Marco Tagliasacchi

Lecture Notes on Fundamentals of Audio-Video Signal Processing

November 12, 2007

Politecnico di Milano

Contents

Part I

Fundamentals of Digital Signal Processing

1

Introduction to the Discrete Fourier Transform

This chapter introduces the Discrete Fourier Transform (DFT). Before we get started on the DFT, let's look for a moment at the Fourier transform (FT) and explain why we are not talking about it instead. The Fourier transform of a continuous-time signal $x(t)$ may be defined as

$$\boxed{X(\omega) \triangleq \int_{-\infty}^{+\infty} x(t)e^{-j\omega t}dt, \quad \omega \in (-\infty, +\infty).} \tag{1.1}$$

Thus, we need calculus. The DFT, on the other hand, replaces the infinite integral with a finite sum:

$$\boxed{X(\omega_k) \triangleq \sum_{n=0}^{N-1} x(t_n)e^{-j\omega_k t_n}, \quad k = 0, 1, 2, \ldots, N-1,} \tag{1.2}$$

where the various quantities in this formula are defined on the next page. Calculus is not needed to define the DFT (or its inverse, as we will see), and with finite summation limits, we cannot encounter difficulties with infinities (provided that $x(t_n)$ is finite, which is always true in practice). Moreover, in the field of digital signal processing, signals and spectra are processed only in sampled form, so that the DFT is what we really need anyway (implemented using the FFT - Fast Fourier Transform - when possible). In summary, the DFT is simpler mathematically, and more relevant computationally than the Fourier transform. At the same time, the basic concepts are the same.

1.1 Discrete Fourier Transform

The Discrete Fourier Transform (DFT) of a signal may be defined by:

$$X(\omega_k) \triangleq \sum_{n=0}^{N-1} x(t_n)e^{-j\omega_k t_n}, \quad k = 0, 1, 2, \ldots, N-1, \tag{1.3}$$

where $\triangleq$ means "is defined as" or "equals by definition"

$$\sum_{n=0}^{N-1} f(n) \triangleq f(0) + f(1) + \ldots + f(N-1)$$

$x(t_n) \triangleq$ input signal *amplitude* (real or complex) at time t_n (sec)

$t_n \triangleq nT = n$th sampling instant (sec), n an integer ≥ 0

$T \triangleq$ sampling interval (sec)

$X(\omega_k) \triangleq$ *spectrum* of x (complex valued), at frequency ω_k

$\omega_k \triangleq k\Omega = k$th frequency sample (radians per second)

$\Omega \triangleq \dfrac{2\pi}{NT} =$ radian-frequency sampling interval (rad/sec)

$f_s \triangleq 1/T =$ *sampling rate* (samples/sec, or Hertz (Hz))

$N =$ number of time samples = no. frequency samples (integer).

The sampling interval is also called the sampling *period*.

When all N signal samples $x(t_n)$ are real, we say $x \in \mathbb{R}^N$. If they may be complex, we write $x \in \mathbb{C}^N$. Finally, $n \in \mathbb{Z}$ means n is an integer.

1.2 Inverse Discrete Fourier Transform

The inverse DFT (the IDFT) is given by

$$\boxed{x(t_n) = \frac{1}{N} \sum_{k=0}^{N-1} X(\omega_k) e^{j\omega_k t_n}, \quad n = 0, 1, 2, \ldots, N-1,} \tag{1.4}$$

1.3 Mathematics of the DFT

In the signal processing literature, it is common to write the DFT and its inverse in the more pure form below, obtained by setting $T = 1$ in the previous definition:

$$\boxed{X(k) \triangleq \sum_{n=0}^{N-1} x(n) e^{-j2\pi nk/N}, \quad k = 0, 1, 2, \ldots, N-1,} \tag{1.5}$$

$$\boxed{x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{j2\pi nk/N}, \quad n = 0, 1, 2, \ldots, N-1,} \tag{1.6}$$

where $x(n)$ denotes the input signal at time (sample) n, and $X(k)$ denotes the kth spectral sample. This form is the simplest mathematically, while the previous form is easier to interpret physically.

Recall the Euler's Identity:

$$e^{j\theta} = \cos(\theta) + j\sin(\theta). \tag{1.7}$$

Euler's Identity is the key to understanding the meaning of expressions like

$$s_k(t_n) \triangleq e^{j\omega_k t_n} = \cos(\omega_k t_n) + j\sin(\omega_k t_n). \tag{1.8}$$

Such expression defines a sampled complex sinusoid.

Finally, we need to understand what the summation over n is doing in the definition of the DFT. This can be interpreted as the computation of the inner product of the signals x and s_k defined above, so that we may write the DFT, using inner-product notation, as

$$X(k) \triangleq \langle x, s_k \rangle, \tag{1.9}$$

where $s_k(n) = e^{j2\pi nk/N}$ is the sampled complex sinusoid at (normalized) radian frequency $\omega_k = 2\pi k/N$, and the inner product operation $\langle \cdot, \cdot \rangle$ is defined by

$$\langle x, y \rangle \triangleq \sum_{n=0}^{N-1} x(n)\bar{y}(n). \tag{1.10}$$

The inner product of x with the kth basis sinusoid s_k is a measure of "how much" of s_k is present in x and at "what phase" (since it is a complex number).

Listing 1.1. Matlab

```
x = [-20:20];
N = length(x);

omega = 2*pi*[0:N-1]/N;

X = fft(x);

subplot(2,1,1); plot(omega, abs(X));
subplot(2,1,2); plot(omega, angle(X));

y = ifft(X);
```

1.4 Derivation of the Discrete Fourier Transform

Geometric series

Recall that for any complex number $z_1 \in \mathbb{C}$, the signal

$$x(n) \triangleq z_1^n, \quad n = 0, 1, 2, \ldots, \tag{1.11}$$

defines a geometric sequence, i.e. each term is obtained by multiplying the previous term by a (complex) constant. A geometric series is the sum of a geometric sequence:

$$S_N(z_1) \triangleq \sum_{n=0}^{N-1} z_1^n = 1 + z_1 + z_1^2 + z_1^3 + \ldots + z_1^{N-1} \tag{1.12}$$

If $z_1 \neq 1$, the sum can be expressed in closed form

$$S_N(z_1) = \frac{1 - z_1^N}{1 - z_1} \quad (z_1 \neq 1) \tag{1.13}$$

Orthogonality of sinusoids

A key property of sinusoids is that they are orthogonal at different frequencies. That is,

$$\omega_1 \neq \omega_2 \Rightarrow A_1 \sin(\omega_1 t + \phi_1) \perp A_2 \sin(\omega_2 t + \phi_2). \tag{1.14}$$

This is true whether they are complex or real, and whatever amplitude and phase they may have. All that matters is that the frequencies be different. Note, however, that the sinusoidal durations must be infinity.

For length N sampled sinusoidal signal segments, such as used by the DFT, exact orthogonality holds only for the harmonics of the sampling-rate-divided-by-N, i.e., only for the frequencies

$$\omega_k = 2\pi k \frac{f_s}{N}, \quad k = 0, 1, 2, \ldots, N-1. \tag{1.15}$$

These are the only frequencies that have a whole number of periods in samples.

The complex sinusoids corresponding to the frequencies ω_k are

$$s_k(n) \triangleq e^{j\omega_k nT}, \qquad \omega_k \triangleq k\frac{2\pi}{N} f_s, \quad k = 0, 1, 2, \ldots, N-1. \tag{1.16}$$

Figure 1.1 shows the sampled sinusoids $(W_N^k)^n = e^{j2\pi kn/N} = e^{j\omega_k nT}$ used by the DFT. Note that taking successively higher integer powers of the point W_N^k on the unit circle generates samples of the kth DFT sinusoid, giving $[W_N^k]^n, n = 0, 1, 2, \ldots, N-1$. The kth sinusoid generator W_N^k is in turn the kth Nth root of unity (kth power of the primitive Nth root of unity W_N).

Note that in Figure 1.1 the range of k is taken to be $[-N/2, N/2-1] = [-4, 3]$ instead of $[0, N-1] = [0, 7]$. This is the most physical choice since it corresponds with our notion of "negative frequencies". However, we may add any integer multiple of N to k without changing the sinusoid indexed by k. In other words, $k \pm mN$ refers to the same sinusoid $e^{j\omega_k nT}$ for all integers m.

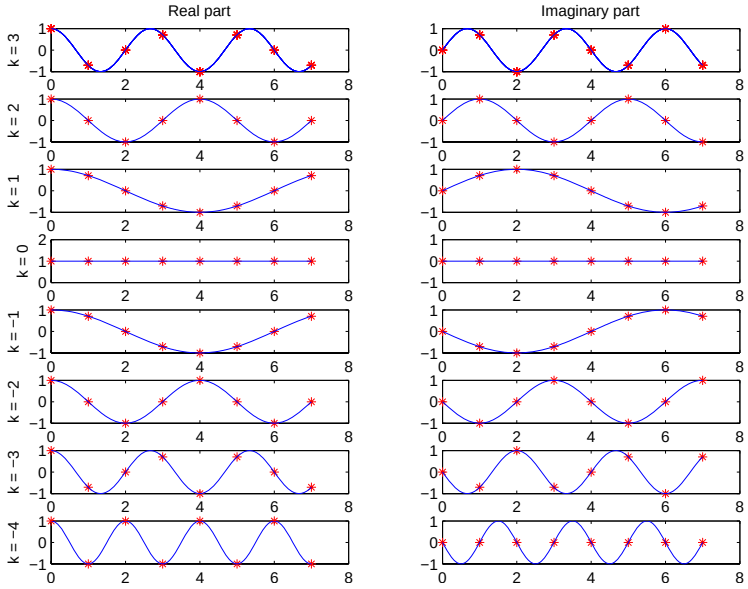


Fig. 1.1. DFT sinusoids, for $N = 8$

Orthogonality of the DFT sinusoids

We now show mathematically that the DFT sinusoids are exactly orthogonal. Let

$$s_k(n) \triangleq e^{j\omega_k nT} = e^{j2\pi kn/N} = [W_N^k]^n, \quad n = 0, 1, 2, \ldots, N-1 \tag{1.17}$$

denote the kth complex DFT sinusoid. Then

$$\begin{aligned} \langle s_k, s_l \rangle &\triangleq \sum_{n=0}^{N-1} s_k(n)\bar{s}_l(n) = \sum_{n=0}^{N-1} e^{j2\pi kn/N} e^{-j2\pi ln/N} = \\ &= \sum_{n=0}^{N-1} e^{j2\pi(k-l)n/N} = \frac{1 - e^{j2\pi(k-l)}}{1 - e^{j2\pi(k-l)/N}} \end{aligned} \tag{1.18}$$

where the last step made use of the closed-form expression for the sum of a geometric series. If $k \neq l$, the denominator is nonzero while the numerator is zero. This proves

$$s_k \perp s_l, \quad k \neq l \tag{1.19}$$

While we only looked at unit amplitude, zero phase complex sinusoids, as used by the DFT, it is readily verified that the (nonzero) amplitude and phase have no effect on orthogonality.

Norm of the DFT sinusoids

For $k = l$, we follow the previous derivation to the next-to-last step to get

$$\langle s_k, s_k \rangle = \sum_{n=0}^{N-1} e^{j2\pi(k-k)n/N} = N \tag{1.20}$$

which proves

$$\|s_k\| = \sqrt{N} \tag{1.21}$$

The Discrete Fourier Transform (DFT)

Given a signal $x(\cdot) \in \mathbb{C}^N$, its DFT is defined by

$$X(\omega_k) \triangleq \langle x, s_k \rangle = \sum_{n=0}^{N-1} x(n)\bar{s}_k(n), \quad s_k(n) \triangleq e^{j\omega_k nT}, \quad \omega_k \triangleq 2\pi\frac{k}{N}f_s, \quad k = 0, 1, 2, \ldots, N-1, \tag{1.22}$$

or, as it is most often written

$$X(\omega_k) \triangleq \sum_{n=0}^{N-1} x(n)e^{-j\frac{2\pi kn}{N}}, \quad k = 0, 1, 2, \ldots, N-1. \tag{1.23}$$

We may also refer to X as the spectrum of x, and $X(\omega_k)$ is the kth sample of the spectrum at frequency ω_k.

Given two vectors x and y, the projection of x onto y is defined as

$$\mathbf{P}_y(x) = \frac{\langle x, y \rangle}{\|y\|^2} y \tag{1.24}$$

where $\frac{\langle x,y \rangle}{\|y\|^2}$ is the coefficient of projection. The kth sample $X(\omega_k)$ of the spectrum of x is defined as the inner product of x with the kth DFT sinusoid s_k. This definition is N times the coefficient of projection of x onto s_k, i.e.,

$$\frac{\langle x, s_k \rangle}{\|s_k\|^2} = \frac{X(\omega_k)}{N} \tag{1.25}$$

The projection of x onto s_k is

$$\mathbf{P}_{s_k}(x) = \frac{X(\omega_k)}{N} s_k \tag{1.26}$$

Since the $\{s_k\}$ are orthogonal and span $\mathbb{C}^N$, we have that the inverse DFT is given by the sum of projections

$$x = \sum_{k=0}^{N-1} \frac{X(\omega_k)}{N} s_k \tag{1.27}$$

or, as we normally write,

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(\omega_k) e^{j\frac{2\pi kn}{N}} \tag{1.28}$$

In summary, the DFT is proportional to the set of coefficients of projection onto the sinusoidal basis set, and the inverse DFT is the reconstruction of the original signal as a superposition of its sinusoidal projections. This basic "architecture" extends to all linear orthogonal transforms, including wavelets, Fourier transforms, Fourier series, the discrete-time Fourier transform (DTFT), and certain short-time Fourier transforms (STFT).

1.5 Frequencies in the "cracks"

The DFT is defined only for frequencies $\omega_k = 2\pi k f_s/N$. If we are analyzing one or more periods of an exactly periodic signal, where the period is exactly N samples (or some integer divisor of N), then these really are the only frequencies present in the signal, and the spectrum is actually zero everywhere but at $\omega = \omega_k$, $k \in [0, N-1]$. However, we use the DFT to analyze arbitrary signals from nature. What happens when a frequency ω is present in a signal x that is not one of the DFT-sinusoid frequencies ω_k?

To find out, let's project a length N segment of a sinusoid at an arbitrary frequency ω onto the kth DFT sinusoid:

$$\begin{aligned} x(n) &\triangleq e^{j\omega nT} \\ s_k(n) &\triangleq e^{j\omega_k nT} \\ \mathbf{P}_{s_k}(x) = \frac{\langle x, s_k\rangle}{\langle s_k, s_k\rangle} s_k &\triangleq \frac{X(\omega_k)}{N} s_k \end{aligned} \tag{1.29}$$

The coefficient of projection is proportional to

$$\begin{aligned} X(\omega_k) \triangleq \langle x, s_k\rangle &\triangleq \sum_{n=0}^{N-1} x(n)\bar{s}_k(n) = \\ &= \sum_{n=0}^{N-1} e^{j\omega nT} e^{-j\omega_k nT} = \sum_{n=0}^{N-1} e^{j(\omega-\omega_k)nT} = \frac{1 - e^{j(\omega-\omega_k)NT}}{1 - e^{j(\omega-\omega_k)T}} \\ &= e^{j(\omega-\omega_k)(N-1)T/2} \frac{\sin[(\omega-\omega_k)NT/2]}{\sin[(\omega-\omega_k)T/2]}, \end{aligned} \tag{1.30}$$

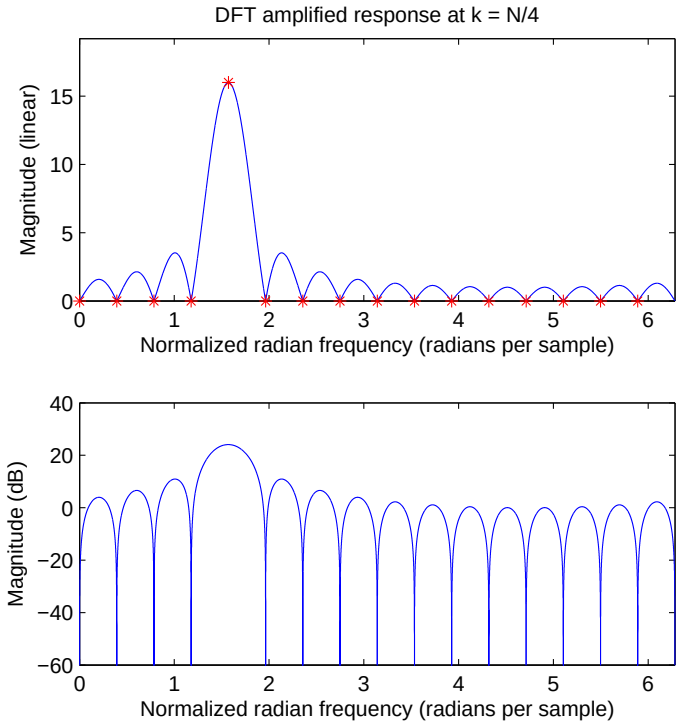


Fig. 1.2. Frequency response magnitude of a single DFT output sample

using the closed-form expression for a geometric series sum. The sum is N at $\omega = \omega_k$ and zero at ω_l, for $l \neq k$. However, the sum is nonzero at all other frequencies.

The typical way to think about this in practice is to consider the DFT operation as a digital filter. The frequency response of this filter is what we just computed, and its magnitude is

$$\boxed{|X(\omega_k)| = \left|\frac{\sin[(\omega - \omega_k)NT/2]}{\sin[(\omega - \omega_k)T/2]}\right|} \tag{1.31}$$

At all other integer values of k (see Figure 1.2), the frequency response is the same but shifted (circularly) left or right so that the peak is centered on ω_k. The secondary peaks away from ω_k are called sidelobes of the DFT response, while the main peak may be called the main lobe of the response. Since we are normally most interested in spectra from an audio perspective, the same plot is repeated using a decibel vertical scale in Figure 1.2 (clipped at -60dB). We see that the sidelobes are really quite high from an audio perspective. Sinusoids with frequencies near $\omega_{k\pm1.5}$, for example, are only attenuated approximately 13dB in the DFT output $X(\omega_k)$.

We see that $X(\omega_k)$ is sensitive to all frequencies between dc and the sampling rate except the other DFT-sinusoid frequencies ω_l for $l \neq k$. This is sometimes called *spectral*

leakage or *cross-talk* in the spectrum analysis. Again, there is no error when the signal being analyzed is truly periodic and we can choose to be exactly a period, or some multiple of a period. Normally, however, this cannot be easily arranged, and spectral leakage can be a problem.

Note that peak spectral leakage is not reduced by increasing N. It can be thought of as being caused by abruptly truncating a sinusoid at the beginning and/or end of the N-sample time window. Only the DFT sinusoids are not cut off at the window boundaries. All other frequencies will suffer some truncation distortion, and the spectral content of the abrupt cut-off or turn-on transient can be viewed as the source of the sidelobes. Remember that, as far as the DFT is concerned, the input signal $x(n)$ is the same as its periodic extension. If we repeat samples of a sinusoid at frequency $\omega \neq \omega_k$ (for any $k \in \mathbb{Z}$), there will be a "glitch" every N samples since the signal is not periodic in N samples. This glitch can be considered a source of new energy over the entire spectrum.

To reduce spectral leakage (cross-talk from far-away frequencies), we typically use a window function, such as a "raised cosine" window, to taper the data record gracefully to zero at both endpoints of the window. As a result of the smooth tapering, the main lobe widens and the sidelobes decrease in the DFT response. Using no window is better viewed as using a rectangular window of length N, unless the signal is exactly periodic in N samples.

Since the kth spectral sample $X(\omega_k)$ is properly regarded as a measure of spectral amplitude over a range of frequencies, nominally $k-1/2$ to $k+1/2$, this range is sometimes called a *frequency bin* (as in a "storage bin" for spectral energy). The frequency index k is called the *bin number*, and can be regarded as the total energy in the kth bin.

In the very special case of *truly periodic signals* $x(t) = x(t+NT)$, for all $t \in (-\infty, +\infty)$, the DFT may be regarded as computing the Fourier series coefficients of $x(n)$ from one period of its sampled representation $x(nT)$, $n = 0, 1, , \ldots, N-1$. The period of x must be exactly NT seconds for this to work.

Listing 1.2. Matlab

```
T = 1;

N = 1000; t = 0:T:(N-1)*T;

K = 10;
%spectral leakage occurs if K is not integer
%K = 10.5;
f = K*(1/(N*T));

x = sin(2*pi*f*t);

X = fft(x); omega = 2*pi*[0:N-1]/N;

subplot(2,1,1)
plot(x)
subplot(2,1,2)
plot(omega, 10*log10(abs(X).^2));
```

1.6 Matrix Formulation of the DFT

The DFT can be formulated as a complex matrix multiply, as we show in this section.

The DFT consists of inner products of the input signal $\underline{x}$ with sampled complex sinusoidal sections $\underline{s}_k$:

$$X(\omega_k) \triangleq \langle \underline{x}, \underline{s}_k \rangle \triangleq \sum_{n=0}^{N-1} x(t_n) e^{-j2\pi kn/N}, \quad k = 0, 1, 2, \ldots, N-1, \tag{1.32}$$

By collecting the DFT output samples into a column vector, we have

$$\begin{bmatrix} X(\omega_0) \\ X(\omega_1) \\ X(\omega_2) \\ \vdots \\ X(\omega_{N-1}) \end{bmatrix} = \begin{bmatrix} s_0(0) & s_0(1) & \cdots & s_0(N-1) \\ s_1(0) & s_1(1) & \cdots & s_1(N-1) \\ s_2(0) & s_2(1) & \cdots & s_2(N-1) \\ \vdots & \vdots & \ddots & \vdots \\ s_{N-1}(0) & s_{N-1}(1) & \cdots & s_{N-1}(N-1) \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ \vdots \\ x(N-1) \end{bmatrix} \tag{1.33}$$

or

$$\boxed{\underline{X} = \mathbf{S}_N \underline{x}} \tag{1.34}$$

where $\mathbf{S}_N$ denotes the DFT matrix $\mathbf{S}_N \triangleq W_N^{-kn} \triangleq e^{-j2\pi kn/N}$, i.e.

$$\mathbf{S}_N = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & e^{-j2\pi/N} & e^{-j4\pi/N} & \ldots & e^{-j2\pi(N-1)/N} \\ 1 & e^{-j4\pi/N} & e^{-j8\pi/N} & \ldots & e^{-j2\pi 2(N-1)/N} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & e^{-j2\pi(N-1)/N} & e^{-j2\pi 2(N-1)/N} & \ldots & e^{-j2\pi(N-1)(N-1)/N} \end{pmatrix} \tag{1.35}$$

We see that the kth row of the DFT matrix is the kth DFT sinusoid. Since the matrix is symmetric, $\mathbf{S}_N^T = \mathbf{S}_N$ (where transposition does not include conjugation), we observe that the kth column of $\mathbf{S}_N$ is also the kth DFT sinusoid.

The inverse DFT matrix is simply $\bar{\mathbf{S}}_N/N$. That is, we can perform the inverse DFT operation as

$$\boxed{\underline{x} = \frac{1}{N} \mathbf{S}_N^* \underline{X}} \tag{1.36}$$

where $A^* \triangleq \bar{A^T}$ denotes the Hermitian transpose of the complex matrix A (transposition and complex conjugation). Since $\underline{X} = \mathbf{S}_N \underline{x}$, the above implies

$$\mathbf{S}_N^* \mathbf{S}_N = N \cdot \mathbf{I}. \tag{1.37}$$

The above equation succinctly implies that the columns of $\mathbf{S}_N$ are *orthogonal*, which, of course, we already knew.

1.7 Fourier Theorems for the DFT

The DFT and its inverse restated

Let $x(n)$, $n = 0, 1, 2, \ldots, N-1$, denote the N-sample complex sequence, i.e. $x \in \mathbb{C}^N$. Then the spectrum of x is defined by the Discrete Fourier Transform (DFT)

$$X(k) \triangleq \sum_{n=0}^{N-1} x(n) e^{-j2\pi kn/N}, \quad k = 0, 1, 2, \ldots, N-1. \tag{1.38}$$

The Inverse DFT (IDFT) is defined by

$$x(n) = \sum_{n=0}^{N-1} X(k) e^{j2\pi kn/N}, \quad n = 0, 1, 2, \ldots, N-1. \tag{1.39}$$

In this section, we will omit mention of an explicit sampling interval $T = 1/f_s$, as is most typical in the digital signal processing literature. It is often said that the sampling frequency is $f_s = 1$. In this case, a radian frequency $\omega_k = 2\pi k/N$ is in units of "radians per sample." Another term we use in connection with the $f_s = 1$ convention is normalized frequency. All normalized radian frequencies lie in the range $[-\pi, +\pi)$, and all normalized frequencies in cycles/sample lie in the range $[-0.5, +0.5)$. Note that physical units of seconds and Herz can be reintroduced by the substitution

$$e^{j2\pi nk/N} = e^{j2\pi k(f_s/N)nT} = e^{\omega_k t_n}. \tag{1.40}$$

Notation and terminology

If X is the DFT of x, we say that x and X form a transform pair and write

$$x \leftrightarrow X \quad (\text{"}x \text{ corresponds to } X\text{"}) \tag{1.41}$$

Another notation we will use is

$$DFT(x) \triangleq X, \tag{1.42}$$

$$DFT_k(x) \triangleq X(k), \tag{1.43}$$

If we need to indicate the length of the DFT explicitly we write $DFT_N(x) = X$ and $DFT_{N,k} = X(k)$.

Modulo indexing, periodic extension

The DFT sinusoids $s_k(n) = e^{j\omega_k n}$ are all periodic having periods which divide N. That is, $s_k(n + mN) = s_k(n)$ for any integer m.

Since a length N signal x can be expressed as a linear combination of the DFT sinusoids in the time domain,

$$x(n) = \frac{1}{N} \sum_k X(k) s_k(n), \tag{1.44}$$

it follows that the "automatic" definition of $x(n)$ beyond the range $[0, N-1]$ is periodic extension, i.e., $x(n + mN) = x(n)$ for every integer m.

Moreover, the DFT also repeats naturally every N samples, since

$$X(k + mN) \triangleq \langle x, s_{k+mN} \rangle = \langle x, s_k \rangle = X(k) \tag{1.45}$$

because $s_{k+mN}(n) = e^{j2\pi n(k+mN)/N} = e^{j2\pi nk/N} e^{j2\pi nm} = e^{j2\pi nk/N} = s_k(n)$. Accordingly, for purposes of DFT studies, we may define all signals in $\mathbb{C}^N$ as being single periods from an infinitely long periodic signal with period N samples:

For any signal $x \in \mathbb{C}^N$, we define

$$x(n + mN) \triangleq x(n) \tag{1.46}$$

for every integer m.

As a result of this convention, all indexing of signals and spectra can be interpreted modulo N, and we may write $x(n(\mod N))$ to emphasize this. Formally, "$n(\mod (N))$" is defined as $n - mN$ with m chosen to give $n - mN$ in the range $[0, N-1]$.

As an example, when indexing a spectrum X, we have that $X(N) = X(0)$ which can be interpreted physically as saying that the sampling rate is the same frequency as dc for discrete time signals. Periodic extension in the time domain implies that the signal input to the DFT is mathematically treated as being samples of one period of a periodic signal, with the period being exactly NT seconds (N samples). The corresponding assumption in the frequency domain is that the spectrum is exactly zero between frequency samples ω_k. It is also possible to adopt the point of view that the time-domain signal $x(n)$ consists of samples preceded and followed by zeros. In that case, the spectrum would be nonzero between spectral samples ω_k, and the spectrum between samples would be reconstructed by means of bandlimited interpolation.

Signal operators

Flip operator

$$\mathrm{FLIP}_n(x) \triangleq x(-n) \tag{1.47}$$

which, by modulo indexing is also $x(N-n)$

Shift operator

$$\mathrm{SHIFT}_{\Delta,n}(x) \triangleq x(n-\Delta), \quad \Delta \in \mathbb{Z} \tag{1.48}$$

Note that since indexing is modulo N, the shift is circular (or cyclic). We often use the shift operator in conjunction with zero padding (appending zeros to the signal) in order to avoid the "wrap-around" associated with a circular shift.

Convolution

$$\boxed{(x * y)_n \triangleq \sum_{m=0}^{N-1} x(m)y(n-m)} \tag{1.49}$$

Note that this is *circular* convolution (or cyclic convolution).

Convolution is commutative, i.e. $x * y = y * x$.

Note that the cyclic convolution operation can be expressed in terms of the previously defined operators as

$$y(n) = \sum_{m=0}^{M-1} x(m)h(n-m) = \langle x, \mathrm{SHIFT}_n(\mathrm{FLIP}(h))\rangle \quad h \text{ real}) \tag{1.50}$$

We may interpret h as the impulse response of a digital filter. It is instructive to interpret the notation graphically (see Figure 1.3).

Correlation

$$\boxed{(x \star y)_n \triangleq \sum_{m=0}^{N-1} \bar{x}(m)y(m+n)} \tag{1.51}$$

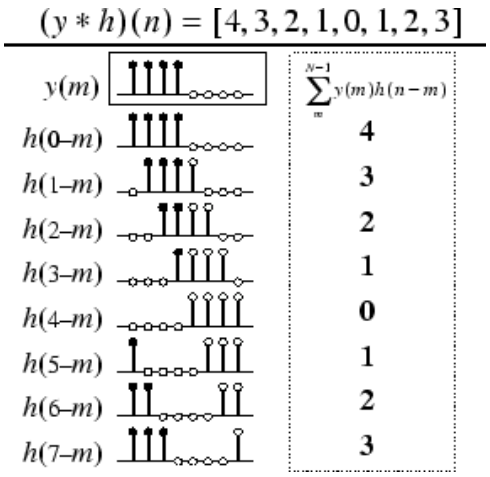


Fig. 1.3. Illustration of convolution of $y = [1, 1, 1, 1, 0, 0, 0, 0]$ and $h = [1, 0, 0, 0, 0, 0, 1, 1, 1]$

Zero padding

Zero padding consists of extending a signal (or spectrum) with zeros to extend its time (or frequency band) limits. It maps a length N signal to a length $M > N$ signal, but M need not be an integer multiple of N. To unify the time-domain and frequency-domain definitions of zero-padding, it is necessary to regard the original time axis as indexing positive-time samples from 0 to $N/2-1$ (for N even), and negative times in the interval $n \in [N-N/2+1, N-1] = [-N/2+1, -1]$. Furthermore, we require $x(N/2) = x(-N/2)$ when N is even, while N odd requires no such restriction.

$$\text{ZEROPAD}_{M,m} \triangleq \begin{cases} x(m), & |m| < N/2 \\ 0, & \text{otherwise} \end{cases} \tag{1.52}$$

where $m = 0, \pm 1, \pm 2, \ldots, \pm M_h$, with $M_h \triangleq (M-1)/2$ for M odd, and $M/2-1$ for M even.

When a signal $x(n)$ is causal, that is, $x(n) = 0$ for all negative samples, then zero-padding can be carried out by simply appending zeros to the original signal, for example

$$\text{ZEROPAD}_{10}([1, 2, 3, 4, 5]) = [1, 2, 3, 4, 5, 0, 0, 0, 0, 0] \quad \text{causal case} \tag{1.53}$$

The Fourier theorems

Linearity

For any $x, y \in \mathbb{C}^N$ and $\alpha, \beta \in \mathbb{C}$, the DFT satisfies

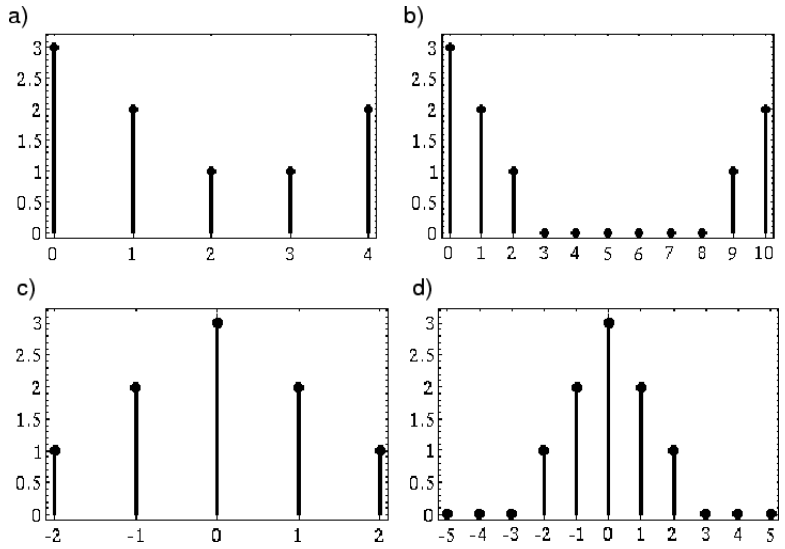


Fig. 1.4. Illustration of zero padding a) Original signal (or spectrum) plotted over the domain $n \in [0, N-1]$. b) $\mathrm{ZEROPAD}_{11}(x)$. c) The same signal x plotted over the domain $n \in [-(N-1)/2, (N-1)/2]$ which is more natural for interpreting negative times (frequencies). d) $\mathrm{ZEROPAD}_{11}(x)$ plotted over the zero-centered domain

$$\alpha x + \beta y \leftrightarrow \alpha X + \beta Y \tag{1.54}$$

Thus the DFT is a linear operator.

Conjugation and reversal

For any $x \in \mathbb{C}^N$

$$\bar{x} \leftrightarrow \mathrm{FLIP}(\bar{X}) \tag{1.55}$$

$$\mathrm{FLIP}(\bar{x}) \leftrightarrow \bar{X} \tag{1.56}$$

For any $x \in \mathbb{R}^N$

$$\mathrm{FLIP}(x) \leftrightarrow \bar{X} \tag{1.57}$$

Thus, conjugation in the frequency domain corresponds to reversal in the time domain. Another way to say it is that negating spectral phase flips the signal around backwards in time.

$$\mathrm{FLIP}(X) = \bar{X} \tag{1.58}$$

The property $X(-k) = \bar{X}(k)$ is called Hermitian symmetry or "conjugate symmetry".

$$\boxed{x \in \mathbb{R}^N \leftrightarrow X \text{ is Hermitian}} \tag{1.59}$$

Symmetry

In the previous section we found $\mathrm{FLIP}(X) = \bar{X}$ when x is real. This fact is of high practical importance. Due to this symmetry, we may discard all negative-frequency spectral samples of a real signal and regenerate them later if needed from the positive-frequency samples. Also, spectral plots of real signals are normally displayed only for positive frequencies; e.g., spectra of sampled signals are normally plotted over the range 0 Hz to $f_s/2$ Hz. On the other hand, the spectrum of a complex signal must be shown, in general, from $-f_2/2$ to $f_s/2$ (or from 0 to f_s), since the positive and negative frequency components of a complex signal are independent.

