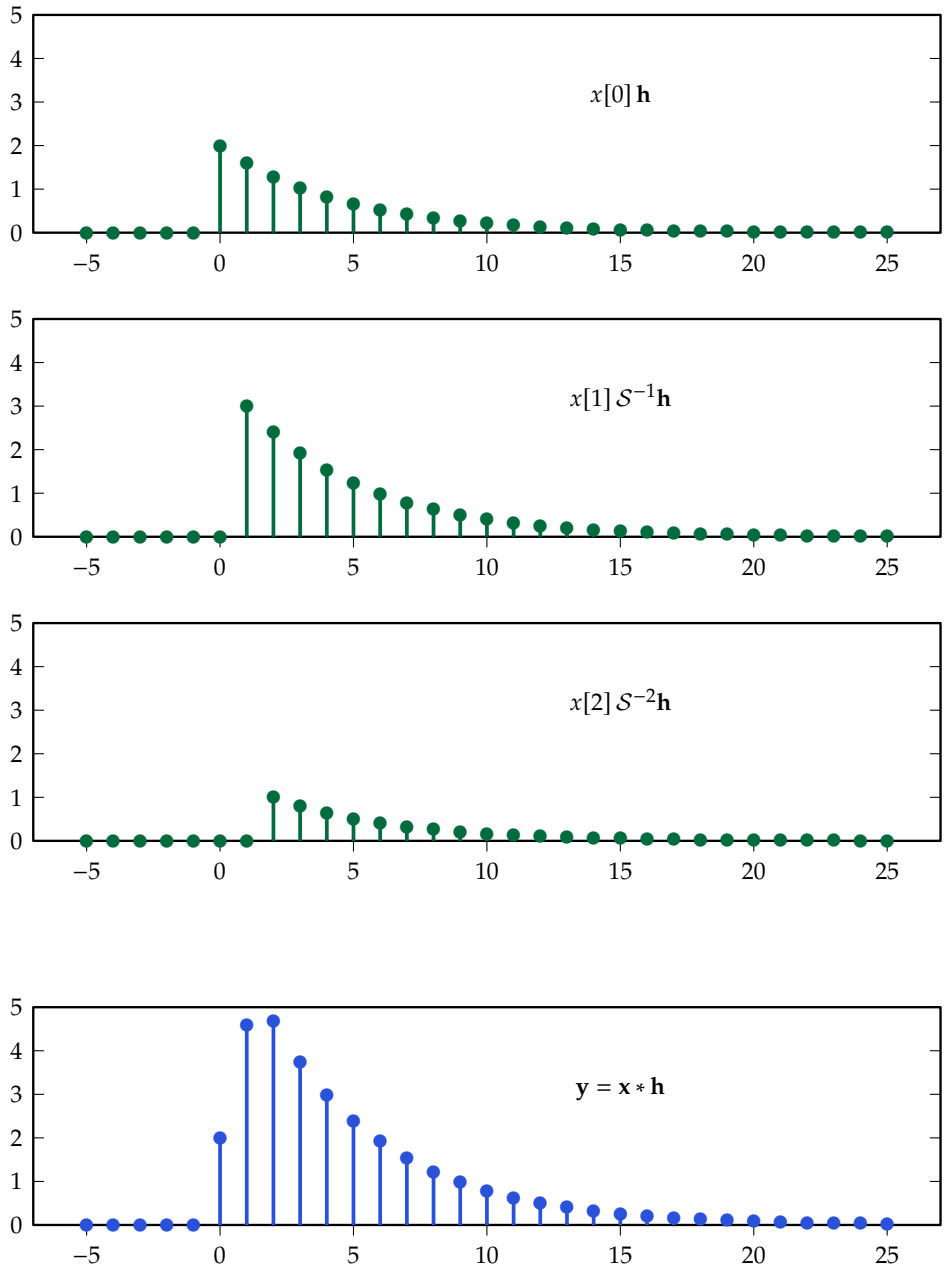

Figure 6.8: *Convolution as the sum of scaled and shifted copies of the impulse response.*

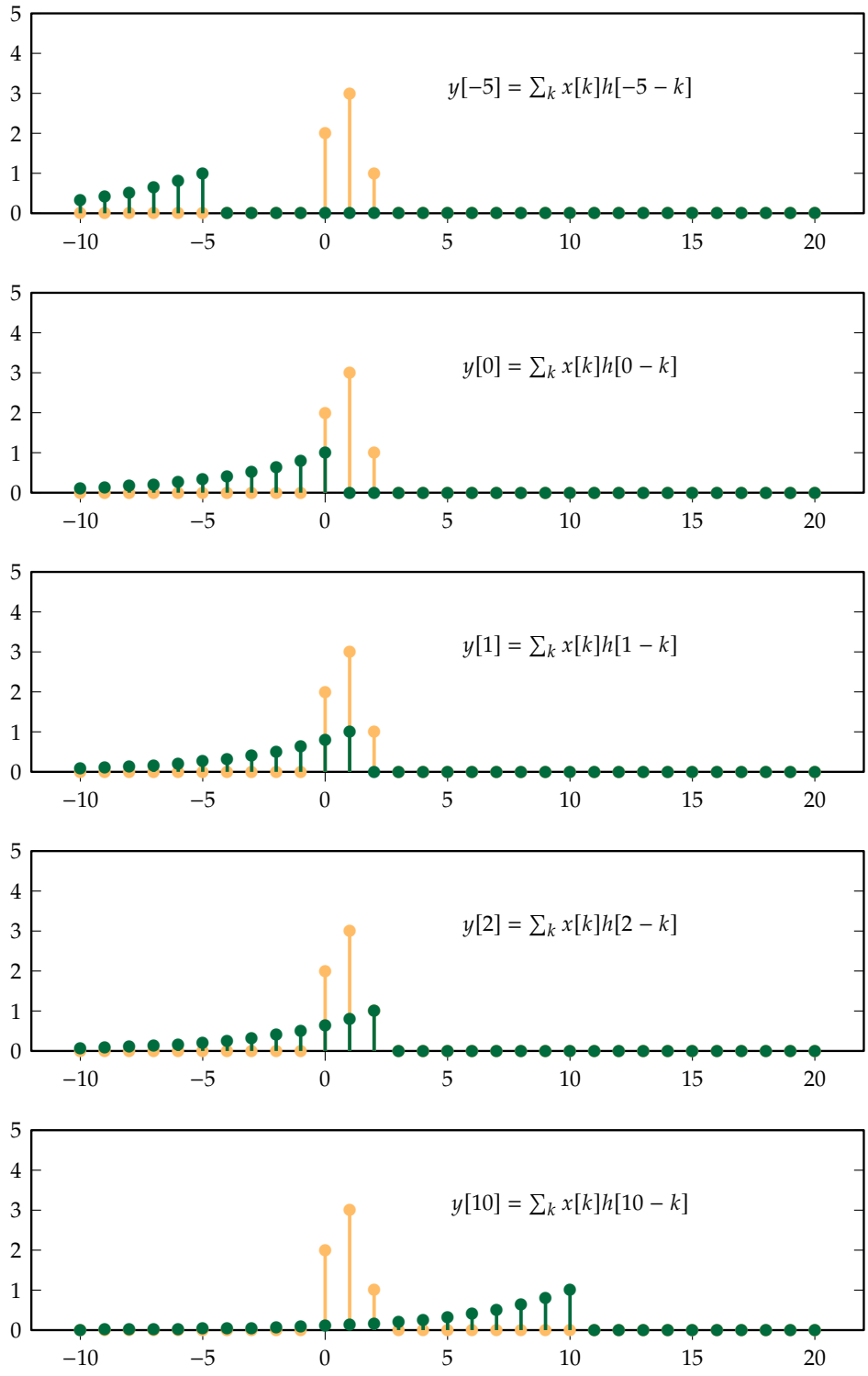

Figure 6.9: *Convolution: algorithmic computation of individual samples.*

to the computation of a single value, convolution reveals its algorithmic nature:

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

This expression represents the classic "recipe" for filtering via convolution and, explicitly, the steps to compute the filter's output at time n are:

- *time-reverse* the impulse response: $h[k] \to h[-k]$
- center it at index n: $h[-k] \to h[n-k]$
- multiply it element-wise with the input and sum all the products.

