Selected Advanced Topics in Digital Signal Processing

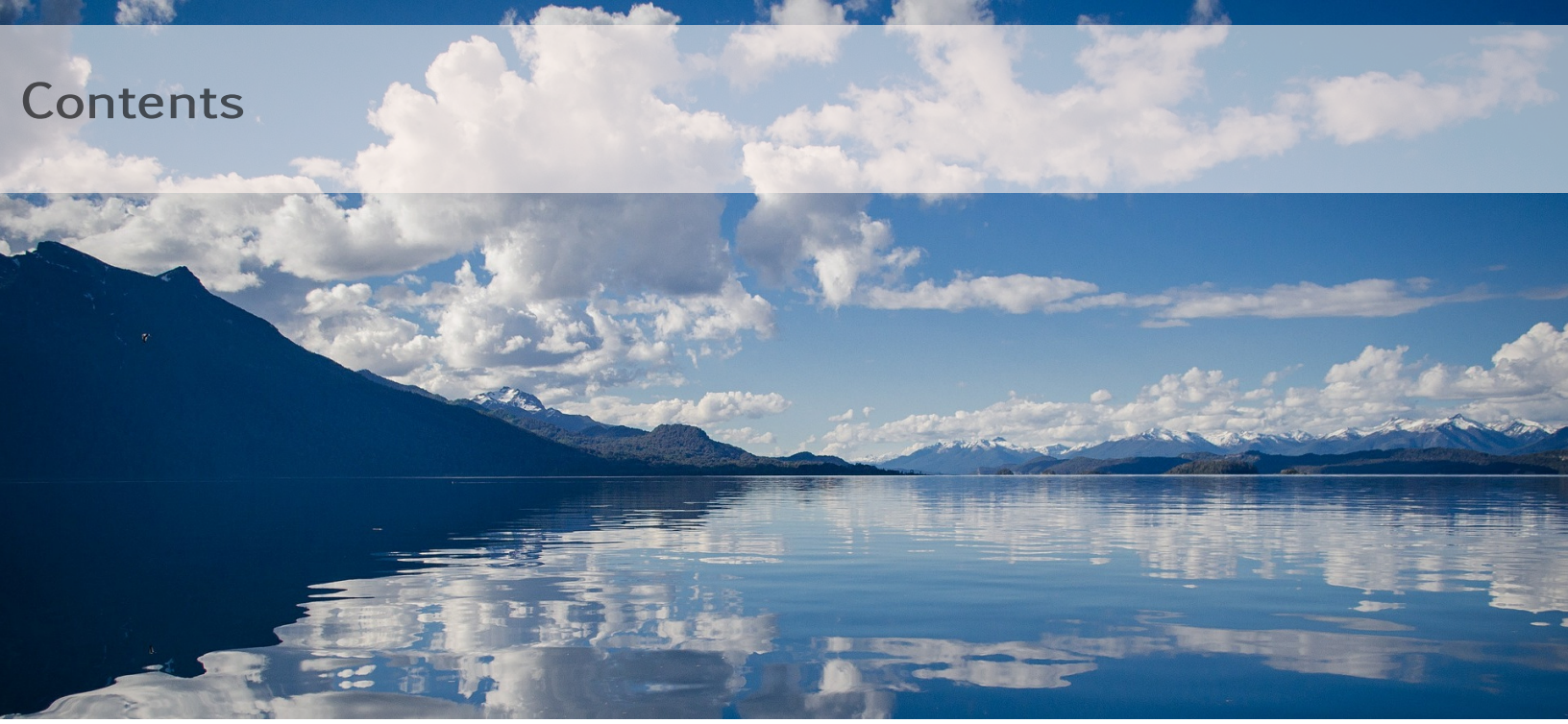
Richard M. Stern

ADSP Press

Art. No xxxxx
ISBN xxx–xx–xxxx–xx–x
Edition 0.8

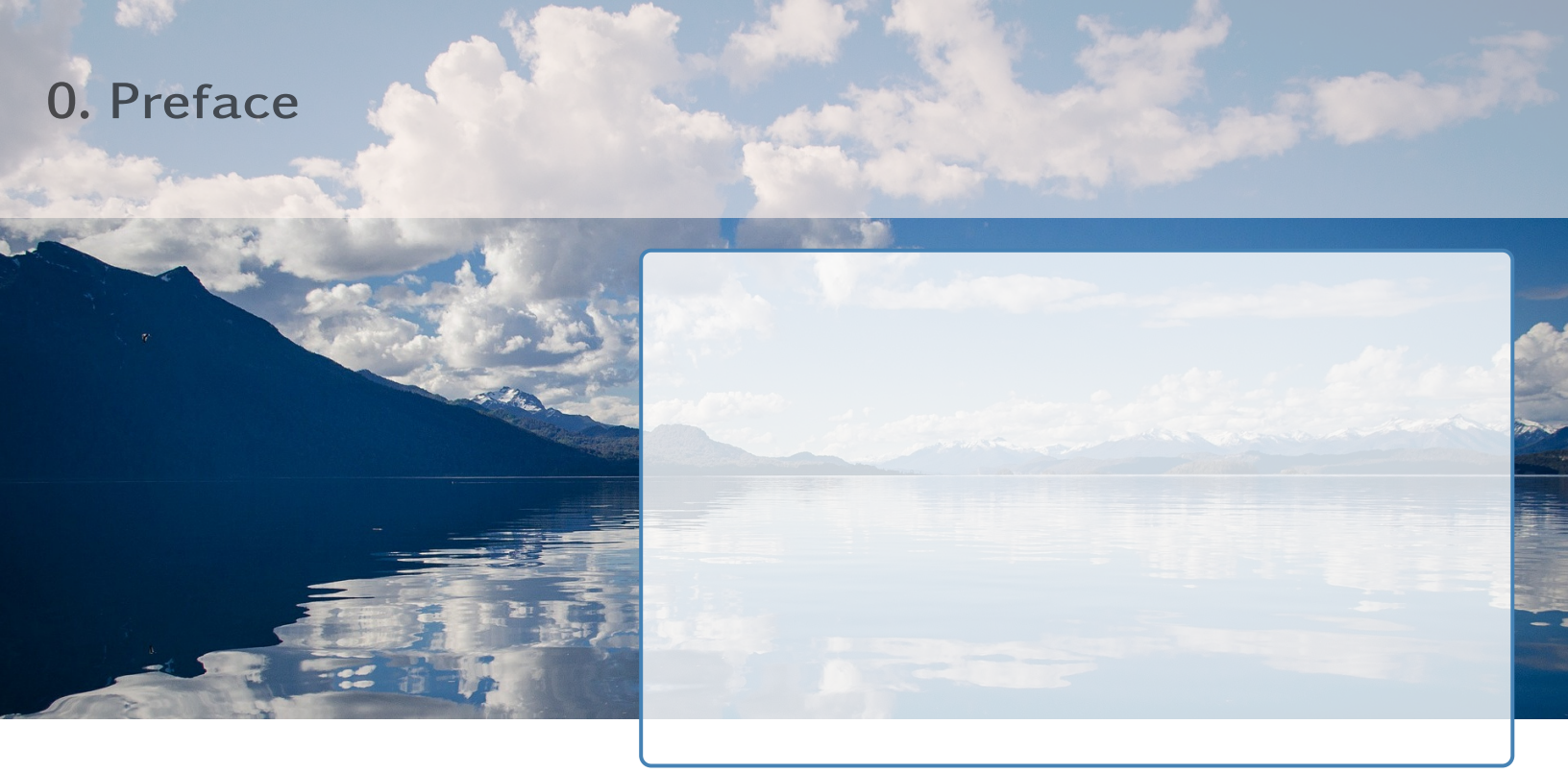
Cover design by Cover Designer

Published by ADSP Press
Printed in Pittsburgh, Pennsylvania USA



Contents

0. Preface



I first volunteered to teach a second course beyond the standard DSP course in the spring of 1985. At that time, as well as now, the content of the standard DSP course like 18-491 was fairly consistent across universities, typically following texts such as those written by Oppenheim and Schafer or Proakis and Manolakis. Nevertheless, I was surprised to find that the content for courses at the second level was all over the map. After a bit of experimentation, I came to decide that the set of topics presently covered in 18-792, which include both the deterministic discussions on change of sampling rate and short-time Fourier analysis, and a prolonged discussion of selected topics related to random signals including spectral estimation, linear prediction, adaptive filtering, and adaptive arrays.

Teaching the course became much easier by the publication in 1988 of the book *Advanced Topics in Signal Processing*, edited by Jae Lim and Al Oppenhiem, the former a fellow student and the latter an early mentor from my college and graduate school days. I used, and continue to use, chapters from the Lim and Oppenheim book[1], as well as the excellent book on speech processing *Digital Processing of Speech Signals* by Larry Rabiner and Ron Schafer, published in 1978 and updated in 2010.

Over the years there would be times at which I would have to miss lectures because of conflicts with professional conferences, obligatory meetings with research sponsors, etc. After some less-than-successful experiments with substitute lecturers I began in 2000 to pre-record the lectures I missed and have a teaching assistant present the lecture in class and respond to questions. Especially early on, it was not so easy to see the board at times in the videos, so I would supplement the lectures with very rough printed notes that were

[1]The Lim and Oppenheim book is currently out of print, but we have permission to reprint selected chapters for internal use.

distributed in class. As time went by, the coverage of the course by the notes grew to about half of the lectures and the notes improved with editing. I started distributing the notes even for lectures when I was present in person because they appeared to be helpful to some.

As we all know, our style of teaching this year (along with pretty much everything else) has been radically affected by COVID-19. Because every lecture this fall will be presented online (although the technology has improved quite a bit), I took on the task of expanding the notes to cover the entire course, formatting the material as a pseudo-book. My motivations were several: making the contents of the lectures more accessible (even though the lectures themselves and in-class notes are recorded and available to the students), helping the students navigate and integrate material from multiple primary sources with better transitions, and in part having the opportunity simply to present the material in my own fashion.

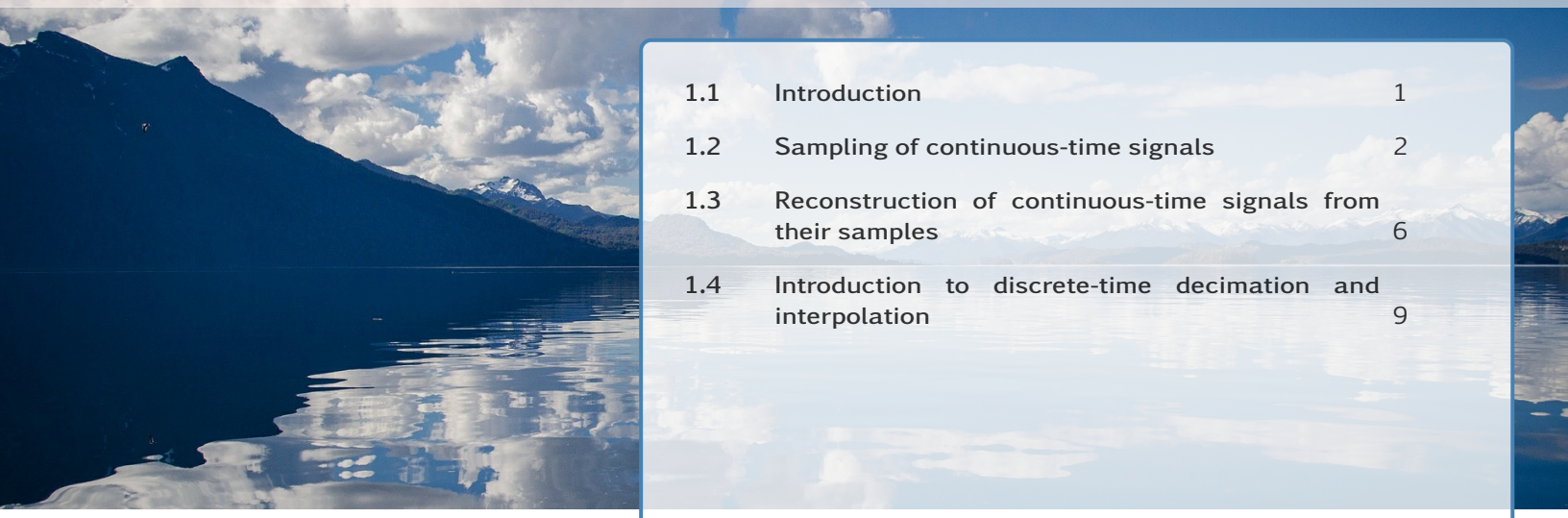
It should be clearly understood, though, what these notes are and what they are not. Specifically, these notes are no more than a condensed prose transcription of what I present in class. They do not provide anything close to the context that is available from reading the original sources, which are identified at the beginnings of each chapter. Similarly, they are not guaranteed to provide all the information that is in the original lectures. Hence, these notes should be considered to be a supplement to, rather than a replacement for, the original lectures and text material. All of these disclaimers notwithstanding, I do hope that they will be useful to at least some of you.

I deeply thank Mark Lindsey, Jade Traiger, Tyler Vuong, and Yangyang (Raymond) Xia for reviewing earlier drafts of this text and providing many useful corrections, comments, and suggestions. I also thank Vrishrab Commuri for translating earlier versions of some chapters from FrameMaker to LaTeX, and for a great deal of additional advice and comments. Finally, I thank Danish Mohammed Danish, Stella Getz, Loïc Lescoat, Davis Polito, John Shi, Carlos Taveras, Naoki Tsuda, Zach Vickland, Hanzhi Yin, and especially Ruiyang Jin and Yinghao Ma for pointing out additional errors in the initial manuscript. Nevertheless, please let me know about any remaining errors, as well as any suggestions you might have that would make the text more correct, complete, and helpful. Thank you all!

–Richard Stern
August 27, 2020

To Lauren, Sarah, and all my students

1. Working Between Continuous and Discrete Time



In signal processing we need to be able to move fluently between the continuous-time and discrete-time worlds. Natural signals, and our perception of them, nearly always occur in the continuous-time world. Nevertheless, almost all signal processing operations are implemented in the discrete-time world. Hence we need to be able to understand how we move from one world to the other, and to understand the implications that sampling has on the nature of the signals in time and frequency.

1.1 Introduction

In this chapter we review the mathematics that are the basis for sampling of continuous-time signals and their reconstruction. We will also discuss the closely-related mathematics that underly discrete-time decimation and interpolation that enable changes in the effective sampling rate of a discrete-time signal.

1.2 Sampling of continuous-time signals

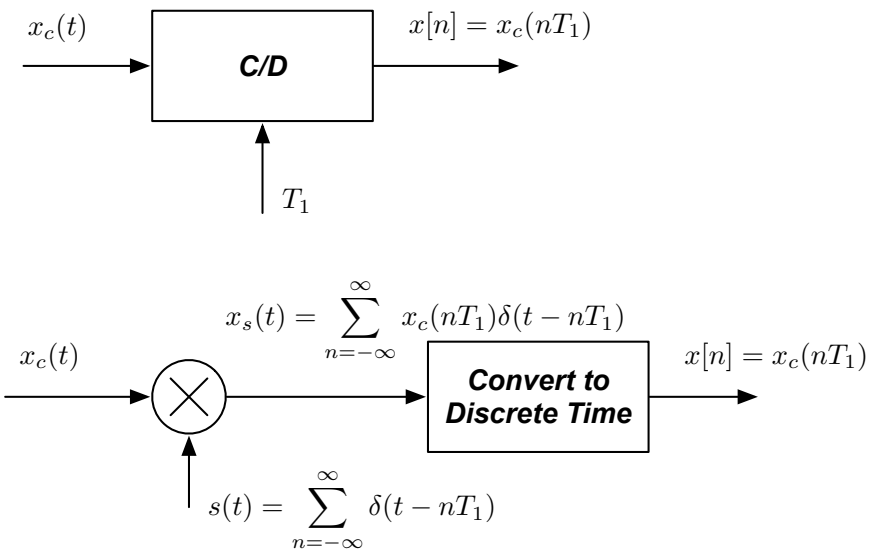


Figure 1.1: Upper panel: block diagram of C/D conversion. Lower panel: Specification of the underlying signal processing.

The block diagram in the upper panel of Fig. 1.1 shows the basic structure of the ideal *continuous-to-discrete-time converter* or C/D converter. The input is a continuous-time function $x_c(t)$, which we assume to be a bandlimited time function such that its continuous-time Fourier transform $X(j\Omega)$ equals zero for $|\Omega| > W$.[1] The output is a discrete-time function $x[n]$ that is equal to $x_c(t)$ evaluated at times $t = nT_1$, where T_1 is the sampling period in seconds.

In order to understand the mathematical representation of sampling and its implications, we use the more detailed description in the lower panel of Fig. 1.1. Specifically, the input is first multiplied by $s(t)$, an infinite train of delta functions of area 1, which are separated by an interval of T_1 seconds. The function $x_s(t)$ is the product of $x_c(t)$ and $s(t)$, and is a train of delta functions $x_s(t)$ that are separated in time by intervals of T_1 seconds and have areas equal to the amplitude of the original signal $x_c(nT_1)$ at the times at which the impulses occur. This sequence is then converted magically (in our mathematical model) into discrete time pulses $x[n]$ which have *amplitudes* equal to the *areas* of the corresponding delta functions in $x_s(t)$, so $x[n]$ will be equal to $x_c(nT_1)$.

Time-domain representations. It is helpful to be able to visualize these signals in both time and frequency. The left side of Fig. 1.2 depicts, for an arbitrary time function and its spectrum, the various continuous-time functions and discrete-time functions depicted in Fig. 1.1. Their Fourier transforms are shown on the right side of the figure. More specifically,

$$s(t) = \sum_{n=-\infty}^{\infty} \delta(t - nT_1) \tag{1.1}$$

[1] We will adopt the notational convention of using the Greek letter Ω for continuous-time frequency in radians per second and using ω for discrete-time frequency in radians.

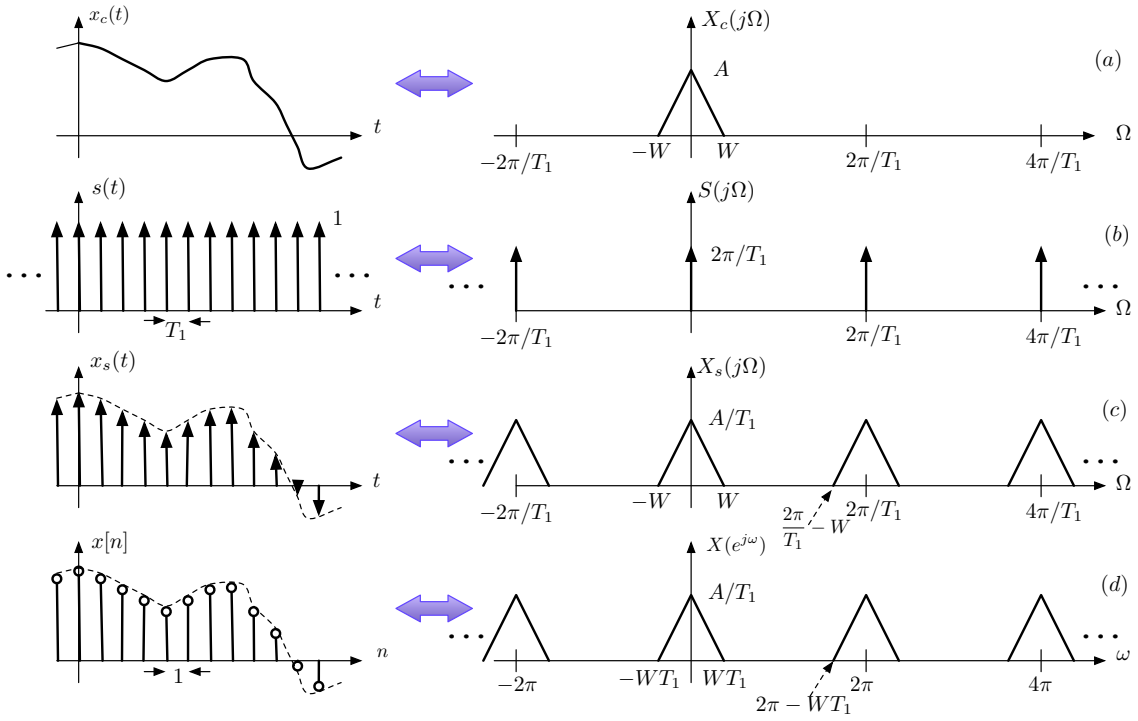


Figure 1.2: Left column: the time functions (a) $x_c(t)$, (b) $s(t)$, (c) $x_s(t)$, and (d) $x[n]$. Right column: The corresponding Fourier transforms in continuous and discrete time.

As described above,

$$x_s(t) = x_c(t)s(t) = \sum_{n=-\infty}^{\infty} x_c(nT_1)\delta(t - nT_1) \tag{1.2}$$

The function $x[n]$ is a train of discrete-time samples that have *amplitudes* that are equal to the *areas* of the corresponding impulses of $x_s(t)$. We also note that $x_s(t)$ is a function of t, which is a real variable, while $x[n]$ is a function of n, which is meaningful only when it is an integer.

$$x[n] = \sum_{l=-\infty}^{\infty} x_c(lT_1)\delta[n - l] \tag{1.3}$$

Frequency-domain representations. Considering now the corresponding functions in the frequency domain, let us assume that $X_c(j\Omega)$, the continuous-time Fourier transform (CTFT) of $x_c(t)$, is of arbitrary shape, nonzero only for $|\Omega| \leq W$, and with a maximum amplitude equal to A, as depicted in Fig. 1.2. It can be shown that $S(j\Omega)$, the CTFT of $s(t)$, is also an infinite train of delta functions:

$$S(j\Omega) = \frac{2\pi}{T_1} \sum_{k=-\infty}^{\infty} \delta\left(\Omega - \frac{2\pi k}{T_1}\right) \tag{1.4}$$

Now, because $x_s(t) = x_c(t)s(t)$ in the time domain, we obtain in the frequency domain

$$X_s(j\Omega) = \frac{1}{2\pi} X_c(j\Omega) * S(j\Omega) = \frac{1}{2\pi} \int_{-\infty}^{\infty} X_c(j\theta)S(j(\Omega - \theta))d\theta \tag{1.5}$$

where the symbol $*$ indicates convolution. Because convolving $X_c(j\Omega)$ with an infinite train of delta functions merely replicates and shifts the spectrum, we obtain

$$X_s(j\Omega) = \frac{1}{2\pi} X_c(j\Omega) * \sum_{k=-\infty}^{\infty} \frac{2\pi}{T_1} \delta\left(\Omega - \frac{2\pi k}{T_1}\right) = \frac{1}{T_1} \sum_{k=-\infty}^{\infty} X_c\left(j\left(\Omega - \frac{2\pi k}{T_1}\right)\right) \tag{1.6}$$

as shown in Row (c) of Fig. 1.2.

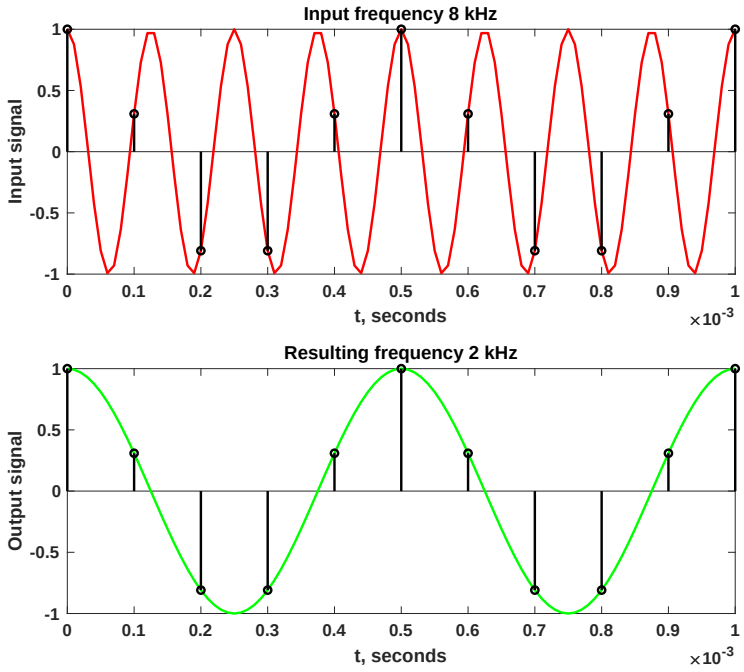


Figure 1.3: Example of aliasing distortion, with an input signal of 8 kHz (upper panel) sampled at 10 kHz producing an output signal of 2 kHz (lower panel).

Avoiding aliasing distortion. In order for the continuous-time signal to ultimately be recovered without distortion, it is necessary that the replications of the original spectrum in Fig. 1.2 not overlap one another. It can easily be seen that the separation of the replications is related to the sampling period T_1 while the width of the replications is twice the signal bandwidth W. It is clear from Fig. 1.2 that overlap will be avoided if

$$\frac{2\pi}{T_1} - W > W \text{ or } W < \frac{\pi}{T_1} \tag{1.7}$$

In other words, the sampling frequency must be at least twice as great as the bandwidth of the incoming signal to avoid aliasing distortion. This, of course, is merely a restatement of the Nyquist constraint that determines the minimum sampling frequency for distortionless recovery of the continuous-time signal.

Figure 1.3 shows an example of the aliasing that occurs when the sampling frequency is not great enough. The upper panel is an 8-kHz sinusoid with sample points superimposed by sampling at 10 kHz, which imposes an upper bound of 5 kHz for sampling

without aliasing. The lower panel of the figure depicts the function that emerges from the sampling and reconstruction process: a 2-kHz signal. Note that the discrete-time sample points fit both the 8-kHz input signal and the 2-kHz output signal equally well. To prevent the effects of aliasing, it is common to precede a C/D converter by a lowpass filter with gain ideally equal to 1 and a cutoff frequency of $\Omega = \pi/T_1$.

Relating continuous-time and discrete-time frequency. The relationship between $X_s(j\Omega)$, the CTFT of $x_s(t)$, and $X(e^{j\omega})$, the discrete-time Fourier transform (DTFT) of $x[n]$, is a bit subtle. From the definition of the DTFT, $X(e^{j\omega})$ is

$$X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n} = \sum_{n=-\infty}^{\infty} x_c(nT_1)e^{-j\omega n} \tag{1.8}$$

Writing the definition of $X_s(j\Omega)$ and incorporating Eq. (1.2) produces

$$X_s(j\Omega) = \int_{-\infty}^{\infty} x_s(t)e^{-j\Omega t}dt = \int_{-\infty}^{\infty} \sum_{n=-\infty}^{\infty} x_c(nT_1)\delta(t-nT_1)e^{-j\Omega t}dt \tag{1.9}$$

Interchanging the sum and the integral and applying the procedures for integrating expressions with delta functions (see the Appendix for a detailed discussion of integration with delta functions) produces

$$X_s(j\Omega) = \sum_{n=-\infty}^{\infty} x_c(nT_1) \int_{-\infty}^{\infty} \delta(t-nT_1)\, e^{-j\Omega t}dt = \sum_{n=-\infty}^{\infty} x_c(nT_1)e^{-j\Omega nT_1} \tag{1.10}$$

We note that the final terms of Eqs. (1.8) and (1.10) are identical, except that

$$\omega = \Omega T_1 \tag{1.11}$$

where again T_1 is the sampling period. This means that $X(e^{j\omega})$, the DTFT of $x[n]$, is identical to $X_s(j\Omega)$, the CTFT of $x_s(t)$, except that the frequency axis is scaled by the sampling period T_1.[2] (Although it is easy to confuse discrete-time and continuous-time frequency in expressions like Eq. (1.11), keep in mind the dimensional analysis that discrete-time frequency in radians is equal to continuous-time frequency in radians per second times the sampling period in seconds.)

[2] If $X_s(j\Omega)$ includes delta functions, the areas of the delta functions would be scaled according to the relationship $\delta(at) = (1/|a|)\delta(t)$, as discussed in the Appendix.

1.3 Reconstruction of continuous-time signals from their samples

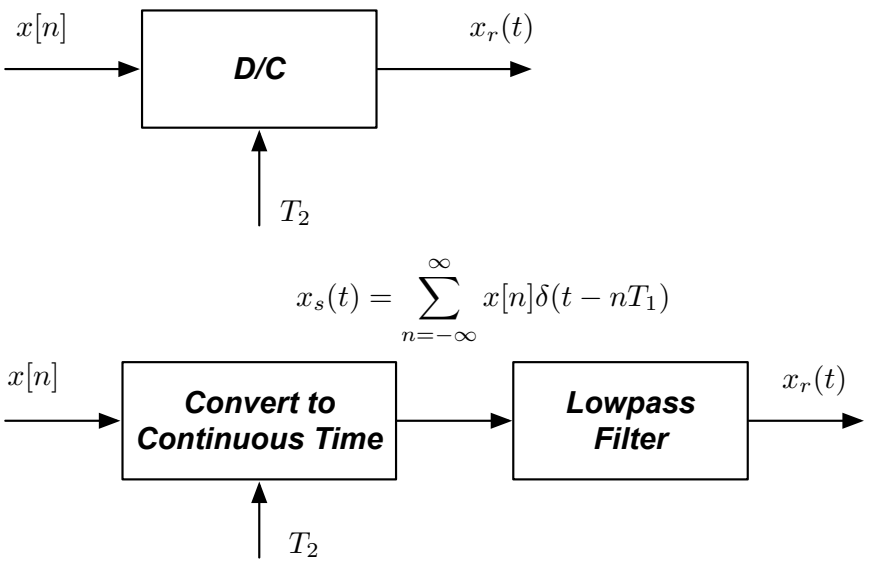


Figure 1.4: Upper panel: block diagram of D/C conversion. Lower panel: Specification of the underlying signal processing.

Let us now turn our attention to the reconstruction process, which is in some ways a reversal of the sampling process. Figure 1.4 summarizes the major processing steps. The discrete-time sequence $x[n]$ is converted into the continuous-time sequence of delta functions $x_s(t)$, in which the areas of the impulses in continuous time are equal to the corresponding amplitudes of the discrete-time samples:

$$x_s(t) = \sum_{n=-\infty}^{\infty} x[n]\delta(t - nT_2) \tag{1.12}$$

This sequence of delta functions is passed through an ideal lowpass filter with transfer function

$$H_{LP}(j\Omega) = \begin{cases} T_2, & |\Omega| < \pi/T_2 \\ 0, & \text{otherwise} \end{cases} \tag{1.13}$$

The gain factor of T_2 in $H_{LP}(j\Omega)$ ensures that if the original signal is sampled with a great enough sampling frequency to avoid aliasing, and if the sampling period T_1 equals the time T_2 between the samples used in reconstruction, then the continuous-time recovered signal $x_r(t)$ will be identical to the original input $x_c(t)$. In effect, when $T_1 = T_2$ the gain factor of T_2 in the lowpass filter in the D/C converter is intended to compensate for the factor of $1/T_1$ that incurred in the C/D conversion process. If $T_2 \neq T_1$, the signal passing through will nominally incur a gain of T_2/T_1.

The reconstruction process is illustrated in the frequency domain in Fig. 1.5 which can be seen to retrace most of the functions shown in Fig. 1.2 in reverse order. In this figure we assume that the reconstruction sample period T_2 is equal to the original sampling period T_1, although this is not always the case. The discrete-time sequence $x[n]$, depicted in the upper panel of Fig. 1.5 is first converted into the continuous-time sequence of

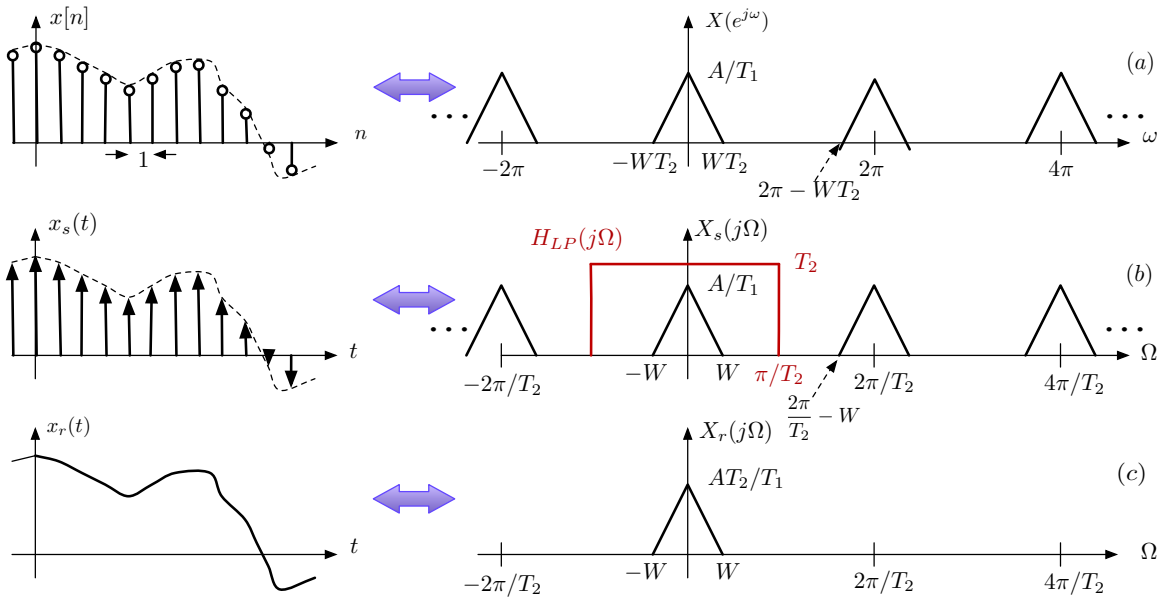


Figure 1.5: Reconstruction of a continuous-time function from its samples. Left side: the time functions (a) $x[n]$, (b) $x_s(t)$, and (d) $x_r(t)$. Right side: The corresponding Fourier transforms in discrete and continuous time. The ideal lowpass filter $H_{LP}(j\Omega)$ is also shown in panel (b) in red. In plotting, we assume that $T_2 = T_1$ in Fig. 1.2.

delta functions, $x_s(t)$, which is depicted in the central panel of Fig 1.5. As discussed in conjunction with Eq. (1.8) through Eq. (1.11), $X_s(j\Omega)$, the CTFT of $x_s(t)$, is identical to $X(e^{j\omega})$, the DTFT of $x[n]$ except that frequencies are scaled according to the relationship that $\omega = \Omega T_2$, which produces an infinite train of replications of the original frequency response, as in the central panel of Fig. 1.5. The lowpass reconstruction filter is depicted in red in the central panel of Fig. 1.5, and the product of $X_s(j\Omega)$ and $H_{LP}(j\Omega)$ is the single-mode spectrum $X_r(j\Omega)$ which is (ideally) equal to the original input spectrum $X_c(j\Omega)$ depicted in Fig. 1.2.

It is helpful to consider what is going on in the time domain to understand the nature of the reconstruction of the continuous-time output $x_r(t)$ from the sequence of delta functions $x_s(t)$. The output in the time domain is (as usual) the convolution of the input with the unit impulse response of the filter:

$$x_r(t) = x_s(t) * h_{LP}(t) \tag{1.14}$$

The unit impulse response of the filter is easily obtained directly:

$$h_{LP}(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} H_{LP}(j\Omega)e^{j\Omega t}d\Omega = \frac{1}{2\pi}\int_{-\pi/T_2}^{\pi/T_2} T_2 e^{j\Omega t}d\Omega = \frac{\sin(\pi t/T_2)}{\pi t/T_2} \tag{1.15}$$

This is a continuous-time sinc function with amplitude equal to 1 at $t = 0$ and regularly-spaced zero crossings at $t = nT_2$. Because the convolution of any function with a train of impulses produces replication of the original function at the times of the impulses, weighted by the impulse areas, we obtain

$$x_r(t) = h_{LP}(t) * x_s(t) = h_{LP}(t) * \sum_{n=-\infty}^{\infty} x[n]\delta(t - nT_2) = \sum_{n=-\infty}^{\infty} x[n]\frac{\sin(\pi(t - nT_2)/T_2)}{\pi(t - nT_2)/T_2} \tag{1.16}$$

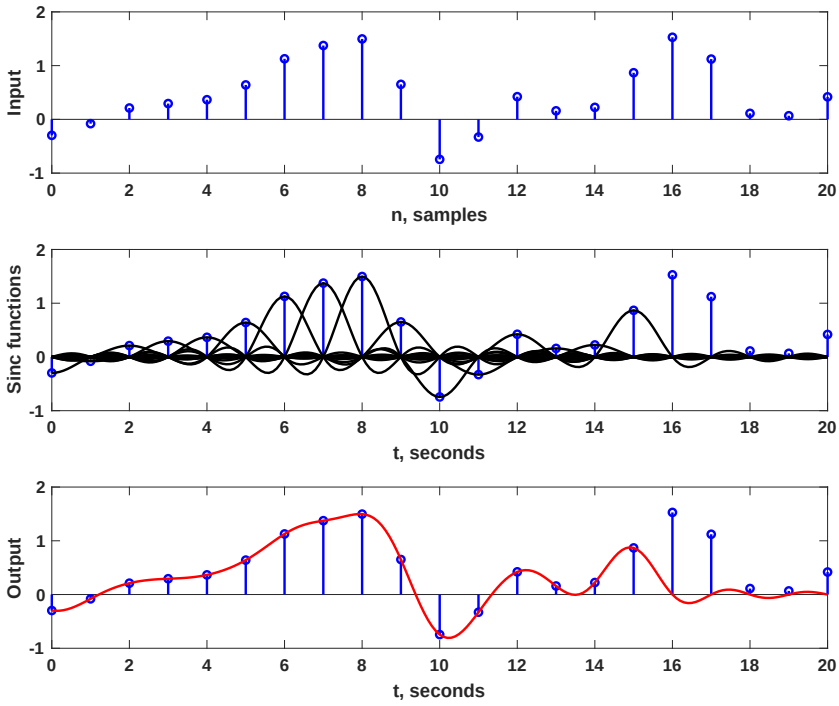


Figure 1.6: Upper panel: a segment of a discrete-time signal. Center panel: representation of that signal in continuous time by a linear combination of sinc functions. Lower panel: the sum of the sinc functions as of $t = 15$ seconds.

In other words, the reconstructed continuous-time function $x_r(t)$ can be represented as the sum of an infinite train of sinc functions, delayed by intervals of T_2 and scaled according to the corresponding value of $x[n]$. This is illustrated in Fig. 1.6, which uses the convenient but unrealistic value of $T_2 = 1$ second. The upper panel of Fig. 1.6 depicts a short segment of a discrete-time signal. The central panel shows its reconstruction as a series of weighted and delayed sinc functions. Note that the reconstruction is exact at the time of the original samples as the sinc function centered each sample point has an amplitude of 1 while the other sinc functions are all zero. The values of the reconstructed signal (lower panel) between the sample points are obtained by summing all the sinc functions together. While there is an infinite number of sinc functions, they decay away fairly rapidly. The lower two panels of Fig. 1.6 are computed only through 15 seconds. It can be seen that the reconstruction of $x_r(t)$ appears to be accurate except for values of t that are near to or greater than 15 seconds.

In practice, of course, the actual lowpass filter to reconstruct the continuous-time signal would not be ideal for multiple reasons including the fact that all ideal filters are fundamentally unrealizable. Nevertheless, the reconstructed signal would still be an infinite linear combination of the unit impulse response of the lowpass filter that is actually used.

1.4 Introduction to discrete-time decimation and interpolation

It is frequently necessary or desirable to change the effective sampling rate of discrete-time signals. We consider three standard ways of manipulating the effective sampling frequency in discrete time:

- *Decimation* or *downsampling* the signal by an integer factor of M, which decreases the effective sampling rate by a factor of M
- *Interpolation* or *upsampling* the signal by an integer factor L, which increases the effective sampling rate by a factor of L
- A combination of interpolation and decimation which changes the effective sampling rate of the signal by the rational factor of L/M.

There are multiple potential motivations for changing the sampling rate. For example, the sampling rate may be needed to combine signals that had been recorded at different different sampling rates, as would be necessary if audio from a CD recorded at the standard rate of 44.1 kHz were used as background music for a film, which uses 48 kHz as the standard sampling rate. Multi-rate signal processing is useful for delaying a signal by a fractional number of samples. For example, a delay of 1/4 sample can be accomplished by upsampling a signal by a factor of 4, delaying by a single sample, and then downsampling by a factor of 4. Finally, multi-rate signal processing techniques are frequently useful in the design of filters that must have a very narrow passband (or stopband) with narrow transition bands.

Figure 1.7 illustrates downsampling and upsampling for a short segment of a discrete-time speech signal. The original signal is depicted in the upper panel, while the central panel shows the signal after downsampling by a factor of 3, and the lower panel shows the signal after upsampling by a factor of 2. The solid curve in red shows the original continuous-time signal from which the discrete-time signal was derived. The left column shows the three functions with the horizontal axes scaled to maintain the continuous-time envelope unchanged, while the right column plots the three discrete-time samples in a fashion that preserves the spacing between the samples. It can be seen that the decimation and interpolation operations can be interpreted either as a decrease or increase of the effective sampling rate of the discrete-time functions relative to the continuous-time functions (as in the left column) or as an intrinsic compression or expansion in time of the discrete-time functions (as in the right column).

In the following sections we describe and discuss the fundamental mathematics of discrete-time decimation, interpolation, and change of sample rate by a rational factor. We will find that the operations associated with decimation and interpolation are closely related to ideal C/D and D/C conversion, respectively, that had been described in Secs. 1.2 and 1.3 above. We will discuss computationally-efficient ways of realizing these operations in Chapter 2 below.

1.4.1 Decimation: reducing the sampling rate by an integer factor M

In continuous-time signal processing, Fourier transform properties relate compression in time to expansion in frequency and vice-versa. Specifically, if

$$x(t) \Leftrightarrow X(j\Omega) \tag{1.17}$$

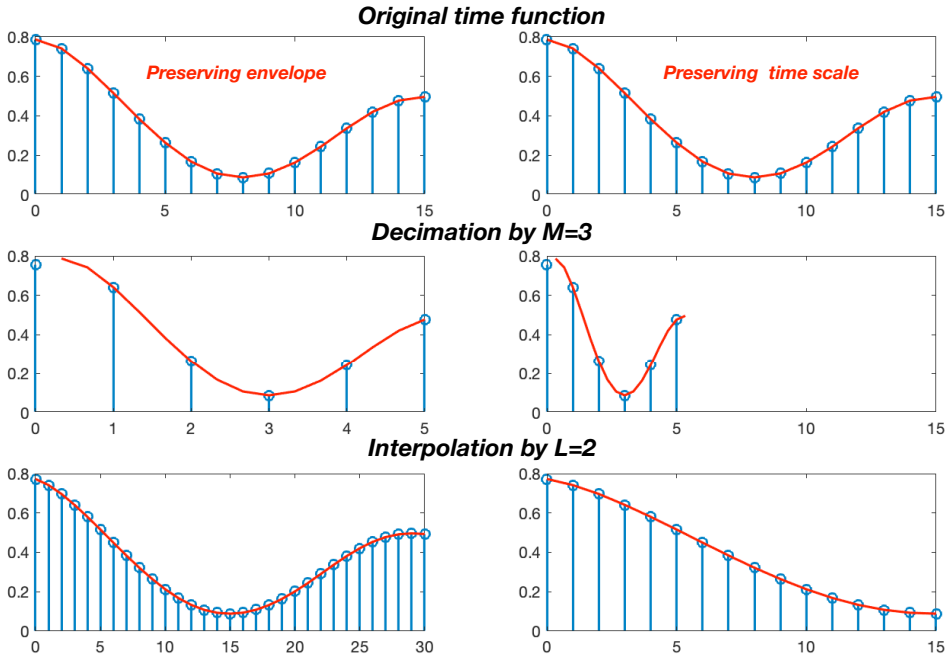


Figure 1.7: Comparison of an original discrete-time function (upper row), the function after decimation by $M = 3$ (central row), and the function after interpolation by $L = 2$ (lower row). Left column: the spacing of the samples is scaled according to the change in sample rate. Right column: the spacing of the samples is constant across all three sample rates.

then

$$x(at) \Leftrightarrow \frac{1}{|a|} X\left(j\frac{\Omega}{|a|}\right) \tag{1.18}$$

As we noted above, we can think of decimation as a process by which we compress in the signal along the time axis. Nevertheless, the time-compression is not straightforward because of the constraint that discrete-time signals are meaningful only for sample values that are integer. For example, if we were downsampling by a factor of 2, each of the odd samples of the original time function would disappear after decimation. In order to represent the impact of the samples that do not "survive" the decimation process on the frequency response, we must set those samples to zero explicitly in our mathematical analysis of the processing. Consequently, we mathematically describe the decimation process as in lower panel of the block diagrams in Fig. 1.8. The upper panel of this figure shows the basic structure of a system that performs decimation or downsampling by a factor of M. The input is a discrete-time function $x[n]$, which we assume to be a bandlimited time function such that its discrete-time-time Fourier transform $X(e^{j\omega})$ equals zero for $W < |\omega| \leq \pi$. The output is the decimated discrete-time function $x_d[m]$ which is equal to $x[n]$ evaluated at times $n = mM$, where M is the downsampling ratio. (We use the index m rather than n to call attention to the fact that the effective time base changes after decimation.)

The mathematical representation of the decimation process is specified in more detail in

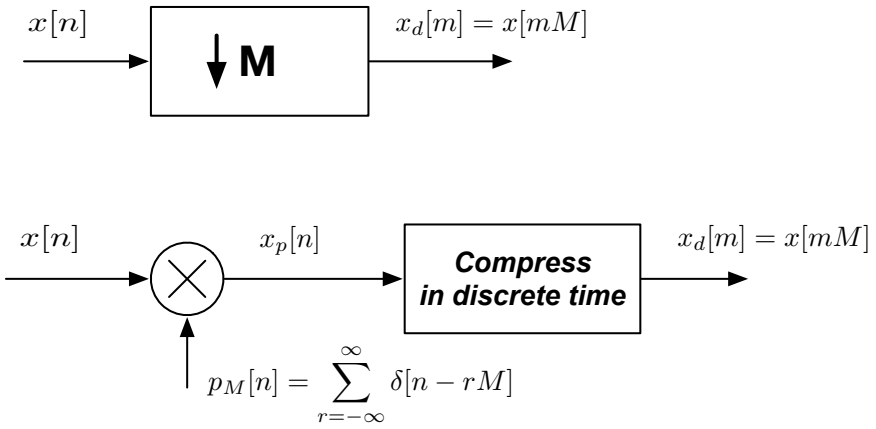


Figure 1.8: Upper panel: block diagram of decimation by M. Lower panel: Specification of the underlying signal processing.

the lower panel of Fig. 1.8. Specifically, the input is first multiplied by $p_M[n]$, an infinite train of delta functions in discrete time of amplitude 1, which are separated by an interval of M samples. The function $x_p[n]$ is the product of $x[m]$ and $p_M[n]$, and is a train of delta functions of amplitude $x[nM]$ which are separated in time by intervals of M samples, with the values of $x_p[n]$ set equal to zero when n is not an integer multiple of M. This sequence is then compressed in time so that $x_d[m] = x[mM]$.

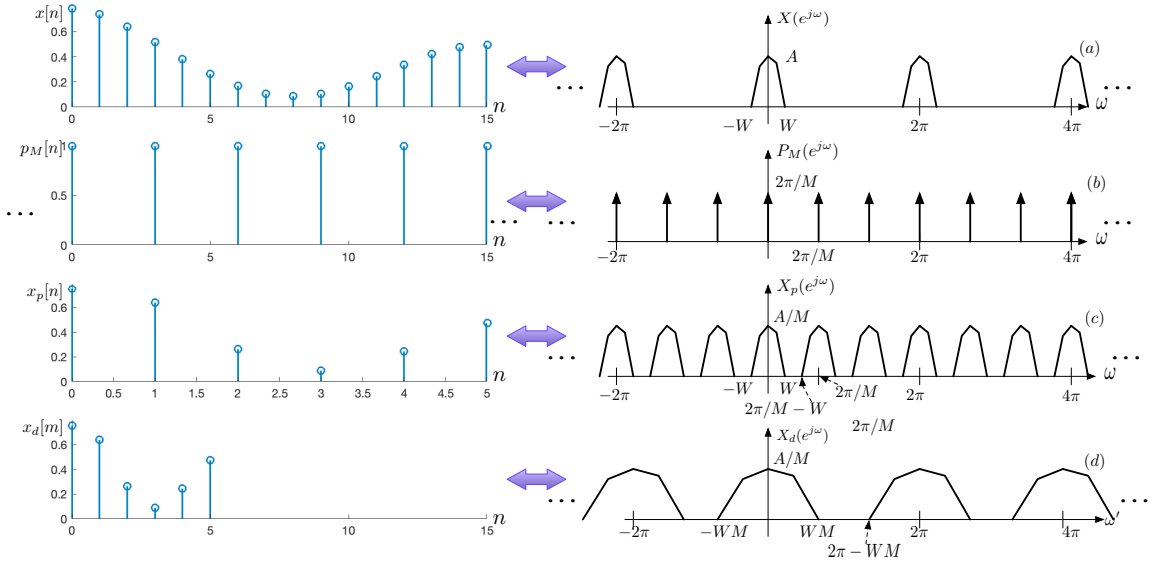


Figure 1.9: Representative time functions and their spectra in discrete-time decimation, illustrated for $M = 3$. Left side: the time functions (a) $x[n]$, (b) $p_M[n]$, (c) $x_p[n]$, and (d) $x_d[m]$. Right side: The corresponding discrete-time Fourier transforms.

Time-domain representations. It is helpful to be able to visualize these signals in both time and frequency. The left side of Fig. 1.9 depicts, for an arbitrary time function and its spectrum, the various discrete-time functions depicted in Fig. 1.8 for the downsampling ratio $M = 3$. The corresponding Fourier transforms are shown on the right side of the

figure. More specifically,

$$p_M[n] = \sum_{r=-\infty}^{\infty} \delta[n - rM] \tag{1.19}$$

$$x_p[n] = x[n]p_M[n] = \begin{cases} x[n], & n = rM \\ 0, & \text{otherwise} \end{cases} \tag{1.20}$$

The functions $x[n]$, $p_M[n]$, and $x_p[n]$ are depicted in panels (a), (b), and (c), respectively in the left column of Fig. 1.9. The function $x_p[n]$ consists of the samples of $x[n]$ that will survive the decimation process. The output of the decimation process, $x_d[m]$, consists of these same nonzero samples, but the time axis is now compressed so that the nonzero delta functions are now found at successive values of the new time index m.

Frequency-domain representations. Considering now the corresponding functions in the frequency domain, we will assume that $X(e^{j\omega})$, the DTFT of $x[n]$, is of arbitrary shape, bandlimited such that $X(e^{j\omega}) = 0$ for $W < |\omega| \leq \pi$, and with a maximum amplitude equal to A, as depicted in Fig. 1.9. Like all DTFTs, $X(e^{j\omega})$ is periodic with period 2π.

It is easy to demonstrate by computing the inverse DTFT that $P_M(e^{j\omega})$, the DTFT of $p_M[n]$, is also an infinite train of delta functions:

$$P_M(e^{j\omega}) = \frac{2\pi}{M} \sum_{k=-\infty}^{\infty} \delta\left(\omega - \frac{2\pi k}{M}\right) = \frac{2\pi}{M} \sum_{r=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} \delta\left(\omega - \frac{2\pi l}{M} - 2\pi r\right) \tag{1.21}$$

$P_M(e^{j\omega})$ is expressed in double-sum form to emphasize the fact that the DTFT is periodic with period 2π and that there are M equally-spaced delta functions within each periodic cycle, separated by $\omega = 2\pi/M$, and each with amplitude $2\pi/M$. Because $x_p[n] = x[n]p_M[n]$ in the time domain, the corresponding DTFTs are

$$X_p(e^{j\omega}) = \frac{1}{2\pi} X(e^{j\omega}) \circledast P_M(e^{j\omega}) = \frac{1}{2\pi} \int_{-\pi}^{\pi} X(e^{j\theta}) P_M(e^{j(\omega-\theta)}) d\theta \tag{1.22}$$

where the symbol $\circledast$ indicates circular convolution. Because convolving $X(e^{j\omega})$ with the limited train of delta functions within a span of $\omega = 2\pi$ for $P_M(e^{j\omega})$ merely replicates and shifts the spectrum $M - 1$ additional times, we obtain

$$x_p(e^{j\omega}) = \frac{1}{2\pi} X(e^{j\omega}) \circledast \frac{2\pi}{M} \sum_{k=-\infty}^{\infty} \delta\left(\omega - \frac{2\pi k}{M}\right) = \frac{1}{M} \sum_{r=0}^{M-1} X(e^{j(\omega - \frac{2\pi r}{M}))}) \tag{1.23}$$

as shown in Row (c) of Fig. 1.9.