If $x \in \mathbb{R}^N$, then $\mathrm{re}\{X\}$ is even and $\mathrm{im}\{X\}$ is odd.

If $x \in \mathbb{R}^N$, then $|X|$ is even and $\angle X$ is odd.

A real even signal has a real even transform:

$$\boxed{x \text{ real and even} \leftrightarrow X \text{ real and even}} \tag{1.60}$$

A signal with a real spectrum (such as any real, even signal) is often called a zero phase signal. However, note that when the spectrum goes negative (which it can), the phase is really $\pm\pi$, not 0. We can say that all real spectra are piecewise constant-phase spectra, where the two constant values are 0 and $\pm\pi$.

Shift theorem

For any $x \in \mathbb{C}^N$ and any integer Δ,

$$\boxed{DFT_k[\mathrm{SHIFT}_\Delta(x)] = DFT_k[x(n-\Delta)] = e^{-j\omega_k\Delta}X(k)} \tag{1.61}$$

The shift theorem says that a delay in the time domain corresponds to a linear phase term in the frequency domain. Note that spectral magnitude is unaffected by a linear phase term.

The reason $e^{j\omega_k\Delta}$ is called a linear phase term is that its phase is a linear function of frequency.

$$\angle e^{-j\omega_k\Delta} = -\Delta\omega_k \tag{1.62}$$

Thus, the slope of the phase, viewed as a linear function of radian-frequency ω_k, is $-\Delta$.

A positive time delay (waveform shift to the right) adds a negatively sloped linear phase to the original spectral phase. A negative time delay (waveform shift to the left) adds a positively sloped linear phase to the original spectral phase.

A signal is said to be a linear phase signal if its phase is of the form

$$\Theta(\omega_k) = \pm\Delta \cdot \omega_k \pm \pi I(\omega_k), \tag{1.63}$$

where $I(\omega_k)$ is an indicator function which takes on values 0 or 1 over the points $\omega_k, k = 0, 1, 2, \ldots, N-1$.

A zero-phase signal is thus a linear phase signal for which the phase-slope Δ is zero.

Convolution theorem

For any $x, y \in \mathbb{C}^N$

$$\boxed{x * y \leftrightarrow X \cdot Y} \tag{1.64}$$

$$\begin{aligned}
DFT_k(x*y) &\triangleq \sum_{n=0}^{N-1} (x*y)_n e^{-j2\pi nk/N} \\
&= \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} x(m) y(n-m) e^{-j2\pi nk/N} \\
&= \sum_{m=0}^{N-1} x(m) \sum_{n=0}^{N-1} y(n-m) e^{-j2\pi nk/N} \\
&= \sum_{m=0}^{N-1} x(m) \left(e^{-j2\pi mk/N} Y(k) \right) \text{ by the shift theorem} \\
&= \left(\sum_{n=0}^{N-1} x(m) e^{-j2\pi mk/N} \right) Y(k) \triangleq X(k)Y(k)
\end{aligned} \tag{1.65}$$

This is perhaps the most important single Fourier theorem of all. It is the basis of a large number of applications of the FFT. Since the FFT provides a fast Fourier transform, it also provides fast convolution, thanks to the convolution theorem. It turns out that using the FFT to perform convolution is really more efficient in practice only for reasonably long convolutions, such as $N > 100$. For much longer convolutions, the savings become enormous compared with "direct" convolution. This happens because direct convolution requires on the order of N^2 operations (multiplications and additions), while FFT-based convolution requires on the order of $N \lg N$ operations, where lg denotes the logarithm-base-2 of N.

The dual of the convolution theorem says that multiplication in the time domain is convolution in the frequency domain:

$$x \cdot y \leftrightarrow \frac{1}{N} X * Y \tag{1.66}$$

where convolution at the right hand side is the cyclic convolution, since X and Y are periodic with period equal to N samples. This theorem also bears on the use of FFT windows. It implies that windowing in the time domain corresponds to smoothing in the frequency domain. That is, the spectrum of $w \cdot x$ is simply X filtered by W, or, $W * Y$. This smoothing reduces sidelobes associated with the rectangular window (which is the window one gets implicitly when no window is explicitly used).

Correlation theorem

For any $x, y \in \mathbb{C}^N$

$$\boxed{x \star y \leftrightarrow \bar{X} \cdot Y} \tag{1.67}$$

Power theorem

For any $x, y \in \mathbb{C}^N$

$$\langle x, y \rangle = \frac{1}{N} \langle X, Y \rangle \tag{1.68}$$

Proof:

$$\langle x, y \rangle \triangleq \sum_{n=0}^{N-1} x(n)\bar{y}(n) = (x \star y)_0 = DFT_0^{-1}(\bar{Y} \cdot X) = \frac{1}{N} \sum_{k=0}^{N-1} X(k)\bar{Y}(k) \triangleq \frac{1}{N} \langle X, Y \rangle \tag{1.69}$$

Rayleigh energy theorem (Parseval theorem)

For any $x \in \mathbb{C}^N$

$$\|x\|^2 = \frac{1}{N} \|X\|^2 \tag{1.70}$$

i.e.

$$\boxed{\sum_{n=0}^{N-1} |x(n)|^2 = \frac{1}{N} \sum_{k=0}^{N-1} |X(k)|^2} \tag{1.71}$$

1.8 Fourier transforms for Continuous/Discrete Time/Frequency

The Fourier transform can be defined for signals which are

- discrete or continuous in time, and
- finite or infinite in duration.

This results in four cases. Quite naturally, the frequency domain has the same four cases,

- discrete or continuous in frequency,
- finite or infinite in bandwidth.

When time is discrete, the frequency axis is finite, and vice versa. Table 1.1 summarizes all four Fourier-transform cases corresponding to discrete or continuous time and/or frequency.

Time duration	.	.
Finite	Infinite	
Discrete FT (DFT) $X(k) = \sum_{n=0}^{N-1} x(n)e^{-j\omega_k n}$ $k = 0, 1, \ldots, N-1$	Discrete Time FT (DTFT) $X(\omega) = \sum_{-\infty}^{+\infty} x(n)e^{-j\omega n}$ $\omega \in [-\pi, +\pi)$	discr. time t
Fourier Series (FS) $X(k) = \frac{1}{P}\int_{-\infty}^{+\infty} x(t)e^{-j\omega_k t}dt$ $k = -\infty, \ldots, +\infty$	Fourier Transform (FT) $X(\omega) = \int_{-\infty}^{+\infty} x(t)e^{-j\omega t}dt$ $\omega \in (-\infty, +\infty)$	cont. time t
discrete freq. k	continuous freq. ω	

Table 1.1. Four cases of the sampled/continuous time and frequency

Discrete Time Fourier Transform (DTFT)

The Discrete Time Fourier Transform (DTFT) can be viewed as the limiting form of the DFT when its length N is allowed to approach infinity:

$$\boxed{X(\omega) = \sum_{-\infty}^{+\infty} x(n)e^{-j\omega n}} \tag{1.72}$$

where $\omega \in [-\pi, +\pi)$ denotes the continuous radian frequency variable, and $x(n)$ is the signal amplitude at sample number n.

The inverse DTFT is

$$\boxed{x(n) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} X(\omega) e^{j\omega n} d\omega} \tag{1.73}$$

Instead of operating on sampled signals of length N (like the DFT), the DTFT operates on sampled signals $x(n)$ defined over all integers $n \in \mathbb{Z}$. As a result, the DTFT frequencies form a continuum. That is, the DTFT is a function of continuous frequency $\omega \in [-\pi, +\pi)$, while the DFT is a function of discrete frequency ω_k, $k \in [0, N-1]$. The DFT frequencies $\omega_k = 2\pi k/N$, $k = 0, 1, 2, \ldots, N-1$, are given by the angles of N points uniformly distributed along the unit circle in the complex plane. Thus, as $N \to \infty$, a continuous frequency axis must result in the limit along the unit circle in the z plane. The axis is still finite in length, however, because the time domain remains sampled.

References

- Smith, J.O. Mathematics of the Discrete Fourier Transform (DFT), 2003, ISBN 0-9745607-0-7,

  `http://ccrma.stanford.edu/~jos/mdft/`

2

Introduction to digital filters

In this chapter, the important concepts of linearity and time-invariance (LTI) are discussed. The great majority of audio filters are LTI, for several reasons: first, no new spectral components are introduced by LTI filters. Time-varying filters, on the other hand, can generate audible sideband images of the frequencies present in the input signal (when they vary at audio rates). Time-invariance is not overly restrictive, however, because the static analysis holds very well for filters that change slowly with time. (One rule of thumb is that a filter's coefficients should be substantially constant over its impulse-response duration.) Nonlinear filters create new sinusoidal components at all sums and differences of the frequencies present in the input signa. This includes both harmonic distortion (when the input signal is periodic) and intermodulation distortion (when at least two inharmonically related tones are present). A truly linear filter does not cause harmonic or intermodulation distortion.

In the following sections, linearity and time-invariance will be formally introduced, together with some elementary mathematical aspects of signals.

2.1 Digital filters

A real digital filter $\mathcal{T}_n$ is defined as any real-valued function of a signal for each integer n.

Thus, a real digital filter maps every real, discrete-time signal to a real, discrete-time signal. A complex filter, on the other hand, may produce a complex output signal even when its input signal is real.

We may express the input-output relation of a digital filter by the notation

$$y(n) = \mathcal{T}_n\{x(\cdot)\} \tag{2.1}$$

Linear filters

A filter $\mathcal{L}_n$ is said to be linear if for any pair of signals $x_1(\cdot)$, $x_2(\cdot)$ and for all constant gains g, we have

$$\text{Scaling}: \quad \mathcal{L}_n\{gx_1(\cdot)\} = g\mathcal{L}_n\{x_1(\cdot)\} \tag{2.2}$$

$$\text{Superposition}: \quad \mathcal{L}_n\{x_1(\cdot) + x_2(\cdot)\} = \mathcal{L}_n\{x_1(\cdot)\} + \mathcal{L}_n\{x_2(\cdot)\} \tag{2.3}$$

The scaling property of linear systems states that scaling the input of a linear system (multiplying it by a constant gain factor) scales the output by the same factor. The superposition property of linear systems states that the response of a linear system to a sum of signals is the sum of the responses to each individual input signal. Another view is that the individual signals which have been summed at the input are processed independently inside the filter. They superimpose and do not interact. (The addition of two signals, sample by sample, is like converting stereo to mono by mixing the two channels together equally.)

Another example of a linear signal medium is the earth's atmosphere. When two sounds are in the air at once, the air pressure fluctuations that convey them simply add (unless they are extremely loud). Since any finite continuous signal can be represented as a sum (i.e., superposition) of sinusoids, we can predict the filter response to any input signal just by knowing the response for all sinusoids. Without superposition, we have no such general description and it may be impossible to do any better than to catalog the filter output for each possible input.

Time invariant filters

In plain terms, a time-invariant filter (or shift-invariant filter) is one which performs the same operation at all times. What we want to say is that if the input signal is delayed (shifted) by, say, N samples, then the output waveform is simply delayed by N samples and unchanged otherwise. Thus $y(\cdot)$, the output waveform from a time-invariant filter, merely shifts forward or backward in time as the input waveform $x(\cdot)$ is shifted forward or backward in time.

A filter $\mathcal{L}_n$ is said to be time-invariant if

$$\boxed{\mathcal{L}_n\{\text{SHIFT}_N\{x\}\} = \mathcal{L}_{n-N}\{x(\cdot)\} = y(n-N) = \text{SHIFT}_{N,n}\{y\}} \tag{2.4}$$

In the rest of these notes, all filters discussed will be linear and time-invariant. For brevity, these will be referred to as LTI filters.

2.2 Time domain filter representations

Difference equation

The difference equation is a formula for computing an output sample at time n based on past and present input samples and past output samples in the time domain. We may write the general LTI difference equation as follows:

$$\begin{aligned} y(n) = b_0 x(n) + b_1 x(n-1) + \ldots + b_M x(n-M) \\ - a_1 y(n-1) - \ldots - a_N y(n-N) \end{aligned} \tag{2.5}$$

$$\boxed{y(n) = \sum_{i=0}^{M} b_i x(n-i) - \sum_{j=1}^{N} a_j y(n-j)} \tag{2.6}$$

where x is the input signal, y is the output signal, and the constants b_i, $i = 0, 1, 2, \ldots, M$, a_j, $j = 1, 2, \ldots, N$ are the filter coefficients.

Notice that a filter of the form of (2.6) can use "past" output samples (such as $y(n-1)$ in the calculation of the "present" output $y(n)$. This use of past output samples is called *feedback*. Any filter having one or more feedback paths ($N > 0$) is called *recursive*.

A filter is said to be recursive if and only if $a_i \neq 0$ for some $i > 0$. Recursive filters are also called *infinite-impulse-response (IIR)* filters. When there is no feedback ($a_i = 0, \forall i > 0$), the filters is said to be a *nonrecursive* or *finite-impulse-response (FIR)* digital filter.

Listing 2.1. Matlab

```
N = 1000;

%a(1)*y(n) = -a(2)y(n - 1) + b(1)x(n) + b(2)x(n - 1);
b = [1  1];
a = [1  -0.9];

x = randn(N, 1);
y = filter(b, a, x);
```

Signal flow graph

One possible signal flow graph (or system diagram) for (2.6) is given in Figure 2.1 for the case of $M = 2$ and $N = 2$. Hopefully, it is easy to see how this diagram represents the difference equation (a box labeled "z^{-1}" denotes a one-sample delay in time).

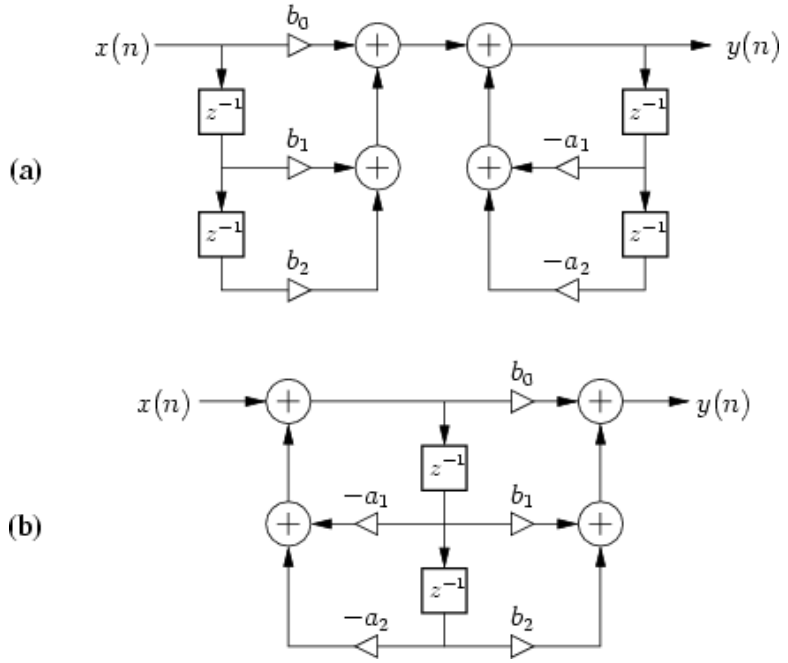


Fig. 2.1. System diagram for the filter difference equation $y(n) = b_0x(n)+b_1x(n-1)-a_1y(n-1)-a_2y(n-2)$. (a) Direct form I. (b) Direct form II

Causal recursive filters

Equation (2.6) does not cover all LTI filters, for it represents only causal LTI filters. A filter is said to be causal when its output does not depend on any "future" inputs.

For example, $y(n) = x(n+1)$ is a non-causal filter because the output anticipates the input one sample into the future. Restriction to causal filters is quite natural when the filter operates in real time. Many digital filters, on the other hand, are implemented on a computer where time is artificially represented by an array index. Thus, noncausal filters present no difficulty in such an "off-line" situation. It happens that the analysis for noncausal filters is pretty much the same as that for causal filters, so we can easily relax this restriction.

Filter order

The maximum delay, in samples, used in creating each output sample is called the order of the filter. In the difference-equation representation, the order is the larger of M and N in (2.6).

Impulse response representation

In addition to difference-equation coefficients, any LTI filter may be represented in the time domain by its response to a specific signal called the *impulse*. This response is called, naturally enough, the impulse response of the filter. Any LTI filter can be implemented by convolving the input signal with the filter impulse response.

The impulse signal is denoted $\delta(n)$ and defined by

$$\delta(n) = \begin{cases} 1, n = 0 \\ 0, n \neq 0 \end{cases} \tag{2.7}$$

The *impulse response* of a filter is the response of the filter to $\delta(n)$ and it is most often denoted $h(n)$

$$h(n) \triangleq \mathcal{L}_n\{\delta(\cdot)\} \tag{2.8}$$

A filter is said to be *stable* is the impulse response $h(n)$ approaches zero as n goes to infinity.

Every finite-order nonrecursive filter is stable. Only the feedback coefficients a_j in (2.6) can cause instability.

Listing 2.2. Matlab

```
N = 100;

b = [1  1];
a = [1  -0.9];

delta = [1; zeros(N-1,1)]';
h = filter(b, a, delta);

stem(h);
```

Convolution representation

Using the basic properties of linearity and time-invariance, we will derive the convolution representation which gives an algorithm for implementing the filter directly in terms of its impulse response. In other words, *the output $y(n)$ of any LTI filter (including recursive LTI filters) may be computed by convolving the input signal with the filter impulse response.*

The convolution formula plays the role of the difference equation when the impulse response is used in place of the difference-equation coefficients as a filter representation. In fact, we will find that, for FIR filters (nonrecursive, i.e., no feedback), the difference

equation and convolution representation are essentially the same thing. For recursive filters, one can think of the convolution representation as the difference equation with all feedback terms "expanded" to an infinite number of feedforward terms.

The first step is to express an arbitrary signal $x(\cdot)$ as a linear combination of shifted impulses, i.e.,

$$x(n) = \sum_{i=-\infty}^{+\infty} x(i)\delta(n-i) \triangleq (x * \delta)(n) \tag{2.9}$$

Equation (2.9) expresses a signal as a linear combination (or weighted sum) of impulses. That is, each sample may be viewed as an impulse at some amplitude and time. As we have already seen, each impulse (sample) arriving at the filter's input will cause the filter to produce an impulse response. If another impulse arrives at the filter's input before the first impulse response has died away, then the impulse response for both impulses will superimpose (add together sample by sample). More generally, since the input is a linear combination of impulses, the output is the same linear combination of impulse responses. This is a direct consequence of the superposition principle which holds for any LTI filter.

We repeat this in more precise terms. First linearity is used and then time-invariance is invoked. We can express the output of any linear (and possibly time-varying) filter by

$$\begin{aligned} y(n) &= \mathcal{L}_n\{x(\cdot)\} \\ &= \mathcal{L}_n\{(x * \delta)(\cdot)\} \\ &= \mathcal{L}_n\{\sum_{i=-\infty}^{+\infty} x(i)\delta(\cdot - i)\} \\ &= \sum_{i=-\infty}^{+\infty} x(i)h(n,i) \end{aligned} \tag{2.10}$$

where we have written $h(n,i) \triangleq \mathcal{L}_n\{\delta(\cdot - i)\}$ to denote the filter response at time n to an impulse which occurred at time i.

If in addition to being linear, the filter is time-invariant, then $h(n,i) = h(n-i)$, which allows us to write

$$y(n) = \sum_{i=-\infty}^{+\infty} x(i)h(n-i) \triangleq (x * h)(n) \tag{2.11}$$

This states that the filter output $y(n)$ is the convolution of the input $x(n)$ with the filter impulse response $h(n)$. The infinite sum can be replaced by more typical practical limits. By choosing time 0 as the beginning of the signal, we may define $x(n)$ to be 0 for $n < 0$ so that the lower summation limit of can be replaced by 0. Also, if the filter

is causal, we have $h(n) = 0$ for $n < 0$, so the upper summation limit can be written as instead n of ∞. This gets us down to the ultimate convolution representation of a linear, time-invariant, causal digital filter:

$$y(n) = \sum_{i=0}^{n} x(i)h(n-i) \triangleq (x * h)(n) \tag{2.12}$$

Since the above equation is a convolution, and since convolution is commutative, we can rewrite it as

$$y(n) = \sum_{i=0}^{n} h(i)x(n-i) \triangleq (x * h)(n) \tag{2.13}$$

or

$$y(n) = h(0)x(n) + h(1)x(n-1) + h(2)x(n-2) + \ldots + h(n)x(0) \tag{2.14}$$

This latter form looks more like the general difference equation presented in (2.6).

It is instructive to compare this method of filter implementation to the use of difference equations. If there is no feedback, then the difference equation and the convolution formula are identical; in this case, $h(i) = b_i$ and there are no a_j coefficients in (2.6). For recursive filters, we can convert the difference equation into a convolution by calculating the filter impulse response. However, this can be rather tedious, since with nonzero feedback coefficients the impulse response generally lasts forever. Of course, for stable filters the response is infinite only in theory; in practice, one may simply truncate the response after an appropriate length of time, such as after it falls below the quantization noise level due to round-off error.

Listing 2.3. Matlab

```
N = 100;
x = rand(N,1);

b = [1 1];
y1 = conv(b, x);
% this is equivalent to
% y2 = filter(b, 1, x);
% but length(y1) = length(x) + length(b) - 1
% and length(y2) = length(x)

a = [1 -0.9];
y3 = filter(1, a, x);

% approximation by truncating impulse response
% at M samples
M = 100;
delta = [1; zeros(M-1,1)];
h = filter(1, a, delta);
y4 = conv(h, x);

plot(y3)
hold on
plot(y4,'r--')
hold off
```

2.3 Transfer function analysis

This section discusses filter transfer functions and associated analysis. The transfer function provides an algebraic representation of a linear, time-invariant (LTI) filter in the frequency domain:

The transfer function of a linear time-invariant discrete-time filter is defined as $Y(z)/X(z)$, where $Y(z)$ denotes the z transform of the filter output signal $y(n)$, and $X(z)$ denotes the z transform of the filter input signal $x(n)$.

Let $h(n)$ denote the impulse response of the filter. It turns out (as we will show) that the transfer function is equal to the z transform of the impulse response $h(n)$:

$$\boxed{H(z) = \frac{Y(z)}{X(z)}} \tag{2.15}$$

The z transform

The bilateral z transform of the discrete-time signal is defined to be

$$X(z) \triangleq \sum_{-\infty}^{+\infty} x(n) z^{-n} \tag{2.16}$$

where z is a complex variable.

Since signals are typically defined to begin (become nonzero) at time $n = 0$, and since filters are often assumed to be causal, the lower summation limit given above may be written as 0 rather than $-\infty$ to yield the unilateral z transform:

$$\boxed{X(z) \triangleq \sum_{n=0}^{+\infty} x(n) z^{-n}} \tag{2.17}$$

We think of this as a "function to function" mapping. We express the fact that X is the z transform of x by writing

$$X \leftrightarrow x \tag{2.18}$$

The z transform of a signal x can be regarded as a polynomial in z^{-1}, with coefficients given by the signal samples. For example, the finite-duration signal

$$x(n) = \begin{cases} n+1, & 0 \le n \le 2 \\ 0, & \text{otherwise} \end{cases} \tag{2.19}$$

has the z transform $X(z) = 1 + 2z^{-1} + 3z^{-2} = 1 + 2z^{-1} + 3(z^{-1})^2$.

Shift theorem

The shift theorem says that a delay of Δ samples in the time domain corresponds to a multiplication by $z^{-\Delta}$ in the frequency domain

$$\mathcal{Z}_z\{\text{SHIFT}_\Delta\{x\}\} = z^{-\Delta}X(z) \tag{2.20}$$

or, using more common notation

$$\boxed{x(n-\Delta) \leftrightarrow z^{-\Delta}X(z)} \tag{2.21}$$

Convolution theorem

The convolution theorem for z transforms states that for any (real or complex) signals x and y, convolution in the time domain is multiplication in the z domain, i.e.,

$$\boxed{x * y \leftrightarrow X \cdot Y} \tag{2.22}$$

From equation (2.11), we have that the output y from a LTI filter with input x and impulse response h is given by the convolution of h and x, i.e.

$$y(n) = (h * x)(n) \tag{2.23}$$

Taking the z transform of both sides and applying the convolution theorem gives

$$Y(z) = H(z)X(z) \tag{2.24}$$

where $H(z)$ is the z transform of the filter impulse response. We may divide equation (2.24) by $X(z)$ to obtain

$$\boxed{H(z) = \frac{Y(z)}{X(z)} \triangleq \text{transfer function}} \tag{2.25}$$

This shows that, as a direct result of the convolution theorem, the z transform of an impulse response $h(n)$ is equal to the transfer function $H(z)$ of the filter, provided the filter is linear and time invariant.

Z transform of difference equations

Using linearity and the shift theorem we can write down the z transform of any difference equation by inspection.

$$\begin{aligned} y(n) = b_0x(n) + b_1x(n-1) + \ldots + b_Mx(n-M) \\ - a_1y(n-1) - \ldots - a_Ny(n-N) \end{aligned} \tag{2.26}$$

Let's take the z transform of both sides. Because $\mathcal{Z}\{\}$ is a linear operator, it might be distributed through the terms on the right hand side as follows:

$$\begin{aligned} \mathcal{Z}\{y(n)\} = Y(z) = b_0X(z) + b_1z^{-1}X(z) + \ldots + b_Mz^{-M}X(z) \\ -a_1z^{-1}Y(z) - \ldots - a_Nz^{-N}Y(z), \end{aligned} \tag{2.27}$$

Factoring out common terms $Y(z)$ and $X(z)$ gives

$$Y(z)[1 + a_1z^{-1} + \ldots + a_Nz^{-N}] = X(z)[b_0 + b_1z^{-1} + \ldots + b_Mz^{-M}] \tag{2.28}$$

Defining the polynomials

$$A(z) \triangleq 1 + a_1z^{-1} + \ldots + a_Nz^{-N} \tag{2.29}$$

$$B(z) \triangleq b_0 + b_1z^{-1} + \ldots + b_Mz^{-M} \tag{2.30}$$

the z transform of the difference equation becomes

$$A(z)Y(z) = B(z)X(z) \tag{2.31}$$

Finally, solving for $H(z)$

$$\boxed{H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1z^{-1} + \ldots + b_Mz^{-M}}{1 + a_1z^{-1} + \ldots + a_Nz^{-N}} \triangleq \frac{B(z)}{A(z)}} \tag{2.32}$$

Factored form

By the fundamental theorem of algebra, every Nth order polynomial can be factored into a product of N first-order polynomials. Therefore, equation (2.24) can be written in factored form as

$$\boxed{H(z) = b_0\frac{(1 - q_1z^{-1})(1 - q_2z^{-1}) \cdots (1 - q_Mz^{-1})}{(1 - p_1z^{-1})(1 - p_2z^{-1}) \cdots (1 - q_Nz^{-1})}} \tag{2.33}$$

The numerator roots $\{q_1, q_2, \ldots, q_M\}$ are called the *zeros* of the transfer function, and the denominator roots $\{p_1, p_2, \ldots, p_N\}$ are called the *poles* of the filter.

Listing 2.4. Matlab

```
rho = 0.9;
theta = pi/8;
a = [1 2*rho*cos(theta) rho^2];
b = [1 2 1];

p = roots(a)
z = roots(b)

zplane(b,a)

a2 = poly(p)
b2 = poly(z)
```

Series and parallel transfer functions

- Transfer functions of filters in series multiply together (Figure 2.2)

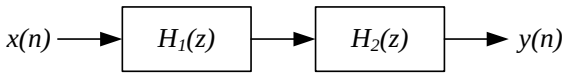


Fig. 2.2. Series combination of transfer functions $H_1(z)$ and $H_2(z)$ to produce $H_1(z)H_2(z)$

  Remark: series combination is commutative, i.e.: $H_1(z)H_2(z) = H_2(z)H_1(z)$. Note, however, that the numerical performance of the overall filter is usually affected by the ordering of filter stages in a series combination.

- Transfer functions of parallel filters sum together (Figure 2.3)

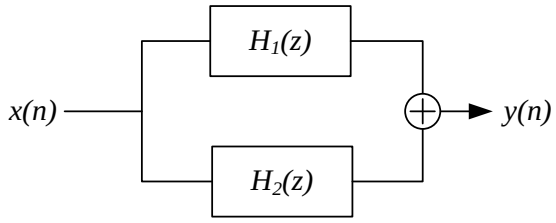


Fig. 2.3. Parallel combination of transfer functions $H_1(z)$ and $H_2(z)$ to produce $H_1(z) + H_2(z)$

Partial Fraction Expansion (PFE)

The partial fraction expansion can be used to expand any rational z transform

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + \ldots + b_M z^{-M}}{1 + a_1 z^{-1} + \ldots + a_N z^{-N}} \tag{2.34}$$

as a sum of first-order terms

$$H(z) = \frac{B(z)}{A(z)} = \sum_{i=1}^{N} \frac{r_i}{1 - p_i z^{-1}} \tag{2.35}$$

for $M < N$, and

$$H(z) = \frac{B(z)}{A(z)} = F(z) + z^{-(K+1)} \sum_{i=1}^{N} \frac{r_i}{1 - p_i z^{-1}} \tag{2.36}$$

for $N \geq M$, where the term $z^{-(K+1)}$ is optional but it is often preferred.

The PFE procedure occurs in two or three steps:

- When $M \geq N$, perform a long division to obtain an FIR part $F(z)$ and a strictly proper IIR part $B'(z)/A(z)$
- Find the N poles p_i, $i = 1, \ldots, N$ (roots of $A(z)$)
- If the poles are distinct, find the N residues r_i, $i = 1, \ldots, N$ from

$$r_i = (1 - p_i z^{-1}) \frac{B(z)}{A(z)} \bigg|_{z=p_i} \tag{2.37}$$

A more general procedure is needed when there are repeated poles, and the general form expression of the PFE becomes

$$H(z) = \frac{B(z)}{A(z)} = F(z) + z^{-(K+1)} \sum_{i=1}^{N_p} \sum_{k=1}^{m_i} \frac{r_{i,k}}{(1 - p_i z^{-1})^k} \tag{2.38}$$

where N_p denotes the number of distinct poles, and $m_i \geq 1$ denotes the multiplicity of the ith pole

Listing 2.5. Matlab

```
p = [0.99; 0.99*exp(j*pi/8);
0.99*exp(-j*pi/8)];
z = [-1; exp(j*pi/16); exp(-j*pi/16)];
a = poly(p);
b = poly(z);
zplane(b, a);
% also:
% zplane(z, p); % note: a, b are row vectors, z, p are column vectors

[r, p, k] = residuez(b, a)
```

2.4 Frequency response analysis

Frequency response

The frequency response of an LTI filter is defined as the spectrum of the output signal divided by the spectrum of the input signal.

We have

$$Y(z) = H(z)X(z) \tag{2.39}$$

A basic property of the z transform is that, over the unit circle $z = e^{j\omega T}$, we find the spectrum. To show this, we set $z = e^{j\omega T}$ in the definition of the z transform, to obtain

$$X(e^{j\omega T}) = \sum_{-\infty}^{+\infty} x(n)e^{-j\omega nT} \tag{2.40}$$

which may be recognized as the definition of the Discrete Time Fourier Transform (DTFT).

Applying this relation to $Y(z) = H(z)X(z)$ gives

$$Y(e^{j\omega T}) = H(e^{j\omega T})X(e^{j\omega T}) \tag{2.41}$$

The frequency response of a linear time-invariant filter equals the transfer function $H(z)$ evaluated on the unit circle in the z plane, that is $H(e^{j\omega T})$.

This immediately implies that the frequency response is the DTFT of the impulse response

$$\boxed{H(e^{j\omega T}) = DTFT_{\omega}(h)} \tag{2.42}$$

The frequency response specifies the gain and phase shift applied by the filter at each frequency.

The frequency response may be decomposed into two real-valued functions, the amplitude response and the phase response. Formally, we may define them as follows:

- Amplitude response

  The amplitude response $G(\omega)$ of a LTI filter is defined as the magnitude of the filter frequency response $H(e^{j\omega T})$

  $$G(\omega) \triangleq |H(e^{j\omega T})| \tag{2.43}$$

  The real-valued amplitude response $G(\omega)$ specifies the amplitude gain that the filter provides at each frequency.

- Phase response

  The phase response $\Theta(\omega)$ of an LTI filter is defined as the phase (or complex angle) of the frequency response $H(e^{j\omega T})$

$$\Theta(\omega) \triangleq \angle H(e^{j\omega T}) \tag{2.44}$$

  The real-valued phase response $\Theta(\omega)$ gives the phase shift in radians that each input component sinusoid will undergo.

The frequency response can be expressed in polar form in terms of the amplitude and phase response

$$H(e^{j\omega T}) = G(\omega)e^{j\Theta(\omega)} \tag{2.45}$$

From $H(z) = B(z)/A(z)$ we have that the amplitude and the phase responses can be expressed as

$$G(\omega) = \frac{|B(e^{j\omega T})|}{|A(e^{j\omega T})|} \tag{2.46}$$

$$\Theta(\omega) = \angle B(e^{j\omega T}) - \angle A(e^{j\omega T}) \tag{2.47}$$

Computing the frequency response using the DFT

In practice, we must work with a sampled frequency axis. That is, instead of evaluating the transfer function $H(z) = B(z)/A(z)$ at $e^{j\omega T}$ to obtain the frequency response $H(e^{j\omega T})$, where ω is the continuous radian frequency, we compute instead

$$H(e^{j\omega_k T}) = \frac{B(e^{j\omega_k T})}{A(e^{j\omega_k T})}, \quad e^{j\omega_k T} \triangleq e^{j2\pi k/N_s}, \quad k = 0, 1, \ldots, N_s - 1 \tag{2.48}$$

where N_s is the desired number of spectral samples.