The recipe, illustrated graphically in Figure 6.9, is simple and easy to remember but the time reversal in the first step is often a bit puzzling and the source of some confusion. To understand why the impulse response is initially flipped, let's consider a simple filter with CCDE

$$y[n] = w_0 x[n] + w_1 x[n-1] + w_2 x[n-2];$$

each output sample is the linear combination of three past[4] input samples and therefore the filter is a causal FIR with impulse response

$$h[k] = \begin{cases} w_k & k = 0, 1, 2 \\ 0 & \text{otherwise.} \end{cases}$$

The internal mechanics of this system can be illustrated as in Figure 6.10 where each input term in the CCDE is linked to the output index via a connector labeled with the corresponding coefficient. To compute the value of the output at time n, we first translate this rigid arrangement of connectors so that the bottom arrow points to $y[n]$; then, for each connector in turn, we compute the product between the label and the associated input sample. As we do this, we can observe that, relative to the output index n,

- we move *backwards* over $\mathbf{x}$ in order to retrieve $x[n]$, $x[n-1]$, $x[n-2]$; but
- we simultaneously move *forward* over $\mathbf{h}$ in order to retrieve the associated coefficients: $h[0]$ for $x[n]$, $h[1]$ for $x[n-1]$, $h[2]$ for $x[n-2]$.

And that's the entire story: in the algorithm for convolution, we flip the impulse response simply because the CCDE "scans" $\mathbf{h}$ and $\mathbf{x}$ in opposite directions. In fact, since convolution is commutative, it makes no difference whether we flip one sequence or the other, and things would be the same if we time-reversed the input instead:

$$(\mathbf{h} * \mathbf{x})[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k] = \sum_{k=-\infty}^{\infty} h[k]x[n-k] = (\mathbf{x} * \mathbf{h})[n].$$

[4]Here and in the following we use "past" as a shorthand for "current and past" samples, that is, nothing from the future.

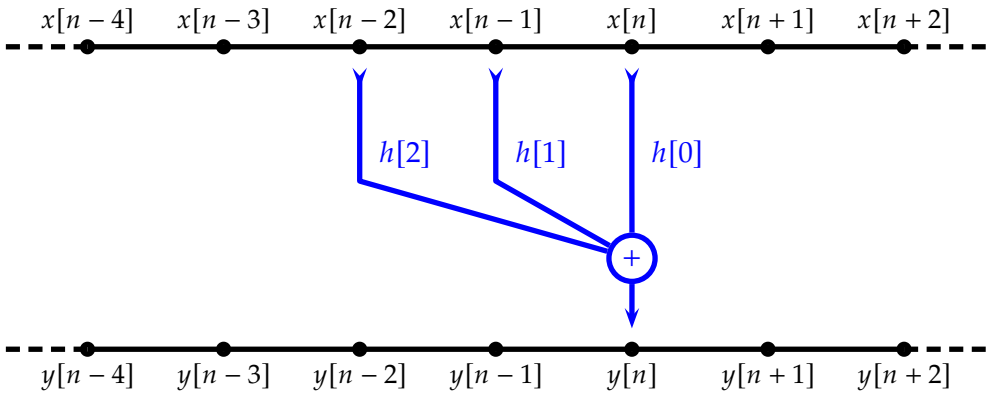

Figure 6.10: *Graphical representation of the CCDE for a short causal FIR.*

There is nevertheless a good reason behind the standard formulation: with causal systems, a plot of the *time-reversed* impulse response provides an intuitive visual representation of a filter's "memory effects," that is, of the way in which past input values affect each output sample. With FIRs, for instance, such a plot would show that past inputs are relevant only up to a finite delay; with IIRs, on the other hand, the plot will never be identically zero no matter how far in the past we go, showing the long-range dependencies of recursive filters.

Now that we have developed an intuitive interpretation for its time-reversed version, let's return to the impulse response in normal form and in particular to equation 6.18, where it appears (with all its possible shifts) in the expression for a filter's output. If we isolate from the sum the term for $k = 0$ we can write:

$$\mathbf{y} = x[0]\mathbf{h} + \mathbf{r}$$

where $\mathbf{r}$ is a sequence that does *not* depend on $x[0]$. Looking at the individual samples it is

$$y[n] = h[n]x[0] + r[n]$$

namely, $h[n]$ shows how the input at time 0 impacts the output at time n. Because of time invariance, we can say in more general terms that the n-th sample of the impulse response measures how an input value still affects the output n samples later. This makes a lot of intuitive sense when paired with our earlier image of "kicking" an impulse response out of a causal filter: for an arbitrary input signal, each sample is an individual kick whose ripples propagate *forward* in time, leaving a trace in future values the output. Compared to the time-reversed case, a plot of the normal impulse response provides the complementary representation of a filter's memory effects; in FIRs the range of influence for each input sample is always finite, while in IIRs it extends to the end of time[5].

[5]In practice, it extends until the value of $x[n]h[n+k]$ becomes too small for the finite numerical precision

6.3.3 Convolution and Inner Product.

In $\ell_2(\mathbb{Z})$, convolution and inner product are both defined as a sum of products:

$$(\mathbf{h} * \mathbf{x})[n] = \sum_{k=-\infty}^{\infty} h[n-k]x[k]$$
$$\langle \mathbf{h}, \mathbf{x} \rangle = \sum_{k=-\infty}^{\infty} h^*[k]x[k];$$

because of this similarity, we can easily relate the two as

$$(\mathbf{h} * \mathbf{x})[n] = \langle \mathcal{S}^{-n} \mathcal{R} \mathbf{h}^*, \mathbf{x} \rangle \tag{6.19}$$

and see each value of the convolution as an inner product featuring a conjugated, time-reversed, and appropriately time-shifted version of the first sequence.

The connection between inner product and convolution can be used to define how the latter operates on other types of signals. In the space of N-periodic sequences, for instance, convolution is computed by taking the sum of products over just one period, and this is consistent with the definition of inner product in $\tilde{\mathbb{C}}^N$ in (**??**):

$$(\tilde{\mathbf{h}} * \tilde{\mathbf{x}})[n] = \sum_{k=0}^{N-1} \tilde{x}[k]\tilde{y}[n-k] \tag{6.20}$$
$$= \langle \mathcal{S}^{-n} \mathcal{R} \tilde{\mathbf{h}}^*, \tilde{\mathbf{x}} \rangle \tag{6.21}$$

The same definition can be applied to length-N sequences if all shifts are considered circular; this provides a method to filter finite-length signals in which border effects are implicitly handled by the circular convolution.

From the inner product definition in $L_2([-\pi, \pi])$ we can also derive the convolution formula for DTFTs, which will be useful later in the context of signal modulation. As stated in (**??**)

$$\langle \mathbf{X}, \mathbf{Y} \rangle = \frac{1}{2\pi} \int_{-\pi}^{\pi} X^*(\sigma)Y(\sigma)d\sigma;$$

if we extend the reversal and shift operators to DTFTs like so

$$(\mathcal{R}\mathbf{Y})(\sigma) = Y(-\sigma)$$
$$(\mathcal{S}^{-\omega}\mathcal{R}\mathbf{Y})(\sigma) = Y(\omega - \sigma)$$

then by analogy we can write

$$(\mathbf{X} * \mathbf{Y})(\omega) = \langle \mathbf{X}^*, \mathcal{S}^{-\omega}\mathcal{R}\mathbf{Y} \rangle$$
$$= \frac{1}{2\pi} \int_{-\pi}^{\pi} X(\sigma)Y(\omega - \sigma)d\sigma. \tag{6.22}$$

of the underlying signal processing device — which always happen at some point if the filter is stable because, to have stability, the impulse response must decay to zero.