Avoiding aliasing distortion. In order for the downsampled discrete-time signal to be ultimately be recovered without distortion, it is necessary that the replications of the original spectrum in Fig. 1.9 not overlap one another. It can easily be seen that the separation of the replications is related to the downsampling ratio of M while the width of the replications is twice the signal bandwidth W. It is clear from Fig. 1.9 that overlap will be avoided if

$$\frac{2\pi}{M} - W > W \text{ or } W < \frac{\pi}{M} \tag{1.24}$$

In other words, the sampling frequency must be at least $2M$ as great as the bandwidth of the signal to avoid aliasing distortion. Most practical downsampling systems include

a lowpass filter of gain 1 and a nominal cutoff frequency of π/M to avoid distortion by aliasing.

Changing the sampling rate. The final step in the decimation process is changing the time scale of the discrete-time function so that the non-zero samples have successive indices. Specifically, we define a new function and a new time axis $x_d[m] = x_p[mM]$. The DTFT of $x_d[m]$ can be computed directly:

$$X_d(e^{j\omega'}) = \sum_{m=-\infty}^{\infty} x_d[m]e^{-j\omega' m} = \sum_{m=-\infty}^{\infty} x_p[mM]e^{-j\omega' m} \tag{1.25}$$

Letting $l = mM$ or $m = l/M$ produces

$$X_d(e^{j\omega'}) = \sum_{l=-\infty}^{\infty} x_p[l]e^{-j\omega'\left(\frac{l}{M}\right)} = \sum_{l=-\infty}^{\infty} x_p[l]e^{-j\left(\frac{\omega'}{M}\right)l} = X_p\left(\frac{\omega'}{M}\right) \tag{1.26}$$

In other words, the output $X_d(e^{j\omega'})$ is identical to the DTFT $x_p(e^{j\omega})$, except that it is stretched in frequency to be wider by a factor of M. We use the symbol ω' to represent the frequency after downsampling to recognize the fact that the frequency scale has changed because the time scale has changed. This function is depicted in the right column of the bottom panel of Fig. 1.9. While you may be concerned that $m = l/M$ is not integer for some values of m, these are exactly the values of m for which the function $x_p[m]$ is equal to zero.

1.4.2 Interpolation: increasing the sampling rate by an integer factor L

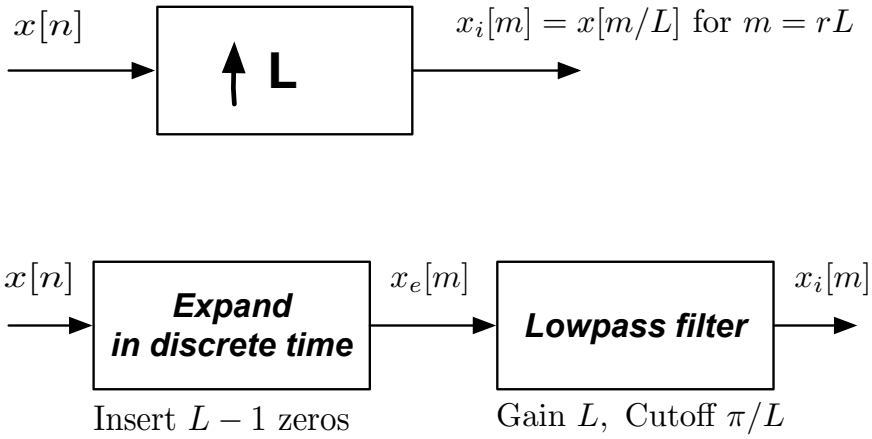


Figure 1.10: Upper panel: block diagram of interpolation by L. Lower panel: Specification of the underlying signal processing.

Just as decimation or downsampling is similar in some ways to C/D conversion, discrete-time interpolation or upsampling has some similarities to D/C conversion, which is at the end of the day itself an interpolation process in continuous time. Interpolation by a factor of L includes two major steps: (1) "expanding" the discrete-time function by inserting $L-1$ samples of amplitude zero between each successive sample of the input

and (2) lowpass filtering the resulting signal using an ideal filter with gain L and cutoff frequency π/L.

Expansion in time. Figure 1.10 depicts the major functions involved with interpolation in time and frequency. Using $x[n]$ to designate the input, we will represent the expanded time function as

$$x_e[m] = \begin{cases} x[m/L], & m/L \text{ integer} \\ 0, & \text{otherwise} \end{cases} \tag{1.27}$$

to produce the interpolated sequence $x_i[m]$ as the filter output. As stated above, we will use the variable m rather than n for the time index as a reminder that the time scale has changed.

The discrete-time sequence $x_e[m]$ that represents the expanded signal can also be written as

$$x_e[m] = \sum_{l=-\infty}^{\infty} x[l]\delta[m - lL] \tag{1.28}$$

This sequence of weighted delta functions is passed through an ideal lowpass filter with transfer function

$$H_{LP}(e^{j\omega}) = \begin{cases} L, & |\omega| < \pi/L \\ 0, & \pi/L < |\omega| \leq \pi \end{cases} \tag{1.29}$$

The gain factor of L in $H_{LP}(e^{j\omega})$ ensures that if an original signal is upsampled and then downsampled by the same ratio, the original signal will be recovered without distortion and with the same amplitude. There is no need to be concerned with aliasing distortion in the interpolation process because no information is lost.

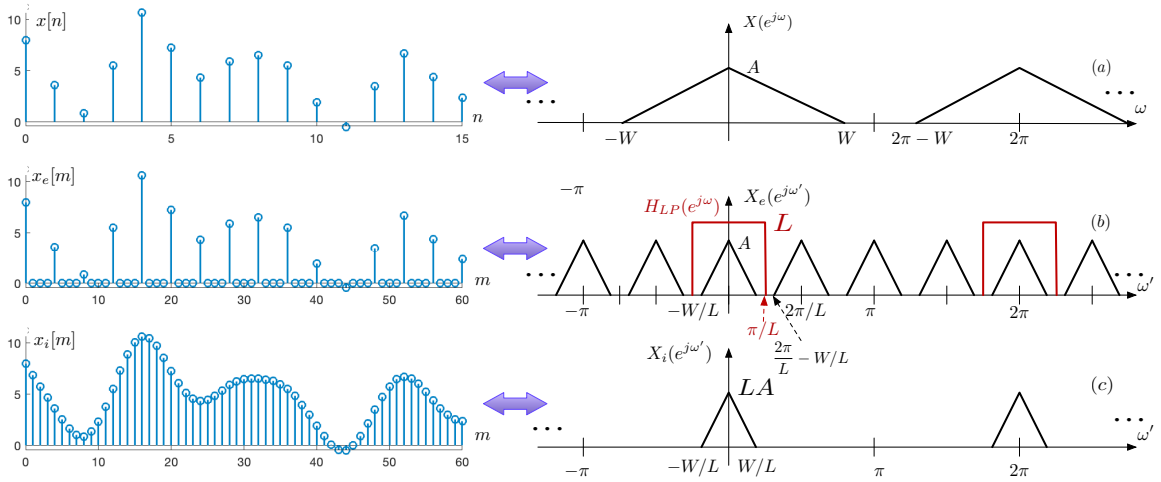


Figure 1.11: Functions involved in process of interpolation by a factor of $L = 4$ in the time and frequency domains. Left side: the time functions (a) $x[n]$, (b) $x_e[m]$, and (c) $x_r[m]$. Right side: The corresponding Fourier transforms in discrete time. The ideal lowpass filter $H_{LP}(e^{j\omega})$ is also shown in panel (b) in red.

The upsampling process is illustrated in the time and frequency domains in Fig. 1.11, which can be seen to retrace most of the functions shown in Fig. 1.9 in reverse order. The discrete-time sequence $x[n]$, depicted in the upper panel of Fig. 1.11 is first converted into

the expanded sequence of delta functions $x_e[m]$, which is depicted in the central panel of Fig 1.9, as describe above. It is easy to demonstrate that the $X_e(e^{j\omega'})$, the DTFT of $x_e[m]$ is identical to $X(e^{j\omega})$, the DTFT of $x[m]$ but contracted in frequency by a factor of L:

$$X_e(j\omega') = \sum_{m=-\infty}^{\infty} x_e[m]e^{-j\omega' m} = \sum_{m=-\infty, m=rL}^{\infty} x_d[m/L]e^{-j\omega' m} \tag{1.30}$$

Letting $l = m/L$ or $m = lL$, we obtain

$$X_e(j\omega') = \sum_{m=-\infty, m=rL}^{\infty} x[m/L]e^{-j\omega' m} = \sum_{l=-\infty}^{\infty} x[l]e^{-j\omega' lL} = \sum_{l=-\infty}^{\infty} x[l]e^{-j(\omega' L)l} = X(e^{j\omega' L}) \tag{1.31}$$

Lowpass smoothing. The output of the decimation process is obtained by passing the expanded signal $x_e[m]$ through the lowpass filter. In the time domain we convolve $x_e[m]$ with the unit sample response of the filter:

$$x_i[m] = x_e[m] * h_{LP}[m] \tag{1.32}$$

The unit sample response of the filter is easily obtained directly:

$$h_{LP}[m] = \frac{1}{2\pi}\int_{-\pi}^{\pi} H_{LP}(e^{j\omega})e^{j\omega n}d\omega = \frac{1}{2\pi}\int_{-\pi/L}^{\pi/L} Le^{j\Omega m}d\Omega = L\frac{\sin(\pi m/L)}{\pi m/L} \tag{1.33}$$

This is a discrete-time sinc function with amplitude equal to L at $m = 0$ and regularly-spaced zero crossings at $m = rL$. Because the convolution of any function with a train of impulses produces replication of the original function at the times of the impulses, weighted by the impulse areas, we obtain

$$x_i[m] = h_{LP}[m] * x_e[m] = h_{LP}[m] * \sum_{r=-\infty}^{\infty} x[r]\delta(m - rL) = L\sum_{r=-\infty}^{\infty} x[r]\frac{\sin(\pi(m - rL)/L)}{\pi(m - rL)/L} \tag{1.34}$$

In other words, the interpolated discrete-time function $x_i[m]$ can be represented as the sum of an infinite train of sinc functions in discrete time, delayed by intervals of L and scaled according to the corresponding value of $x[m]$. The interpolation process works exactly the same way as the reconstruction of continuous-time functions in D/C conversion, as illustrated in Fig. 1.11, except that the resulting function $x_i[m]$ is discrete in time rather than continuous in time.

As in C/D conversion, the actual lowpass filter to reconstruct the interpolated signal would not be ideal. Nevertheless, the interpolated signal would still be an infinite linear combination of delayed and scaled unit sample response of the lowpass filter that is actually used.

In the frequency domain, the the lowpass filter limits the frequencies passed to $|\omega'| \le \pi/L$, as depicted in red in the central panel of Fig. 1.11. As depicted in the lower panel of Fig. 1.11, the output frequency response is

$$X_i(e^{j\omega'}) = \begin{cases} X(e^{j\omega'}), & |\omega'| \le \pi/L \\ 0, & \pi/L < |\omega'| \le \pi \end{cases} \tag{1.35}$$

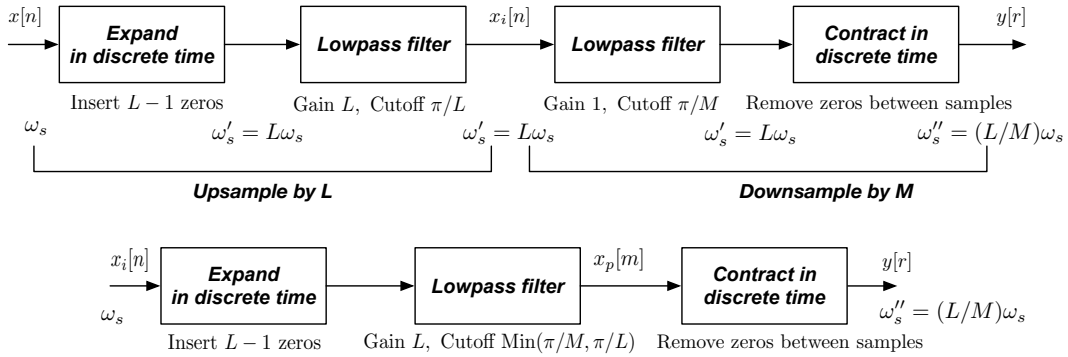


Figure 1.12: Block diagram of a system that changes the sampling rate by the rational fraction L/M. Upper panel: direct implementation by cascading interpolation by L with decimation by M. Lower panel: integrated system in which the cascade of two lowpass filters is replaced by a single lowpass filter.

1.4.3 Changing the sampling rate by a rational factor L/M

Changing the effective sampling rate by a rational factor of L/M can be thought of as first upsampling by L and then downsampling by M. For example, to change the sample rate of a speech signal from 20 kHz to 16 kHz, we would first upsample by $L = 4$ to 80 kHz and then downsample by $M = 5$ to 16 kHz. Note that 80 is the least common multiple of the upsampling and downsampling ratios L and M.

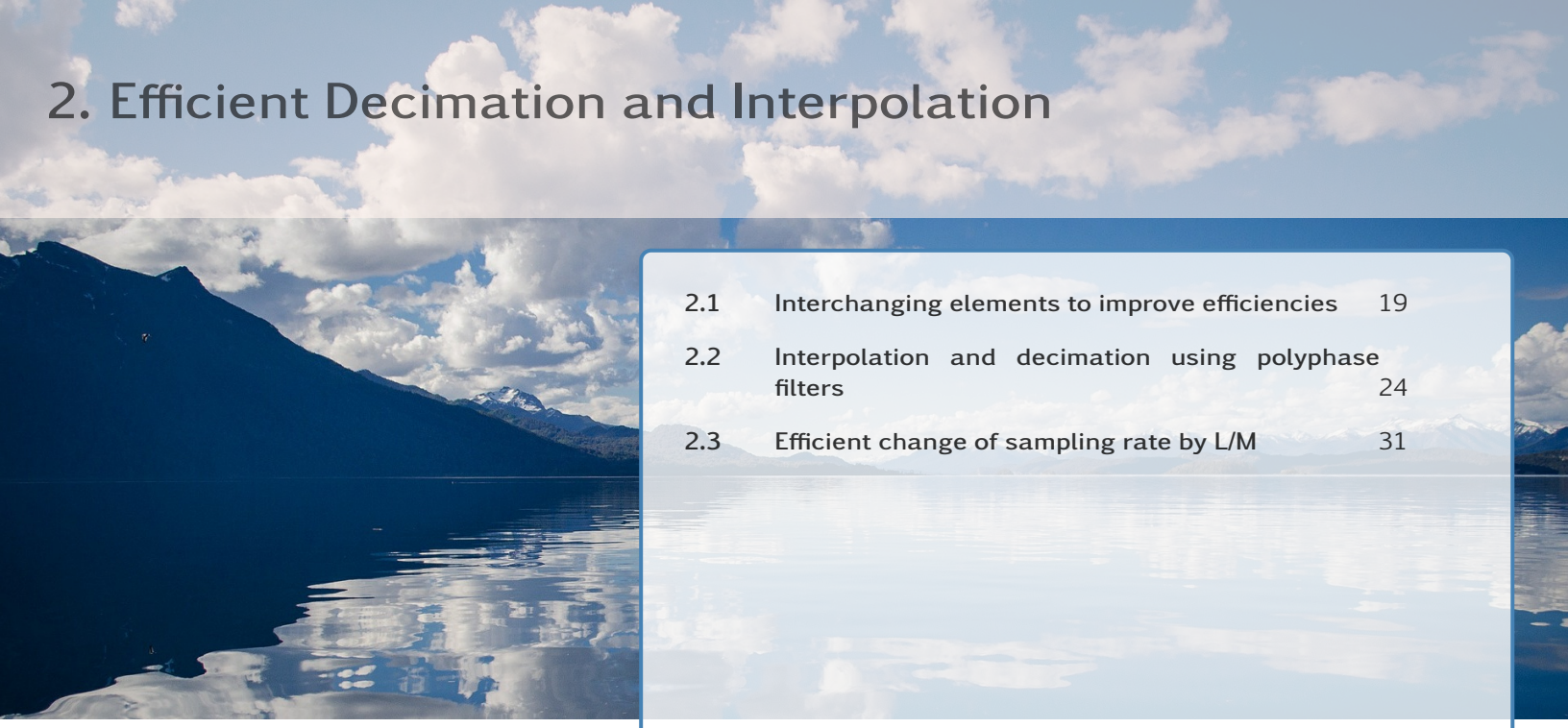
Figure 1.12 depicts two implementations of a system that changes the effective sampling rate by the fraction L/M. The upper panel shows the direct implementation of such a system, consisting of the cascade of upsampling by L followed by decimation by a factor of M. The lowpass antialiasing filter that is part of most practical decimation systems is included explicitly in the figure. In addition to the change in sampling rate, the amplitude of the input will be scaled by the factor of L/M by the processing. As discussed in the section the ideal antialiasing filter prior to decimation has a gain of 1 and a cutoff frequency of π/M. The lowpass filter that performs the interpolation in the upsampling module has a gain of L and a cutoff frequency of π/L. The initial block in the upper panel simply inserts $L-1$ zeros between each successive sample of the input. The final block contracts its input in time by a factor of M, preserving only those samples for which the input sample index is an integer multiple of M.

The lower panel of Fig. 1.12 depicts a more efficient implementation that replaces the cascade of the two ideal lowpass filters. It will have a gain of L and a cutoff frequency that is the minimum of π/L and π/M.

In summary, we have reviewed in this chapter the mathematics that are used to describe the processes of ideal conversion of signals from continuous time to discrete time and vice versa, known as ideal C/D and D/C conversion. We also discussed the basic mathematics that describe discrete-time change in sampling rate, including interpolation (or upsampling), decimation (or downsampling), and change of sampling rate by a rational fraction. We noted that decimation is similar to C/D conversion in a number of respects and that interpolation is similar to D/C conversion.

In the following chapter we will discuss techniques used to make the upsampling and downsampling operations much more computationally efficient.

2. Efficient Decimation and Interpolation



While the core decimation and interpolation procedures are easy to implement, the filters associated with them are not. This chapter discusses efficient ways of computing the algorithms that change the sampling rate of discrete-time signals.

The approaches introduced in the last chapter for changing the sampling rate are only useful insofar as they can be computed efficiently. This chapter discusses three general methods by which the computational load associated with upsampling, downsampling, and change of sampling rate can be very substantially reduced. We begin with a discussion of efficiencies enabled by simply interchanging the order of upsampling or downsampling with other elements of the filtering operations. We then introduce the topic of polyphase filtering, which enables us to combine these efficiencies with the use of fast Fourier transforms. Finally we discuss the special case of the implementation of algorithms that accomplish change in sampling rate by a rational fraction, which has an efficient solution based on filters with time-varying coefficients.[1]

2.1 Interchanging elements to improve efficiencies

The most straightforward approach to computational efficiency is to perform as much computation as possible on the low-frequency side of the system. We discuss how this can be done in this section.

2.1.1 Signal flowgraph notation

As you most likely already know, signal flowgraphs are an efficient graphical representation that enable us to easily design, interpret, and understand systems used in digital signal processing.

[1] This material is based on the corresponding discussions in Secs. 3.2 and 3.3. in Lim and Oppenheim (1988).

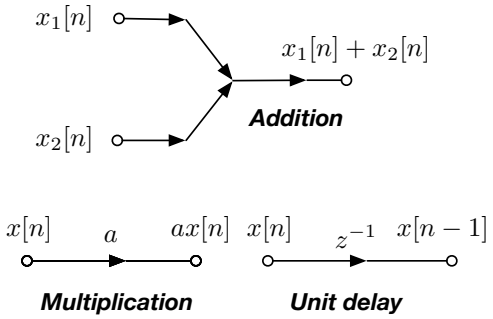


Figure 2.1: Signal flowgraph notation for the basic signal processing operations of addition, multiplication, and unit delay.

As an example, consider the difference equation that implements a generic causal LSI system:

$$y[n] = \sum_{k=1}^{N} a_k y[n-k] + \sum_{l=0}^{M} b_l x[n-l] \tag{2.1}$$

It can easily be shown that the corresponding transfer function of the filter is

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{l=0}^{M} b_l z^{-l}}{1 - \sum_{k=1}^{N} a_k z^{-k}} = \frac{B(z)}{A(z)} \tag{2.2}$$

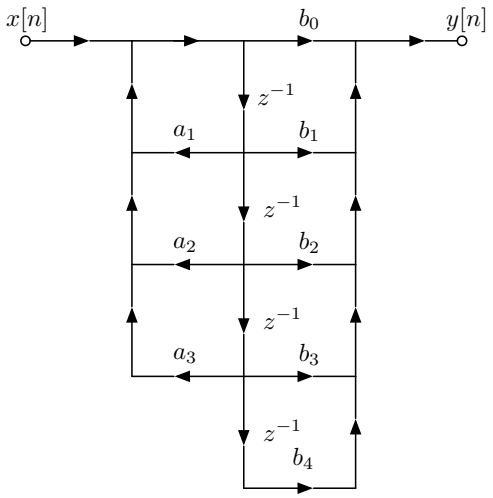


Figure 2.2: Flowgraph of a generic IIR filter with three poles and four zeros.

As is well known, the coefficients of the denominator polynomial $A(z)$ determine the pole locations while the coefficients of the numerator polynomial $B(z)$ determine the locations of the zeros of $H(z)$. We note that this very ubiquitous signal processing component is realized with only three distinct operations: addition, multiplication by a constant, and single-sample delay. Figure 2.1 shows the representation of these three operations in signal flowgraph form. As is discussed in basic DSP courses, there are multiple structures that implement Eq. (2.1) of which the most straightforward is the *direct form*, which is

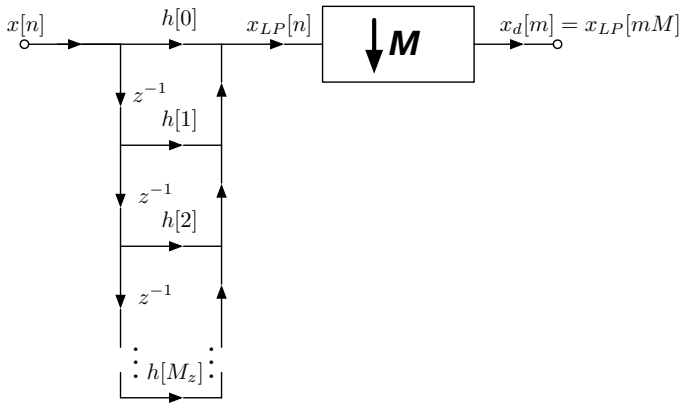


Figure 2.3: Realization of a simple decimator without any optimization for computational efficiency.

illustrated in Fig. 2.2 for $M = 4$ and $N = 3$. *Finite-impulse response* (FIR) filters are an important subclass of filters which have a unit sample response that lasts for only a finite number of samples, as the name implies. These filters are realized as in Eq. (2.1) and Fig. 2.2 but with all of the coefficients a_k set equal to zero. Note that FIR filters have feedforward sections but no feedback.

2.1.2 Efficient downsampling: the direct method

Now let us consider a simple decimation or downsampling system, which consists of an antialiasing filter followed by a device that compresses the time axis by a factor of M, discarding all samples except for $n = rM$. Specifically, $x_d[m] = x_{LP}[mM]$, as discussed in Sec. 1.4.1. Figure 2.3 illustrates such a system, using an FIR filter with an arbitrary number of zeros, M_z.[2]

It is straightforward to verify that the downsampling system in Fig. 2.3 incurs $M_z + 1$ multiplications and M_z additions for each sample that is input to the system. But it is equally easy to recognize that this type of organization is very wasteful, as the decimator immediately discards $M-1$ out of every M samples that appear at the output of the filter. To the extent possible, what we would *really* like to do is perform only the multiplications and additions that will actually remain after the downsampling operation.

Another way of thinking about this optimization is that we would like to perform as many multiplications and additions on the side of the low-frequency side of the decimation operation, which would imply interchanging the order of computation of the various processing elements depicted in Fig. 2.3 to do as much of the computation as possible after the downsampling. Figure 2.4 compares inputs and outputs for the three major signal processing operations depicted in Fig. 2.1. In this figure the input sequence is assumed to be $x[n] = n + 1$ and the second input to the addition operation is assumed to be $x_2[n] = 10(n + 1)$. The inputs and outputs for $n = 0, 1, 2$ are depicted explicitly in the

[2]We adopt in this section only the notational convention of using the variable M_z to designate the number of zeros in the FIR filter, as opposed to the customary M. This is to avoid confusion with the use of M to indicate the downsampling ratio.

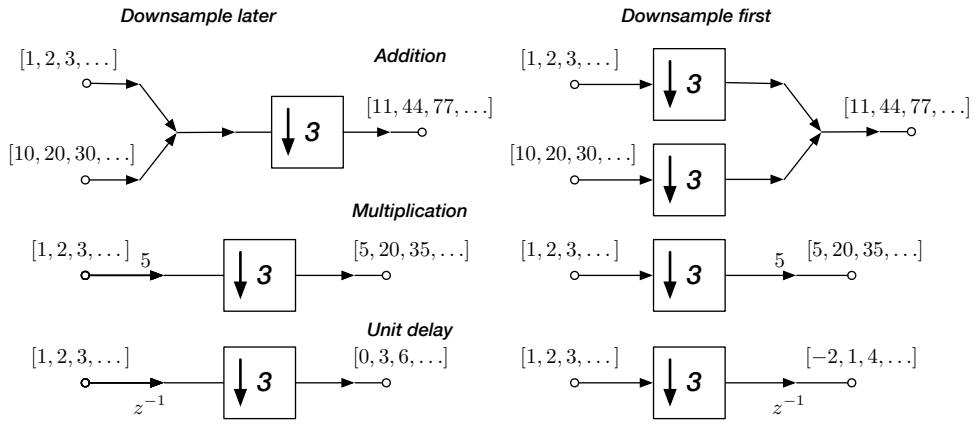


Figure 2.4: Comparison of inputs and outputs of the three signal processing operations when they are followed by or preceded by downsampling.

figure. It can be easily seen that interchanging the downsampling operation with addition or multiplication has no effect on the output, but interchanging the unit delay with the downsampling does change the output sequence. Hence, we can freely interchange the downsampling operation with addition and multiplication but not with delay. Figure

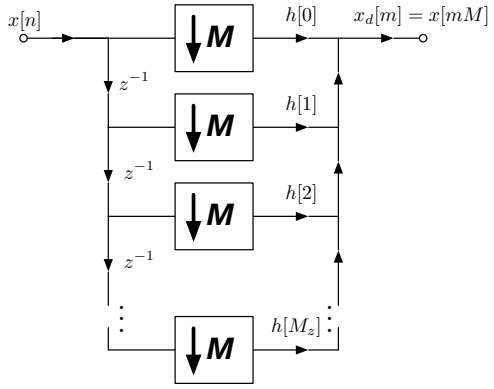


Figure 2.5: Realization of an efficient decimator by interchanging system elements.

2.5 depicts a more efficient realization of the decimation operation obtained by performing the decimation before the multiplication and addition operations. It can be seen that while the original decimator structure in Fig. 2.3 incurs approximately $M_z + 1$ multiplications and M_z additions per input point, the reconfigured structure shown in Fig. 2.5 reduces the number of multiplications and additions per input point by a factor of M by the simple artifice of re-ordering the components of the decimator.

2.1.3 Efficient upsampling using transposed filter forms.

Figure 2.6 is a signal flowgraph of a standard interpolation system. As we discussed in Sec. 1.4.2, the initial box is a direct interpolator, which inserts $L-1$ samples of amplitude zero between each successive input point. This is followed by an ideal lowpass filter, which is shown as a standard direct-form implementation of an arbitrary FIR filter in Fig. 2.6. Unfortunately, there is no opportunity to interchange the upsampling operation

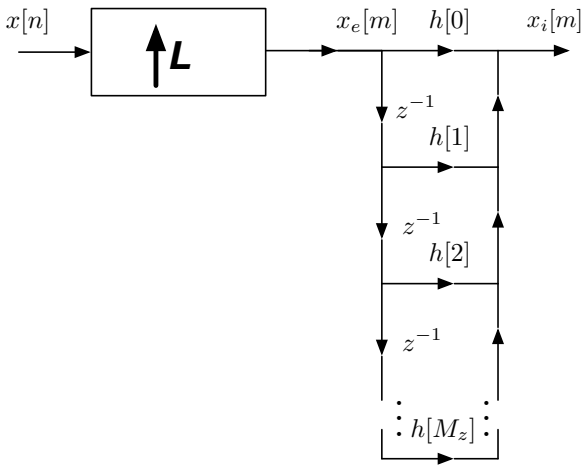


Figure 2.6: A simple interpolation system using the direct-form implementation of the FIR lowpass filter.

with any other operation to place the multiplications and additions on the low-frequency side because in virtually all branches the upsampling operation would need to be interchanged with a delay block before reaching the multiplications and additions, which would change the output.

This problem can be overcome by using the *transposed-form* implementation of the lowpass filter. Specifically, the network transposition theorem by S. J. Mason states that if one were to reverse the direction of flow of the arrows in the flowgraph with a single input and a single output, and interchange the input and output, the output of the network would remain the same for any input.

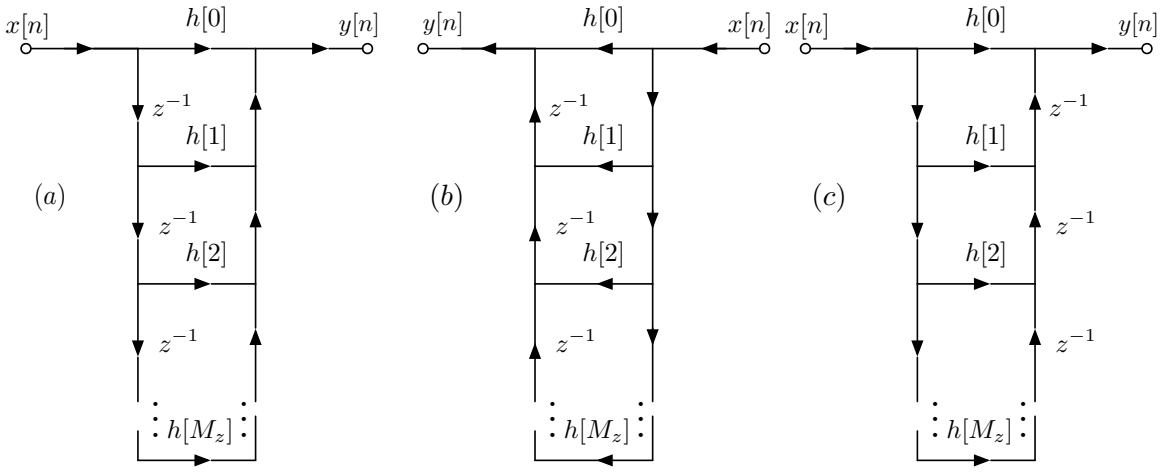


Figure 2.7: Implementation of transposition for a simple FIR network. (a) The original direct-form implementation of the filter. (b) The network after reversing the directions of signal flow and interchanging input and output. (c) The network of panel (b) redrawn so that the signals flow from left to right.

Figure 2.7 illustrates these operations for a simple direct-form FIR filter. Panel (a) shows

the original direct-form implementation of the FIR filter. Panel (b) shows the effect of interchanging input and output and reversing the direction of the signal flow. Panel (c) is simply Panel (b) redrawn so that the signal flow is left to right, producing the transposed-form realization that is very similar to the original direct form except that the delays are now on the output side of the structure. It is easily verified that the outputs of the networks in panels (a) and (c) are identical for any input.

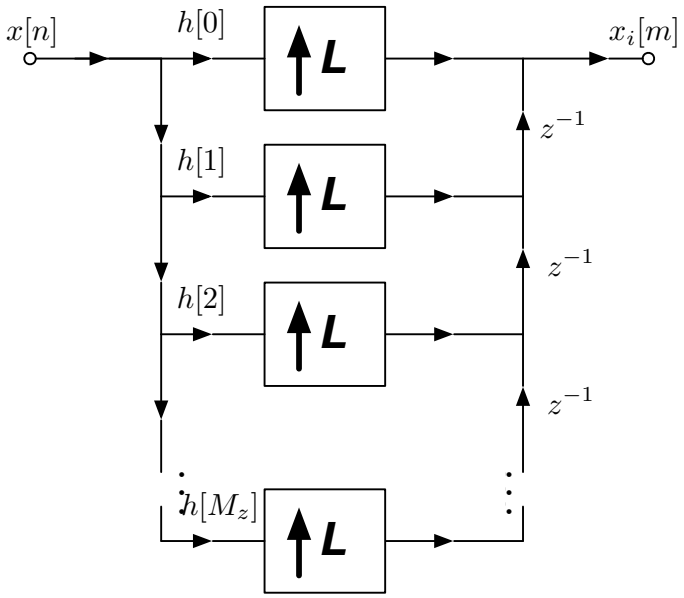


Figure 2.8: Realization of an efficient interpolator by interchanging system elements.

Figure 2.8 shows the signal flowgraph of an efficient implementation of the interpolation operation, developed by interchanging the upsampling operation with the multiplications, reducing the number of multiplications by a factor of L by moving them over to the low sampling-rate side, although the number of additions remains unchanged.

By comparing Fig. 2.8 with Fig. 2.5 (or by comparing 2.6 with Fig 2.3), it can be seen that the interpolation and decimation operations can be thought of as network transposes of *each other*, provided that the basic downsampling operation is converted into an upsampling operation, and vise versa, in performing the network transpositions. We will make use of this property in the section below in our discussion of polyphase implementations.

2.2 Interpolation and decimation using polyphase filters

Consider the block diagram in Fig. 2.9, which is a generic specification of a system that increases the sample rate by a factor of L, as discussed in Chapter 1. The input $x[n]$ can

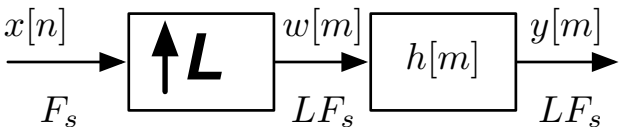


Figure 2.9: Block diagram of a system that interpolates by a factor of L.

be any signal of any bandwidth. As before, the expanded version of the signal $w[m]$ is defined as follows:

$$w[m] = \begin{cases} x[m/L], & m/L \text{ integer} \\ 0, & \text{otherwise} \end{cases} \tag{2.3}$$

Note that the effective frequency of the output time index m is L times that of the input time index n. The filter $h[m]$ is ideally a lowpass filter with cutoff frequency π/L and gain L. The input and output of the filter are related (as usual) by the convolution sum

$$y[m] = \sum_{k=-\infty}^{\infty} h[m-k]w[k] = \sum_{k=-\infty,\ k=rL}^{\infty} h[m-k]x[k/L] \tag{2.4}$$

Let us define $r = k/L$ or $k = rL$. We can then rewrite the convolution as

$$y[m] = \sum_{r=-\infty}^{\infty} h[m-rL]x[r] \tag{2.5}$$

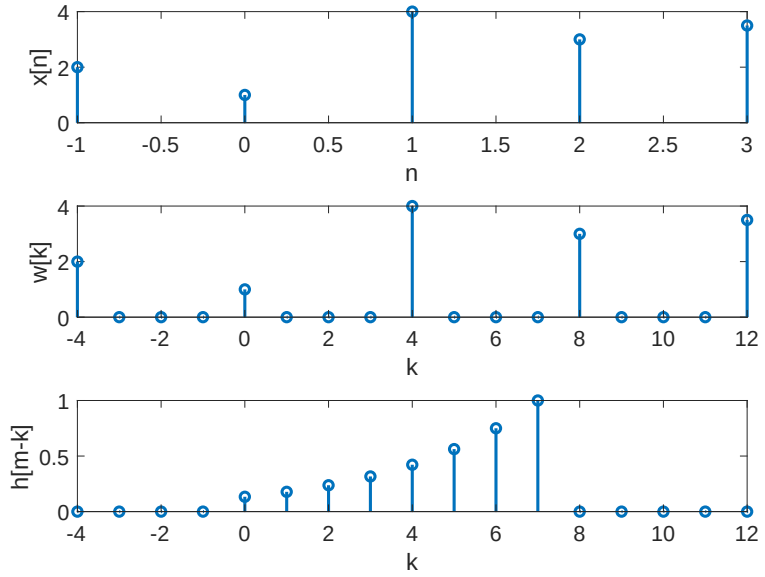


Figure 2.10: Examples of the functions $x[n]$, $w[k]$, and $h[m-k]$ for a value of $m = 7$. The filter $h[m]$ is assumed to be FIR and of length $N = 8$.

At this point it is helpful to get an idea of what these functions look like on the ground. Figure 2.10 above plots representative examples of the functions $x[n]$, $w[k]$, and $h[m-k]$

for $m = 7$ using a simple FIR lowpass filter with length $N = 8$ and a unit sample response equal to

$$h[n] = \begin{cases} (3/4)^n, & 0 \le n \le 7 \\ 0, & \text{otherwise} \end{cases} \tag{2.6}$$

Note that for $m = 7$, the only elements involved in the computation of the result are $h[3]$ (which multiplies $x[1]$) and $h[7]$ (which multiplies $x[0]$). The previous output sample, $y[6]$. would have been obtained by multiplying two other filter coefficients $h[2]$ and $h[6]$ by the same two values of $x[n]$, $x[1]$ and $x[7]$, respectively. We will elaborate below on this observation that successive output points are obtained by multiplying the same input points by different filter coefficients.

It is convenient for us to redefine r (without changing anything) as

$$r = \left\lfloor \frac{m}{L} \right\rfloor - n \tag{2.7}$$

where n is now a new auxiliary variable. Recall that the floor operator $\lfloor m/L \rfloor$ denotes the largest integer that is less than or equal to m/L. (Note that this n is not the time index for the input. The duplication of notation is unfortunate, but is used to maintain consistency with the notation of the Lim and Oppenheim chapter.)

Note that

$$m - \left\lfloor \frac{m}{L} \right\rfloor L = ((m))_L \tag{2.8}$$

where $((m))_L$ represents m modulo L. Substituting Eq. (2.7) into Eq. (2.5) we obtain

$$y[m] = \sum_{n=-\infty}^{\infty} h\left[m - \left(\left\lfloor \frac{m}{L} \right\rfloor L - nL\right)\right] x\left[\left\lfloor \frac{m}{L} \right\rfloor - n\right] = \sum_{n=-\infty}^{\infty} h[nL + ((m))_L] x\left[\left\lfloor \frac{m}{L} \right\rfloor - n\right] \tag{2.9}$$

Equations (2.4), (2.5), and (2.9) reflect the fact that with the ideal interpolator, only $1/L$ of the filter coefficients are involved in the computation of any given output sample. Generalizing the example shown in Fig. 2.10, Table 2.1 below summarizes which values of $x[n]$ and $h[m]$ are involved in the computation of each output sample $y[m]$ for situations in which the filter $h[n]$ is assumed to be FIR with length $N = 8$.

Note that the coefficients of $h[m]$ used to calculate the output point $y[m]$ rotate cyclically with increasing m with period L, while the coefficients of $x[n]$ are fixed for L successive output points. For more realistic filters with a larger value of N, N/L coefficients will be involved in the calculation of a given output point $y[m]$, but they will still rotate cyclically with period L. This phenomenon can be interpreted in two different ways, as discussed below.

2.2.1 Interpolation using polyphase filters

Consider a set of filters $p_L[n]$ that are obtained from the lowpass filter $h[n]$ in Eq. (2.9), advancing the samples in time, and then decimating them by a factor of L. Specifically, we define the polyphase filters $p_l[n]$ to be

$$p_l[n] = h[nL + l] \text{ for } 0 \le l \le L - 1 \tag{2.10}$$

m	$x[n]$	$h[m]$
0		$h[0], h[4]$
1	$x[0], x[-1]$	$h[1], h[5]$
2		$h[2], h[6]$
3		$h[3], h[7]$
4		$h[0], h[4]$
5	$x[1], x[0]$	$h[1], h[5]$
6		$h[2], h[6]$
7		$h[3], h[7]$
8	$x[2], x[1]$	$h[0], h[4]$

Table 2.1: Intermediate values in upsampling.

Fig. 2.11 provides an example for how polyphase filters are derived for the simple case of $N = 9$ and $L = 3$. The upper graph represents the original unit sample response $h[n]$, which is designed to approximate an ideal lowpass filter with cutoff frequency π/L. The lower three graphs in the left panel depict the three polyphase filters that are derived from $h[n]$, $p_l[n]$ for $0 \le l \le L-1$, with the correspondence between the samples of $h[n]$ and $p_l[n]$ indicated by the colors red, blue, and green, respectively. The efficient interpolation is accomplished by inputing the low-sample-rate input $x[n]$ to all L polyphase filters simultaneously, upsampling the outputs, then delaying the outputs successively by one sample, and finally adding the resulting signals to produce the output $y[m]$ defined by Eq. (2.9). This structure is illustrated in the left panel of Fig. 2.12 below.

An alternative interpretation of polyphase interpolation is depicted in the right panel. Here the upsampled output $y[m]$ is obtained from the outputs $y_l[m]$ of the successive polyphase filters using a switch (sometimes called a *commutator*) that rotates cyclically with period L. After L outputs are obtained, the input $x[n]$ is updated by one sample. In either implementation of polyphase interpolation, there are L times as many output samples as input samples. Each output sample is obtained from the output of a different polyphase filter in succession, and after we get to the output of $p_{L-1}[n]$ we return to the output of $p_0[n]$ but with a new input sample.

Note that this structure is efficient using either implementation because the filter computations are performed at the low sample rate. In fact, the polyphase approach provides the same reduction of computation that was obtained using the direct approach that interchanges the processing element, as described in Sec. 2.1. Nevertheless, the polyphase implementations also enables further efficiencies. Because the filter structure itself remains intact using the polyphase implementations, the convolutions can be implemented using Fast Fourier transforms combined with the overlap-add or overlap-save algorithm.

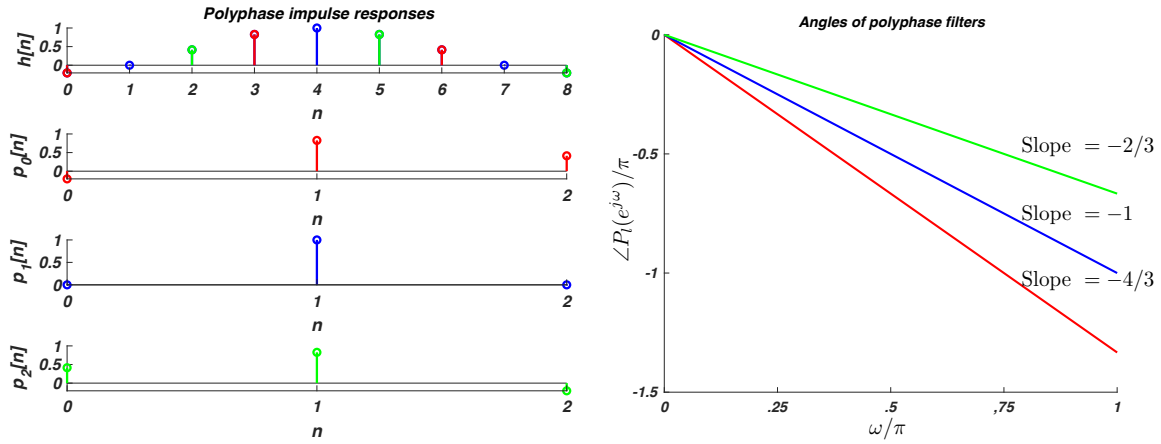


Figure 2.11: Left panel: Development of polyphase filters $p_l[n]$ by decimating an original filter $h[n]$ illustrated for $L = 3$ and $N = 9$. Right panel: phase responses for the three polyphase filters shown in the left panel. Magnitudes are equal to 1 for all ω.

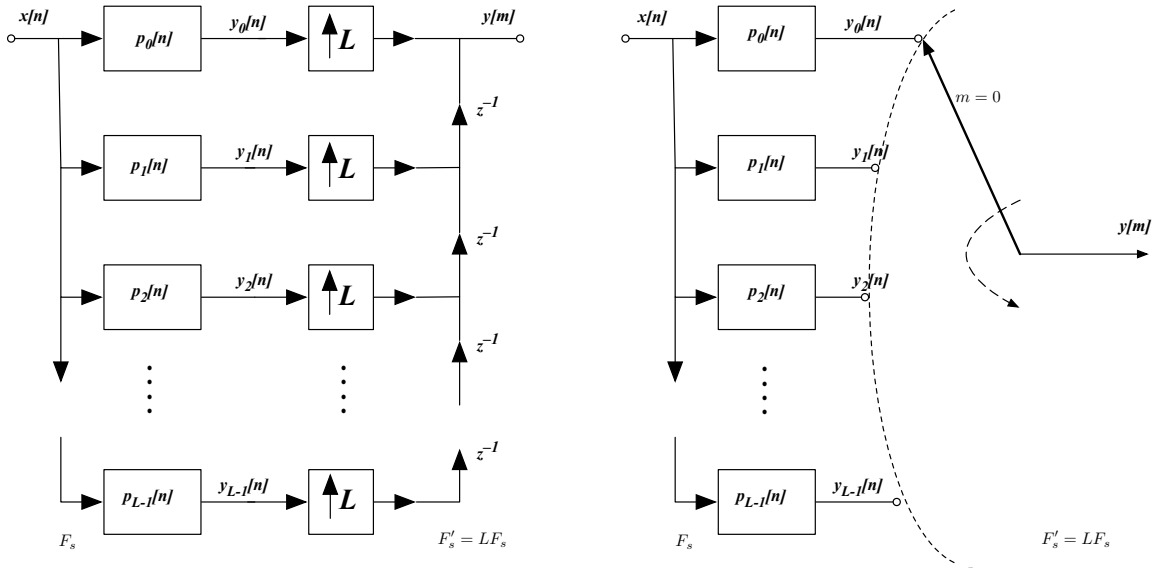


Figure 2.12: Left panel: Block diagram of interpolation by a factor of L using polyphase filters. Right panel: alternate implementation using a rotary switch.

It is worthwhile to consider the frequency responses of the individual polyphase filters. As described in Eq. (2.10), the polyphase filters are derived from the prototype filter by advancing the sample response of $h[n]$ successively for each value of l and then decimating the resulting response by L. The prototype filter is assumed to be an ideal lowpass filter with magnitude of the form

$$H(e^{j\omega}) = \begin{cases} L, & |\omega| \le \pi/L \\ 0, & \pi/L < |\omega| \le \pi \end{cases} \tag{2.11}$$

Decimating the sample response by L expands the frequency response by a factor of L,

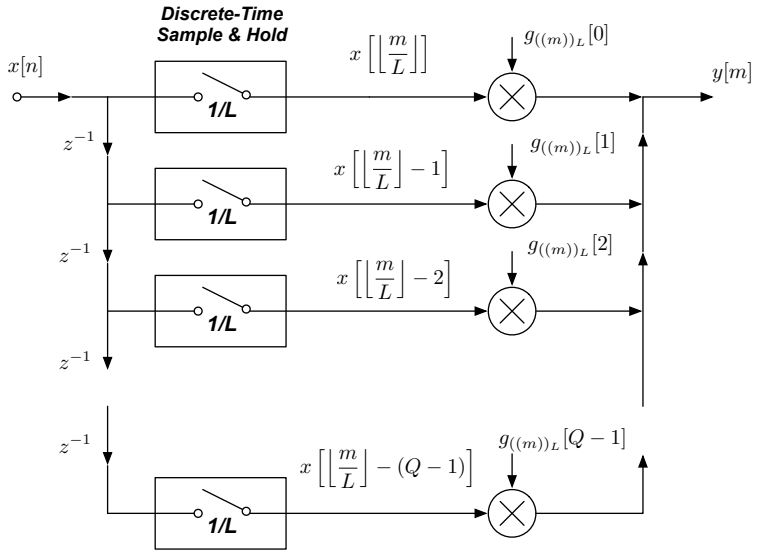


Figure 2.13: Block diagram of interpolation by a factor of L using filters with time-varying coefficients.

which causes $|P_l(e^{j\omega})|$ to be allpass with gain 1 for all frequencies. The phase of the polyphase filters is different for each filter. Let us assume that the original prototype lowpass filter $h[n]$ is implemented as an FIR linear-phase lowpass filter of length N. This means that its unit sample response will be symmetric about the sample point $n = (N-1)/2$ (which equals 4 in Fig. 2.11), causing the phase response to be

$$\angle P_l(e^{j\omega}) = e^{-j\frac{\omega}{L}\left(\frac{N-1}{2}-l\right)} \tag{2.12}$$

where l represents the index of the polyphase filter for $0 \le l \le L-1$, N is the length of the FIR filter, and L is the upsampling ratio, as before.

In other words, the decimated filters $p_l[n]$ will each have different linear phase shifts, which is of course why the filters are collectively referred to as "polyphase filters." The right panel of Fig. 2.11 shows the phase responses of the polyphase filters for $N = 9$ and $L = 3$.

2.2.2 Implementation of interpolation using a filter with time-varying coefficients

We can rewrite Eq. 2.9 as

$$y[m] = \sum_{n=-\infty}^{\infty} g_m[n]\, x\left[\left\lfloor \frac{m}{L} \right\rfloor - n\right] \tag{2.13}$$

where

$$g_m[n] = h[nL + ((m))_L] \tag{2.14}$$

Note again that the coefficients $g_m[n]$ vary cyclically with period L. Equations (2.13) and (2.14) suggest that the convolution of $x[n]$ with $h[n]$ to produce the upsampled $y[m]$ can be thought of as a convolution of the flipped and shifted version of $x[n]$ with a filter that

has time-varying coefficients, with the pattern of coefficients repeating after L successive samples of $y[m]$ are output. The input function $x[n]$ is input at the low sampling rate F_s while the output $y[m]$ emerges at the higher sampling rate, $F_s' = LF_s$. The alternate form with time-varying coefficients is depicted in Fig. 2.13 below.

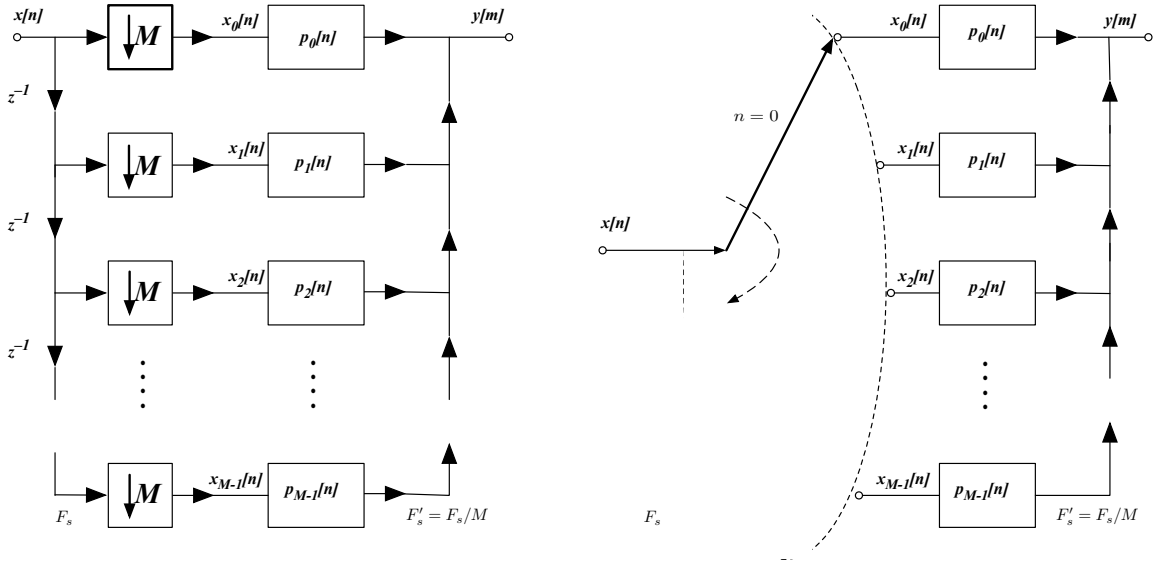


Figure 2.14: Left panel: Block diagram of decimation by a factor of M using polyphase filters. Right panel: alternate implementation using a rotary switch.

Note that in Fig. 2.13, the system samples the values of $x[n]$ and holds them, updating the samples after L successive values of the output $y[m]$ are generated. We assume that the length of $h[n]$, N, is an integer multiple of the upsampling ratio L, and specifically $N = QL$ or $Q = N/L$. In this rendering, the number of rows in the block diagram is equal to Q, which is also the length of the polyphase filters. In fact, for each value of the output time index m, the vertical column of coefficients $g_{((m))_L}$ is identical to the coefficients of the polyphase filter $p_l[n]$ that would be used to produce the value of $y[m]$ in question. After L successive output values, the samples of $x[n]$ at the left shift downward, and a new value of $x[n]$ is input.