The transfer function can be written as

$$H(e^{j\omega T}) = \frac{B(e^{j\omega T})}{A(e^{j\omega T})} = \frac{DTFT_\omega(B)}{DTFT_\omega(A)} \tag{2.49}$$

Sampling the DTFT at $\omega = \omega_k$, $k = 0, 1, \ldots, N_s - 1$ we get the Discrete Fourier Transform (DFT). Thus, we can write

$$H(e^{j\omega_k T}) = \frac{DFT_{\omega_k}(B)}{DFT_{\omega_k}(A)}, \quad k = 0, 1, \ldots, N_s - 1 \tag{2.50}$$

where $\omega_k = 2\pi k/N_s$.

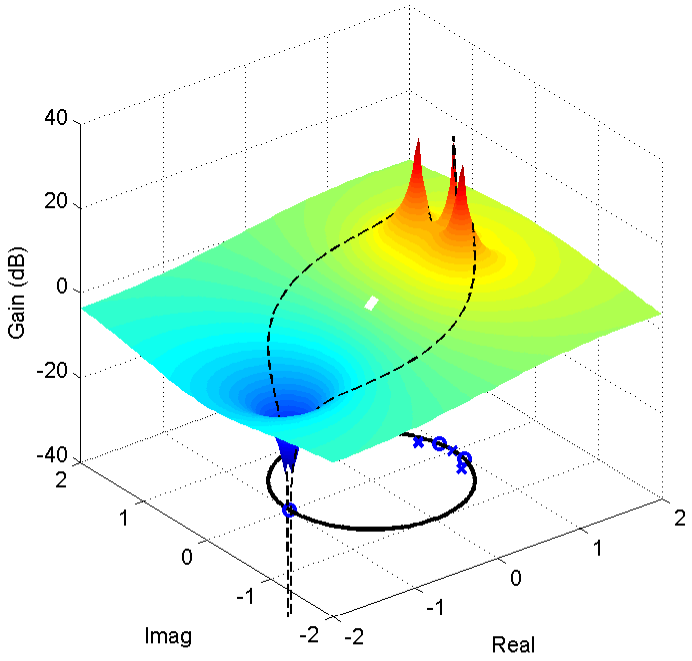


Fig. 2.4. *Z*-transform and frequency response of the filter used in Listing 2.6

Listing 2.6. Matlab

```
p = [0.99; 0.99*exp(j*pi/8); 0.99*exp(-j*pi/8)];
z = [-1; exp(j*pi/16); exp(-j*pi/16)];
a = poly(p); b = poly(z);

Ns = 1024;
[H, w] = freqz(b, a, N2);

figure(1)
subplot(2,1,1) plot(w, 10*log10(abs(H).^2));
axis([0 pi -50 50])
subplot(2,1,2) plot(w, phase(H))
axis([0 pi -pi pi])

% equivalently, the frequency response
% is computed as follows
B = fft(b, 2*Ns);
A = fft(a, 2*Ns);
w = 2*pi*[0:2*Ns - 1]/(2*Ns);
H = B./A;

figure(2)
subplot(2,1,1)
plot(w, 10*log10(abs(H).^2));
axis([0 pi -50 50])
subplot(2,1,2)
plot(w, phase(H))
axis([0 pi -pi pi])
```

To avoid undersampling, $N_s \geq M$ and $N_s \geq N$. The impulse response $h(n)$ obtained from the sampled frequency response will be

$$\tilde{h}(n) = IDFT_n(H) = \frac{1}{N_s}\sum_{k=0}^{N_s-1} H(e^{j\omega_k T})e^{j\omega_k nT} \tag{2.51}$$

It will be time aliased in the IIR case, i.e. $\tilde{h} \neq h$. In other words, an infinitely long impulse response cannot be Fourier transformed using a finite-length DFT, and this corresponds to not being able to sample the frequency response of an IIR filter without some loss of information. In practice, we simply choose N_s sufficiently large so that the sampled frequency response is accurate enough for our needs. A conservative practical rule of thumb when analyzing stable digital filters is to choose $N_s > 7/(1-R_{\max})$, where $R_{\max}$ denotes the maximum pole magnitude.

Graphical frequency response

Consider the frequency response of a LTI filter expressed in factored form

$$H(e^{j\omega T}) = g\frac{(1-q_1e^{-j\omega T})(1-q_2e^{-j\omega T})\cdots(1-q_Me^{-j\omega T})}{(1-p_1e^{-j\omega T})(1-p_2e^{-j\omega T})\cdots(1-q_Ne^{-j\omega T})} \tag{2.52}$$

Consider first the amplitude response $G(\omega) \triangleq |H(e^{j\omega T})|$

$$\begin{aligned} G(\omega) &= |g|\frac{|1-q_1e^{-j\omega T}|\cdot|1-q_2e^{-j\omega T}|\cdots|1-q_Me^{-j\omega T}|}{|1-p_1e^{-j\omega T}|\cdot|1-p_2e^{-j\omega T}|\cdots|1-q_Ne^{-j\omega T}|} \\ &= |g|\frac{|e^{j\omega T}-q_1|\cdot|e^{j\omega T}-q_2|\cdots|e^{j\omega T}-q_M|}{|e^{j\omega T}-p_1|\cdot|e^{j\omega T}-p_2|\cdots|e^{j\omega T}-p_N|} \end{aligned} \tag{2.53}$$

In the z plane, each term in the previous equation is the length of a vector drawn from a pole or zero to a single point on the unit circle, as shown in Figure 2.5 for two poles and two zeros. The gain of this two-pole two-zero filter is $G(\omega) = (d_1d_2)/(d_3d_4)$.

The phase response is almost as easy to evaluate graphically as is the amplitude response:

$$\begin{aligned} \Theta(\omega) &\triangleq \angle\left\{g\frac{(1-q_1e^{-j\omega T})(1-q_2e^{-j\omega T})\cdots(1-q_Me^{-j\omega T})}{(1-p_1e^{-j\omega T})(1-p_2e^{-j\omega T})\cdots(1-q_Ne^{-j\omega T})}\right\} \\ &= \angle g + (N-M)\omega T + \angle(e^{j\omega T}-q_1) + \angle(e^{j\omega T}-q_2) + \ldots + \angle(e^{j\omega T}-q_M) \\ &\quad - \angle(e^{j\omega T}-p_1) - \angle(e^{j\omega T}-p_2) - \ldots - \angle(e^{j\omega T}-p_N) \end{aligned} \tag{2.54}$$

The angle of $e^{j\omega T} - z$ is the angle of the constructed vector (where a vector pointing horizontally to the right has an angle of 0). The angles of the lines from the zeros are added, and the angles of the lines from the poles are subtracted. Thus, at the frequency ω the phase response of the two-pole two-zero filter in the figure is $\Theta(\omega) = \theta_1 + \theta_2 - \theta_3 - \theta_4$.

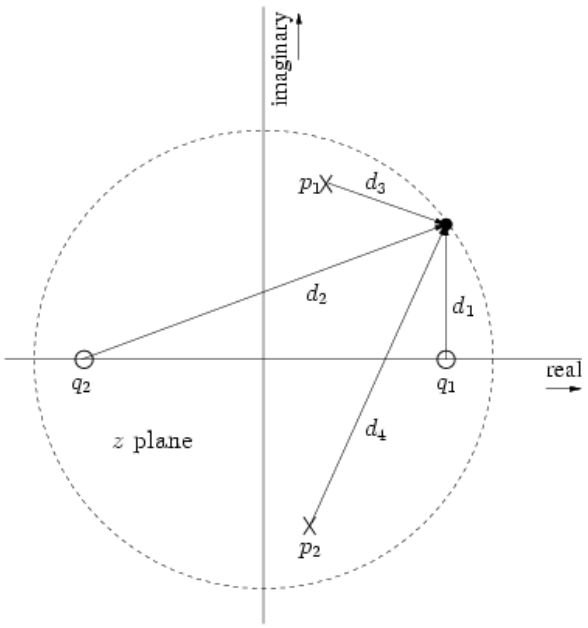


Fig. 2.5. Measurement of amplitude response from a pole-zero diagram. A pole is represented in the complex plane by 'x', a zero by 'o'

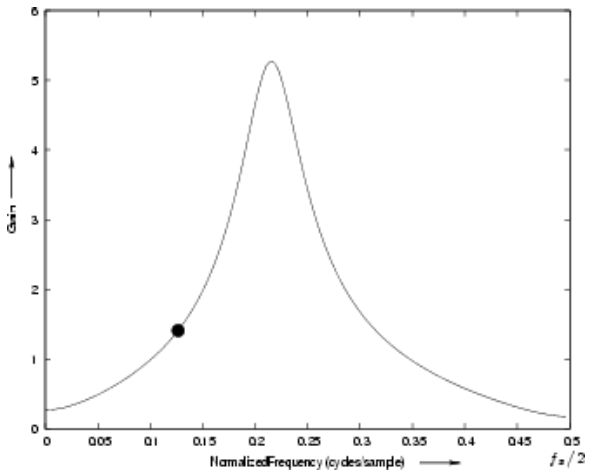


Fig. 2.6. Amplitude response obtained by traversing the entire upper semicircle in Figure 2.5.

Stability

A filter is said to be stable if its impulse response $h(n)$ decays to 0 as n goes to infinity.

In terms of poles and zeros, an irreducible filter transfer function is stable if and only if *all the poles* are inside the unit circle in the plane. This is because the transfer function is the z transform of the impulse response, and if there is an observable pole outside the unit circle, then there is an exponentially increasing component of the impulse response.

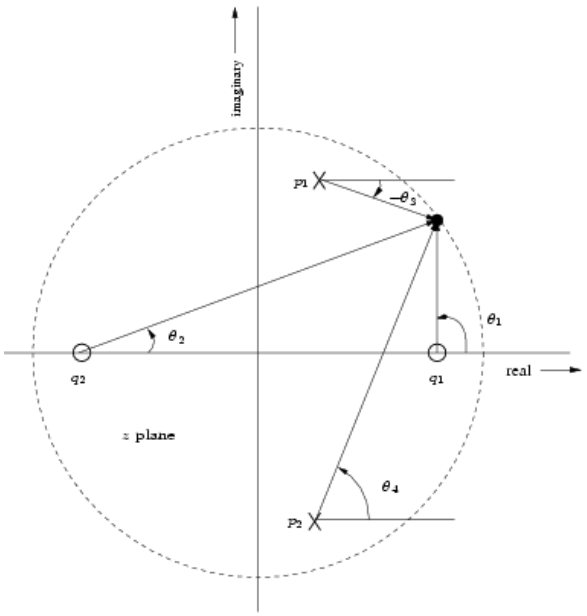


Fig. 2.7. Measurement of phase response from a pole-zero diagram.

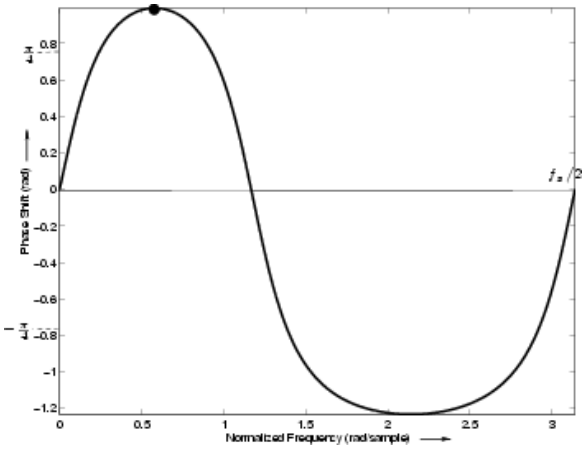


Fig. 2.8. Phase response obtained from Figure 2.8 for positive frequencies.

Poles on the unit circle may be called marginally stable. The impulse response component corresponding to a pole on the unit circle never decays, but neither does it grow. In physical modeling applications, marginally stable poles occur often in lossless systems, such as ideal vibrating strings.

Phase response analysis

In the previous sections we showed that the frequency response of a digital LTI filter can be decomposed into the amplitude response $G(\omega)$ times the phase response $e^{j\Theta(\omega)}$.

In the following we look at two alternative forms of the phase response: phase delay and group delay.

Phase delay

The phase response $\Theta(\omega)$ of an LTI filter gives the radian phase shift experienced by each sinusoidal component of the input signal. In is often more intuitive to consider instead the phase delay, defined as

$$P(\omega) \triangleq -\frac{\Theta(\omega)}{\omega} \tag{2.55}$$

The phase delay gives the time delay in seconds experienced by each sinusoidal component of the input signal. Consider, for example, the filter $H(z) = 1 + z^{-1}$. Its phase response is $\Theta(\omega) = -\omega T/2$, which corresponds to a phase delay $P(\omega) = T/2$, or one-half sample.

From a sine-wave analysis point of view, if the input to a filter with frequency response $H(e^{j\omega T}) = G(\omega)e^{j\Theta(\omega)}$ is $x(n) = \cos(\omega nT)$, then the output is

$$y(n) = G(\omega)\cos[\omega nT + \Theta(\omega)] = G(\omega)\cos\{\omega[nT - P(\omega)]\} \tag{2.56}$$

and it can be clearly seen in this form that the phase delay expresses the phase response as a time delay.

Phase unwrapping

In working with phase delay, it is often necessary to "unwrap" the phase response $\Theta(\omega)$. Phase unwrapping ensures that all appropriate multiples of 2π have been included in $\Theta(\omega)$. We defined $\Theta(\omega)$ simply as the complex angle of the frequency response $H(e^{j\omega T})$, and this is not sufficient for obtaining a phase response which can be converted to true time delay. If multiples of are discarded, as is done in the definition of complex angle, the phase delay is modified by multiples of the sinusoidal period. Since LTI filter analysis is based on sinusoids without beginning or end, one cannot in principle distinguish between "true" phase delay and a phase delay with discarded sinusoidal periods when looking at a sinusoidal output at any given frequency. Nevertheless, it is often useful to define the filter phase response as a continuous function of frequency with the property that $\Theta(0) = 0$ or π (for real filters). This specifies how to unwrap the phase response at all frequencies where the amplitude response is finite and nonzero. When the amplitude

response goes to zero or infinity at some frequency, we can try to take a limit from below and above that frequency.

Matlab has a function called `unwrap()` which implements a numerical algorithm for phase unwrapping. Figure 2.9 shows the effect of the unwrap function on the phase response of the example elliptic lowpass filter, modified to contract the zeros from the unit circle to a circle of radius 0.95 in the z-plane. In Figure 2.9a, the phase-response minimum has "wrapped around" to the top of the plot. In Figure 2.9b, the phase response is continuous. We have contracted the zeros away from the unit circle in this example, because the phase response really does switch discontinuously by radians when frequency passes through a point where the phases crosses zero along the unit circle. The unwrap function need not modify these discontinuities, but it is free to add or subtract any integer multiple of in order to obtain the "best looking" discontinuity. Typically, for best results, such discontinuities should alternate between $+\pi$ and $-\pi$, making the phase response resemble a distorted "square wave".

Listing 2.7. Matlab

```
% design lowpass filter (order, passband ripple, stopband attenuation, cutoff frequency)
[B,A] = ellip(4,1,20,0.5);
B = B .* (0.95).^[1:length(B)]; % contract zeros by 0.95
[H,w] = freqz(B,A);       % frequency response
theta = angle(H);         % phase response
thetauw = unwrap(theta); % unwrapped phase response
% also thetauw = phase(theta);


subplot(2,1,1), plot(w, theta)
title('Phase response')
xlabel('Normalized frequency (radians / sample)');
ylabel('\Theta(\omega)');
subplot(2,1,2), plot(w, thetauw)
title('Phase response unwrapped')
xlabel('Normalized frequency (radians / sample)');
ylabel('\Theta(\omega)');
```

Group delay

A more commonly encountered representation of filter phase response is called the group delay, defined by

$$D(\omega) = -\frac{d}{d\omega}\Theta(\omega) \tag{2.57}$$

For linear phase responses, i.e. $\Theta(\omega) = -\alpha\omega$ for some constant α, the group delay and the phase delay are identical, and each might be interpreted as time delay (equal to α when $\omega \in [-\pi, +\pi]$). If the phase response is nonlinear, then the relative phases of the sinusoidal signal components are generally altered by the filter. A nonlinear phase response normally causes a "smearing" of attack transients such as in percussive sounds. Another term for this type of phase distortion is phase dispersion.

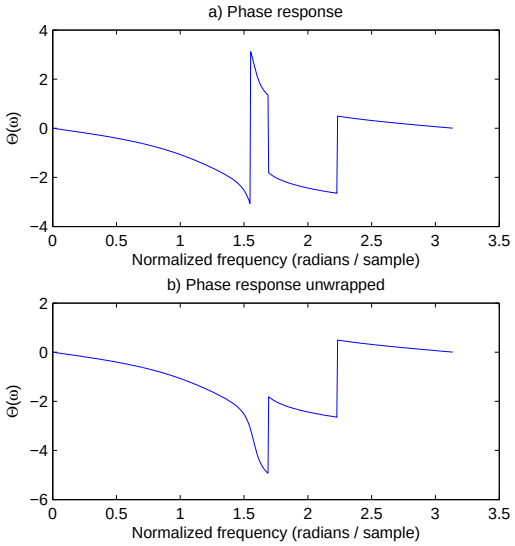


Fig. 2.9. Phase unwrapping

For any phase function the group delay $D(\omega)$ may be interpreted as the time delay of the amplitude envelope of a sinusoid at frequency ω. The bandwidth of the amplitude envelope in this interpretation must be restricted to a frequency interval over which the phase response is approximately linear. Thus, the name "group delay" for $D(\omega)$ refers to the fact that it specifies the delay experienced by a narrow-band "group" of sinusoidal components which have frequencies within a narrow frequency interval about ω. The width of this interval is limited to that over which $D(\omega)$ is approximately constant.

Suppose we write a narrowband signal centered at frequency ω_c as

$$x(n) = a_m(n)e^{j\omega_c n} \tag{2.58}$$

where ω_c is defined as the carrier frequency (in radians per sample), and $a_m(n)$ is some narrowband amplitude modulation signal. It can be shown that the output of a LTI filter is given by

$$y(n) = a^f[n - D(\omega_c)]e^{j\omega_c[n-P(\omega_c)]} \tag{2.59}$$

where $a^f(n)$ denotes a zero phase filtering of the amplitude envelope $a(n)$ by $G(\omega+\omega_c)$. We have shown that, for narrowband signals expressed as a modulation envelope times a sinusoidal carrier, the carrier wave is delayed by the filter phase delay, while the modulation is delayed by the filter group delay, provided that the filter phase response is approximately linear over the narrowband frequency interval.

Filters preserving phase

When one wishes to modify only a signal's magnitude-spectrum and not its spectral phase, then a linear-phase filter is desirable.

Linear-phase filters have a symmetric impulse response, i.e.,

$$\boxed{h(n) = h(N-1-n)} \tag{2.60}$$

for $n = 0, 1, 2, \ldots, N-1$, where $h(n)$ is the length N impulse response of a causal FIR filter. Recursive filters cannot have symmetric impulse responses.

A zero phase filter is a special case of a linear phase filter in which the phase is zero. The real impulse response $h(n)$ of a zero-phase filter is even. That is, it satisfies

$$\boxed{h(n) = h(-n)} \tag{2.61}$$

Note that a zero-phase filter cannot be causal.

It is a well known Fourier symmetry that real, even signals have real, even Fourier transforms. Therefore a real, even impulse response corresponds to a real, even frequency response. We have that

$$H(e^{j\omega T}) = \sum_{-\infty}^{+\infty} h(n)\cos(\omega n T) \tag{2.62}$$

This is a real and even function of ω. In practice, the filter is usually precisely zero-phase in all "pass bands", while it oscillates between 0 and π in the stop bands.

Let us prove that if $h(n)$ satisfies the condition $h(n) = h(N-1-n)$, then it has linear phase. We assume here that N is odd. As a result, the filter

$$h_{zp}(n) = h(n + \frac{N-1}{2}), \quad n = -\frac{N-1}{2}, \ldots, \frac{N-1}{2} \tag{2.63}$$

is a zero-phase filter. Thus, every linear-phase filter can be expressed as a delay of some zero-phase filter,

$$h(n) = h_{zp}(n - \frac{N-1}{2}), \quad n = 0, 1, \ldots, N-1 \tag{2.64}$$

By the shift theorem for z transforms, the transfer function of a linear phase filter is

$$H(z) = z^{-\frac{N-1}{2}} H_{zp}(z) \tag{2.65}$$

and the frequency response is

$$H(e^{j\omega T}) = e^{-j\omega\frac{N-1}{2}T} H_{zp}(e^{j\omega T}) \tag{2.66}$$

Since $H_{zp}(e^{j\omega T})$ can go negative, the phase response is

$$\Theta(\omega) = \begin{cases} \frac{N-1}{2}\omega T, & H_{zp}(e^{j\omega T}) \geq 0 \\ \frac{N-1}{2}\omega T + \pi, & H_{zp}(e^{j\omega T}) < 0 \end{cases} \tag{2.67}$$

Listing 2.8. Matlab

```
h = [3 4 6 1];
h_lp = [fliplr(h(2:end)) h];

freqz(h_lp, 1, 1024);
```

Allpass filters

The allpass filter passes all frequencies with equal gain. This is in contrast with a lowpass filter, which passes only low frequencies, a highpass which passes high-frequencies, and a bandpass filter which passes an interval of frequencies. An allpass filter may have any phase response. The only requirement is that its amplitude response be constant. Normally, this constant is $|H(e^{j\omega})| = 1$.

From a physical modeling point of view, a unity-gain allpass filter models a lossless system in the sense that it preserves signal energy. Specifically, if $x(n)$ denotes the input to an allpass filter $H(z)$, and if $y(n)$ denotes its output, then we have

$$\sum_{-\infty}^{+\infty} |x(n)|^2 = \sum_{-\infty}^{+\infty} |y(n)|^2 \tag{2.68}$$

All an allpass filter can do is delay the sinusoidal components of a signal by differing amounts.

The transfer function of every finite-order, causal, lossless IIR digital filter (recursive allpass filter) can be written as

$$H(z) = e^{j\phi} z^{-K} \frac{\tilde{A}(z)}{A(z)} \tag{2.69}$$

where $K \geq 0$,

$$A(z) \triangleq 1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_N z^{-N}, \tag{2.70}$$

and

$$\tilde{A}(z) \triangleq z^{-N} \bar{A}(z^{-1}) = \bar{a}_N + \bar{a}_{N-1} z^{-1} + \bar{a}_{N-2} z^{-2} + \ldots + a_1 z^{-(N-1)} + z^{-N}. \tag{2.71}$$

Thus, $\tilde{A}(z)$ is obtained from by simply reversing the order of the coefficients and conjugating them when they are complex.

In terms of the poles and zeros of a filter $H(z) = B(z)/A(z)$, an allpass filter must have a zero at $z = 1/\hat{p}$ for each pole at $z = p$.

Listing 2.9. Matlab

```
b = [1 3 1 5];
a = conj(fliplr(b));

freqz(b, a, 1024);
```

Minimum phase digital filters

An LTI filter $H(z) = B(z)/A(z)$ is said to be minimum phase if all its poles and zeros are inside the unit circle $|z| = 1$ (excluding the unit circle itself). Note that minimum-phase filters are stable by definition since the poles must be inside the unit circle. In addition, because the zeros must also be inside the unit circle, the inverse filter $1/H(z)$ is also stable when is minimum phase. A filter is minimum phase if both the numerator and denominator of its transfer function are minimum-phase polynomials in z^{-1}:

A polynomial of the form

$$B(z) = b_0 + b_1 z{-}1 + b_2 z^{-2} + \ldots + b_M z^{-M} = b_0(1 - \xi_1 z^{-1})(1 - \xi_2 z^{-1}) \cdots (1 - \xi_M z^{-1}) \tag{2.72}$$

is said to be minimum phase if all the roots ξ_i are inside the unit circle.

A signal $h(n)$ is said to be minimum phase if its z transform $H(z)$ is minimum phase. Note that every stable, causal, all-pole, (causal) filter $H(z) = b_0/A(z)$ is minimum phase. The opposite of minimum phase is maximum phase.

Example

An easy case to classify is the set of all first-order FIR filters

$$H(z) = 1 + h_1 z^{-1} \tag{2.73}$$

where we have normalized h_1 to 1 for semplicity. We have a single zero at $z = -h_1$. If $|h_1| < 1$, the filter is minimum phase. If $|h_1| > 1$ it is maximum phase. Note that the minimum phase case is the one in which the impulse response $[1, h_1, 0, \ldots]$ decays instead of grows. It can be shown that this is a general property of minimum-phase sequences:

Among all signals $h_i(n)$ having the identical magnitude spectra, the minimum phase signal $h_{mp}(z)$ has the fastest decay in the sense that

$$\sum_{n=0}^{K} |h_{mp}(n)|^2 \geq \sum_{n=0}^{K} |h_i(n)|^2, \quad K = 0, 1, 2, \ldots \tag{2.74}$$

That is, the signal energy in the first $K+1$ samples of the minimum-phase case is at least as large as any other causal signal having the same magnitude spectrum. Thus, minimum-phase signals are maximally concentrated toward time 0 among the space of causal signals having a given magnitude spectrum. As a result of this property, minimum-phase signals are sometimes called minimum-delay signals.

Minimum phase/allpass decomposition

Every causal stable filter $H(z)$ can be factored out into a minimum-phase filter in cascade with a causal stable allpass filter:

$$H(z) = H_{mp}(z)S(z) \tag{2.75}$$

where $H_{mp}(z)$ is minimum phase, $S(z)$ is an allpass filter

$$S(z) = \frac{\bar{s}_L + \bar{s}_{L-1}z{-}1 + \bar{s}_{L-2}z^{-2} + \ldots + z^{-L}}{1 + s_1 z{-}1 + s_2 z^{-2} + \ldots + s_L z^{-L}} \tag{2.76}$$

and L is the number of non-minimum-phase zeros of $H(z)$.

Listing 2.10. Matlab

```
z = [2; 3*exp(j*pi/8); 3*exp(-j*pi/8); 0.5*exp(j*pi/4); 0.5*exp(-j*pi/4)];
p = [0.9; 0.8*exp(j*pi/2); 0.8*exp(-j*pi/2)];

figure(1)
zplane(z, p)

b = poly(z);
a = poly(p);

z_minp = z(abs(z) <  1);
z_maxp = z(abs(z) >= 1);

b_allpass = poly(z_maxp);
a_allpass = conj(fliplr(b_allpass));

b_minp = poly(z_minp);
a_minp = poly(p);

[H, w] = freqz(b, a, 1024);
[H_minp, w] = freqz(b_minp, a_minp, 1024);
[H_allpass, w] = freqz(b_allpass, a_allpass, 1024);

figure(2)
subplot(3,2,1) plot(w, abs(H)); title('|H(\omega)|')
subplot(3,2,2); plot(w, phase(H)); title('\angle H(\omega)')

subplot(3,2,3) plot(w, abs(H_minp));  title('|H_{minp}(\omega)|')
subplot(3,2,4); plot(w, phase(H_minp)); title('\angle H_{minp}(\omega)')
```

```
subplot(3,2,5) plot(w, abs(H_allpass)); title('|H_{allpass}(\omega)|')
subplot(3,2,6); plot(w, phase(H_allpass)); title('\angle H_{allpass}(\omega)')
```

References

- Smith, J.O. Introduction to Digital Filters, September 2005 Draft,

  http://ccrma.stanford.edu/~jos/filters05/

3

Windowing and Short Time Fourier Transform

3.1 Overview of windows

Windows are used to convert infinite duration signals to finite duration signals. For practical reasons, we must have a finite duration signal in order to compute the DFT (FFT). For the moment this is a severe limitation, but we will see how we can effectively compute the DTFT, through a succession of DFT's.

Windows are commonly used for:

- Spectral analysis (Short Time Fourier Transform)
- FIR Filter design

Example of windowing

Lets look at a simple example of windowing to demonstrate what happens when we turn an infinite duration signal into a finite duration signal through windowing. For example, consider the complex sinusoid:

$$x(n) = e^{j\omega_0 nT}, \quad -\pi \leq \omega_0 \leq +\pi \tag{3.1}$$

We notice that:

- real part $= \cos(\omega_0 nT)$
- the frequencies present in the signal are only positive (it is an *analytic signal*)

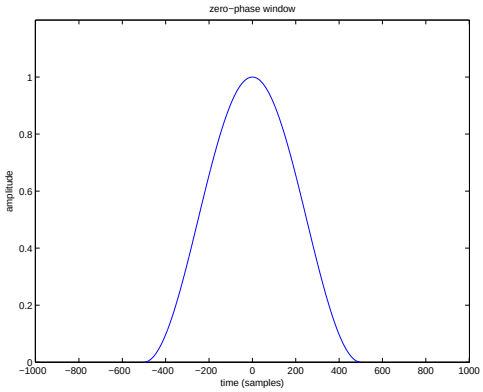


Fig. 3.1. Example of a zero-phase window (Hanning) of length $N = 1001$ samples

This signal is infinite duration. In order to end up with a signal which dies out eventually (so we can use the DFT), we need to multiply our signal by a window. The window in Figure 3.1 is real and even. Its Fourier transform is real and even, therefore it is a zero-phase signal

We might require that our window is zero for time values less than 0. We define such a window as causal. This is necessary for real time processing.

By shifting the original window in time by half of its length, we have turned the original non-causal window into a causal window. The shift property of the Fourier transform tells us that we have introduced a linear phase term.

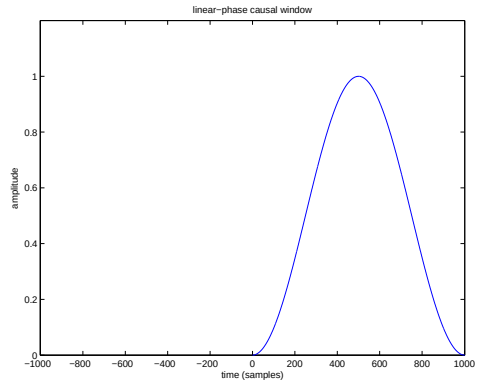


Fig. 3.2. Example of a linear-phase causal window (Hanning) of length $N = 1001$ samples

Our original signal (unwindowed, infinite duration) is $x(n) = e^{j\omega_0 nT}$, $n \in \mathbb{Z}$. The DTFT of this infinite duration signal is a delta function at ω_0: $X(\omega) = \delta(\omega - \omega_0)$ (remember that the DTFT is continuous and periodic, with period 2π, i.e. $X(\omega) = X(\omega + 2\pi m)$). The windowed version is $x_w(n) = w(n)e^{-j\omega_0 nT}$, as shown in Figure 3.3.

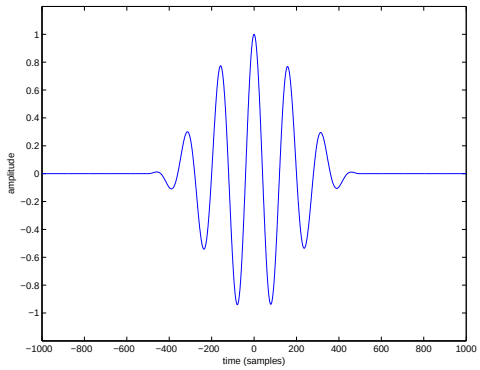


Fig. 3.3. Example of windowed signal (real part)

In Chapter 1 we showed that, for the DFT, multiplication in the time domain corresponds to (cyclic) convolution in the frequency domain. The same result holds also for the DTFT. Hence, in our case, we are left with the (cyclic) convolution of a delta function at ω_0 and the DTFT of the window. The result of convolution with a delta function, is the original function, shifted to the location of the delta function.

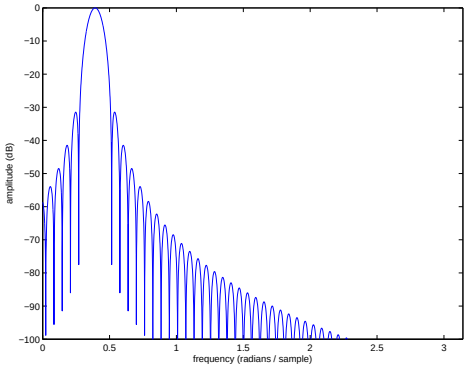


Fig. 3.4. Example of windowed signal spectrum.

- Our windowing operation in the time domain resulted in a 'smearing' or 'smoothing' in the frequency domain. We need to be aware of this if we are trying to resolve sinusoids which are close together in frequency
- Windowing also introduces side lobes. This is important when we are trying to resolve low amplitude sinusoids in the presence of higher amplitude signals

Types of windows

There are many type of windows which serve various purposes and exhibit various properties.

Rectangular window

The causal rectangular window may be defined as

$$w_R(n) = \begin{cases} 1/M, & n = 0, 1, 2, \ldots, M-1 \\ 0, & \text{elsewhere} \end{cases} \tag{3.2}$$

To see what happens in the frequency domain, we need to look at the DTFT of the window:

$$\begin{aligned} W_R(\omega_k) &= \text{DTFT}(w_R) \\ &= \sum_{n=-\infty}^{+\infty} w_R(n) e^{-j\omega nT} \\ &= \sum_{n=0}^{M-1} \frac{1}{M} (e^{-j\omega T})^n \\ &= \frac{1}{M} \frac{1 - e^{-j\omega TM}}{1 - e^{-j\omega T}} \qquad (3.3) \\ &= C \frac{\sin(\omega MT/2)}{\sin(\omega T/2)} \qquad (3.4) \end{aligned}$$

This function is sometimes called digital sinc and it is denoted by

$$\boxed{sinc_M(\omega T) \triangleq \frac{\sin(\omega MT/2)}{\sin(\omega T/2)}} \tag{3.5}$$

and it is shown in Figure 3.5 for $M = 31$.

Some important points:

- zero crossings at integer multiples of $\Omega_M \triangleq 2\pi/M$ (Ω_M is the frequency sampling interval for a length M DFT)
- main lobe width of $2\Omega_M = \frac{4\pi}{M}$
- as M gets bigger, the mainlobe narrows (better frequency resolution)
- M has no effect on the height of the side lobes
- 1st sidelobe only 13dB down from the mainlobe peak
- side lobes roll off at approximately 6dB / octave (as $T \to 0$)
- the phase term comes from the fact that we shifted the window to make it causal.