6.4 The Frequency Response

The impulse response provides us with a time-domain view of a filter's characteristics; while essential, this perspective is only half of the story and we need to move to the frequency domain to truly appreciate the processing power of LTI systems. To do so, we will first prove the following fundamental result: complex exponential sequences are *eigensequences*[6] of linear time-invariant systems.

Consider a stable linear time-invariant system $\mathcal{H}$ whose input is a complex exponential sequence $w[n] = e^{j\omega_0 n}$; a delay of k samples operates on $\mathbf{w}$ as

$$\mathcal{S}^{-k}\mathbf{w} = e^{-j\omega_0 k}\,\mathbf{w}$$

and so we can use (6.9) to write

$$\begin{aligned}\mathcal{H}\mathbf{w} &= \mathbf{h} * \mathbf{w} \\ &= \sum_{k=-\infty}^{\infty} h[k]\,\mathcal{S}^{-k}\mathbf{w} \\ &= \mathbf{w} \sum_{k=-\infty}^{\infty} h[k] e^{-j\omega_0 k} \\ &= H(\omega_0)\,\mathbf{w}. \end{aligned} \tag{6.23}$$

$H(\omega)$ is the DTFT of the impulse response and it is called the *frequency response* of the filter. This result states that a complex exponential at frequency ω_0 will exit a filter as a scaled version of itself and that the scaling factor is the frequency response of the filter computed at ω_0. If we express the scaling factor in polar form as $H(\omega_0) = A_0\, e^{j\theta_0}$, we can write

$$(\mathcal{H}\mathbf{w})[n] = A_0\, e^{j(\omega_0 n + \theta_0)} \tag{6.24}$$

and we can see that the output oscillation is modified in amplitude by $A_0 = \left|H(\omega_0)\right|$ and it is shifted in phase by $\theta_0 = \measuredangle H(\omega_0)$; yet there is no change in frequency and so we can state that *LTI system cannot change the frequency of a sinusoidal input.* Note that the stability assumption for $\mathcal{H}$ can be relaxed to requiring simply that the impulse response is square summable, in order to guarantee the existence of $H(\omega)$.

6.4.1 The Convolution Theorem

A straighforward extension of (6.23) is that *the DTFT of the convolution of two sequences is the product of their DTFTs.* Indeed, consider two sequences $\mathbf{x}$ and $\mathbf{h}$, both absolutely

[6] In continuous time, complex exponential functions are *eigenfunctions* of LTI systems. In discrete time we use the slightly less standard term "eigensequences" to indicate an input signal whose shape is not fundamentally changed by a filtering operation.

summable, and their convolution $\mathbf{y} = \mathbf{x} * \mathbf{h}$; in the expression for the DTFT of $\mathbf{y}$

$$Y(\omega) = \sum_{n=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} x[k]h[n-k]\, e^{-j\omega n}$$

we can safely interchange the order of summation because of absolute summability and, by splitting the complex exponential, we obtain

$$Y(\omega) = \sum_{k=-\infty}^{\infty} x[k]\, e^{-j\omega k} \sum_{n=-\infty}^{\infty} h[n-k]\, e^{-j\omega(n-k)}$$

from which, after a change of variable, we obtain the so-called *convolution theorem:*

$$Y(\omega) = X(\omega)H(\omega) \tag{6.25}$$

The theorem gives us the means to analyze the effect of a filter in the frequency domain, since its effect is that of a multiplicative mask applied to the spectrum of the input signal. Just as the impulse response is a complete description of a filter in the discrete-time domain, the frequency response completely characterizes the filter in the frequency domain and, indeed, the properties of LTI systems are almost always described primarily in terms of the magnitude and phase of their frequency response.

6.4.2 The Magnitude Response

The most powerful intuition arising from the convolution theorem is obtained by considering the magnitudes of the spectra involved in a filtering operation. Recall that the square magnitude of the DTFT represents a signal's energy distribution in frequency; by appropriately "shaping" the magnitude of a filter's frequency response we can therefore amplify or attenuate specific portions of the frequency content of the input. We can for instance boost the lower frequencies in an audio signal to compensate for the acoustic properties of a room; or limit the spectral support of a transmitted signal to match it to a specific transmission channel. Since the magnitude response acts in a multiplicative manner, if an input signal does not contain energy at a given frequency ω_0, a stable LTI filter will not be able to create spectral content at that frequency; in other words, LTI systems cannot create new frequency content (see also Exercise **??**).

According to the way that it affects the input, a filter can be placed into one of four broad categories[7]:

- **Lowpass filters**, for which the magnitude response is concentrated around $\omega = 0$; these filters preserve the low-frequency energy of the input signals and attenuate or eliminate the high-frequency components.

[7] Unless explicitly stated otherwise, we always assume that the impulse response of the filter is real-valued and therefore the frequency response is symmetric in magnitude. Furthermore, although by definition all frequency responses of discrete-time filters are 2π-periodic functions, in the interest of conciseness will usually describe only the $[-\pi, \pi]$ interval and tacitly assume 2π-periodicity.

- **Highpass filters**, for which the magnitude response is concentrated around $\omega = \pm\pi$; these filters preserve the high-frequency energy of the input signals and attenuate or eliminate the low-frequency components.
- **Bandpass filters**, for which the magnitude response is concentrated around $\omega = \pm\omega_c$ with $0 < \omega_c < \pi$; these filters preserve the energy of the input signals around the frequency ω_c and attenuate the signals everywhere else, notably around $\omega = 0$ and $\omega = \pm\pi$.
- **Allpass filters**, for which the magnitude response is a *constant* over the entire $[-\pi, \pi]$ interval. These filters only affect the phase of the input; a typical example is a filter that only introduces a delay).

The frequency interval for which the magnitude response is close to zero is called the *stopband*; conversely, the frequency interval for which the magnitude response is larger than one is called the *passband*.

6.4.3 The Phase Response

The phase response of a filter affects the shape of the output signal. If we think of the input as the sum of an infinite number of complex exponentials (as in the inverse DTFT formula) we can see from (6.24) that the filter adds a different phase offset to each component as a function of its phase response at the frequency of the component. A phase offset in a complex exponential corresponds to a delay in the time domain and therefore, at the output of the filter, the original frequency components of the input will each be delayed by a varying amount. We have seen in Section **??** how phase alignment determines the *shape* of a signal in the time domain; the phase response of a filter, therefore, can either preserve the original shape or completely scramble it.

Linear Phase. In a linear-phase filter the phase response is proportional to the frequency:

$$\measuredangle H(\omega) = e^{-j\omega d} \qquad d \in \mathbb{R}. \tag{6.26}$$

An elementary example of a linear-phase system is a simple delay by m samples:

$$\mathbf{y} = \mathcal{S}^{-m}\mathbf{x};$$

the impulse response of this system is clearly $\mathbf{h} = \boldsymbol{\delta}_m$ and the frequency response is

$$H(\omega) = e^{-j\omega m}$$

which coincides with the phase response. A delay obviously preserves the phase of its input and this property applies to all strictly linear-phase systems, that is, systems for which the frequency response can be expressed as

$$H(\omega) = \left|H(\omega)\right| e^{-j\omega d}.$$

The expression describes the filter response as a cascade of two subsystems: a real-valued zero-phase filter, which is necessarily noncausal and affects only the spectral magnitude of the input, followed by a delay. We can distinguish two cases:

- when $d \in \mathbb{N}$, the delay spans an integer number of samples;
- when $d \notin \mathbb{N}$, we are in the presence of a so-called *fractional delay*, which implies a subsample shift between input and output. We will study the nature of fractional delays more in detail in Section **??**.