2.2.3 Decimation using polyphase filtering

In our discussions about polyphase implementations we have focussed on interpolation because the mathematics are more intuitive. Nevertheless, polyphase forms for decimators exist as well, using both the original polyphase filters themselves and the implementation using a single filter with time-varying coefficients.

The easiest way to develop the polyphase filter form for decimation is through the use of the network transposition theorems. Specifically, we noted in Sec. 2.1.3 that the transpose of a signal-flow network is obtained by reversing the directions of the arrows and interchanging input and output. And if in addition we reverse the direction of the downsampling and upsampling arrows in decimation and interpolation, the transpose of interpolation becomes decimation and vice versa. Fig. 2.14 depicts the signal flow-graph structure of the polyphase decimator, both in the original form and in the modified form using a rotary switch. In the latter case, the rotary switch operates at the input sampling

frequency and applies each input sample to the inputs of each of the polyphase filters in succession. A single output is generated from each of M successive inputs, causing the output sampling frequency to be $1/M$ times the input sampling frequency.

2.3 Efficient change of sampling rate by L/M

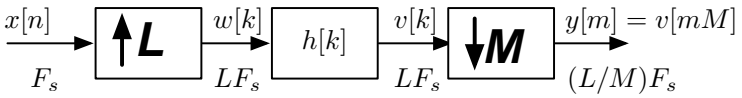


Figure 2.15: Block diagram of a system that interpolates by L and decimates by M.

A system that changes the sampling rate by the ratio L/M is illustrated in Fig. 2.15 above. In comparing this figure to that of interpolation by L (Fig. 2.9), we note that the output of the interpolator is now replaced by the function $v[k]$ and that $y[m] = v[mM]$. In other words, the output for a system that changes the sampling rate by the ratio L/M is identical to that of the output of an interpolator by a factor of L except that the time argument m in the block diagram for the interpolator replaced by mM. This implies that for the case of a rational change of sampling rate, Eqs. (2.13) and (2.14) can be rewritten as

$$y[m] = \sum_{n=-\infty}^{\infty} g_m[n] x\left[\left\lfloor \frac{mM}{L} \right\rfloor - n\right] \tag{2.15}$$

where

$$g_m[n] = h[nL + ((mM))_L] \tag{2.16}$$

$y[m]$	$x[\lfloor \frac{mM}{L} \rfloor]$	$g_m[0]$
m	$\lfloor \frac{2m}{3} \rfloor$	$((2m))_3$
0	0	0
1	0	2
2	1	1
3	2	0
4	2	2
5	3	1
6	4	0

Table 2.2: Timing relationships between $y[m]$ and $x[n]$ for $M = 2$ and $L = 3$.

To get some insight about the meaning of the arguments in Eqs. (2.15) and (2.16) as they vary in time, let us consider the very simple case of $M = 2$ and $L = 3$. Table 2.2 describes how the key arguments in Eqs. (2.15) and (2.16) vary with changes in m. By comparing

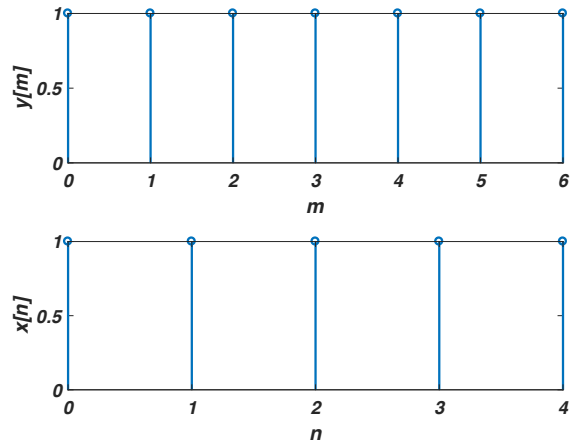


Figure 2.16: Comparison of two signals with a sampling-rate ratio of 3/2.

the entries of Table 2.2 with the arrival times of the samples of $x[n]$ and $y[m]$ in Fig. 2.16, it can be seen that the argument $\lfloor mM/L \rfloor$ in Eq. (2.15) denotes the sample of $x[n]$ that arrived most recently for each value of $y[m]$. The coefficients for $g_{((m))_L}$ rotate cyclically, as before.

This in turn implies that the efficient structure for the change of sampling rate by the ratio L/M is only slightly different from the structure in Fig. 2.13 for interpolation by L, as depicted in Fig. 2.17 below. The only change from the interpolator system in Fig. 2.13 is that the input is now sampled at the new input rate, which varies asynchronously with the output rate. The multiplication by the coefficients $g_m[n]$ is as before.

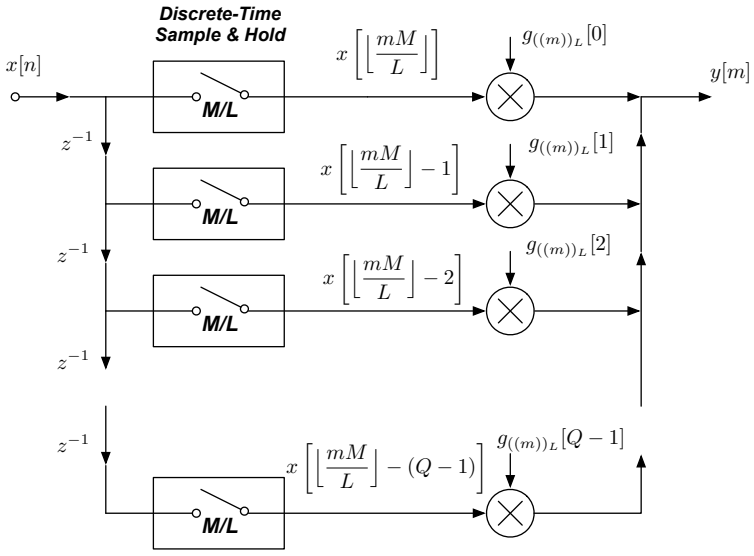
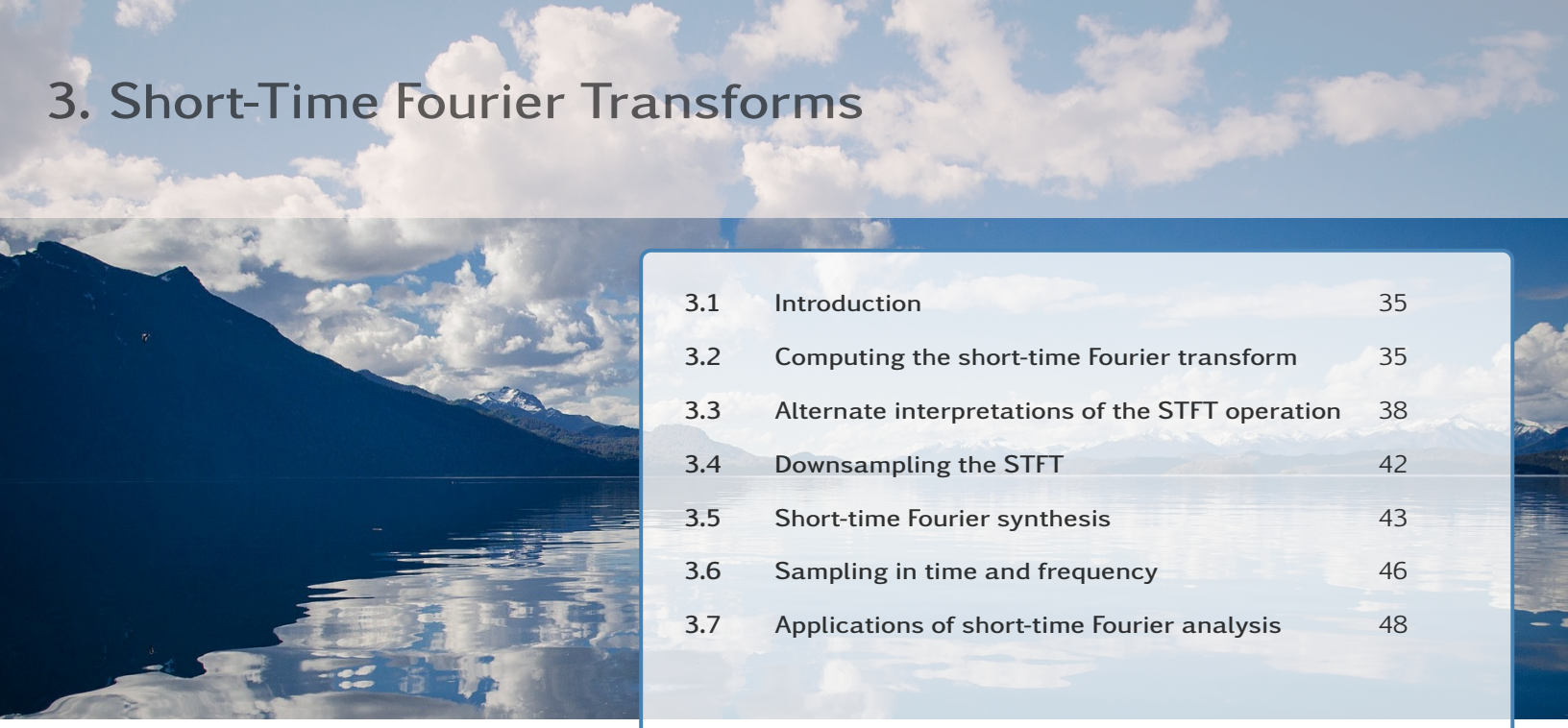


Figure 2.17: Block diagram of an efficient system that upsamples by L and downsamples by M.

In summary, we have discussed three ways of decreasing the computation associated with decimation and interpolation, typically by factors on the order of M or L. The direct interchange-based approach is especially useful when non-standard filter implementations (such as the linear phase form) are used. The polyphase implementation is especially valuable because it enables the use of fast Fourier transform techniques in performing the actual filtering. The method involving time-varying coefficients is especially useful in implementing changes of sampling rate by the rational factor L/M.

3. Short-Time Fourier Transforms



Traditional Fourier transforms describe the frequency components in a signal averaged over all time. The most important and informative aspects of signals like speech and music, however, is how these frequency components evolve over time. In this chapter we discuss short-time Fourier transforms, which enable us to characterize signals with time-varying frequency components.

3.1 Introduction

While frequency-domain representations such as the DTFT and the DFT are useful, they both are obtained by summing the time function $x[n]$ from $-\infty$ to ∞. This means that the DTFT and DFT describe frequency components in the signal averaged over all time. Interesting signals like music and speech are characterized the ways in which frequency components change over time. (These components could represent objects such as the phonemes that constitute a spoken word or the individual notes that constitute a musical composition.) These observations motivated the development of the *short-time Fourier transform* (STFT).[1]

3.2 Computing the short-time Fourier transform

The STFT considers only a short-duration segment of a longer signal and computes its Fourier transform. Typically this is accomplished by multiplying a longer time function $x[n]$ by a window function $w[n]$ that is brief in duration. Two commonly-used finite-duration windows are the rectangular window, which essentially extracts only the desired short sequence without further modification, and the Hamming window, which applies a taper to the ends to improve the representation in the frequency domain. If the continu-

[1] Most of these discussions follow the treatment of the material in the books on speech signal processing by Rabiner and Schafer (1978, 2010).

ous frequency variable ω is used (as in the DTFT), the STFT can be described as

$$X[n,\omega] = \sum_{m=-\infty}^{\infty} w[n-m]x[m]e^{-j\omega m} \tag{3.1}$$

In principle, the window could be either finite or infinite in duration, and in the latter case exponential windows are popular. The delimiters in $X[n,\omega)$ are unconventional, of course, and this usage is intended to highlight the fact that the frame index n is discrete while the frequency variable ω is continuous.

In practice, it is common to evaluate the STFT at only a finite set of equally-spaced points along the frequency axis, just as the DFT is frequently used instead of the DTFT in conventional applications of digital signal processing. We will use the variable N to specify the number of discrete frequency channels used in the STFT, and the variable N_w to specify the length of the window function $w[n]$ when it is finite in duration. Using these notational conventions, the frequencies over which the STFT is evaluated become $\omega_k = 2\pi k/N$. With a finite-duration window with nonzero values of n from 0 to $N_w - 1$, the STFT equation becomes

$$X[n,k] = \sum_{m=n-(N_w-1)}^{n} w[n-m]x[m]e^{-j\omega_k m} = \sum_{m=n-(N_w-1)}^{n} w[n-m]x[m]e^{-j2\pi mk/N} \tag{3.2}$$

Note that $X[n,k]$ is a function of both time and frequency and now both the time and frequency variables are discrete. The variable n denotes the location of the analysis window along the time axis, and the segment of time delimited by the window is frequently referred to as the *analysis frame*. The variable k is a frequency index, and is sometimes referred to as a *frequency bin*. We can think of the STFT as representing the DFT of the finite-duration time function $x[m]w[n-m]$. Here the variable m is a "dummy" time argument and the variable n identifies the location of the short segment of the original time function as it is extracted using the window $w[n-m]$, which moves along the m-axis according to the value of n.

3.2.1 Impact of window size and shape

Let us begin by turning our attention to the impact of the window shape and duration on the nature of the STFT. We can formalize the interaction between the original time function, the window size and shape, and the resulting STFT as follows. Considering first the continuous-frequency version of the STFT, recall that $X[n,\omega)$ is the DTFT of the input function $x[n]$ multiplied by the window function. The Fourier transform of $w[n-m]$ is

$$w[n-m] \Leftrightarrow \sum_{m=-\infty}^{\infty} w[n-m]e^{-j\omega m} \tag{3.3}$$

Letting $l = n-m$ and $m = n-l$ we obtain

$$w[n-m] \Leftrightarrow \sum_{l=-\infty}^{\infty} w[l]e^{-j\omega(n-l)} = e^{-j\omega n}W(e^{-j\omega}) \tag{3.4}$$

Hence,

$$w[n-m]x[m] \Leftrightarrow \frac{1}{2\pi}(W(e^{-j\omega})e^{-j\omega n}) \circledast X(e^{j\omega}) \tag{3.5}$$

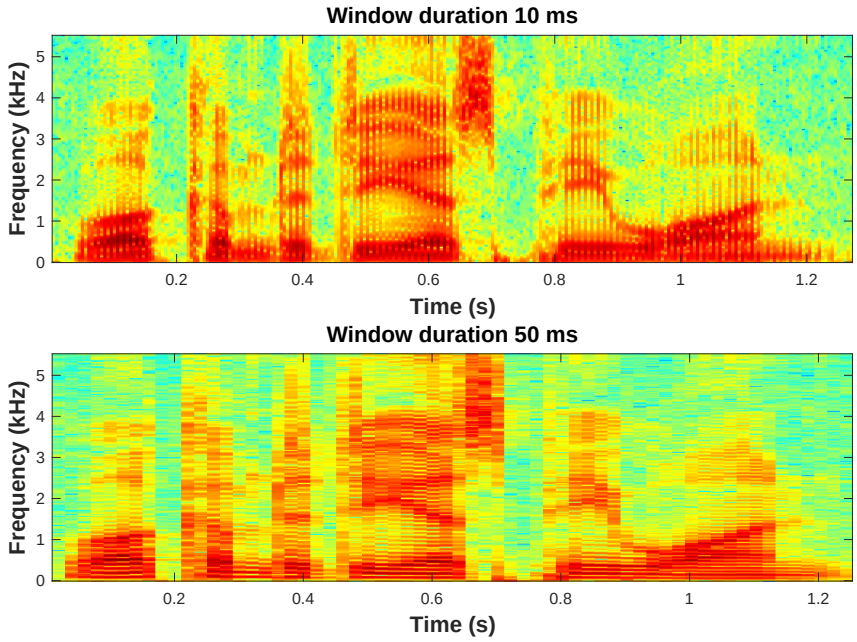


Figure 3.1: Impact of window duration on the STFT. A spectrogram of a brief utterance is shown, using Hamming windows of duration 10 ms (upper panel) and 50 ms (lower panel).

where the symbol $\circledast$ indicates circular convolution.

In other words, the STFT can also be thought of as the circular convolution in frequency of the Fourier transform of the original input signal with the Fourier transform of the window function time reversed and shifted. Because briefer functions in time produce broader Fourier transforms in frequency, the use of a brief analysis window $w[n]$ will give us good temporal resolution at the expense of a lot of blurring in frequency, while a broader temporal window will provide sharp spectral resolution at the expense of reduced temporal resolution. This applies to the discrete-frequency implementation of the STFT as well.

The tradeoff between temporal and spectral resolution is illustrated in Fig. 3.1. Figure 3.1 shows two examples of a *spectrogram* of a brief utterance ("Welcome to DSP-I") spoken by the author. The horizontal axis represents time while the vertical axis represents frequency. For now it is sufficient to note that the speech waveform can be modeled by a filtered pulse train called *glottal pulses*, which are generated by the vocal chords with time-varying fundamental frequency. The glottal pulses are input to an acoustic filter with time-varying frequency response that is shaped by the configuration of the throat, tongue, and lips, etc. (It should also be noted that some phonemes such as /s/ , /f/, and /th/ are produced by exciting the acoustic filter with broadband noise instead of the quasi-periodic glottal pulses.) As the colors in the display go from blue to green to yellow to orange to red for a particular spectro-temporal element, the power of the signal becomes greater for that frequency and time. In fact, the desire to analyze, display, interpret, and manipulate the time-varying characteristics of speech and music in a useful fashion has been the prime motivation toward the development of the mathematics that

are the basis for the STFT.

In the example of Fig. 3.1, we can observe that the /c/ sound in "welcome" occurs just after 0.2 seconds and the /s/ in "DSP" occurs at about 0.65 seconds. The window functions used in the figure are Hamming windows of duration 10 ms (upper panel) and 40 ms (lower panel). The fundamental frequency of the vocal tract pulses varies, but it is about 100 Hz, and the corresponding period is about 10 ms. The windows are overlapped by 50 percent for reasons to be discussed below. The image in the upper panel appears to show vertical bars, which occur because the window duration is comparable to the 10-ms period of the glottal pulses, so some of the so-called *analysis frames* occur at the time of the glottal pulses and some of them occur between them. In contrast, horizontal bars are seen in the lower panel of Fig. 3.1, which is computed using windows of duration 50 ms. In this case, the window duration is long enough so that the window smears over successive glottal pulses, but the frequency resolution is now sufficiently fine that the horizontal bars appear at analysis frequencies that are multiples of the fundamental frequency, which is about 100 Hz. It can be seen that the separation of the vertical bars in the upper panel of Fig. 3.1 is approximately .01 seconds and the separation of the horizontal bars in the lower panel is approximately 100 Hz.

In practice, the duration of the analysis window is set according to the needs of the application. For example, the window duration is typically between 20 and 35 ms for automatic speech recognition, but longer than that (75-120 ms) for speaker identification.

3.2.2 Inversion of the STFT

As we have stated, we can think of the STFT as the Fourier transform of the windowed time function:

$$x[m]w[n-m] \Leftrightarrow X[n,\omega) \text{ so} \tag{3.6}$$

$$w[n-m]x[m] = \frac{1}{2\pi}\int_{-\pi}^{\pi} X[n,\omega)e^{j\omega m}d\omega \tag{3.7}$$

For $n = m$ we can write

$$x[n]w[0] = \frac{1}{2\pi}\int_{-\pi}^{\pi} X[n,\omega)e^{j\omega n}d\omega \tag{3.8}$$

or, solving for $x[n]$,

$$x[n] = \frac{1}{2\pi w[0]}\int_{-\pi}^{\pi} X[n,\omega)e^{j\omega n}d\omega \tag{3.9}$$

Hence the only absolute constraint for being able to recover $x[n]$ from $X[n,\omega)$ is that $w[0] \neq 0$.

3.3 Alternate interpretations of the STFT operation

Although so far we have talked about the STFT simply as being the DTFT or DFT of a time function after it is multiplied by a sliding window in time, there are two other mathematically-equivalent ways of formulating the STFT. We discuss and compare the three implementations of the STFT in this section. We will use the DFT-based formulation of the STFT in this section because it is this formulation that is most commonly used in practice.

3.3.1 Fourier transform interpretation of the STFT

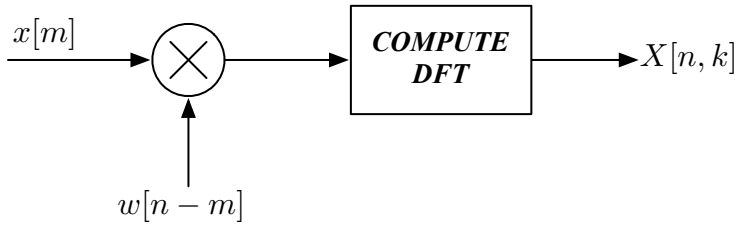


Figure 3.2: The Fourier transform implementation of the STFT.

As discussed above, the most straightforward way of thinking about the calculation of the STFT is as a multiplication of the time function $x[m]$ by a finite- or infinite-duration window function $w[n-m]$, followed by the computation of the Fourier transform of their product:

$$X[n,k] = \sum_{m=n-(N_w-1)}^{n} (x[m]w[n-m])e^{-j2\pi mk/N} = \sum_{m=n-(N_w-1)}^{n} (x[m]w[n-m])e^{-j\omega_k m} \qquad (3.10)$$

In the expression above, which assumes a finite-duration window of length N_w, the variable n indicates the position of the window, which designates the location of the "analysis frame." The variable k refers to the *frequency bin* in question. This is referred to as the *Fourier transform implementation* of the STFT.

3.3.2 Lowpass filter interpretation of the STFT

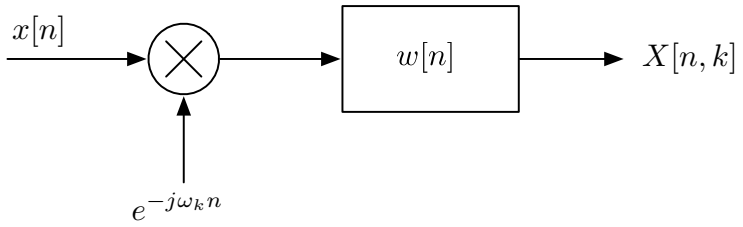


Figure 3.3: The lowpass filter implementation of the STFT.

We can rearrange the terms of the STFT equation slightly to produce

$$X[n,k] = \sum_{m=n-(N_w-1)}^{n} w[n-m]x[m]e^{-j\omega_k m} = \sum_{m=n-(N_w-1)}^{n} w[n-m](x[m]e^{-j\omega_k m}) \qquad (3.11)$$

This corresponds to multiplying the signal $x[n]$ by the complex exponential function $e^{-j2\pi nk/N} = e^{-j\omega_k n}$ and passing the product through a filter with unit sample response $w[n]$. This implementation of the STFT, which is referred to as the *lowpass filter implementation*, is depicted in the Fig 3.3 above.

Multiplying $x[n]$ by $e^{-j\omega_k n}$ shifts the spectrum to the left by ω_k, so that the components that were originally at frequency ω_k now lie at frequency 0. Because the unit sample responses of typical windows are lowpass in nature, the STFT $X[n,k]$ reflects the smoothed frequency content of the original function $x[n]$ at frequency ω_k as it evolves over time (represented by the variable n). As noted above, the lowpass filter implementation is mathematically equivalent to the Fourier transform implementation of the STFT $X[n,k]$.

3.3.3 Bandpass filter interpretation of the STFT

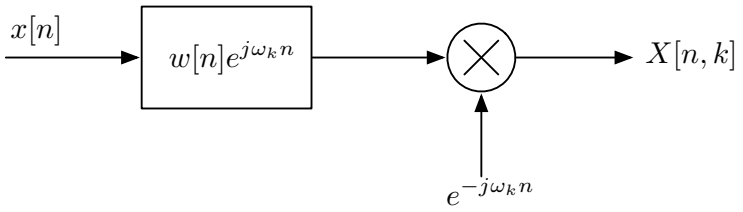


Figure 3.4: The bandpass filter implementation of the STFT.

A third mathematically-equivalent interpretation can be obtained by a simple manipulation of the STFT expression:

$$X[n,k] = \sum_{m=n-(N_w-1)}^{n} w[n-m]x[m]e^{-j\omega_k m} = \left(\sum_{m=n-(N_w-1)}^{n} \left(w[n-m]e^{j\omega_k (n-m)} \right) x[m] \right) e^{-j\omega_k n} \tag{3.12}$$

This implies that the input signal is passed through a bandpass filter consisting of the original lowpass window filter, frequency shifted so that it now passes frequency components centered around frequency ω_k. The spectrum of the output is then translated to the left so that the components of $X[n,k]$ that were originally at frequency ω_k are ultimately centered around frequency zero. This interpretation is shown in the Fig. 3.4 above and is referred to as the *bandpass filter implementation*.

Keep in mind that all three interpretations of the STFT are mathematically equivalent and that they all characterize $X[n,k]$ as representing the smoothed frequency content of the original function $x[n]$ at ω_k, evolving over time.

3.3.4 Implementations of the STFT using real time functions and impulse responses

The original lowpass and bandpass filter implementations assume multiplication by complex exponentials, and in the bandpass-filter implementation the filters have complex unit sample responses. For example, the lowpass filter implementation described in Eq. (3.11) above can be illustrated as

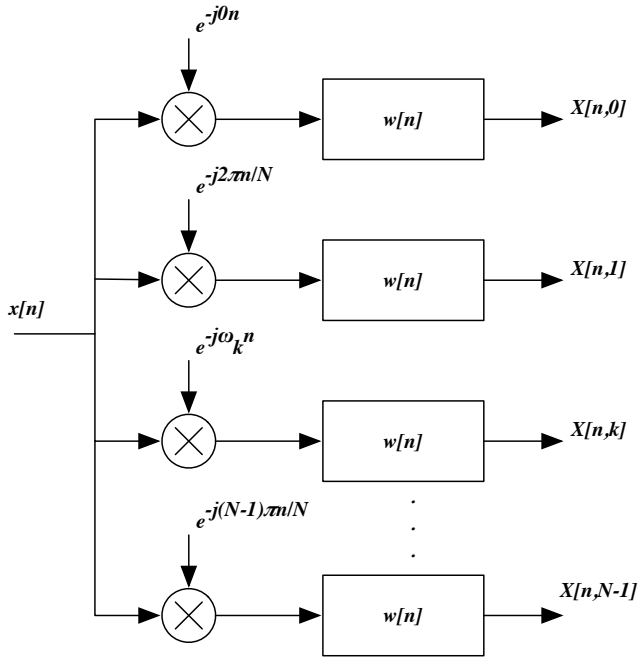


Figure 3.5: The lowpass filter STFT implementation on a channel-by-channel basis.

Each channel in Fig. 3.5 can be written as

$$X[n,k] = w[n] * \left(x[n]e^{-j\omega_k n}\right) = w[n] * (x[n]\cos(\omega_k n)) - jw[n] * (x[n]\sin(\omega_k n)) \tag{3.13}$$

which we can rewrite as

$$X[n,k] = X_r[n,k] - jX_i[n,k] \tag{3.14}$$

Note that the functions $X_r[n,k]$ and $X_i[n,k]$ are both real. This system is illustrated in Fig. 3.6

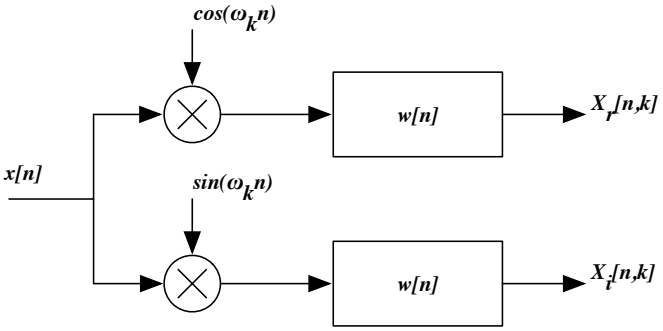


Figure 3.6: Single channel of the lowpass STFT implementation using real sample responses.

The corresponding bandpass filter implementation is a bit more algebraically involved, but can be obtained using similar principles, as is seen in Fig. 3.7.

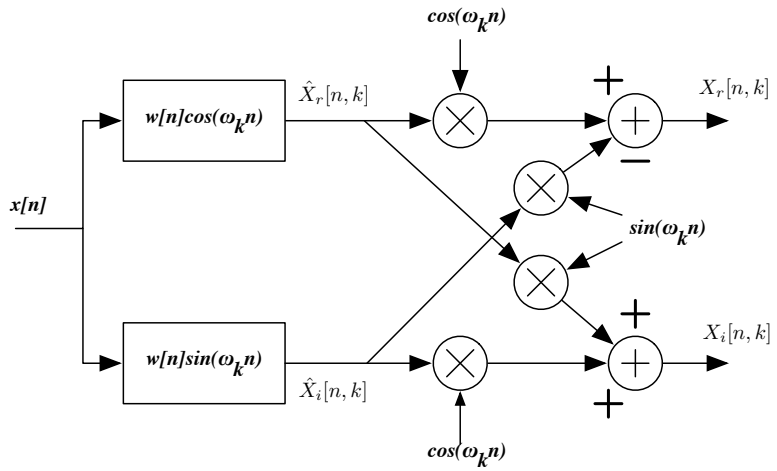


Figure 3.7: Single channel of the bandpass STFT implementation using real sample responses.

Note that the functions $\hat{X}_r[n,k]$ and $\hat{X}_i[n,k]$ (immediately before the cosine multiplications) have magnitudes that are equal to the final outputs $X_r[n,k]$ and $X_i[n,k]$. If we are only interested in computing the short-time *magnitude* spectrum (which is frequently the case), we can eliminate all the computation after the filter outputs, which causes the bandpass implementation to be more computationally efficient than the lowpass implementation because the multiplications are folded into the filtering.

3.4 Downsampling the STFT

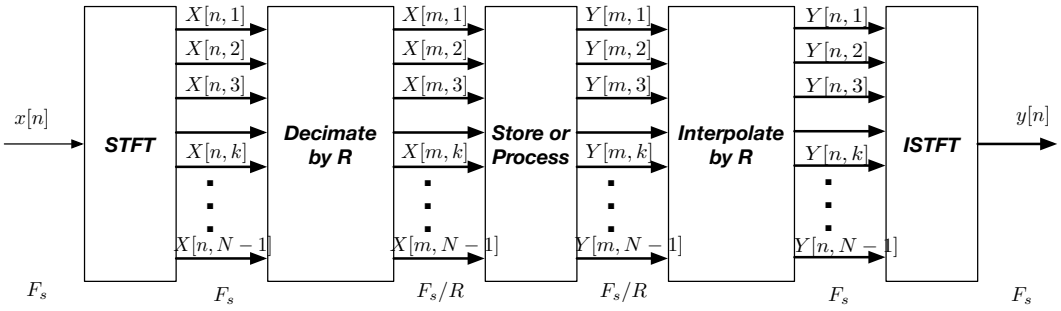


Figure 3.8: Block diagram of a complete STFT analysis/synthesis with downsampling.

In practice, the STFT coefficients are frequently downsampled as depicted in Fig. 3.8, either for efficiency in computation or for efficiency in manipulation or modification of the representation in the STFT domain. This is possible because STFT coefficients in a fixed frequency bin evolve slowly as a function of time. It is easiest to understand how this happens by reviewing the lowpass filter STFT implementation, as depicted in Fig. 3.3, which describes the STFT calculation as shifting the spectrum to the left by ω_k radians and then lowpass filtering the frequency-shifted signal by a filter having the unit sample response of $w[n]$. As an example, let us consider the Hamming window, which

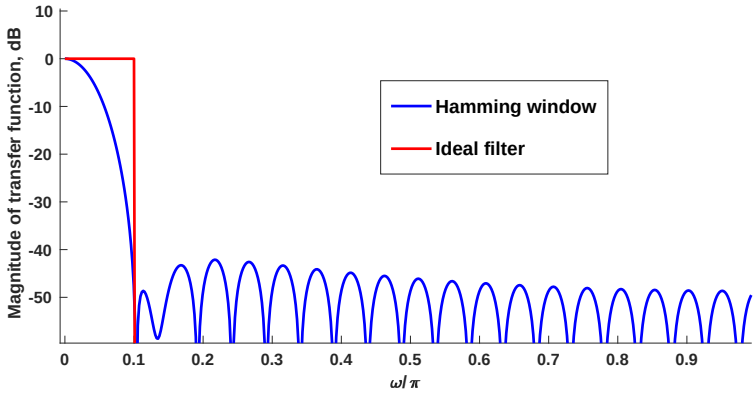


Figure 3.9: Magnitude in dB of the DTFT of a Hamming window of length 41 samples (blue curve). The red curve is the response of the corresponding ideal filter.

is frequently (although far from exclusively) used as the window function for STFTs. It is well known that the main lobe of the DTFT of the Hamming window has a mainlobe width of $8\pi/(N_w - 1)$ radians (including both positive and negative frequencies) where N_w is the length of the window. For example, a Hamming window of length 41 would have its first zero crossing in positive frequency at $\pi/10$ radians, which suggests that the output of this filter could be downsampled by a factor of 10.

Figure 3.9 compares the actual magnitude of the transfer function of a Hamming window of length 41 (blue curve) with the corresponding ideal filter (red curve). It is clear that the response of the Hamming window itself used as a lowpass filter is far from ideal, either in terms of the flatness of the response in the "passband" or in the suppression of sidelobes in the "stopband." Nevertheless, this approximation is commonly used in short-time Fourier transforms and normally it works well enough. And, in general, the location of the first zero-crossing in the frequency response of the window function is typically used as the nominal cutoff frequency for windows of other shapes as well.

3.5 Short-time Fourier synthesis

We will consider two methods of recovering the time function $x[n]$ from the STFT $X[n,k]$, one based on the filtering implementations and the other based on the Fourier transform implementation. We will first consider the filterbank implementation.

3.5.1 The Filter Bank Summation (FBS) method

Let us assume that $w[n]$ is of finite duration, and that $w[n] \neq 0$ for$0 \leq n \leq N_w - 1$. Because

$$X[n,k] = \sum_{m=n-(N_w-1)}^{n} (x[m]w[n-m])e^{-j\omega_k m} \tag{3.15}$$

where as usual $\omega_k = 2\pi k/N$, we can write

$$w[n-m]x[m] = \frac{1}{N}\sum_{k=0}^{N-1} X[n,k]e^{j\omega_k m} \tag{3.16}$$

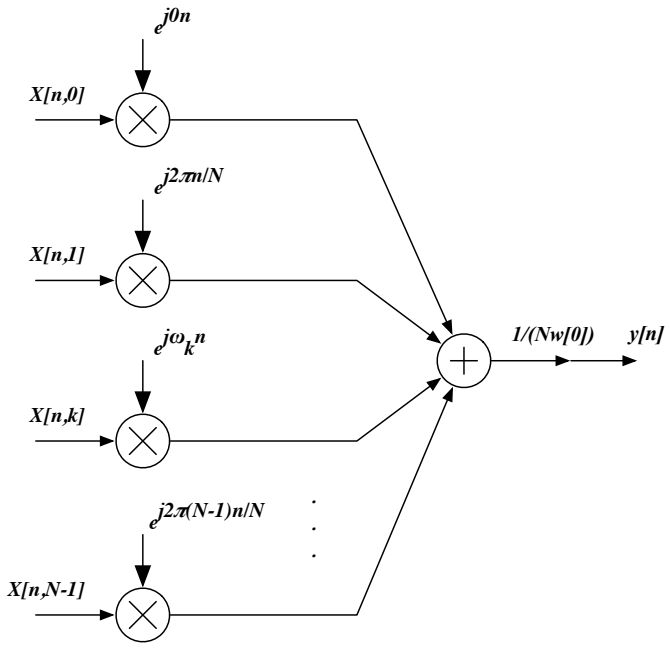


Figure 3.10: Block diagram of the filterbank summation (FBS) method of STFT synthesis.

Hence, for $w[0] \neq 0$ we obtain

$$x[n] = \frac{1}{Nw[0]} \sum_{k=0}^{N-1} X[n,k] e^{j\omega_k n} \tag{3.17}$$

This implies that in principle we can obtain the time function by multiplying the various short-time Fourier transform coefficients $X[n,k]$ by the function $e^{j\omega_k n}$, adding all the products together, and dividing by $Nw[0]$.

Are there any constraints on the window size and shape that are required for this to work? Let us designate the output of a complete analysis-synthesis system as $y[n]$. Then we can write

$$y[n] = \frac{1}{Nw[0]} \sum_{k=0}^{N-1} X[n,k] e^{j\omega_k n} = \frac{1}{Nw[0]} \sum_{k=0}^{N-1} \left(\sum_{m=-\infty}^{\infty} x[m] w[n-m] e^{-j\omega_k m} \right) e^{j\omega_k n} \tag{3.18}$$

Since the argument of the inner sum can be written as $x[m]w[n-m]e^{j\omega_k(n-m)}$, this expression can be rewritten as

$$y[n] = \frac{1}{Nw[0]} \sum_{k=0}^{N-1} x[n] * \left(w[n] e^{j\omega_k n} \right) = \frac{1}{Nw[0]} x[n] * \left(w[n] \sum_{k=0}^{N-1} e^{j\omega_k n} \right) \tag{3.19}$$

In order for $y[n]$ to be at least proportional to $x[n]$, we would need for the expression inside the parentheses to be proportional to $\delta[n]$. As you know, we can write the inner sum as

$$\sum_{k=0}^{N-1} e^{j2\pi nk/N} = \frac{1 - e^{j2\pi n}}{1 - e^{j2\pi n/N}} \tag{3.20}$$

which is equal to N for $n = rN$ for integer r and zero otherwise. Hence we must require that $w[n] = 0$ for $n = rN$ and $n \neq 0$ in order for the filter-bank summation (FBS) method of synthesis to be used. This condition is sometimes called the "FBS constraint." The FBS constraint is satisfied trivially for finite-duration windows of length N, but it is also satisfied by another class of windows known as *Nyquist windows* such as the familiar $\sin(x)/x$ and $\sin(Nx)\sin(x)$ functions, which have an infinite number of equally-spaced repeating zeros in the time function.

Another way of expressing the FBS constraint is that we require that $w[n]$ satisfy the constraint

$$w[n]p_N[n] = \delta[n] \tag{3.21}$$

where N is the number of equally-spaced frequency channels and $p_N[n]$ is a pulse train with period N:

$$p_N[n] = \begin{cases} 1, & n = rN \\ 0, & \text{otherwise} \end{cases} \tag{3.22}$$

This, of course is simply a compact way of stating that $w[n] = 0$ for $n = rN$ and $n = 0$ otherwise, as before. Taking the DTFT of Eq. (3.21) above produces

$$\frac{1}{2\pi} W(e^{j\omega}) \circledast P_N(e^{j\omega}) = 1 \tag{3.23}$$

where $P_N(e^{j\omega})$ is the DTFT of $p_N[n]$ and the symbol $\circledast$ indicates circular convolution as before.

As you will recall, the DTFT $P_N(e^{j\omega})$ is an infinite train of delta functions in frequency, each with area $2\pi/N$, separated by $2\pi/N$ along the frequency axis. This means that an alternate form of the FBS constraint is

$$\frac{1}{N} \sum_{l=0}^{N-1} W(e^{j(\omega - 2\pi l/N)}) = \text{some constant} \tag{3.24}$$

In other words, the FBS constraint is satisfied if the DTFTs of the window translated every $2\pi/N$ radians along the ω-axis and added together sum to a constant that is independent of ω.

3.5.2 The Overlap-Add (OLA) method

The overlap-add (OLA) method, which is based on the Fourier transform implementation of the STFT, is easier to describe and analyze than the FBS method. In our discussion here we will make use of the DTFT-based definition of the STFT, but the OLA method using the DFT-based definition works in exactly the same way.

Consider samples of the STFT spaced apart by R frames, $X[rR, \omega)$, which we will also refer to as $Y_r(e^{j\omega})$. The inverse DTFT of $Y_r(e^{j\omega})$ is $w[rR - m]x[m] \equiv y_r[m]$, as defined in Eq. (3.7). We obtain the output function $y[m]$ by simply adding all of the inverse transforms $y_r[m]$ together:

$$y[m] = \sum_{r=-\infty}^{\infty} y_r[m] = \sum_{r=-\infty}^{\infty} w[rR - m]x[m] = x[m] \sum_{r=-\infty}^{\infty} w[rR - m] \tag{3.25}$$

Clearly, for $y[n]$ to be equal to $x[n]$ we must ensure that the sum of all the window functions equals 1:

$$\sum_{r=-\infty}^{\infty} w[rR-m] = 1 \tag{3.26}$$

This is satisfied, for example, by rectangular windows that are abutted and triangular or Hamming windows that are overlapped by 50 percent, as depicted in Fig. 3.11, among many other window shapes and spacings.

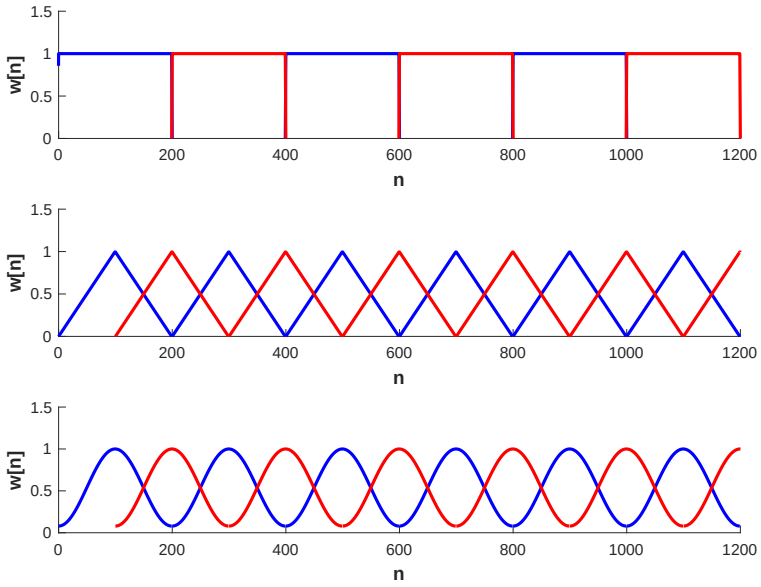


Figure 3.11: Examples of sequences of rectangular, triangular (Bartlett), and Hamming window functions of length 200 that sum to 1. Note that the the rectangular windows may be abutted, while the triangular and Hamming windows must be overlapped by 50 percent.

3.6 Sampling in time and frequency

As in many other engineering applications, we are interested in being as efficient in computation and storage as possible. We briefly discuss in this section the number of numbers that are (nominally) needed to obtain a complete characterization of a time function using short-time Fourier analysis, specifically reviewing how many frequency channels we need and how sparsely we can sample in time for a given window $w[n]$. As will see, the answer to these questions can depend on whether OLA or FBS synthesis is used. We will consider only FIR windows at this time, although the same principles hold for IIR windows as well.

3.6.1 The Fourier-transform implementation with overlap-add synthesis

Let us consider first the Fourier-transform implementation with overlap-add synthesis. As has been discussed above, time sampling using OLA is determined by the length and shape of the window. Specifically, the spacing must be such that the sum of the windows

(in the locations that they occur at) must add to a constant, as discussed above. This means, for example, that if we have a rectangular window with length N_w, successive windows can be abutted, causing the window spacing to be N_w. If Hamming windows are used, on the other hand, a 50% overlap will be needed, requiring the windows to be spaced apart by $N_w/2$ samples. Since we are computing DFTs for each window, we need a DFT size that is at least as large as the duration of the window to avoid temporal aliasing, or at least N_w channels. Now let us define the constant N_s to represent the total number of numbers per second of input needed to store the signal using the STFT representation. For a sampling rate of F_s and a rectangular window of length N_w the N_s would be equal to the sampling rate times the number of channels divided by the spacing of the windows. For rectangular windows this would be

$$N_s = \frac{F_s N_w}{N_w} = F_s \tag{3.27}$$

For Hamming windows this would be

$$N_s = \frac{F_s N_w}{(N_w/2)} = 2F_s \tag{3.28}$$

3.6.2 The lowpass and bandpass implementations with filterbank-summation synthesis

For either the lowpass or bandpass implementation with FBS resynthesis, the number of channels is determined by the FBS constraint that for a given window length N_w the window shape $w[n]$ must be such that

$$w[n] = 0 \text{ for } n = rN_w \text{ with } r \neq 0 \tag{3.29}$$

This constraint is automatically satisfied for an FIR window if the number of channels N is greater than or equal to N_w. The time sampling is determined by the effective bandwidth of the window. This is determined by looking for the first frequency ω at which the Fourier transform of the window, $W(e^{j\omega})$ is zero, as shown in Fig. 3.9. For a rectangular window this frequency is $2\pi/N_w$ and for the Hamming window this frequency is $4\pi/(N_w - 1)$. Recall from our discussion of multi-rate DSP that if a discrete-time signal is limited to frequencies of π/M_d, we can downsample it by a factor of M_d. Hence, for a sampling rate of F_s and a downsampling rate of M_d, the total number of samples per second would be the sampling rate times the number of channels divided by the decimation rate, which for rectangular windows would be

$$N_s = \frac{F_s N_w}{(N_w/2)} = 2F_s \tag{3.30}$$

For Hamming windows the total number of samples per second would be

$$N_s = \frac{F_s N_w}{(N_w - 1)/4} \approx 4F_s \tag{3.31}$$

As can be seen, the number of numbers per second using FBS resynthesis needed to represent a signal is twice as many as with OLA resynthesis, which in turn is twice as many as the original sampled waveform in the time domain. In fact, the representation is even

more inefficient because the STFT coefficients are complex, requiring two real numbers each, although if the time function is real, the coefficients representing positive and negative frequencies would be complex conjugates of each other. Nevertheless, the STFT representation is widely used in all cases because of the insight it can provide in analyzing signals as well as the signal-manipulation operations that it enables.

3.7 Applications of short-time Fourier analysis

3.7.1 Spectral subtraction

Spectral subtraction was first proposed by Boll in 1979 as a means to remove additive noise from speech and other signals. It was the foundational algorithm for a huge quantity of work in speech enhancement. It is extremely simple, but it has been improved on by dozens if not hundreds of subsequent research papers. Some of the latter work is reviewed superficially in a subsection of the book *Spoken Language Processing*,[2] by Huang, Acero, and Hon (all former CMU students and faculty). In essence, the goal of spectral subtraction is to remove (or at least attenuate) background noise in speech. This is accomplished by first estimating the power spectrum of the background noise at a time when speech is believed not to be present. The magnitude of the noise estimate is subtracted from the magnitude of the DFT coefficients in each frame. The difference of the magnitudes is combined with the original phase in each frame to produce a new set of DFT coefficients. The output time function is reconstructed by computing the inverse DFT of each frame, and then reconstructing the time function from each STFT frame, typically using the overlap-add method as discussed above.

A great deal of work has been performed with the goal of improving the basic spectral subtraction algorithm. Typically these efforts address one of the following issues:

- With the actual noise (and signal) changing over time, it is easily possible for the subtraction to result in a negative magnitude which is impossible in theory and problematical in practice. The most basic solution to this problem is to simply force the floor of the subtraction to be zero. Many more sophisticated schemes have been proposed that typically involve "under-subtracting" the estimated noise magnitude from the magnitude of the spectral of the incoming signal.

- In reality the masking noise is not stationary. Many algorithms have been proposed to update the estimate of the power spectrum of the background noise dynamically, in an adaptive fashion. These approaches typically depend on being able to detect the absence of the target speech, though, which can be a difficult problem in many practical applications.

- In recent years especially, greater attention has been focused on the consequences of the mismatch between the magnitude of the frequency response in the individual frames and the corresponding phases. (This happens, of course, because we are combining the phase of the original speech with the magnitude of the signal after subtracting the magnitude of the noise estimate.) Some approaches to this have been successful, but they all involve iterative nonlinear estimation techniques.

[2] Huang, X. D., Acero, A., and Hon, H.-W., *Spoken Language Procesing*, Prentice-Hall PRT, 2001.

3.7.2 Phase vocoding

Phase vocoding was originally developed by Flanagan and Golden (1966) as a technique to accomplish high-quality speech coding. While phase vocoding did not prove to be a commercially-successful method to encode speech, it does have a number of properties that make it useful for expanding and contracting speech in time, and for changing the pitch of music with relatively small changes in the musical timbre. In this section we describe the basic principles of phase vocoding and describe how it is applied to nonlinear transformations in time and frequency.

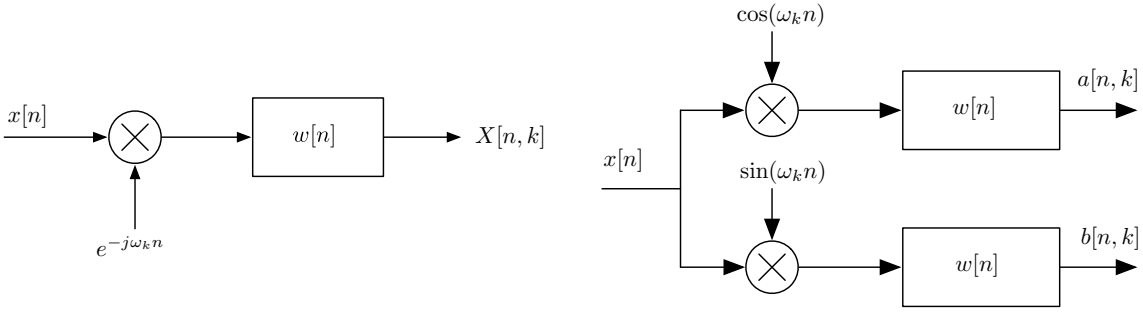


Figure 3.12: The lowpass filter interpretation of the STFT in complex form (left panel) and with real coefficients (right panel)s.

Figure 3.12 recapitulates a single channel of the lowpass filter implementation of short-time Fourier analysis, in both the original complex exponential form (left panel), and using real coefficients (right panel). Note that we are now using a slightly different notation to represent the STFT coefficients as

$$X[n,k] = a[n,k] - jb[n,k] \tag{3.32}$$

where $a[n,k] = Re[X[n,k]]$ and $b[n,k] = -Im[X[n,k]]$.

Now let us consider the STFT coefficients in terms of their magnitude and phase. Specifically, we can represent $X[n,k]$ as

$$X[n,k] = |X[n,k]|e^{j\theta[n,k]} \tag{3.33}$$

where $\theta[n,k]$ represents the instantaneous phase at frame n and frequency bin k. Clearly, we can obtain the magnitude and phase of the STFT directly from the real and imaginary parts $a[n,k]$ and $b[n,k]$ via the standard trigonometric relations:

$$|X[n,k]| = \sqrt{a^2[n,k] + b^2[n,k]} \text{ and } \theta[n,k] = -\tan^{-1}\left(\frac{b[n,k]}{a[n,k]}\right) \tag{3.34}$$

Note that if the STFT coefficients $X[n,k]$ are Hermitian symmetric, by taking the inverse transform we can represent the corresponding time function for that channel as

$$x_k[n] = |X[n,k]|\left(e^{j\theta[n,k]}e^{j\omega n} + e^{-j\theta[n,k]}e^{-j\omega_k n}\right) = 2|X[n,k]|\cos(\omega_k n + \theta[n,k]) \tag{3.35}$$

and in principle the entire waveform could be reconstructed by summing these cosines across all channels:

$$x[n] = \sum_k x_k[n] = \sum_k 2|X[n,k]|\cos(\omega_k n + \theta[n,k]) \tag{3.36}$$

The original idea of phase vocoding was to extract and transmit the magnitude and phase terms representing the signal at each frame, reconstructing the waveform at the far end from this information only.

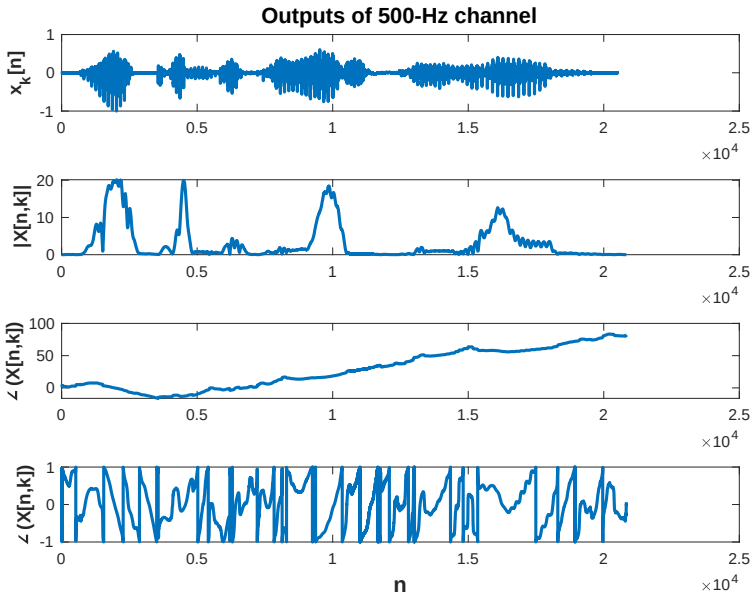


Figure 3.13: Representations obtained via phase vocoding for a channel at $\omega_k = 2\pi 500$. From top to bottom, depicted for that channel are (a) $x_k[n]$, $|X[n,k]|$, $\angle X[n,k]$ (unwrapped), and $\angle X[n,k]$ (wrapped and normalized by dividing by π).