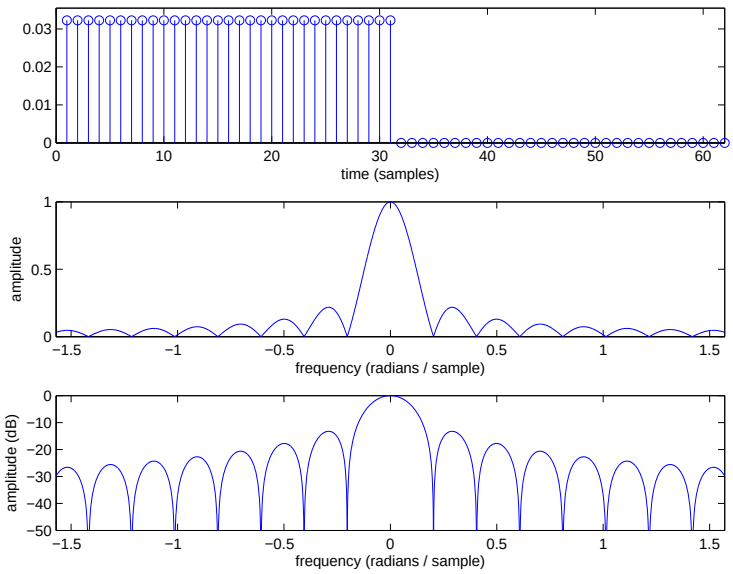


Fig. 3.5. Rectangular window of length $M = 31$ and its spectrum.

Example: resolution of two cosines (in-phase case) (Figure 3.6)

- 2 cosines separated by $\Delta\omega = \frac{2\pi}{40}$
- rectangular windows of lengths: 20, 30, 40, 80 ($\Delta\omega = \frac{1}{2}\Omega_M, \frac{3}{4}\Omega_M, \Omega_M, 2\Omega_M$)

Listing 3.1. Matlab

```
M = [20,30,40,80];

for i = 1:length(M)

    N = 2^14;

    Omega_M = 2*pi / (N/1000);

    Omega_1 = Omega_M;
    Omega_2 = Omega_M + 2*pi / 40;

    n = [0:N-1];
    T = 1;
    x = cos(Omega_1 * n * T)' + cos(Omega_2 * n * T)';

    w_R = 1/M(i) *ones(M(i),1);
    w_R = [w_R; zeros(N - M(i),1)];

    y = x.*w_R;

    Y = fft(y, N);

    omega = (2*pi*[-N/2: N/2-1] / N)';
    subplot(2,2,i), plot(omega, fftshift(20*log10(abs(Y))));
    title(['M = ' num2str(M)])
    xlabel('frequency (radians / sample)');
    ylabel('amplitude (dB)');

    axis([0,pi/2,-30,0])

end
```

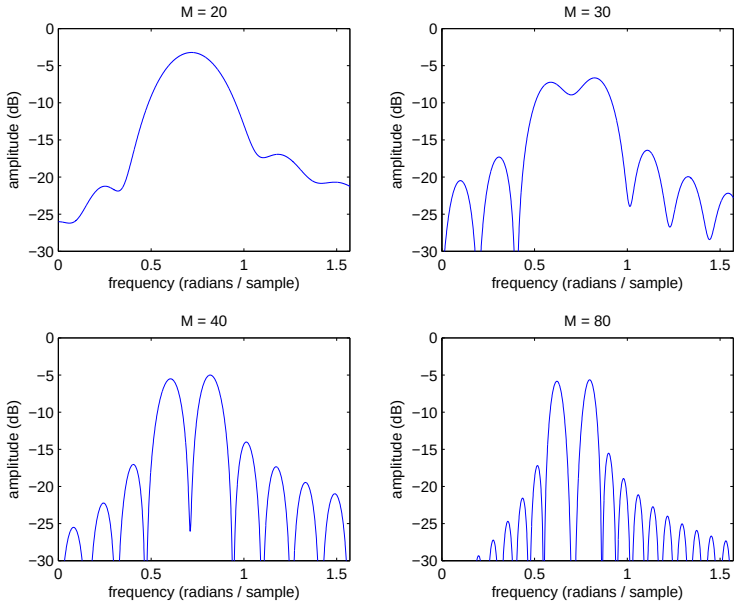


Fig. 3.6. Separation of two in-phase sinusoids with a rectangular window of length $M = 20, 30, 40, 80$.

Example: resolution of two cosines (phase quadrature case) (Figure 3.7)

- as above, but 1 sine and 1 cosine
- note: least-resolved case resolved
- note: $M = 40$ case suddenly looks much worse
- Only the $M = 80$ case looks good at all phases

Note: both in the in-phase and phase quadrature cases, peak locations are biased.

Figure 3.6 and Figure 3.7 suggest that, for a rectangular window, two sinusoids can be most easily resolved when they are separated in the frequency by

$$\Delta\omega \geq 2\Omega_M \tag{3.6}$$

This means that there must be at least two full cycles of the difference frequency under the window.

The rectangular window provides an abrupt transition at its edge. Lets look at some other windows which have a more gradual transition. This is usually done to reduce the height of the side lobes, but increasing the width of the main lobe.

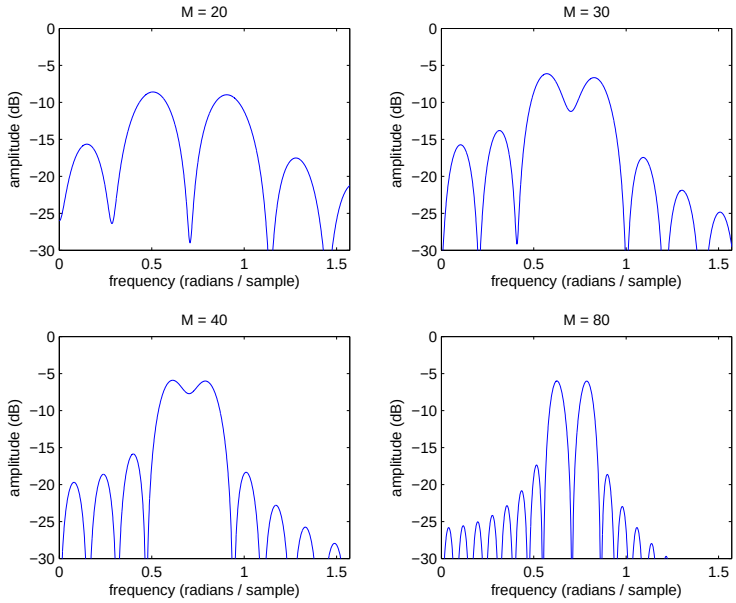


Fig. 3.7. Separation of two in-phase sinusoids with a rectangular window of length $M = 20, 30, 40, 80$.

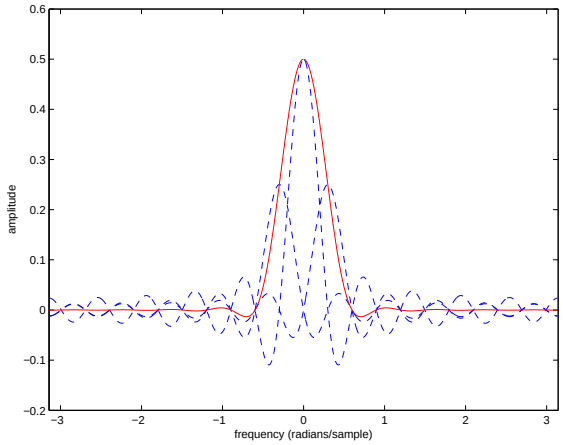


Fig. 3.8. Construction of the frequency response of the Hanning window.

Tapered windows

Consider Figure 3.8 in the frequency domain

We have added two extra digital sinc functions (shifted at $+\Omega_M$ and $-\Omega_M$) which results in the following behavior

- There is some cancelation of the side lobes

- The width of the main lobe is doubled

The frequency response can be written in terms of $W_R(\omega)$

$$\begin{aligned} W_H(\omega) &\triangleq \alpha W_R(\omega) + \beta W_R(\omega - \Omega_M) + \beta W_R(\omega + \Omega_M) \\ &= \alpha sinc_M(\omega) + sinc_M(\omega - \Omega_M) + \beta sinc_M(\omega + \Omega_M) \end{aligned} \tag{3.7}$$

Using the shift theorem, we can take the inverse DTFT of the above equation:

$$w_H = \alpha w_R(n) + \beta e^{-j\Omega_M n} w_R(n) + \beta e^{j\Omega_M n} w_R(n) = w_R(n) \left[\alpha + 2\beta \cos\left(\frac{2\pi n}{M}\right) \right] \tag{3.8}$$

Hanning window or Hann

The Hanning window is defined by setting α to 1/2 and β to 1/4 (see Figure 3.9)

$$w_H(n) = w_R(n) \left[\frac{1}{2} + \frac{1}{2} \cos(\Omega_M n) \right] = w_R(n) \cos^2\left(\frac{\Omega_M}{2} n\right) \tag{3.9}$$

Hanning window properties

- Mainlobe is $4\Omega_M$ wide
- 1st lobe is at -31dB
- side lobes roll off at approx. 18dB /octave

Hamming

This window is determined by optimizing over α and β in such a way as to reduce the worst case side lobe level (see Figure 3.9). Doing this, results in the following values:

- $\alpha = 0.5436$ ($\sim 25/46$)
- $\beta = (1 - \alpha)/2$

Hamming window properties

- Discontinuous slam to zero at endpoints
- main lobe is $4\Omega_M$
- roll off is approx 6dB / octave (as $T \to 0$)
- 1st side lobe is improved over Hanning (approx $-43dB$)
- side lobes closer to equal ripple

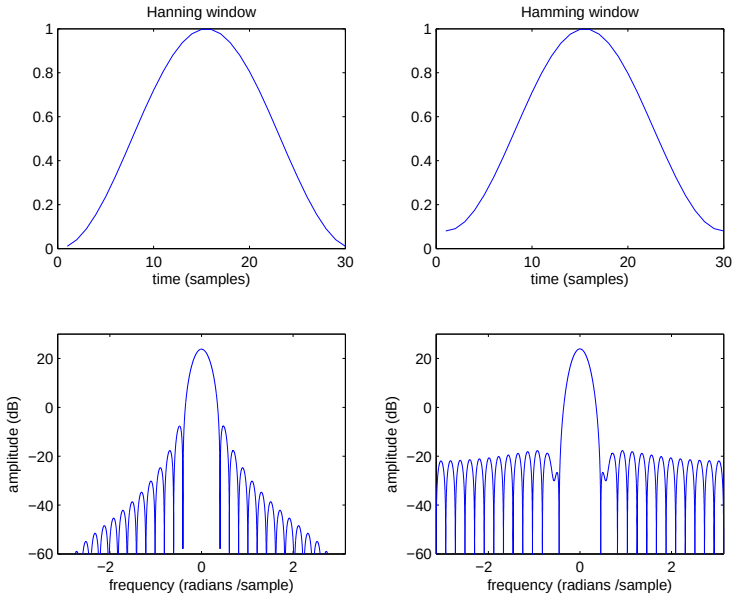


Fig. 3.9. Hanning window (left) and Hamming window (right).

Blackman-Harris

The Blackman-Harris family of windows is basically a generalization of the Hamming family.

The Blackman-Harris family is derived by considering a more general summation of shifted sinc functions:

$$w_B(n) = w_R(n) \sum_{l=0}^{L-1} \alpha_l \cos(l\Omega_M n) \tag{3.10}$$

- $L = 1$: rectangular
- $L = 2$: Generalized Hamming
- $L = 3$: Blackman

Blackman

The Blackman window is a specific case where $\alpha_0 = 0.42$, $\alpha_1 = 0.5$ and $\alpha_2 = 0.08$.

Properties

- side lobe roll off about 18dB per octave (as $T \to 0$)
- -58.11dB side lobe level (worst case)

Kaiser window

The Kaiser window maximizes the energy in the main lobe of the window

$$\max_{w} \left[\frac{\text{main lobe energy}}{\text{side lobe energy}} \right] \tag{3.11}$$

Kaiser discovered an approximation based upon Bessel functions:

$$w_K(n) \triangleq \begin{cases} \frac{I_0\left(\beta\sqrt{1-\left(\frac{n}{M/2}\right)^2}\right)}{I_0(\beta)} & \frac{M-1}{2} \leq n \leq \frac{M-1}{2} \\ 0 & \text{elsewhere} \end{cases} \tag{3.12}$$

where I_0 is a Bessel function of the first kind, and it is equal to

$$I_0(x) \triangleq \sum_{k=0}^{\infty} \left[\frac{(x/2)^k}{k!} \right]^2 \tag{3.13}$$

Sometimes you see the Kaiser window parametrized by α, where $\beta \triangleq \pi\alpha$

- β is equal to 1/2 time-bandwidth product
- β trades off side lobe level for main lobe width. Larger β implies lower side lobe level, wider main lobe

Dolph-Chebyshev window

The Dolph-Chebychev window minimizes the Chebychev norm of the side lobes, for a given main lobe width $2\omega_c$:

$$\min_{w} ||\text{sidelobes(W)}||_\infty \triangleq \min_{w} \{ \max_{\omega > \omega_c} |W(\omega)| \} \tag{3.14}$$

The Chebychev norm is also called the $L-$infinity norm, uniform norm, minimax norm or simply the maximum absolute value.

The optimal Dolph-Chebychev window transform can be written in closed form:

$$W(\omega_k) = (-1)^k \frac{\cos\{M \cos^{-1}[\beta \cos(\pi k/M)]\}}{\cosh[M \cosh^{-1}(\beta)]}, \quad |k| \leq M-1 \tag{3.15}$$

$$\beta = \cosh -1 \left[\frac{1}{M} cosh^{-1}(10^\alpha) \right] \quad \alpha \approx 2, 3, 4 \tag{3.16}$$

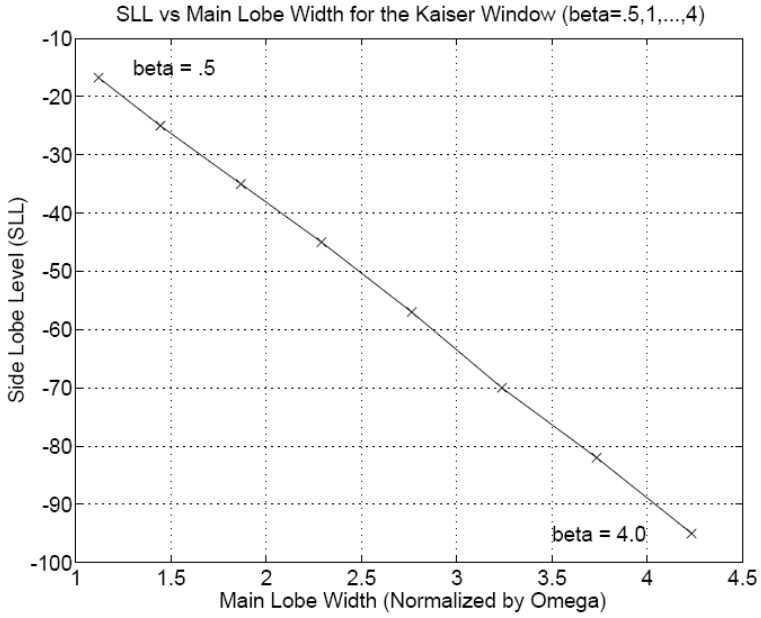


Fig. 3.10. Kaiser window: main lobe vs side lobe level tradeoff

The zero-phase Dolph-Chebychev window, $w(n)$, is then computed as the inverse DFT of $W(\omega_k)$. The α parameter controls the side lobe level via the formula

$$\text{Side-Lobe Level in dB} = -20\alpha \tag{3.17}$$

Since the side lobes of the Dolph-Chebyshev window transform are equal height, they are often called "ripple in the stop-band" (thinking now of the window transform as a low-pass filter frequency response). The smaller the ripple specification, the larger ω_c has to become to satisfy it, for a given window length M.

Note: The elegant equal ripple property in the frequency domain (a perfect solution to a minimum-sidelobe-attenuation specifcation) has the unfortunate consequence of introducing impulsive ears at the endpoints of the window in the time-domain. These ears can be the source of pre-echo or post-echo distortion which are time-domain effects not reflected in a simple sidelobe level specification.

Gaussian window

The Gaussian curve is the only smooth function that transforms to itself:

$$e^{-t^2/2\sigma^2} \leftrightarrow \sqrt{2\pi\sigma^2}e^{-\omega^2/2(1/\sigma)^2} \tag{3.18}$$

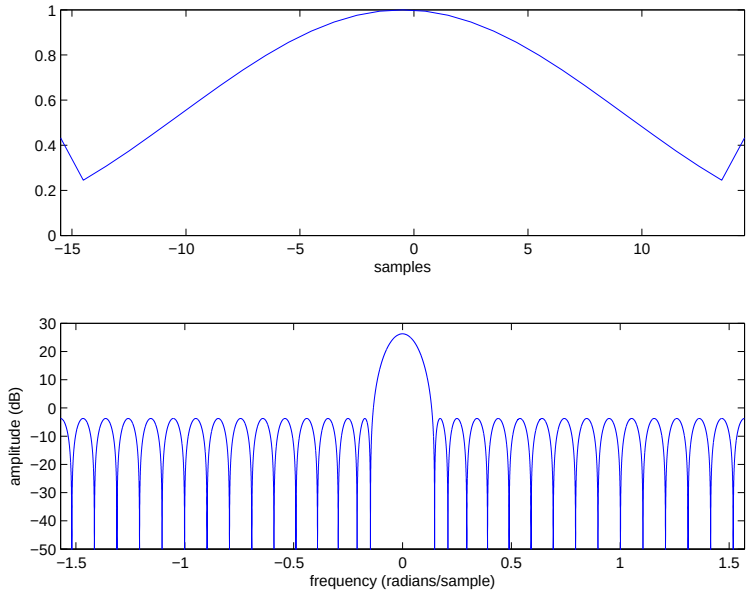


Fig. 3.11. Chebychev window $M = 31$, ripple -30dB

Since the true Gaussian function has infinite duration, in practice we must window it with some usual finite window, or at least truncate it.

Property: On a dB scale the Gaussian is quadratic $\Rightarrow$ parabolic interpolation of a sampled Gaussian transform is exact.

Listing 3.2. Matlab

```
wvtool(hamming(64),hann(64),gausswin(64))

% see also
% wintool
```

Windows summary

Table 3.1. Characteristics of popular windows

window	Main-lobe width	Side-lobe level [dB]	Roll-off [dB/octave]
Rectangular	$2\Omega_M$	-13.3	-6
Hanning	$4\Omega_M$	-31.5	-18
Hamming	$4\Omega_M$	-42.7	-6
Blackman	$6\Omega_M$	-58.1	-18

3.2 Overlap and add

Finite length signals

In Chapter 2 we showed that the output of a LTI digital filter can be obtained by computing the (linear) convolution of the input signal with the impulse response of the filter, i.e.

$$y(n) = \sum_{i=-\infty}^{+\infty} x(i)h(n-i) = \sum_{i=-\infty}^{+\infty} h(i)x(n-i) \tag{3.19}$$

If the filter is causal and FIR with N_h taps, we get

$$y(n) = \sum_{i=0}^{N_h-1} h(i)x(n-i) \tag{3.20}$$

If the input signal is finite length, with N_x samples, the computation of the linear convolution requires $N_h \cdot N_x$ multiplications and $N_h \cdot N_x - 1$ additions, i.e. the computational complexity order is $\mathcal{O}(N_h \cdot N_x)$.

In the following, we will show how to compute linear convolution in the frequency domain by means of the FFT. First, recall the convolution theorem introduce in Chapter 1 for discrete sequences of length N:

$$(x * y) \leftrightarrow XY \tag{3.21}$$

or

$$\mathrm{DFT}_k(x * y) = X(k)Y(k) \tag{3.22}$$

It is important to remember that the specific form of convolution implied in the above equation is circular or cyclic convolution:

$$y(n) \triangleq (x * h) \triangleq \sum_{m=0}^{N-1} x(m)h(n-m)_N \tag{3.23}$$

where $(n-m)_N$ means $(n-m)$ modulo N.

Idea: If we add enough trailing zeros to the signals being convolved, we can get the same results as in linear convolution.

- If we perform a linear convolution of two signals, x and h, with lengths N_x and N_h, as in (3.20), the resulting signal has length $N_y = N_x + N_h - 1$.
- We must therefore add enough zeros to x and h so that the cyclic convolution result is length N_y or longer.

- If we don't add enough zeros, some of our convolution terms wrap around and add back upon others (due to modulo indexing).
- This can be thought of as time domain aliasing.

Therefore, in order to compute linear convolution in the frequency domain, we need to perform the following steps

1. Pad both the input signal x and the filter h with zeros, in such a way that both signals have length $N \geq N_x + N_h - 1$. In practice, it is convenient to choose N to be the smallest power of 2 satisfying the above inequality. This is due to the fact that the computation of the FFT is fastest when the input signal lengths are powers of 2.
2. Compute the FFT of x and h, i.e. X and H ($\mathcal{O}(2N \log N)$).
3. Perform term-by-term multiplication of X and H, i.e. $Y(k) = X(k)H(k)$ ($\mathcal{O}(N)$). This is equivalent to computing cyclic convolution of the zero-padded signals, thus avoiding wrap around effects.
4. Compute the inverse FFT of Y to get the output signal y ($\mathcal{O}(N \log N)$).

Therefore, the computational complexity of this procedure is $\mathcal{O}(2N \log N + N \log N + N) = \mathcal{O}(N \log N) = \mathcal{O}((N_x + N_h - 1) \log(N_x + N_h - 1))$.

We are given with two methods to compute linear convolution of two finite length sequences: time-domain and frequency-domain convolution.

- if N_h is small, in the order of a dozen samples, it is typically faster to compute linear convolution in the time-domain.
- if N_h is large, frequency-domain convolution should be preferred.

The exact value of N_h for which frequency-domain convolution becomes faster, depends on N_x and on the specific algorithm used to implement the FFT.

When working with image signals, the sizes of the filters involved are typically small, thus making time-domain convolution preferable. Conversely, in audio signal processing, filters have often several hundreds or even thousands of samples.

- The nominal integration time of the ear, defined as the reciprocal of the critical bandwidth of hearing, is around 10ms below 500Hz
- At 50kHz sampling rate, this is 500 samples
- Therefore FIR filters shorter than the ear's integration time can easily be hundreds of taps long
- FIR filters shorter than ear's integration time can be generally characterized by their magnitude frequency response (no perceivable delay effects)
- FFT convolution is consequently an important implementation tool for FIR filters in digital audio.

Infinite length signals

We saw that we can perform efficient linear convolution of two finite length sequences using Fourier based techniques.

There are some situations where it will not be practical to perform the convolution of two signals using one FFT

- N_x is extremely large
- Real time operation (we can't wait until the signal ends)

Theoretically, there is no problem doing this with direct convolution. Since h is finite in length we only need to store the past $N_h - 1$ samples of the input signal x to calculate the next output. Unfortunately, this procedure can be extremely time-consuming when N_h is large.

The overlap-and-add algorithm (OLA) described below provides an efficient way of performing FIR filtering on infinite length signals by taking advantage of the FFT.

Idea: We might be able to perform convolution on a block at time. Basically, we chop up the input signal, x by windowing, and perform frequency domain convolution on each block separately. We need to make sure we put it all back together correctly.

OLA algorithm summary

Inputs:

- x is indefinitely long
- h can be any FIR filter of known maximum length N_h

OLA algorithm:

1. Extract the mth block of data of length M samples, starting from the sample index mR (R denotes the hop-size).
2. Shift it to the base time interval $[0, M-1]$ (or $[-(M-1)/2, (M-1)/2]$).
3. Apply the analysis window w (causal or zero phase, as preferred).
4. Zero-pad the windowed data out to N samples (N should be a power of 2).
5. Take the N-point FFT of the windowed data (X) and the filter (H).
6. Apply the filter H in the frequency domain by performing term-by-term multiplication.
7. Take the N point inverse FFT.
8. Shift the origin of the N-point result out to sample mR where it belongs.
9. Sum into the output buffer containing the results from prior frames (OLA step).

The following conditions must be satisfied:

- $\sum_m w(n - mR) = 1$
- To avoid time domain aliasing: $N > M + N_h - 1$

In the following, we will show that if both conditions are satisfied, the OLA algorithm computes exactly the same result as linear convolution in the time-domain. As for finite-length signals, the computational complexity of the OLA algorithm is smaller than direct linear convolution when the size of the filter N_h is large, thanks to the use of the FFT.

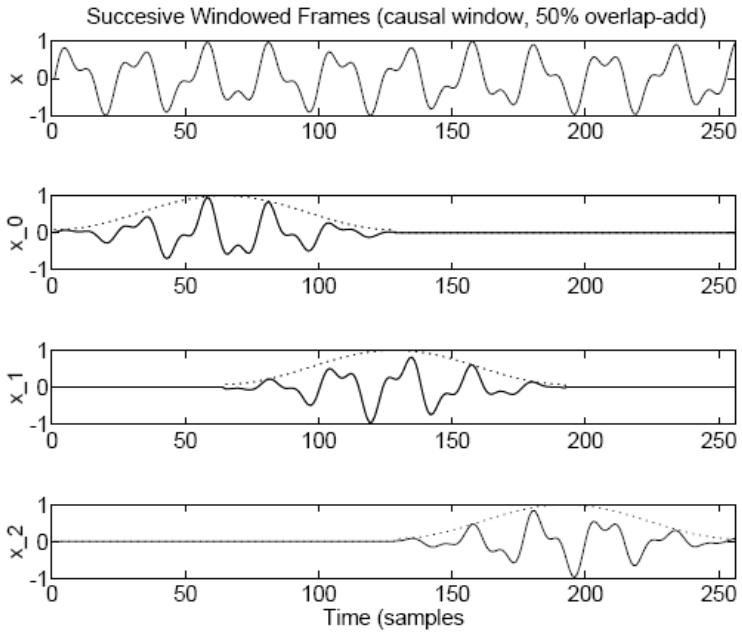


Fig. 3.12. Successive windowed frames (causal window, 50% overlap

The mathematics of OLA

Consider breaking the input signal x, into frames using a finite, zero phase length M(odd) window. Let x_m denote the mth frame.

$$x_m(n) \triangleq x(n)w(n - mR) \quad n \in (-\infty, +\infty) \tag{3.24}$$

where $R \triangleq$ frame step (hop size) and $m \triangleq$ frame index.

The hop size is the number of samples between adjacent frames. Specifically, it is the number of samples by which we advance each successive window (see Figure 3.12).

For this frame-by-frame spectral processing to work, we must be able to reconstruct x from the individual overlapping frames. This can be written as

$$\begin{aligned} x(n) &= \sum_{m=-\infty}^{+\infty} x_m(n) \\ &= \sum_{m=-\infty}^{+\infty} x(n)w(n-mR) \\ &= x(n) \sum_{m=-\infty}^{+\infty} w(n-mR) \end{aligned} \tag{3.25}$$

Hence, $x = \sum_m x_m$ if and only if

$$\sum_m w(n-mR) = 1 \tag{3.26}$$

This is the constant-overlap-add (COLA) constraint for the analysis window w.

- All windows which obey the constant-overlap-add constraint will yield perfect reconstruction of the original signal from the data frames by overlap-add (OLA).
- There is no constraint on window type, only that the window overlap-adds to a constant for the hop size used.
- Examples:
  - Rectangular window at 0% overlap (hop size R = window size M)
  - Rectangular window at 50% overlap ($R = M/2$)
  - Bartlett window at 50% overlap ($R = M/2$)
  - Hamming window at 50% overlap ($R = M/2$)
  - Hamming window at 75% overlap ($R = M/4$)
  - Any window with $R = 1$ (sliding FFT)

In order to continue, let us analyze the relationship between the DTFT and the DFT of time limited signals. Let $X_m(\omega)$ denote the DTFT of the sequence $x_m(n) = x(n)w(n-mR)$, i.e. of the mth windowed frame of x.

- Note that $X_m(\omega)$ is defined on the continuous unit circle because $x_n(m)$ is defined over $-\infty < n < +\infty$, although is non-zero only at M samples.
- Each frame, $x_m(n)$ has been windowed, and hence is time limited, or band limited in the time domain.

- Since $x_m(n)$ is time limited to M nonzero samples, we can critically sample $X_m(\omega)$ at the points $\omega_k = 2\pi k/M$, for $k = 0, 1, 2, \ldots, M-1$, without any loss of information (dual of the sampling theorem).
- Sampling $X_m(\omega)$ to get $X_m(\omega_k)$ aliases $x_m(n)$ into the Mth sample time range $[0, M-1]$ (or $[-(M-1)/2, (M-1)/2]$)

We can define $\tilde{x}_m \triangleq x_m(n+mR)$ as the mth frame shifted back to the time origin. We have

$$X_m(\omega_k) \triangleq X_m(\omega)|_{\omega=\omega_k}, \quad \omega_k \triangleq \frac{2\pi k}{MT} \tag{3.27}$$

and

$$\tilde{X}_m(\omega_k) \triangleq \tilde{X}_m(\omega)|_{\omega=\omega_k} = e^{jmR\omega_k} X_m(\omega_k) = \mathrm{DFT}(\tilde{x}_m) \tag{3.28}$$

We see that the DTFT of a signal which is time-limited to the interval $[0, M-1]$ is exactly equal at points $\omega_k = 2\pi k/MT$ to the length M DFT over the interval $[0, M-1]$.

We are now ready to show the mathematics of the OLA algorithm. Getting back to the acyclic convolution (for x, y, $h \in \mathbb{C}^\infty$, in general)

$$\begin{aligned}
y(n) &= (x * h)(n) \\
&= ((\sum_m x_m) * h)(n) \quad \text{COLA constraint} \\
&= \sum_m (x_m * h)(n) \quad \text{linearity of convolution} \\
&= \sum_m \sum_l \tilde{x}_m(l - mR) h(n-l) \\
&= \sum_m \sum_{l'} \tilde{x}_m(l') h(n - mR - l') \\
&= \sum_m (\mathrm{DTFT}^{-1}(\mathrm{DTFT}(\tilde{x}_m) \cdot \mathrm{DTFT}(h)))(n - mR) \,\text{convolution theorem for the DTFT} \\
&= \sum_m (\mathrm{DTFT}^{-1}(\tilde{X}_m \cdot H))(n - mR)
\end{aligned} \tag{3.29}$$

In the last equation, we still need to calculate infinite duration DTFTs. At this point we need to make use of the following facts

- $\tilde{x}_m$ is time limited to $[0, M-1]$ (or $[-(M-1)/2, (M-1)/2]$)
- We need to assume h is time limited to N_h. This fact implies that $\tilde{x}_m * h$ will be time limited to $M + N_h - 1$

Since $\tilde{x}_m * h$ is time limited, we can sample its DTFT at intervals $\Omega \leq \frac{2\pi}{(N_h+M-1)T}$ along the unit circle.

This last observation implies that we can get by with length $N \geq N_h + M - 1$ DFT. Our final expression is given by:

$$\begin{aligned} y(n) &= \sum_m (\mathrm{DFT}_N^{-1}(\mathrm{DFT}_N(\tilde{x}_m) \cdot \mathrm{DFT}_N(h)))(n - mR) \\ &= \sum_{m=-\infty}^{\infty} \mathrm{SHIFT}_{mR}(\mathrm{DFT}^{-1}\{H \cdot \mathrm{DFT}_N[\mathrm{Shift}_{-mR}(x) \cdot w]\}) \end{aligned} \tag{3.30}$$

where H is the length N DFT of h, zero padded out to length N, with $N \geq N_h + M - 1$.

Frequency domain interpretation of COLA

In the previous section we showed that in order to compute filtering by means of the OLA algorithm, the following time-domain constraint on the window must be satisfied

$$\sum_m w(n - mR) = 1 \tag{3.31}$$

We also showed that some popular windows can be used, provided that their length M and the hop size R are properly selected.

In this section, we provide a frequency-domain interpretation of the COLA constraint, i.e. we show how (3.31) is mapped in the frequency domain. This allows, for example, to design customized windows by means of digital filter design methods (see Chapter 4) that require frequency-domain specifications as input.

First consider the summation of N complex exponentials

$$\begin{aligned} x(n) \triangleq \frac{1}{N} \sum_{k=0}^{N-1} e^{j\omega_k n} &= \begin{cases} 1, \text{ n = 0(mod N)} \\ 0, \text{ elsewhere} \end{cases} \\ &= IDFT_n(1, 1, \cdots, 1) \end{aligned} \tag{3.32}$$

where $\omega_k \triangleq 2\pi k/NT$.

Let $N = R$ and $T = 1$, in which case we have

$$\sum_m \delta(n - mR) = \frac{1}{R} \sum_{k=0}^{R-1} e^{j\omega_k n} \tag{3.33}$$

where $\omega_k \triangleq 2\pi k/R$ (harmonics of the frame rate).

Let us consider these equivalent signals as inputs to an LTI system, with an impulse response given by $w(n)$ and frequency response equal to $W(\omega)$.