A filter is said to have *generalized linear phase* if its frequency response can be expressed as

$$H(\omega) = A(\omega)e^{-j(\omega d+\phi)}, \quad A(\omega) \in \mathbb{R}. \tag{6.27}$$

In this case, besides the additional constant phase shift ϕ, possible changes of sign in $A(\omega)$ will cause a jump of π in the phase response; nevertheless, generalized linear phase systems possess many of the same properties as in the case of strictly linear phase.

Group Delay. As we will see in Chapter **??**, in practice the only realizable filters with linear phase must be FIR; in all other cases (and, in particular, for all realizable IIR filters) the phase will be nonlinear. It is often important to study how much the response deviates from linearity; for instance, if the phase is approximately linear in the passband, we can safely ignore an otherwise highly nonlinear response in the stopband.

A filter's *group delay* is defined as the negative of the first derivative of the phase response:

$$\mathrm{grd}\{H(\omega)\} = -\frac{d\measuredangle H(\omega)}{d\omega}. \tag{6.28}$$

The idea is that we can use this quantity to approximate the phase response around any given frequency using a first-order Taylor approximation. Define $\varphi(\omega) = \measuredangle H(\omega)$ and approximate $\varphi(\omega)$ around ω_0 as $\varphi(\omega_0 + \tau) = \varphi(\omega_0) + \tau\varphi'(\omega_0)$; we can write

$$\begin{aligned} H(\omega_0 + \tau) &= |H(\omega_0 + \tau)|\ e^{j\varphi(\omega_0+\tau)} \\ &\approx \Big[|H(\omega_0 + \tau)|\ e^{j\varphi(\omega_0)}\Big]\ e^{j\varphi'(\omega_0)\tau} \end{aligned} \tag{6.29}$$

so that the phase response of the filter is approximately linear for at a *group* of frequencies around a given ω_0. The delay for this group of frequencies is the negative of the derivative of the phase, hence the definition of group delay, whose unit of measures is samples. For linear-phase systems, the group delay is obviously a constant.

6.4.4 Key Examples Revisited in the Frequency Domain

Let's now examine the frequency-domain characteristics of the filters we introduced in Section 6.1.1.

Moving Average. The frequency response of the moving average filter is

$$H(\omega) = \frac{1}{N}\,\frac{\sin(\omega N/2)}{\sin(\omega/2)}\,e^{-j\frac{N-1}{2}\omega} = A(\omega)\,e^{-j\frac{N-1}{2}\omega} \tag{6.30}$$

as we already computed in (**??**). The expression shows the product of a real-valued term $A(\omega)$ times a pure phase factor so that magnitude and phase responses are immediate to determine. The former is plotted in Figure 6.11 for varying values of M; from its shape we can deduce that the filter is a lowpass, with a passband that shrinks as the support of the finite-length impulse response becomes larger. Note how the magnitude response of a Moving Average of length M has $M-1$ zero crossings that take place at all multiples of $2\pi/M$ except in 0.

Looking at the phase response, it is clear from (6.27) that the filter is generalized linear-phase with an effective delay of $(N-1)/2$ samples, as informally discussed in Section 6.2.2. Figure 6.11 shows both the magnitude response and the term $A(\omega)$ in the top panel, and the corresponding phase response in the bottom panel for $M = 8$. Note how each change in sign for $A(\omega)$ causes the phase to jump by π.

Leaky Integrator The impulse response of the Leaky Integrator is a one-sided exponentially decreasing sequence and therefore its frequency response is, as we showed in (**??**),

$$H(\omega) = \frac{1-\lambda}{1-\lambda e^{-j\omega}} \tag{6.31}$$

After a little complex algebra, magnitude and phase turn out to be

$$\big|H(\omega)\big|^2 = \frac{(1-\lambda)^2}{1+\lambda^2-2\lambda\cos(\omega)} \tag{6.32}$$

$$\tag{6.33}$$

$$\measuredangle H(\omega) = \arctan\left[-\frac{\lambda\sin(\omega)}{1-\lambda\cos(\omega)}\right]; \tag{6.34}$$

the group delay, also plotted in Figure **??**, is obtained by differentiating the phase response:

$$\mathrm{grd}\big\{H(\omega)\big\} = \frac{\lambda\cos(\omega)-\lambda^2}{1+\lambda^2-2\lambda\cos(\omega)} \tag{6.35}$$

The magnitude response is plotted in Figure 6.13 for a couple of values of λ; we can see that the Leaky Integrator is also a lowpass filter and, as shown in the denoising examples at the beginning of this chapter, its bandwidth decreases as λ gets closer to unity. The phase response and group delay are shown in Figure 6.14; we can see from the group delay that the delay for the frequencies in the passband increases with the smoothing power of the filter.

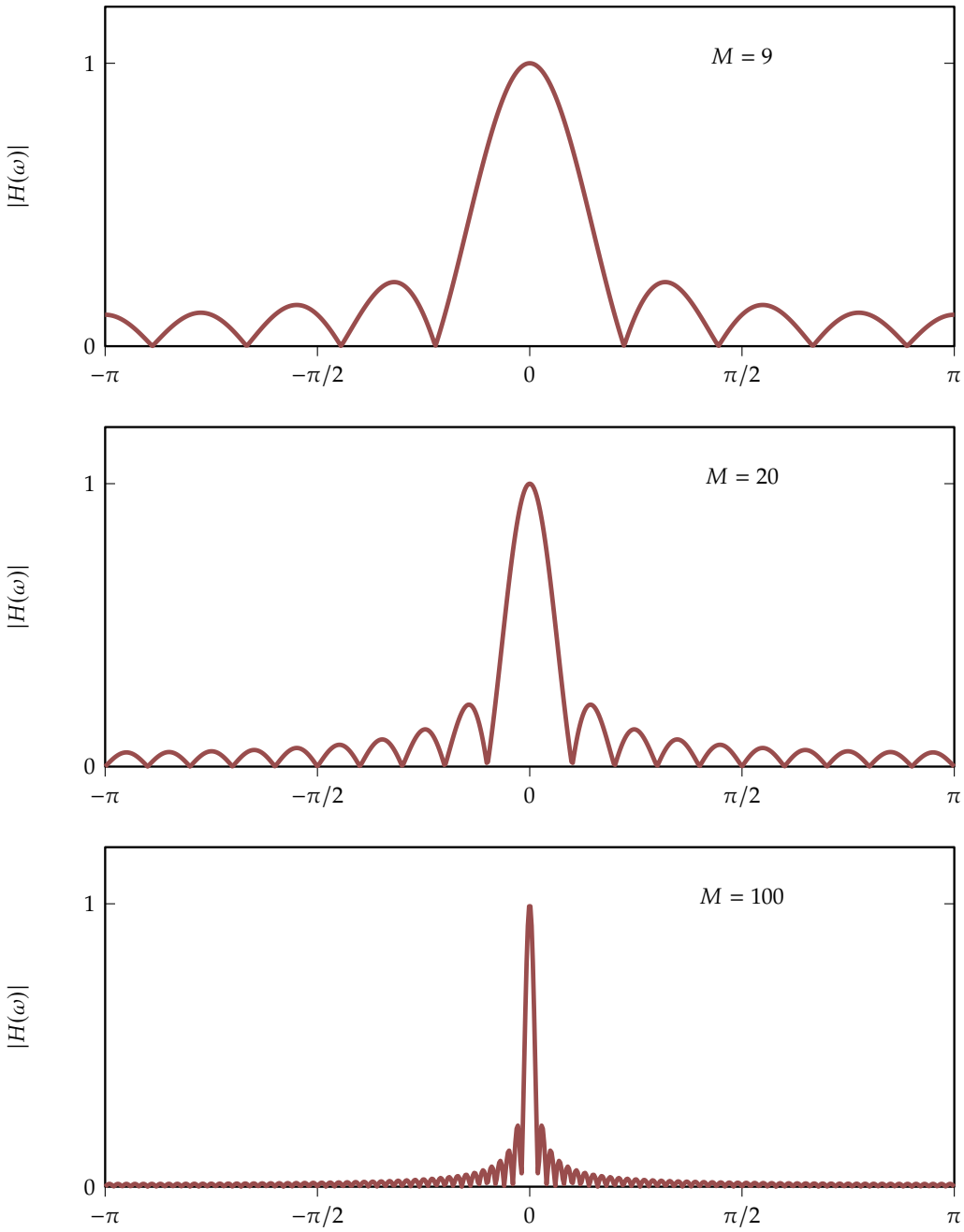

Figure 6.11: *Magnitude response of the Moving Average filter for different values of M.*