Figure 3.13 depicts the magnitude and phase of the "Welcome to DSP-I" utterance for a typical channel at 500 Hz. The figures show the representation of the waveform in that channel as reconstructed using Eq. (3.35), along with the magnitude and phase of the STFT representation. Note that the phase is presented twice: in its original "unwrapped" form and in terms of the principal value, with the values of $\theta[n,k]$ constrained to lie between $-\pi$ and π radians.

From Fig. 3.13 it can be seen that both forms of the phase function are problematic. Specifically, if the phase is transmitted in its original unwrapped form, the magnitude of the phase is unbounded and in general will get large quickly, ultimately causing overflow after a sufficient time. If only the principal value is transmitted, that signal will have a very large bandwidth because of the abrupt transitions by 2π radians when the magnitude of the phase exceeds π in either direction. The solution to this problem is to transmit an approximation to the *derivative with respect to time* of the phase function.

To simplify the analysis of this approach, we will briefly detour into continuous-time processing. Specifically, let the continuous-time and continuous-frequency representation that is similar to that of Eqs. (3.35) and (3.36) be:

$$x(t) = \sum_k x_k(t) = \sum_k 2|X[t,\Omega]|\cos(\omega_k t + \theta[t,\Omega]) \tag{3.37}$$

where

$$x_k(t) = 2|X(t,\Omega)|\cos(\omega_k t + \theta(t,\Omega)) \tag{3.38}$$

Note that the derivative of the argument of the cosine term, $\omega_k + \dot{\theta}(t,\Omega_k)$, has the dimension of *frequency*. In fact, the derivative of the phase term $\dot{\theta}(t,\Omega_k)$ is considered to be the instantaneous frequency of that channel. One of the motivations for transmitting the magnitude and phase derivative rather than the magnitude and phase is that the instantaneous frequency of a signal typically changes slowly because of physical limitations on how that signal was produced, regardless of whether the source is a human or a machine.

Generalizing, we can define the continuous-time analog of the STFT as

$$X_c(t,\Omega_k) = \int_{-\infty}^{\infty} x_c(\tau) w_c(t-\tau) e^{-j\omega_k \tau} d\tau \tag{3.39}$$

where $w_c(t)$ is a continuous-time window function, and the other variables are in direct correspondence to their discrete-time counterparts. This suggests that the STFT coefficients in continuous time and frequency can be represented as

$$X_c(t,\Omega_k) = |X_c(t,\Omega_k)| e^{j\theta_c(t,\Omega_k)} = a_c(t,\Omega_k) - j b_c(t,\Omega_k) \tag{3.40}$$

where

$$|X_c(t,\Omega_k)| = \left[a_c^2(t,\Omega_k) + b_c^2(t,\Omega_k)\right]^{1/2} \tag{3.41}$$

and

$$\theta_c(t,\Omega_k) = -\tan^{-1}\left[\frac{b_c(t,\Omega_k)}{a_c(t,\Omega_k)}\right] \tag{3.42}$$

Using the relationship

$$\frac{d}{dx}\tan^{-1}(x) = \frac{1}{1+x^2} \tag{3.43}$$

it follows directly that *instantaneous frequency* can be represented as

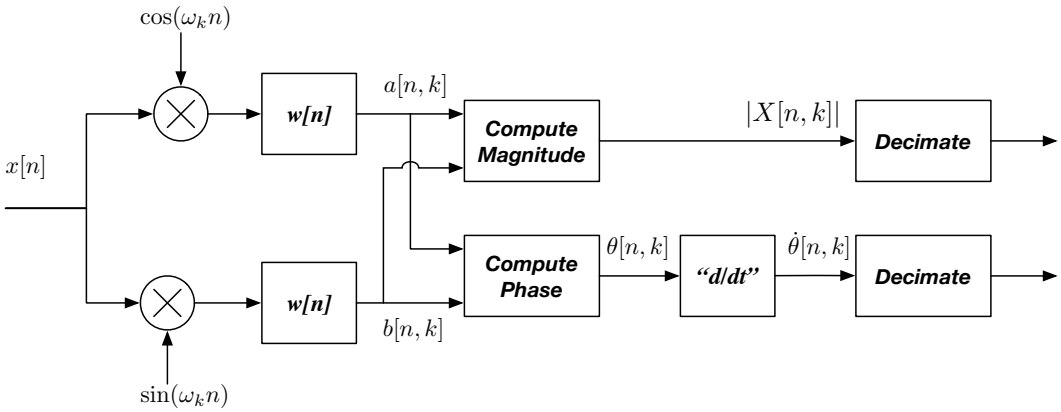


Figure 3.14: Block diagram of the encoding portion of the phase vocoder.

$$\dot{\theta}_c(t,\Omega_k) = \frac{b_c(t,\Omega_k)\dot{a}_c(t,\Omega_k) - a_c(t,\Omega_k)\dot{b}_c(t,\Omega_k)}{a_c^2(t,\Omega_k) + b_c^2(t,\Omega_k)} \tag{3.44}$$

The discrete-time analogy to Eq. (3.44) is

$$\dot{\theta}[n,\omega_k] = \frac{b[n,\omega_k]\dot{a}[n,\omega_k] - a[n,\omega_k]\dot{b}[n,\omega_k]}{a^2[n,\omega_k] + b^2[n,\omega_k]} \tag{3.45}$$

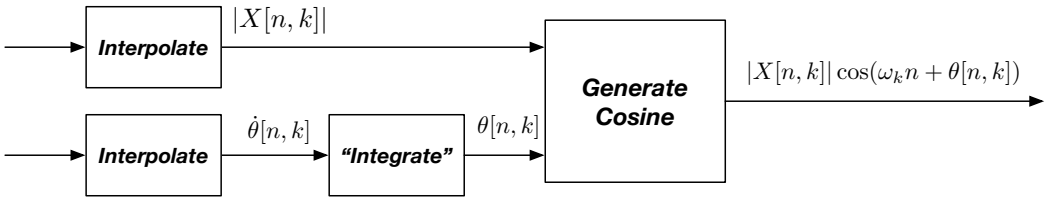


Figure 3.15: Block diagram of the decoding portion of the phase vocoder.

These equations have the advantage that they do not require the arctan operation to be evaluated directly.

Figures 3.14 and 3.15 summarize the encoding and decoding steps for phase vocoding. In the encoding process, the magnitude and phase derivative, $|X[n,k]|$ and $\dot{\theta}[n,k]$, are computed from the real and imaginary parts of the STFT coefficients, $a[n,k]$ and $b[n,k]$ according to Eq. (3.34) for the magnitude and Eq. (3.45) for the phase derivative. While the differentiation in time in the expressions $\dot{a}[n,k]$ and $\dot{b}[n,k]$ in Eq. (3.44) cannot be implemented exactly in discrete time, there are many discrete-time approximations to continuous-time differentiation of which the simplest is the *first difference*:

$$x_{\text{diff}}[n] = \frac{1}{T}(x[n] - x[n-1]) \tag{3.46}$$

where T is the sampling period. This is a reasonable approximation to differentiation at low frequencies for which $\sin(\omega n/2) \approx \omega n/2$. In addition, there are many standard filter design techniques (including the Parks-McClellan equiripple filter design method) that produce approximations to ideal differentiation that are valid over a much greater frequency range.

As Fig. 3.15 indicates, conversion from the magnitude and phase-derivative representation to the time function is begun by "integrating" the phase derivative to obtain the instantaneous running phase. This is normally approximated by computing the running cumulative sum of the phase derivative after processing is completed. The instantaneous phase and the magnitude are combined to obtain the time-domain signal in each channel:

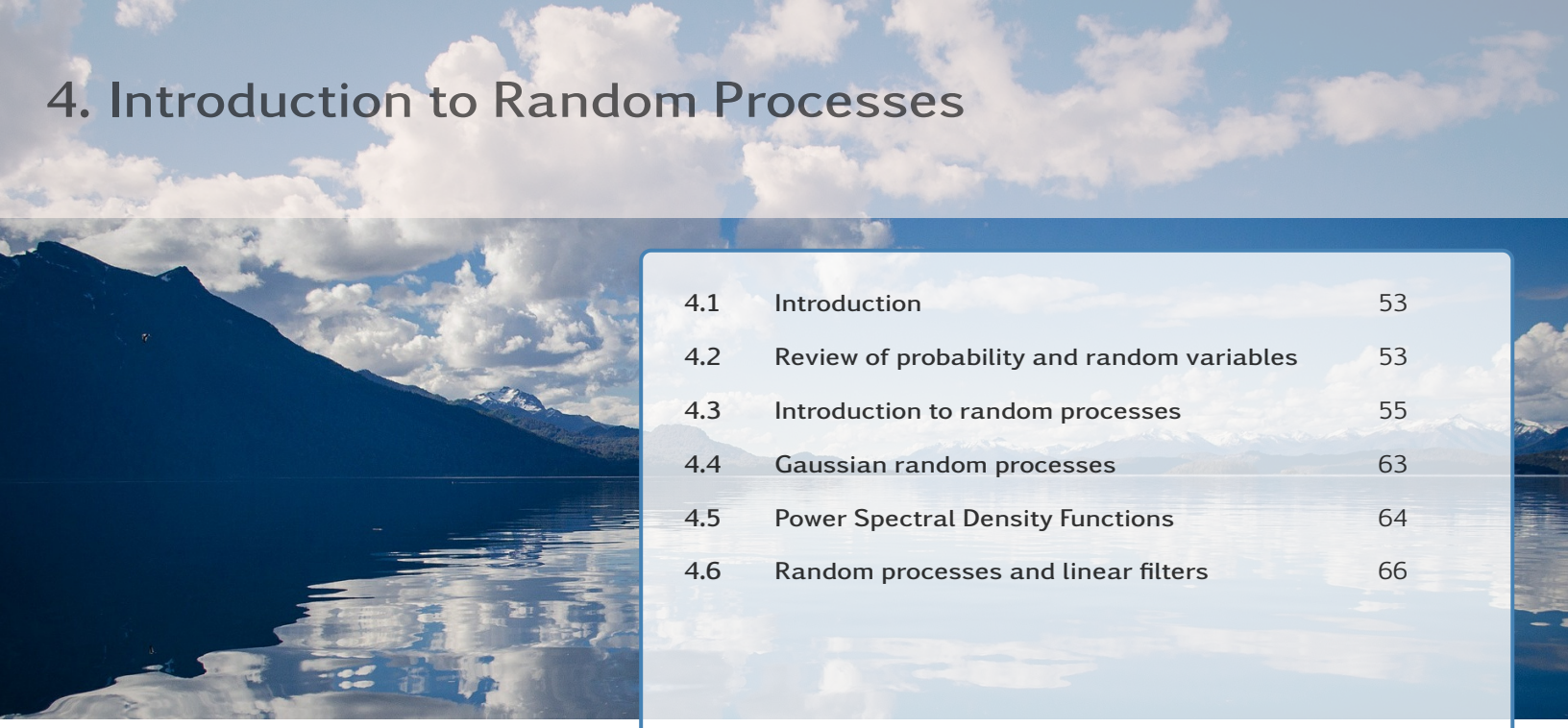
$$x_k[n] = 2|X[n,k]|\cos(\omega_k n + \theta[n,k]) \tag{3.47}$$

where as usual $\omega_k = 2\pi k/N$. The signals in each positive frequency channel $x_k[n]$ are summed over all channels to obtain the output waveform.

Time compression and transposition. Two popular uses of phase vocoding are in changing the rate of speech without affecting intelligibility, and in musical transposition. Specifically, multiplying all of the instantaneous frequencies of the representation of a musical performance by a constant will increase the pitch by the same constant. For example, multiplying the instantaneous frequencies by 6/5 = 1.2 will increase the pitch by the ratio of 6 : 5, which is equivalent to transposing upward by a minor third in music.

Speech rate can be changed by scaling the instantaneous frequencies by a constant factor while changing the sampling rate by that same factor. For example, *slowing down* speech by a factor of 1.2 is easily accomplished by multiplying the instantaneous frequency by 1.2 while increasing the sampling frequency by the same factor.

4. Introduction to Random Processes



One of the most important things about advanced DSP is that the time functions that we encounter vary randomly. In this chapter we discuss how we characterize these "random processes" mathematically. Understanding this characterization is enormously important in order for us to be able to deal with the randomness that is encountered everywhere in the real world.

4.1 Introduction

Random processes are at the heart of most techniques developed in ADSP. They are a natural extension of the concept of random variables in probability theory. In this chapter we briefly and superficially review some of the concepts discussed in the lectures. These topics are treated in greater detail in Appendix A of Oppenheim, Schafer, Yoder, and Padgett (2010), and in somewhat greater detail in Chapter 8 of Oppenheim and Schafer (1975)[1]. There are, of course, many complete text and reference books on random processes.

4.2 Review of probability and random variables

Probability theory is used to describe some things that are difficult to develop using classical techniques in which the outcome of a "random experiment" will vary in an unpredictable fashion from trial to trial, but in which the statistics of the outcomes over a long time are more predictable. For example, tossing a coin is actually nothing more than a very complicated problem in Newtonian mechanics: if we knew the position and velocity of a coin when we throw it up in the air, and if we knew the location and mechanical reflectance of all the objects in the room, we should be able to predict whether it will land "heads" or "tails." But it is hard to solve the problem that way, and the truth is that we do not really care. A more useful characterization is to simply state that each side comes up about half the time, and that successive tosses are independent. We could make a similar

[1] Portions of this chapter are loosely based on the discussions in Chapter 8 of Oppenheim and Schafer (1975).

argument with dice, roulette wheels, etc. The important thing is that probability is just a mathematical characterization that is both consistent and useful.

4.2.1 Probability of events

Many discussions of probability theory begin with a discussion of probabilities of events. Consider, for example, a set of events A_i that are mutually exclusive and assigned probabilities $P[A_i]$. While normally we would expect that these probabilities would reflect the relative frequency of observing the various events A_i, from the mathematical standpoint the only constraint is that the probabilities must be numbers between 0 and 1 and that these numbers must sum to 1 over all possible events. We also talk about the joint probabilities of two events $P[AB]$ as well as the conditional probability of one event given the other, $P[A|B]$. These are related according to the equation

$$P[A|B] = \frac{P[AB]}{P[B]} \tag{4.1}$$

Since

$$P[AB] = P[A|B]P[B] = P[B|A]P[A] \tag{4.2}$$

we can write the well-known *Bayes rule*,

$$P[B|A] = \frac{P[A|B]P[B]}{P[A]} \tag{4.3}$$

Bayes rule is the conceptual foundation for most of statistical decision making and machine learning.

4.2.2 Random variables

Many interesting things that are modeled by probability theory can be described as random experiments of which the outcome is a number. These outcomes are referred to as *random variables*. For example, the random variable x may represent the temperature outside the front door of my home at 9 am today. This number (which could in principle be any real number but is much more restricted in practice) is unknown *a priori* but we have some idea of what it might be like statistically, given the time of year. (The temperature in September is likely to be different from the temperature in January!) Other sources of such *a priori* knowledge might include what the temperature was yesterday, and what the weather announcer said on television in the morning the temperature is likely to be. Another example of a random variable is the integer number representing the number of passengers that are in the first bus that stops at the corner of Fifth and Highland Avenues, not far from my home.

We will use the *probability density function* (pdf) $p_x(\alpha)$ to represent the random variable x. Note that the name of the random variable is identified by the subscript of p, and the dummy variable α represents possible values that x may take on. The probability density function is related to probabilistic events by the equation

$$P[a < x \leq b] = \int_a^b p_x(\alpha) d\alpha$$

We know that $p_x(\alpha)$ cannot be negative, and its integral over all values of α must be 1. We can define the conditional probability density function $p_{x|A}(\alpha|A)$ and the joint probability density function $p_{xy}(\alpha,\beta)$ in similar fashion. Note that the marginal probability density $p_x(\alpha)$ can be obtained by integrating the joint probability density function $p_{xy}(\alpha,\beta)$ over all possible values of β:

$$p_x(\alpha) = \int_{-\infty}^{\infty} p_{xy}(\alpha,\beta)d\beta \tag{4.4}$$

4.2.3 Selected other relationships

You should have covered many useful relationships and properties of random variables in your prior courses in probability theory.

For example, two random variables x and y are *statistically independent* if their joint pdf can be factored into two marginal pdfs:

$$p_{xy}(\alpha,\beta) = p_x(\alpha)p_y(\beta) \tag{4.5}$$

The *expected value* (or statistical average) of any function of x, $g(x)$, can be obtained by

$$\mathbb{E}[g(x)] = \int_{-\infty}^{\infty} g(\alpha)p_x(\alpha)d\alpha \tag{4.6}$$

In other words, the statistical average of the function of $g(x)$ equals every possible value of $g(x)$ weighted by the relative likelihood of observing the particular value of x from which the calculation of the function $g(x)$ is based.

Some specific statistical averages will be used widely in our discussions. For example, the *mean* of x is $m_x = \mathbb{E}[x]$, the *variance* of x is $\sigma_x^2 = \mathbb{E}[(x-m_x)^2] = \mathbb{E}[x^2] - m_x^2$, the *standard deviation* of x is σ_x, and the *covariance* of two random variables x and y is $\sigma_{xy} = \mathbb{E}[(x-m_x)(y-m_y)] = \mathbb{E}[xy] - m_x m_y$.

4.3 Introduction to random processes

As we discussed above, the outcome of a "random experiment" in probability theory is either a discrete event, or a number which we refer to as a random variable. The key difference between random variables and random processes is that random processes (which are also referred to as stochastic processes) can be considered to be the outcome of a random experiment for which the outcome is a time function $x[n]$ (called a "sample function") that in principle extends over all time, from $-\infty$ to ∞.

The value of a sample function at a particular time n_1 is referred to as $x[n_1]$, and it will also be notated from time to time as x_{n_1}. These values of a sample function of a random process at a specific time are actually random variables, and they are represented by the pdf $p_{x[n_1]}(\alpha)$ or $p_{x_{n_1}}(\alpha)$. (Lamentably, neither of these notational conventions is very elegant!)

Examples of brief segments of sample functions of a random process are shown in Figs. 4.1, 4.2, and 4.3, which in succession depict segments of broadband noise, speech, and a random-phase cosine wave of the form $x[n] = \cos(\omega n + \phi)$ where ϕ is a random parameter.

Note once again that in all cases the actual sample functions actually extend from $-\infty$ to ∞ in time, and only a brief portion is depicted of necessity. (In other words, we can only observe a part of one sample function of any particular random process in our lifetime!) Note also that in the case of the random-phase cosine the sample functions appear to be deterministic. Nevertheless, they are random because the phase of each cosine is a random parameter.

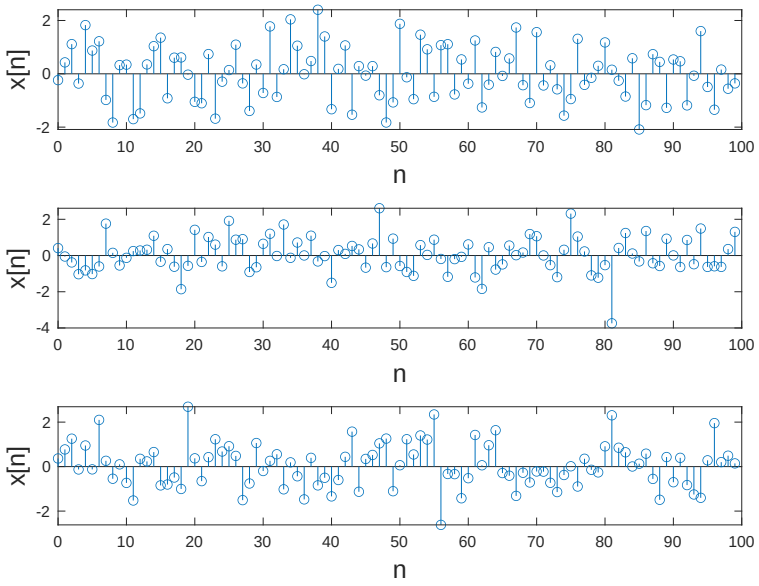


Figure 4.1: Sample functions of a white-noise random process.

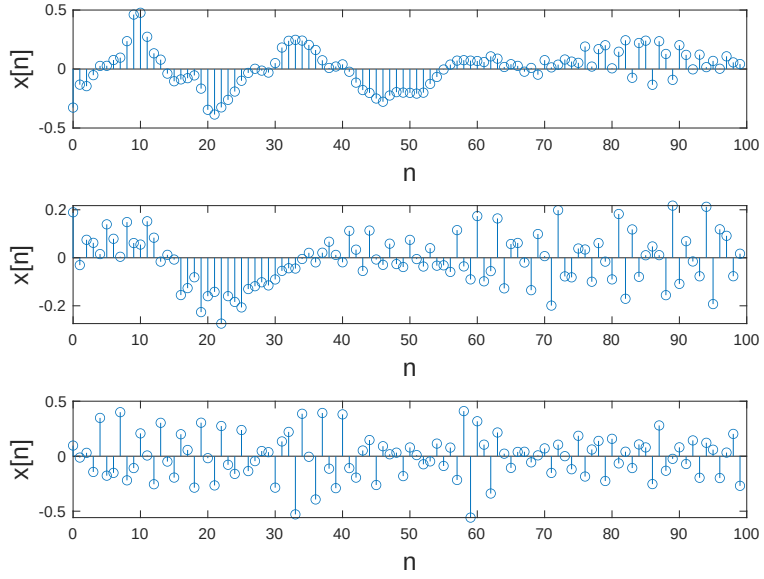


Figure 4.2: Sample functions of a speech waveform.

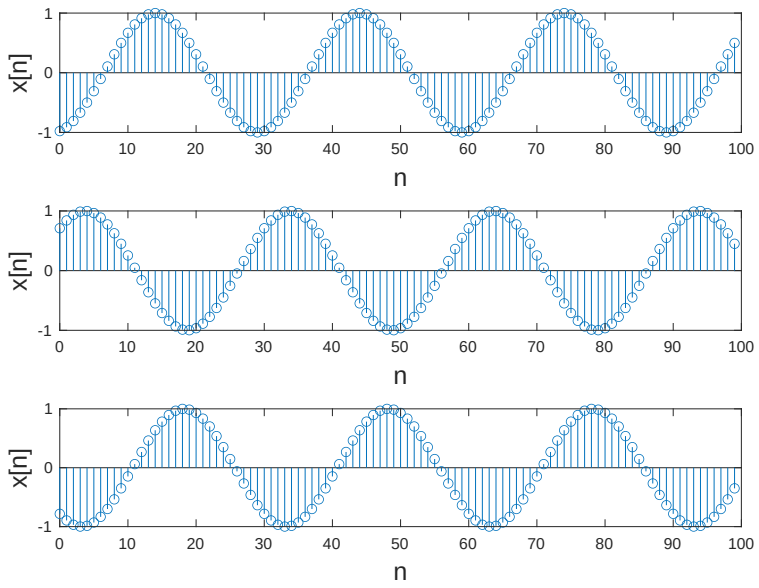


Figure 4.3: Sample functions of random-phase cosine waves.

4.3.1 Ensemble averages

Ensemble averages (commonly known as expected values) describe the average values of random variables (including the random variables obtained by evaluating random processes at specific times), averaged over the ensemble of sample functions. Most of the important characteristics of random processes that we care about, such as their means, variances, covariances are actually ensemble averages.

In general, the expected value of any function g of a random variable $x[n]$ is defined as

$$\mathbb{E}[g(x[n])] = \int_{-\infty}^{\infty} g(\alpha) p_{x[n]}(\alpha) d\alpha \tag{4.7}$$

Clearly this expression is no different from evaluating an arbitrary statistical average of a general random variable as in Eq. (4.6), except that these random variables were obtained by looking at the sample functions of a random process at a particular time.

Some important statistical averages based on $x[n]$ include

- The *mean* of $x[n]$ is

$$m_{x[n]} = \mathbb{E}[x[n]] = \int_{-\infty}^{\infty} \alpha p_{x[n]}(\alpha) d\alpha \tag{4.8}$$

  The mean, of course, represents the average value of x at time n, averaged across all sample functions in the ensemble.

- The *autocorrelation* (or autocorrelation function) of $x[n]$ is

$$\phi_{xx}[n_1, n_2] = \mathbb{E}[x[n_1]x^*[n_2]] = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \alpha\beta^* p_{x[n_1]x[n_2]}(\alpha, \beta) d\alpha d\beta \tag{4.9}$$

  where in this discussion the asterisk symbol $(*)$ indicates complex conjugation. (Most of the random processes that we consider will have strictly real sample functions,

and we will drop the asterisk symbol at some point.) The autocorrelation function is a measure of how much the sample functions of x at time n_1 resemble the same sample functions at time n_2. It plays a very important role in calculating power spectral density functions, as we will discuss below.

- The *autocovariance function* of $x[n]$ is

$$\gamma_{xx}[n_1, n_2] = \mathbb{E}[(x[n_1] - m_{x[n_1]})(x[n_2] - m_{x[n_2]})^*] \tag{4.10}$$

$$= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}(\alpha - m_{x[n_1]})(\beta - m_{x[n_2]})^* p_{x[n_1]x[n_2]}(\alpha, \beta)d\alpha d\beta \tag{4.11}$$

  Clearly the autocovariance function is very similar to the autocorrelation function.

- The *crosscorrelation function* of the random processes $x[n]$ and $y[n]$ is

$$\phi_{xy}[n_1, n_2] = \mathbb{E}[(x[n_1]y^*[n_2]] = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\alpha\beta^* p_{x[n_1]y[n_2]}(\alpha, \beta)d\alpha d\beta \tag{4.12}$$

  Unsurprisingly, the crosscorrelation function is a measure of how much the sample functions of x at time n_1 resemble the sample functions of y at time n_2. The crosscorrelation function plays a very important role in estimating the time delay between two signals, which is a key enabling technology for location services such as GPS.

- The *crosscovariance function* of $x[n]$ and $y[n]$ is

$$\gamma_{xy}[n_1, n_2] = E([x[n_1] - m_{x[n_1]})(y[n_2] - m_{y[n_2]})^*] \tag{4.13}$$

$$= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}(\alpha - m_{x[n_1]})(\beta - m_{y[n_2]})^* p_{x[n_1]y[n_2]}(\alpha, \beta)d\alpha d\beta \tag{4.14}$$

It is easy to show that $\gamma_{xx[n_1,n_2]} = \phi_{xx[n_1,n_2]} - m_{x[n_1]}m^*_{x[n_2]}$. Also, the variance of the random process $x[n]$ is $\sigma^2_{x[n]} = \gamma_{xx}[n, n]$. Here are a few important properties of random processes:

- **Summation of expectations.** $\mathbb{E}[x[n] + y[n]] = \mathbb{E}[x[n]] + \mathbb{E}[y[n]]$ always, because of the linearity of the expectation operator.
- **Multiplication by a constant.** $\mathbb{E}[ax[n]] = a\mathbb{E}[x[n]]$, again because of the linearity of expectation.
- **Summation of variances.** $\mathbb{V}\text{ar}[x[n]+y[n]] = \mathbb{V}\text{ar}[x[n]]+\mathbb{V}\text{ar}[y[n]]$ only if the random processes $x[n]$ and $y[n]$ are statistically independent.

4.3.2 Stationarity

Consider the three random processes with sample functions shown in Figs. 4.4, 4.5, and 4.6. The sample functions in Fig. 4.4 appear to be more or less the same over all time, and for these sample functions the time origin (*i.e.* $n = 0$) could be shifted without visibly affecting the properties of the sample functions. In contrast, the sample functions in Fig. 4.5 appear to be drifting upward after about Sample 50, while the sample functions in Fig. 4.6 appear to become more variable after about Sample 50, and for these processes

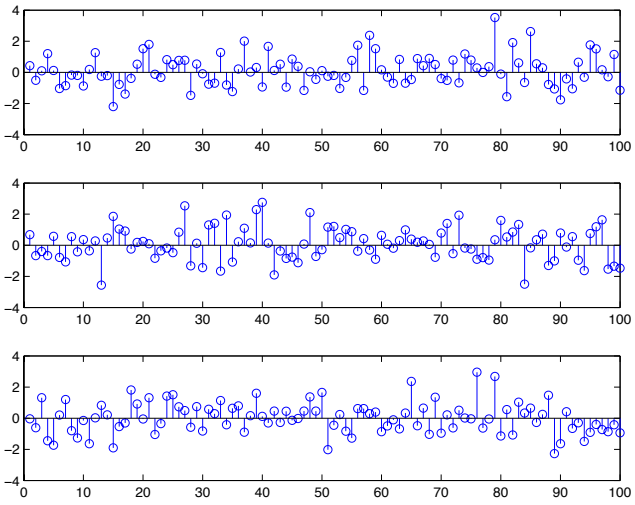


Figure 4.4: Sample functions of a stationary random process.

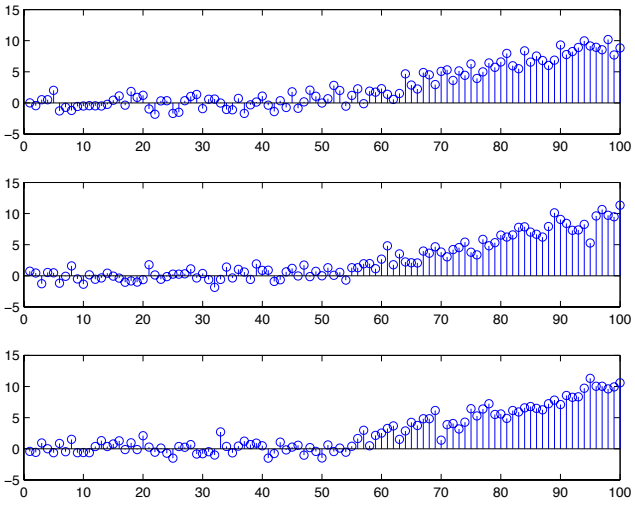


Figure 4.5: Sample functions of a nonstationary random process with increasing mean.

the temporal origin is meaningful (and is depicted at about 50 samples before the changes in the sample functions are observed). We characterize the property that the first process has and that the latter two do not have as *stationarity*.

We typically define two types of stationarity: wide-sense and strict-sense stationarity. A random process is *strict-sense stationary* (SSS) if

1. $\mathbb{E}[x[n]] = \mathbb{E}[x] = m_x$, a constant independent of time
2. $\phi_{xx}[n_1, n_2]$ depends only on $n_2 - n_1$, so we can write $\phi_{xx}[n_1, n_2] = \phi_{xx}[n_2 - n_1] \equiv \phi_{xx}[m]$

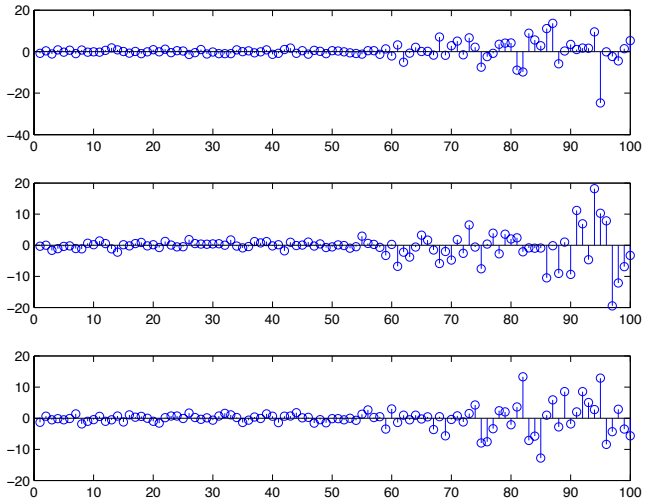


Figure 4.6: Sample functions of a nonstationary random process with increasing variance.

3. For any collection of times $n_1, n_2, n_3, \ldots$ etc., any joint moment such as $\mathbb{E}[x[n_1]x[n_2]x[n_3],\ldots]$ etc., the result depends only on the *differences* between the times in question, $n_2 - n_1, n_3 - n_2, n_3 - n_1, \ldots$, etc.

A random process $x[n]$ is *wide-sense stationary* (WSS) if only the first two conditions above are satisfied.

Stationarity in general (*i.e.* "strict-sense stationarity") requires that similar relations hold true for higher order moments of the random process as well. It is usually quite hard to prove. Knowing that a random process is WSS will be good enough for us almost all of the time. Conceptually, stationarity means that the location of the time origin does not affect the statistics of the process, as noted above.

4.3.3 Time averages

While ensemble averages are well defined and mathematically consistent, they are not very helpful in practice because we never can look at multiple sample functions, as mentioned above. Instead, we estimate ensemble averages by their corresponding time averages, which are meaningful only if a random process is stationary. Specifically, for a stationary random process, we define the time average of an arbitrary function $g(\cdot)$ of the random process $x[n]$ as

$$< g(x[n]) >= \lim_{N\to\infty} \frac{1}{2N+1} \sum_{n=-N}^{N} g(x[n]) \tag{4.15}$$

Note that the random process must be wide-sense stationary in order for the limit of this sum to converge. Figure 4.7 compares the samples in the ensemble used to compute a time average (which are horizontally arranged, in red, within a single sample function) with the samples used to compute an ensemble average (which are vertically arranged, in green, and which represent a single time across all sample functions).

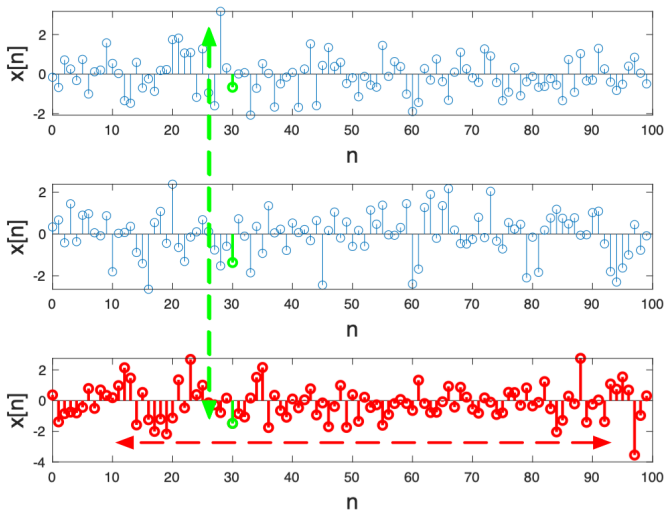


Figure 4.7: Comparison of samples used to compute time averages (horizontally arranged, in red) with those used to compute ensemble averages (vertically arranged, in green).

Some important time averages include

- The *time-averaged mean* of $x[n]$ is

$$< x[n] >= \lim_{N\to\infty} \frac{1}{2N+1} \sum_{n=-N}^{N} x[n] \tag{4.16}$$

  The mean, of course, represents the average over all time of the values of x.

- The *time-averaged autocorrelation* of $x[n]$

$$< x[n]x^*[n+m] >= \lim_{N\to\infty} \frac{1}{2N+1} \sum_{n=-N}^{N} x[n]x^*[n+m] \tag{4.17}$$

  where the asterisk symbol $(*)$ indicates complex conjugation, as before.

Other time averages can be computed in the same fashion.

4.3.4 Ergodicity

Since the theory of random processes is based on ensemble averages but we must estimate the ensemble averages by measuring time averages, it would be very helpful if the corresponding time and ensemble averages are all equal. In other words, we would like it if

$$\mathbb{E}[g(x[n])] =< g(x[n]) > \tag{4.18}$$

and more generally

$$\mathbb{E}[g(x[n_1], x[n_2], x[n_3], \ldots x[n_N])] =< g(x[n_1], x[n_2], x[n_3], \ldots x[n_N]) > \tag{4.19}$$

for all functions $g(\cdot)$ operating on all sample functions of the random process $x[n]$, and all N. This property is called *ergodicity*. More specifically, a random process is "ergodic" if all

of its corresponding time and ensemble averages are equal. Ergodicity is generally hard to prove. Conceptually, ergodicity means that the statistical attributes of each sample function is reincarnated in each of the other sample functions sooner or later. While most stationary random processes are also ergodic, there are some spectacular exceptions to this rule.

Example 4.1 (Ergodicity of the random-phase cosine) Consider the example of the random phase continuous-time cosine $x(t) = \cos(\omega t + \phi)$, where the phase ϕ is a random parameter that has a uniform pdf between values of $\phi = 0$ and $\phi = 2\pi$. We will first calculate the ensemble average mean and autocorrelation:

$$\mathbb{E}[x(t)] = \int_{-\infty}^{\infty} x(t) p_{x(t)}(\alpha) d\alpha$$

Note here that the probability density function specifies the distribution of values of the amplitude of $x(t)$ and that t itself is not a random variable. The expected value of $x(t)$ is much more easily obtained by conditioning it on the random phase parameter:

$$\mathbb{E}[x(t)] = \int_{-\infty}^{\infty} \mathbb{E}[x(t)|\phi = \alpha] p_{\phi}(\alpha) d\alpha = \int_{0}^{2\pi} \cos(\omega t + \alpha) \frac{1}{2\pi} d\alpha = 0$$

(In evaluating the last integral, note again that the variable being integrated is α and not t). Now let us consider the ensemble-average autocorrelation function for the (real) random-phase cosine. We will use the parametric evaluation as before:

$$\mathbb{E}[x(t_1)x(t_2)] = R_{xx}(t_1, t_2) = \int_{0}^{2\pi} \mathbb{E}[(\cos(\omega t_1 + \phi)\cos(\omega t_2 + \phi)|\phi = \alpha)] p_{\phi}(\alpha) d\alpha$$

To evaluate this integral we will use the familiar trig identity

$$\cos(x)\cos(y) = \frac{1}{2}(\cos(x + y) + \cos(x - y))$$

which produces

$$R_{xx}(t_1, t_2) = \int_{0}^{2\pi} \frac{1}{2} \cos(\omega(t_1 + t_2) + 2\alpha) p_{\phi}(\alpha) d\alpha + \int_{0}^{2\pi} \frac{1}{2} \cos(\omega(t_1 - t_2)) p_{\phi}(\alpha) d\alpha$$

Recalling that $p_{\phi}(\alpha) = \frac{1}{2\pi}$ for $0 \leq \alpha < 2\pi$, we note that the first integral integrates to zero as we are integrating the cosine over exactly two periods. The second integral actually does not depend on α at all, and we obtain

$$R_{xx}(t_1, t_2) = \frac{1}{2} \cos(\omega(t_1 - t_2)) \int_{0}^{2\pi} \frac{1}{2\pi} d\alpha = \frac{1}{2} \cos(\omega(t_1 - t_2)) = \frac{1}{2} \cos(\omega\tau) = R_{xx}(\tau)$$

where $\tau = t_1 - t_2 = t_2 - t_1$ because the cosine is an even function. Note that because the random-phase cosine has a mean that is a constant and an autocorrelation that depends only on $\tau = t_2 - t_1$, the random-phase cosine is WSS.

Now, using similar techniques, we can obtain the time-averaged mean and autocorrelation function for the random process $x(t) = \cos(\omega t + \phi)$ where this time ϕ is an

unknown but arbitrary constant. Specifically, we can calculate the continuous-time averages

$$< x(t) >= \lim_{T\to\infty} \frac{1}{2T} \int_{-T}^{T} \cos(\omega t + \phi) dt \text{ and}$$

$$< x(t)x(t+\tau) >= \lim_{T\to\infty} \frac{1}{2T} \int_{-T}^{T} \cos(\omega t + \phi)\cos(\omega(t+\tau) + \phi) dt$$

We obtain the same answers:

$$\mathbb{E}[x(t)] =< x(t) >= 0 \text{ and}$$

$$R_{xx}(\tau) = \mathbb{E}[x(t)x(t+\tau)] =< x(t)x(t+\tau) >= \frac{1}{2}\cos(\omega\tau)$$

indicating that the random-phase cosine is indeed ergodic, at least in the second-order sense.

Example 4.2 (Ergodicity of the constant random process.) Now consider the very simple discrete-time random process $x[n] = k$, where k is in turn a random variable, for example a Gaussian with mean zero and variance 1. It can easily be shown that $x[n]$ is a stationary random process with the ensemble-average mean equal to the mean of the Gaussian, zero. Note that the time-average means of each of the sample functions are all different (and equal to the value of the sample function, of course). But the time average means are generally not the same as the ensemble average mean which would be zero in this case. Hence, this process is stationary but not ergodic. This is unusual, but there are a few other similar examples (mostly nearly as contrived) that can be developed.

4.4 Gaussian random processes

A random process is *Gaussian* if the random variables formed by sampling the random process at specific instants of time all are jointly Gaussian random variables. Although this may seem like a circular definition, it really isn't ... to test for Gaussianity, it is only necessary to show (or be told) that the pdf of the corresponding random variables are really Gaussian. In other words, individual random variables are Gaussian if

$$p_x(\alpha) = \frac{1}{\sqrt{2\pi\sigma_x^2}} \exp\left(\frac{(\alpha - m_x)^2}{2\sigma_x^2}\right) \tag{4.20}$$

For Gaussian random vectors we generally use the matrix form of the pdf:

$$p_{\mathbf{x}}(\alpha) = \frac{1}{(2\pi)^{N/2}|\mathbf{C}_x|^{1/2}} \exp\left(-\frac{1}{2}(\alpha - m_{\mathbf{x}})^T \mathbf{C}_x^{-1}(\alpha - m_{\mathbf{x}})\right) \tag{4.21}$$

where $m_{\mathbf{x}}$ is the N-dimensional mean vector, $\mathbf{C}_{\mathbf{x}}$ is the $N \times N$ covariance matrix of the jointly Gaussian random variables, and $|\cdot|$ is the determinant operator.

A Gaussian random process is not necessarily stationary, but if it is wide-sense stationary, it is also strict-sense stationary. (This is because higher-order moments of Gaussian random variables can always be factored into first-order and second-order moments.)

4.5 Power Spectral Density Functions

The *power spectral density function* is the mathematical representation that enables us to characterize the frequency distribution of a random process. As you may recall, the total energy of a discrete-time function may be represented as

$$\text{Energy} = \sum_{n=-\infty}^{\infty} |x[n]|^2 \tag{4.22}$$

The problem with the use of energy is that all stationary random processes have infinite energy. This is because stationarity implies that $\mathbb{E}[|x[n]|^2] = \phi_{xx}(0)$ is a constant that is independent of n, so when we sum that quantity over all time the result becomes infinite. (Deterministic periodic time functions also have infinite energy for basically the same reasons.)

A more appropriate measure is *power*, which refers to energy divided by time. We can define the power associated with a sample function of a stationary random process $x[n]$ in terms of the time average

$$\text{Power} = \lim_{N\to\infty} \frac{1}{2N+1} \sum_{n=-N}^{N} |x[n]|^2 \tag{4.23}$$

Note that this is also the time average of the quantity $|x[n]|^2$, so if the random process is ergodic, power will also be equal to the ensemble average $\mathbb{E}[|x[n]|^2]$, or $\mathbb{E}[x^2[n]]$ if the sample functions of the random process are all real. While the total power is obtained from this average, we will consider now the very important *power spectral density function*, which describes how the total power is distributed over the constituent frequency components. To provide greatest insight, we will consider three approaches to the definition of power spectral density functions.

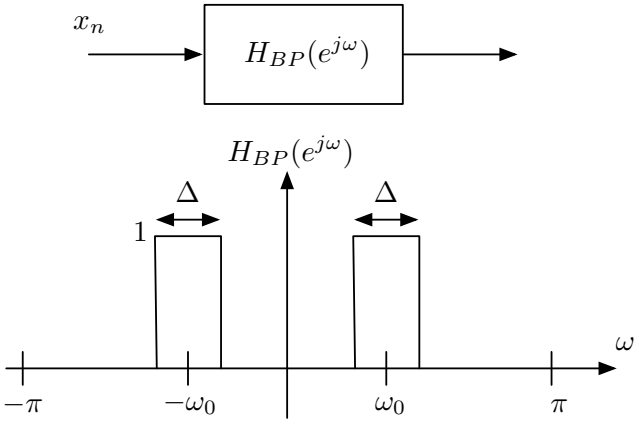


Figure 4.8: The linear filter definition of power spectral density functions.

Physical definition. Consider the system in Fig. 4.8. Assume that the input to the filter $x[n]$ is a stationary random process with a particular distribution of power respect to frequency, and with a total power equal to σ_x^2. The filter is an ideal bandpass filter with

bandwidth Δ and center frequency ω_0. As Δ becomes small, the amount of power in the signal that emerges from the output of the filter is equal by definition to

$$\text{Power in band} = \frac{2\Delta}{2\pi} P_{xx}(\omega_0) \tag{4.24}$$

In the expression above, the function $P_{xx}(\omega_0)$ represents the power spectral density function of the random process $x[n]$ at the frequency ω_0. The factor of 2 in the numerator represents the fact that both positive and negative frequencies are passed through the filter. The factor of $1/2\pi$ is needed because the total power in $x[n]$ is obtained by integrating $P_{xx}(\omega)$ over all frequencies:

$$\text{Total power} = \frac{1}{2\pi} \int_{-\pi}^{\pi} P_{xx}(\omega) d\omega = \sigma_x^2 + m_x^2 \tag{4.25}$$

Similarly, the power between two frequencies $\omega_1 \leq |\omega| \leq \omega_2$ (considering both positive and negative frequencies) is

$$\text{Power in frequency band} = \frac{2}{2\pi} \int_{\omega_1}^{\omega_2} P_{xx}(\omega) d\omega \tag{4.26}$$

In other words, the power spectral density function is proportional to the amount of power in a random process that is present at a given frequency.

Fourier transform definition. Recall that we defined the total power as

$$\text{Power} = \lim_{N\to\infty} \frac{1}{2N+1} \sum_{n=-N}^{N} |x[n]|^2 \tag{4.27}$$

From Parseval's theorem we can also derive the power spectral density function as what we get when we compute the magnitude squared of the DTFT of the sample functions windowed over a finite duration, divided by the duration of the observations. We then compute the expected value of the result over all of the sample functions of a random process, and then taking the limit as the duration goes to infinity. Hence we can define the power spectral density function $P_{xx}(\omega)$ of a real random process $x[n]$ as

$$P_{xx}(\omega) = \lim_{N\to\infty} \frac{1}{2N+1} \mathbb{E}\left[\left| \sum_{n=-N}^{N} x[n] e^{-j\omega n} \right|^2 \right] \tag{4.28}$$

Note that this expresses the power spectral density function as energy per unit time, broken down into frequency components

Autocorrelation function definition. Again assuming that the random process $x[n]$ is real, by expanding the magnitude squared expression (as in Eq. (4.28)), we obtain

$$P_{xx}(\omega) = \lim_{N\to\infty} \frac{1}{2N+1} \mathbb{E}\left[\left| \sum_{n=-N}^{N} x[n] e^{-j\omega n} \right|^2 \right] = \lim_{N\to\infty} \frac{1}{2N+1} \mathbb{E}\left[\left(\sum_{n=-N}^{N} x[n] e^{-j\omega n} \right) \left(\sum_{l=-N}^{N} x[l] e^{-j\omega l} \right)^* \right] \tag{4.29}$$

$$\text{or } P_{xx}(\omega) = \lim_{N\to\infty} \frac{1}{2N+1} \mathbb{E}\left[\sum_{n=-N}^{N} \sum_{l=-n}^{N} x[n] x^*[l] e^{-j\omega(n-l)} \right] \tag{4.30}$$

Letting $m = l - n$ we can rewrite this as

$$P_{xx}(\omega) = \lim_{N\to\infty} \frac{1}{2N+1} \mathbb{E}\left[\sum_{n=-N}^{N} x[n]x^*[l]e^{j\omega m}\right] = \sum_{n=-\infty}^{\infty} \sum_{m=n-N}^{n+N} \phi_{xx}[m]e^{j\omega m} \tag{4.31}$$

Replacing the variable m by $-m$ we obtain

$$P_{xx}(\omega) = \sum_{m=-\infty}^{\infty} \phi_{xx}[-m]e^{-j\omega m} = \sum_{m=-\infty}^{\infty} \phi_{xx}[m]e^{-j\omega m} \tag{4.32}$$

The latter equality is valid because, again, the random process $x[n]$ is assumed to be real, so that $\phi_{xx}[m]$ is even. In other words, the power spectral density function of a real random process is the DTFT of the corresponding autocorrelation function. This relationship is known as the Wiener-Khinchin theorem, and it is typically the easiest way to evaluate power spectral density functions.

A random process is said to be *white* if it has a power spectral density function that is a constant over all frequency, *i.e.*

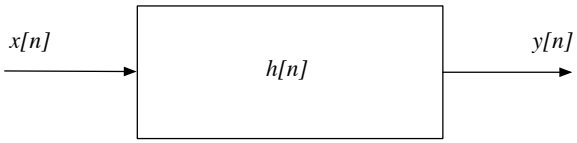
$$P_{xx}(\omega) = \sigma_x^2 \tag{4.33}$$

This implies that the autocorrelation function of a white random process is an impulse:

$$\phi_{xx}[m] = \sigma_x^2 \delta[m] \tag{4.34}$$

Since a white random process must also be zero mean (otherwise there would be a DC bias and hence an impulse in the power spectral density function at $\omega = 0$), this implies that the values of a sample function of a white random process at different times are all statistically independent.

4.6 Random processes and linear filters



We are often interested in what happens to the statistics of a random process when it is passed through a linear filter. Specifically, let us assume that a real wide-sense stationary random process $x[n]$ is input to a linear filter with unit sample response $h[n]$ producing the output process $y[n]$. As in the case of deterministic signals, the input and output are related by the convolution sum

$$y[n] = \sum_{k=-\infty}^{\infty} h[k]x[n-k] \tag{4.35}$$

The *mean* of the output is easily obtained as

$$m_y = \mathbb{E}[y[n]] = \mathbb{E}\left[\sum_{n=-\infty}^{\infty} h[k]x[n-k]\right] = \sum_{n=-\infty}^{\infty} h[k]\mathbb{E}[x[n-k]] = m_x \sum_{k=-\infty}^{\infty} h[k]$$

The *autocorrelation* of the output is also straightforward to compute but requires a little more algebra:

$$\phi_{yy}[m] = \mathbb{E}[y[n]y[n+m]] = \mathbb{E}\left[\sum_{k=-\infty}^{\infty} h[k]x[n-k] \sum_{l=-\infty}^{\infty} h[l]x[n+m-l]\right] \text{ or} \tag{4.36}$$

$$\phi_{yy}[m] = \sum_{k=-\infty}^{\infty}\sum_{l=-\infty}^{\infty} h[k]h[l]\mathbb{E}[x[n-k]x[n+m-l]] = \sum_{k=-\infty}^{\infty}\sum_{l=-\infty}^{\infty} h[k]h[l]\phi_{xx}[m+k-l] \tag{4.37}$$

Now let $r = l - k$ or $l = r + k$. Substituting, we obtain

$$\phi_{yy}[m] = \sum_{r} \phi_{xx}[m-r] \sum_{k} h[k]h[r+k] \tag{4.38}$$

The inner sum, $\sum_k h[k]h[r+k]$, can be considered to be the convolution of the unit sample response $h[n]$ with itself time-reversed, which is actually the (unnormalized) autocorrelation function of the deterministic function $h[n]$. Recognizing this, we can rewrite the equation for the output autocorrelation as

$$\phi_{yy}[m] = \sum_{r} \phi_{xx}[m-r]\phi_{hh}[r] = \phi_{xx}[m] * \phi_{hh}[m] = \phi_{xx}[m] * h[m] * h[-m] \tag{4.39}$$

Hence, the autocorrelation function of the output random process is obtained by convolving the autocorrelation function of the input convolved with the unit sample response and convolved again with the unit sample response time reversed. Finally, noting the Fourier transform pairs

$$h[m] \Leftrightarrow H(e^{j\omega})$$
$$h[-m] \Leftrightarrow H^*(e^{j\omega})$$
$$\phi_{xx}[m] \Leftrightarrow P_{xx}(\omega)$$

we can obtain a relationship between the input and output *power spectral density functions*:

$$\phi_{yy}[m] = \phi_{xx}[m] * h[m] * h[-m] \Leftrightarrow P_{xx}(\omega)H(e^{j\omega})H^*(e^{j\omega}) = P_{xx}(\omega)\left|H(e^{j\omega})\right|^2 \tag{4.40}$$

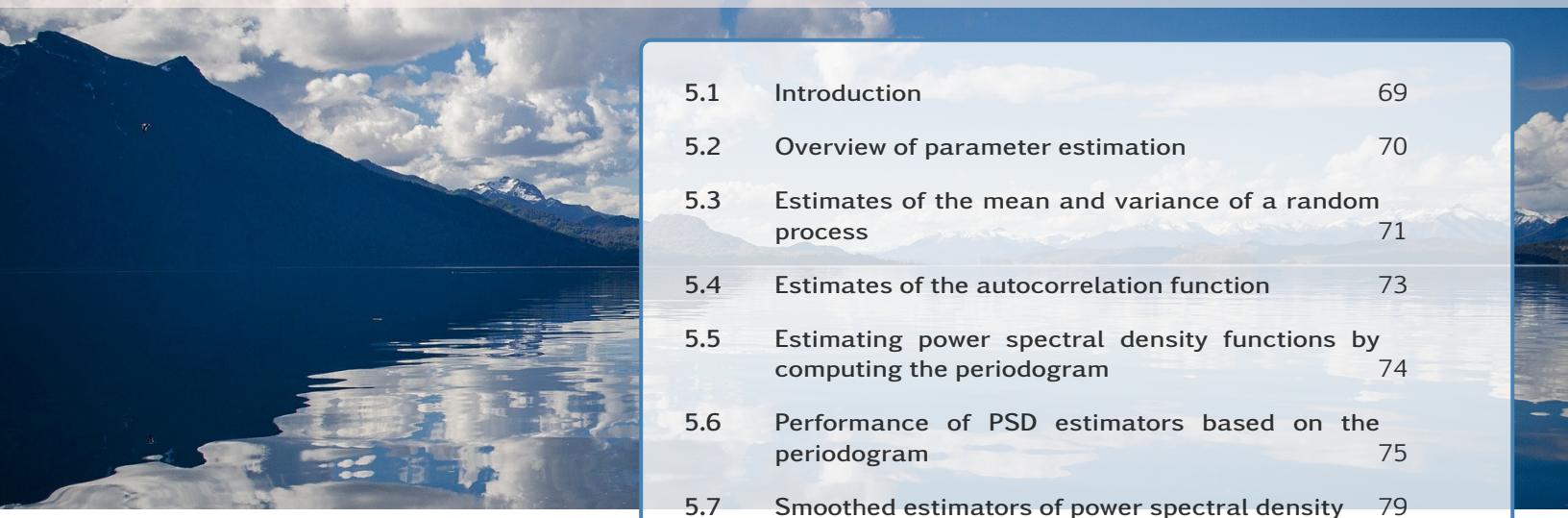
In other words, the output PSD equals the input PSD multiplied by the magnitude squared of the transfer function of the filter. It can also be shown quite easily that the *cross-correlation function* between the input and the output can be written as

$$\phi_{xy}[m] = \mathbb{E}[x[n]y^*[n+m]] = \phi_{xx}[m] * h[m] \tag{4.41}$$

and

$$P_{xy}(\omega) = P_{xx}(\omega)H(e^{j\omega}) \tag{4.42}$$

5. Classical Power Spectral Density Estimation



The power spectral density function is the means by which we characterize the frequency content of a random process. Nevertheless, estimating power spectra accurately is a surprisingly difficult problem. This chapter describes how we can use "classical" methods to estimate power spectral density functions accurately, despite some challenging fundamental limitations.