Since the input are equal, the corresponding outputs must be equal too:

$$y(n) = \sum_m w(n - mR) = \frac{1}{R} \sum_{k=0}^{R-1} W(\omega_k) e^{j\omega_k n} \tag{3.34}$$

This derives the Poisson summation formula

$$\sum_m w(n - mR) = \frac{1}{R} \sum_{k=0}^{R-1} W(\omega_k) e^{j\omega_k n} \quad \omega_k \triangleq \frac{2\pi k}{R} \tag{3.35}$$

- Dual of the sampling theorem
- To reconstruct OLA exactly, we require the constant overlap-add (COLA) constraint on window

$$\sum_m w(n - mR) = \text{constant} \tag{3.36}$$

- COLA = $W(\omega_k) = 0$, $|k| = 1, 2, \ldots, R - 1$
- COLA assured when window transform is zero at all harmonics of the frame rate

When the COLA condition is met, we have

$$\sum_m w(n - mR) = \frac{1}{R} W(0) \tag{3.37}$$

- Weak COLA: Window transform has zeros at frame-rate harmonics:

$$W(\omega_k) = 0, \quad k = 1, 2, \ldots, R - 1, \quad \omega_k \triangleq 2\pi k/R \tag{3.38}$$

  - Perfect OLA reconstruction
  - Relies on aliasing cancellation in frequency domain
  - Aliasing cancellation disturbed by spectral modifications

- Strong COLA: Window transform is bandlimited consistent with downsampling by the frame rate

$$W(\omega) = 0, \quad |\omega| \geq \pi/R \tag{3.39}$$

  - Perfect OLA reconstruction
  - No aliasing
  - Ideal for spectral modifications
  - Time domain window infinitely long

Listing 3.3. Matlab

```
clear
[x, Fs, nbits] = wavread('../file sources/flute.wav');

% anti-aliasing to be sampled at 8000 Hz;
Wn = 2*4000/Fs;  % normalized cut-off frequency
N_h = 51;
h = fir1(N_h-1,Wn)'; % FIR filter design (window method)

[H, omega] = freqz(h,1,1000); f = Fs*omega/2*pi;

figure(1)
subplot(2,1,1) plot(f, 20*log10(abs(H)))
subplot(2,1,2) plot(f, phase(H))

M = 0.050*Fs;     %window length (50msec)
R = floor(M/2);   %hop size
N = 4096          %power of 2 larger than M + N_h - 1

% weak COLA - nulls at frequencies w_k = 2*pi*k/R
w = bartlett(M);

figure(2)
[W, omega] = freqz(w,1,10000);

subplot(2,1,1)
plot(omega, 20*log10(abs(W)))
subplot(2,1,2)
plot(omega, phase(W))

% strong COLA - W = 0, |w| > pi/R
% f = [0 1/R 1]; m = [R 0 0]; w = fir2(M - 1,f,m)';

% figure(2)
% [W, omega] = freqz(w,1,10000);

% subplot(2,1,1)
% plot(omega, 20*log10(abs(W)))
% subplot(2,1,2)
% plot(omega, phase(W))

y = zeros(length(x),1);
xm = zeros(M,1);
ym = zeros(M,1);

% check time domain COLA constraint
for m = 0:floor((length(x)-N)/R)

    ym = w;
    y(m*R+1:m*R+M) = y(m*R+1:m*R+M) + ym;

end

figure(3) plot(y)

H = fft(h, N);

for m = 0:floor((length(x)-N)/R)

    xm = x(m*R+1:m*R+M);

    ym = w.*xm;
    ym = ifft(H.*fft(ym, N));

    y(m*R+1:m*R+N) = y(m*R+1:m*R+N) + ym;

end
```

Short Time Fourier Transform (STFT)

In the previous section we have defined $X_m = DTFT[x_m]$. This is the Short Time Fourier Transform (STFT).

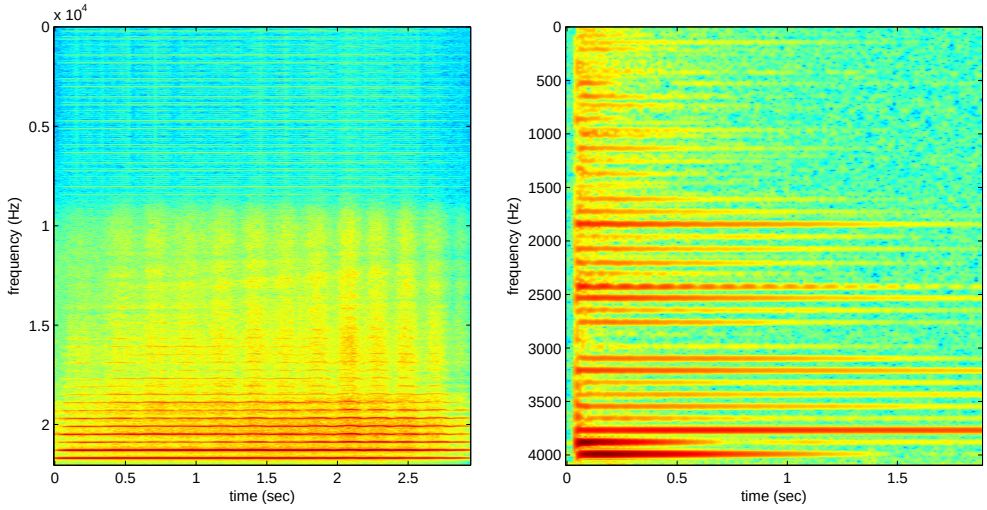
- It is a function of both time m and frequency ω_k
- The STFT is one example of a time-frequency distribution
- M determines time and frequency resolution of the data frame
- N determines time and frequency resolution of spectral modifications
  - N depends upon M
- Since we are using the DFT, frequency bins are linearly spaced

We notice that when STFT is used only as an analysis tool, i.e. no signal is synthesized in output by recombining the blocks together, we are left with greater flexibility in the choice of the window. In other words, the COLA constraint must not be satisfied. In this scenario, which occurs often in practice when analyzing audio signals, the choice of the parameters of the STFT is performed on the basis of the following considerations:

- The window length M determines the frequency resolution. An higher value of M implies closer spacing between frequency bins ($\Omega_M = 2\pi/M$).
- The type of the window determines the trade-off between main-lobe width (thus affecting frequency resolution) and side-lobe level.
- If the length of the FFT N is chosen greater than M, bandlimited interpolation of spectral samples is performed. Typically N is chosen to be a power of 2 to speed up the computation of the FFT.
- The hope size R determines the temporal resolution.

Listing 3.4. Matlab

```
[x, Fs, nbits] = wavread('../file sources/flute.wav');

M = floor(0.050*Fs);     %window length (50msec)
R = floor(M/4);          %hop size
N = 2^14;                %power of 2 larger than M + N_h - 1

w = hanning(M);

xm = zeros(M,1); ym = zeros(M,1);

Nframes = floor((length(x)-N)/R) + 1;

STFT = zeros(Nframes, N/2);

for m = 0:Nframes

    xm = x(m*R+1:m*R+M);

    ym = w.*xm;

    temp = abs(fft(ym, N));

    STFT(m+1,:) = temp(1:N/2);
end

STFT = fliplr(STFT)';
t = [0:Nframes - 1]*R/Fs;
f = Fs*[0:N/2 - 1]/N;
imagesc(t, f, 10*log10(abs(STFT).^2))
xlabel('time (sec)');
ylabel('frequency (Hz)');
```



a) Flute b) Guitar

Fig. 3.13. Short-time Fourier transform of a flute (a) and a guitar (b).

4

Digital filter design techniques

The design of a digital filter involves three basic steps:

- the specification of the desired properties of the system
- the approximation of these properties using a discrete-time system
- the realization of the system using finite precision arithmetic

In this chapter, we consider the first two steps, describing a few simple filter design algorithms, distinguishing between IIR and FIR filter design. Further details about more sophisticated design techniques and the implementation of digital filters using finite precision arithmetic can be found in the references listed at the end of this chapter.

4.1 Filter specifications

Filter specifications are often given in the frequency domain. For the case of lowpass filter, for example, the specifications take the form of a tolerance scheme, such as depicted in Figure 4.1.

There is a passband wherein the magnitude of the response must approximate 1 with an error of $\pm\delta$, i.e.

$$1 - \delta_1 \leq |H(e^{j\omega})| \leq 1 + \delta_1, \quad |\omega| \leq \omega_p \tag{4.1}$$

There is a stopband in which the magnitude response must approximate zero with an error less than δ_2, i.e.

$$|H(e^{j\omega})| \leq \delta_2, \quad \omega_s \leq |\omega| \leq \pi \tag{4.2}$$

To make it possible to approximate the ideal lowpass filter in this way we must also provide a transition band of nonzero width $(\omega_s - \omega_p)$ in which the magnitude response drops smoothly from the passband to the stopband

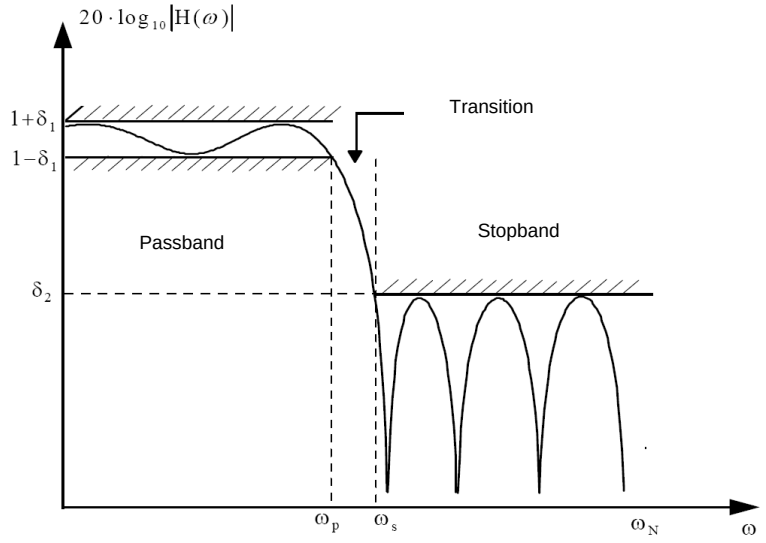


Fig. 4.1. Tolerance limits for approximation of ideal lowpass filter

4.2 IIR filter design

Design of IIR filters from analogue filters

The traditional approach consists in designing digital IIR filters transforming a known analogue filter. This is a reasonable approach because:

- the art of analogue filter design is highly advanced
- many useful analogue design methods have relatively simple closed-form design formulas
- in many applications it is of interest to use a digital filter to simulate the performance of an analogue linear time-invariant filter

The filter design algorithm proceeds as follows

1. Provide the specifications of the digital filter in the frequency domain $H_d(\omega)$, $-\pi \leq \omega \leq +\pi$.
2. Translate the specifications of the digital filter into equivalent specifications of an analogue filter $H_a(\Omega)$, $-\infty \leq \Omega \leq +\infty$
3. Design an analogue filter $H_a(\Omega)$ that satisfies the specifications

4. Find the transfer function $H_a(s)$ of the analogue filter corresponding to the frequency response $H_a(\Omega)$.

5. Find the transfer function $H(z)$ of the digital filter by applying a proper transformation from the s-domain to the z-domain.

Before describing this procedure in detail, let us review some important concepts related to analogue filters. Consider an analogue system function

$$H_a(s) = \frac{\sum_{k=0}^{M} d_k s^k}{\sum_{k=0}^{N} c_k s^k} = \frac{Y_a(s)}{X_a(s)} \tag{4.3}$$

where $x_a(t)$ is the input and $y_a(t)$ is the output and $X_a(s)$ and $Y_a(s)$ are their respective Laplace transforms. An alternative representation of the same system is obtained by means of the differential equation

$$\sum_{k=0}^{N} c_k \frac{d^k y_a(t)}{dt^k} = \sum_{k=0}^{M} d_k \frac{d^k x_a(t)}{dt^k} \tag{4.4}$$

The corresponding rational system function for digital filters has the form

$$H(z) = \frac{\sum_{k=0}^{M} b_k z^{-k}}{\sum_{k=0}^{N} a_k z^{-k}} = \frac{Y(z)}{X(z)} \tag{4.5}$$

or, equivalently, by the difference equation

$$\sum_{k=0}^{N} a_k y(n-k) = \sum_{k=0}^{M} b_k x(n-k) \tag{4.6}$$

We are interested in finding a transformation that allows us to find a digital system from its analogue counterpart and vice-versa. Specifically, in transforming an analogue system to a digital system we must obtain $h(n)$ or $H(z)$ from the analogue filter design. In such transformations we generally require that the essential properties of the analogue frequency response be preserved in the frequency response of the resulting digital filter.

Loosely speaking, this implies that we want the imaginary axis of the s-plane to map into the unit circle of the z-plane. A second condition is that a stable analogue filter should be transformed to a stable digital filter. That is, if the analogue system has poles only in the left-half s-plane, then the digital filter must have poles only inside the unit circle.

In the following, we consider two transformations that attain this goal: bilinear transformation and impulse response invariance. We will show that only the former is adequate as far as filter design is concerned.

Bilinear transformation

Consider the following first-order differential equation

$$c_1 y_a'(t) + c_0 y_a(t) = d_0 x_a(t) \tag{4.7}$$

where $y_a'(t)$ is the first derivative of $y_a(t)$. The corresponding analogue system function is

$$H_a(s) = \frac{d_0}{c_1 s + c_0} \tag{4.8}$$

We can write $y_a(t)$ as an integral of $y_a'(t)$, as in

$$y_a(t) = \int_{t_0}^{t} y_a'(t)dt + y_a(t_0) \tag{4.9}$$

In particular, if $t = nT$ and $t_0 = (n-1)T$,

$$y_a(nT) = \int_{(n-1)T}^{nT} y_a'(t)dt + y_a((n-1)T) \tag{4.10}$$

If the integral is approximated by the trapezoidal rule, we can write

$$y_a(nT) = y_a((n-1)T) + \frac{T}{2}[y_a'(nT) + y_a'((n-1)T)] \tag{4.11}$$

However, from equation (4.7)

$$y_a'(nT) = \frac{-c_0}{c_1} y_a(nT) + \frac{d_0}{c_1} x_a(nT) \tag{4.12}$$

Substituting into equation (4.11) we obtain

$$[y(n) - y(n-1)] = \frac{T}{2}\left[\frac{-c_0}{c_1}(y(n) + y(n-1)) + \frac{d_0}{c_1}(x(n) + x(n-1)\right] \tag{4.13}$$

where $x(n) = x_a(nT)$ and $y(n) = y_a(nT)$. Taking the z-transform and solving for $H(z)$ gives

$$H(z) = \frac{Y(z)}{X(z)} = \frac{d_0}{c_1 \frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}} + c_0} \tag{4.14}$$

From equation (4.8) it is clear that $H(z)$ is obtained from $H_a(s)$ by substitution

$$s = \frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}} \tag{4.15}$$

That is

$$H(z) = H_a(s)|_{s=\frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}}} \tag{4.16}$$

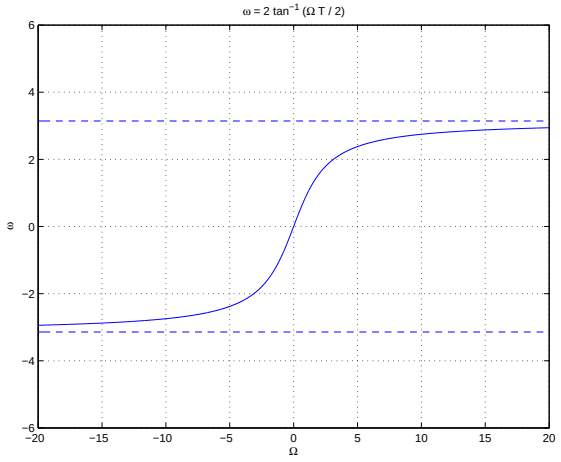


Fig. 4.2. Mapping of the analogue frequency axis onto the unit circle using the bilinear transformation

This result can be shown to hold in general since an Nth order differential equation can be written as a set of N first order equations of the form (4.7).

Solving (4.15) for z gives

$$z = \frac{1 + (T/2)s}{1 - (T/2)s} \tag{4.17}$$

To demonstrate that this mapping has the property that the imaginary axis in the s-plane maps onto the unit circle, consider $z = e^{j\omega}$:

$$\begin{aligned} s &= \frac{2}{T}\frac{1 - z^{-1}}{1 + z^{-1}} \\ &= \frac{2}{T}\frac{1 - e^{-j\omega}}{1 + e^{j\omega}} \\ &= \frac{2}{T}\frac{j\sin(\omega/2)}{\cos(\omega/2)} \\ &= \frac{2}{T} j\tan(\omega/2) = \sigma + j\Omega \end{aligned} \tag{4.18}$$

Thus, for z on the unit circle, $\sigma = 0$ and Ω and ω are related by

$$\frac{T\Omega}{2} = \tan(\omega/2) \tag{4.19}$$

In addition to the fact that the imaginary axis in the s-plane maps to the unit circle in the z-plane, the left half of the s-plane maps to the inside of the unit circle and the right half of the s-plane maps to the outside of the unit circle.

Example: RC system

Consider a RC system, described by the following frequency response

$$V_r(\Omega) = \frac{E(\Omega)}{1 + j\Omega\tau}, \quad \tau = RC \tag{4.20}$$

In the discrete case, $e(t)$ is sampled as $e(n)$. Using the bilinear transformation, the digital transfer function becomes

$$\frac{V(z)}{E(z)} = \frac{1}{1 + \tau\frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}}} = \frac{1 + z^{-1}}{1 + a + (1-a)z^{-1}} \quad a = \frac{2\tau}{T} \tag{4.21}$$

Taking the inverse z transform we get

$$v(n) = \frac{a-1}{a+1}v_{n-1} + \frac{1}{1+a}(e(n) + e(n-1)) \tag{4.22}$$

The pole is within the unit circle, in fact.

$$\left|\frac{a-1}{a+1}\right| < 1 \quad \text{if } \tau > 0 \tag{4.23}$$

Analogue filter prototypes

Filter design starts by selecting an analogue filter prototype that matches the characteristics of the desired frequency response. Below, we consider the following families of analogue filters:

- **Butterworth filters**
  - the squared magnitude frequency response has the form

$$|H_a(\Omega)|^2 = \frac{1}{1 + (\Omega/\Omega_c)^{2N}} \tag{4.24}$$

  - the analogue transfer function is

$$H_a(s) = \frac{\Omega_c^N}{\prod_{i=0}^{N}(s - s_i)} \tag{4.25}$$

    where the poles s_i can be found with these equations (proof is omitted)
    - If N is odd: $\Omega_c e^{jm\pi/N}$, $0 \le m < 2N$, such that $\mathrm{Re}\{s\} < 0$
    - If N is even: $\Omega_c e^{j\pi/2N + m\pi/N}$, $0 \le m < 2N$, such that $\mathrm{Re}\{s\} < 0$
  - magnitude response is maximally flat in the passband

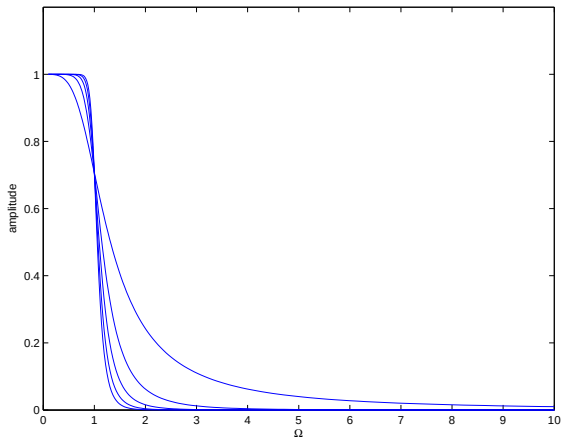


Fig. 4.3. Dependence of Butterworth magnitude characteristics on the order N

- for an N order lowpass filter, this means that the first $2N-1$ derivatives of the squared magnitude function are zero at $\Omega = 0$
- approximation is monotonic in the passband and stopband
- as the parameter N increases, the filter characteristics becomes sharper

- **Chebychev filters**
  - magnitude of the frequency response is either equiripple in the passband and monotonic in the stopband or monotonic in the passband and equiripple in the stopband
  - the accuracy of the approximation is uniformly distributed over the passband or the stopband

- **Elliptic filters**
  - magnitude of the frequency response is equiripple both in the passband and in the stopband
  - it is possible to meet filter specifications with a lower order than Butterworth and Chebychev filters

For the Butterworth filter, we provide a step-by-step design procedure that starts from specifications of the digital filter in the ω domain. As an example, let us consider the design of a Butterworth lowpass digital filter.

1. Provide the specifications of the digital filter in the frequency domain:

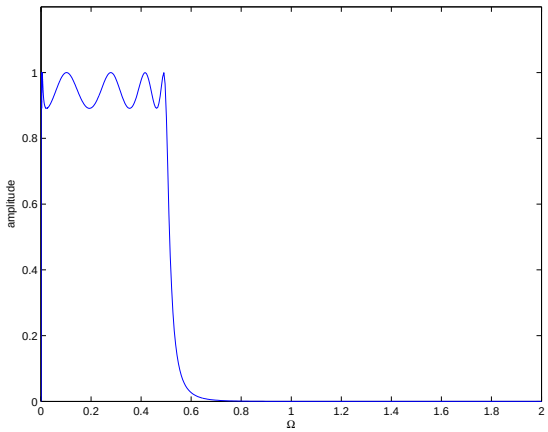


Fig. 4.4. Frequency response of a ninth-order analog Chebychev filter

- passband: $[0 - 0.2\pi]$, constant within 1dB
- stopband: $[0.3\pi, \pi]$, attenuation greater than 15db

$$20\log_{10}|H(e^{j0.2\pi})| \geq -1 \quad 20\log_{10}|H(e^{j0.3\pi})| \leq -15 \tag{4.26}$$

2. Translate the specifications of the digital filter into equivalent specifications of an analogue filter $H_a(\Omega)$. Applying the bilinear transformation we get

$$20\log_{10}|H_a(j2\tan(0.2\pi/2))| \geq -1 \quad 20\log_{10}|H_a(j2\tan(0.3\pi/2))| \leq -15 \tag{4.27}$$

3. Design an analogue filter $H_a(\Omega)$ that satisfies the specifications Solving the previous equations with equality gives

$$1 + \left(\frac{2\tan(0.1\pi)}{\Omega_c}\right)^{2N} = 10^{0.1} \tag{4.28}$$

$$1 + \left(\frac{2\tan(0.15\pi)}{\Omega_c}\right)^{2N} = 10^{1.5} \tag{4.29}$$

so

$$N = \frac{1}{2}\frac{\log[(10^{1.5}-1)/(10^{0.1}-1)]}{\log[\tan(0.15\pi)/\tan(0.1\pi)]} = 5.30466 \tag{4.30}$$

In order to meet the specifications, N must be chosen as 6. If we determine Ω_c by substituting $N = 6$, we obtain $\Omega_c = 0.76622$.

4. Find the transfer function $H_a(s)$ of the analogue filter corresponding to the frequency response $H_a(\Omega)$.

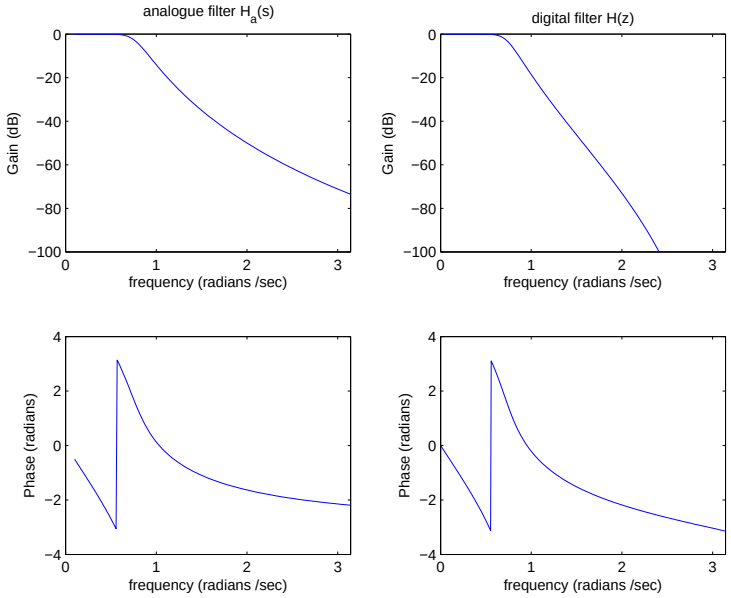


Fig. 4.5. Frequency response of a sixth-order Butterworth filter

5. Find the transfer function $H(z)$ of the digital filter by applying the inverse bilinear transformation from the s-domain to the z-domain.

Listing 4.1. Matlab

```
% digital specs
% [0 - 0.2pi] within 1dB
% [0.3 - pi] below -15dB

% analogue filter design
[c,d] = butter(6,0.76622,'s');
[Ha, Omega] = freqs(c,d);

subplot(2,2,1), plot(Omega, 20*log10(abs(Ha)));
title('analogue filter H_a(s)');
ylabel('Gain (dB)'); xlabel('frequency (radians /sec)'); axis([0 pi -100 0])

subplot(2,2,3), plot(Omega, angle(Ha));
ylabel('Phase (radians)'); xlabel('frequency (radians /sec)'); axis([0 pi -4 4])


% digital filter design
[b,a] = bilinear(c,d,1); [H,omega] = freqz(b,a);

subplot(2,2,2), plot(omega, 20*log10(abs(H)));
title('digital filter H(z)');
ylabel('Gain (dB)'); xlabel('frequency (radians /sec)'); axis([0 pi -100 0])

subplot(2,2,4), plot(omega, angle(H));
ylabel('Phase (radians)'); xlabel('frequency (radians /sec)'); axis([0 pi -4 4])
```

Listing 4.2. Matlab

```
fp  = 0.2; %passband ends (1 --> omega = pi)
fst = 0.3; %passband begins
ap  = 1;    %ripple in the passband (dB)
ast = 15;    %attenuation in the stopband (dB)
```

```
d = fdesign.lowpass(fp,fst,ap,ast); hd = design(d, 'butter');
% hd = design(d, 'cheby1');
% hd = design(d, 'cheby2');
% hd = design(d, 'ellip');

fvtool(hd)

%see also
%fdatool
```

Impulse invariance

Another way to transform an analog filter into a digital filter consists in sampling the impulse response of the analogue filter. Consider, for example, the following frequency response of an analogue filter

$$H_a(j\Omega) = \frac{1}{1 + j\Omega\tau} \longleftrightarrow h(t) = \frac{1}{\tau} e^{-t/\tau} \tag{4.31}$$

Sampling the impulse response gives

$$h(n) = \frac{T}{\tau} e^{-nT/\tau} \longleftrightarrow H(z) = \frac{T}{\tau} \frac{1}{1 - e^{-T/\tau} z^{-1}} \tag{4.32}$$

Stable systems are transformed into stable systems, impulse responses are similar, but frequency responses are rather different apart from very low frequencies and $T << \tau$. In fact, sampling is performed without prefiltering, thus causing aliasing.

In conclusion, it should be noted that the impulse invariance design procedure is only appropriate for essentially bandlimited filters. For example, highpass and bandstop filters would require additional band-limiting to avoid severe aliasing distortions.

Least squares inverse design

In this procedure, the filter is specified in terms of the first L samples of the desired impulse response

$$\{h_d(n)\}, \quad n = 0, 1, \ldots, L-1 \tag{4.33}$$

In our discussion, we shall assume that the filter transfer function is of the form

$$H(z) = \frac{b_0}{1 - \sum_{r=1}^{N} a_r z^{-r}} \tag{4.34}$$

The filter design is based on the criterion that the output of the inverse of $H(z)$ must approximate a unit sample, when the input is $h_d(n)$. If $v(n)$ denotes the output of the inverse system with transfer function $1/H(z)$, then

$$V(z) = \frac{H_d(z)}{H(z)} \tag{4.35}$$

Thus we can write the recursion formula

$$b_0 v(n) = h_d(n) - \sum_{r=1}^{N} a_r h_d(n-r) \tag{4.36}$$

Recall that we want $v(n)$ be a unit sample. Thus it is reasonable to require that

$$b_0 = h_d(0) \tag{4.37}$$

and that $v(n)$ be as small as possible for $n > 0$. Therefore, choose the remaining coefficients so as to minimize

$$E = \sum_{n=1}^{\infty} (v(n))^2 \tag{4.38}$$

From equation (4.36),

$$E = \frac{1}{b_0^2} \sum_{n=1}^{\infty} (h_d(n))^2 - 2 \sum_{n=1}^{\infty} h_d(n) \sum_{r=1}^{N} a_r h_d(n-r) + \sum_{n=1}^{\infty} \left[\sum_{r=1}^{N} a_r h_d(n-r) \right]^2 \tag{4.39}$$

The coefficients a_i that minimize E satisfy the equations

$$\frac{\partial E}{\partial a_i} = 0 \quad i = 1, 2, \ldots, N \tag{4.40}$$

This results in

$$\sum_{r=1}^{N} a_r \sum_{n=1}^{\infty} h_d(n-r) h_d(n-i) = \sum_{n=1}^{\infty} h_d(n) h_d(n-i) \tag{4.41}$$

If we define $\phi(i, r)$ as

$$\phi(i, r) = \sum_{n=1}^{\infty} h_d(n-r) h_d(n-i) \tag{4.42}$$

Then the coefficients a_i satisfy the set of linear equations

$$\sum_{r=1}^{N} a_r \phi(i, r) = \phi(i, 0), \quad i = 1, 2, \ldots, N \tag{4.43}$$

This equations can be solved by any conventional technique. A particularly efficient procedure is given by Levinson (see Section 8.4).

Listing 4.3. Matlab

```
clear
% approximate butterworth filter design
```

```
% fp  = 0.2; %passband ends (1 --> omega = pi)
% fst = 0.3; %passband begins
% ap  = 1;    %ripple in the passband (dB)
% ast = 15;    %attenuation in the stopband (dB)
%
% d = fdesign.lowpass(fp,fst,ap,ast);
% h_filter = design(d, 'butter');
%
% % compute impulse response
% sos = h_filter.sosMatrix;
% [S, K] = size(sos);
%
% g = h_filter.ScaleValues;
%
% b = sos(:,1:K/2);
% a = sos(:,K/2+1:end);
%
% delta = [1; zeros(200,1)];
% y = delta;
%
% for s = 1:S
%     y = g(s)*filter(b(s,:), a(s, :), y);
% end
%
% y = g(S+1)*y;

% approximate IIR zero-pole filter
delta = [1; zeros(200,1)];
a = [1 0.9 0.32 0.2 0.1 0.05];
b = [1 1 1];
y = filter(b, a, delta);

% inverse least squares design
L = 40;
N = 10;

hd = y(1:L);

[phi, lags] = xcorr(hd, N); phi = phi(N+1:end);

[a_ls, e] = levinson(phi,N);

h = filter(sqrt(e), a_ls, delta);

stem(y) hold on stem(h,'r--')
hold off
```

4.3 FIR filter design

Design of FIR filters using windows

The most straightforward approach to FIR filter design is to obtain a finite-length impulse response by truncating an infinite duration impulse response sequence. If we suppose that $H_d(e^{j\omega})$ is an ideal desired frequency response, then

$$H_d(e^{j\omega}) = \sum_{n=-\infty}^{\infty} h_d(n)e^{-j\omega n} \tag{4.44}$$

where $h_d(n)$ is the corresponding impulse response sequence, i.e.

$$h_d(n) = \frac{1}{2\pi}\int_{-\pi}^{+\pi} H_d(e^{j\omega})e^{j\omega n}d\omega \tag{4.45}$$

In general, $h_d(n)$ is of infinite duration and it must be truncated to obtain a finite duration impulse response. Therefore, we can represent $h(n)$ as the product of the desired impulse response and a finite-duration window $w(n)$

$$h(n) = h_d(n)w(n) \tag{4.46}$$

Using the convolution theorem of the DTFT we see that

$$H(e^{j\omega}) = \frac{1}{2\pi}\int_{-\pi}^{+\pi} H_d(e^{j\omega})W(e^{j(\omega-\theta)})d\theta \tag{4.47}$$

That is $H(e^{j\omega})$ is the periodic continuous convolution of the desired frequency response with the Fourier transform of the window. Thus, the frequency response $H(e^{j\omega})$ will be a smeared version of the desired response $H_d(e^{j\omega})$.

If $W(e^{j\omega})$ is narrow compared to variations in $H_d(e^{j\omega})$, then $H(e^{j\omega})$ will look like $H_d(e^{j\omega})$. Thus the choice of window is governed by the desire to have $w(n)$ as short as possible in duration so as to minimize computation in the implementation of the filter, while having $W(e^{j\omega})$ as narrow as possible in frequency so as to faithfully reproduce the desired frequency response. These are conflicting requirements.

The windows described in Section 3 can be conveniently used to this purpose. Through the choice of the window shape and duration, we can exercise some control over the design process. For example, for a given stopband attenuation, it is generally true that N satisfies an equation of the form

$$N = \frac{A}{\Delta\omega} \tag{4.48}$$

where $\Delta\omega$ is the transition width (roughly the width of the main lobe of $W(e^{j\omega})$ and A is a constant that is dependent upon the window shape.

A difficulty with this technique is in the evaluation of $h_d(n)$ from the frequency response $H_d(e^{j\omega})$. If the latter cannot be expressed in terms of simple functions, an approximation to $h_d(n)$ can be obtained using the frequency sampling method described below.

Listing 4.4. Matlab

```
N = 30;   %filter order
Wc = 0.2; %cutoff frequency
b = fir1(N,Wc, kaiser(N+1, 5)); %design lowpass filter
figure(1) freqz(b,1,2048);
figure(2) stem(b);
```

Design of FIR filters using frequency sampling

Given a desired frequency response, the frequency sampling design method designs a filter with a frequency response exactly equal to the desired response at a particular set of frequencies ω_k.

Assume that we wish to design a FIR filter with an impulse response $h(n)$ for $n = 0, \ldots, M-1$, that approximates a desired frequency response $H_d(\omega)$. In order to do this, we sample the frequency response $H_d(\omega)$ and we compute $h(n)$ by performing the IDFT, i.e.:

$$H(k) = H_d(e^{j\omega})\big|_{\omega=2\pi k/M} \tag{4.49}$$

$$h(n) = \frac{1}{M}\sum_{k=0}^{M-1} H(k)e^{j\frac{2\pi kn}{M}} \tag{4.50}$$

The frequency response of the FIR filter obtained above is given by its DTFT, i.e.:

$$\begin{aligned} H(e^{j\omega}) &= \sum_{n=0}^{M-1} h(n)e^{-j\omega n} \\ &= \sum_{n=0}^{M-1}\left[\frac{1}{M}\sum_{k=0}^{M-1} H(k)e^{j\frac{2\pi kn}{M}}\right]e^{-j\omega n} \\ &= \frac{1}{M}\sum_{k=0}^{M-1} H(k)\sum_{n=0}^{M-1} e^{j\left(\frac{2\pi k}{M}-\omega\right)n} \end{aligned} \tag{4.51}$$

This expression means that the actual frequency response $H(e^{j\omega})$ matches the desired frequency response $H_d(e^{j\omega})$ at frequencies $\omega_k = \frac{2\pi k}{M}$. At other frequencies, $H(e^{j\omega})$ is obtained by interpolating between $H(k)$ samples. The main problem of frequency sampling design is that $H(e^{j\omega})$ might be very different from $H_d(e^{j\omega})$ at points $\omega \neq \omega_k$.

In the time domain, frequency sampling causes aliasing. In fact

$$h(n) = \frac{1}{M}\sum_{k=0}^{M-1} H(k)e^{j\frac{2\pi kn}{M}} = \frac{1}{M}\sum_{k=0}^{M-1} H_d(e^{j2\pi k/M})e^{j\frac{2\pi kn}{M}} = \sum_{r=-\infty}^{\infty} h_d(n+rM) \tag{4.52}$$

The last equality stems from the Poisson formula that is introduced in Section 3.