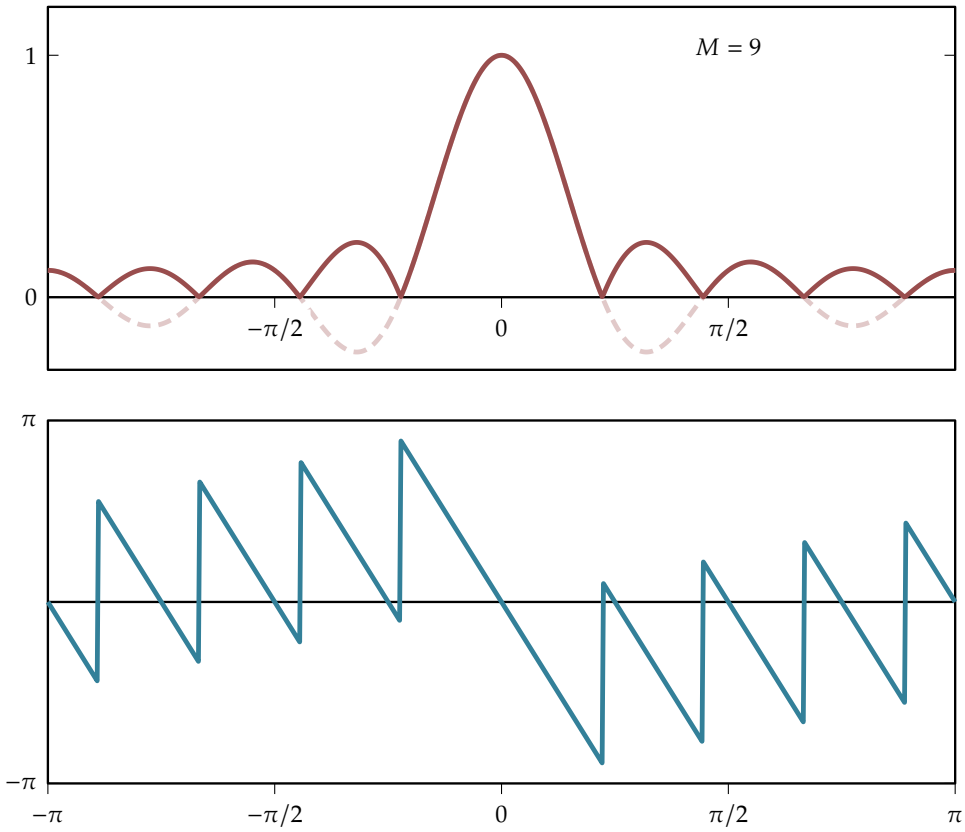

Figure 6.12: *Magnitude and phase response of the Moving Average.*

Denoising in the frequency domain: the Signal-to-Noise Ratio. The Moving Average and Leaky Integrator were introduced in Section 6.1.1 to solve a *denoising* problem: we had a slow-varying signal to which a fast-varying noisy perturbation had been added, and we performed some sort of averaging in the time domain to reduce the effect of the additive noise. Now that we know that both filters are lowpass, how can we interpret this denoising ability from a frequency-domain perspective?

A precise mathematical model for additive noise will be discussed in Chapter **??**, within the framework of stochastic signal processing; all details aside, however, the gist of the model is that a fast-varying random signal has a spectrum that is also random-looking and that contains energy at all frequencies, as shown in Figure 6.15. Since the clean slow-varying signal contains most of its energy in the low-frequency region of the spectrum, denoising via lowpass filters works by reducing the energy of the noise outside of the frequency support of the clean data. While we won't be able to eliminate the additive noise within the baseband portion that contains most of the signal's energy, by removing all the out-of-band noise we can improve the overall *signal to noise ratio* (SNR). Assume that the original signal $\mathbf{s}$ has a spectrum $S(\omega)$ that is zero outside of the $[-\omega_c, \omega_c]$ interval,

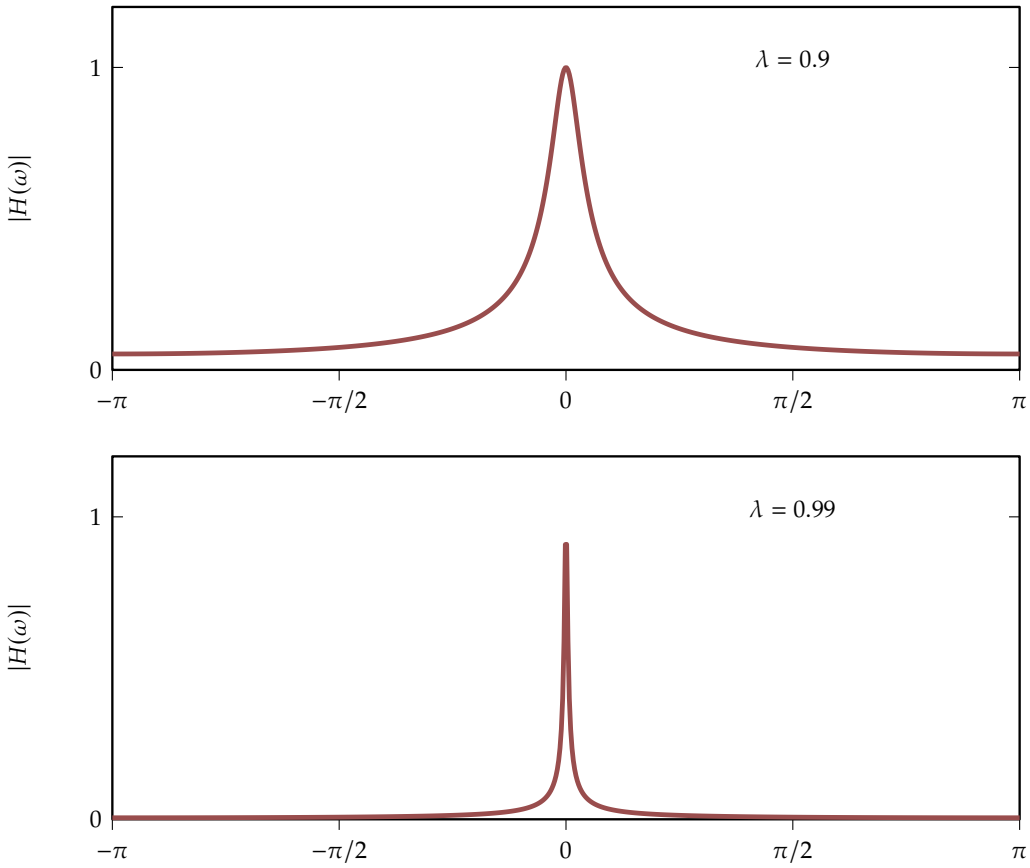

Figure 6.13: *Magnitude response of the Leaky Integrator for different values of λ.*

whereas the noise **n** has a spectrum $N(\omega)$ with full spectral support, as shown in the left panel of Figure 6.16. Remembering (**??**), the expression for the energy of a signal in the frequency domain, the original SNR is[8]

$$\text{SNR}_{\text{orig}} = 10 \log_{10} \frac{\int_{-\omega_c}^{\omega_c} |S(\omega)|^2 \, d\omega}{\int_{-\pi}^{\pi} |N(\omega)|^2 \, d\omega}.$$

Assume now we filter **s** + **n** via a lowpass whose magnitude response is sketched with a dashed line in the right panel of Figure 6.16; the filter will remove all frequency components outside of the support $[-\omega_c, \omega_c]$ of the original signal and the resulting SNR will be

$$\text{SNR}_{\text{denoised}} = 10 \log_{10} \frac{\int_{-\omega_c}^{\omega_c} |S(\omega)|^2 \, d\omega}{\int_{-\omega_c}^{\omega_c} |N(\omega)|^2 \, d\omega} \geq \text{SNR}_{\text{orig}}$$

[8]The SNR is the ratio between the energy of the signal and the energy of the noise. In general this dimensionless quantity is expressed in decibels (dB) on a logarithmic scale.