5.1 Introduction

The power spectral density function (PSD function) is the means by which we characterize the frequency content of a random process $x[n]$. As it turns out, estimating power spectral density functions in a fashion that is unbiased and consistent is surprisingly difficult to do, even though the problem of spectral density function estimation has received a great deal of attention for decades. This difficulty is reflected in the fact that there are currently a large number of spectral estimation algorithms that are widely used for different types of random processes (*cf.* Kay and Marple, 1991).

There are two major types of approaches to solving the power spectral density estimation problem. The first family of approaches, which are referred to collectively as "classical" approaches, is based on estimation of the Fourier transform of the autocorrelation function, followed by further processing to reduce the variance of this estimate. The second type of approach, which will be addressed in the following two chapters, characterizes the spectral density function as a parametric function, and the individual parameters of this function are estimated using various techniques.

In the sections below we begin by discussing various aspects of parameter estimation itself, including what makes an estimation algorithm a good algorithm. We continue with some discussion of estimators for the mean, variance, and autocorrelation functions of a random process. Next we define the *periodogram*, which is an estimate of the Fourier transform of the autocorrelation function of a random process, and consequently the basis for estimating power spectral density functions. We show that the variance of the

periodogram is unacceptably high, but that this variance can be reduced through the use of multiple types of smoothing.[1]

5.2 Overview of parameter estimation

It is frequently necessary to estimate parameters of random processes. These parameters could be truly deterministic parameters such as the phase of the random-phase cosine that we have studied in some detail, and they also could be statistics of random processes such as their expected value or the coefficients of the autocorrelation function. We begin by discussing the general process of parameter estimation, and we then discuss some attributes of good estimators. Subsequently we provide some simple examples of estimates of statistical parameters of random processes.

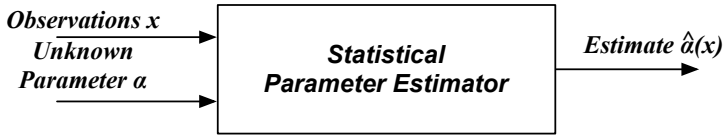


Figure 5.1: Conceptual diagram of the estimation process.

Consider a deterministic but unknown parameter α that characterizes a random process. Typically these parameters are random because we only can estimate them by observing a finite portion of a single sample function of a random process that they affect. The estimate that we obtain will vary from trial to trial because the samples of the random process from which we derive the estimate are themselves stochastic. We refer to the *estimate* of the random parameter as $\hat{\alpha}$. Note that the "hat" symbol over the $\hat{\alpha}$ is used to distinguish it from the parameter α itself.

We will focus on the following common measures of goodness of parameter estimates:

Bias. The bias of an estimate of a random parameter is defined by

$$B = \alpha - \mathbb{E}[\hat{\alpha}] \tag{5.1}$$

Bias, obviously, represents the extent to which the average value of the estimate differs from the true value of the parameter that is being estimated. While zero bias is typically preferred, it is acceptable for a parameter to have a known bias, since that bias can be added to the estimate to obtain a new estimate with zero bias.

Variance. The variance of the estimate is defined by

$$\mathbb{V}\text{ar}[\hat{\alpha}] = \mathbb{E}[(\hat{\alpha} - \mathbb{E}[\hat{\alpha}])^2] = \sigma_{\hat{\alpha}}^2 \tag{5.2}$$

As always, the variance describes how spread about the mean the individual samples of the random parameter are likely to be. We will frequently observe a tradeoff between bias and variance in choosing which estimate to use for a particular parameter.

[1] This material is based on the corresponding discussion in Chapter 11 of Oppenheim and Schafer (1975).

Mean square error (MSE). The mean square error of an estimate is defined as

$$\mathrm{MSE}(\hat{\alpha}) = \mathbb{E}[(\hat{\alpha} - \alpha)^2] = \sigma_{\hat{\alpha}}^2 + B^2 \tag{5.3}$$

Consistency. An estimator is consistent if

$$\lim_{N\to\infty} B = 0 \text{ and } \lim_{N\to\infty} \sigma_{\hat{\alpha}}^2 = 0 \tag{5.4}$$

where N is the number of observed samples of the random process. Consistency is an important measure of the asymptotic properties of estimates. Unsurprisingly, we would normally want to have the bias, variance, and MSE be as small as possible, and for the estimator to be consistent.

5.3 Estimates of the mean and variance of a random process

Let us now consider a discrete-time random process $x[n]$. We will assume that the samples of $x[n]$ are statistically independent and identically distributed, with true mean and variance m_x and σ_x^2. (Note that this process would be white if its mean were equal to zero.)

A pair of "reasonable" estimates of the mean and variance of $x[n]$ are the corresponding sample mean and sample variance:

$$\hat{m}_x \equiv \frac{1}{N}\sum_{n=0}^{N-1} x[n] \text{ (the sample mean)} \tag{5.5}$$

$$\hat{\sigma}_x^2 \equiv \frac{1}{N}\sum_{n=0}^{N-1} (x[n] - \hat{m}_x)^2 \text{ (the sample variance)} \tag{5.6}$$

Now let's look at these estimates in terms of some of the attributes of estimators defined above.

The expected value of the sample mean. We can obtain the statistical average of the sample mean quite easily:

$$\mathbb{E}[\hat{m}_x] = \mathbb{E}\left[\frac{1}{N}\sum_{n=0}^{N-1} x[n]\right] = \frac{1}{N}\sum_{n=0}^{N-1} \mathbb{E}[x[n]] = \frac{1}{N}\sum_{n=0}^{N-1} m_x = m_x \tag{5.7}$$

Consequently $B = \mathbb{E}[\hat{m}_x] - m_x = 0$ for the sample mean.

The variance of the sample mean. The variance of the sample mean is obtained by computing

$$\sigma_{\hat{m}_x}^2 = \mathbb{E}[(\hat{m}_x - m_x)^2] = \mathbb{E}[\hat{m}_x^2] - m_x^2 \tag{5.8}$$

$$\sigma_{\hat{m}_x}^2 = \mathbb{E}\left[\frac{1}{N}\sum_{n=0}^{N-1} x[n]\frac{1}{N}\sum_{l=0}^{N-1} x[l]\right] - m_x^2 = \frac{1}{N^2}\left\{\sum_{n=0}^{N-1}\mathbb{E}[x^2[n]] + \sum_{n=0}^{N-1}\sum_{l=0,l\neq n}^{N-1}\mathbb{E}[x[n]]\mathbb{E}[x[l]]\right\} - m_x^2 \tag{5.9}$$

$$\sigma_{\hat{m}_x}^2 = \frac{\sigma_x^2 + m_x^2}{N} + \frac{m_x^2(N-1)}{N} - m_x^2 = \frac{\sigma_x^2}{N} \tag{5.10}$$

We can also see that the sample mean is consistent because its bias and variance asymptote to zero as the number of samples increases to infinity. (Actually, the bias is always zero, as noted above.)

Mean of the sample variance. The mean of the sample variance is obtained by

$$\mathbb{E}[\hat{\sigma}_x^2] = \mathbb{E}\left[\frac{1}{N}\sum_{n=0}^{N-1}(x[n]-\hat{m}_x)^2\right] = \frac{1}{N}\sum_{n=0}^{N-1}\mathbb{E}\left\{x^2[n]-2x[n]\hat{m}_x+(\hat{m}_x)^2\right\} \tag{5.11}$$

where again

$$\hat{m}_x = \frac{1}{N}\sum_{n=0}^{N-1}x[n] \tag{5.12}$$

The first term of the expression for $\mathbb{E}[\hat{\sigma}_x^2]$ is

$$\frac{1}{N}\sum_{n=0}^{N-1}\mathbb{E}[x^2[n]] = \frac{1}{N}\sum_{n=0}^{N-1}(\sigma_x^2+m_x^2) = \sigma_x^2+m_x^2 \tag{5.13}$$

The second term of the expression for $\mathbb{E}[\hat{\sigma}_x^2]$ is

$$\frac{1}{N}\sum_{n=0}^{N-1}\mathbb{E}[-2x[n]\hat{m}_x] = \frac{-2}{N}\sum_{n=0}^{N-1}\mathbb{E}\left\{x[n]\frac{1}{N}\sum_{l=0}^{N-1}x[l]\right\} = \frac{-2}{N^2}\sum_{n=0}^{N-1}\sum_{l=0}^{N-1}\mathbb{E}[x[n]x[l]] \tag{5.14}$$

$$= \frac{-2}{N^2}\sum_{n=0}^{N-1}\mathbb{E}[x^2[n]] - \frac{-2}{N^2}\sum_{n=0}^{N-1}\sum_{l=0,l\neq n}^{N-1}\mathbb{E}[x[n]x[l]] = -\frac{2}{N}(\sigma_x^2+m_x^2) - \frac{2}{N}(N-1)m_x^2 = -2\frac{\sigma_x^2}{N} - 2m_x^2 \tag{5.15}$$

The third term of the expression for $\mathbb{E}[\hat{\sigma}_x^2]$ is

$$\frac{1}{N}\sum_{n=0}^{N-1}\mathbb{E}\left[\hat{m}_x^2\right] = \frac{1}{N}\sum_{n=0}^{N-1}\frac{1}{N}\sum_{l=0}^{N-1}\frac{1}{N}\sum_{r=0}^{N-1}\mathbb{E}[x[l]x[r]] = \frac{1}{N}\sum_{n=0}^{N-1}\left\{\frac{1}{N^2}\sum_{l=0}^{N-1}\mathbb{E}[x^2[l]] + \sum_{r=0}^{N-1}\sum_{l=0,l\neq r}^{N-1}\mathbb{E}[x[r]x[l]]\right\} \tag{5.16}$$

$$= \frac{1}{N}\sum_{n=0}^{N-1}\left\{\frac{\sigma_x^2+m_x^2}{N} + \frac{(N-1)m_x^2}{N}\right\} = \frac{\sigma_x^2}{N}+m_x^2 \tag{5.17}$$

Adding the three terms together produces

$$\mathbb{E}[\hat{\sigma}_x^2] = \sigma_x^2+m_x^2-2\frac{\sigma_x^2}{N}-2m_x^2+\frac{\sigma_x^2}{N}+m_x^2 = \frac{N-1}{N}\sigma_x^2 \tag{5.18}$$

Note that the variance of the sample mean does not equal the mean of the sample variance of $x[n]$.

Variance of the sample variance. The derivation of the variance of the sample variance goes beyond the scope of the present discussion. Suffice it to say that it can be shown that

$$\mathbb{V}\mathrm{ar}[\hat{\sigma}_x^2] = \frac{1}{N}\left\{\mathbb{E}[x^4[n]] - (\mathbb{E}[x^2[n]]^2)\right\} \tag{5.19}$$

This suggests that the sample variance is a consistent estimate of the variance, even though it is a biased one. If the process $x[n]$ is Gaussian, we can break the fourth moment

$\mathbb{E}[x^4[n]]$ into the sums and products of second-order moments like $\mathbb{E}[x^2[n]]$ according to the moment factoring theorem; otherwise there is no general approach to simplifying this term. We discuss some of the issues associated with estimating variance in the next chapter when we consider the estimation of power spectral density functions.

5.4 Estimates of the autocorrelation function

Let us now turn our attention to the problem of estimating autocorrelation functions. Consider a wide-sense stationary ergodic random process $x[n]$ with mean $m_x = 0$, so that we know that the power spectral density functions exists without delta functions. Under these conditions we can write

$$\gamma_{xx}[m] = \phi_{xx}[m] = \mathbb{E}[x[n]x[n+m]] =< x[n]x[n+m] > \tag{5.20}$$

In general, given a sequence of N consecutive samples of $x[n]$ we can write the sample autocorrelation sequence

$$c'_{xx}[m] = \frac{1}{N-|m|} \sum_{n=0}^{N-1-|m|} x[n]x[n+m], \ |m| < N \tag{5.21}$$

Note that the number of samples involved in computing the sample autocorrelation function $c'_{xx}[m]$ depends on the value of the lag parameter m and is equal to $N-|m|$. It is easy to show that this sample autocorrelation function is unbiased:

$$\mathbb{E}[c'_{xx}[m]] = \frac{1}{N-|m|} E\left[\sum_{n=0}^{N-1-|m|} x[n]x[n+m]\right] = \frac{1}{N-|m|} \sum_{n=0}^{N-1-|m|} \mathbb{E}[x[n]x[n+m]] = \phi_{xx}[m] \tag{5.22}$$

It can be shown (*cf.* Jenkins and Watts) that the variance of $c'_{xx}[m]$ can be approximated by

$$\mathbb{V}\text{ar}[c'_{xx}[m]] \approx \frac{1}{[N-|m|]^2} \sum_{l=-\infty}^{\infty} [\phi^2_{xx}[l] + \phi_{xx}[l+m]\phi_{xx}[l-m]] \tag{5.23}$$

with the expression above being valid for $|m| \ll N$. So for fixed m and large N, the variance of the sample autocorrelation function $c'_{xx}[m]$ goes to zero, leading to a consistent estimator.

A second plausible estimator of the autocorrelation function is

$$c_{xx}[m] = \frac{1}{N} \sum_{n=0}^{N-1-|m|} x[n]x[n+m] = \frac{N-|m|}{N} c'_{xx}[m] \tag{5.24}$$

(Note the absence of the prime symbol on the left hand side of the equation above.) From Eq. (5.24) we can see that

$$\mathbb{E}[c_{xx}[m]] = \frac{N-|m|}{N} \phi_{xx}[m] \tag{5.25}$$

Hence the estimator $c_{xx}[m]$ is biased but it is asymptotically unbiased because

$$B = \phi_{xx}[m] - \mathbb{E}[c_{xx}[m]] = \phi_{xx}[m]\left(1 - \frac{N-|m|}{N}\right) = \frac{|m|}{N}\phi_{xx}[m] \tag{5.26}$$

Since

$$c_{xx}[m] = \frac{N-|m|}{N} c'_{xx}[m] \tag{5.27}$$

we can write its variance using Eq. (5.23) as

$$\mathbb{V}\mathrm{ar}[c_{xx}[m]] = \frac{(N-|m|)^2}{N^2} \mathbb{V}\mathrm{ar}[c'_{xx}[m]] \tag{5.28}$$

or

$$\mathbb{V}\mathrm{ar}[c_{xx}[m]] \approx \frac{1}{N^2} \sum_{l=-\infty}^{\infty} [\phi_{xx}^2[l] + \phi_{xx}[l+m]\phi_{xx}[l-m]] \tag{5.29}$$

Note that for the biased estimator $c_{xx}[m]$, the variance of the estimate goes to zero as the number of samples becomes infinite for all values of m. In addition to a more fundamental consideration to be discussed in the following chapter, estimates of power spectral density functions tend to be based on the estimator $c_{xx}[m]$ even though it has a (known) bias because the variance of the estimator $c'_{xx}[m]$ can become arbitrarily large as $|m|$ approaches N. (Again, this happens because only a few samples are available for the calculation when $|m|$ approaches N.) As before, in comparing these two estimators we see a tradeoff between bias and variance: $c_{xx}[m]$ is biased while $c'_{xx}[m]$ is not, but the variance of $c_{xx}[m]$ asymptotes to zero for large N while that of $c'_{xx}[m]$ does not. These issues are especially important in estimating power spectral density functions, because the tails of estimates of $\phi_{xx}[m]$ contribute just as much to its Fourier transform as do the samples with small lags.

5.5 Estimating power spectral density functions by computing the periodogram

We noted in Chapter 4 that the Wiener-Khinchin theorem showed that the power spectral density (PSD) function is the Fourier transform of the autocorrelation function of a wide-sense stationary random process. We also discussed two ways of estimating the autocorrelation function, along with the tradeoff between bias and variance that results.

The most obvious approach to the estimation of the power spectral density function is to first obtain a consistent estimate of the autocorrelation function and then compute the DTFT of that sample autocorrelation function, applying the Wiener-Khinchin theorem. While both $c'_{xx}[m]$ and $c_{xx}[m]$ have means and variances that asymptote to zero as the number of samples goes to infinity, the biased estimator of the autocorrelation function $c_{xx}[m]$ is more widely used because it has been conjectured that the mean-square error of the biased estimator is less than that for the unbiased estimator for many practical cases. We will also discuss a more fundamental reason for using $c_{xx}[m]$ below. We define the *periodogram* $I_N(\omega)$ as the DTFT of the biased sample autocorrelation function $c_{xx}[m]$:

$$I_N(\omega) = \sum_{m=-(N-1)}^{N-1} c_{xx}[m] e^{-j\omega m} \tag{5.30}$$

Substituting the definition for $c_{xx}[m]$ from (5.24) we obtain

$$I_N(\omega) = \sum_{m=-(N-1)}^{N-1} \frac{1}{N} \sum_{n=0}^{N-1} x[n]x[n+m]e^{-j\omega m} = \frac{1}{N} \sum_{n=0}^{N-1} x[n] \sum_{m=-(N-1)}^{N-1} x[n+m]e^{-j\omega m} \tag{5.31}$$

Keep in mind in the above equation that $x[n]$ is assumed to be real. Now let $l = n + m$, so $m = l - n$:

$$I_N(\omega) = \frac{1}{N}\sum_{n=0}^{N-1} x[n] \sum_{l=n-(N-1)}^{n+(N-1)} x[l]e^{-j\omega(l-n)} = \frac{1}{N}\sum_{n=0}^{N-1} x[n]e^{+j\omega n} \sum_{l=0}^{N-1} x[l]e^{-j\omega l} = \frac{1}{N}X_N^*(e^{j\omega})X_N(e^{j\omega}) \tag{5.32}$$

where the subscript N in $X_N(e^{j\omega})$ is used as a reminder that we are computing the DTFT of only a finite segment of the time function. The limits on the inner sum were changed because $x[n]$ is nonzero only for $0 \le n \le N-1$, as noted above. This leads to our second way of thinking about the periodogram:

$$I_N(\omega) = \frac{1}{N}\left|X_N(e^{j\omega})\right|^2 \tag{5.33}$$

where as before $X_N(e^{j\omega})$ is the DTFT of the segment of the random process $x[n]$ for $0 \le n \le N-1$. As we observed in the discussions leading up to Eq. (4.28), the power spectral density function can be thought about as the expected value of the magnitude squared of the Fourier transform of a limited-duration segment of a random process divided by that duration, taking the limit as the duration is increased to infinity. In Eq. (5.33) we see that the periodogram can be thought of as the instantaneous approximation (without the expectation operation) of the true power spectral density, in that it is the magnitude squared of the Fourier transform of a limited-duration segment of a sample function of $x[n]$, again divided by the duration of that segment.

5.6 Performance of PSD estimators based on the periodogram

In this section we will consider the mean and and variance of the periodogram.

5.6.1 The mean of the periodogram

The mean of the periodogram is relatively easy to compute:

$$\mathbb{E}[I_N(\omega)] = \mathbb{E}\left[\sum_{m=-(N-1)}^{N-1} c_{xx}[m]e^{-j\omega m}\right] = \sum_{m=-(N-1)}^{N-1} \mathbb{E}\left[c_{xx}[m]\right]e^{-j\omega m} \tag{5.34}$$

Because the biased estimate of the autocorrelation function can be expressed as

$$c_{xx}[m] = \frac{1}{N}\sum_{n=0}^{N-1-|m|} x[n]x[n+m] = \frac{N-|m|}{N}c'_{xx}[m] \tag{5.35}$$

[*cf.* Eq. (5.24)], the expectation of the periodogram can be written as

$$\mathbb{E}[I_N(\omega)] = \sum_{m=-(N-1)}^{N-1} \mathbb{E}\left[c_{xx}[m]\right]e^{-j\omega m} = \sum_{m=-(N-1)}^{N-1} \frac{N-|m|}{N}\mathbb{E}[c'_{xx}[m]]e^{-j\omega m} \tag{5.36}$$

$$= \sum_{m=-(N-1)}^{N-1} \frac{N-|m|}{N}\phi_{xx}[m]e^{-j\omega m} \tag{5.37}$$

as $c'_{xx}[m]$ is an unbiased estimate of $\phi_{xx}[m]$. Another way to look at this calculation is to think of the expected value $\mathbb{E}[c_{xx}[m]]$ as being a windowed version of $\phi_{xx}[m]$:

$$\mathbb{E}[c_{xx}[m]] = \begin{cases} \frac{N-|m|}{N}\phi_{xx}[m], & |m| < N \\ 0, & \text{otherwise} \end{cases} \tag{5.38}$$

Defining the *Bartlett window* as

$$w_B[m] = \begin{cases} \frac{N-|m|}{N}, & |m| < N \\ 0, & \text{otherwise} \end{cases} \tag{5.39}$$

We can write the expected value of the biased autocorrelation function as

$$\mathbb{E}[c_{xx}[m]] = w_B[m]\phi_{xx}[m] \tag{5.40}$$

Hence we can redefine the expected value of the periodogram as

$$\mathbb{E}[I_N(\omega)] = \sum_{m=-(N-1)}^{N-1} w_B[m]\phi_{xx}[m]e^{-j\omega m} \tag{5.41}$$

The windowing operation that produces $\mathbb{E}[c_{xx}[m]]$ from $\phi_{xx}[m]$ is depicted graphically in the left column of Fig. 5.2. Since $\mathbb{E}[I_N(\omega)]$ is the Fourier transform of the product of the true autocorrelation function and the Bartlett window in time, it can be expressed in the frequency domain as the circular convolution of the true power spectral density function $P_{xx}(\omega)$ with $W_B(e^{j\omega})$, the Fourier transform of the Bartlett window:

$$\mathbb{E}[I_N(\omega)] = \frac{1}{2\pi}P_{xx}(\omega) \circledast W_B(e^{j\omega}) = \frac{1}{2\pi}\int_{-\pi}^{\pi} P_{xx}(\theta)W_B(e^{j(\omega-\theta)})d\theta \tag{5.42}$$

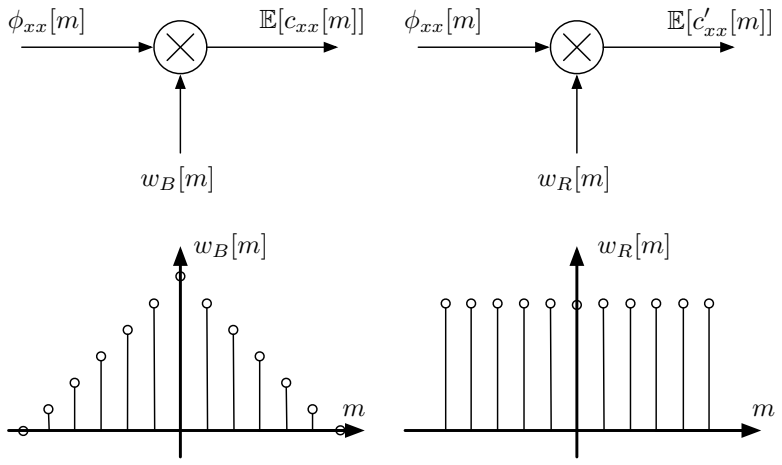


Figure 5.2: Examples of the formation of the expected values of $c_{xx}[m]$ (left column) and $c'_{xx}[m]$ (right column) by multiplying the true autocorrelation $\phi_{xx}[m]$ by $w_B[m]$ and $w_R[m]$ depicted below for $N = 6$ in each case, which implies a total window length of $2N-1 = 11$.

We had chosen to define the periodogram in terms of the biased autocorrelation function, $c_{xx}[m]$. Had we used instead the unbiased autocorrelation function, $c'_{xx}[m]$, the expression for the mean of the autocorrelation function would have been

$$\mathbb{E}[c'_{xx}[m]] = \mathbb{E}\left[\frac{1}{N-|m|}\sum_{n=0}^{N-1-|m|} x[n]x[n+m]\right] = \frac{1}{N-|m|}\sum_{n=0}^{N-1-|m|}\phi_{xx}[m] = w_R[m]\phi_{xx}[m] \tag{5.43}$$

where

$$w_R[m] = \begin{cases} 1, & |m| < N \\ 0, & \text{otherwise} \end{cases} \tag{5.44}$$

The expected value of the corresponding periodogram, which we label as $I'_N(\omega)$ for convenience would be

$$\mathbb{E}[I'_N(\omega)] = \frac{1}{2\pi}P_{xx}(\omega) \circledast W_R(e^{j\omega}) = \frac{1}{2\pi}\int_{-\pi}^{\pi} P_{xx}(\theta)W_R(e^{j(\omega-\theta)})d\theta \tag{5.45}$$

where as before $W_R(e^{j\omega})$ is the Fourier transform of $w_R[m]$. This operation is depicted in the upper right panel of Fig. 5.2.

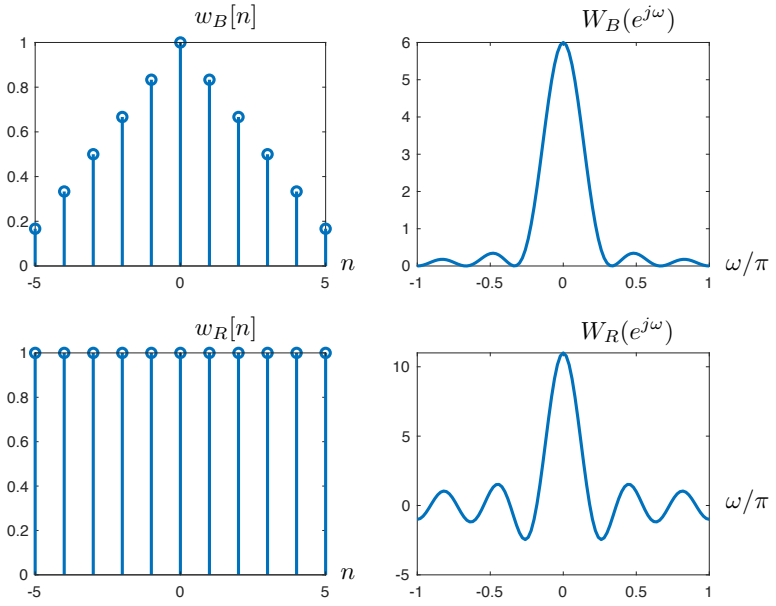


Figure 5.3: The rectangular and Bartlett window functions, $w_R[n]$ and $w_B[n]$, and their Fourier transforms, $W_R(e^{j\omega})$ and $W_B(e^{j\omega})$. Functions are plotted with a value of $N = 6$, producing a total window duration of 11, as in Fig. 5.2.

Using basic discrete-time Fourier transform definitions and properties, it is easy to show that the Fourier transforms for the rectangular and Bartlett (or triangular) windows $w_R[m]$ and $w_B[m]$ are

$$W_R(e^{j\omega}) = \frac{\sin(\omega(2N-1)/2)}{\sin(\omega/2))} \tag{5.46}$$

and

$$W_B(e^{j\omega}) = \frac{1}{N}\left(\frac{\sin(\omega N/2)}{\sin(\omega/2)}\right)^2 \tag{5.47}$$

The Fourier transforms $W_B(e^{j\omega})$ and $W_R(e^{j\omega})$ are plotted in Fig.5.3 for the value of $N = 6$ for a total window duration of 11. As N goes to infinity, both $W_B(e^{j\omega})$ and $W_R(e^{j\omega})$ converge to a delta function of area N, so both $I_N(\omega)$ and $I'_N(\omega)$ are asymptotically unbiased. As N becomes finite and smaller, the periodogram will become increasingly biased as $\mathbb{E}[I_N(\omega)]$ becomes "blurred" by the convolution of the main lobe of $W_B(e^{j\omega})$ or $W_R(e^{j\omega})$ with the true power spectrum $P_{xx}(\omega)$.

We note that the Fourier transform of the Bartlett window is strictly positive while the Fourier transform of the rectangular window is both positive and negative at different frequencies. This is important because it means that if the unbiased autocorrelation function $c'_{xx}[m]$ were used as the basis for an estimate of the power spectral density function, the power spectrum could have an expected value that is negative for some frequencies. This violates the constraint that a power spectral density function must be non-negative for physical reasons, which is why the biased autocorrelation function $c_{xx}[m]$ is preferred in estimating power spectral density functions using the periodogram. We elaborate on this issue further in Sec. 5.7.2.

5.6.2 The variance of the periodogram

The major problem with the periodogram is that its variance is not well behaved. In fact, the variance of the periodogram cannot be easily calculated except for the simplest of stationary random processes. While some analytical solutions to this problem have been proposed by Jenkins and Watts (1968), Oppenheim and Schafer (1975) describe an approach that provides approximate expressions for the variance obtained when using the periodogram to estimate the power spectrum of nonwhite Gaussian random processes. Their approach leads to the approximate expression

$$\mathbb{V}\mathrm{ar}[I_N(\omega)] \approx P_{xx}^2(\omega)\left\{1 + \left(\frac{\sin(\omega N)}{N\sin(\omega)}\right)^2\right\} \approx P_{xx}^2(\omega) \text{ for large } N \tag{5.48}$$

The most important thing about this result is that the approximate variance is proportional to the true power spectral density function and does *not* become smaller as N increases. This is in part a consequence of the fact that as $|m|$ approaches N, the values of the autocorrelation function are estimated with increasing unreliability because these coefficient estimates are based on a smaller amount of actual overlapping data points. Because the autocorrelation is converted to the periodogram through Fourier transformation, the unreliability of the estimates of the autocorrelation coefficients in the tails causes estimates of the values of *all* the coefficients in the power spectral density function to become unreliable. This fact is illustrated in Fig. 5.4, which depicts sample periodograms for a stationary random process that is Gaussian but not white. We note that when the sequence length $N = 256$, there is substantial spectral blurring and the dip in the spectrum at approximately $\omega = 0.4$ is filled in because of the convolution in frequency of the true PSD with the DTFT of the window function as described in Eq. (5.42).

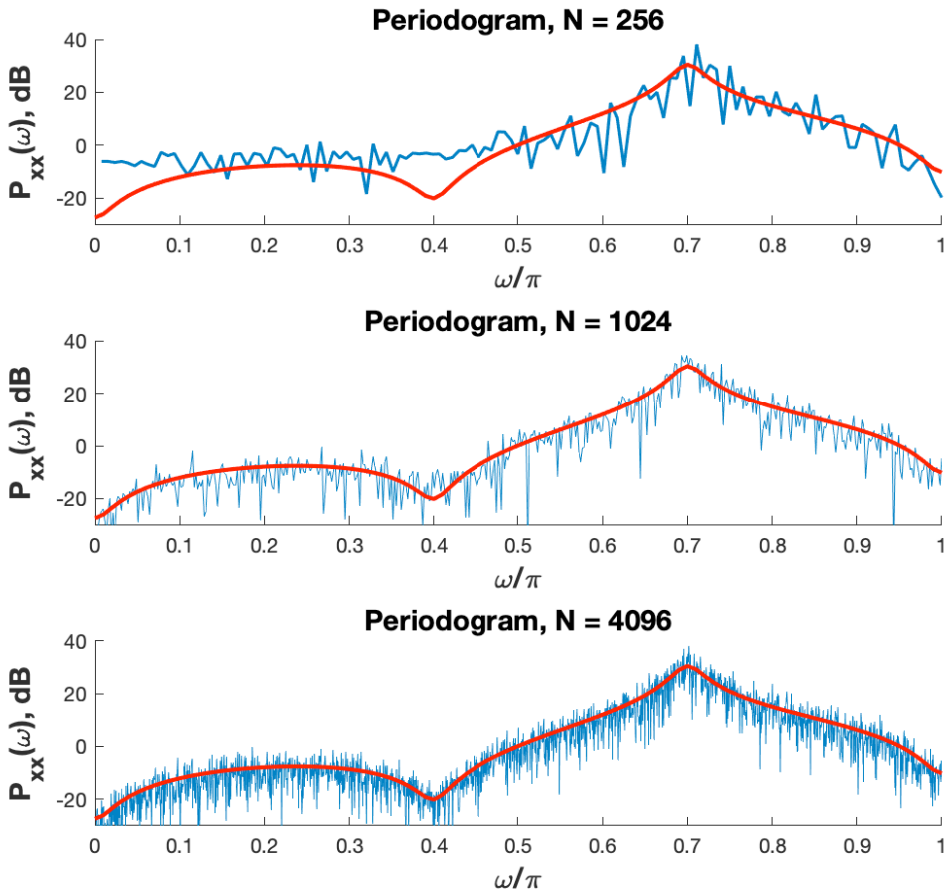


Figure 5.4: Example periodograms of a nonwhite Gaussian random process for segments of three different length. The red curve indicates the true power spectral density function.

The most significant thing that we can observe from Fig. 5.4, though, is that the variance of the periodogram does not diminish as the number of samples used to estimate it increases from 256 to 4096, which is consistent with the result of Eq. (5.48). In fact, the only way to reduce the variance of the periodogram is by averaging multiple estimates of it. We discuss two methods to accomplish this in Sec. 5.7 below.

5.7 Smoothed estimators of power spectral density

The key to reducing the variance of the periodogram as an estimate of power spectral density is to invoke statistical averaging in some fashion. The two major ways of doing this are averaging the periodograms of subsets of the observed samples over time and (implicitly) averaging values of the estimate in frequency. This section describes and discusses these approaches.

5.7.1 The Bartlett method

The most straightforward way to reduce variance is to compute periodograms over successive samples of the random process and average these periodograms over time. As

usual, let us assume that we have a total of N samples of data: we observe the random process $x[n]$ for $0 \le n \le N-1$. We begin by partitioning the data into K abutting sections of length M, so that $N = MK$:

$$x^{(i)}[n] = x[n + ((i-1)M)], \quad 0 \le n \le M-1, \quad 1 \le i \le K \tag{5.49}$$

Computing the periodograms for each subsequence we obtain

$$I_M^{(i)}(\omega) = \frac{1}{M}\left|\sum_{n=0}^{M-1} x^{(i)}[n]e^{-j\omega n}\right|^2 \tag{5.50}$$

The Bartlett spectral estimate is obtained by averaging the K individual periodograms:

$$B_{xx}(\omega) = \frac{1}{K}\sum_{i=1}^{K} I_M^{(i)}(\omega) \tag{5.51}$$

The expected value of the PSD estimate provided by Bartlett averaging is easy to obtain:

$$\mathbb{E}[B_{xx}(\omega)] = \frac{1}{K}\sum_{i=1}^{K}\mathbb{E}[I_M^{(i)}(\omega)] = \frac{1}{2\pi}P_{xx}(\omega) \circledast W_B(e^{j\omega}) \tag{5.52}$$

with the last equality following Eq. 5.42. In this case, $W_B(e^{j\omega})$ is the DTFT of a Bartlett window of duration M. We note that the expected value of the Bartlett average is always the convolution of the true PSD with the DTFT of the window function regardless of the number of sections averaged, K, but as K increases, the duration of each individual segment M will decrease, causing the width of the main lobe of $W_B(e^{j\omega})$ to become wider and the estimate $B_{xx}(\omega)$ to become more biased.

The variance of the estimate $B_{xx}(\omega)$ will depend in part on the extent to which the successive subsequences $x^{(i)}[n]$ are statistically independent. If M is sufficiently large that $\phi_{xx}[m]$ is relatively small for $|m|$ close to M, then successive subsequences of $x[n]$ will be approximately statistically independent. In that case, we can use the property that the variance of the sum is the sum of the individual variances, allowing us to write

$$\mathbb{V}\mathrm{ar}[B_{xx}(\omega)] \approx \frac{1}{K}\sum_{i=1}^{K}\mathbb{V}\mathrm{ar}[I_M^{(i)}(\omega)] \approx \frac{1}{K}P_{xx}^2(\omega) \tag{5.53}$$

with the last approximate equivalence following the discussion leading up to Eq. (5.48). In other words, the resulting variance is inversely proportional to the number of segments that are used for the averaging. The price that is paid, though, is that with a fixed amount of input data, the duration of each subsequence M decreases, which causes the bias to increase as the effective width of the mainlobe of $W_B(e^{j\omega})$ increases. Hence, there is a tradeoff between better variance and worse bias as the number of subsequences K is increased for a fixed segment length N of the observed random process.

These effects can be seen in Fig. 5.5, which shows the sample estimates of $P_{xx}(\omega)$ obtained by extracting 1, 4, and 16 abutting subsequences from 1024 samples of a Gaussian random process, and subsequently averaging the periodograms that are obtained. It can be seen that as the number of subsequences increases, the variance decreases but the bias increases.

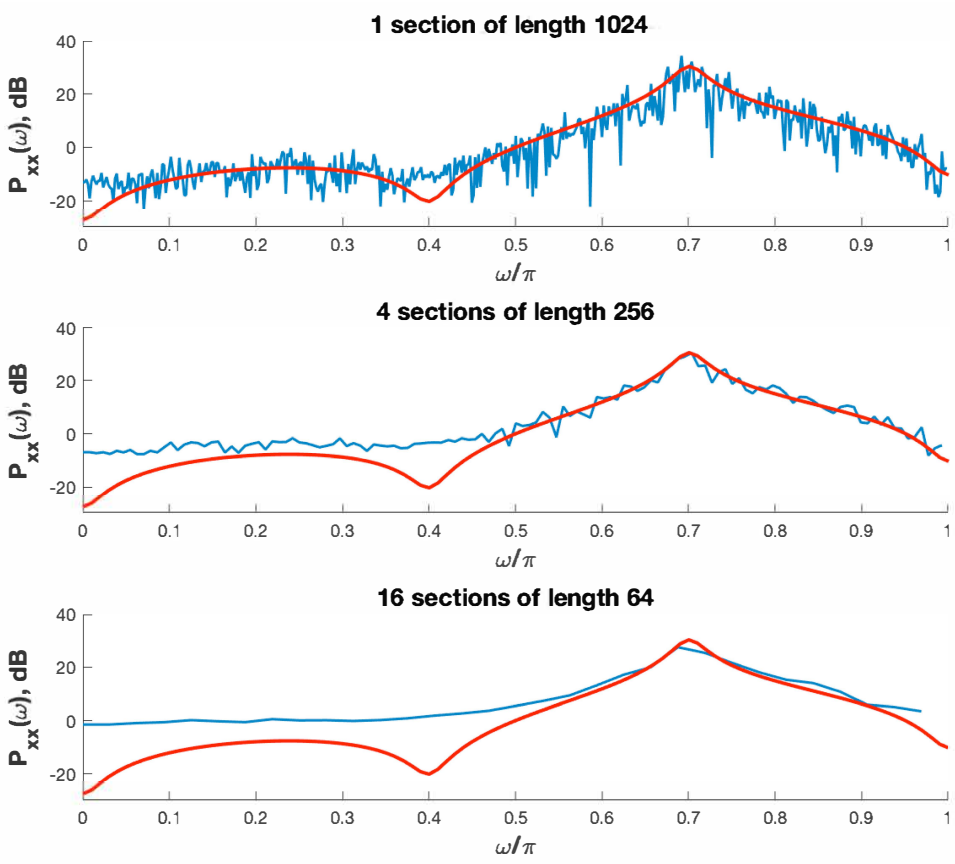


Figure 5.5: Effects of the number of segments on bias and variance using the Bartlett method. Estimates of power spectra are shown for 1024 samples of a stationary Gaussian random process, comparing results obtained by averaging 1,4, and 16 subsequences of the random process. The true power spectral density is shown in red.

5.7.2 Windowing the autocorrelation function: the Welch method

A second way of reducing the variance of the periodogram is by averaging it over frequency over a neighborhood of frequencies. For example, we can define the smoothed (or "smeared") periodogram as

$$S_{xx}(\omega) = \frac{1}{2\pi}\int_{-\pi}^{\pi} I_N(\theta) W_s(e^{j(\omega-\theta)}) d\theta \tag{5.54}$$

where $W_s(e^{j\omega})$ is a function that has a smearing pulse in the frequency domain, not unlike the functions $W_B(e^{j\omega})$ and $W_R(e^{j\omega})$ that we have already discussed. Let us define the generic window function $w[m]$ to be the inverse DTFT of $W_s(e^{j\omega})$:

$$w_s[m] = \frac{1}{2\pi}\int_{-\pi}^{\pi} W_s(e^{j\omega}) e^{j\omega m} d\omega \tag{5.55}$$

so that we obtain the Fourier transform pair

$$S_{xx}(\omega) = \sum_{m=-(M-1)}^{M-1} c_{xx}[m] w_s[m] e^{-j\omega m} \tag{5.56}$$

It is clear that it is necessary for $w[m]$ to be real and even in order for $S_{xx}(\omega)$ to be real and even, which is required for $S_{xx}(\omega)$ to be a valid power spectral density function. While we also know that $S_{xx}(\omega)$ must be non-negative, it is less obvious what constraints this places on the function $w_s[m]$. As we have discussed above, a sufficient but not necessary condition for $S_{xx}(\omega)$ to be nonnegative is for $W_s(e^{j\omega})$ itself to be nonnegative, since we know that $S_{xx}(\omega)$ is the convolution of $W_s(\omega)$ with the periodogram $I_M(\omega)$, which we know will be nonnegative.

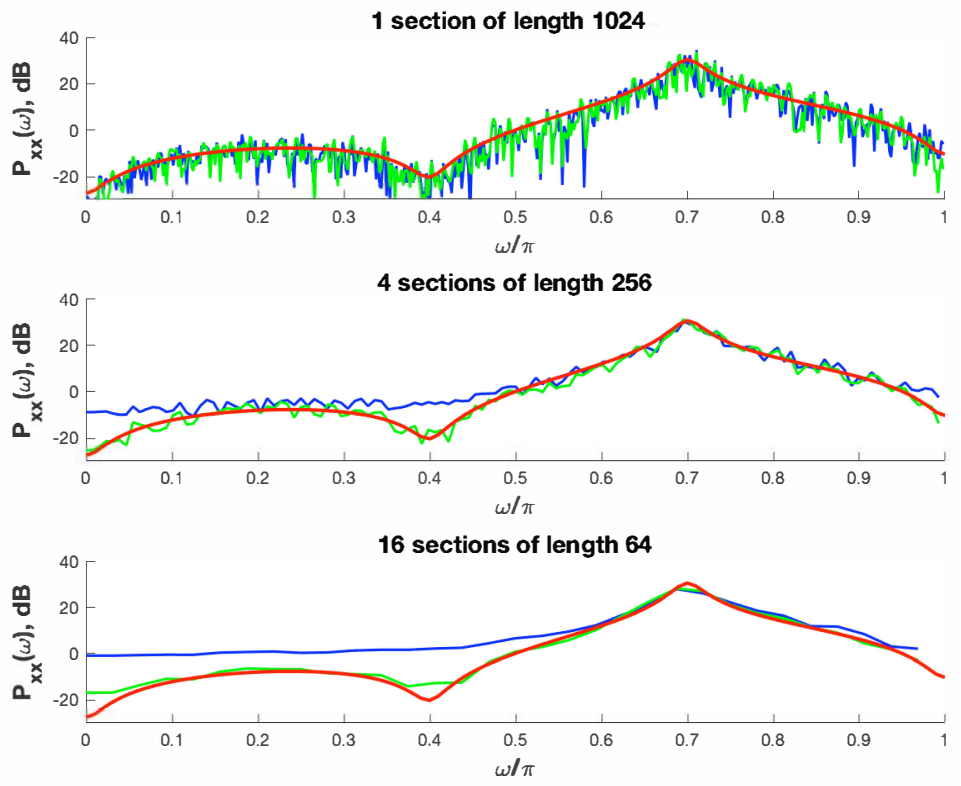


Figure 5.6: Effects of windowing the time function before calculating the power spectrum. Estimates of power spectra are shown for 1024 samples of a stationary Gaussian random process, comparing results obtained by averaging 1,4, and 16 subsequences of the random process. The true power spectrum is shown in red, the Bartlett estimate obtained by averaging periodograms is shown in blue, and the Welch estimate obtained by windowing the data before computing the averaged periodogram is shown in green.

One way of assuring that $W_s(e^{j\omega})$ is nonnegative is to require that $w_s[m]$ be *itself* an autocorrelation function, since the Fourier transform of an autocorrelation function will have the properties of a power spectral density function and hence be non-negative. In other words, we need to choose $w_s[m]$, the inverse DTFT of the smoothing function $W_s(e^{j\omega})$ to be of the form

$$w_s[m] = \sum_{k=-\infty}^{\infty} w[k]w[m+k] \tag{5.57}$$

for some finite-duration $w[m]$. This can be guaranteed if we apply the window $w[m]$ to the original *sample function* rather than to the autocorrelation function as is done implicitly

in the discussions leading up to Eqs. (5.41) and (5.45). Specifically, the *Welch method* proposes that smoothed power spectral density function estimates be obtained by averaging series of *modified periodograms*. Specifically, let

$$B_{xx}^{w}(\omega) = \frac{1}{K}\sum_{i=1}^{K} J_M^{(i)}(\omega) \tag{5.58}$$

where

$$J_M^{(i)}(\omega) = \frac{1}{MU}\left|\sum_{n=0}^{M-1} x^{(i)}[n]w[n]e^{-j\omega n}\right|^2 \quad \text{for } i = 1, 2, \ldots, K \tag{5.59}$$

and $x^{(i)}[n]$ was defined in Eq. (5.49). The normalizing constant U is needed for the smoothed estimate to be unbiased, and is equal to

$$U = \frac{1}{M}\sum_{n=0}^{M-1} w^2[n] \tag{5.60}$$

Note that if $w[n]$ is rectangular with

$$w[n] = \begin{cases} 1, & 0 \le n \le M-1 \\ 0, & \text{otherwise} \end{cases} \tag{5.61}$$

the resulting value of U will be equal to 1. The original Bartlett window used in Eq. (5.42) can be thought of as the autocorrelation of the window $w[n]$ in Eq. (5.61), so the normalization term U is already a part of Eq. (5.42) implicitly. Figure 5.6 shows the original estimated power spectra calculated according to the Bartlett method (in blue) to the same calculation obtained by imposing a Hamming window on the segments of the original data (in green), following the Welch method. It can be seen that the imposition of the window is helpful in reducing the bias, especially in the notch of the spectral estimate, without adversely affecting the variance of the estimate. As before, expressions for the variance of the smoothed estimate of the power spectrum, $S_{xx}(\omega)$, and using the Welch method, $B_{xx}^{w}(\omega)$, are difficult to develop without further assumptions and approximations. Welch (1970) noted that if the segments $x^{(i)}[n]$ are non-overlapping,

$$\mathbb{V}\text{ar}[B_{xx}^{w}] \approx \frac{1}{K}P_{xx}^2(\omega) \tag{5.62}$$

Figure 5.6 shows the effect of windowing the data explicitly before computing the modified periodograms following the Welch method [Eqs. (5.58) through (5.60)].

Further discussion on the variance of periodograms and smoothed periodograms may be found in Oppenheim and Schafer (1975).

In summary, we described and discussed in this chapter some of the most widely-used approaches to classical spectral estimation. All of these techniques are based on the periodogram, which can be thought of as either the Fourier transform of the sample autocorrelation function of a finite segment of observed data or as the squared magnitude of

the Fourier transform of the same data segment divided by the number of samples in the segment. It was shown that with reasonable computation procedures the expected value of the periodogram would asymptote to the true power spectral density function as the number of samples in the calculation increased to infinity. Unfortunately, the variance of the periodogram does not decrease, even when a large number of observed samples is used in the calculation. We showed that variance can be reduced either by dividing the observed data into sections and computing the average of the periodograms obtained for each section or by smoothing the periodogram in frequency, which is equivalent to multiplying the original data by a window function in time.

All of the methods described in this chapter work from either a finite segment of observed data or the autocorrelation function of those data, calculated over only a finite number of lags. In effect, this is equivalent to assuming that the values of the autocorrelation function beyond the limit of calculation are equal to zero. While this may be approximately true in some cases, it is certainly not true in many practical cases of interest. In the next chapter we begin a discussion about an entirely different way of thinking about estimating power spectra: we will make no assumptions at all about the tails of the autocorrelation function, but we will constrain the values of these unknown coefficients to create an estimated power spectral density function that is as "smooth" as possible.

6. Introduction to Maximum Entropy Spectral Estimation

In the previous chapter we developed estimates of the power spectral density function of a random process from a finite number of samples of its autocorrelation function, implicitly assuming that autocorrelation values outside the limited range are equal to zero. In this chapter we develop an alternate approach that makes no assumptions at all about the unknown autocorrelation values. The only constraint on the PSD estimate is that it must be as "smooth" as possible.

6.1 Introduction

In this chapter we discuss the derivation of the maximum entropy method (MEM) for estimation of the power spectral density function.[1]

In the classical spectral estimation techniques that we have discussed (including the Bartlett, Blackman-Tukey, and Welch methods) we have estimated $\phi_{xx}[m]$, the autocorrelation function of the random process $x[n]$ for $m \leq P$ and developed spectral estimates from the periodogram, which is the DTFT of the estimated autocorrelation function. These methods all make an implicit assumption that $\phi_{xx}[m] = 0$ for $|m| > P$. (The variable P is used in the MEM spectral estimation literature in the same way that N was used in discussing autocorrelation in Chapter 5.) In contrast, the MEM approach makes no assumption about the values of $\phi_{xx}[m] = 0$ for $|m| > P$, but seeks instead to obtain the estimated power spectrum that is as "smooth" as possible.

6.2 Information and Entropy

Implementation of the MEM method would require that we have a measure of "smoothness" of an estimated power spectral density function. The entropy H associated with a power spectral density function is proposed as the corresponding measure of smoothness.

[1] Much of the discussion in this chapter follows Sec. XV in the review/tutorial article by E. A. Robinson, E.A., "A historical perspective of spectrum estimation," Proceedings of the IEEE, vol.70, no.9, pp. 885-907 (1982).

The concept of entropy is encountered in a number of fields. For example, in thermodynamics, entropy is a measure of the amount of "disorder" of a system. Entropy is a central concept as well in aspects of statistical information theory and coding, as initially proposed by Claude Shannon in 1948. (Shannon received an honorary doctorate from CMU for these contributions, among many other things, in the 1980s.) In information theory, the information associated with learning that a discrete probabilistic event X_i is true is the log of the event's inverse probability:

$$I(X_i) = \log\left(\frac{1}{P[X_i]}\right) = -\log(P(X_i]) \tag{6.1}$$

In other words, the information associated with an event can be thought of as the amount of "surprise" engendered by learning that the event is true. If the log is calculated using base 2, the information associated with an event is measured in *bits*. Note that the information associated with an event of *a priori* probability 0.5 is equal to one bit, and that if we learn of the outcomes of two statistically independent events, the information associated with that joint event is equal to the sum of the information associated with the two events individually.