Figure 4.6 shows the desired (blue) and actual (green) frequency response obtained by sampling the desired frequency response of an ideal low-pass filter with cut-off frequency $\pi/2$. The phase of the desired filter is chosen to be linear, i.e. $\angle H_d(e^{j\omega}) = e^{j\omega(M-1)/2}$. In design (a) the transition is quite sharp, thus causing a noticeable overshoot at the start of the transition region. In design (b) the first zero sample in the transition region

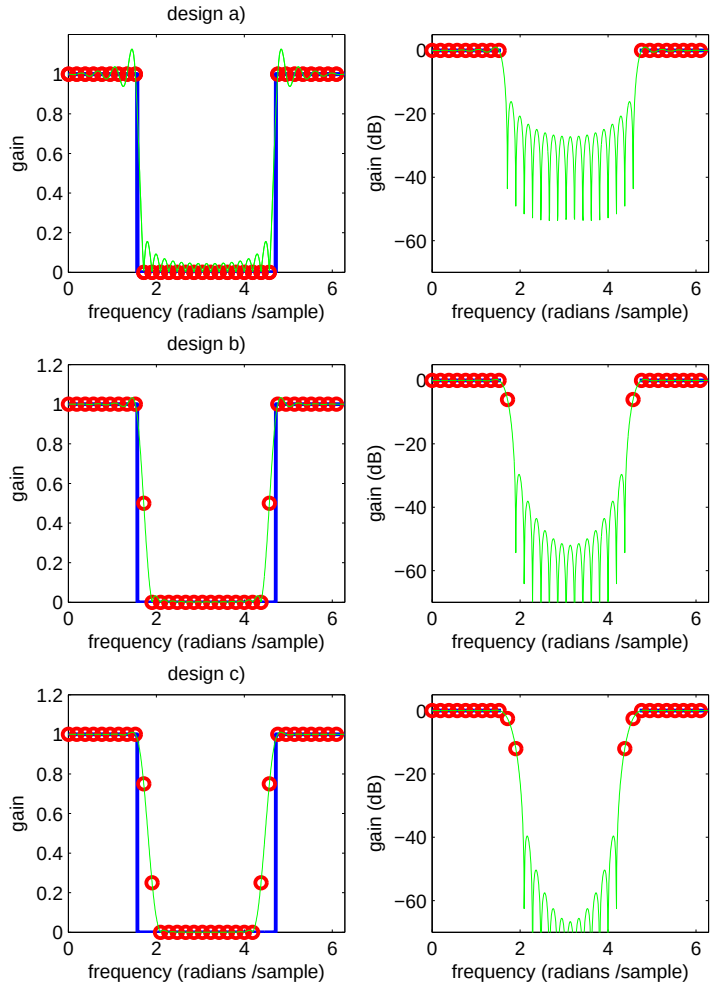


Fig. 4.6. Design of a low-pass FIR filter by frequency sampling.

has been changed to 0.5, resulting in a wider transition region, but far less overshoot. In design (c) the first two samples in the transition region are set to 0.75 and 0.25, enlarging the transition region but further decreasing the overshoot.

Listing 4.5. Matlab

```
% ideal low-pass filter
Fc = 0.25; %normalized cut-off frequency
M = 33; % number of filter taps
N = 1024

kd = [0: N - 1]/N;
Hd = zeros(length(kd), 1);
Hd(kd < Fc | kd > 1 - Fc) = 1;
```

```
k = [0: M - 1]/M;
H = zeros(length(k), 1); H(k < Fc | k > 1 - Fc) = 1;

h = fftshift(ifft(H));
Hr = freqz(h,1,2*pi*kd);

subplot(3,2,1)
plot(2*pi*kd, abs(Hd), 'b', 'LineWidth',2);
axis([0 2*pi 0 1.2])
hold on
plot(2*pi*k, abs(H), 'ro', 'LineWidth',2);
plot(2*pi*kd, abs(Hr), 'g');
title('design a)');
xlabel('frequency (radians /sample)');
ylabel('gain');

subplot(3,2,2)
plot(2*pi*kd, 20*log10(abs(Hd)), 'b', 'LineWidth',2);
axis([0 2*pi -70 5])
hold on
plot(2*pi*k, 20*log10(abs(H)), 'ro', 'LineWidth',2);
plot(2*pi*kd, 20*log10(abs(Hr)), 'g');
xlabel('frequency (radians /sample)');
ylabel('gain (dB)');
```

Listing 4.6. Matlab

```
f = [0 0.50 0.50 1];
m = [1    1    0 0];
N = 30;  %filter order
b = fir2(N,f,m,1024,1,rectwin(N+1))

% f = [0 0.2 0.2 0.4 0.4 0.6 0.6 0.8 0.8 1.0];
% m = [0   0   1   1   0   0   1   1   0   0];
% N = 31; %filter order
% b = fir2(N, f, m);

[H, w] = freqz(b, 1, 1024);

figure(2)
subplot(2,1,1) plot(f, m) hold on
plot(w/pi,abs(H),'r--') hold off

subplot(2,1,2) stem(b)
```

4.4 Optimum FIR filter design

Suppose that we want to design a lowpass filter according to the tolerance scheme showed in Fig 4.1. There are, of course, five parameters to specify: the passband upper limit ω_p, the stopband lower limit ω_s, the maximum error in the passband (δ_1) and in the stopband (δ_2), and the filter order N. We want the filter to be real and linear phase, so we can write its frequency response as:

$$H(e^{j\omega}) = h(0) + \sum_{n=1}^{M} 2h(n)\cos(\omega n) \tag{4.53}$$

where $h(n)$ is the symmetric impulse response of the filter and $N = 2M + 1$. A causal system can be simply obtained by delaying $h(n)$ by M samples. Design algorithms have been developed in which some of the parameters are fixed and the best values of

the remaining ones are found by an iterative procedure. Parks and McClellan found a solution to the problem in the particular case in which the fixed parameters are M, ω_p and ω_s. The desired filter frequency response is:

$$H_d(e^{j\omega}) = \begin{cases} 1 \; 0 \leq \omega \leq \omega_p \\ 0 \; \omega_s \leq \omega \leq \pi \end{cases} \tag{4.54}$$

We define over the passband and the stopband the approximation error function

$$E(\omega) = W(\omega)[H_d(e^{j\omega}) - H(e^{j\omega})] \tag{4.55}$$

where $W(\omega)$ is a weighting function that specifies the relative sizes of the passband and stopband approximation errors and can be defined as:

$$W(\omega) = \begin{cases} 1/K & 0 \leq \omega \leq \omega_p \\ 1 & \omega_s \leq \omega \leq \pi \end{cases} \tag{4.56}$$

K should be equal to the desired ratio δ_1/δ_2, so that the optimum filter is the one which minimizes $\max|E(\omega)|$, which is equivalent to minimize δ_2. Parks and McClellan showed that the following theorem holds:

Alternation Theorem. *in order that $H(e^{j\omega})$ be the unique best approximation to $H_d(e^{j\omega})$ over the passband and the stopband, it is necessary and sufficient that $E(\omega)$ exhibits at least $M + 2$ "alternations", thus $E(\omega_i) = -E(\omega_{i-1}) = \max|E(\omega)|$ with $\omega_0 \leq \omega_1 \leq \omega_2 \leq \ldots \leq \omega_{M+1}$ and ω_i contained either in the passband or in the stopband.*

Since $H_d(e^{j\omega})$ is piecewise constant, the frequencies ω_i corresponding to the peaks in the error function (4.55) are the frequencies at which $H(e^{j\omega})$ meets the error tolerance, i.e. the frequencies ω in the passband at which $H(e^{j\omega}) = 1 \pm \delta_1$ and the frequencies ω in the stopband at which $H(e^{j\omega}) = \pm\delta_2$.

We want $H(e^{j\omega})$ to have an equiripple behavior, so that the approximation error is spread out uniformly in frequency. It can be showed that this implies that $H(e^{j\omega})$ has either a local maximum or a local minimum at $\omega = 0$ and $\omega = \pi$. Furthermore, there are at most $M - 1$ local extrema in the opened interval $(0, \pi)$ and we have $H(e^{j\omega}) = 1 - \delta_1$ in $\omega = \omega_p$ and $H(e^{j\omega}) = \delta_2$ in $\omega = \omega_s$. Thus, we always have $(M-1)+2 = M+1$ peaks in the error function. In accord to the alternation theorem, we need one more peak to reach optimality, and we can get it just by setting either $H(e^{j\omega}) = 1 \pm \delta_1$ in $\omega = 0$ or $H(e^{j\omega}) = \pm\delta_2$ in $\omega = \pi$. We can also choose to impose both the conditions, obtaining $M + 3$ peaks in the error function.

These considerations lead Parks and McClellan to develop the following algorithm:

1. Estimate $M+2$ frequencies $\omega_0 \leq \omega_1 \leq \omega_2 \leq \ldots \leq \omega_{M+1}$ at which $|E(\omega)|$ has to be maximum. Note that, since ω_p and ω_s are fixed, for some k such that $0 < k < M+1$ we need $\omega_k = \omega_p$ and $\omega_{k+1} = \omega_s$. Furthermore, we can choose $\omega_0 = 0$ or $\omega_{M+1} = \pi$ or both.

2. Compute the error function in the estimated frequencies and impose that it equals the magnitude of the peak error p, obtaining by means of (4.53) and (4.55) the set of $M+2$ equations

$$W(\omega_i)[H_d(e^{j\omega}) - h(0) - \sum_{n=1}^{M} 2h(n)\cos(\omega_i n)] = (-1)^{i+1}p \qquad i = 0, 1, \ldots, M+1 \tag{4.57}$$

   in the $M+2$ unknowns p and $h(n)$ for $n = 0, 1, \ldots, M$.

3. Solve the set of equations (4.57) only for p and determine the trigonometric polynomial which has the correct value at the frequencies ω_i, i.e. $1 \pm Kp$ in the passband and $\pm p$ in the stopband. It's possible, of course, to solve the set of equations (4.57) for all the $M+2$ unknowns, but this is not an efficient approach.

4. Get the new estimate of the extremum frequencies ω_i as the frequencies corresponding to the peaks of the interpolating polynomial, and iterate until convergence, i.e. until the variation of p is lower than a fixed threshold.

5. Finally, compute $h(n)$ as the IDFT of the sampled version of the optimum frequency response, i.e. the last interpolating polynomial.

Estimation of FIR filter order

It has been shown that the filter order N is related to the filter specifications by means of the following approximate formula

$$N \simeq \frac{-20\log_{10}(\sqrt{\delta_p \delta_s}) - 13}{14.6(\omega_s - \omega_p)/2\pi} \tag{4.58}$$

Note from the above formula that the filter order N of a FIR filter is inversely proportional to the transition bandwidth $(\omega_s - \omega_p)$ and does not depend on the actual location of the transition band. This implies that a sharp cutoff FIR filter with a narrow transition band would be of very long length, whereas a FIR filter with a wide transition band will have a very short length. Another interesting property is that the length depends

on the product $\delta_p\delta_s$. This implies that if the values of δ_p and δ_s are interchanged, the length remains the same.

The formula provides a reasonably good estimate of the filter order in the case of FIR filters with moderate passband width and may not work well in the case of very narrow passband or very wide passband filters. In the case of a narrowband filter, the stopband ripple essentially controls the order and an alternative formula provides a more reasonable estimate of the filter order

$$N \simeq \frac{-20\log_{10}(\delta_s) + 0.22}{(\omega_s - \omega_p)/2\pi} \tag{4.59}$$

On the other hand, in the case of a very wide band filter, the passband ripple has more effect on the order, and a more reasonable estimate of the filter order can be obtained using the following formula

$$N \simeq \frac{-20\log_{10}(\delta_p) + 5.94}{27(\omega_s - \omega_p)/2\pi} \tag{4.60}$$

4.5 Selection of filter type

An important issue is the selection of filter type, i.e. whether an IIR of a FIR digital filter is to be employed.

For IIR digital filters, the IIR transfer function is a real rational function of z^{-1}:

$$H(z) = \frac{b_0 + b_1 z^{-1} + \ldots + b_M z^{-1}}{1 + a_1 z^{-1} + \ldots + a_N z^{-N}} \tag{4.61}$$

Moreover, $H(z)$ must be a stable transfer function, and for reduced computational complexity, it must be of lowest order N.

On the other end, for FIR filter design, the FIR transfer function is a polynomial in z^{-1}:

$$H(z) = \sum_{k=0}^{N} h(n) z^{-n} \tag{4.62}$$

For reduced computational complexity, the degree N of $H(z)$ must be as small as possible. In addition, if linear phase is desired, then the FIR filter coefficients must satisfy the constraint:

$$h(n) = h(N - 1 - n) \tag{4.63}$$

There are several advantages in using a FIR filter, since it can be designed with exact linear phase and the filter structure is always stable when quantized. However, in most

cases, the order N_{FIR} of an FIR filter is considerably higher than the order N_{IIR} of an equivalent IIR filter meeting the same magnitude specifications. It has been shown that for most practical filter specifications, the ratio N_{FIR}/N_{IIR} is typically of the order of tens or more and, as a result, the IIR filter is usually computationally more efficient. However, if the group delay of the IIR filter is equalized by cascading an allpass equalizer, then the savings in computation may no longer be that significant. In many applications, the linearity of the phase response of the digital filter is not an issue, making the IIR filter preferable because of the lower computational requirements.

As shown in Figure 4.7, nowadays some of the disadvantages of IIR filters are no longer valid, if we assume to be able to perform block-wise processing (thus introducing delay). Let us partition the input signal into blocks of length M. This length must be much larger that the effective length L of the impulse response of the IIR filter. Each block can be filtered with the difference equation, but the output of the block will be L samples longer than the input block. Successive blocks of the output must then be pieced together using overlap-and-add idea. The first L samples computed by the filter for each block must be added to the last L samples produced by the filter for the previous block. Other samples are used directly. Let us analyze the limitations conventionally attributed to IIR filters:

- Stability: If we allow block-wise processing, IIR recursive filters no longer need to be causal recursive filters. We used to be afraid of a pole outside the unit circle because it meant instability. If we use a filter iteration that runs backward in time, then poles that are outside the unit circle are not unstable. So we can have a stable recursive filter with poles everywhere except exactly on the unit circle.
- Linear phase: The same idea of filtering backward in time allows to achieve linear phase. For example, suppose that $H(z)$ is the transfer function of any filter. We apply that filter to the data first in one time direction and then again in the opposite time direction. The resulting filter's transfer function is $H(z)H^*(1/z)$ which has real values everywhere on the unit circle. In fact, if $H(z)$ has any nontrivial poles inside the unit circle, $H^*(1/z)$ has matching poles outside the unit circle. A real frequency response means one with zero phase. By processing one block at the time and filtering in both directions we can achieve zero phase with recursive filters.

References

- [1] Oppenheim, A.V., Schafer, R.W. Digital Signal Processing, Prentice Hall, January 1975, ISBN 0132146355.

PROPERTY TRANSFER FUNCTION	1970 FIR ZEROS ONLY	1970 IIR POLES AND/ OR ZEROS	1980 FIR ZEROS ONLY	1980 IIR POLES AND/ OR ZEROS	NOW FIR ZEROS ONLY	NOW IIR POLES AND/ OR ZEROS
DESIGN METHODS FOR FREQUENCY SELECTIVITY	SUB-OPTIMAL USING WINDOWS	OPTIMAL ANALYTIC, CLOSED FORM	OPTIMAL USING ITERATIVE METHODS	OPTIMAL ANALYTIC, CLOSED FORM	OPTIMAL USING ITERATIVE METHODS	OPTIMAL ANALYTIC, CLOSED FORM
MULTIPLICATIONS/REGISTERS NEEDED FOR SELECTIVITY	MANY	FEW	MORE	FEWER	MORE	FEWER
CAN BE EXACTLY ALLPASS	NO	YES	NO	YES	NO	YES
UNSTABLE	NEVER	FOR POLES p_i, $\|p_i\| > 1$	NEVER	FOR POLES p_i, $\|p_i\| > 1$	NEVER	NEVER
DEADBAND EXISTS	NO	YES	NO	YES	NO	NO
CAN BE EXACTLY LINEAR PHASE	YES	NO	YES	NO	YES	YES
CAN BE ADAPTIVE			YES	DIFFICULT OR IMPOSSIBLE	YES	DIFFICULT OR IMPOSSIBLE
OPPORTUNITIES FOR PARALLELISM			MANY	SOME	MANY	MANY
HILBERT TRANSFORMER	INEFFICIENT	IMPRACTICAL BECAUSE NOT CAUSAL	INEFFICIENT	IMPRACTICAL BECAUSE NOT CAUSAL	INEFFICIENT	EFFICIENT

Fig. 4.7. A comparison of the nonrecursive (FIR) and recursive (FIR) filters over the years. Colored boxes indicate when one technique offers an advantage (in gray) over the other (in orange) [3].

- [2] Mitra, A.K., Digital Signal Processing - A computer based approach, McGraw-Hill, Third edition, January 2005, ISBN 0073048372
- [3] Rader, C.M., The Rise and Fall of Recursive Digital Filters, Signal Processing Magazine, IEEE Volume 23, Issue 6, Nov. 2006, Page(s):46 - 49

5

Introduction to multirate processing

Multirate processing techniques are used in several application scenarios:

- subband coding of speech and image
- beamforming (steered microphone arrays)
- spectrum estimation

5.1 Downsampling

The operation of downsampling by factor M describes the process of keeping every M^{th} sample and discarding the rest. This is denoted by $\downarrow M$ in block diagrams, as in Figure 5.1.

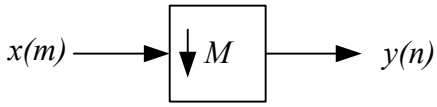


Fig. 5.1. Downsampling by a factor M

Formally, downsampling can be written as $y(n) = x(nM)$ in the z domain:

$$\begin{aligned} Y(z) &= \sum_{n=-\infty}^{+\infty} y(n) z^{-n} = \sum_{n=-\infty}^{+\infty} x(nM) z^{-n} = \\ &= \sum_{m=-\infty}^{+\infty} x(m) \left[\frac{1}{M} \sum_{p=0}^{M-1} e^{j2\pi pm/M} \right] z^{-m/M} \end{aligned} \tag{5.1}$$

where:

$$\frac{1}{M}\sum_{p=0}^{M-1} e^{j2\pi pm/M} = \begin{cases} 1 \text{ if } m \text{ is multiple of } M \\ 0 \text{ otherwise} \end{cases} \tag{5.2}$$

$$\begin{aligned} Y(z) &= \frac{1}{M}\sum_{p=0}^{M-1}\sum_{m=-\infty}^{+\infty} x(m)\left[e^{-j2\pi p/M} z^{\frac{1}{M}}\right]^{-m} = \\ &= \frac{1}{M}\sum_{p=0}^{M-1} X(e^{-j2\pi p/M} z^{\frac{1}{M}}) \end{aligned} \tag{5.3}$$

Translating to the frequency domain (assuming $T = 1$):

$$Y(e^{j\omega}) = \frac{1}{M}\sum_{p=0}^{M-1} X(e^{j\frac{\omega-2\pi p}{M}}) \tag{5.4}$$

As shown in Figure 5.2, downsampling expands each 2π-periodic repetition of $X(e^{j\omega})$ by a factor of M along the ω axis, and reduces the gain by a factor of M. If $x(m)$ is not bandlimited to π/M , aliasing may result from spectral overlap.

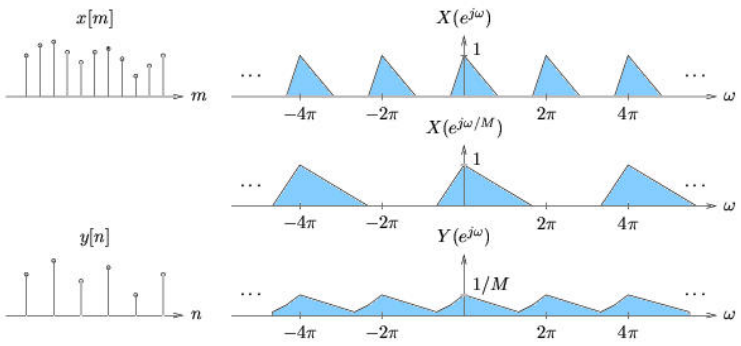


Fig. 5.2. Downsampling by a factor M

Listing 5.1. Matlab

```
N = 41; %try a low value of N to see aliasing
x = bartlett(N);
x_down = x(1:2:end);

[H, w] = freqz(x, 1, 1024);
[H_down, w] = freqz(x_down, 1, 1024);

plot(w, 10*log10(abs(H).^2));
hold on
plot(w, 10*log10(abs(H_down).^2),'r--');
hold off
```

5.2 Upsampling

The operation of upsampling by factor L describes the insertion of $L-1$ zeros between every sample of the input signal. This is denoted by L in block diagrams, as in Figure 5.3.

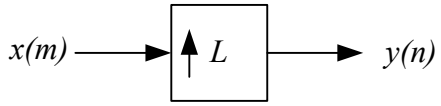


Fig. 5.3. Upsampling by a factor L

Formally, upsampling can be expressed in the z domain as:

$$Y(z) = \sum_{n=-\infty}^{+\infty} y(n)z^{-n} = \sum_{n=-\infty}^{+\infty} x(\frac{n}{L})z^{-n} = \sum_{k=-\infty}^{+\infty} x(k)(z^L)^{-k} = X(z^L) \tag{5.5}$$

In the frequency domain:

$$Y(e^{j\omega}) = X(e^{j\omega L}) \tag{5.6}$$

As shown in Figure 5.4, upsampling compresses the DTFT by a factor of L along with the ω axis.

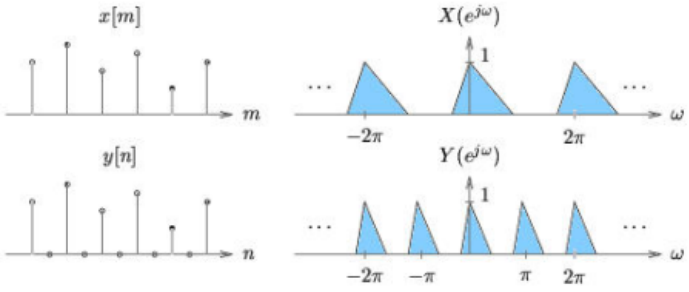


Fig. 5.4. Upsampling by a factor L

Listing 5.2. Matlab

```
N = 21;
x = bartlett(N);
x_up = zeros(2*N,1);
x_up(1:2:end) = x;

[H, w] = freqz(x, 1, 1024);
[H_up, w] = freqz(x_up, 1, 1024);
```

```
plot(w, 10*log10(abs(H).^2));
hold on
plot(w, 10*log10(abs(H_up).^2),'r--');
hold off
```

5.3 Decimation

Decimation is the process of filtering and downsampling a signal to decrease its effective sampling rate, as illustrated in Figure 5.5. The filtering is employed to prevent aliasing that might otherwise result from downsampling.

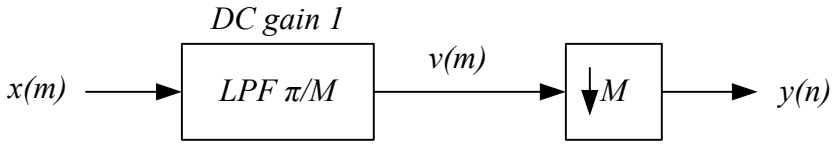


Fig. 5.5. Decimation by a factor M

To be more specific, say that:

$$x_c(t) = x_l(t) + x_b(t) \tag{5.7}$$

where $x_l(t)$ is a lowpass component bandlimited to $\frac{1}{2MT}$ Hz and $x_b(t)$ is a bandpass component with energy between $\frac{1}{2MT}$ and $\frac{1}{2T}$Hz. If sampling $x_c(t)$ with interval T yields an unaliased discrete representation $x(m)$, then decimating $x(m)$ by a factor of M will yield $y(n)$, an unaliased MT-sampled representation of lowpass component $x_l(t)$.

We offer the following justification of the previously described decimation procedure. From the sampling theorem, we have:

$$X(e^{j\omega}) = \frac{1}{T}\sum_{k=-\infty}^{+\infty} X_l(e^{j\frac{\omega-2\pi k}{T}}) + \frac{1}{T}\sum_{k=-\infty}^{+\infty} X_b(e^{j\frac{\omega-2\pi k}{T}}) \tag{5.8}$$

The bandpass component $X_b(\cdot)$ is then removed by π/M-lowpass filtering, giving

$$V(e^{j\omega}) = \frac{1}{T}\sum_{k=-\infty}^{+\infty} X_l(e^{j\frac{\omega-2\pi k}{T}}) \tag{5.9}$$

Finally, downsampling yields:

$$Y(e^{j2\pi f}) = \frac{1}{MT}\sum_{p=0}^{M-1}\sum_{k=-\infty}^{+\infty} X_l(e^{j\frac{\frac{\omega-2\pi p}{M}-2\pi k}{T}})$$
$$= \frac{1}{MT}\sum_{p=0}^{M-1}\sum_{k=-\infty}^{+\infty} X_l(e^{j\frac{\omega-2\pi(kM+p)}{MT}}) = \frac{1}{MT}\sum_{l=-\infty}^{+\infty} X_l(e^{j\frac{\omega-2\pi l}{MT}}) \qquad (5.10)$$

which is a MT-sampled version of $x_l(t)$. A frequency domain illustration for $M = 2$ appears in Figure 5.6.

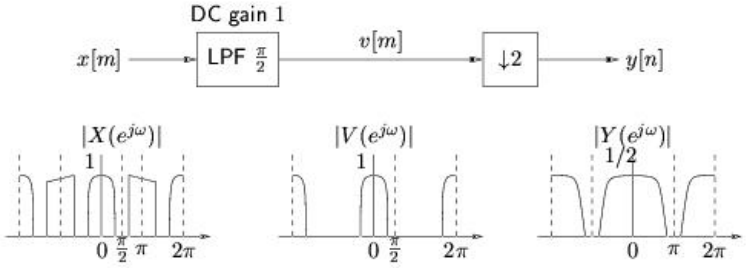


Fig. 5.6. Decimation by a factor 2. (there is an error in the left figure, as the spectrum needs to be symmetric with respect to $-\pi$ and $+\pi$

Listing 5.3. Matlab

```
M = 4;
b = fir1(30,1/M);

figure(1)
freqz(b, 1)

N = 41;
x = [zeros(50, 1); bartlett(N); zeros(50,1)];
x_filt = conv(b, x);
x_dec = x_filt(1:M:end);

[H, w] = freqz(x, 1, 1024);
[H_filt, w] = freqz(x_filt, 1, 1024);
[H_dec, w] = freqz(x_dec, 1, 1024);

figure(2)
plot(w, 10*log10(abs(H).^2));
hold on
plot(w, 10*log10(abs(H_filt).^2),'b-.');
plot(w, 10*log10(abs(H_dec).^2),'r--');
hold off

figure(3)
subplot(3,1,1), plot(x)
subplot(3,1,2), plot(x_filt)
subplot(3,1,3), plot(x_dec)

% see also
% decimate(x, M, N, 'FIR') %FIR window method
% or
% decimate(x, M) %chebychev IIR filter
```

5.4 Interpolation

Interpolation is the process of upsampling and filtering a signal to increase its effective sampling rate. To be more specific, say that $x(m)$ is an (unaliased) T-sampled version of $x_c(t)$ and $v(n)$ is an L-upsampled version version of $x(m)$. If we filter $v(n)$ with an ideal π/L-bandwidth lowpass filter (with DC gain L) to obtain $y(n)$, then $y(n)$ will be a T/L-sampled version of $x_c(t)$. This process is illustrated in Figure 5.7.

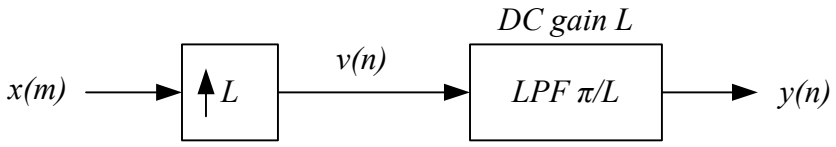


Fig. 5.7. Interpolation by a factor L

We justify our claims about interpolation using frequency-domain arguments. From the sampling theorem, we know that T-sampling $x_c(t)$ to create $x(n)$ yields:

$$X(e^{j\omega}) = \frac{1}{T}\sum_{k=-\infty}^{+\infty} X_c\left(e^{j\frac{\omega-2\pi k}{T}}\right) \tag{5.11}$$

After upsampling by a factor L, we obtain:

$$V(e^{j\omega}) = \frac{1}{T}\sum_{k=-\infty}^{+\infty} X_c\left(e^{j\frac{\omega L-2\pi k}{T}}\right) = \frac{1}{T}\sum_{k=-\infty}^{+\infty} X_c\left(e^{j\frac{\omega-\frac{2\pi k}{L}}{\frac{T}{L}}}\right) \tag{5.12}$$

Lowpass filtering with cutoff π/L and gain L yields:

$$Y(e^{j\omega}) = \frac{L}{T}\sum_{l=-\infty}^{+\infty} X_c\left(e^{j\frac{\omega-2\pi l}{\frac{T}{L}}}\right) \tag{5.13}$$

This process yields a T/L-sampled version of $x_c(t)$. Figure 5.8 illustrates these frequency-domain arguments for $L = 2$.

Listing 5.4. Matlab

```
M = 8; N = 30;
b = fir1(N,1/M,kaiser(N+1, 5));     % sinc interpolator (approx)
%b = (1/M)*ones(M,1);          % hold
%b = (1/M)*bartlett(2*M+1); % linear interpolation

figure(1)
freqz(b, 1)

N = 41;
```

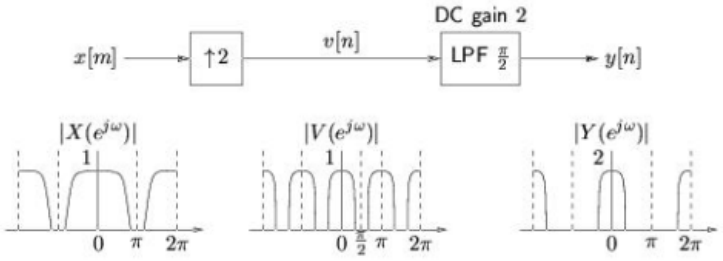


Fig. 5.8. Interpolation by a factor 2

```
x = [zeros(50, 1); hanning(N); zeros(50,1)];

x_up = zeros(M*length(x), 1); x_up(1:M:end) = x;

x_int = filter(M*b, 1, x_up);

[H, w] = freqz(x, 1, 1024);
[H_up, w] = freqz(x_up, 1, 1024);
[H_int, w] = freqz(x_int, 1, 1024);

figure(2)
plot(w, 10*log10(abs(H).^2));
hold on
plot(w, 10*log10(abs(H_up).^2),'b-.');
plot(w, 10*log10(abs(H_int).^2),'r');
hold off

figure(3)
subplot(3,1,1), plot(x)
subplot(3,1,2), plot(x_up)
subplot(3,1,3), plot(x_int)

% see also
% interp(x, M);
```

5.5 Multirate identities

Interchange of upsampling and filtering

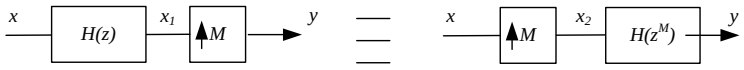


Fig. 5.9. Interchange of upsampling and filtering

Proof:

$Y(z) = X_1(z^M) = H(z^M)X(z^M)$. Since $X_2(z) = X(z^M)$, filter with $H(z^M)$ to get $Y(z)$.

Fig. 5.10. Interchange of downsampling and filtering

Interchange of downsampling and filtering

Proof:

$$X_1(z) = H(z^M)X(z) \tag{5.14}$$

$$\begin{aligned} Y(z) &= \frac{1}{M}\sum_{k=0}^{M-1} H((e^{j2\pi k/M}z^{1/M})^M X(e^{j2\pi k/M}z^{1/M}) \\ &= \frac{1}{M}\sum_{k=0}^{M-1} H(z)X(e^{j2\pi k/M}z^{1/M}) \\ &= H(z)\frac{1}{M}\sum_{k=0}^{M-1} X(e^{j2\pi k/M}z^{1/M}) \\ &= H(z)X_2(z) \end{aligned} \tag{5.15}$$

5.6 Polyphase filters

Polyphase decimation

Consider the block diagram in Figure 5.11. where $h(n)$ is a generic lowpass FIR filter of length L.

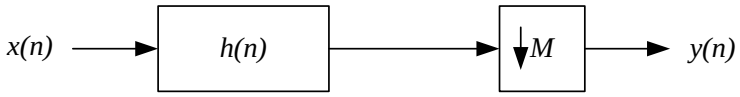


Fig. 5.11. Block diagram of a decimation system. Decimation factor M

The direct implementation structure of such FIR filter is show in Figure 5.12

This structure is described by the following equation

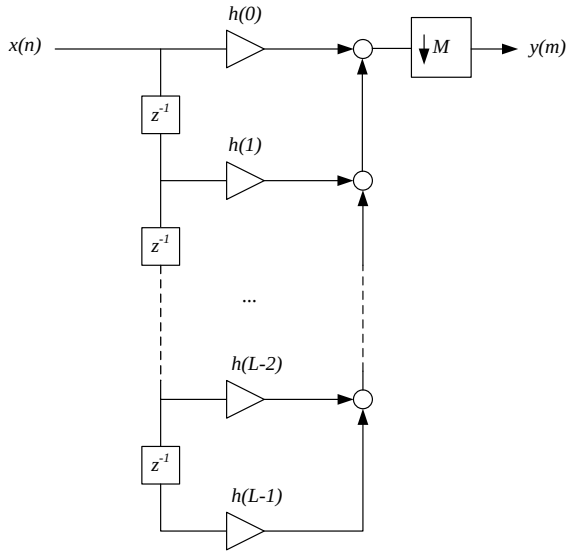


Fig. 5.12. Expanded structure of a decimation system.

$$y(m) = x(n) * h(n)|_{n=Mm} = \sum_{l=0}^{L-1} h(l)x(n-l)\,|_{n=Mm} \tag{5.16}$$

The computational efficiency of this solution is not satisfactory. In fact, the result of several operations is discarded when the output of filtering is decimated. For this reason it is more convenient to swap the order of the two blocks. This way we reduce the computational burden since everything that follows the decimator module works at a rate M times slower than before. This result is shown in Figure 5.13

At this point we can further elaborate this structure. Setting $N = 9$ and $M = 3$ we have the system depicted in Figure 5.14.

Merging together the outputs of the decimators in groups of size $M = 3$ we get the structure shown in Figure 5.15.