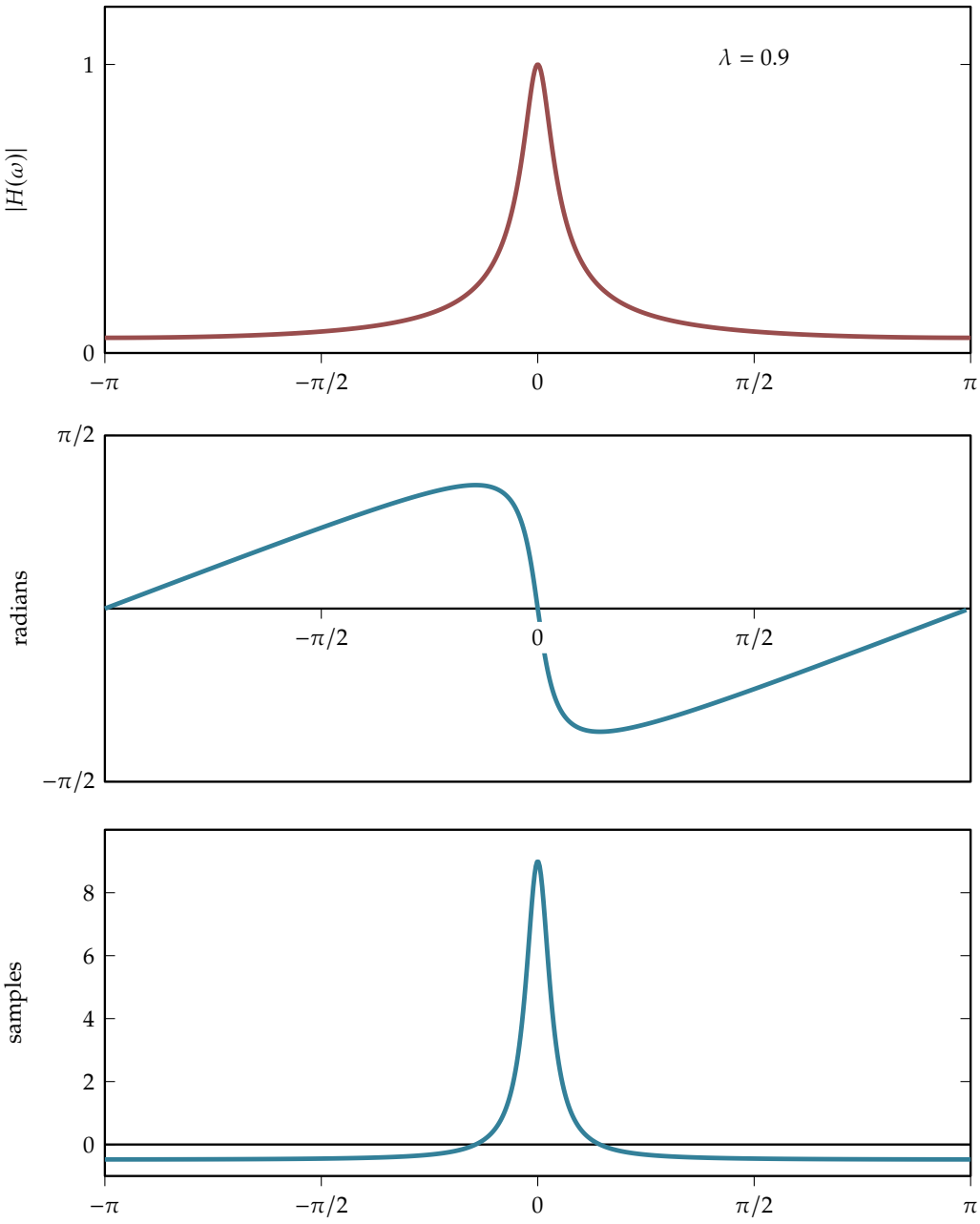

Figure 6.14: *Magnitude, phase response, and group delay of the Leaky Integrator.*

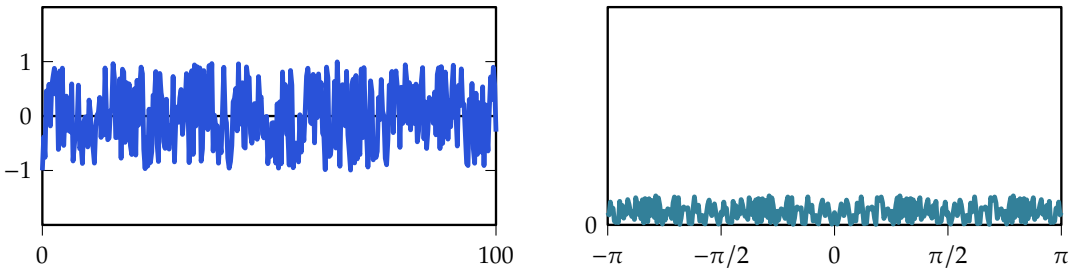

Figure 6.15: *Random noise and its magnitude spectrum.*

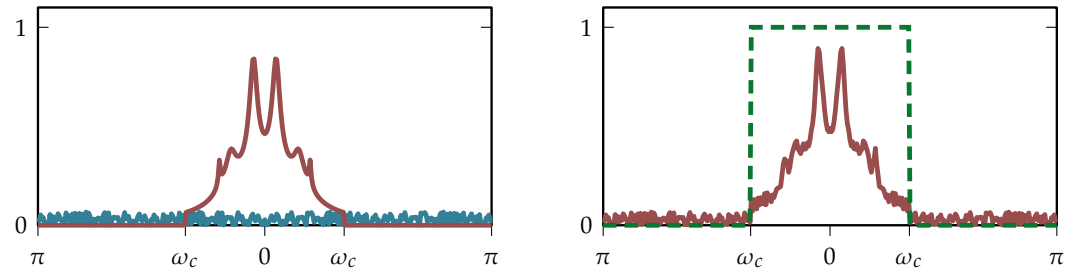

Figure 6.16: *Spectra of baseband signal and of noise (left panel); spectrum of noisy signal and outline (dashed line) of a denoising lowpass filter.*

This simple type of denoising is suitable to the case in which the clean data is baseband while the noise is wideband, and this is generally the case for audio and speech signals. More sophisticated denoising techniques would try to incorporate as many assumptions as possible on the spectral properties of the desired signal.

6.5 Ideal Filters

The characterization of filters in the frequency domain immediately leads to the question of what would the "best" filter be for each category; for instance, what is the "best" lowpass filter that we can think of? The answer leads to the world of *ideal filters*, which owe their name to the fact that, while possessing highly desirable properties, they cannot be exactly implemented in practice. Before tackling the problem of approximating ideal filters in applications, let us review the most common examples.