By definition, entropy is the expected value of the information over a collection of events:

$$H = \sum_{i=1}^{N} -\log(P(X_i])P[X_i] \tag{6.2}$$

Example 6.1 (Entropy associated with the roll of a die) As an example, consider the information associated with the outcomes of roll of a fair die (*i.e.* the singular form of "dice"), which has six surfaces, each landing on top with probability 1/6:

$$H = \sum_{i=1}^{6} -\log_2(P[X_i]P[X_i] = -\log_2(1/6)(1/6) = 2.585 \text{ bits}$$

Now consider in contrast the information associated with an unfair die for which $P[1] = 0.5$ and the probability of surfaces 2 through 5 is 0.1:

$$H = \sum_{i=1}^{N} -\log_2(P[X_i])P[X_i] = -\log_2(1/2)(1/2) + 5\log_2(1/10)(1/10) = 2.161 \text{ bits}$$

Note that the entropy associated with those probability densities is greater when the probabilities are uniformly distributed than when the probability is more concentrated in a single outcome. This is consistent with the idea that a PSD function will be "smoothest" when its entropy is maximized.

For a continuous random variable x the information associated with its probability density function is can be written as

$$H(p_x(\alpha)) = \int_{-\pi}^{\pi} -\log(p_x(\alpha))p_x(\alpha)d\alpha \tag{6.3}$$

Similarly, the entropy associated with a power spectral density function $P_{xx}(\omega)$ is

$$H = \frac{1}{2\pi}\int_{-\pi}^{\pi} \log(P_{xx}(\omega))d\omega \tag{6.4}$$

Note again that the greatest value of H for $P_{xx}(\omega)$ will occur when $P_{xx}(\omega)$ is uniform over frequency (*i.e.* "white"), while H will decrease as spectral density becomes more focussed in a narrow band of frequencies.

6.3 Spectral estimation by maximizing entropy

In general, we assume that we begin the MEM estimation problem by being given the first P lags of an autocorrelation function (or obtaining estimates of them),

$$\phi_{xx}[m], \text{ for } 0 \le |m| \le P \tag{6.5}$$

We then attempt to obtain the power spectral density function $\hat{P}_{xx}(\omega)$ that is "smoothest" subject to the constraint that the inverse DTFT of $\hat{P}_{xx}(\omega)$, $\hat{\phi}_{xx}[m]$, equals the true autocorrelation function, $\phi_{xx}[m]$ for $|m| \le P$. ($\hat{\phi}_{xx}[m]$ can have any value for $|m| > P$). As discussed above, this means that we wish to maximize the entropy of the estimate of the spectral density function $\hat{P}_{xx}(\omega)$,

$$H = \frac{1}{2\pi}\int_{-\pi}^{\pi} (\log[\hat{P}_{xx}(\omega)])d\omega \tag{6.6}$$

subject to the constraint that

$$\hat{\phi}_{xx}[m] = \frac{1}{2\pi}\int_{-\pi}^{\pi} \hat{P}_{xx}(\omega)e^{j\omega m}d\omega = \phi_{xx}[m], \text{ for } 0 \le |m| \le P \tag{6.7}$$

Although this problem can be solved in principle using Lagrange interpolation, we will obtain our result using more prosaic means. As usual,

$$\hat{P}_{xx}(\omega) = \sum_{m=-\infty}^{\infty} \hat{\phi}_{xx}[m]e^{-j\omega m} \tag{6.8}$$

We will try to find the values of $\hat{\phi}_{xx}[m]$ for $|m| > P$ that maximizes the expression for H above. We begin by considering the partial derivative

$$\frac{\partial \hat{P}_{xx}(\omega)}{\partial \hat{\phi}_{xx}[m]} = e^{-j\omega m} \tag{6.9}$$

which implies

$$\frac{\partial \log(\hat{P}_{xx}(\omega))}{\partial \hat{\phi}_{xx}[m]} = \frac{e^{-j\omega m}}{\hat{P}_{xx}(\omega)} \tag{6.10}$$

So, maximizing the entropy of the process requires that for $|m| > P$.

$$\frac{\partial H}{\partial \hat{\phi}_{xx}[m]} = \frac{\partial}{\partial \hat{\phi}_{xx}[m]}\frac{1}{2\pi}\int_{-\pi}^{\pi} \log(\hat{P}_{xx}(\omega))d\omega = 0 \tag{6.11}$$

Taking the derivative inside the integral we obtain

$$\frac{\partial H}{\partial \hat{\phi}_{xx}[m]} = \frac{1}{2\pi}\int_{-\pi}^{\pi} \frac{\partial \log(\hat{P}_{xx}(\omega))}{\partial \hat{\phi}_{xx}[m]} d\omega = \frac{1}{2\pi}\int_{-\pi}^{\pi} \frac{e^{-j\omega m}}{\hat{P}_{xx}(\omega)} d\omega = 0 \tag{6.12}$$

Hence we need to constrain

$$\frac{1}{2\pi}\int_{-\pi}^{\pi} (\hat{P}_{xx}(\omega))^{-1} e^{-j\omega m} d\omega = 0 \text{ for } |m| > P \text{ where } (\hat{P}_{xx}(\omega))^{-1} = \frac{1}{\hat{P}_{xx}(\omega)} \tag{6.13}$$

For notational convenience, let us define $\psi_{xx}[m]$ to be the inverse DTFT of $(\hat{P}_{xx}(\omega))^{-1}$, with

$$\psi_{xx}[m] = 0 \text{ for } |m| > P \tag{6.14}$$

Hence we can write $(\hat{P}_{xx}(\omega))^{-1}$ as

$$(\hat{P}_{xx}(\omega))^{-1} = \sum_{m=-P}^{P} \psi_{xx}[m] e^{-j\omega m} \tag{6.15}$$

or

$$\hat{P}_{xx}(\omega) = \frac{1}{\sum_{m=-P}^{P} \psi_{xx}[m] e^{-j\omega m}} \tag{6.16}$$

Generalizing from DTFTs to z-transforms, this corresponds to

$$\hat{P}_{xx}(z) = \frac{1}{\sum_{m=-P}^{P} \psi_{xx}[m] z^{-m}} \tag{6.17}$$

with

$$\hat{P}_{xx}(\omega) = \hat{P}_{xx}(z)\Big|_{z=e^{j\omega}} \tag{6.18}$$

Because $\hat{P}_{xx}(\omega)$ is a real and even function of frequency, its reciprocal, $1/\hat{P}_{xx}(\omega)$, is a real and even function of frequency, so $\psi_{xx}[m]$, the inverse transform of the reciprocal, will be a real and even function of m. This means that the poles of $\hat{P}_{xx}(z)$ will occur at reciprocal locations in the z-plane, so the existence of a pole at location $z = z_0$ implies that there will also be a pole at z_0^{-1}. Mathematically, this can be expressed as

$$\hat{P}_{xx}(\omega) = \frac{\sigma_x^2}{A(z)A(z^{-1})} = \frac{\sigma_x^2}{(1 - \alpha_1 z^{-1} - \alpha_2 z^{-2} \cdots - \alpha_P z^{-P})(1 - \alpha_1 z - \alpha_2 z^2 \cdots - \alpha_P z^P)} \tag{6.19}$$

From the point of view of power spectral density, this suggests that the random process $x[n]$ can be modeled as the output of a linear filter that has a white noise process $w[n]$ as its input.

Specifically, if $w[n]$ has a power spectral density function (and variance) equal to σ_w^2, the power spectral density of $x[n]$ is equal to

$$\hat{P}_{xx}(\omega) = H(z)H(z^{-1})\Big|_{z=e^{j\omega}} \tag{6.20}$$

where in this case, the z-transform of $h[n]$ is of the form

$$H(z) = \frac{G}{1 - \sum_{k=1}^{P} \alpha_k z^{-k}} \tag{6.21}$$

In other words, the maximum entropy solution states that the power spectrum $P_{xx}(\omega)$ that matches the autocorrelation function $\phi_{xx}[m]$ for $|m| \leq P$ and at the same time is as "smooth" as possible can always be characterized as the result of passing a white-noise process through an *all-pole filter*. Of course, we have not yet specified exactly what that power spectrum actually is. We will address the solution to this problem in detail in Chapter 7 below.

6.4 The all-pole model of a spectral estimate

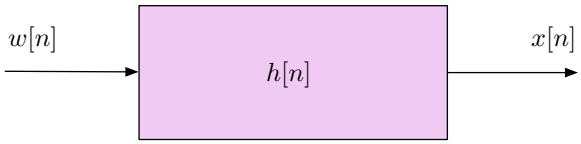


Figure 6.1: Parametric power spectral density function estimation by modeling. The observed random process $x[n]$ is assumed to be generated by passing a white random process $w[n]$ through an LSI filter of a known type.

In the last section we noted that the maximum entropy solution to power spectral density estimation can be characterized by passing a white-noise process $w[n]$ through an all-pole LSI filter, as illustrated in Fig. 6.1. This is an example of a type of *parametric spectral estimation*, and the LSI filter is sometimes referred to as a *model* of that power spectrum. We then attempt to estimate the parameters that characterize the filter model that provides the best parametric approximation to the unknown power spectrum. (In contrast, in the methods of power spectral density estimation discussed in Chapter 5 we worked from a general estimate of the entire autocorrelation function and then applied smoothing of some sort to reduce the variance of the estimated PSD function.)

In general, there are three types of parametric models that can be considered. The *moving average* (MA) model has zeros but not poles:

$$H(z) = B(z) = \sum_{l=0}^{M} b_l z^{-l} \tag{6.22}$$

The name of the MA model derives from the fact that the output is indeed a weighted linear combination of a moving average of input data points. In DSP, of course, this type of model is simply referred to as a finite-impulse response (FIR) filter.

The *autoregressive* (AR) model has poles but no zeros:

$$H(z) = \frac{G}{1 - \sum_{k=1}^{N} a_k z^{-k}} \tag{6.23}$$

As we have noted, this is the type of model that we found to be the solution to the MEM formulation of the PSD estimation problem.

The third type of model has both poles and zeros and is called (unsurprisingly) the *autoregressive moving-average* (ARMA) model:

$$H(z) = \frac{B(z)}{A(z)} = \frac{\sum_{l=0}^{M} b_l z^{-l}}{1 - \sum_{k=1}^{N} a_k z^{-k}} \tag{6.24}$$

Of the three models above, the autoregressive (all-pole) model is the most widely used in part because the equations are most easily solved. While we will consider solutions for the all-pole model in detail in the following chapter on linear prediction, it is worthwhile to develop here the equations that will form the basis for the solutions.

The equations above specify

$$H(z) = \frac{G}{1 - \sum_{k=1}^{N} a_k z^{-k}} = \frac{X(z)}{W(z)} \tag{6.25}$$

Cross-multiplying and taking the inverse z-transform, and now using P to denote the number of poles produces

$$x[n] - \sum_{k=1}^{P} \alpha_k x[n-k] = Gw[n] \tag{6.26}$$

or

$$x[n] = \sum_{k=1}^{P} \alpha_k x[n-k] + Gw[n] \tag{6.27}$$

The white noise process is wide-sense stationary by definition, and hence the observed process $x[n]$ is as well. Its autocorrelation function is

$$\phi_{xx}[m] = \mathbb{E}[x[n]x[n+m]] = \mathbb{E}\left[x[n]\left(\sum_{k=1}^{P} \alpha_k x[n+m-k] + Gw[n+m]\right)\right] \tag{6.28}$$

$$= \mathbb{E}\left[\sum_{k=1}^{P} \alpha_k x[n]x[n+m-k]\right] + G\mathbb{E}[x[n]w[n+m]] \tag{6.29}$$

Taking the expectation inside the sum produces

$$\phi_{xx}[m] = \sum_{k=1}^{P} \alpha_k \phi_{xx}[k] + \sigma_w^2 \text{ for } m = 0 \tag{6.30}$$

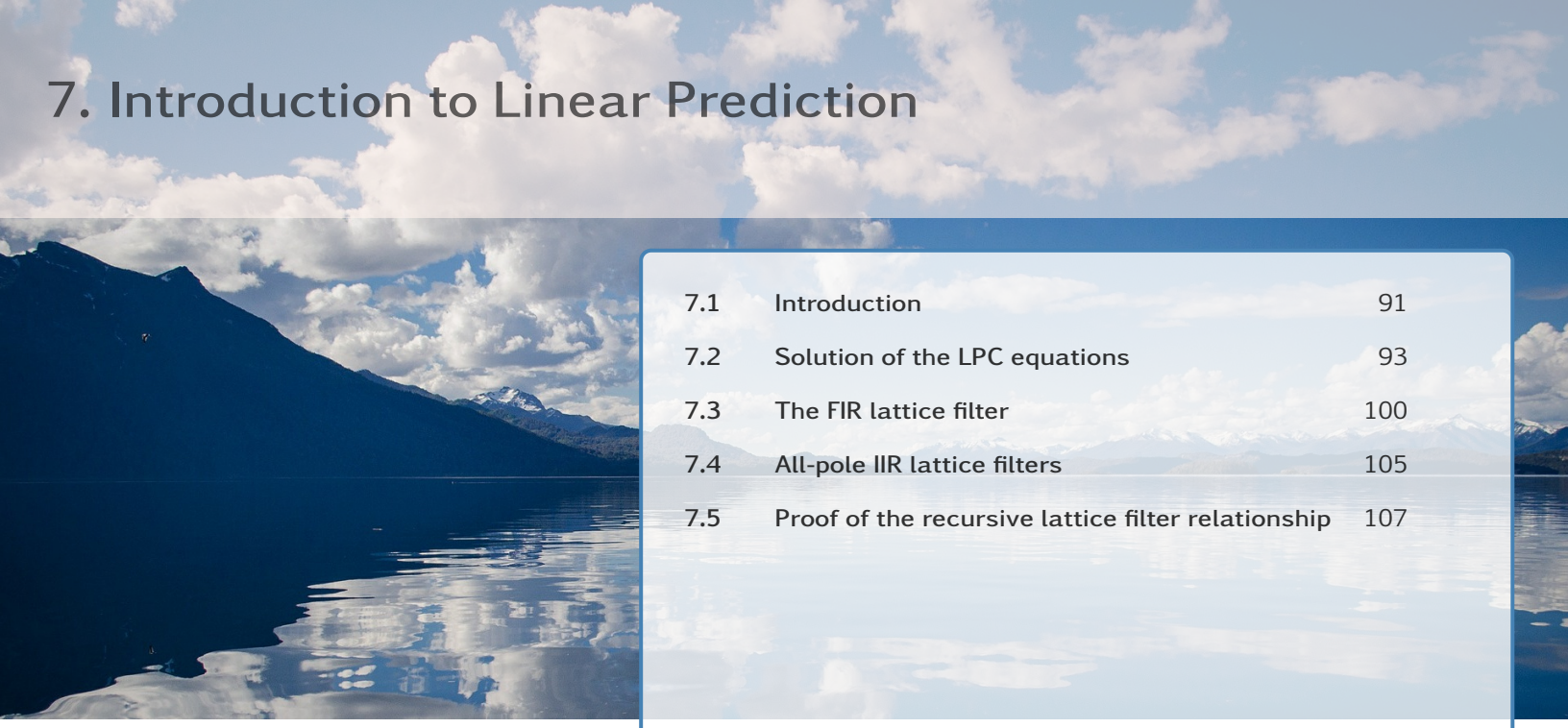
and

$$\phi_{xx}[m] = \sum_{k=1}^{P} \alpha_k \phi_{xx}[|m-k|] \text{ for } m \geq 1 \tag{6.31}$$

(The absolute value in the argument is used in Eq. (6.31) because the autocorrelation function is even for real random processes. The derivation of the equations above also assumes that the filter $h[n]$ is causal.)

Equations (6.30) and (6.31), referred to as the *Yule-Walker equations*, are the basis for many solutions to problems involving all-pole modeling. We will discuss some solutions to these equations in our discussion of linear prediction in the following chapter.

7. Introduction to Linear Prediction



Linear prediction is one of the most widely used techniques to model speech signals, and the underlying mathematics are the basis for a number of other foundational signal processing techniques as well. We also introduce the lattice filter in this chapter, which has many interesting properties.

7.1 Introduction

In this chapter we discuss most of the key concepts of linear prediction and lattice filters. As we noted in Chapter 6, the linear prediction approach is one of several ways to accomplish *filter design by modeling*, where we try to come up with a parametric representation that most closely matches the power spectral density function of an unknown random process.[1]

Specifically, we discussed three times of parametric models for PSD functions at the end of Chapter 6: the moving average (MA) model which has zeros only, the autoregressive (AR) model which has poles only, and the autoregressive-moving average (ARMA) model which has both poles and zeros Of the three types of parametric models, the all-pole AR model is the most commonly used, largely because the design equations used to obtain the best-fit AR model are simpler than those used for MA or ARMA modelling. Serendipitously, the all-pole model also has the ability to describe most types of speech sounds quite well, and for both of these reasons the all-pole model has become widely used in speech processing.

In these notes, we will begin with some general comments about linear prediction, which drives us to consider an all-pole model of the power spectrum of a one-dimensional signal. We will then consider in varying degrees of depth the three major ways in obtaining the parameters of the model, the autocorrelation method, the covariance method and

[1] The material in this chapter is based on the discussion of LPC in the text *Digital Processing of Speech Signals* (1978) by L. R. Rabiner and R. W. Schafer, which was updated in 2010 with the title *Digital Speech Processing*.

the partial correlation (PARCOR) method. We will conclude with a short introduction to lattice filters.

7.1.1 Linear prediction of the current sample of a random process

In our initial consideration of linear prediction, let us imagine that we are observing a random process $x[n]$. We would like to determine how to obtain a "best" prediction of the current sample of $x[n]$ from the previous P samples of the random process. Specifically, let the predicted value $\hat{x}[n]$ be defined by

$$\hat{x}[n] = \sum_{k=1}^{P} \alpha_k x[n-k] \tag{7.1}$$

We can define the error of the approximation to be

$$e[n] = x[n] - \hat{x}[n] = x[n] - \sum_{k=1}^{P} \alpha_k x[n-k] \tag{7.2}$$

Later we will consider in some detail the z-transform of the error function:

$$E(z) = X(z)\left(1 - \sum_{k=1}^{P} \alpha_k z^{-k}\right) \equiv X(z)A(z) \tag{7.3}$$

It is convenient to define the expected value of the square of the error function:

$$\xi^2 = \mathbb{E}[e^2[n]] = \mathbb{E}[(x[n] - \hat{x}[n])^2] = \mathbb{E}\left[\left(x[n] - \sum_{k=1}^{P} \alpha_k x[n-k]\right)^2\right] \tag{7.4}$$

Our immediate goal is to determine the set of $\{\alpha_k\}$ that minimize ξ^2. For a particular coefficient α_i this is typically accomplished by differentiating ξ^2 with respect to α_i, setting the derivative equal to zero, and solving for α_i,

$$\frac{\partial \xi^2}{\partial \alpha_i} = \mathbb{E}\left[2\left(x[n] - \sum_{k=1}^{P} \alpha_k x[n-k]\right)x[n-i]\right] = 0 \tag{7.5}$$

or

$$\mathbb{E}[x[n]x[n-i]] = \sum_{k=1}^{P} \hat{\alpha}_k \mathbb{E}[x[n-i]x[n-k]] \tag{7.6}$$

where $1 \le i \le P$ and $\hat{\alpha}_i$ represents the estimated value of α_i. Defining

$$\phi[i,k] = \mathbb{E}[x[n-i]x[n-k]] \tag{7.7}$$

we obtain

$$\sum_{k=1}^{P} \alpha_k \phi[i,k] = \phi[i,0] \tag{7.8}$$

It can also be shown that

$$\xi^2 = \phi[0,0] - \sum_{k=1}^{P} \alpha_k \phi[0,k] \tag{7.9}$$

You may recognize Eqs. (7.8) and (7.9) as being a form of the Yule-Walker equations, which you have already been exposed to (using a slightly different set of notational conventions) in the context of our discussion of MEM spectral estimation. We will solve these equations making use of two specific assumptions about the definition of $\phi[i,k]$ in Sec. 7.2.

7.1.2 How linear prediction relates to the all-pole (autoregressive) model

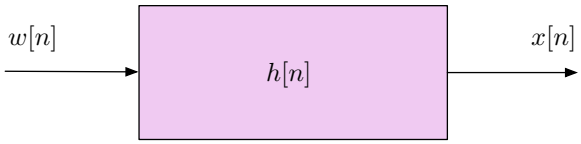


Figure 7.1: Random process modeled by passing white noise through a linear filter.

Although the discussion of the previous section was framed strictly in the context of determining the coefficients $\{\alpha_k\}$ that produce the "best" linear prediction of the current sample of $x[n]$ from the previous P samples, it is also often useful to consider $x[n]$ to be the output of an all-pole filter as in Fig. 7.1 above, which is reprinted from the previous chapter. Assume that the unit sample response of the filter $h[n]$ has a z-transform of the form

$$H(z) = \frac{X(z)}{W(z)} = \frac{G}{1 - \sum_{k=1}^{P} \alpha_k z^{-k}} \equiv \frac{G}{A(z)} \tag{7.10}$$

Note that the notational is somewhat unusual, as $x[n]$ represents the system output (rather than its input). In spectral estimation and system identification problems, the input function is typically assumed to be white noise, and the squared magnitude of the frequency response of the filter is a parametric estimate of the power spectral density function of the output random process. In speech processing, the input $w[n]$ represents the excitation of the vocal tract, which could be a quasi-periodic function corresponding to glottal pulses for voiced speech segments, and a broadband noise source for unvoiced speech segments.

By taking the inverse z-transform of both sides of Eq. (7.10) we obtain the expression

$$e[n] = x[n] - \hat{x}[n] = x[n] - \sum_{k=1}^{P} \alpha_k x[n-k] = Gw[n] \tag{7.11}$$

It is typically assumed that the predictor coefficients $\{\alpha_k\}$ that minimize the average square of $e[n]$ provide a good model of the vocal tract configuration that is used to produce a particular segment of speech. The gain parameter G is used in synthesizing an output waveform from the input excitation that has a power that best matches that of the observed output.

7.2 Solution of the LPC equations

7.2.1 General solution of the LPC equation

We will first consider the solution of the LPC equations for the general expression of $\phi[i,k]$ as defined above, and then consider the two special cases that lead to the so-called "autocorrelation" and "covariance" solutions.

Let us assume for the sake of example that $P = 4$. The system of equations in Eq. (7.8) can be written for $1 \leq k \leq P$ as

$$\begin{aligned}
\alpha_1\phi[1,1] + \alpha_2\phi[1,2] + \alpha_3\phi[1,3] + \alpha_4\phi[1,4] &= \phi[1,0] \\
\alpha_1\phi[2,1] + \alpha_2\phi[2,2] + \alpha_3\phi[2,3] + \alpha_4\phi[2,4] &= \phi[2,0] \\
\alpha_1\phi[3,1] + \alpha_2\phi[3,2] + \alpha_3\phi[3,3] + \alpha_4\phi[3,4] &= \phi[3,0] \\
\alpha_1\phi[4,1] + \alpha_2\phi[4,2] + \alpha_3\phi[4,3] + \alpha_4\phi[4,4] &= \phi[4,0]
\end{aligned} \tag{7.12}$$

These equations can be written in matrix-vector form as

$$\begin{bmatrix} \phi[1,1] & \phi[1,2] & \phi[1,3] & \phi[1,4] \\ \phi[2,1] & \phi[2,2] & \phi[2,3] & \phi[2,4] \\ \phi[3,1] & \phi[3,2] & \phi[3,3] & \phi[3,4] \\ \phi[4,1] & \phi[4,2] & \phi[4,3] & \phi[4,4] \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{bmatrix} = \begin{bmatrix} \phi[1,0] \\ \phi[2,0] \\ \phi[3,0] \\ \phi[4,0] \end{bmatrix} \tag{7.13}$$

Note that Eq. (7.13) is of the form

$$\mathbf{R}\alpha = \mathbf{P} \tag{7.14}$$

where $\mathbf{R}$ is a $P \times P$ matrix of autocorrelation coefficients, α is a $P \times 1$ vector of elements $\{\alpha_k\}$ and $\mathbf{P}$ is a $P \times 1$ vector of autocorrelation coefficients. This equation is known as the *Wiener-Hopf* equation, which is encountered frequently in optimal signal processing.

In general, a direct solution to the Wiener Hopf equation can be obtained by pre-multiplying both sides of Eq. (7.14) by the inverse of $\mathbf{R}$:

$$\alpha = \mathbf{R}^{-1}\mathbf{P} \tag{7.15}$$

The inversion of the $\mathbf{R}$ matrix can be accomplished by Gaussian elimination and other similar techniques, which are $O(P^3)$ in computational complexity. In the next section we discuss a particular formulation of the LPC problem that produces a solution that is far more computationally efficient.

7.2.2 The autocorrelation method and Levinson-Durbin recursion

As you may have noted, we still have not specified how the autocorrelation coefficients $\phi[i,k]$ are obtained from the original waveform $x[n]$. One reasonable approach is to obtain the correlation coefficients by first extracting a finite-duration segment of $x[n]$ by multiplying by a window function and then computing the first P autocorrelation coefficients of the finite length segment, as illustrated in the upper panel of Fig. 7.2. In other words, we assume that

$$x[n] = x[n]w[n] \tag{7.16}$$

where $w[n]$ is nonzero only for $0 \leq n \leq N-1$. In this case we can define the autocorrelation function as

$$\phi[i,k] = \sum_{m=\max(i,k)}^{N-1+\min(i,k)} x[m-i]x[m-k] = \sum_{m=-i+\max(i,k)}^{N-1-i+\min(i,k)} x[m]x[m+i-k] \tag{7.17}$$

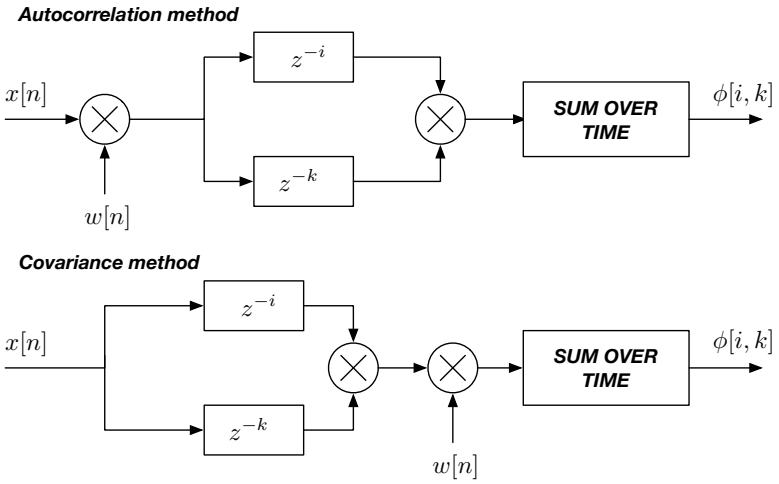


Figure 7.2: Comparison of the autocorrelation and covariance methods of obtaining autocorrelation coefficients for use in linear prediction. The window $w[n]$ is of finite duration.

for $1 \le i \le P$ and $1 \le k \le P$. Note that despite the inelegant notation, there will always be $N - |i-k|$ non-zero terms in the sum. It can easily be seen that Eq. (7.17) represents the short-term autocorrelation function of a segment of $x[n]$ and that

$$\phi[i,k] = \phi[|i-k|] \tag{7.18}$$

Because the number of nonzero elements of the argument of the summation decreases as $|i-k|$ increases, it is common to use a tapered window such as the Hamming window for $w[n]$.

Using this definition of the autocorrelation function and again dropping the subscripts n, Eq. (7.13), the Wiener-Hopf equation, reduces to

$$\begin{bmatrix} \phi[0] & \phi[1] & \phi[2] & \phi[3] \\ \phi[1] & \phi[0] & \phi[1] & \phi[2] \\ \phi[2] & \phi[1] & \phi[0] & \phi[1] \\ \phi[3] & \phi[2] & \phi[1] & \phi[0] \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{bmatrix} = \begin{bmatrix} \phi[1] \\ \phi[2] \\ \phi[3] \\ \phi[4] \end{bmatrix} \tag{7.19}$$

This method of solution is known as the *autocorrelation solution* of the LPC equations. Because the resulting correlation matrix $\mathbf{R}$ is *Toeplitz* (i.e. the elements of each diagonal of the matrix, major and minor, are identical), a simpler solution known as Levinson-Durbin recursion is possible. As we will see in a moment, Levinson-Durbin recursion is $O(P^2)$ in complexity.

The equations of the Levinson-Durbin recursion, which are used to compute the corre-

sponding reflection coefficients and LPC parameters are

$$E^{(0)} = \phi[0] \tag{7.20}$$

$$k_i = \frac{\left\{\phi[i] - \sum_{j=1}^{i-1} \alpha_j^{(i-1)} \phi[i-j]\right\}}{E^{(i-1)}}, \text{ calculated for } 1 \le i \le P \tag{7.21}$$

$$\alpha_i^{(i)} = k_i \tag{7.22}$$

$$\alpha_j^{(i)} = \alpha_j^{(i-1)} - k_i \alpha_{i-j}^{(i-1)}, \text{ for } 1 \le j \le i-1 \tag{7.23}$$

$$E^{(i)} = (1 - k_i^2) E^{(i-1)} \tag{7.24}$$

Equations (7.21) through (7.24) are solved recursively for $i = 1, 2, ..., P$ and the final solution is given by

$$\alpha_j = \alpha_j^{(P)} \text{ for } 1 \le j \le P \tag{7.25}$$

The coefficients $\{k_i\}$ for $1 \le i \le P$ are referred to as the *reflection coefficients*. They constitute an alternate specification of the random process $x[n]$ that is as unique and complete as the LPC predictor coefficients $\{\alpha_k^{(P)}\}$. The reflection coefficients are actually far more robust to coefficient quantization than the predictor coefficients, so they are frequently the representation of choice in applications such as speech coding or speech compression.

If the magnitude of the reflection coefficients $|k_i|$ is less than 1 for $1 \le i \le P$, all of the roots of the polynomial $A(z) = 1 - \sum_{k=1}^{P} \alpha_k^{(P)} z^{-k}$ will lie inside the unit circle. This means that if $|k_i| < 1$, the resulting filter $H(z)$ will be stable. It can be shown that deriving the $\{k_i\}$ in the fashion described above using Levinson-Durbin recursion guarantees that $|k_i| < 1$. We will make extensive use of the reflection coefficients $\{k_i\}$ in our discussion of lattice filters.

7.2.3 The covariance solution of the LPC equations

An alternate way of deriving the autocorrelation functions used to solve the LPC equations would be to always compute all lags of the autocorrelation function over the same time indices, as illustrated in the lower panel of Fig. 7.2. In a sense, this means that we are correlating and then windowing, while with the autocorrelation method we window and then correlate.

Under these circumstances the autocorrelation computation becomes

$$\phi[i,k] = \sum_{m=0}^{N-1} x[m-i]x[m-k] = \sum_{m=-i}^{N-1-i} x[m]x[m+i-k] \tag{7.26}$$

which as before is evaluated for $1 \le i \le P$ and $0 \le k \le P$. Although this computation is similar to that of Eq. (7.17), it is not identical to it because the limits of the summations are different. Specifically, it is very important to note that

$$\phi[i,k] = \phi[k,i] \ne \phi[|i-k|] \tag{7.27}$$

Because the number of samples used in the autocorrelation computation is always N regardless of the magnitude of the correlation lag, it is not considered necessary to multiply the argument of the summation by a tapered window before summing.

Under these circumstances, Eq. (7.13), the Wiener-Hopf equation reduces to

$$\begin{bmatrix} \phi[1,1] & \phi[1,2] & \phi[1,3] & \phi[1,4] \\ \phi[2,1] & \phi[2,2] & \phi[2,3] & \phi[2,4] \\ \phi[3,1] & \phi[3,2] & \phi[3,3] & \phi[3,4] \\ \phi[4,1] & \phi[4,2] & \phi[4,3] & \phi[4,4] \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{bmatrix} = \begin{bmatrix} \phi[1,0] \\ \phi[2,0] \\ \phi[3,0] \\ \phi[4,0] \end{bmatrix} \tag{7.28}$$

This solution to the LPC equations is called the *covariance solution* because the autocorrelation matrix on the left has the Hermitian symmetry that is characteristic of an arbitrary covariance matrix.

We note that the Levinson-Durbin recursion cannot be used to solve this equation because the autocorrelation matrix is Hermitian symmetric but not Toeplitz. This equation is typically solved using the Cholesky decomposition method which was discussed superficially in class and is treated in more detail in RS.

In general, the covariance solution provides a linear prediction of the current sample of a random process with somewhat less mean-squared prediction error than the autocorrelation solution. Nevertheless, the autocorrelation solution is far more commonly used in applications such as speech processing because it is so much more computationally efficient.

The pole locations will be inside the unit circle provided that the reflection coefficients $\{k_i\}$ are all less than one in magnitude, but this is not guaranteed when the covariance method is used. In practice, if it is observed that at least one reflection coefficient is greater than one in magnitude, it is quite straightforward to calculate the actual pole locations that correspond to that particular analysis segment, and then reflect the poles that lie outside the unit circle to mirror-image locations inside the unit circle.

7.2.4 Recursive relationships between the LPC coefficients and reflection coefficients

In Secs. 7.2.2 and 7.2.3 above, we discussed how the LPC coefficients can be obtained from the autocorrelation coefficients of an observed random process. We also noted in that section that the reflection coefficients k_i completely specify the LPC characterization of a random process just as the LPC coefficients α_i do. In fact, given either set of coefficients, we can always obtain the other by a simple linear recursion. Specifically, to convert from the reflection coefficients k_i to the LPC coefficients α_i, we use the recursion

$$\begin{aligned} &\text{Let } \alpha_i^{(i)} = k_i \text{ starting at } i = 1 \\ &\qquad \alpha_l^{(i)} = \alpha_l^{(i-1)} - k_i \alpha_{i-l}^{(i-1)} \text{ for } 1 \le l \le i-1 \\ &\text{Repeat for } i = 1, 2, \ldots, P \end{aligned} \tag{7.29}$$

Similarly, we can convert from the LPC coefficients α_i to the reflection coefficients k_i if we have all of the $\alpha_l^{(i)}$ for $i = 1, 2, ..., P$:

$$\begin{aligned} &\text{Let } k_P = \alpha_P^{(P)} \\ &\text{Starting with } i = P, \text{ let } k_i = \alpha_i^{(i)} \\ &\alpha_l^{(i-1)} = \frac{\alpha_l^{(i)} + k_i \alpha_{i-l}^{(i)}}{1 - k_i^2} \text{ for } i = P, P-1, ..., 2, 1 \text{ and } l = i, i-1, \ldots, 2, 1 \end{aligned} \tag{7.30}$$

7.2.5 Computation of the LPC gain parameter

The LPC gain parameter G can be computed in either of two ways. From the zeroth-order Yule-Walker equation,

$$\phi[0,0] = \sum_{k=1}^{P} \alpha_k \phi[0,k] + \sigma_x^2 = \sum_{k=1}^{P} \alpha_k \phi[0,k] + G^2 \tag{7.31}$$

we easily obtain

$$G^2 = \phi[0,0] - \sum_{k=1}^{P} \alpha_k \phi[0,k] \tag{7.32}$$

More prosaically, we can also estimate the gain parameter if both the input and output of the filter are available. From Eqs. (7.2) and (7.3) we note that

$$e[n] = x[n] - \hat{x}[n] = x[n] - \sum_{k=1}^{P} \alpha_k x[n-k] \tag{7.33}$$

The z-transform of $e[n]$ is

$$E(z) = X(z) - \sum_{k=1}^{P} \alpha_k X(z) z^{-k} = X(z)\left(1 - \sum_{k=1}^{P} \alpha_k z^{-k}\right) = X(z)A(z) \tag{7.34}$$

Combining the equation above with the defining relationship $X(z) = W(z)H(z) = W(z)G/A(z)$ we obtain

$$A(z) = \frac{G}{H(z)} = \frac{GW(z)}{X(z)} = \frac{E(z)}{X(z)} \tag{7.35}$$

implying, of course, that

$$G = \frac{E(z)}{W(z)} \text{ or } e[n] = Gw[n] \tag{7.36}$$

Since both $e[n]$ and $w[n]$ are typically time-varying stochastic signals, we typically estimate G from the short-term ratio of their energies:

$$G^2 = \frac{\sum_{m=0}^{N-1} e^2[m]}{\sum_{m=0}^{N-1} w^2[m]} \tag{7.37}$$

7.2.6 The LPC error in time and frequency

Consider once again the all-pole filter representation discussed in Sec. 7.2, with the all-pole transfer function

$$H(z) = \frac{X(z)}{W(z)} = \frac{G}{1 - \sum_{k=1}^{P} \alpha_k z^{-k}} \equiv \frac{G}{A(z)} \tag{7.38}$$

As noted above in Eqs. (7.33) and (7.36),

$$e[n] = x[n] - \hat{x}[n] = x[n] - \sum_{k=1}^{P} \alpha_k x[n-k] = Gw[n] \tag{7.39}$$

These relations also imply that

$$E(z) = X(z)A(z) \tag{7.40}$$

as we will note below in our discussion of lattice filters (with slightly different notation). The error signal $e[n]$ is "peakier" than the original time function $x[n]$ and its spectrum $E(e^{j\omega})$ is flatter than that of $X(e^{j\omega})$. Because of this peakiness, the error signal is frequently used as the basis for estimating the instantaneous fundamental frequency of a signal.

It is also useful to consider the error signal in the frequency domain in more detail. If we are using the autocorrelation method of the LPC solution, we can write by applying Parseval's theorem

$$\text{Energy} = \sum_{m=0}^{N-1} e^2[m] = \frac{1}{2\pi} \int_{-\pi}^{\pi} |E(e^{j\omega})|^2 d\omega = \frac{1}{2\pi} \int_{-\pi}^{\pi} |X(e^{j\omega})|^2 |A(e^{j\omega})|^2 d\omega \tag{7.41}$$

where

$$A(e^{j\omega}) = 1 - \sum_{k=1}^{P} \alpha_k^{(P)} e^{-j\omega k} \tag{7.42}$$

Since $H(e^{j\omega}) = \frac{G}{A(e^{j\omega})}$ we obtain

$$\text{Energy} = \frac{G^2}{2\pi} \int_{-\pi}^{\pi} \frac{|X(e^{j\omega})|^2}{|H(e^{j\omega})|^2} d\omega \tag{7.43}$$

As you may recall from our discussion of maximum entropy spectral estimation, the autocorrelation coefficients of the filter with impulse response $h[n]$ are identical to those of the random process $x[n]$ for lags of magnitude less than or equal to P. In other words,

$$\phi_{xx}[m] = \phi_{hh}[m] \text{ for } m \leq P \tag{7.44}$$

Hence,

$$\lim_{P\to\infty} \phi_{hh}[m] = \phi_{xx}[m] \tag{7.45}$$

$$\lim_{P\to\infty} H(e^{j\omega}) = X(e^{j\omega}) \text{ and} \tag{7.46}$$

$$\lim_{P\to\infty} E_n = G^2 \tag{7.47}$$

We note from Eq. (7.43) that the energy of the error signal is the integrated ratio of the squared magnitude of $|X(e^{j\omega})|^2$ and $|H(e^{j\omega})|^2$. This causes the match between these two quantities to be closer for frequencies where these functions are of greater magnitude than those in which the functions have lesser magnitudes. In other words, $|X(e^{j\omega})|^2$ and $|H(e^{j\omega})|^2$ will match each other more closely at their peaks than at their valleys. This is also especially good for the application of LPC analysis to speech processing, as perceptual studies indicate that the peaks of the frequency response are much more important in determining percep- tion than the corresponding valleys.

7.3 The FIR lattice filter

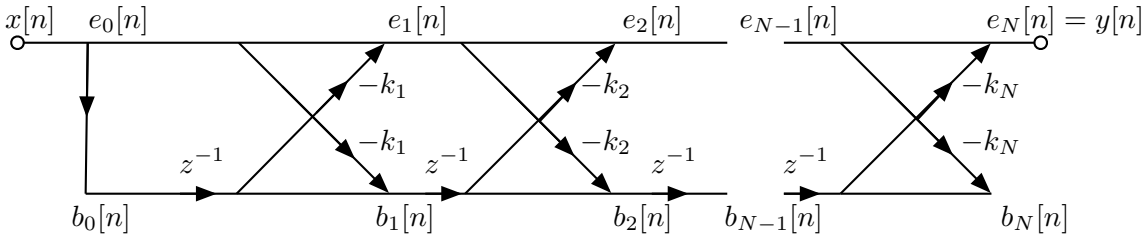


Figure 7.3: The FIR lattice filter.

Consider the basic lattice filter structure in the figure above. It should be obvious that this is an FIR filter structure, as it contains no feedback loops. In addition, if we set the input $x[n]$ to be equal to $\delta[n]$, we can observe easily by inspection that $h[0] = 1$ and $h[N] = -k_N$. The value of $h[n]$ for other values of n is obtained by observing all the different ways of passing a signal through the lattice while incurring exactly n delays, and adding all of the corresponding branch transmittances. It can be seen that the sample response will be a linear combination of the k_i.

7.3.1 Time-domain and frequency-domain characterization of the lattice filter

As can be seen from Fig. 7.3 above, the FIR lattice filter is defined by the following recursive relations:

$$x[n] = e_0[n] = b_0[n] \tag{7.48}$$
$$e_i[n] = e_{i-1}[n] - k_i b_{i-1}[n-1] \tag{7.49}$$
$$b_i[n] = -k_i e_{i-1}[n] + b_{i-1}[n-1] \tag{7.50}$$
$$y[n] = e_N[n] \tag{7.51}$$

Because the structure is FIR, we can make use of the following general characterization of its transfer function for the entire filter:

$$\frac{Y(z)}{X(z)} = 1 - \sum_{l=1}^{N} \alpha_l^{(N)} z^{-l} \equiv A(z) \tag{7.52}$$

We will also make use of the transfer function from the input to the $e_i[n]$ at a given stage of the lattice. For this, let

$$A_i(z) \equiv \frac{E_i(z)}{E_0(z)} = 1 - \sum_{l=1}^{i} \alpha_l^{(i)} z^{-l} \tag{7.53}$$

The corresponding transfer function from the input to the $b_i[n]$ at a given stage of the lattice is similarly

$$\tilde{A}_i(z) \equiv \frac{B_i(z)}{B_0(z)} \tag{7.54}$$

We note that $A_0(z) = \tilde{A}_0(z) = 1$ and $A_N(z) = Y(z)/X(z)$.

Using this notation, we can write the z-transforms of the equations that define the lattice as

$$X(z) = E_0(z) = B_0(z) \tag{7.55}$$

$$E_i(z) = E_{i-1}(z) - k_i z^{-1} B_{i-1}(z) \tag{7.56}$$

$$B_i(z) = -k_i E_{i-1}(z) + z^{-1} B_{i-1}(z) \tag{7.57}$$

$$Y(z) = E_N(z) \tag{7.58}$$

It is shown in Sec. 7.5 that if the $\{\alpha_l\}$ and $\{k_i\}$ are related by the Levinson-Durbin equation [and specifically Eq. (7.23) above], then

$$A_i(z) = A_{i-1}(z) - k_i z^{-i} A_{i-1}(z^{-1}) \tag{7.59}$$

and

$$\tilde{A}_i(z) = z^{-i} A_i(z^{-1}) \tag{7.60}$$

These equations are important because they enable us to develop a recursive characterization of the transfer function of the lattice filter stage by stage. Substituting Eq. (7.53) into Eq. (7.59) we obtain:

$$1 - \sum_{l=1}^{i} \alpha_l^{(i)} z^{-l} = 1 - \sum_{l=1}^{i-1} \alpha_l^{(i-1)} z^{-l} - k_i z^{-i} \left(1 - \sum_{l=1}^{i-1} \alpha_l^{(i-1)} z^{l} \right) \tag{7.61}$$

Matching the coefficients of an arbitrary term of power z^{-r} we obtain

$$-\alpha_r^{(i)} z^{-r} = -\alpha_r^{(i-1)} z^{-r} + k_i \alpha_{i-r}^{(i-1)} z^{-r} \tag{7.62}$$

or, of course

$$\alpha_r^{(i)} = \alpha_r^{(i-1)} - k_i \alpha_{i-r}^{(i-1)} \tag{7.63}$$

as specified by the Levinson-Durbin recursion. In other words, the FIR lattice filter is a structure in which the visible (and manipulable) coefficients of the filter are the reflection coefficients k_i of the Levinson-Durbin algorithm! Of course, you have already seen other filter structures in which the manipulable coefficients are parameters in the formal mathematical description. For example, the coefficients of an FIR filter structure are the values of the unit sample response of that filter. In addition, the coefficients of the frequency-sampling structure form of an FIR filter are the DFT coefficients that specify that filter's response in the frequency domain. In this case, the all-zero transfer function of this lattice filter is the reciprocal of the all-pole model used to describe the original random process, or in other words the filter $A(z)$ is the *inverse* of $H(z)$ in Eq. (7.10) if we set the gain parameter G equal to 1.

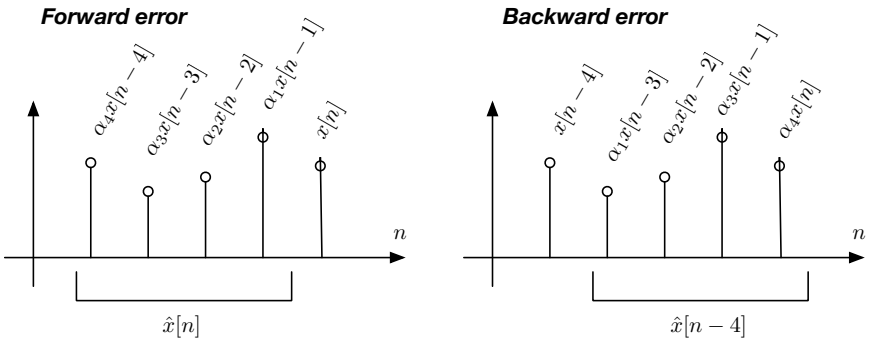


Figure 7.4: Comparison of the sample values used for calculation of forward and backward error with $P = 4$.

7.3.2 Physical interpretation of the functions $e_i[n]$ and $b_i[n]$

Up until now we have been thinking of the functions $e_i[n]$ and $b_i[n]$ as arbitrary internal functions. Nevertheless, they each do have a physical meaning relating to linear prediction error. Consider first the transfer function to the functions in the upper "rail" of the lattice:

$$A_i(z) = \frac{E_i(z)}{E_0(z)} = \frac{E_i(z)}{X(z)} = 1 - \sum_{l=1}^{i} \alpha_l^{(i)} z^{-l} \tag{7.64}$$

Taking the inverse z-transform we obtain

$$e_i[n] = x[n] - \sum_{l=1}^{i} \alpha_l^{(i)} x[n-l] = x[n] - \hat{x}[n] \tag{7.65}$$

which is identical (to within a sign) to the linear prediction error defined in Eq. (7.2). Again, this expression describes the difference between the current sample $x[n]$ and the "best" linear prediction of $x[n]$ using the previous i samples. Hence the expression $e_i[n]$ is referred to as the *i^{th}-order forward prediction error*, as illustrated in the left panel of Fig. 7.4.

Let us now consider the functions $b_i[n]$ in the lower "rail" of the lattice. Combining Eqs. (7.54) and (7.60) we obtain

$$\tilde{A}_i(z) = z^{-i} A_i(z^{-1}) = \frac{B_i(z)}{B_0(z)} = \frac{B_i(z)}{X(z)} \tag{7.66}$$

$$\frac{B_i(z)}{X(z)} = z^{-i}\left(1 - \sum_{l=1}^{i} \alpha_l^{(i)} z^{l}\right) = z^{-i} - \sum_{l=1}^{i} \alpha_l^{(i)} z^{l-i} \tag{7.67}$$

Again, taking the inverse z-transform we obtain

$$b_i[n] = x[n-i] - \sum_{l=1}^{i} \alpha_l^{(i)} x[n+(l-i)] \tag{7.68}$$

Comparing Eqs. (7.65) and (7.68) we observe that $b_i[n]$ represents the difference between $x[n-i]$, the value of the input function i samples ago, and some linear combination of the

following i samples of the input, running from $x[n-(l-i)]$ right up to $x[n]$. In fact, the same linear prediction coefficients are used, but they are applied backward. One way of thinking about this is that $b_i[n]$ is what we would have obtained if we calculated $e_i[n]$ but with the input function $x[n]$ presented in time-reversed order. Because of all this, $b_i[n]$ is referred to as the i^{th}- *order backward prediction error*, as illustrated in the right panel of Fig. 7.4.

7.3.3 Deriving the reflection coefficients from the forward and backward prediction errors: the PARCOR method

In Sec. 7.2.2 we derived the LPC coefficients i and the reflection coefficients k_i of the best-fit all-pole model to the samples of a random process by implementing the Levinson-Durbin recursion to solve the autocorrelation equations. As you will recall, equations were developed by starting with the difference equation relating the input and output,

$$x[n] = \sum_{k=1}^{P} \alpha_k^{(P)} x[n-k] + Gw[n] \tag{7.69}$$

and finding the values of the $\alpha_k^{(P)}$ that minimize the expected value of the square of the forward error. Specifically, starting with

$$E = \sum_m \left(x[m] - \sum_{k=1}^{P} \alpha_k x[m-k] \right)^2 \tag{7.70}$$

we computed the partial derivative of E with respect to each of the α_k we obtained the equations

$$\sum_{k=1}^{P} \alpha_k^{(P)} \phi[|i-k|] = \phi[i] \tag{7.71}$$

where

$$\phi[i] = \mathbb{E}[x[n]x[n+i]] \tag{7.72}$$

With knowledge of the values of the autocorrelation coefficients for $i = 0, 1, 2, ..., P$ we can use the Levinson-Durbin recursion to obtain all the LPC coefficients $\alpha_k^{(i)}$ for model orders 1 through P and the corresponding reflection coefficients k_i.

We can also obtain estimates of the reflection coefficients k_i (and subsequently the LPC coefficients $\alpha_k^{(i)}$) using expressions for forward and backward error developed in the previous section. Specifically, if we let

$$E_i^f = \sum_{n=0}^{N-1} (e_i[n])^2 \tag{7.73}$$

the corresponding mean square error using statistical averages would be

$$\xi^2 = \mathbb{E}[e_i^2[n]] = \mathbb{E}\left[(e_{i-1}[n] - k_i b_{i-1}[n-1])^2 \right] \tag{7.74}$$

Computing the derivative of ξ^2 with respect to k_i, setting the derivative to zero, and solving for k_i produces the estimate

$$k_i = \frac{\mathbb{E}[e_{i-1}[n]b_{i-1}[n-1]]}{\mathbb{E}[b_{i-1}^2[n-1]]} \tag{7.75}$$

Typically the expected values are implemented as time averages, resulting in the expression

$$k_i^f = \frac{\sum_{n=0}^{N-1} e_{i-1}[n]b_{i-1}[n-1]}{\sum_{n=0}^{N-1} b_{i-1}^2[n-1]} \tag{7.76}$$

where the superscript f in the symbol k_i^f reminds us that this version of the reflection coefficient was derived using the mean squared forward error $e_i[n]$.

Note that this estimate for the reflection coefficient at a given stage of the lattice is expressed in terms of the expected values of the products of the forward and backward errors of the previous stage in the numerator, and the expected value of the square of the backward prediction error in the denominator. The expression in the numerator is actually the *cross-correlation* of the forward and backward error functions of the previous stage, and the expression in the denominator is the *energy* of the backward error of the previous stage.

Because of these physical interpretations, this method of obtaining the estimate of the reflection coefficients is referred to as the *partial correlation* or *PARCOR* method. Generalizing to an abitrary number of stages, the calculation begins by computing the zeroth-order forward and backward errors directly by setting them equal to the energy of the input $x[n]$, calculating k_1 using Eq. (7.76), and then obtaining the next set of forward and backward errors using Eqs. (7.49) and (7.50). Reflection coefficients for the remaining stages are obtained easily by iterating over Eqs. (7.76), (7.49) and (7.50). As you will recall, with the autocorrelation and covariance methods described in Sec. 7.2, we obtained the LPC parameters and reflection coefficients by beginning with a complete set of estimates of the autocorrelation functions of the input. In contrast, the PARCOR method computes the autocorrelation function coefficients indirectly, through a recursive computation of cross-correlation of prediction error functions. This approach has some very attractive statistical properties and is widely used.