In the general case, consider a filter $h(n)$ of length $L = KM$, with K and M integers. We can write

$$\begin{aligned} H(z) &= \sum_{l=0}^{L-1} h(l)z^{-l} = \sum_{k=0}^{K-1}\sum_{m=0}^{M-1} h(kM+m)z^{-(kM+m)} = \\ &= \sum_{m=0}^{M-1} z^{-m} \sum_{k=0}^{K-1} h(kM+m)(z^M)^{-k} = \sum_{m=0}^{M-1} z^{-m} E_m(z^M) \end{aligned} \tag{5.17}$$

where we indicate

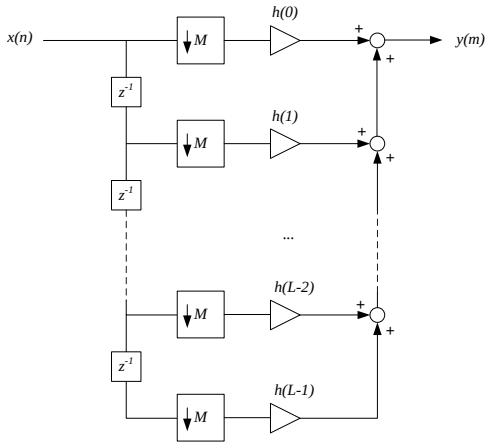


Fig. 5.13. Expanded structure of a decimation system. The decimation module is interleaved with filtering.

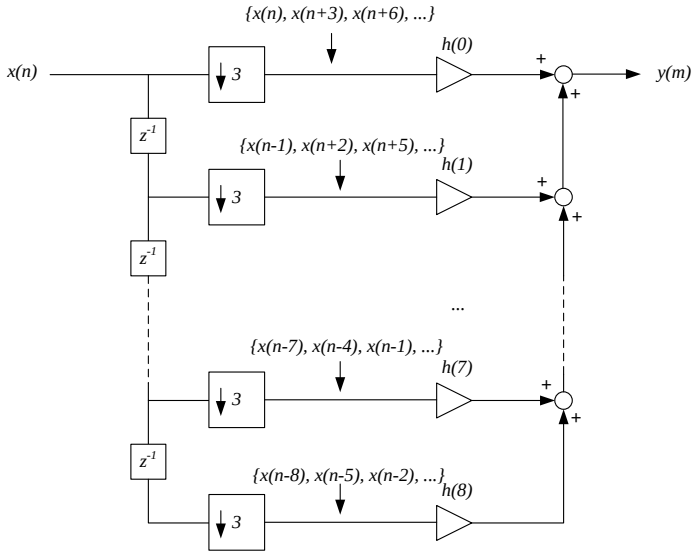


Fig. 5.14. Example of implementation structure of a FIR filter with $N = 9$ samples and decimation factor $M = 3$.

$$E_m(z) = \sum_{k=0}^{K-1} e_m(k) z^{-k} \quad e_m(k) = h(kM + m) \tag{5.18}$$

Note that the argument of E_m is z^M instead of z. This means that E_m works at a rate M times slower than the original filter. The impulse responses of the polyphase filters e_m are obtained by downsampling by a factor M the original impulse response $h(n)$ delayed by m samples.

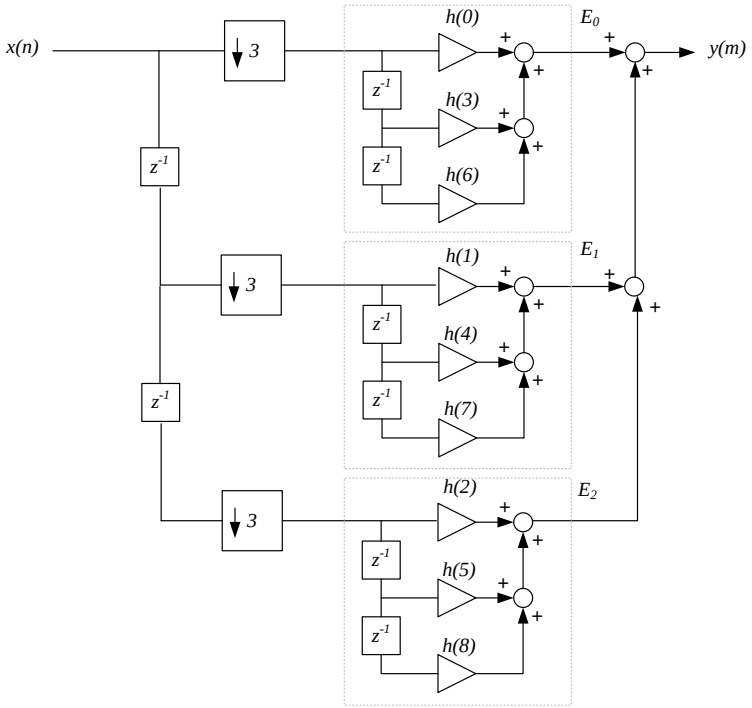


Fig. 5.15. Example of implementation structure of a FIR filter with $N = 9$ samples and decimation factor $M = 3$.

Figure 5.16 shows how to compute decimation by means of polyphase filters. First, the filter $H(z)$ in (a) is expanded in its polyphase representation in (b), according to equation (5.17). Then, for each sub-filter $E_m(z^M)$, multirate identity in Figure 5.10 are used to invert the order of filtering and downsampling (c).

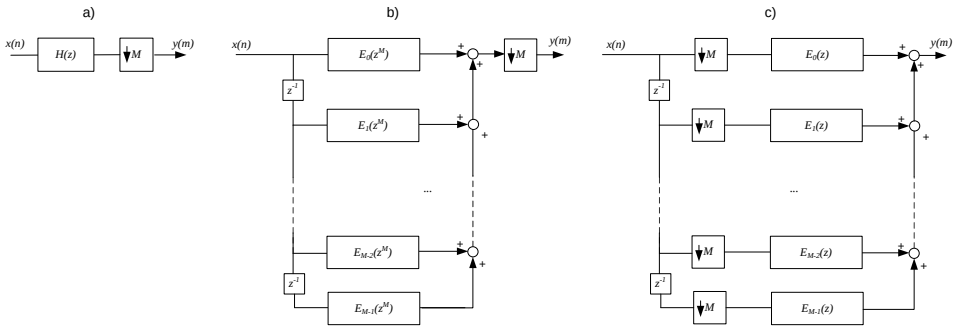


Fig. 5.16. Polyphase implementation of a decimation system

The input stage of the polyphase structure in Figure 5.16c can be interpreted as a sort of multiplexing that distributes samples to the M polyphase filters according to a round-robin policy (Figure 5.17).

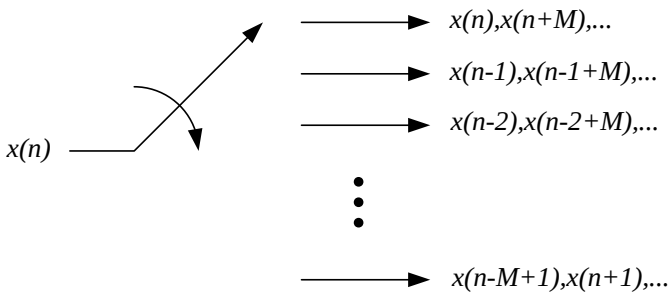


Fig. 5.17. Multiplexing operated by the input stage of a polyphase implementation of a decimation system (decimation factor M).

A decimation filter implemented without recurring to the polyphase structure requires N memory cells. By using the polyphase structure the number of cells is reduced to $M + \frac{N}{M}$.

Listing 5.5. Matlab

```
% Create input signal and filter
x=1:21;
h=[1 2 3 4 5 6 7 8 9 10 0 0]; % Pad zeros to make length equal to integer multiple of M

%%%%%% Direct Form (Inefficient) %%%%%%
y=filter(h,1,x); % Compute filter output
y_dec=y(1:4:end) % Throw away unneeded output samples

%%%%%% Polyphase Form (Efficient) %%%%%%
% Select polyphase filters
p0=h(1:4:end)
p1=h(2:4:end)
p2=h(3:4:end)
p3=h(4:4:end)

% Select polyphase signals
% Put a zero in front to provide the x[-3], x[-2], and x[-1] terms
x0=x(1:4:end)
x1=[0 x(4:4:end)]
x2=[0 x(3:4:end)]
x3=[0 x(2:4:end)]
% filter each polyphase component and add together
y_poly_dec=filter(p0,1,x0)+filter(p1,1,x1)+filter(p2,1,x2)+filter(p3,1,x3)
```

Polyphase interpolation

The same arguments apply in the case of interpolation. Figure 5.18 shows how the conventional interpolation scheme in (a) is converted in the polyphase implementation in (c) using equation (5.17) and the multirate identity in Figure 5.9.

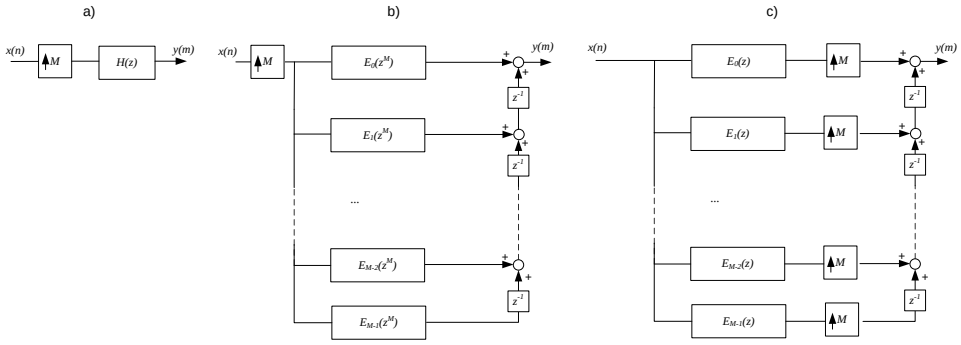


Fig. 5.18. Polyphase implementation of a interpolation system (interpolation factor M).

5.7 Polyphase filter banks

A filter bank is a set of parallel filters that enables to decompose the input signal into a number of subbands. Figure 5.19 show an example where the number of $M = 4$ subbands.

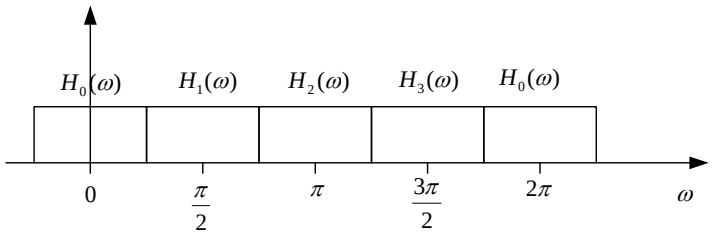


Fig. 5.19. Frequency response of a uniform filter bank with $M = 4$ subbands

Each of the $H_m(z)$ filters can be seen as obtained by translating the frequency response of $H_0(z)$ by $2\pi\frac{m}{M}$:

$$H_m(z) = H_0(ze^{+j\frac{2\pi m}{M}}) \tag{5.19}$$

We can write $H_0(z)$ as in (5.17),

$$H_0(z) = \sum_{k=0}^{M-1} z^{-k} E_k(z^M) \tag{5.20}$$

Therefore, the expression of $H_m(z)$ becomes

$$H_m(z) = H_0(ze^{+j\frac{2\pi m}{M}}) = \sum_{k=0}^{M-1} z^{-k} e^{-j\frac{2\pi m}{M}k} E_k(z^M e^{+j\frac{2\pi m}{M}M}) \tag{5.21}$$

Since $e^{+j\frac{2\pi m}{M}M} = 1$, we get:

$$H_m(z) = \sum_{k=0}^{M-1} z^{-k} e^{-j\frac{2\pi m}{M}k} E_k(z^M) \tag{5.22}$$

By setting $W = e^{-\frac{2\pi}{M}}$ we can further simplify the expression that becomes

$$H_m(z) = \sum_{k=0}^{M-1} z^{-k} W^{mk} E_k(z^M) \tag{5.23}$$

By grouping together the expressions of $H_m(z)$ for $m = 0, \ldots, M-1$ in matrix form we obtain:

$$\begin{bmatrix} H_0(z) \\ H_1(z) \\ H_2(z) \\ \cdots \\ H_{M-1}(z) \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & \cdots & W^0 \\ W^0 & W^1 & W^2 & \cdots & W^{M-1} \\ W^0 & W^2 & W^4 & \cdots & W^{2(M-1)} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ W^0 & W^{M-1} & W^{2(M-1)} & \cdots & W^{(M-1)^2} \end{bmatrix} \begin{bmatrix} E_0(z^M) \\ z^{-1}E_1(z^M) \\ z^{-2}E_2(z^M) \\ \cdots \\ z^{-(M-1)}E_{M-1}(z^M) \end{bmatrix} \tag{5.24}$$

The $M \times M$ matrix is the one that implements the DFT. Therefore, the implementation of the filter bank is shown in Figure 5.20.

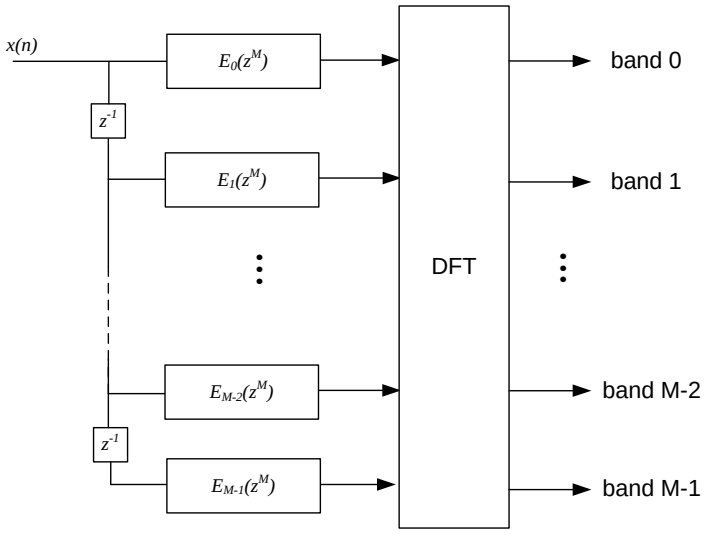


Fig. 5.20. Implementation of a polyphase filterbank with the DFT

This structure is computationally efficient because

- the input stage takes advantage of the polyphase filter structure

- the DFT is implemented using the FFT, thus reducing the complexity from $O(M^2)$ to $O(M \log M)$.

Listing 5.6. Matlab

```
x = randn(1000,1) + j*randn(1000,1); % input signal
out = zeros(M, length(x));           % output subbands

M = 4; % number of subbands
K = 10;
L = M*K; % filter taps


h =  fir1(L-1, 1/M); %design filter with window method

h_m = zeros(M,length(h));

% compute polyphase filters
for m = 1:M
    h_m(m,m:M:end) = h(m:M:end);
end

% compute filterbank output
for m = 1:M
    y = filter(h_m(m,:), 1, x);
    out(m, 1:length(y)) = y;
end
out = fft(out);

% show output spectrum
for m = 1:M
    subplot(M,1,m)
    Hs=spectrum.welch;
    psd(Hs,out(m,:))
end
```

5.8 Perfect reconstruction filter banks

By combining the functional modules described in the previous sections it is possible to obtain a system that has enormous importance in audio, image and video processing: subband coding.

Although it is possible to generalize this scheme to an arbitrary number of M subbands, here we focus our attention on the system depicted in Figure 5.21 where $M = 2$.

The input signal is decomposed into two subbands in the analysis phase.

- the low-pass subband $y_0(m)$ is a low-pass filtered ($h_0(n)$) and downsampled version of the input
- the high-pass subband $y_1(m)$ is a high-pass filtered and downsampled version of the input

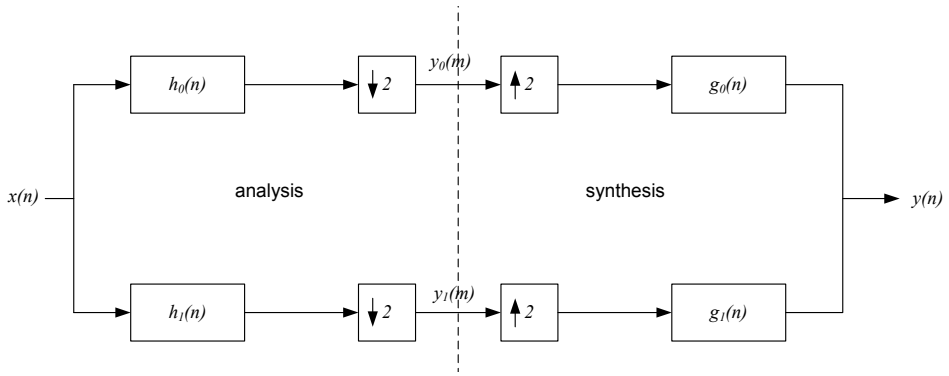


Fig. 5.21. Bank of two filters

The signal is reconstructed from the resulting subbands in the synthesis phase by upsampling and appropriate filtering (low-pass filter $g_0(n)$ and high-pass filter $g_1(n)$).

The goal of subband coding is to design the filters $h_0(n)$, $h_1(n)$, $g_0(n)$ and $g_1(n)$ in such a way that $y(n) = x(n)$, i.e. perfect reconstruction is achieved.

Before proceeding, we rewrite the z domain expression for downsampling and upsampling when $M = L = 2$:

- downsampling

$$x_{down}(n) = x(2n) \Longleftrightarrow X_{down}(z) = \frac{1}{2}[X(z^{1/2}) + X(-z^{1/2})] \tag{5.25}$$

- upsampling

$$x^{up}(n) = \begin{cases} x(n/2) & n = 0, 2, 4, \dots \\ 0 & otherwise \end{cases} \Longleftrightarrow X^{up}(z) = X(z^2) \tag{5.26}$$

If we consider the system in Figure 5.22 that depicts a cascade of downsampling and upsampling by a factor of 2 we can combine the previous expressions to obtain:

$$V(z) = \frac{1}{2}[X(z) + X(-z)] \tag{5.27}$$

The $X(-z)$ term in this equation is the Z-transform of an aliased or modulated version of the sequence $x(n)$:

$$X(-z) = \sum x(n)(-z)^{-n} = \sum x(n)(-1)^n z^{-n} \longleftrightarrow (-1)^n x(n) \tag{5.28}$$

In the frequency domain, this is equivalent to shifting the frequency axis by $\omega = \pi$.

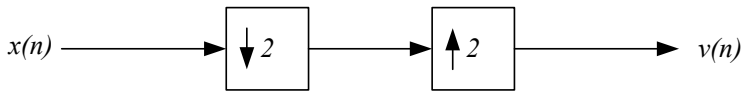


Fig. 5.22. Downsampling followed by upsampling $M = L = 2$

According to equation (5.27) we can write the output $y(n)$ of the analysis-synthesis system in the z domain as

$$Y(z) = \frac{1}{2}G_0(z)[H_0(z)X(z) + H_0(-z)X(-z)] + \frac{1}{2}G_1(z)[H_1(z)X(z) + H_1(-z)X(-z)] \tag{5.29}$$

The previous expression can be rewritten as:

$$Y(z) = \frac{1}{2}[H_0(z)G_0(z) + H_1(z)G_1(z)]X(z) + \frac{1}{2}[H_0(-z)G_0(z) + H_1(-z)G_1(z)]X(-z) \tag{5.30}$$

In order to obtain $Y(z) = X(z)$ the following constraints must be satisfied:

$$T(z) = H_0(z)G_0(z) + H_1(z)G_1(z) = 2 \tag{5.31}$$

$$A(z) = H_0(-z)G_0(z) + H_1(-z)G_1(z) = 0 \tag{5.32}$$

The first equation imposes perfect reconstruction, while the second enables to cancel alias. Note that alias cancelation can be achieved also with non-ideal filters.

The solution to this problem is not unique. In the following we define how to obtain one specific solution: QMF (Quadrature Mirror Filters).

Since $A(z) = 0$, we can write:

$$\frac{G_0(z)}{G_1(z)} = -\frac{H_1(-z)}{H_0(-z)} \tag{5.33}$$

Therefore:

$$G_0(z) = C(z)H_1(-z) \tag{5.34}$$

$$G_1(z) = -C(z)H_0(-z) \tag{5.35}$$

The simplest choice for $C(z)$ is $C(z) = 1$, thus:

$$G_0(z) = H_1(-z) \longleftrightarrow g_0(n) = (-1)^n h_1(n) \tag{5.36}$$

$$G_1(z) = -H_0(-z) \longleftrightarrow g_1(n) = (-1)^{n+1} h_0(n) \tag{5.37}$$

So far, we have expressed the synthesis filters impulse responses as a function of the analysis filters impulse responses.

Then, we exploit the constraint $T(z) = 2$. We can slightly relax this constraint by writing $T(z) = dz^l$, i.e. the output is a scaled and delayed copy of the input.

Let us impose that all the filters have finite impulse response (FIR) and linear phase, i.e. $h(n) = h(N - 1 - n)$.

Having fixed $H_0(z)$, let us set $H_1(z)$ in such a way that the modulus of its frequency response is a mirrored copy of the modulus of $H_0(z)$ with respect to $\omega = \pi/2$. This is equivalent to modulating the frequency response at $\omega = \pi$

$$H_1(e^{j\omega}) = H_0(e^{j(\omega-\pi)}) \tag{5.38}$$

In the z domain this is equivalent to:

$$H_1(z) = H_0(-z) \tag{5.39}$$

$$G_0(z) = H_1(-z) = H_0(z) \tag{5.40}$$

$$G_1(z) = -H_0(-z) = -H_1(z) \tag{5.41}$$

If we plug these terms into equation (5.31) we obtain:

$$T(z) = \frac{1}{2}[H_0^2(z) - H_1^2(z)] \tag{5.42}$$

Since $H_0(z)$ has linear phase:

$$H_0(e^{j\omega}) = e^{-j\omega(N-1)/2} \cdot |H_0(e^{j\omega})| \tag{5.43}$$

and

$$\begin{aligned} H_1(e^{j\omega}) &= H_0(e^{j(\omega-\pi)}) = \\ &= e^{-j(\omega-\pi)(N-1)/2} \cdot |H_0(e^{j(\omega-\pi)})| = \\ &= e^{-j\omega(N-1)/2} \cdot e^{-j\pi(N-1)/2} |H_0(e^{j(\omega-\pi)})| = \\ &= e^{-j\omega(N-1)/2} \cdot (-1)^{\frac{N-1}{2}} |H_0(e^{j(\omega-\pi)})| \end{aligned} \tag{5.44}$$

therefore:

$$T(e^{j\omega}) = \frac{1}{2} e^{-j\omega(N-1)} [|H_0(e^{j\omega})|^2 - (-1)^{N-1} |H_0(e^{j(\omega-\pi)})|^2] \tag{5.45}$$

If N is odd, $T(z) = 0$. We want N to be even. In this case we obtain:

$$T(e^{j\omega}) = \frac{1}{2} e^{-j\omega(N-1)} [|H_0(e^{j\omega})|^2 + |H_0(e^{j(\omega-\pi)})|^2] \tag{5.46}$$

which implies:

$$|H_0(e^{j\omega})|^2 + |H_0(e^{j(\omega-\pi)})|^2 = |H_0(e^{j\omega})|^2 + |H_1(e^{j\omega})|^2 = \text{const} \tag{5.47}$$

To summarize, the QMF conditions are:

- linear phase FIR filters (N even)
- $H_1(e^{j\omega}) = H_0(e^{j(\omega-\pi)})$
- $|H_0(e^{j\omega})|^2 + |H_0(e^{j(\omega-\pi)})|^2 = \text{const}$

Listing 5.7. Matlab

```
% generate filters based on QMF conditions
h0 = [sqrt(2)/2, sqrt(2)/2];

h1 = h0.*[1 -1]; g0 = h0; g1 = -h1;

% check QMF conditions in the frequency domain
[H0, W] = freqz(h0, 1, 1000); [H1, W] = freqz(h1, 1, 1000);

T = abs(H0).^2 + abs(H1).^2;

figure(1)
plot(W, abs(H0).^2)
hold on
plot(W, abs(H1).^2, 'r')
plot(W, T, 'g');
axis([0 pi 0 2.2]), xlabel('\omega'); ylabel('|H_i(\omega)|^2')
legend('|H_0(\omega)|^2', '|H_1(\omega)|^2', 'T(\omega)');
hold off

[x Fs Nbits] = wavread('../file sources/trumpet.wav'); x = x(:,1);

% analysis
% low pass subband
y0 = conv(x, h0); % filtering
y0 = y0(1:2:end); % downsampling

% high pass subband
y1 = conv(x, h1); % filtering
y1 = y1(1:2:end); % downsampling

%synthesis
% low pass subband
y0_up = upsample(y0, 2); y0_up = conv(y0_up, g0);

% high pass subband
y1_up = upsample(y1, 2); y1_up = conv(y1_up, g1);

y = y0_up + y1_up;

y = y(2:end-2);

e = x - y; display(['max error: ' num2str(max(abs(e)))]);
```

References

- Mitra, S, Digital Signal Processing, McGraw-Hill, 3rd edition, January 2005, ISBN 0073048372

Part II

Fundamentals of statistical signal processing

6

Introduction to discrete random processes

6.1 Random sequences

A discrete time random sequence - DTRS - (also called random process, stochastic process) is a sequence of random variables $x(n)$, where for a given n, $x(n)$ is a random variable described by a probability distribution function $F_{x(n)}(x, n)$ that, in general, depends on the data sequence number n and it is given by

$$F_{x(n)}(x, n) = Pr\{x(n) \leq x\} \tag{6.1}$$

where Pr means the probability that the event in braces occurs and x is a real number.

The random variable can assume

- a continuous range of values
- a discrete set of values

If the partial derivative of $F_{x(n)}(x, n)$ with respect to x exists for all x, we can define the probability density function (pdf)

$$f_{x(n)}(x, n) = \frac{\partial}{\partial x} F_{x(n)}(x, n) \tag{6.2}$$

On the other hand is $x(n)$ assumes only a countable set of values, it is described by a discrete pdf (also called pmf)

$$f_{x(n)}(x, n) = \sum_{k=-\infty}^{+\infty} p_{x(n)}(x_k, n)\delta(x - x_k) \tag{6.3}$$

Expectations

Let $g(x)$ be a function of a random variable x. The expected value of $g(x)$ is

$$E[g(x)] = \int_{-\infty}^{+\infty} g(x) f_x(x) dx \tag{6.4}$$

$E[g(x)]$ is also called the average value or the mean value of $g(x)$.

If $g(x) = x$

$$\eta_x = E[x] = \int_{-\infty}^{+\infty} x f_x(x) dx \tag{6.5}$$

The variance of x is defined as:

$$\sigma_x^2 = E[(x - \eta_x)^2] \tag{6.6}$$

Statistical independence and uncorrelation

Two random variables x and y are statistically independent if their joint pdfs factor into the product of the marginal pdfs

$$f_{xy}(x, y) = f_x(x) f_y(y) \tag{6.7}$$

They are uncorrelated if

$$E[xy] = E[x]E[y] \tag{6.8}$$

If x and y are independent they are uncorrelated. The converse is not in general true (it is true if they are jointly Gaussian).

Independent identically distributed (i.i.d.) random sequences

Consider the case when the p.d.f. does not depend on the time index n

$$f_{x(n)}(x, n) = f_x(x) \tag{6.9}$$

If, in addition, the random variables at different time instant n, m are independent the sequence $\{x(n)\}$ is said to be an independent identically distributed random sequence.

In other words, a i.i.d. sequence is characterized by a p.d.f. (or p.m.f) $f_x(x)$. The sequence is generated by taking independent samples of the p.d.f.

Bernoulli sequence

Consider the experiment of tossing a coin. The p.m.f. that describes the experiment is the Bernoulli p.m.f.

$$f_x(x) = \begin{cases} 0 \text{ with probability } p \\ 1 \text{ with probability } 1-p \end{cases} \tag{6.10}$$

By repeatedly tossing a coin we generate a Bernoulli i.i.d. sequence.

Listing 6.1. Matlab

```
% generate i.i.d. Bernoulli sequence of 1000 samples, p = 0.5
x = round(rand(1000,1));

% generate i.i.d. Bernoulli sequence of 1000 samples, arbitrary p
y = rand(1000,1);
x = zeros(length(y), 1);
p = 0.2;
x(y <  p) = 0;
x(y >= p) = 1;
```

Gaussian i.i.d.

We obtain a Gaussian i.i.d. sequence by sampling a Gaussian p.d.f. with mean η_x and variance σ_x^2.

$$f_x(x) = \frac{1}{\sqrt{2\pi}\sigma_x^2} e^{-\frac{(x-\eta_x)^2}{2\sigma_x^2}} \tag{6.11}$$

Listing 6.2. Matlab

```
% generate i.i.d. Gaussian sequence of 1000 samples, zero mean, variance = 1
x = randn(1000,1);

% generate i.i.d. Gaussian sequence of 1000 samples, mean = m, variance = v
m = 1;
v = 2;
x = sqrt(v)*randn(1000,1) + m;
```

6.2 Jointly distributed random sequences

Two discrete-time sequences $x(n)$ and $y(m)$ are described by a joint probability function $F_{x(n),y(m)}(x,n,y,m)$

$$F_{x(n),y(m)}(x,n,y,m) = Pr\{x(n) \leq x \text{ and } y(m) \leq y\} \tag{6.12}$$

If $x(n)$ and $y(m)$ assume a continuous range of values and $F_{x(n),y(m)}(x,n,y,m)$ is differentiable with respect to x and y the joint probability density function is defined as

$$f_{x(n),y(m)}(x,n,y,m) = \frac{\partial^2}{\partial x \partial y} F_{x(n),y(m)}(x,n,y,m) \tag{6.13}$$

Example

Let $\{x(n)\}$ and $\{z(n)\}$ two Gaussian i.i.d. sequences having zero mean and variances σ_x^2 and σ_n^2. Consider the random sequence $\{y(n)\}$, defined as

$$y(n) = x(n) + z(n) \tag{6.14}$$

Sequences $\{x(n)\}$ and $\{y(n)\}$ are statistically dependent. Therefore, the knowledge of $f_{x(n)}(x,n)$ and $f_{y(n)}(y,n)$ is not enough to describe the joint p.d.f. $f_{x(n),y(m)}(x,n,y,m)$. Since we are dealing with i.i.d. sequences we can drop the time index, i.e. $f_{x(n),y(m)}(x,n,y,m) = f_{x,y}(x,y) \neq f_x(x)f_y(y)$. Listing 6.3 shows how to generate and visualize samples of the joint p.d.f. $f_{x,y}(x,y)$.

Listing 6.3. Matlab

```
var_x = 1; var_z = 0.1;

x = sqrt(var_x)*randn(1000,1);
z = sqrt(var_z)*randn(1000,1);

y = x + n;

plot(x, y, 'o');
```

Stationary discrete time random sequences

A DTRS is stationary if its statistical characterization is not affected by a shift in the data sequence origin, i.e.

$$F_{x(n)}(x,n) = F_{x(n+k)}(x,n+k) = F_x(x) \quad \forall k,n \tag{6.15}$$

Correlation and covariance sequences

Given two DTRS $x(n)$ and $y(n)$ the cross-correlation is defined as

$$r_{xy}(i,j) = E[x(n-i)y^*(n-j)] \tag{6.16}$$

The auto-correlation function is defined as

$$r_x(i,j) = E[x(n-i)x^*(n-j)] \tag{6.17}$$

In the following we restrict our attention on wide-sense stationary (WSS) sequences:

- the mean value is independent of the data sequence index

$$E[x(n)] = \eta(n) = \eta \quad \forall n \tag{6.18}$$

- the correlation is function only of the difference in the time indices of the two random variables

$$r_{xy}(m) = E[x(n)y^*(n-m)] = E[x(n+m)y^*(n)] \tag{6.19}$$

$$r_x(m) = E[x(n)x^*(n-m)] = E[x(n+m)x^*(n)] \tag{6.20}$$

The autocovariance and the cross-covariance sequences of WSS DTRSs are defined respectively by,

$$c_x = E\{[x(n) - \eta_x][x(n-m) - \eta_x]^*\} = r_x(m) - |\eta_x|^2 \tag{6.21}$$

$$c_{xy} = E\{[x(n) - \eta_x][y(n-m) - \eta_y]^*\} = r_{xy}(m) - \eta_x\eta_y^* \tag{6.22}$$

Example

Consider a zero-mean i.i.d. sequence $\{z(n)\}$ and the sequence $\{x(n)\}$ generated by the following first-order difference equation

$$x(n) = \rho x(n-1) + z(n) \tag{6.23}$$

with $|\rho| < 1$. It is possible to show that the sequence $\{x(n)\}$ is WSS with:

$$\eta_x = 0 \tag{6.24}$$

$$\begin{aligned}\sigma_x^2 &= E[x(n)^2] = E[(\rho x(n-1) + z(n))^2] = \\ &= \rho^2 E[x(n-1)^2] + 2\rho E[x(n-1)z(n)] + E[z(n)^2] = \\ &= \rho^2\sigma_x^2 + \sigma_z^2 \Longrightarrow \sigma_x^2 = \frac{\sigma_z^2}{1-\rho^2}\end{aligned} \tag{6.25}$$

$$r_x(m) = \sigma_x^2\rho^{-|m|} \tag{6.26}$$

Listing 6.4. Matlab

```
rho = 0.99;
var_z = 0.1;

z = sqrt(var_z)*randn(1000,1);
x = filter(1, [1 -rho], z);

plot(x);
```

Time averages and ergodicity

Time averages are often used to infer statistical properties for DTRSs. Let $x(x)$ be a WSS DTRS. Then the $(2N+1)$-point time average for the $x(n)$ provides an estimate of the mean

$$< x(n) >_N = \frac{1}{2N+1} \sum_{n=-N}^{N} x(n) \tag{6.27}$$

If the estimate converges for $N \longrightarrow \infty$ we define

$$< x(n) >= \lim_{N \longrightarrow \infty} < x(n) >_N \tag{6.28}$$

as the average of the entire sequence.

By comparison, $E[x(n)]$ is an ensemble average, i.e. it is the mean of all possible values of $x(n)$ at a specific value of n.