Ideal Lowpass. The ideal lowpass is a zero-phase filter that completely eliminates all frequency content outside of a passband $-\omega_p < \omega < \omega_p$, while leaving all frequencies within the passband untouched. Its frequency response over $[-\pi, \pi]$ can therefore be

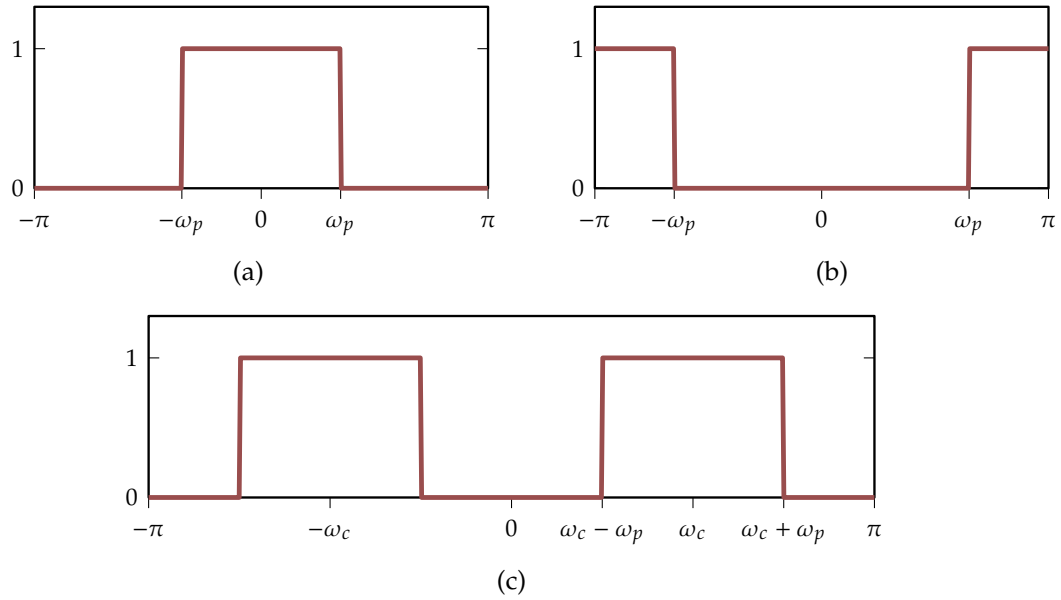

Figure 6.17: *Real-valued frequency responses for an ideal lowpass (a), ideal highpass (b), ideal bandpass (c).*

expressed as[9]

$$H_L(\omega) = \begin{cases} 1 & |\omega| \leq \omega_p \\ 0 & \omega_p < |\omega| \leq \pi \end{cases} \tag{6.36}$$

an example of which is shown in Figure 6.17-(a). The impulse response can be easily determined via the inverse DTFT as

$$h_L[n] = \frac{1}{2\pi} \int_{-\omega_p}^{\omega_p} e^{j\omega n} d\omega = \frac{\sin(\omega_p n)}{\pi n} \tag{6.37}$$

The impulse response is therefore an infinite-support, two-sided sequence and, as we discussed in Section 6.2.1, this implies that the filter cannot be implemented in practice. Nevertheless, the ideal lowpass and its associated DTFT pair are so important as a theoretical paradigm that two special function names are commonly used; from the definitions

$$\text{rect}(x) = \begin{cases} 1 & |x| \leq 1/2 \\ 0 & |x| > 1/2 \end{cases} \tag{6.38}$$

(6.39)

$$\text{sinc}(x) = \begin{cases} \dfrac{\sin(\pi x)}{\pi x} & x \neq 0 \\ 1 & x = 0 \end{cases} \tag{6.40}$$

[9]See also footnote on page 23 concerning the assumed 2π periodicity of the frequency response.

we can write

$$H_L(\omega) = \text{rect}\left(\frac{\omega}{2\omega_p}\right) \tag{6.41}$$

(obviously 2π-periodized over all $\mathbb{R}$) and

$$h_L[n] = \frac{\omega_p}{\pi}\,\text{sinc}\left(\frac{\omega_p}{\pi}\,n\right) \tag{6.42}$$

The DTFT pair:

$$\frac{\omega_p}{\pi}\,\text{sinc}\left(\frac{\omega_p}{\pi}\,n\right) \overset{\text{DTFT}}{\longleftrightarrow} \text{rect}\left(\frac{\omega}{2\omega_p}\right) \tag{6.43}$$

constitutes one of the fundamental relationships of digital signal processing. Note that as $\omega_p \to \pi$, we re-obtain the well-known DTFT pair $\boldsymbol{\delta} \leftrightarrow \mathbf{1}$, while as $\omega_p \to 0$ we can re-normalize by (π/ω_p) to obtain $\mathbf{1} \leftrightarrow \tilde{\delta}(\omega)$.

The frequency ω_p is also called the *cutoff* frequency of the lowpass filter; the impulse response of an ideal lowpass with cutoff $\omega_p = \pi/3$ is shown in Figure 6.18. From the definition, the sinc function is zero for all integer values of its argument except zero so, in this case, every third sample is equal to zero.

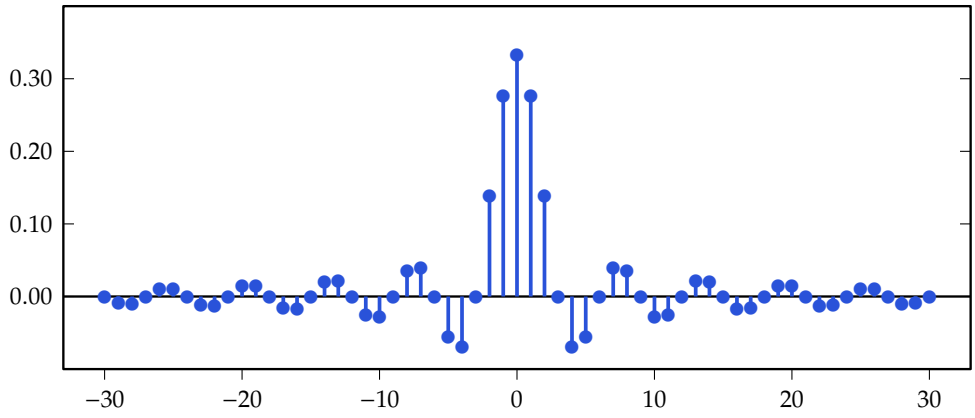

Figure 6.18: *Impulse response of an ideal lowpass with $\omega_p = \pi/3$.*

Ideal Highpass. An ideal highpass filter is the complementary filter to an ideal lowpass, in the sense that it eliminates all frequency content below a cutoff frequency ω_p; its (positive) passband extends therefore from ω_p to π. Its frequency response is

$$H_H(\omega) = \begin{cases} 0 & |\omega| \le \omega_p \\ 1 & \omega_c < |\omega| \le \pi \end{cases} \tag{6.44}$$

an example of which is shown in Figure 6.17-(b). From the relation $H_H(\omega) = 1 - \text{rect}(\omega/2\omega_p)$ the impulse response is easily obtained as

$$h_H[n] = \delta[n] - \frac{\omega_p}{\pi}\,\text{sinc}\left(\frac{\omega_p}{\pi}\,n\right)$$

Ideal Bandpass. The frequency response of an ideal bandpass filter with center frequency ω_c and bandwidth $2\omega_p$, with $\omega_p < \omega_c$, is

$$H_B(\omega) = \begin{cases} 1 & |\omega \pm \omega_c| \le \omega_p \\ 0 & \text{elsewhere} \end{cases} \tag{6.45}$$

an example of which is shown in Figure 6.17-(c). It is left as an exercise to prove that the impulse response is

$$h_B[n] = 2\cos(\omega_c n)\,\frac{\omega_p}{\pi}\,\text{sinc}\left(\frac{\omega_p}{\pi}\,n\right) \tag{6.46}$$

Hilbert Filter. The frequency response of the Hilbert filter is

$$H_F(\omega) = \begin{cases} -j & 0 \le \omega < \pi \\ +j & -\pi \le \omega < 0. \end{cases} \tag{6.47}$$

Its impulse response can be computed via an inverse DTFT as

$$h_F[n] = \begin{cases} \dfrac{2}{n\pi} & n \text{ odd} \\ 0 & n \text{ even} \end{cases} \tag{6.48}$$

and it is shown in Figure 6.19. Since $\left|H_F(\omega)\right| = 1$, the Hilbert filter is an allpass. Its phase response, on the other hand, is designed to introduce a phase shift such that, if $s[n] = sin(\omega_0 n)$ and $c[n] = \cos(\omega_0 n)$,

$$\mathbf{h_F} * \mathbf{c} = -\mathbf{s}$$
$$\mathbf{h_F} * \mathbf{s} = -\mathbf{c};$$

this can be quickly verified using the generalized DTFTs for sine and cosine, (**??**) and (**??**). More generally, the Hilbert filter is a basic building block for data demodulation, as we will see in Section **??**.