Of course, there is nothing magic about the forward prediction error. We can just as easily perform a similar calculation with the backward prediction error $b_i[n]$. Performing a similar set of operations on the backward prediction error produces the very similar estimate for the reflection coefficient

$$k_i^b = \frac{\sum_{n=0}^{N-1} e_{i-1}[n]b_{i-1}[n-1]}{\sum_{n=0}^{N-1} e_{i-1}^2[n]} \tag{7.77}$$

Various methods have been proposed for combining the two estimates of the reflection coefficients obtained using the PARCOR method, k_i^f and k_i^b . For example, the *Itakura estimate* of the reflection coefficients is obtained by combining these two results according to the equation

$$k_l^I = \sqrt{k_l^f k_l^b} \tag{7.78}$$

The *Burg estimate* of the reflection coefficients produced by combining these two results according to the equation

$$k_l^B = \frac{2k_l^f k_l^b}{k_l^f + k_l^b} \tag{7.79}$$

7.4 All-pole IIR lattice filters

As noted above, we developed in Sec. 7.3 an all-zero lattice filter with the transfer function

$$A(z) = 1 - \sum_{l=1}^{N} \alpha_l^{(N)} z^{-l} = \frac{E_N(z)}{E_0(z)} \tag{7.80}$$

Referring to the figure at the beginning of Sec. 7.3, we note that the input is $x[n] = e_0[n]$ and the output is $y[n] = e_N[n]$. If we could maintain the same filter structure but interchange the input and output, we would obtain the transfer function

$$\frac{E_0(z)}{E_N(z)} = \frac{1}{1 - \sum_{l=1}^{N} \alpha_l^{(N)} z^{-l}} \tag{7.81}$$

which clearly is an all-pole transfer function, and in fact is exactly the transfer function of the original filter considered, $H(z)$, with the gain factor G set equal to 1.

Recall that the original definitions of the stages of the FIR lattice filter were

$$e_i[n] = e_{i-1}[n] - k_i b_{i-1}[n-1] \tag{7.82}$$
$$b_i[n] = -k_i e_{i-1}[n] + b_{i-1}[n-1] \tag{7.83}$$

With a trivial amount of algebra, Eq. (7.82) can be rewritten as

$$e_{i-1}[n] = e_i[n] + k_i b_{i-1}[n-1] \tag{7.84}$$

Eqs. (7.83) and (7.84) suggest the following lattice structure for a single stage:

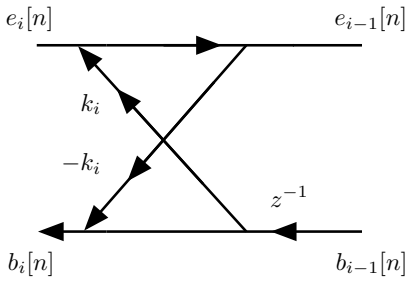


Figure 7.5: Signal flow diagram of a single stage of the IIR lattice filter.

Combining into multiple stages, we obtain the following IIR lattice structure:

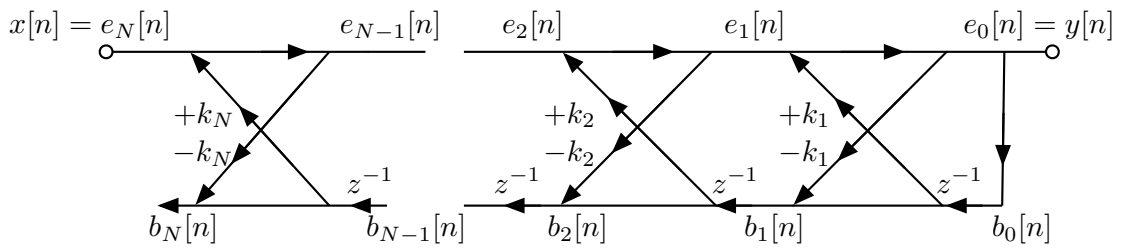


Figure 7.6: Signal flow diagram of a multistage IIR lattice filter.

Note that $e_N[n]$ is now the input and that $e_0[n]$ is the output. This filter will have the transfer function

$$H(z) = \frac{1}{1 - \sum_{l=1}^{N} \alpha_l^{(N)} z^{-l}} \tag{7.85}$$

where the LPC parameters are related to the reflection coefficients according to the usual Levinson-Durbin relationship. Since the filter is IIR with feedback loops, it does have the potential to be unstable. However, it is guaranteed to remain stable if

$$|k_i| < 1 \text{ for all } i \tag{7.86}$$

As we noted above, this condition is guaranteed to be satisfied for the autocorrelation and PARCOR solutions of the LPC equations because these solutions are based on the equations of the Levinson-Durbin recursion.

7.5 Proof of the recursive lattice filter relationship

To prove: $A^{(i)}(z) = A^{(i-1)}(z) - k_i z^{-l} A^{(i-1)}(z^{-1})$

From the definition of $A^{(i)}(z)$ and the Levinson-Durbin relation we have

$$A^{(i)}(z) = 1 - \sum_{j=1}^{i} \alpha_j^{(i)} z^{-j}$$
$$\alpha_j^{(i)} = \alpha_j^{(i-1)} - k_i \alpha_{i-j}^{(i-1)}, 1 \le j \le i-1$$
$$\alpha_i^{(i)} = k_i$$

Substituting for $\alpha_i^{(i)}$, we have

$$A^{(i)}(z) = 1 - \sum_{j=1}^{i-1} \alpha_j^{(i)} z^{-j} - k_i z^{-i} \tag{7.87}$$

Substituting for $\alpha_j^{(i)}$

$$A^{(i)}(z) = 1 - \sum_{j=1}^{i-1} [\alpha_j^{(i-1)} z^{-j} - k_i \alpha_{i-j}^{(i-1)} z^{-j}] - k_i z^{-i}$$
$$= \left[1 - \sum_{j=1}^{i-1} \alpha_j^{(i-1)} z^{-j}\right] + k_i \sum_{j=1}^{i-1} \alpha_{i-j}^{(i-1)} z^{-j} - k_i z^{-i}$$

In the second term, let $j = i - j'$, then

$$A^{(i)}(z) = \left[1 - \sum_{j=1}^{i-1} \alpha_j^{(i-1)} z^{-j}\right] + k_i \sum_{j'=i-1}^{1} \alpha_{j'}^{(i-1)} z^{j'-i} - k_i z^{-i}$$
$$= \left[1 - \sum_{j=1}^{i-1} \alpha_j^{(i-1)} z^{-j}\right] - k_i z^{-i} \left[1 - \sum_{j'=1}^{i-1} \alpha_{j'}^{(i-1)} (z^{-1})^{-j'}\right]$$
$$= A^{(i-1)}(z) - k_i z^{-l} A^{(i-1)}(z^{-1})$$

8. Introduction to Adaptive Filtering

Classical discrete-time filters have fixed coefficients that are determined to meet predefined specifications such as passband and stop band ripple, critical frequencies, etc. This chapter describes the structure and function fo simple adaptive filters, which have coefficients that vary with time in order to realize a performance goal such as noise suppression or signal prediction. The optimal values of these time-varying coefficients are typically not known a priori *or they may vary with time as environmental conditions change.*

8.1 Introduction

Conventional filters with fixed coefficients are designed with a particular goal in mind, such as attenuating all frequencies above a particular "cutoff" frequency. The design procedure frequently consists of choosing the set of coefficients that would best approximate a desired frequency response. There are multiple well-developed techniques to accomplish this, as was discussed in basic digital signal processing courses.

Adaptive filters, in contrast, have coefficients that are allowed to vary over time. They are used when the filter response that best accomplishes a particular task is not known *a priori*, or when the nature of the operating environment is expected to change over time. A typical system goal would be for a filter to suppress undesired noise to the greatest extent possible while leaving the target signal (which frequently is speech or music) intact to the extent possible.[1]

For various historical reasons, the notational conventions commonly used in the literature to describe adaptive filters and the signals that pass through them are different from the standard notation that we have used above to describe frequency analysis and conventional filters. Specifically, the system input and output are frequently denoted as x_k and y_k, respectively, rather than $x[n]$ and $y[n]$ as in previous chapters. (Note that the time

[1] The material in this chapter is a condensation of the chapter on adaptive filters by Stearns in the text edited by Lim and Oppenheim. The Stearns chapter is in turn an abstraction of many topics in the book *Adaptive Signal Processing* published in 1985 by B. Widrow and S. D. Stearns.

variable is now a subscript rather than a function argument, and that we are now using k rather than n to denote time.)

Figure 8.1 above is a diagram of the standard adaptive signal processor. Using the modified notation just introduced, x_k represents the input to and y_k represents the output from the adaptive processor. The signal d_k represents the so-called *desired signal*. The signal $e_k = d_k - y_k$ is referred to as the *error signal*, which is the difference between the desired signal (which is what we presumably want) and the adaptive processor's output (which is what we are presently getting). The coefficients of the adaptive processor are manipulated in a fashion that minimizes the square of the error signal $\xi^2 = \mathbb{E}[e_k^2]$, the *mean-square error* or MSE. This tends to make the signal y_k at the output of the adaptive processor to be as much as possible like the desired signal d_k. The diagonal arrow through the adaptive processor box indicates that the coefficients of the adaptive processor are variable, and that their values are controlled by the error signal $e_k = d_k - y_k$.

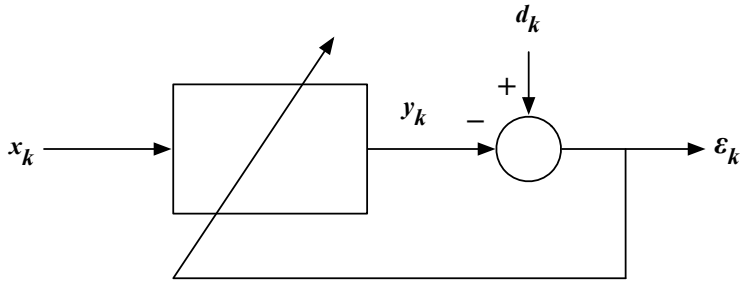


Figure 8.1: Block diagram of a standard adaptive signal processor.

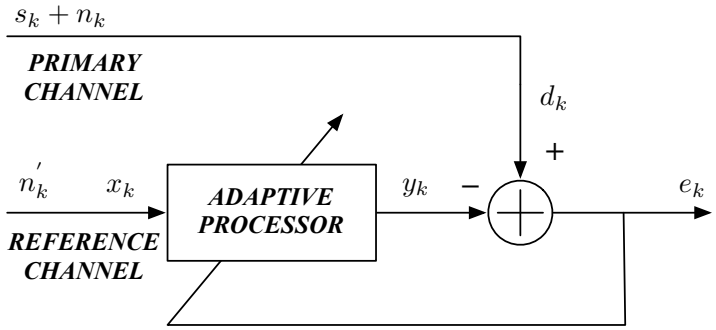


Figure 8.2: Block diagram of a standard noise cancellation system.

While it may not be obvious why the structure in Fig. 8.1 is potentially useful, the purpose of the various elements becomes more evident when considered in the context of the common *adaptive noise cancellation* structure as in Fig. 8.2 above. The primary channel consists of the sum of a target speech or music signal s_k to which a noise signal n_k is added. (The term "noise signal" refers to any kind of noise or interference (including echoes) that is added to the target signal; n_k could, for example, represent background speech or music that is simply not what is desired.) The reference channel (which is also sometimes called the secondary channel) contains a second noise signal n_k' that is correlated with n_k.

As an example, consider one of the earliest applications of adaptive noise cancellation: communication among fighter pilots wearing oxygen masks in military aircraft. The audio signal input to the primary channel is detected by a microphone inside the mask, and it consists of the pilot's speech plus a filtered version of the very loud ambient noise inside the aircraft. The reference channel signal n'_k is obtained from a microphone within the cockpit but outside the mask. This signal consists of the ambient cockpit noise without the filtering that occurs when the cabin noise is propagated through the pilot's mask, and essentially none of the pilot's speech (which is at a much lower intensity level compared to the noise in the cabin). The adaptive processor attempts to filter the noise signal n_k in the reference channel so that it most closely approximates the noise component n_k in the primary channel. If y_k were to closely approximate n_k, the two would almost cancel each other out, and the system output would consist mostly of the target signal. In practice, the amount of noise cancellation that is obtained depends on the degree of correlation between the noise in the primary and reference channels (greater correlation provides greater cancellation). In addition, it is very important that the target signal not leak into the reference channel and that the target and noise signals be statistically independent.

8.2 The adaptive linear combiner

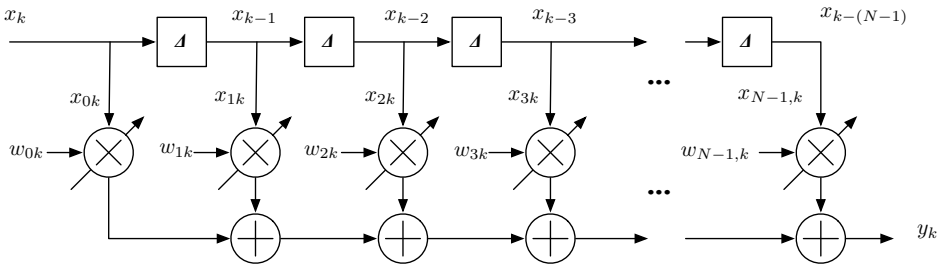


Figure 8.3: FIR filter structure as used in adaptive filtering.

While there are a number of different types of adaptive processors, the easiest type to study is the *adaptive linear combiner* (ALC). The output of an ALC can be described by the equation

$$y_k = \sum_{l=0}^{L-1} x_{lk} w_{lk} \tag{8.1}$$

where the $\{w_{lk}\}$ represent the variable weights of the adaptive filter and the signals $\{x_{lk}\}$ are the inputs to those weights. While this equation can be applied directly to any system for which the output is obtained by computing the linear combination of the outputs of an array of sensors, the most common implementation for the adaptive processor is that of an FIR filter with variable coefficients. This structure is depicted in Fig. 8.3 above with the notational conventions commonly used to describe adaptive systems. We note that the filter coefficients (or "weights") are now variables with the variable name w_{lk}, where the first subscript identifies the individual filter coefficient under consideration and the second coefficient indicates the time index. The diagonal arrows through the multiplication symbols again indicate that the filter coefficients vary over time. Another

important variable shown in the picture is the *observation vector*, which is by definition the set of signal inputs to the variable weights. If the adaptive processor is in the form of an FIR filter, the observation vector components x_{lk} are equal to delayed versions of the original input where $x_{lk} = x_{k-l}$, which approximates n'_k for the application in Figure 3.[2]

It is frequently convenient to represent the inputs $\{x_{lk}\}$ and variable filter coefficients w_{lk} as column vectors:

$$\boldsymbol{X}_k = \begin{bmatrix} x_{0k} \\ x_{1k} \\ x_{2k} \\ \vdots \\ x_{L-1,k} \end{bmatrix} \text{ and } \boldsymbol{W}_k = \begin{bmatrix} w_{0k} \\ w_{1k} \\ w_{2k} \\ \vdots \\ w_{L-1,k} \end{bmatrix} \tag{8.2}$$

Using the notation of linear algebra it is easy to show that

$$y_k = \sum_{l=0}^{L-1} x_{lk} w_{lk} = \boldsymbol{X}_k^T \boldsymbol{W}_k = \boldsymbol{W}_k^T \boldsymbol{X}_k \tag{8.3}$$

where the T in the equation above denotes the matrix transpose operator.

8.3 The performance function

In order to develop algorithms for minimizing the mean square error we must first develop an analytical expression for what that error is. As before we define the error as

$$e_k = d_k - y_k = d_k - \boldsymbol{X}_k^T \boldsymbol{W}_k = d_k - \boldsymbol{W}_k^T \boldsymbol{X}_k \tag{8.4}$$

and

$$e_k^2 = d_k^2 + \boldsymbol{W}_k^T \boldsymbol{X}_k \boldsymbol{X}_k^T \boldsymbol{W}_k - 2 d_k \boldsymbol{W}_k^T \boldsymbol{X}_k \tag{8.5}$$

If d_k, e_k, and x_k are all zero mean and wide-sense stationary, and statistically independent of one another (which in real life is normally not true), then

$$\xi^2 = \mathbb{E}[e_k^2] = \mathbb{E}[d_k^2] + \boldsymbol{W}_k^T \mathbb{E}[\boldsymbol{X}_k \boldsymbol{X}_k^T] \boldsymbol{W}_k - 2\mathbb{E}[d_k \boldsymbol{X}_k^T] \boldsymbol{W}_k \tag{8.6}$$

To simplify the notation, let us define

$$\boldsymbol{R} = \mathbb{E}[\boldsymbol{X}_k \boldsymbol{X}_k^T] \text{ and } \boldsymbol{P} = \mathbb{E}[d_k \boldsymbol{X}_k] \tag{8.7}$$

Note that $\boldsymbol{R}$ is an $L \times L$ matrix and that $\boldsymbol{P}$ is an $L \times 1$ vector. Hence it follows directly that

$$\xi^2 = \mathbb{E}[e_k^2] = \mathbb{E}[d_k^2] + \boldsymbol{W}^T \boldsymbol{R} \boldsymbol{W} - 2\boldsymbol{P}^T \boldsymbol{W} \tag{8.8}$$

As noted above, the coefficients in the adaptive filter are normally manipulated to minimize the statistically-averaged squared value of the error signal ξ^2. Note that the expression for the MSE ξ^2 above is a quadratic function of the weight vector $\boldsymbol{W}$. As a

[2] The multiple use of the variable x in this equation is unfortunate. The variable x with a single subscript denotes delayed versions of the input; the variable x with double subscripts denotes individual components of the observation vector.

consequence, there is only one single minimum value of ξ^2 over all the possible values of the coefficients in $\boldsymbol{W}$.

For purposes of illustration, let us consider an adaptive filter that has only two variable coefficients, w_0 and w_1. Figure 8.4 depicts a typical example of the "performance surface" for such an adaptive filter, which is a plot of the MSE ξ^2 as a function of w_0 and w_1. As can be seen, the performance surface is bowl-shaped, and the surface is in the form of a parabola in two dimensions (which is actually called a hyperparabola). As noted above, every quadratic performance surface has a single minimum, which in this example is at $w_0 = 40$ and $w_1 = 60$. These coefficient values are the best possible values for the (imaginary) filter under consideration. In principle, the shape of the performance surface depends on the specific optimal values of the coefficients for a particular application (40 and 60 in this case), the correlations between the components of the observation vector $\boldsymbol{X}_k$ and the desired signal d_k and (especially) the correlations of the components of the observations $\boldsymbol{X}_k$ with each other, which is 0.7 in the example in Fig. 8.4.

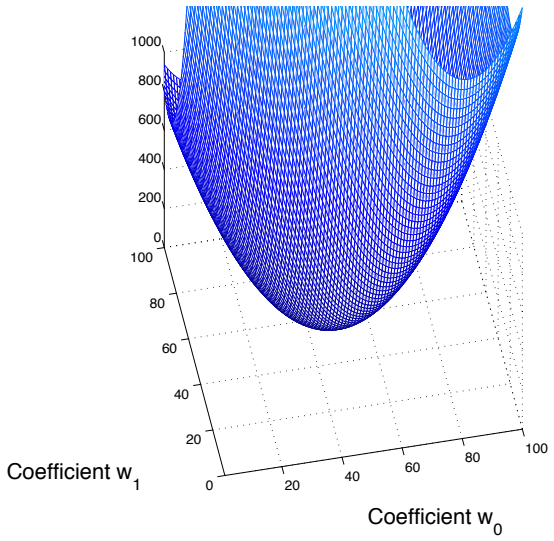


Figure 8.4: Plot of the mean-square error ξ^2 as a joint function of the two filter coefficients, w_0 and w_1.

Another way of describing the performance surface is in terms of its "contour lines" as in Fig. 8.5. The left panel of Fig. 8.5 depicts a performance surface for a system in which the optimal adaptive filter coefficients are 40 and 60, and the correlation ρ between the two components x_0 and x_1 of the observations is equal to 0.7, as in Fig. 8.4. The right panel of Fig. 8.5 depicts the contour lines of a similar performance surface, but in this case the correlation between the components of the observations equals 0.0 (i.e., there is no correlation between x_0 and x_1). In general, the contours lines of the performance surface are circles if and are uncorrelated (as in the right panel of the figure), and they become increasingly-elongated ellipses as the correlation increases. (In linear algebra terms, increasingly-elongated ellipsoidal contour lines reflect increasing spread of the eigenvalues of the correlation matrix of the observations.) The degree of correlation of the observations is important, as will be discussed below.

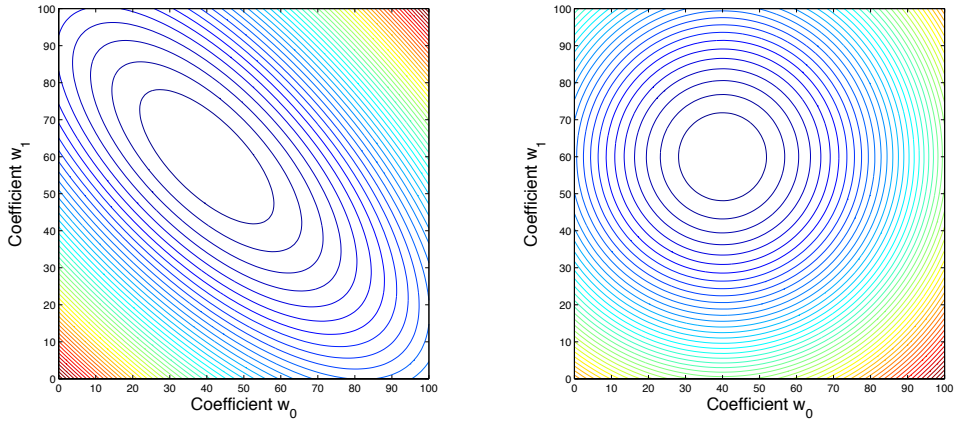


Figure 8.5: Contour plots of mean-squared error for two values of the correlation ρ between x_0 and x_1: 0.7 (left panel) and 0.0 (right panel). The minimum value of ξ^2 is obtained with the values of the filter coefficients w_0 and w_1 are equal to 40 and 60, respectively, in both cases.

8.4 Finding the minimum MSE analytically

If there were only a single weight w, we would obtain the value of w that minimizes ξ^2 by differentiating with respect to w, setting the derivative equal to zero, and solving for w. With the vector $\boldsymbol{W}$ we perform a similar series of operations, working from the gradient operator. Define the *gradient* of ξ^2 to be

$$\nabla = \frac{\partial \xi^2}{\partial \boldsymbol{W}} = \begin{bmatrix} \frac{\partial \xi^2}{\partial w_0} \\ \frac{\partial \xi^2}{\partial w_1} \\ \vdots \\ \frac{\partial \xi^2}{\partial w_{L-1}} \end{bmatrix} \tag{8.9}$$

Recall from Eq. (8.7) above that

$$\xi^2 = \mathbb{E}[e_k^2] = \mathbb{E}[d_k^2] + \boldsymbol{W}^T \boldsymbol{R} \boldsymbol{W} - 2\boldsymbol{P}^T \boldsymbol{W}$$

Differentiating with respect to the elements of $\boldsymbol{W}$ it follows directly that

$$\nabla = \frac{\partial \xi^2}{\partial \boldsymbol{W}} = 2\boldsymbol{R}\boldsymbol{W} - 2\boldsymbol{P} \tag{8.10}$$

Let $\boldsymbol{W}^*$ represent the value of $\boldsymbol{W}$ that minimizes ξ^2. $\boldsymbol{W}^*$ is easily obtained by solving

$$\nabla = \boldsymbol{0} = 2\boldsymbol{R}\boldsymbol{W}^* - 2\boldsymbol{P} \text{ which produces } \boldsymbol{W}^* = \boldsymbol{R}^{-1}\boldsymbol{P} \tag{8.11}$$

We note that this is the familiar *Wiener-Hopf* equation, Eq. (7.15), which you have encountered before in conjunction with the discussion on linear prediction. We can also

easily obtain an expression for the minimum MSE by plugging $\boldsymbol{W}^*$ into the expression for the MSE:

$$\begin{aligned}\xi^2_{min} &= \mathbb{E}[d_k^2] + \boldsymbol{W}^{T*}\boldsymbol{R}\boldsymbol{W}^* - 2\boldsymbol{P}^T\boldsymbol{W}^* \\ &= \mathbb{E}[d_k^2] + (\boldsymbol{R}^{-1}\boldsymbol{P})^T\boldsymbol{R}\boldsymbol{R}^{-1}\boldsymbol{P} - 2\boldsymbol{P}^T\boldsymbol{R}^{-1}\boldsymbol{P}\end{aligned} \tag{8.12}$$

Combining terms we obtain

$$\begin{aligned}\xi^2_{min} &= \mathbb{E}[d_k^2] + \boldsymbol{P}^T\boldsymbol{R}^{-1}\boldsymbol{P} - 2\boldsymbol{P}^T\boldsymbol{R}^{-1}\boldsymbol{P} \\ &= \mathbb{E}[d_k^2] - \boldsymbol{P}^T\boldsymbol{R}^{-1}\boldsymbol{P} \\ &= \mathbb{E}[d_k^2] - \boldsymbol{P}^T\boldsymbol{W}^*\end{aligned} \tag{8.13}$$

Note that the MMSE depends only on $\boldsymbol{R}$ and $\boldsymbol{P}$. And please remember that $\boldsymbol{W}^*$ refers to the optimal coefficients, not complex conjugation.

8.5 Finding the minimum MSE empirically

In this section we will discuss two methods to find the minimum MSE empirically: Newton's method and gradient descent.

8.5.1 Newton's method

Recall that we discussed the gradient in the section above:

$$\nabla = \frac{\partial \xi^2}{\partial \boldsymbol{W}} = \begin{bmatrix} \frac{\partial \xi^2}{\partial w_0} \\ \frac{\partial \xi^2}{\partial w_1} \\ \vdots \\ \frac{\partial \xi^2}{\partial w_{L-1}} \end{bmatrix} = 2\boldsymbol{R}\boldsymbol{W} - 2\boldsymbol{P} \tag{8.14}$$

Pre-multiplying both sides of the above equation by $\frac{1}{2}\boldsymbol{R}^{-1}$ produces

$$\frac{1}{2}\boldsymbol{R}^{-1}\nabla = \frac{1}{2}\boldsymbol{R}^{-1}[2\boldsymbol{R}\boldsymbol{W} - 2\boldsymbol{P}] = \boldsymbol{W} - \boldsymbol{R}^{-1}\boldsymbol{P} = \boldsymbol{W} - \boldsymbol{W}^* \tag{8.15}$$

Rearranging the terms of the above equation gives us a different way of thinking about $\boldsymbol{W}^*$:

$$\boldsymbol{W}^* = \boldsymbol{W} - \frac{1}{2}\boldsymbol{R}^{-1}\nabla \tag{8.16}$$

This is essentially Newton's root finding method set up to find the zero of the gradient operator. What this means in principle is that if we have perfect knowledge of $\boldsymbol{R}$, we can move to the "bottom of the bowl" (i.e., the minimum value of ξ^2) in one step because knowing the position and slope is sufficient to detect the bottom of the bowl and because the surface is quadratic and the statistics are known. Of course neither of these is true in real life: we do not know $\boldsymbol{R}$ and we do not know ∇ ... both must be estimated empirically.

8.5.2 Gradient descent

One way to "hedge our bets" about the value of that provides the MMSE is to move toward the bottom of the bowl in small increments, rather than in one big step as in the case of Newton's method. Specifically, we can move toward the MMSE value at the bottom of the bowl one step at a time using the relationship

$$\boldsymbol{W}_{k+1} = \boldsymbol{W}_k - \mu \boldsymbol{R}^{-1} \nabla \tag{8.17}$$

Keep in mind that the gradient operator produces a vector that points in the direction of "up", so what this equation teaches us to do is locate the direction on the plane that corresponds to "down" and walk a little ways in the direction of "down". The parameter μ in the equation above is the stepsize parameter. Clearly, larger values of μ would enable us to get to the bottom of the bowl faster, but the residual error once we are close to the bottom would be greater for larger values of μ as the coefficients are likely to oscillate around their true values because of the larger stepsizes.

Since $\nabla = 2\boldsymbol{R}\boldsymbol{W} - 2\boldsymbol{P}$, we can rewrite the iterative equation above as

$$\boldsymbol{W}_{k+1} = \boldsymbol{W}_k - \mu \boldsymbol{R}^{-1}(2\boldsymbol{R}\boldsymbol{W} - 2\boldsymbol{P}) = (1 - 2\mu)\boldsymbol{W}_k + 2\mu \boldsymbol{W}^* \tag{8.18}$$

We can get some idea how the coefficient trajectories evolve by assuming an arbitrary initial coefficient vector $\boldsymbol{W}_0$ and iterating Eq. (8.18) a few times:

$$\boldsymbol{W}_1 = (1 - 2\mu)\boldsymbol{W}_0 + 2\mu \boldsymbol{W}^* \tag{8.19}$$

$$\boldsymbol{W}_2 = (1 - 2\mu)\boldsymbol{W}_1 + 2\mu \boldsymbol{W}^* = (1 - 2\mu)\left[(1 - 2\mu)\boldsymbol{W}_0 + 2\mu \boldsymbol{W}^*\right] + 2\mu \boldsymbol{W}^* \tag{8.20}$$

$$= (1 - 2\mu)^2 \boldsymbol{W}_0 + 2\mu \boldsymbol{W}^*(1 + (1 - 2\mu)) \tag{8.21}$$

Similarly, $$\boldsymbol{W}_3 = (1 - 2\mu)^3 \boldsymbol{W}_0 + 2\mu \boldsymbol{W}^*(1 + (1 - 2\mu) + (1 - 2\mu)^2) \tag{8.22}$$

and $$\boldsymbol{W}_k = (1 - 2\mu)^k + 2\mu \boldsymbol{W}^* \sum_{i=0}^{k-1} (1 - 2\mu)^i \tag{8.23}$$

Using the formula for the finite sum of exponentials we obtain

$$\boldsymbol{W}_k = (1 - 2\mu)^k \boldsymbol{W}_0 + \boldsymbol{W}^* \frac{1 - (1 - 2\mu)^k}{1 - (1 - 2\mu)} \tag{8.24}$$

$$= (1 - 2\mu)^k \boldsymbol{W}_0 + \boldsymbol{W}^*(1 - (1 - 2\mu)^k) \tag{8.25}$$

While it may not be obvious, $(1 - 2\mu)$ in the first term is a sampled decaying exponential and $(1 - (1 - 2\mu)^k)$ in the second term is a sampled rising exponential. Equation (8.25) would have been more transparent had it been written in the form of

$$\boldsymbol{W}_k = e^{-t_k/\tau} \boldsymbol{W}_0 + (1 - e^{-t_k/\tau}) \boldsymbol{W}^* \tag{8.26}$$

where $t_k = k$ in this expression. We can expand the exponential in a Taylor series

$$e^{-1/\tau} = 1 - 1/\tau + \frac{(-1/\tau)^2}{2!} + \frac{(-1/\tau)^3}{3!} + \ldots \tag{8.27}$$

Keeping only the first two terms produces

$$e^{-1/\tau} \approx 1 - 1/\tau \tag{8.28}$$

This implies that

$$e^{-k/\tau} = e^{(-1/\tau)^k} \approx (1 - 1/\tau)^k \text{ which corresponds to } (1 - 2\mu)^k \tag{8.29}$$

Matching terms provides the correspondence

$$1/\tau = 2\mu \text{ or } \tau = 1/2\mu \tag{8.30}$$

Summarizing, this tells us that the recursion for $\boldsymbol{W}_k$ causes the coefficient vector to move exponentially from $\boldsymbol{W}_0$ toward the true optimal coefficient vector $\boldsymbol{W}^*$, and that the time constant τ is inversely proportional to the stepsize parameter μ. The average error curve for the coefficients also decays exponentially.

8.6 The least mean squares (LMS) adaptation algorithm

The gradient descent solution developed in Eq. (8.17) in the previous section

$$\boldsymbol{W}_{k+1} = \boldsymbol{W}_k - \mu \boldsymbol{R}^{-1} \nabla$$

has two sets of problems: we don't know the values of the statistics $\boldsymbol{R}$ and $\boldsymbol{P}$ and we don't know the value of ∇, either. We get around these problems by making two changes:

1. Ignore the factor of $\boldsymbol{R}^{-1}$ in the above equation.
2. Replace the average statistics $\boldsymbol{R}$ and $\boldsymbol{P}$ by their instantaneous values. Specifically, we replace $\boldsymbol{R} = \mathbb{E}[\boldsymbol{X}_k \boldsymbol{X}_k^T]$ by $\boldsymbol{X}_k \boldsymbol{X}_k^T$ and we replace $\boldsymbol{P} = \mathbb{E}[d_k \boldsymbol{X}_k]$ by $d_k \boldsymbol{X}_k$.

In other words, we replace

$$\begin{aligned} \boldsymbol{W}_{k+1} &= \boldsymbol{W}_k - \mu \boldsymbol{R}^{-1} \nabla \\ &= \boldsymbol{W}_k - \mu \boldsymbol{R}^{-1} (2\boldsymbol{R}\boldsymbol{W}_k - 2\boldsymbol{P}) \\ &= \boldsymbol{W}_k - \mu \boldsymbol{R}^{-1} (2\mathbb{E}[\boldsymbol{X}_k \boldsymbol{X}_k^T]\boldsymbol{W}_k - 2\mathbb{E}[d_k \boldsymbol{X}_k]) \end{aligned}$$

by

$$\boldsymbol{W}_{k+1} = \boldsymbol{W}_k - 2\mu(\boldsymbol{X}_k \boldsymbol{X}_k^T \boldsymbol{W}_k - d_k \boldsymbol{X}_k) \tag{8.31}$$

Based on our definitions for y_k and e_k, we can rewrite this expression as

$$\boldsymbol{W}_{k+1} = \boldsymbol{W}_k - 2\mu \boldsymbol{X}_k (y_k - d_k) \text{ or} \tag{8.32}$$

$$\boldsymbol{W}_{k+1} = \boldsymbol{W}_k + 2\mu \boldsymbol{X}_k e_k \tag{8.33}$$

This extremely simple expression is the *least-mean squares* or LMS algorithm, developed by Widrow in the 1970s. It is one of the most widely used adaptation algorithms. We note that the coefficient trajectories produced by the LMS algorithm become a noisy approximation to the ideal trajectories that would have been obtained using the gradient descent algorithm if the actual statistics $\boldsymbol{R}$ and $\boldsymbol{P}$ were known. For this reason, the LMS algorithm is sometimes called a "stochastic gradient" algorithm.

As an example, Fig. 8.6 compares the coefficient tracks produced by the LMS algorithm (irregular black curves) with the corresponding gradient-descent tracks (smooth red curves) for the two performance surfaces with correlations of and between the two

coefficients. As can be seen, the actual tracks show a degree of randomness and irregularity because the calculations are based on instantaneous observations rather than the corresponding mathematically-correct statistical averages. Nevertheless, it can also be seen that the tracks converge to the correct values, despite the fact that the journey is more irregular. This occurs because calculating the coefficient values iteratively provides parameter estimates that are averaged over time, which provides reasonable accuracy

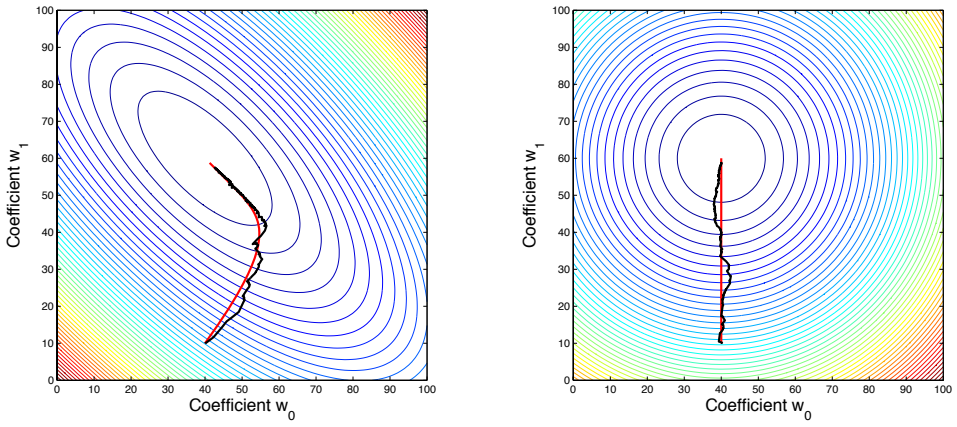


Figure 8.6: Comparison of tracks of steepest descent (smooth red curves) from initial coefficient values of 40 and 10 to final coefficient values of 40 and 60 with actual LMS coefficient tracks (irregular black curves). Values of correlation between the components of the observations are 0.7 (left panel) and 0.0 (right panel) as in the two previous figures.

The LMS algorithm is remarkably simple and quite effective. Nevertheless, it does suffer from potentially slow convergence when the components of the observation vector are highly correlated with one another as in the left panel of Fig. 8.6. As we have discussed, the LMS algorithm causes the tracks of the filter coefficients to follow (on average) trajectories that are perpendicular to the contour lines, which is not the most direct path to the bottom of the bowl when the observations are correlated. This is of particular concern when the FIR filter structure is used for the adaptive filter in this discussion, because in this case the observation components are merely successive delayed values of the original input signal.

Several approaches to the original LMS formulation using FIR adaptive filters have been proposed to ameliorate the problem of slow convergence when the observations are correlated. These include:

1. **The use of alternate adaptation algorithms**. Other coefficient-update algorithms have been proposed that converge rapidly than the LMS algorithm when observations are correlated, at the expense of increased computational complexity. The most widely-used such approach is the *recursive least-squares* (or RLS) algorithm. These algorithms attempt to decorrelate the observations by estimating iteratively and applying the inverse of the correlation matrix.

2. **Alternate filter structures**. There are other filter structures, such as the *FIR lattice structure*, which provide intrinsic decorrelation of the observations.

3. **Frequency-domain approaches**. The convolution performed by the adaptive FIR filter can also be performed in the frequency domain using fast Fourier transform techniques. Performing the adaptation in the frequency domain as well has the further advantage that the Fourier transform operation is intrinsically orthogonalizing, and hence will decorrelate the observations naturally.

4. **Subband approaches**. Adapting separate frequency bands in the frequency domain in parallel also reduces the correlation in the observations because the power spectrum in local regions tends to be flatter (which implies reduced eigenvalue spread) than the entire power spectrum.

We will discuss the RLS and lattice implementations in Secs. 8.7 and 8.8 below.

8.7 The recursive least squares (RLS) adaptation algorithm

As noted above, the LMS algorithm is widely used but it is subject to slow convergence because its coefficient trajectories follow the path of steepest descent, which can be quite indirect if the observations $\{\boldsymbol{X}_k\}$ are highly correlated with one another. Recall that the original iterative gradient descent algorithm derived directly from the Newton solution, Eq. (8.18), was

$$\boldsymbol{W}_{k+1} = \boldsymbol{W}_k - 2\mu\boldsymbol{R}^{-1}\nabla_k = \boldsymbol{W}_k + 2\mu e_k \boldsymbol{R}_k^{-1}\boldsymbol{X}_k \tag{8.34}$$

The contribution of the matrix $\boldsymbol{R}_k^{-1}$ in Eq. (8.17) is to decorrelate or "whiten" the observations, and indeed it is the removal of $\boldsymbol{R}^{-1}$ while formulating the LMS algorithm, that causes the coefficient trajectories to move indirectly to the minimum error by following the perpendicular to the contour lines, rather than moving directly to the optimal coefficient values regardless of whether the eigenvalue spread is small or large.

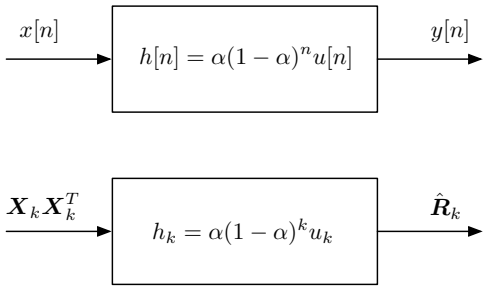


Figure 8.7: Upper panel: conventional LSI filter that implements exponential smoothing. Lower panel: implementation that realizes the estimated correlation matrix $\hat{\boldsymbol{R}}_k$.

The recursive least squares (RLS) algorithm, which is also referred to in the Stearns chapter in Lim and Oppenheim as the "LMS-Newton" algorithm, produces a coefficient trajectory that moves directly to the optimal coefficients $\boldsymbol{W}^*$ by iteratively estimating the matrix $\boldsymbol{R}^{-1}$ and the stochastic gradient simultaneously. We can approach the iterative estimation of $\boldsymbol{R}_k$ by first considering the adaptive estimation of a parameter by exponentially-weighted smoothing. For example, consider the very simple one-dimensional LSI system depicted in the upper panel of Fig. 8.7, which has the unit sample response $h[n] =$

$\alpha(1-\alpha)^n u[n]$ where $0 < \alpha < 1$. It is easy to show that the difference equation that relates the output to the input is

$$y[n] = (1-\alpha)y[n-1] + \alpha x[n] \tag{8.35}$$

The lower panel of Fig. 8.7 depicts a similar LSI system with the sample response, but using the notation of the Stearns chapter in Lim and Oppenheim. This filter operates element by element on its input, which is the $L \times L$ matrix $\boldsymbol{X}_k \boldsymbol{X}_k^T$, producing as its output $\boldsymbol{R}_k$, which is a running exponential estimate of the autocorrelation matrix $\boldsymbol{R} = \mathbb{E}[\boldsymbol{X}_k \boldsymbol{X}_k^T]$. The corresponding difference equation relating output to input in the lower panel is

$$\hat{\boldsymbol{R}}_k = (1-\alpha)\hat{\boldsymbol{R}}_{k-1} + \alpha \boldsymbol{X}_k \boldsymbol{X}_k^T \tag{8.36}$$

Let us now premultiply Eq. (8.36) by $\hat{\boldsymbol{R}}_k^{-1}$ and postmultiply it by $\hat{\boldsymbol{R}}_{k-1}^{-1}$:

$$\hat{\boldsymbol{R}}_k^{-1} \hat{\boldsymbol{R}}_k \hat{\boldsymbol{R}}_{k-1}^{-1} = (1-\alpha)\hat{\boldsymbol{R}}_k^{-1} \hat{\boldsymbol{R}}_{k-1} \hat{\boldsymbol{R}}_{k-1}^{-1} + \alpha \hat{\boldsymbol{R}}_k^{-1} \boldsymbol{X}_k \boldsymbol{X}_k^T \hat{\boldsymbol{R}}_{k-1}^{-1} \tag{8.37}$$

Cleaning up Eq. (8.37) produces

$$\hat{\boldsymbol{R}}_{k-1}^{-1} = (1-\alpha)\hat{\boldsymbol{R}}_k^{-1} + \alpha \hat{\boldsymbol{R}}_k^{-1} \boldsymbol{X}_k \boldsymbol{X}_k^T \hat{\boldsymbol{R}}_{k-1}^{-1} \tag{8.38}$$

Postmultiplying Eq. (8.38) by $\boldsymbol{X}_k$ produces

$$\hat{\boldsymbol{R}}_{k-1}^{-1} \boldsymbol{X}_k = (1-\alpha)\hat{\boldsymbol{R}}_k^{-1} \boldsymbol{X}_k + \alpha \hat{\boldsymbol{R}}_k^{-1} \boldsymbol{X}_k \boldsymbol{X}_k^T \hat{\boldsymbol{R}}_{k-1}^{-1} \boldsymbol{X}_k \tag{8.39}$$

or

$$\text{or } \hat{\boldsymbol{R}}_{k-1}^{-1} \boldsymbol{X}_k = \hat{\boldsymbol{R}}_k^{-1} \boldsymbol{X}_k [(1-\alpha) + \alpha \boldsymbol{X}_k^T \hat{\boldsymbol{R}}_{k-1}^{-1} \boldsymbol{X}_k] \tag{8.40}$$

Now, let $\hat{\boldsymbol{S}}_k = \hat{\boldsymbol{R}}_{k-1}^{-1} \boldsymbol{X}_k$. We can then write trivially

$$\hat{\boldsymbol{S}}_k = \hat{\boldsymbol{R}}_k^{-1} \boldsymbol{X}_k [(1-\alpha) + \alpha \boldsymbol{X}_k^T \hat{\boldsymbol{S}}_k] \tag{8.41}$$

Because the matrices $\boldsymbol{R}_k$ and $\boldsymbol{R}_k^{-1}$ are symmetric, we can write $\hat{\boldsymbol{S}}_k^T = \boldsymbol{X}_k^T \hat{\boldsymbol{R}}_{k-1}^{-1}$. Dividing both sides of Eq. (8.41) by $[1-\alpha+\alpha \boldsymbol{X}_k^T \hat{\boldsymbol{S}}_k]$ and postmultiplying by $\hat{\boldsymbol{S}}_k^T = \boldsymbol{X}_k^T \hat{\boldsymbol{R}}_{k-1}^{-1}$ produces

$$\hat{\boldsymbol{R}}_k^{-1} \boldsymbol{X}_k \boldsymbol{X}_k^T \hat{\boldsymbol{R}}_{k-1}^{-1} = \frac{\hat{\boldsymbol{S}}_k \hat{\boldsymbol{S}}_k^T}{1-\alpha+\alpha \boldsymbol{X}_k^T \hat{\boldsymbol{S}}_k} \tag{8.42}$$

where again

$$\hat{\boldsymbol{S}}_k = \hat{\boldsymbol{R}}_{k-1}^{-1} \boldsymbol{X}_k$$

Now ... substituting (8.42) into (8.38) produces

$$\hat{\boldsymbol{R}}_{k-1}^{-1} = (1-\alpha)\hat{\boldsymbol{R}}_k^{-1} + \alpha \frac{\hat{\boldsymbol{S}}_k \hat{\boldsymbol{S}}_k^T}{1-\alpha+\alpha \boldsymbol{X}_k^T \hat{\boldsymbol{S}}_k}$$

which is easily rewritten as

$$\hat{\boldsymbol{R}}_k^{-1} = \frac{1}{(1-\alpha)} \left(\hat{\boldsymbol{R}}_{k-1}^{-1} - \alpha \frac{\hat{\boldsymbol{S}}_k \hat{\boldsymbol{S}}_k^T}{1-\alpha+\alpha \boldsymbol{X}_k^T \hat{\boldsymbol{S}}_k} \right) \tag{8.43}$$

This equation is (finally!) the update equation for $\hat{\boldsymbol{R}}_k^{-1}$ that has been the missing piece in the RLS algorithm. Please note that the only new concept is the recursive estimation of $\hat{\boldsymbol{R}}_k$ in Eq. (8.36). Everything after that was just a series of algebraic operations that enabled us to convert the iterative estimate of $\hat{\boldsymbol{R}}_k$ into an iterative estimate of $\hat{\boldsymbol{R}}_k^{-1}$. We can now write the complete RLS update equations as:

1. Let $\hat{\boldsymbol{R}}_0^{-1} = \boldsymbol{R}^{-1}$ if $\boldsymbol{R}$ is known or $\frac{1}{\sigma^2}\boldsymbol{I}$ if it is not, where σ^2 is the variance of the input signal x_k.
2. Let $\hat{\boldsymbol{S}}_k = \hat{\boldsymbol{R}}_{k-1}^{-1}\boldsymbol{X}_k$.
3. Update the estimated inverse covariance matrix:

$$\hat{\boldsymbol{R}}_k^{-1} = \frac{1}{(1-\alpha)}\left(\hat{\boldsymbol{R}}_{k-1}^{-1} - \alpha\frac{\hat{\boldsymbol{S}}_k\hat{\boldsymbol{S}}_k^T}{1-\alpha+\alpha\boldsymbol{X}_k^T\hat{\boldsymbol{S}}_k}\right)$$

4. Calculate the error function from the filter output: $e_k = d_k - \boldsymbol{X}_k^T\boldsymbol{W}_k$
5. Update the coefficients: $\boldsymbol{W}_{k+1} = \boldsymbol{W}_k + 2\mu e_k\hat{\boldsymbol{R}}_k^{-1}\boldsymbol{X}_k$

We iterate Steps 2-5 in the normal update of the filter coefficients. As you will recall, the LMS algorithm in contrast is simply

$$\boldsymbol{W}_{k+1} = \boldsymbol{W}_k + 2\mu e_k\boldsymbol{X}_k$$

As discussed in class, the RLS algorithm converges faster than the LMS algorithm when the performance surface is elongated in shape, which happens when the components of $\boldsymbol{X}_k$ are correlated or (alternatively) the matrix $\boldsymbol{R}$ has large eigenvalue spread.

8.8 The adaptive lattice algorithm

The adaptive lattice filter is an alternate way to develop fast adaptation by exploiting the decorrelation of the signals to subsequent stages of the adaptive lattice. Of course we have already discussed lattice filters in Secs. 7.3 and 7.4 in our discussion of linear prediction, so we will not repeat most of the basics other than to point out notational changes. The process of obtaining the adaptive algorithm parallels the development of the LMS algorithm, so it should be easy to understand.

Let us begin by comparing the notational conventions for the lattice structure from our discussion of linear prediction with the notational conventions used by Stearns in Fig. 8.8 below. The figure above compares the notational conventions used to describe the (fixed) FIR lattice filter in the text by Rabiner and Schafer and also in Secs. 7.3 and 7.4 and the (adaptive) FIR lattice filter developed in Chapter 5 in the Lim and Oppenheim text, which we will adopt in this chapter. It is important to keep in mind that the coefficients k_i in the adaptive lattice filter are actually time varying (*i.e.*, $k_i = k_i[k]$). Consequently we must take some care not to confuse the (unsubscripted) symbol k that indicates the time index with the (subscripted) $k_i[k]$, which are the time-varying reflection coefficients. Also note the change in letter symbol to denote the forward error, the reversal in sign in the reflection coefficients, and the change in indexing of the reflection coefficients. The lattice update equations using the notational conventions of Stearns are:

$$f_{i+1}[k] = f_i[k] + k_i b_i[k-1] \tag{8.44}$$
$$b_{i+1}[k] = k_i f_i[k] + b_i[k-1] \tag{8.45}$$

Let us consider the MSE of the forward error signal:

$$\xi_f^2 = \mathbb{E}[f_{i+1}^2[k]] = \mathbb{E}[(f_i[k] + k_i b_i[k-1])^2] \tag{8.46}$$

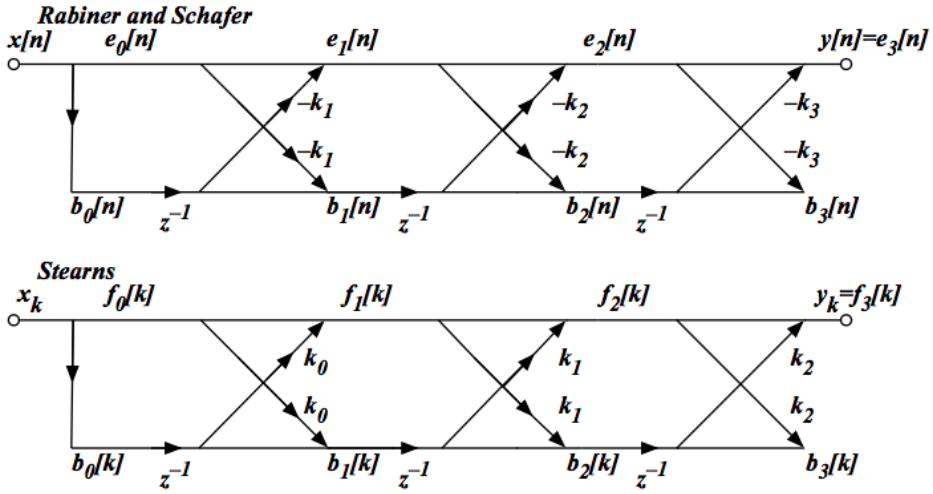


Figure 8.8: Comparison of fixed lattice filters from Rabiner and Schafer with adaptive lattice filters of Stearns.

As you will recall, the free parameters of the lattice filter are the reflection coefficients $\{k_i[k]\}$. Proceeding as before, we will first obtain the optimal solution for the error assuming that statistics are known by differentiating the MSE with respect to the parameter k_i, setting the derivative to zero, and solving for k_i.

$$\frac{\partial \xi_f^2}{\partial k_i} = \mathbb{E}[2(f_i[k] + k_i b_i[k-1])b_i[k-1]] = 0 \tag{8.47}$$

Solving for k_i:

$$\mathbb{E}[f_i[k]b_i[k-1]] = k_i \mathbb{E}[b_i^2[k-1]] \tag{8.48}$$

$$k_i^* = \frac{-\mathbb{E}[f_i[k]b_i[k-1]]}{\mathbb{E}[b_i^2[k-1]]} \tag{8.49}$$

This solution, of course, is exactly the same as the PARCOR method of estimating reflection coefficients that we have seen before in our discussion of linear prediction. The only differences (including the sign change) are a consequence of the changes in notation.

Now let us consider the correlation across subsequent stages when the optimal value of k_i is used. We will consider the forward errors:

$$\mathbb{E}[f_{i+1}[k]f_i[k]] = \mathbb{E}[(f_i[k] + k_i b_i[k-1])f_i[k]] \tag{8.50}$$

Plugging in the optimal value of k_i^* we observe

$$\mathbb{E}[f_{i+1}[k]f_i[k]] = \mathbb{E}\left[\left(f_i[k] - \frac{f_i[k]b_i[k-1]b_i[k-1]}{b_i^2[k-1]}\right)f_i[k]\right] = 0 \tag{8.51}$$

In other words, the forward error stages are uncorrelated when the coefficients are fully adapted. The same is true for the backward error stages.

Now let us consider the LMS update equations for the $k_i[k]$ coefficients. As before, we can write

$$k_i[k+1] = k_i[k] - \mu_i \nabla \tag{8.52}$$

where in this case

$$\nabla = \frac{\partial \xi_f^2}{\partial k_i} = \mathbb{E}[2(f_i[k] + k_i b_i[k-1])b_i[k-1]] \tag{8.53}$$

Note that we are subscripting the stepsize parameter μ_i so that it may vary in magnitude from stage to stage if desired. Replacing the expected value of the right side of Eq. (8.53) by its instantaneous value and substituting back in to Eq. (8.52), we obtain

$$k_i[k+1] = k_i[k] - \mu_i[2(f_i[k] + k_i[k]b_i[k-1])b_i[k-1]] \tag{8.54}$$

Note that the expression inside the parenthesis above is just $f_{i+1}[k]$, so we obtain for the LMS lattice update equation the very simple expression

$$k_i[k+1] = k_i[k] - 2\mu_i f_{i+1}[k]b_i[k-1] \tag{8.55}$$

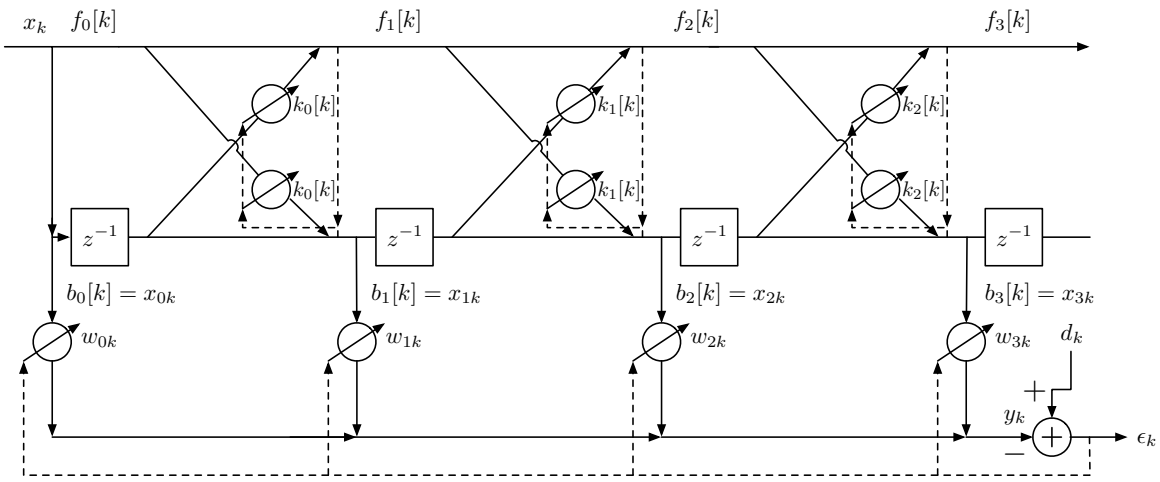
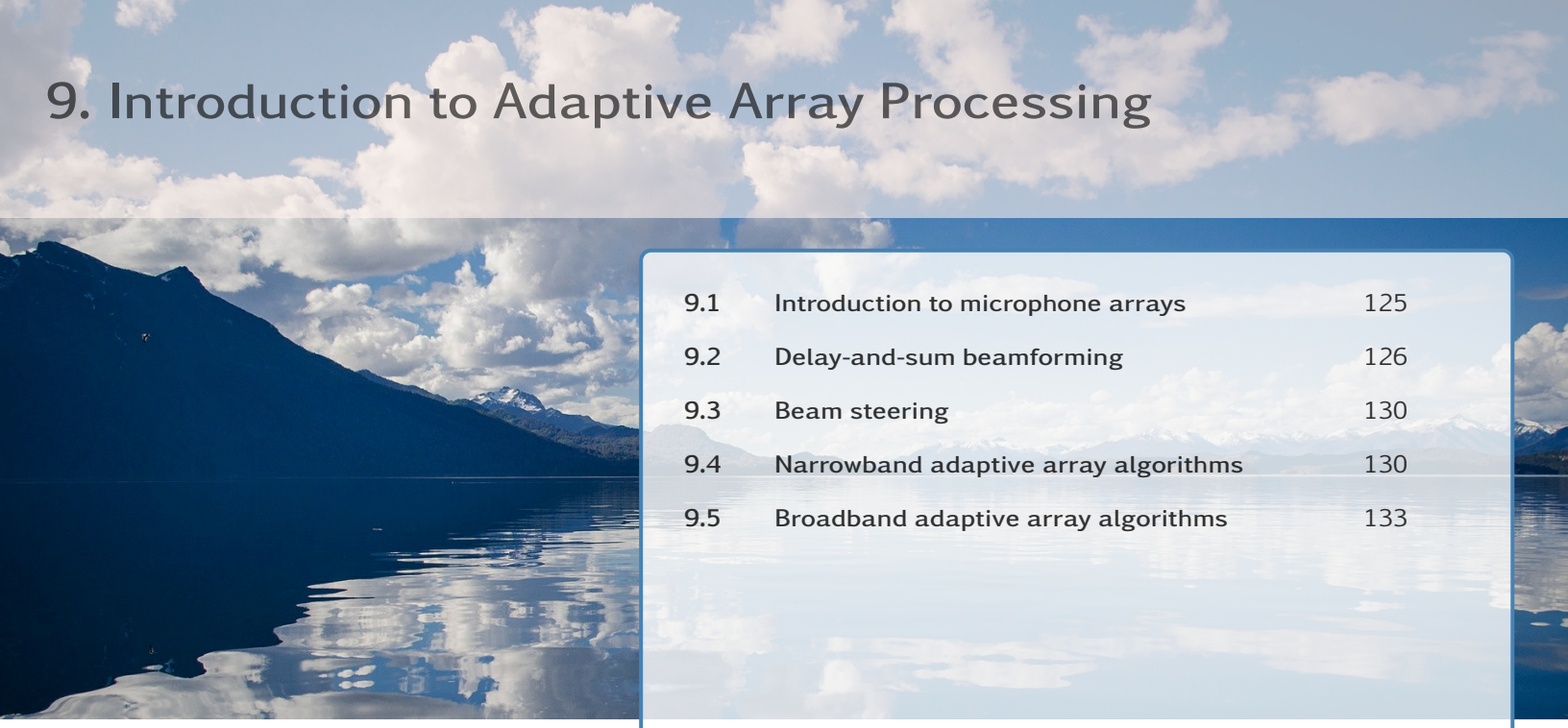


Figure 8.9: The adaptive lattice filter used as a preprocessor that decorrelates the observations for the conventional adaptive FIR filter.

The adaptive lattice algorithm can achieve faster convergence than the original LMS algorithm because the observations at successive stages are uncorrelated once the coefficients become sufficiently adapted. One common application of the adaptive lattice is as a preprocessor for the conventional adaptive FIR filter. As an example, Fig. 8.9 describes this configuration. The lower half of the system in Fig. 8.9 is a conventional FIR filter with the coefficients updated according to the LMS algorithm. The upper half of the figure is an adaptive lattice filter, with the coefficients updated according to the LMS-like update method described in Eq. (8.55). While the adaptive FIR filter using the LMS algorithm would normally converge slowly because the observations tend to be mutually correlated, especially when the input is spectrally concentrated, the combined lattice-FIR structure converges quickly because the observations of the FIR filter are the backward errors at each stage of the lattice, which are mutually uncorrelated as discussed above in conjunction with Eq. (8.51).

9. Introduction to Adaptive Array Processing



Adaptive array systems are able to develop a response that favors signals arriving from a desired "look" direction over signals arriving from other directions. This chapter discusses the principles of adaptive arrays and introduces some of the basic algorithms used to implement them.

9.1 Introduction to microphone arrays

The use of sensor arrays enables us to develop systems that exhibit greater response from waves coming from one direction than from other directionss. In general we seek to maintain a constant undistorted response from the direction of the desired signal (sometimes called the "look" direction), while suppressing to the extent possible the response from signals from other directions, which could be simple background noise or malevolent interfering signals. While in principle many types of sensors can be used, such as the elements of antennas used in broadcast television and other types of communication antennas, vibration sensors for earthquake detection and other types of seismic analysis, etc., we are primarily concerned with acoustical applications in speech and audio processing using omnidirectional microphones as the sensors.

In this chapter we first review the basic properties of the simplest type of array, the delay-and-sum beamformer. Although delay-and-sum beamforming performs only modestly (it typically is used as a baseline to which the behavior of better-performing systems can be compared), understanding the physics that underly its performance will facilitate a more general understanding of sensor arrays. We then continue by reviewing the design and function of selected classic array-processing algorithms.[1]

[1] This material on adaptive array algorithms is in part a condensation of material from Chapter 13 on narrowband arrays and Chapter 14 on broadband arrays in the text *Adaptive Signal Processing* by Widrow and Stearns, published by Prentice-Hall in 1986.

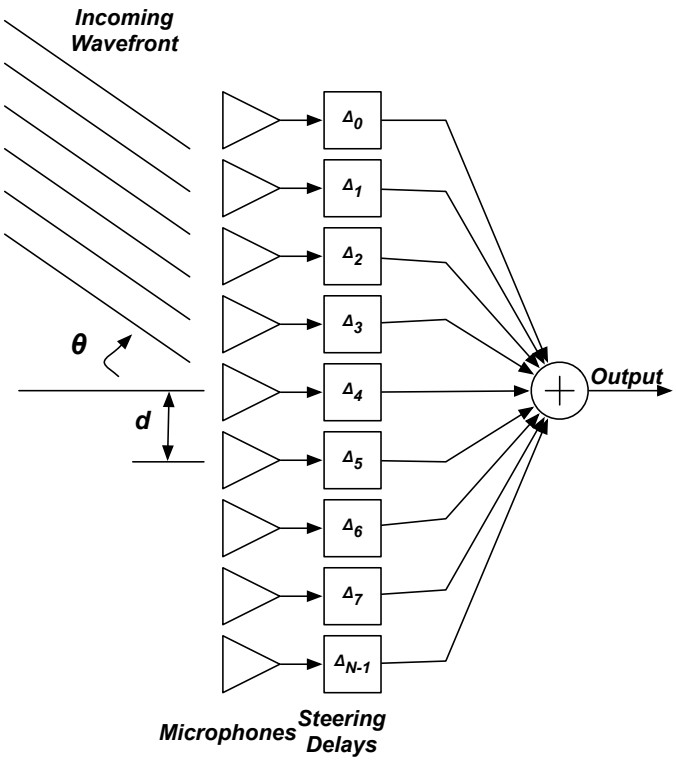


Figure 9.1: Basic block diagram of a delay-and-sum array

9.2 Delay-and-sum beamforming

Consider the system depicted in Fig. 9.1. Let us assume a set of N sensors ($N = 9$ in the Fig. 9.1) in a straight line separated by a distance of d meters. The outputs of the sensors are passed through a set of *steering delays* Δ_k and then added together. An incoming plane wave of sound is arriving from the upper left corner of the figure, at an angle θ relative to a line that passes through the microphones.

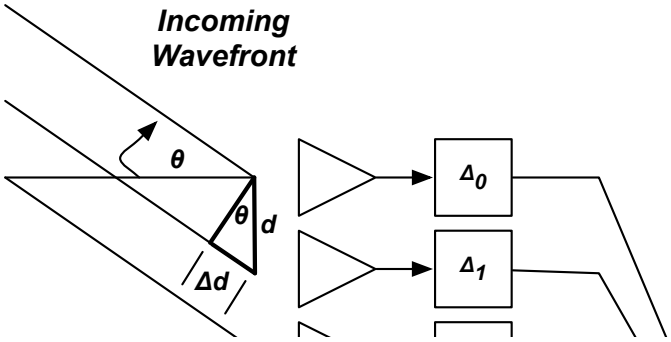


Figure 9.2: Detailed view of array geometry.

Figure 9.2 above is a more detailed view of the upper area of the array. Note that if the arriving wavefront is planar and the wavefront arrives at the angle θ indicated, the dis-

tance from the source to Microphone 1 is $\Delta d = d\sin(\theta)$ meters greater than the distance to Microphone 0. The corresponding difference in arrival time is $\Delta t = \Delta d/c = d\sin(\theta)/c$. The constant c represents the speed of sound, nominally 340 m/s for standard temperature and humidity.

Let us suppose that the input to the system is $x(t) = \cos(\omega t)$. If all the steering delays Δ_k are zero, the output will be

$$y(t) = \sum_{k=0}^{N-1} \cos(\omega(t - k\Delta t)) = \sum_{k=0}^{N-1} Re[e^{j(\omega(t-k\Delta t))}] \tag{9.1}$$

Further expansion of this expression produces:

$$y(t) = \sum_{k=0}^{N-1} Re[e^{j\omega(t-k\Delta t)}] = Re\left[\sum_{k=0}^{N-1} e^{j\omega(t-k\Delta t)}\right] = Re\left[e^{j\omega t}\sum_{k=0}^{N-1} e^{-j\omega k\Delta t}\right] \tag{9.2}$$

Note that the above expression is of the form $Re[Ae^{j\omega t}]$ where the complex number A is referred to as the *complex amplitude* or *phasor*. Let the complex amplitude A be equal to $A = |A|e^{j\theta}$. In this case we have

$$Re[Ae^{j\omega t}] = Re[|A|e^{j\theta}e^{j\omega t}] = Re[|A|e^{j(\omega t+\theta)}] = |A|cos(\omega t + \theta) \tag{9.3}$$

In other words, the magnitude of A is the amplitude of the resulting cosine while the phase of A is the phase of the cosine. At the moment we are primarily concerned with the magnitude of A.

Continuing our analysis of the phasor of Eq. (9.2) we observe that

$$\begin{aligned} A = \sum_{k=0}^{N-1} e^{-j\omega k\Delta t} &= \sum_{k=0}^{N-1} e^{(-j\omega\Delta t)^k} = \frac{(1 - e^{-j\omega N\Delta t})}{(1 - e^{-j\omega\Delta t})} \\ &= e^{-j\omega\Delta t\frac{N-1}{2}} \frac{e^{\frac{j\omega N\Delta t}{2}} - e^{-\frac{j\omega N\Delta t}{2}}}{e^{\frac{j\omega\Delta t}{2}} - e^{-\frac{j\omega\Delta t}{2}}} \\ &= e^{-j\omega\Delta T\frac{N-1}{2}} \frac{-2j\sin(\frac{N\omega\Delta t}{2})}{-2j\sin(\frac{\omega\Delta t}{2})} \end{aligned} \tag{9.4}$$

The first factor is a pure phase term, leaving as the magnitude of the complex amplitude A the expression

$$|A| = \frac{\sin(\frac{N\omega\Delta t}{2})}{\sin(\frac{\omega\Delta t}{2})} = \frac{\sin(\frac{N\omega d\sin(\theta)}{2c})}{\sin(\frac{\omega d\sin(\theta)}{2c})} = \frac{\sin(\frac{N\pi d\sin(\theta)}{\lambda})}{\sin(\frac{\pi d\sin(\theta)}{\lambda})} \tag{9.5}$$

The latter expression was obtained using the relationship $c = \omega\lambda/2\pi$ from physics. The expression for $|A|$ is readily recognized as a form of the familiar $\sin(Nx)/\sin(x)$ expression that we frequently encounter in discrete-time spectra. We plot $|A(\theta)|$ in Fig. 9.3 as a function of θ for $d = 8.5$ cm, $N = 9$, and $v = \omega/2\pi = 500$ Hz. We show the plots using both the conventional cartesian coordinate system and the polar coordinate system (which is

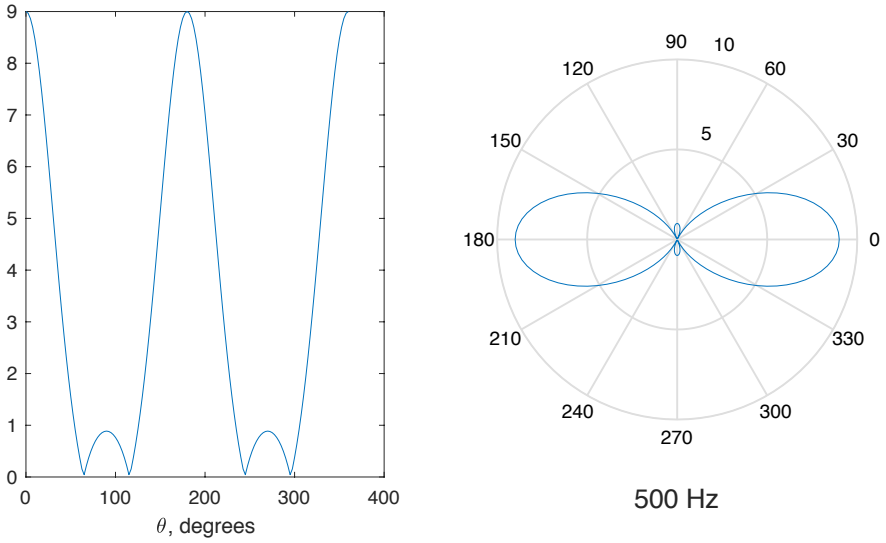


Figure 9.3: Comparison of directional sensitivity of a simple linear array in rectangular coordinates (left panel) and polar coordinates (right panel).

most commonly used for directional sensitivity curves), plotting $|A(\theta)|$ as a function of θ in the latter case.

Figure 9.4 below compares polar plots as a function of frequency for frequencies of 500 to 4000 Hz in steps of 500 Hz. Again, we use the parameters $d = 8.5$ cm and $N = 9$. We show only the half-circle on the right side (*i.e.* arrival angles between -90 and 90 degrees); the left side is identical because of symmetry.

It is worth noting the following attributes of the curves shown in Fig. 9.4:

- The beampatterns change as a function of frequency. This occurs because the actual beam is formed by interference and cancellation involving interactions between the distance betwSpecifically, In particular it can be seen that as frequency increases, the width of the main lobe of the directivity pattern becomes increasingly narrow. This is usually undesirable because it means that as a target moves slightly off-axis, the frequency response in response to it will change, with high frequencies becoming increasingly attenuated.

- Significant sidelobes appear at the frequency $v = 4000$ Hz for the values of N and d under consideration. This is because at that frequency the response begins to be affected by *spatial aliasing*. Conventional temporal aliasing occurs when the incoming signal that is sampled has frequency components that exceed half the sampling rate, the Nyquist frequency. Stated another way, to avoid temporal aliasing (which causes high-frequency components to appear at the output as lower-frequency components) we must be assured that the sampling is "dense" enough so that at least two samples occur within a single period of the highest frequency component of the signal. With spatial aliasing, in contrast, we must ensure that the spatial sampling is dense enough so that the sensors are not separated by a distance of more than *half*

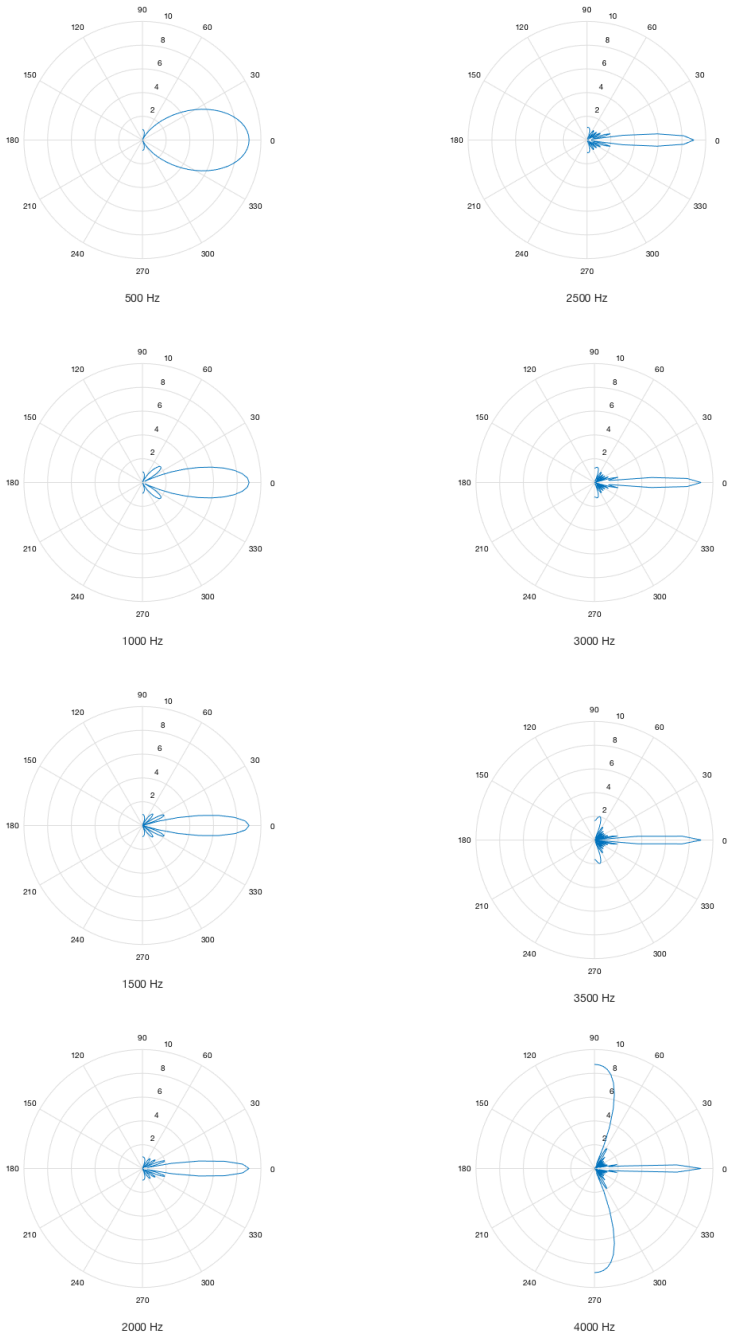


Figure 9.4: Comparison of delay-and-sum polar plots as a function of source frequency. Only responses from the right side ($|\theta| \leq 90$ degrees) are depicted.

a wavelength along the direction of arrival. In other words, to avoid spatial aliasing, we must ensure that

$$d\sin(\theta) < \frac{\lambda}{2} = \frac{\pi c}{\omega} \text{ or } v < \frac{c}{2d\sin(\theta)} \text{ where } v \text{ is the frequency in Hz.} \tag{9.6}$$

While the temporal aliasing constraint is expressed only in terms of frequency, the spatial aliasing constraint depends on frequency ω, sensor distance d, and the arrival angle θ. If the spatial aliasing constraint is violated, signals arriving from a direction toward the side will appear as if they arrived from a location closer to the center (or the look direction in this case).

- While we derived these results using a continuous-time input, the result is the same for a discrete-time input if we take into account the effects of temporal sampling.

9.3 Beam steering

The calculations and examples above were presented as if the desired direction of the beam (or the "look direction") were zero degrees, the perpendicular to the line of the sensors. To rotate the beam so that the maximum response is toward a different direction, it is only necessary to manipulate the steering delays so that they cancel the time difference of arrival when the signal is from that particular direction, such that the signals from each microphone representing a source emanating from the look direction are time aligned after the delays. For example, if we define the coordinates as in Fig. 9.1 and the arrival angle is 30 degrees, with a separation of 8.5 cm, we would obtain $\Delta t = d\sin(\theta)/c = 125\mu s$, and the corresponding steering delays would be

$$\Delta_k = (N-1-k)\Delta t \text{ for } 0 \leq k \leq N-1 \tag{9.7}$$

While it was noted above that the beampatterns are the same for continuous time and discrete time in principle, the steering delay increments Δ_k are real numbers that would in general be affected by rounding to the nearest integer. For example, a sampling frequency of 16 kHz implies a sampling period of 62.5 μs. The steering delay of 125 μs is implemented easily for this sampling frequency as it is exactly two samples, but this is just coincidental for this particular arrival angle, and in practice quantization of the steering delays can be a serious problem. One solution to this problem is to increase the sampling rate of the input to the steering delays by upsampling and then downsampling the output by the same rate factor, effectively providing the opportunity to obtain fractional delays at the original sampling frequency as discussed in Sec. 1.4.

9.4 Narrowband adaptive array algorithms

We now turn our attention to some of the ways in which practical adaptive array systems are implemented. In this section we introduce some of the basic principles that govern how narrowband signals are processed. We generalize this discussion to systems that process broadband signals in Sec. 9.5.

Let us begin by considering the simple two-element narrowband array depicted in Fig. 9.5. The primary and reference channels are assumed to receive plane waves arriving at a propagation angle of θ. Internal noise from the sensors and other sources is represented by n_{1k} and n_2k, which are assumed to be white and statistically independent of each other. The system with input x_k and output y_k effectively functions as an ordinary filter that can produce a transfer function with any desired magnitude and phase, but only at a single frequency.

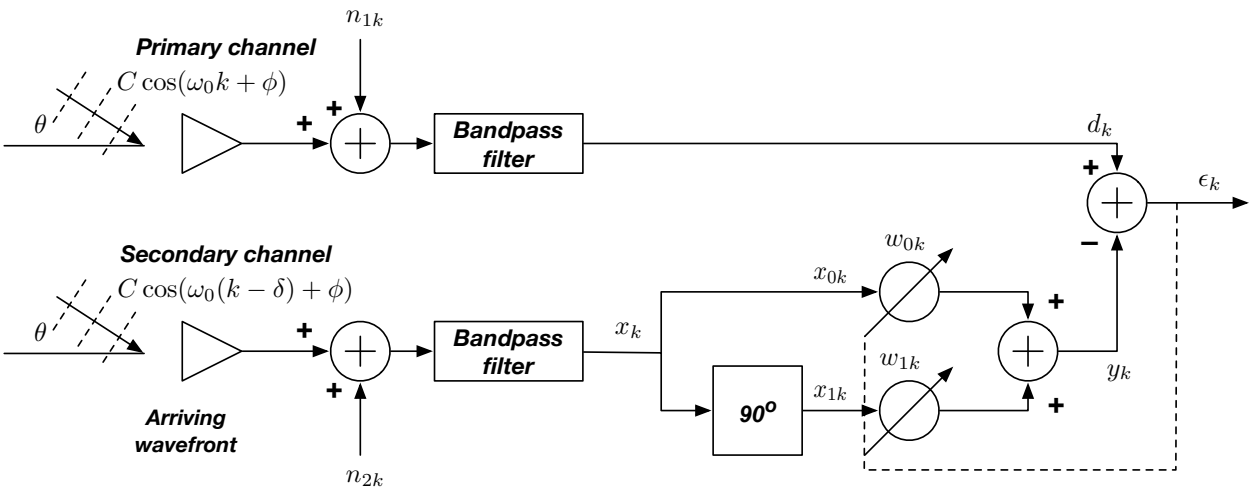


Figure 9.5: Block diagram of a narrowband array processing system.

We will assume for now that the input to the primary channel is simply $s_k = C\cos(\omega_0 k+\phi)$, where ϕ represents an unknown uniformly-distributed absolute phase shift. Based on the physical considerations discussed in Sec. 9.2 and illustrated in Fig. 9.2, the input to the secondary or reference channel based on the would be $C\cos(\omega_0(k+\delta)+\phi)$, where the time delay is represented by

$$\delta = \frac{d\sin(\theta)}{cT} = \frac{2\pi d\sin(\theta)}{\lambda_0\omega_0 T} \tag{9.8}$$

The delay δ is expressed in terms of the number of samples where T is the time between samples (or the sampling period). As usual, ω_0 represents the frequency of s_k, λ_0 is the corresponding wavelength at propagation speed c, and d is the spacing between the sensors. The bandpass filter is assumed to be an ideal narrowband filter that includes the signal frequency ω_0. As in the previous chapter, the output of the of the adaptive filter is

$$y_k = \sum_{l=0}^{1} x_{lk}w_{lk} = \boldsymbol{X}_k^T\boldsymbol{W}_k = \boldsymbol{W}_k^T\boldsymbol{X}_k \tag{9.9}$$

where

$$\boldsymbol{W}_k = \begin{pmatrix} w_{0k} \\ w_{1k} \end{pmatrix} \text{ and } \boldsymbol{X}_k = \begin{pmatrix} x_{0k} \\ x_{1k} \end{pmatrix} \tag{9.10}$$

where the elements of $\boldsymbol{X}_k$ are the inputs to the weights in $\boldsymbol{W}_k$.

As before, the optimal coefficient values are determined by the Wiener-Hopf equation $\boldsymbol{W} = \boldsymbol{R}^{-1}\boldsymbol{P}$ where

$$\boldsymbol{R} = \mathbb{E}[\boldsymbol{X}\boldsymbol{X}^T] = \begin{pmatrix} \overline{x_{0k}x_{0k}} & \overline{x_{0k}x_{1k}} \\ \overline{x_{1k}x_{0k}} & \overline{x_{1k}x_{1k}} \end{pmatrix} = \begin{pmatrix} \sigma_r^2 + \frac{C^2}{2} & 0 \\ 0 & \sigma_r^2 + \frac{C^2}{2} \end{pmatrix} \tag{9.11}$$

where σ_r^2 represents the variance of the components of the internal noise signals n_{1k} and

n_{2k} that pass through the bandpass filters. Similarly,

$$\boldsymbol{P} = \mathbb{E}[d_k \boldsymbol{X}] = \begin{pmatrix} \overline{d_k x_{0k}} \\ \overline{d_k x_{1k}} \end{pmatrix} = \frac{C^2}{2} \begin{pmatrix} \cos(\delta\omega_0) \\ \sin(\delta\omega_0) \end{pmatrix} \tag{9.12}$$

Hence, we can obtain the optimal coefficients using the Wiener-Hopf equation:

$$\boldsymbol{W}^* = \begin{pmatrix} w_0^* \\ w_1^* \end{pmatrix} = \boldsymbol{R^{-1}P} = \frac{C^2}{2} \begin{pmatrix} \frac{2}{2\sigma_r^2 + C^2} & 0 \\ 0 & \frac{2}{2\sigma_r^2 + C^2} \end{pmatrix} \begin{pmatrix} \cos(\delta\omega_0) \\ \sin(\delta\omega_0) \end{pmatrix} = \frac{C^2}{2\sigma_r^2 + C^2} \begin{pmatrix} \cos(\delta\omega_0) \\ \sin(\delta\omega_0) \end{pmatrix} \tag{9.13}$$

If $\theta = 0$, δ will be zero as well and we will obtain

$$\boldsymbol{W}^* = \begin{pmatrix} \frac{C^2}{2\sigma_r^2 + C^2} \\ 0 \end{pmatrix} \tag{9.14}$$

These results were obtained by assuming that the arriving signal is a WSS random-phase cosine wave, with the phase parameter being independent (as usual) of the statistics characterizing the internal noise sources.

In general, the 90° phase shift in the lower branch of the reference channel combined with the weights w_0 and w_1 of the adaptive filter means that a cancelling signal in the reference channel can be developed with arbitrary magnitude and phase at a single frequency. In the original Howells-Appelbaum configuration shown in Fig. 9.5, this means that a single interfering source can be cancelled in the reference channel. If only a single source is present (as shown in Fig. 9.5), the signal source itself would be cancelled if its power is large compared to that of the components of the internal noise sources that are passed through the bandpass filter. On the other hand, if the signal power is small compared to that of the internal noise, the output of the system will be proportional to the input.

In the most common application of this system, the weights of the system are first adapted to a strong signal arriving from the direction of desired cancellation. Once the weights are adapted, the system will pass signals arriving from any *other* direction without significant attenuation, but would severely attenuate signals arriving from the direction of arrival of the signal with which the system was adapted. If it is necessary to cancel sources arriving from more than one direction simultaneously, this can be easily accomplished by adding additional identical reference channels in parallel to the one depicted in Fig. 9.5 with two additional adaptive weights. The outputs of the reference channels are summed to form y_k, which is subtracted from the signal from the primary channel, d_k, as before. The number of interfering signals that can be cancelled is equal to the total number of reference channels. The weights of the system will converge to cancel the $N-1$ interfering sources with the greatest power at the time of adaptation, where N is the total number of sensors.

While this system is designed to process narrowband signals, it can be generalized to handle broadband signals by replacing the adaptive elements of Fig. 9.5 (*i.e.* the two weights and the 90° phase shifter by an adaptive LSI filter. This filter is typically implemented as a simple FIR filter as in Fig. 8.3. The adaptive linear filter in each reference channel, in

effect, enables us in principle to control the magnitude and phase of its outputs at multiple frequencies, specifically the DFT frequencies $\omega_k = 2\pi k/L$ where L is the number of taps in the filter. Increasing the FIR filter length will increase the spectral resolution in the directivity pattern, and increasing the number of reference channels will increase the number of interfering signals that can be cancelled simultaneously.

We discuss several practical approaches to broadband adaptive array processing in the following section.

9.5 Broadband adaptive array algorithms

In Sec. 9.4 above we first developed the concept of the delay-and-sum beamformer and beam steering through the imposition of internal delays that compensate for the delays imposed by differences in the propagation times from the source to the various sensors. In addition, we showed that the weights of an adaptive array processor can be adapted to cancel unwanted signals coming from multiple directions.

The systems described in Sec. 9.4 suffer from two serious limitations. First, they are designed to produce the desired response only at a single frequency (or a narrow band of frequencies). In addition, the actual directional response depends on a number of factors such as the ratios of the power of the desired target signal, the undesired interfering signals, and the level of internal noise. In this section we will describe several practical systems that address these issues. All of these systems have two major design goals:

- We would like the system to maintain a fixed and undistorted response to signals arriving from the direction of the target signal (which is commonly referred to as the "look direction"),
- At the same time we would like the system to eliminate the response from the most powerful interfering sources.

As we noted at the conclusion of Sec. 9.4, the narrowband constraint can be mitigated by replacing the simple phase-shift-and-add processing in the reference channel by a general FIR filter. In addition, the number of interfering sources that can be cancelled is equal to the number of microphones in the reference channel, which is equal to the total number of microphones minus one.

In the sections below we describe three different approaches to the problem of simultaneously guaranteeing an undistorted response in the look direction while cancelling the interfering sources to the extent possible.

9.5.1 The Griffiths LMS algorithm

The Griffiths LMS beamformer is based on the LMS algorithm and exploits the fact that with a typical microphone array system we have more *a priori* information about the relevant statistics for the LMS algorithm than we normally would in the general case. As we discussed in Sec. 8.6, the LMS algorithm can be expressed as

$$\boldsymbol{W}_{k+1} = \boldsymbol{W}_k + 2\mu \boldsymbol{X}_k e_k \tag{9.15}$$

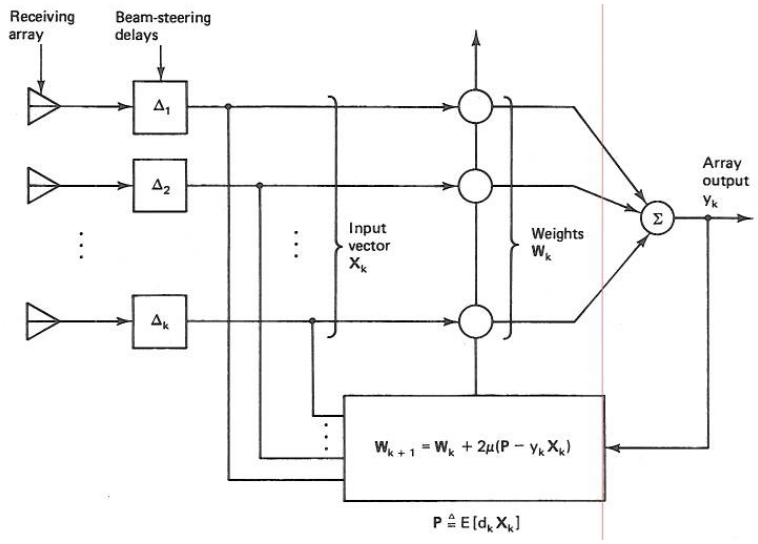


Figure 9.6: Bloick diagram of the Griffiths LMS algorithm. Note the absence of a "desired signal" to the algorithm. From Woodrow and Stearns (1985).

Working this equation more or less in the reverse order from which it was derived, we observe that

$$\boldsymbol{W}_{k+1} = \boldsymbol{W}_k + 2\mu \boldsymbol{X}_k e_k \tag{9.16}$$
$$= \boldsymbol{W}_k + 2\mu \boldsymbol{X}_k (d_k - y_k) \tag{9.17}$$
$$= \boldsymbol{W}_k + 2\mu d_k \boldsymbol{X}_k - 2\mu y_k \boldsymbol{X}_k \tag{9.18}$$

We now replace the instantaneous expression $d_k \boldsymbol{X}_k$ by its expected value $\mathbb{E}[d_k \boldsymbol{X}_k] = \boldsymbol{P}$ While it may be recalled that a key step of the derivation of the LMS algorithm was the replacement of $\boldsymbol{P} = \mathbb{E}[d_k \boldsymbol{X}_k]$ by its instantaneous value, in this particular case we can actually use the ensemble average $\boldsymbol{P}$ because it is determined by the array geometry, the direction of arrival for the target signal, and the autocorrelation function of the target signal, all of which are assumed to be known in this case. (The array geometry and azimuth of the target are generally a part of the system design, while the autocorrelation function of the target is either known *a priori* ore estimated using the techniques discussed in Chapter 5.) The use of the $\boldsymbol{P}$ constraint also causes an input signal arriving from the look direction to be output without distortion.

For this algorithm we define the extended observation vector $\boldsymbol{X_k}$ and the extended weight vector $\boldsymbol{W_k}$ by vertically concatenating the observation and weight vectors of the the K

sensors:

$$\boldsymbol{X_k} = \begin{pmatrix} x_{00k} \\ x_{10k} \\ \vdots \\ x_{L-1,0k} \\ x_{01k} \\ \vdots \\ x_{L-1,1k} \\ \vdots \\ x_{L-1,K-1,k} \end{pmatrix} \qquad \boldsymbol{w_k} = \begin{pmatrix} w_{00k} \\ w_{10k} \\ \vdots \\ w_{L-1,0k} \\ w_{01k} \\ \vdots \\ w_{L-1,1k} \\ \vdots \\ w_{L-1,K-1,k} \end{pmatrix} \tag{9.19}$$

and the associated observation vector $\boldsymbol{X}_k$ to be (as usual) the inputs to $\boldsymbol{W}_k$. Using those definitions we can write the weight update equation for the Griffiths LMS algorithm as

$$\boldsymbol{W}_{k+1} = \boldsymbol{W}_k + 2\mu\boldsymbol{P} - 2\mu y_k \boldsymbol{X}_k \tag{9.20}$$

The resulting system is depicted in block diagram form in Fig. 9.6.

[FIX THIS GAP]

9.5.2 The Frost algorithm

As noted above, the general objectives of the broadband adaptive array algorithms described here are: (1) to maintain an undistorted response with unity gain in the look direction, and (2) to minimize the total amount of energy output from all directions, thereby spprressing to the extent possible the contributions of noise components from other directions of arrival. In the previous section we described the Griffiths LMS algorithm, which achieves these objectives indirectly using "soft" constraints. In this section we describe the *Frost algorithm* which establishes a "hard" constraint to maintain the undistorted response in the look direction. In the following section we describe the *Griffiths-Jim algorithm*, which reformulates the same problem but maintaining the target signal free of distortion through a clever system architecture rather than through iterative adaptive convergence. Finally, in Sec. 9.5.4, we describe the *minimum variance distortionless response (MVDR) algorithm* which in a popular modern algorithm that implements the Griffiths-Jim architecture non-adaptively non-adaptively using the MMSE solution. [MOVE THIS TO THE INTRO?]

Figure 9.7 is a block diagram of the major components that implement the Frost algorithm. The figure depicts a linear array of N sensors, each feeding into an adaptive FIR filter (or tapped delay line), each with $L-1$ delays as in Fig. 8.3. While this organization is similar in principle to that of the Griffiths LMS algorithm, the adaptation algorithms of the two algorithms are different.

Following a modified version of the notation used in Widrow and Stearns, we represent the adaptive weights of the FIR filters using the $L \times N$ matrix

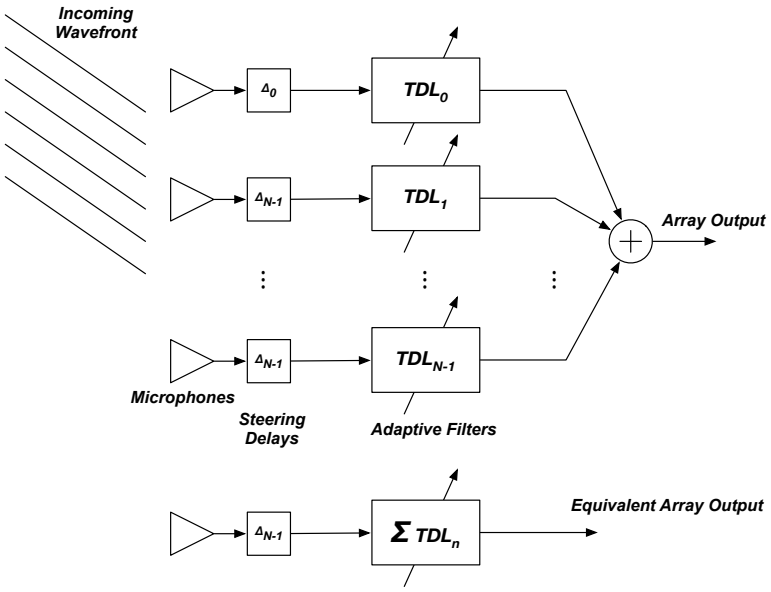


Figure 9.7: Upper panel: Block diagram of the Frost adaptive array algorithm. Lower panel: Equivalent signal processor for the Frost array, with equal to the sum of the individual filters in the upper panel. From Widrow and Stearns (1985).

$$\overline{\boldsymbol{W}}_k \triangleq \begin{bmatrix} w_{0k} & w_{1k} & \cdots & w_{(L-1)k} \\ w_{Lk} & w_{(L+1)k} & \cdots & w_{(2L-1)k} \\ w_{2Lk} & w_{(2L+1)k} & \cdots & w_{(3L1)k} \\ \vdots & \vdots & \ddots & \vdots \\ w_{(N-1)Lk} & \cdots & \cdots & w_{NLk} \end{bmatrix} \tag{9.21}$$

Note that here we are depicting the weights of each individual adaptive filter as a row vector rather than the column-vector notation that we had used up to now. As always, the observation variables are the inputs to the adaptive weights, and they are represented by the corresponding matrix

$$\overline{\boldsymbol{X}}_k \triangleq \begin{bmatrix} x_{0k} & x_{1k} & \cdots & x_{(L-1)k} \\ x_{Lk} & x_{(L+1)k} & \cdots & x_{(2L-1)k} \\ x_{2Lk} & x_{(2L+1)k} & \cdots & x_{(3L1)k} \\ \vdots & \vdots & \ddots & \vdots \\ x_{(N-1)Lk} & \cdots & \cdots & x_{NLk} \end{bmatrix} \tag{9.22}$$

Because the outputs of the N sensors are summed together, the response of the array is equivalent to that of a single adaptive FIR filter with weights equal to the $1 \times L$ vector $\boldsymbol{C}\overline{\boldsymbol{X}}_k$ where

$$\boldsymbol{C} \triangleq \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix} \tag{9.23}$$

9.5.3 The Griffiths-Jim algorithm

9.5.4 The minimum-variance distortionless response (MVDR) formulation

Appendix. Working with Delta Functions

In this Appendix we describe an analytical approach to solving all problems involving delta functions in time or frequency.

The unit impulse function $\delta(t)$ has a long and honorable history in signal processing. In its classic form the unit impulse function is used to represent pulse-like signals that are very brief compared to any of the meaningful time constants of a realizable system. It is *much* easier performing these computations using the idealized delta functions than with the original brief signals, even though we must put up with some mathematical extremes. We will begin by discussing three equivalent definitions of the delta function. Following that we will comment on how to obtain results for computations that involve the delta function.

9.6 Axiomatic definition of the delta function

In many engineering courses, the delta function is defined in the following fashion:

$$\delta(t) = 0, \text{ for } t \neq 0 \tag{9.24}$$

$$\int_{-\infty}^{\infty} \delta(t)dt = 1 \tag{9.25}$$

Since the delta function equals zero by definition for values of t other than zero, it must have infinite amplitude at $t = 0$ in order for it to maintain an area of one at $t = 0$. So under these circumstances we may think of the delta function as being infinitesimally wide but infinitely tall, with unit area.

9.7 Limiting definitions of the delta function

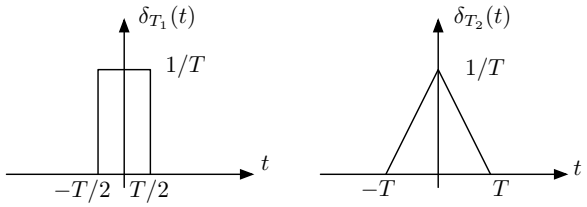


Figure 9.8: Two functions that approach $\delta(t)$ in the limiting case.

Consider a function $\delta_T(t)$ which has finite amplitude and width, but unit area. Two examples of such functions, $\delta_{T_1}(t)$ and $\delta_{T_2}(t)$, are depicted in the figure above. In either case, we can express the delta function as the limit

$$\delta(t) = \lim_{T \to 0} \delta_T(t) \tag{9.26}$$

where again, $\delta_T(t)$ can be any function of t that in the limit as T goes to zero has infinitesimal width and infinite height with unit area. In practice, the shape of the function does not matter, provided that the area remains constant independent of T as is the case with the functions $\delta_{T_1}(t)$ and $\delta_{T_2}(t)$ depicted above. It can easily be seen that this definition is consistent with (and in fact is a generalization of) the first definition.

9.8 Implicit definition of the delta function

The most general definition of the delta function, which we encourage you to use always, is the so-called distributional definition of the delta function. Specifically, let the function $\phi(t)$ be any function of t that is continuous everywhere. The delta function is then defined as

$$\int_{-\infty}^{\infty} \delta(t-a)\phi(t)dt = \phi(a) \tag{9.27}$$

Please note that this is a different kind of definition for a function than you may be used to: $\delta(t)$ is not defined by what it *is* but rather by what it *does* when subjected to the very specific operations of multiplication by a continuous "testing" function and then integration over all time. This type of definition is sometimes referred to as an *implicit* rather than explicit definition. In our work with delta functions, we will *only* work with them in the context of multiplication by a continuous function followed by subsequent integration. In our work, the integration operation will be in the context of either convolution or Fourier transformation.

It is easy to see that this definition is consistent with Eqs. (9.24) and (9.25). Specifically, $\delta(t)$ must equal 0 for (in this case) $t \neq a$ because the result of the integral depends only on the value of $\phi(t)$ at $t = a$. Since we have no idea what $\phi(t)$ is equal to (and in principle it could be nonzero everywhere), the fact that $\int_{-\infty}^{\infty} \delta(t-a)\phi(t)dt = \phi(a)$ implies that $\delta(t)$

equals zero everywhere except for $t = a$. Equation (9.27) also reduces to Eq. (9.25) if we let $a = 0$ and $\phi(t) = 1$ for all t.

The delta function is also sometimes referred to as a "sifting function" because it extracts the value of a continuous function at one point in time.

9.9 Computation with the delta function

We encourage you to approach the evaluation of all integrals involving the delta function using the procedure implied by Eq. (9.27). Specifically, evaluate the integral by applying three-step procedure:

1. Ask the question "What variable is being integrated?" [t in Eq. (9.27)]
2. Ask the question "What is the value of that variable that causes the argument of the delta function to equal zero?" [$t = a$ in Eq. (9.27)]
3. Then the result of the integration is the rest of the integrand evaluated at that value of the variable that is being integrated. [$\phi(a)$ in Eq. (9.27)]

We will illustrate these principles in a few examples below.

Example 9.1

$$\int_{-\infty}^{\infty} x(\tau)\delta(t-\tau-a)d\tau \tag{9.28}$$

This is an equation that may come up in a convolution problem when the system is an ideal delayer. The evaluation of the integral is straightforward following the discussion above:

1. "What variable is being integrated?" [τ in Eq. (9.28)]
2. "What is the value of that variable that causes the argument of the delta function to equal zero?" [$\tau = t - a$ in Eq. (9.28)]
3. The result of the integration is the rest of the integrand evaluated at that value of the variable that is being integrated. [$x(t-a)$ in Eq. (9.28)]

Example 9.2

$$\frac{1}{2\pi}\int_{-\infty}^{\infty} \delta(\Omega-\Omega_0)e^{j\Omega t}d\Omega \tag{9.29}$$

This is an example of an inverse continuous-time Fourier transform, but the evaluation once again is straightforward:

1. "What variable is being integrated?" [Ω in Eq. (9.29)]
2. "What is the value of that variable that causes the argument of the delta function to equal zero?" [$\Omega = \Omega_0$ in Eq. (9.29)]

3. The result of the integration is the rest of the integrand evaluated at that value of the variable that is being integrated. [$\frac{1}{2\pi}e^{j\Omega_0 t}$ in Eq. (9.29)]

Example 9.3

$$\int_{-\infty}^{\infty} \delta(2t)dt \tag{9.30}$$

This integral, which illuminates a property of delta functions, is only slightly less straightforward. In principle, we cannot evaluate this integral directly because Eq. (9.27) is defined in terms of $\delta(t)$ rather than $\delta(2t)$. Nevertheless, we can easily work around this issue with a change of variables. Specifically, let $t' = 2t$. Then $dt' = 2dt$, while $t = t'/2$ and $dt = dt'/2$. Hence we can write directly

$$\int_{-\infty}^{\infty} \delta(2t)dt = \int_{-\infty}^{\infty} \delta(t')dt'/2 = \frac{1}{2}\int_{-\infty}^{\infty} \delta(t')dt' = \frac{1}{2}$$

This last result makes sense, as replacing the argument t in the delta function by $2t$ causes the delta function to be compressed by a factor of 2 in time. Consequently the area of the delta function will be multiplied by a factor of 1/2. In general we can state that

$$\delta(at) = \frac{1}{|a|}\delta(t)$$

Again, we restate that *every* integral involving delta functions can (and should!) be evaluated using the three-step procedure outlined above.

9.10 The unit step function and derivatives of discontinuous functions

As you know, the continuous-time unit step function is defined as

$$u(t) = \begin{cases} 0 & t < 0, \\ 1 & t > 0 \end{cases} \tag{9.31}$$

(We do not need to worry about the definition of $u(0)$ for now or for that matter, ever). The unit step function can be considered to be the integral of the delta function in that

$$u(t) = \int_{-\infty}^{t} \delta(\tau)d\tau \tag{9.32}$$

While this may imply that $\delta(t)$ is the derivative of $u(t)$, this cannot be stated in the ordinary sense because of the discontinuity of $u(t)$ at $t = 0$. Nevertheless, we can use delta functions to represent the derivatives of functions that are continuous except for a finite number of points. For example, if $x(t)$ is continuous everywhere except for $t = a$, and $x(a+) = x(a-) + k$, then the derivative of $x(t)$ would be

$$\frac{dx}{dt} = \begin{cases} \frac{dx}{dt} \text{ in the ordinary sense for } t \neq a, \\ k\delta(t-a) \text{ for } t = a \end{cases} \tag{9.33}$$

In other words, if there are isolated discontinuities in an $x(t)$ that is otherwise continuous, the derivative of $x(t)$ would be the ordinary derivative where $x(t)$ is continuous, and there would be delta functions at the locations along the t axis where the discontinuities are observed. The areas of these delta functions would be equal to the size of the discontinuity at that location.

For example, if

$$x(t) = \begin{cases} t^2 & t < 3, \\ t^2 + 2 & t > 3 \end{cases}$$

then we would observe

$$\frac{dx(t)}{dt} = 2t + 2\delta(t-3)$$

Similarly, if

$$u(t) = \begin{cases} 0 & t < 0, \\ 1 & t > 0 \end{cases}$$

then we would observe

$$\frac{du(t)}{dt} = \delta(t) \tag{9.34}$$