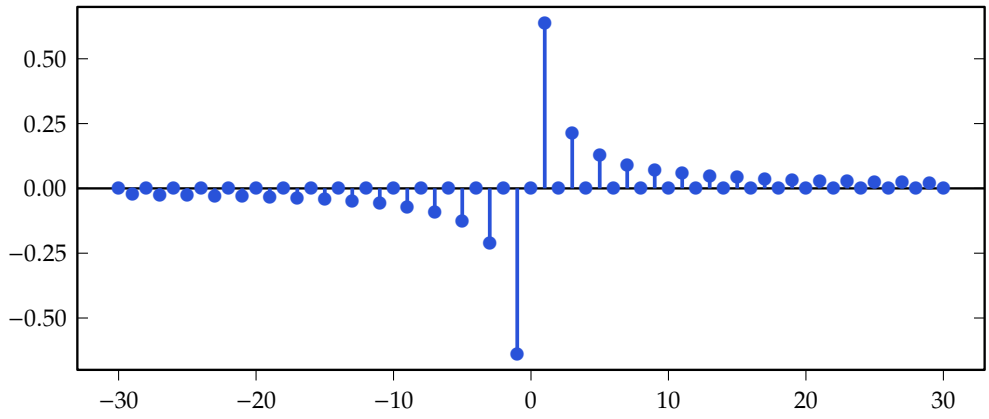

Figure 6.19: *Impulse response of the Hilbert filter.*

6.6 Realizable Filters

A filter is said to be *realizable* if, given a causal input sequence, each output sample can be computed in a finite amount of time using a finite amount of memory. An LTI system is completely characterized by its impulse response **h** and, once **h** is known, the output of the system for a given input **x** can be computed at least in theory via the convolution sum (**??**). In the case of FIR filters, the summation in (**??**) always involves only a finite number of terms: this means that each output sample can be computed performing a finite number of multiplications and additions over a finite number of past samples; an FIR filter, therefore, is always realizable. At the other extreme, the ideal filters we saw in the previous section are all non-realizable since their infinite, two-sided impulse responses require knowledge of an infinite number of future input samples. In between, we have IIR filters with a one-sided impulse response, namely, causal IIRs. The convolution between a causal IIR and a causal input also involves a finite number of operations but, in the general case, the number of terms in the sum grows linearly with the output index n, and this does not fulfill our definition of realizability. We have seen however that if an LTI processing operation can be described algorithmically (such as in the case of the Leaky Integrator), then we can have a realizable filter with an infinite-support impulse response; we are therefore interested in a general description of realizable IIR discrete-time filters.

6.6.1 Constant-Coefficient Difference Equations

Consider the problem of defining a universal algorithmic framework for realizable LTI systems, that is, finding a set of building blocks that can be interconnected to implement a machine that takes one input sample at a time and produces a corresponding output sample. Because of our requirements, the resulting set turns out to be relatively limited:

- linearity in the input-output relationship implies that we can only use linear opera-

tions, i.e. sums and multiplications by scalars;

- time invariance implies that the scalars must be constants;
- realizability requires that we only use a finite number of adders and multipliers and a finite number of memory cells.

With these ingredients, the general mathematical relationship between input and output is a so-called *constant-coefficient difference equation* (CCDE) of the form

$$\sum_{k=0}^{N-1} a_k y[n-k] = \sum_{k=0}^{M-1} b_k x[n-k]. \tag{6.49}$$

We can assume $a_0 = 1$ (or, otherwise, renormalize by a_0) and so we can rearrange the CCDE in a more standard, causal algorithmic form:

$$y[n] = \sum_{k=0}^{M-1} b_k x[n-k] - \sum_{k=1}^{N-1} a_k y[n-k]; \tag{6.50}$$

the expression describes the "recipe" to compute each output sample $y[n]$ as a linear combination of past and present input values and past output values. Clearly a CCDE involves only a finite number of linear operations and a finite amount of memory and it is therefore a realizable algorithm. In the case of FIR filters, all the a_k coefficients are zero for $k \geq 1$ and the CCDE coincides with the convolution sum. If any of the a_k coefficients are nonzero for $k \geq 1$, then the filter will have one or more feedback paths and the resulting impulse response will have infinite support.

CCDEs provide a powerful operational view of filtering and we will devote the next two chapters to studying their properties; in particular:

- the impulse and frequency responses of the filter implemented by a CCDE are well-defined and easy to determine;
- the stability of the associated filter is easy to check;
- there are a wealth of established and robust numerical techniques that allow us to determine the values of the coefficients **a** and **b** as a function of the desired properties of the filter.

6.6.2 Algorithms and Block Diagrams

In stark contrast to the *differential* equations used in the characterization of continuous-time systems, difference equations explicitly describe the algorithm that implements the system. As a simple example, it is straightforward to code a generic CCDE in Python:

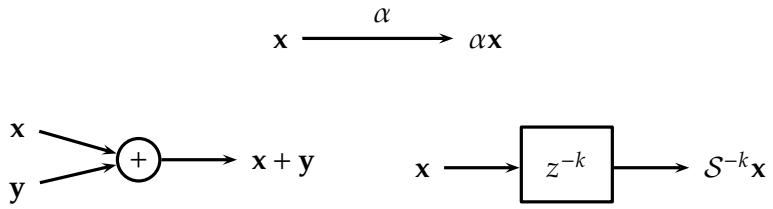

Figure 6.20: *Block diagram elements: scalar multiplication, addition and delay by k samples.*

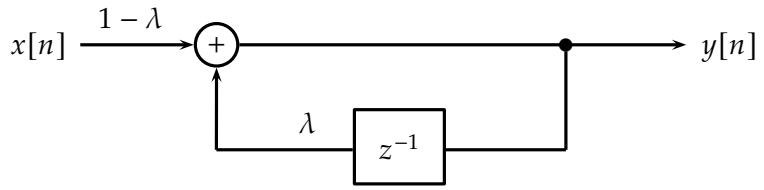

Figure 6.21: *Block diagram of the Leaky Integrator.*

```
class Filter:
    def __init__(self, b, a):
        self.a = a[1:]
        self.b = b
        self.y = [0] * len(self.a)
        self.x = [0] * len(self.b)
        self.ix = 0
        self.iy = -1

def compute(self, x):
    N, M = len(self.a), len(self.b)
    y = 0
    self.x[self.ix] = x
    for k, b_k in enumerate(self.b):
        y += b_k * self.x[(self.ix - k) % M]
    for k, a_k in enumerate(self.a):
        y -= a_k * self.y[(self.iy - k) % N]
    self.ix = (self.ix + 1) % M
    self.iy = (self.iy + 1) % N
    self.y[self.iy] = y
    return y
```

The code uses simple circular buffers of size $N - 1$ and M to store values of $\mathbf{y}$ and $\mathbf{x}$ respectively and sets them to zero upon instantiation of the class to ensure zero initial conditions. A leaky integrator can be instantiated via the statement `f = Filter([1, lambda], [1-lambda])` and used on the incoming samples via `y = f.compute(x)`.

Another common algorithmic representation of realizable filters relies on *block diagrams* showing the interconnections between the basic units shown in Figure **??**; the CCDE describing a Leaky Integrator, for instance, is shown in Figure 6.21. This type of representation provides an abstraction from any given programming language while still offering the ability to illustrate algorithmic implementation details, as we will see more in detail in Section **??**.