Similarly, an estimate of the autocorrelation sequence is

$$< x(n)x^*(n-m) >_N = \frac{1}{2N+1} \sum_{n=-N}^{N} x(n)x^*(n-m) \tag{6.29}$$

and if the estimate converges

$$< x(n)x^*(n-m) >= \lim_{N \longrightarrow \infty} < x(n)x^*(n-m) >_N \tag{6.30}$$

If $x(n)$ is not only WSS but also satisfies the ergodic theorems for the mean and the autocorrelation, then

$$< x(n) >= E[x(n)] \tag{6.31}$$

$$< x(n)x^*(n-m) >= r_x(m) \tag{6.32}$$

Therefore, if the sequence is ergodic, time averages are equivalent to ensemble averages.

Example

Listing 6.5 shows $R = 1000$ realizations of N samples each of the random process defined above. Note that both time statistics and ensemble statistics converge to the same (true) value as R and N tend to ∞.

Listing 6.5. Matlab

```
rho = 0.9; var_z = 0.1;

N = 2000; % number of time samples
R = 2000;  % number of realizations

x = zeros(R,N); z = sqrt(var_z)*randn(R,N);

for r = 1:R
    x(r,:) = filter(1, [1 -rho], z(r,:));
end

% time statistics (_N) (assume only first realization available)
% ensemble statistics (_R) (compute only at time 50 because WSS)

m_N = mean(x(1,:))
m_R = mean(x(:,50))
m_T = 0

v_N = var(x(1,:))
v_R = var(x(:,50))
v_T = var_z / (1 - rho^2)
```

Stationary vs ergodic

Let a be a random number which is - say - normally distributed (the distribution is not critical). Define a trivial DTRS

$$x(n) = a \tag{6.33}$$

(where braces {} imply the ensemble of all possible outcomes).

One realization (outcome) of this process will simply be the constant horizontal time path

$$x_1(n) = a_1 \tag{6.34}$$

another outcome will be

$$x_2(n) = a_2 \tag{6.35}$$

different from the first.

Repeating this experiment many times will give a multitude of outcomes in the form of a sheaf of horizontal lines parallel to the time axis, approximating the infinite number of all possible outcomes corresponding to the ensemble (= the process itself)

$$x(n) = a \tag{6.36}$$

This process is stationary, since it is evident that mean value, variance, probability density function, autocorrelation function, in short all statistics, are invariant with time.

But we can't extract any info on statistics (for instance mean value) by processing data from a SINGLE realization, say

$$x_1(n) = a_1 \tag{6.37}$$

Therefore the process is not ergodic.

The most important ergodic process btw, is white noise. From one realization you may extract all relevant statistics.

7

Spectral estimation

7.1 Introduction to estimation theory

Goal of estimation theory: estimate the value of an unknown parameter from a set of observations of a random variable.

Example: given a set of observations from a Gaussian distribution, estimate the mean or variance from these observations.

Bias of an estimator

The difference between the expected value of the estimate and the actual value θ is called the **bias** and will be denoted by B.

$$B = \theta - E\{\hat{\theta}_N\} \tag{7.1}$$

If the bias is zero, then the expected value of the estimate is equal to the true value, that is

$$E\{\hat{\theta}_N\} = \theta \tag{7.2}$$

and the estimate is said to be **unbiased**.

If $B \neq 0$ then $\hat{\theta}$ is said to be **biased**.

More often an estimate is biased, but $B \to 0$ when $N \to \infty$

$$\lim_{N\to\infty} E\{\hat{\theta}_N\} = \theta \tag{7.3}$$

then the estimate is said to be **asymptotically unbiased**.

Variance of an estimator

It is desirable that an estimator be either unbiased or asymptotically unbiased. Nevertheless, this is not enough to have a good estimator.

For an estimate to be meaningful, it is necessary that

$$\text{Var} \to 0 \quad \text{as} \quad N \to \infty \tag{7.4}$$

or, in other words

$$\lim_{N\to\infty} \text{var}\{\hat{\theta}_N\} = \lim_{N\to\infty} \{|\hat{\theta}_N - E\{\hat{\theta}_N\}|^2\} = 0 \tag{7.5}$$

If $\hat{\theta}_N$ is unbiased, $E\{\hat{\theta}_N\} = \theta$, it follows from the Tchebycheff inequality that, for any $\epsilon > 0$

$$Pr\{|\hat{\theta}_N - \theta| \geq \epsilon\} \leq \frac{\text{var}\{\hat{\theta}_N\}}{\epsilon^2} \tag{7.6}$$

$\Rightarrow$ if var $\to 0$ as $N \to \infty$, then the probability that $\hat{\theta}_N$ differs by more than ϵ from the true value will go to zero.

In this case, $\hat{\theta}_N$ is said to converge to θ with probability one.

Another for of convergence, **stronger** than the convergence with probability one, is the **mean square convergence**.

An estimate $\hat{\theta}_N$ is said to converge to θ in mean-square sense, if

$$\lim_{N\to\infty} E\{|\hat{\theta}_N - \theta|^2\} = 0 \tag{7.7}$$

- For an unbiased estimator this is equivalent to the previous condition that the variance of the estimate goes to zero
- An estimate is said to be consistent if it converges, in some sense, to the true value of the parameter
- We say that the estimator is consistent if it is asymptotically unbiased and has variance that goes to zero as $N \to \infty$

Example: sample mean

What can we say about the sample mean as an estimator?

Let $\{x(n)\}$ be an i.i.d. sequence.

Sample mean:

$$\mu_x = \frac{1}{N}\sum_{n=1}^{N} x(n) \tag{7.8}$$

Expected value of the sample mean (bias)

$$E\{\mu_x\} = \frac{1}{N}\sum_{n=1}^{N} E\{x(n)\} = \mu_x \tag{7.9}$$

Variance of the sample mean

$$\mathrm{var}\{\mu_x\} = \frac{1}{N^2}\sum_{n=1}^{N} \mathrm{var}\{x\} = \frac{\sigma_x^2}{N} \tag{7.10}$$

The sample mean is an unbiased estimator, and it is consistent.

Estimation problem formulation

We seek to determine from a set of data, a set of parameters such that their values would yield the highest probability of obtaining the observed data.

- The unknown parameters may be seen as deterministic or random variables
- There are essentially two alternatives to the statistical case
  - no a priori distribution assumed: Maximum Likelihood
  - a priori distribution known: Bayes
- Key problem: to estimate a group of parameters from a discrete-time signal or dataset

Given an N-point dataset $(x_0, x_1, \ldots, x_{N-1})$ which depends on an unknown parameter θ, define an estimator as some function, g, of the dataset, i.e.

$$\hat{\theta} = g(x(0), x(1), \ldots, x(N-1)) \tag{7.11}$$

which may be used to estimate θ. This is the problem of parameter estimation.

Maximum likelihood estimation

Let $\mathbf{x} = (x_0, x_1, \ldots, x_{N-1})$ be a random sample from the parametric p.d.f. $f(x;\theta)$, where θ is a parameter vector. Let $f(\mathbf{x}|\theta)$ denote the p.d.f. that specifies the probability of observing the data vector $\mathbf{x}$ given the parameter vector θ. The parameter $\theta = (\theta_0, \theta_1, \ldots, \theta_{M-1})$ is a vector defined on a multi-dimensional space.

If individual observations, x_i are statistically independent of one another, than according to the theory of probability, the p.d.f. for the data $\mathbf{x} = (x_0, x_1, \ldots, x_{N-1})$ given the parameter vector θ can be expressed as a multiplication of p.d.f.'s for individual observations,

$$f(\mathbf{x} = (x_0, x_1, \ldots, x_{N-1})|\theta) = f_1(x_1|\theta) f_2(x_2|\theta) \cdots f_{N-1}(x_{N-1}|\theta) \tag{7.12}$$

Given a set of parameter values, the corresponding p.d.f. will show that some data are more probable than other data. In reality, however, we have already observed the data. Accordingly we are faced with an inverse problem: given the observed data and a model of interest, find the one p.d.f., among all the probability densities that the model prescribes, that is most likely to have produced the data. To solve this inverse problem, we define the likelihood function by reversing the roles of the data vector x and the parameter vector θ in $f(x|\theta)$, i.e.

$$L(\theta|\mathbf{x}) = f(\mathbf{x}|\theta) \tag{7.13}$$

Thus $L(\theta|\mathbf{x})$ represents the likelihood of the parameter θ given the observed data $\mathbf{x}$, and as such is a function of θ.

Consider the problem of estimating the mean $\theta = \eta_x = 1$ of a Gaussian p.d.f. with known variance $\sigma_x^2 = 2$ (see Figure 7.1a)

$$f(x;\theta) = \frac{1}{\sqrt{2\pi\sigma_x^2}} e^{-\frac{(x-\theta)^2}{2\sigma_x^2}} \tag{7.14}$$

We can write the likelihood function as

$$\begin{aligned} L(\theta|x) &= f(x = (x_0, x_1, \ldots, x_{N-1})) = f_1(x_1|\theta) f_2(x_2|\theta) \cdots f_{N-1}(x_{N-1}|\theta) = \\ &= \left[\frac{1}{\sqrt{2\pi\sigma_x^2}} e^{-\frac{(x_0-\theta)^2}{2\sigma_x^2}}\right] \cdot \left[\frac{1}{\sqrt{2\pi\sigma_x^2}} e^{-\frac{(x_1-\theta)^2}{2\sigma_x^2}}\right] \cdots \left[\frac{1}{\sqrt{2\pi\sigma_x^2}} e^{-\frac{(x_{N-1}-\theta)^2}{2\sigma_x^2}}\right] \\ &= \left(\frac{1}{\sqrt{2\pi\sigma_x^2}}\right)^N e^{-\frac{1}{2\sigma_x^2}\sum_{n=1}^{N}(x_n-\theta)^2} \end{aligned} \tag{7.15}$$

The shape of the likelihood function is shown in Figure 7.1b. There is an important difference between the p.d.f. $f(x;\theta)$ and the likelihood function $L(\theta|\mathbf{x})$. As illustrated in

Figure 7.1, the two functions are defined on different axes, and therefore are not directly comparable to each other. Specifically, the p.d.f. in Figure 7.1a is a function of the data given a particular set of parameter values, defined on a data scale. On the other hand, the likelihood function is a function of the parameter given a particular set of observed data, defined on the parameter scale. In short, 7.1a tells us the probability of a particular data value for a fixed parameter, whereas Figure 7.1b tells us the likelihood ("unnormalized probability") of a particular parameter value for a fixed data set. Note that the likelihood function in Figure 7.1b is a curve because there is only one parameter which is assumed to be unknown. If the model has two parameters, the likelihood function will be a surface sitting above the parameter space. In general, for a model with k parameters, the likelihood function $L(\theta|\mathbf{x})$ takes the shape of a $k-dim$ geometrical surface.

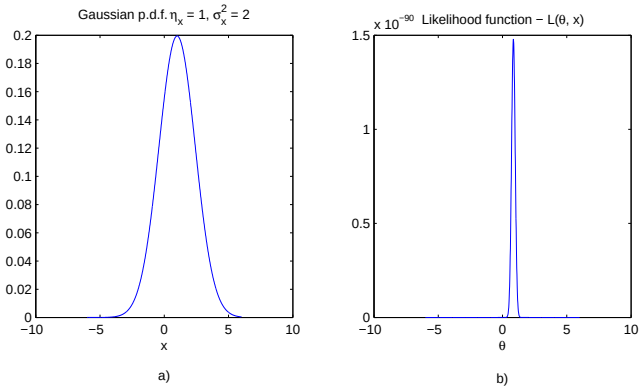


Fig. 7.1. a) Gaussian p.d.f. $\eta_x = 1$, $\sigma_x^2 = 2$, b) Likelihood function $L(\theta|x)$ for a data vector of size $N = 10$

Listing 7.1. Matlab

```
N = 10;      %dataset length
var_x = 2;  %known variance
theta = 1;  %true mean (to be estimated)
x = sqrt(var_x)*randn(N, 1) + theta; %dataset

%pdf
s = [-6:0.01:6]';
pdf = (1/(sqrt(2*pi)*var_x))*exp(-(s - theta).^2/(2*var_x));

%likelihood
theta_s = [-6:0.01:6]';
L = ones(length(theta_s), 1);
for n=1:N
    L = L .* (1/(sqrt(2*pi)*var_x)).*exp((-1/(2*var_x))*(x(n) - theta_s).^2);
end

subplot(1,2,1)
plot(s, pdf);
xlabel({'x';'';'a)'}); title('Gaussian p.d.f. \eta_x = 1, \sigma_x^2 = 2');
subplot(1,2,2)
plot(theta_s, L);
xlabel({'\theta';'';'b)'}); title('Likelihood function - L(\theta, x)');
```

The principle of maximum likelihood estimation (MLE) states that the desired probability distribution is the one that makes the observed data most likely, which means that one must seek the value of the parameter vector that maximizes the likelihood function $L(\theta|\mathbf{x})$. The resulting parameter vector, which is sought by searching the multi-dimensional parameter space, is called the MLE estimate.

$$\theta_{ML} = \arg\max_{\theta} L(\theta|\mathbf{x}) \tag{7.16}$$

Getting back to our example, the MLE of the mean can be obtained by taking the derivative of $L(\theta|\mathbf{x})$ with respect to θ and setting the result to zero. For the problem at hand, it is more convenient to define the log-likelihood function $\log L(\theta|\mathbf{x})$ and compute

$$\frac{\partial \log L(\theta|\mathbf{x})}{\partial \theta} = \frac{\partial}{\partial \theta}\left[N \log(\sqrt{2\pi}\sigma_x^2) - \frac{1}{2\sigma_x^2}\sum_{n=1}^{N}(x_n - \theta)^2\right] = \frac{2}{2\sigma_x^2}\sum_{n=1}^{N}(x_n - \theta) \tag{7.17}$$

$$\frac{\partial \log L(\theta|\mathbf{x})}{\partial \theta} = 0 \Rightarrow \theta_{ML} = \frac{1}{N}\sum_{n=1}^{N} x_n \tag{7.18}$$

Therefore, for the case of Gaussian p.d.f. with known variance, the sample mean is the MLE of the true mean. Note that $\frac{\partial \log L(\theta|\mathbf{x})}{\partial \theta} = 0$ is only a necessary condition for the existence of a MLE estimate. An additional condition must also be satisfied to ensure that $\log L(\theta|\mathbf{x})$ is a maximum and not a minimum. This can be checked by calculating the second derivatives of the log-likelihoods and showing whether they are all negative at $\theta = \theta_{ML}$. For the example above

$$\frac{\partial^2 \log L(\theta|\mathbf{x})}{\partial \theta^2} = -N < 0 \tag{7.19}$$

In practice, however, it is usually not possible to obtain an analytic form solution for the MLE estimate, especially when the model involves several parameters and its PDF is highly non-linear. In such situations, the MLE estimate must be sought numerically using nonlinear optimization algorithms.

Bayesian estimation

The idea of Bayesian estimation is to maximize the a-posteriori probability (MAP) of the unknown parameter(s) θ given the data x, i.e.

$$\theta_{MAP} = \arg\max_{\theta} f(\theta|\mathbf{x}) \tag{7.20}$$

Using the Bayes rule, we can write

$$\theta_{MAP} = \arg\max_{\theta} f(\theta|\mathbf{x}) = \arg\max_{\theta} \frac{f(\mathbf{x}|\theta)p(\theta)}{p(\mathbf{x})} = \arg\max_{\theta} f(\mathbf{x}|\theta)p(\theta) \tag{7.21}$$

where the last equality follows from the fact that $p(\mathbf{x})$ is independent from θ. Unlike the MLE, the MAP estimate (or Bayesian estimate) also depends on some prior knowledge about the parameter θ, expressed by the p.d.f. $p(\theta)$. When no prior knowledge is available, thus $p(\theta)$ is a uniform p.d.f., the MAP estimate is equivalent to the MLE estimate.

7.2 Basic concepts

Goal: from a finite record of a stationary data sequence, estimate how the total power is distributed over frequency

Applications

- Speech processing and audio devices
- Source localization using sensor arrays
- Medical diagnosis
- Control system design
- Hidden periodicity finding
- Radar, sonar

Energy spectral density

Let $\{y(t); t = 0, \pm 1, \pm 2, \ldots\}$ denote a deterministic discrete-time data sequence.

Assume that $\{y(t)\}$ has finite energy, which means that

$$\sum_{t=-\infty}^{+\infty} |y(t)|^2 < \infty \tag{7.22}$$

Then the sequence $\{y(t)\}$ possesses a discrete-time Fourier transform (DTFT) defined as

$$Y(\omega) = \sum_{t=-\infty}^{+\infty} y(t)e^{-j\omega t} \tag{7.23}$$

In the rest of this chapter we will use the symbol $Y(\omega)$ in lieu of the more cumbersome $Y(e^{j\omega})$, to denote the DTFT

Recall the Parseval's equality

$$\sum_{t=-\infty}^{+\infty} |y(t)|^2 = \frac{1}{2\pi} \int_{-\pi}^{+\pi} S(\omega) d\omega \tag{7.24}$$

where

$$S(\omega) \triangleq |Y(\omega)|^2 = \text{Energy Spectral Density (ESD)} \tag{7.25}$$

Define

$$\rho(k) = \sum_{t=-\infty}^{+\infty} y(t) y^*(t-k) \tag{7.26}$$

It is readily verified that

$$\begin{aligned} \sum_{k=-\infty}^{+\infty} \rho(k) e^{-j\omega k} &= \sum_{k=-\infty}^{+\infty} \sum_{t=-\infty}^{+\infty} y(t) y^*(t-k) e^{-j\omega t} e^{j\omega(t-k)} \\ &= \left[\sum_{t=-\infty}^{+\infty} y(t) e^{-j\omega t} \right] \left[\sum_{s=-\infty}^{+\infty} y(s) e^{-j\omega s} \right]^* \\ &= S(\omega) \end{aligned} \tag{7.27}$$

which shows that $S(\omega)$ can be obtained as the DTFT of the autocorrelation of the finite-energy sequence $\{y(t)\}$.

7.3 Power spectral density

The above definitions can be extended in a rather straightforward manner to the case of random signals. In general, a random signal has infinite energy

$$\sum_{t=-\infty}^{+\infty} |y(t)|^2 = \infty \tag{7.28}$$

but finite average power

$$E\{|y(t)|^2\} < \infty \tag{7.29}$$

where $E\{\cdot\}$ denotes the expectation over the ensemble of realizations. For simplicity reasons, in what follows we will use the name **power spectral density** (PSD) for that quantity.

If we assume that $E\{y(t)\} = 0$, the autocovariance sequence (ACS) of $\{y(t)\}$ is defined as

$$r(k) = E\{y(t)y^*(t-k)\} \tag{7.30}$$

Since we consider only second order stationary sequences (WSS), the ACS depends only on the lag between two samples averaged.

Recall the properties of the ACS:

- $r(k) = r^*(-k)$
- $r(0) \geq |r(k)| \quad \forall k$

Let us define the covariance matrix of $\{y(t)\}$ as

$$R_m = \begin{pmatrix} r(0) & r^*(1) & \cdots & r^*(m-1) \\ r(1) & r(0) & \ddots & \vdots \\ \vdots & \vdots & \vdots & r^*(1) \\ r(m-1) & \cdots & r(1) & r(0) \end{pmatrix} \tag{7.31}$$

The covarince matrix R_m is positive semidefinite,

$$a^* R_m a \geq 0 \quad \forall \text{ vector } a \tag{7.32}$$

Power spectral density: first definition

The PSD is defined as the DTFT of the ACS

$$\phi(\omega) = \sum_{k=-\infty}^{+\infty} r(k) e^{-j\omega k} \tag{7.33}$$

Note that

$$r(k) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} \phi(\omega) e^{j\omega k} d\omega \quad \text{inverse DTFT} \tag{7.34}$$

Interpretation:

- $r(0) = E\{|y(t)|^2\} = \frac{1}{2\pi} \int_{-\infty}^{+\infty} \phi(\omega) d\omega$
- $\phi(\omega)d\omega$ = infinitesimal signal power in the band $\omega \pm \frac{d\omega}{2}$

Power spectral density: second definition

$$\phi(\omega) = \lim_{N\to\infty} E\left\{ \frac{1}{N} \left| \sum_{n=1}^{N} y(t) e^{-j\omega t} \right|^2 \right\} \tag{7.35}$$

Note that $\phi(\omega) = \lim_{N\to\infty} E\left\{ \frac{1}{N} |Y_N(\omega)|^2 \right\}$, where

$$Y_N(\omega) = \sum_{t=1}^{N} y(t) e^{-j\omega t} \tag{7.36}$$

is the finite DTFT of $\{y(t)\}$.

Properties

- $\phi(\omega) = \phi(\omega + 2\pi) \quad \forall\omega$, thus we can restrict our attention to $\omega \in [-\pi, +\pi] \Leftrightarrow f \in [-1/2, +1/2]$
- $\phi(\omega) \geq 0$
- If $y(t)$ is real,
  - then $\phi(\omega) = \phi(-\omega)$,
  - otherwise $\phi(\omega) \neq \phi(-\omega)$

Transfer of PSD through linear systems

Consider the following LTI causal system described by its z transform

$$H(z) = \sum_{k=0}^{\infty} h_k z^{-k} \tag{7.37}$$

The transfer function of the system is (with a slight abuse of notation)

$$H(\omega) = H(z)|_{z=e^{j\omega}} = \sum_{k=0}^{\infty} h_k e^{-j\omega k} \tag{7.38}$$

The input $\{e(t)\}$ and the output $\{y(t)\}$ sequences are related via the convolution sum (see Figure 7.2)

$$y(t) = \sum_{k=-\infty}^{+\infty} h_k e(t-k) \tag{7.39}$$

From equation (7.37) we obtain:

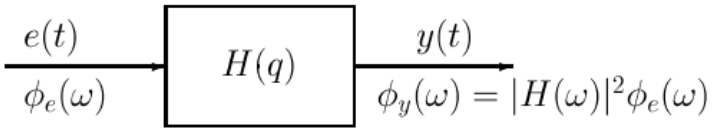


Fig. 7.2. Relationship between the PSDs of the input and output of a linear system

$$\begin{aligned} r_y(k) &= \sum_{p=-\infty}^{+\infty} \sum_{m=-\infty}^{+\infty} h_p h_m^* E\{e(t-p)e^*(t-m-k)\} \\ &= \sum_{p=-\infty}^{+\infty} \sum_{m=-\infty}^{+\infty} h_p h_m^* r_e(m+k-p) \end{aligned} \tag{7.40}$$

Inserting (7.40) in (7.33) gives

$$\begin{aligned} \phi_y(\omega) &= \sum_{k=-\infty}^{+\infty} \sum_{p=-\infty}^{+\infty} \sum_{m=-\infty}^{+\infty} h_p h_m^* r_e(m+k-p) e^{-j\omega(k+m-p)} e^{j\omega m} e^{-j\omega p} \\ &= \left[\sum_{k=-\infty}^{+\infty} h_p e^{-j\omega p}\right] \left[\sum_{m=-\infty}^{+\infty} h_m^* e^{j\omega m}\right] \left[\sum_{\tau=-\infty}^{+\infty} r_e(\tau) e^{-j\omega\tau}\right] \\ &= |H(\omega)|^2 \phi_e(\omega) \end{aligned} \tag{7.41}$$

7.4 Spectral estimation problem

The spectral estimation problem can be stated formally as follows: from a finite-length record $\{y(1), y(2), \ldots, y(N)\}$ of a second order stationary random process, determine an estimate $\hat{\phi}(\omega)$ of its power spectral density $\phi(\omega)$, for $\omega \in [-\pi, +\pi]$.

Remarks:

- It is desirable to obtain an estimate $\hat{\phi}(\omega)$ as close to $\phi(\omega)$ as possible.
- The main limitation on the quality of most PSD estimates is due to the quite small number of data samples N usually available
- Most commonly, N is limited by the fact that the signal under study can be considered wide sense stationary only over short observation intervals

There are two main approaches

- **Nonparametric**
  - based on the definitions in equation (7.33) and equation (7.35)
  - no assumptions on the functional form of $\phi(\omega)$
- **Parametric**
  - Assumes a parametrized functional form of the PSD
  - Can be used only when there is enough information about the studied signal

7.5 Nonparametric spectral estimation

Periodogram method

The periodogram method relies on the definition of the PSD

$$\phi(\omega) = \lim_{N\to\infty} E\left\{ \frac{1}{N} \left| \sum_{n=1}^{N} y(t) e^{-j\omega t} \right|^2 \right\} \tag{7.42}$$

Neglecting the expectation and the limit operation that cannot be performed with a finite number of samples N , we get

$$\hat{\phi}_p(\omega) = \frac{1}{N} \left| \sum_{n=1}^{N} y(t) e^{-j\omega t} \right|^2 \tag{7.43}$$

- Natural estimator
- Used by Schuster (1900) to determine "hidden periodicities" (hence the name)

Listing 7.2. Matlab

```
[x, Fs, nbits] = wavread('../file sources/flute.wav');

t_start = 1; %sec
D = 0.050;   %observation time = 50msec
n = t_start*Fs:1:(t_start + D)*Fs; %in samples

y = x(n); N = length(y);
M = N; %number of frequency bins

phi_p = (1/N)*abs(fft(y,M)).^2;
w = 2*pi*[0:M-1]/M;
plot(w, phi_p);
axis([0 2*pi 0 100])
```

Correlogram

The ACS based definition (7.33) of the PSD leads to the correlogram spectral estimator

$$\hat{\phi}_c(\omega) = \sum_{k=-(N-1)}^{+(N-1)} \hat{r}(k) e^{-j\omega k} \tag{7.44}$$

where $\hat{r}(k)$ denotes the estimate of the covariance lag $r(k)$, obtained from the available sample $\{y(1), y(2), \ldots, y(N)\}$.

There are two standard way to obtain an estimate $\hat{r}(k)$

- Standard unbiased estimate

$$\hat{r}(k) = \frac{1}{N-k} \sum_{t=k+1}^{N} y(t)y^*(t-k), \quad k \geq 0 \tag{7.45}$$

- Standard biased estimate

$$\hat{r}(k) = \frac{1}{N} \sum_{t=k+1}^{N} y(t)y^*(t-k), \quad k \geq 0 \tag{7.46}$$

For both estimators

$$\hat{r}(k) = \hat{r}^*(-k), \quad k < 0 \tag{7.47}$$

The biased estimate is usually preferred, for the following reasons

- the ACS sequence decays rather rapidly so that $r(k)$ is quite small for large lags k. Comparing the definition we see that (7.46) will be small for large k, whereas (7.45) may take erratic values for large k, as it is obtained averaging only a few products.
- the ACS sequence defined by (7.46) is guaranteed to be positive semidefinite, while this is not the case for (7.45). If the ACS is not positive semidefinite, the PSD estimator in (7.44) may lead to negative spectral estimates
- if the biased ACS estimator is used in $\hat{\phi}(\omega)$, then

$$\begin{aligned} \hat{\phi}_p(\omega) &= \frac{1}{N} \left| \sum_{n=1}^{N} y(t)e^{-j\omega t} \right|^2 \\ &= \sum_{k=-(N-1)}^{+(N-1)} \hat{r}(k)e^{-j\omega k} \\ \hat{\phi}_c(\omega) \end{aligned} \tag{7.48}$$

Listing 7.3. Matlab

```
[x, Fs, nbits] = wavread('../file sources/flute.wav');

t_start = 1; %sec
D = 0.050;   %observation time = 50msec
n = t_start*Fs:1:(t_start + D)*Fs; %in samples

y = x(n); N = length(y);

M = 2*N - 1;

r = xcorr(y, 'biased');
r = circshift(r, N);

phi_r = fft(r,M);
w = 2*pi*[0:M-1]/M;
plot(w, phi_r);
```

Statistical performance of $\hat{\phi}_p(\omega)$ and $\hat{\phi}_c(\omega)$

Summary:

- Both are asymptotically (for large N) unbiased:

$$E\{\hat{\phi}_p(\omega)\} \to \phi(\omega) \quad \text{as} \quad N \to \infty \tag{7.49}$$

- Both have large variance, even for large N

Thus, $\hat{\phi}_p(\omega)$ and $\hat{\phi}_c(\omega)$ have **poor performance**.

Intuitive explanation:

- $\hat{r}(k) - r(k)$ may be large for large $|k|$
- even if the errors $\{\hat{r}(k) - r(k)\}_{|k|=0}^{N-1}$ are small, there are "so many" that when summed in $\hat{\phi}_p(\omega) - \phi(\omega)$, the PSD error is large

Bias analysis

$$\begin{aligned} E\{\hat{\phi}_p(\omega)\} = E\{\hat{\phi}_c(\omega)\} &= \sum_{k=-(N-1)}^{(N-1)} E\{\hat{r}(k)\} e^{-j\omega k} \\ &= \sum_{k=-(N-1)}^{(N-1)} \left(1 - \frac{|k|}{N}\right) r(k) e^{-j\omega k} \\ &= \sum_{k=-\infty}^{+\infty} w_B(k) r(k) e^{-j\omega k} \end{aligned} \tag{7.50}$$

$$w_B(k) \begin{cases} \left(1 - \frac{|k|}{N}\right), & |k| \leq N-1 \\ 0, & |k| \geq N \end{cases} \tag{7.51}$$

The above sequence is called the triangular window, or the Bartlett window.

The DTFT of the product of two sequences is equal to the convolution of their respective DTFT's. Thus,

$$E\{\hat{\phi}_p(\omega)\} = \frac{1}{2\pi} \int_{-\pi}^{+\pi} \phi(\zeta) W_B(\omega - \zeta) d\zeta \tag{7.52}$$

Where $W_B(\omega)$ is the DTFT of the triangular window, and it is equal to

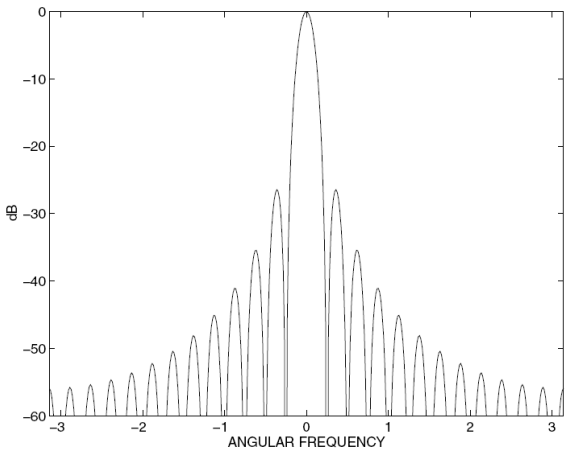


Fig. 7.3. $W_B(\omega)/W_B(0)$ for $N = 25$

$$W_B(\omega) = \frac{1}{N}\left[\frac{\sin(\omega N/2)}{\sin(\omega/2)}\right]^2 \tag{7.53}$$

An illustration of $W_B(\omega)$ is depicted in Figure 7.3 for $N = 25$.

Ideally, to have zero bias, we want $W_B(\omega)$ = Dirac impulse $\delta(\omega)$. The main lobe width decreases as $1/N$. For small values of N, $W_B(\omega)$ may differ quite a bit from $\delta(\omega)$.

If the ACS estimator in (7.45) is used, the corresponding window function that would appear is the DTFT of the rectangular window

$$w_R(k) = \begin{cases} 1 & |k| \le N-1 \\ 0, & |k| \ge N \end{cases} \tag{7.54}$$

A straightforward calculation gives

$$W_R(\omega) = \frac{\sin[(N-1/2)\omega]}{\sin(\omega/2)} \tag{7.55}$$

Note that, unlike $W_B(\omega)$, $W_R(\omega)$ can assume negative values for some values of ω, thus providing estimate of the PSD that can be negative for some frequencies.

Equation (7.52) suggests that bias manifests itself in two different ways

- Main lobe width causes **smearing** (or smooting): details in $\phi(\omega)$ separated in f by less than $1/N$ are not resolvable. Thus: periodogram resolution limit $= 1/N$
- Sidelobe level causes **leakage**:

Summary of periodogram bias properties:

- For small N, severe bias
- As $N \to \infty$, $W_B(\omega) \to \delta(\omega)$, so $\hat{\phi}(\omega)$ is asymptotically unbiased.

Variance analysis

As $N \to \infty$:

$$E\{[\hat{\phi}_p(\omega_1) - \phi_p(\omega_1)][\hat{\phi}_p(\omega_2) - \phi_p(\omega_2)]\} = \begin{cases} \phi^2(\omega_1) & \omega_1 = \omega_2 \\ 0, & \omega_1 \neq \omega_2 \end{cases} \tag{7.56}$$

- inconsistent estimate
- erratic behaviour

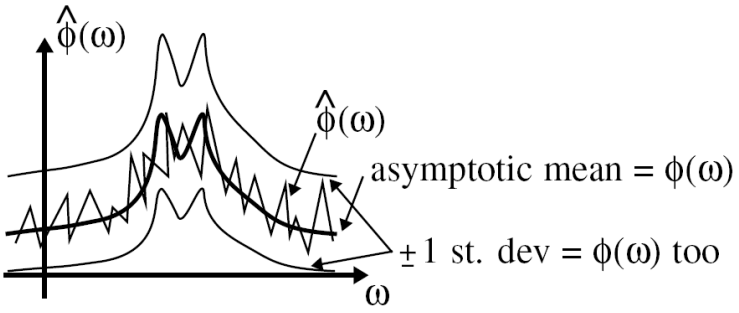


Fig. 7.4. The periodogram gives an inconsistent estimate of the PSD

Listing 7.4 and Figure 7.5 demonstrate the effect of bias and variance for the case of periodogram-based and Blackman-Tukey spectral estimation. Notice that in the latter case variance is significantly reduced at a cost of increased bias.

Listing 7.4. Matlab

```
clear
N = 2^14;
Nrealizations = 1000;
Ntot = N*Nrealizations;
M = floor(N/8);
LINEAR = 0; %linear or log y-axis

% Filter transfer function H(z) = 1/A(z)
rho_1 = 0.99;
theta_1 = 0.2*pi;
rho_2 = 0.95;
theta_2 = 0.3*pi;
a = conv([1 -2*cos(theta_1)*rho_1 rho_1^2], [1 -2*cos(theta_2)*rho_2 rho_2^2]);

Nfft = N/2;
[H, w] = freqz(1, a, Nfft);


w_p = 2*pi*[0:N/2-1]/N;
w_BT = 2*pi*[0:M - 1]/(2*M - 1);

phi_p  = zeros(Nrealizations, N);
phi_BT = zeros(Nrealizations, 2*M - 1);

for i = 1:Nrealizations

    e = randn(N,1);
    y = filter(1, a, e);
```

```
    % periodogram
    phi_p(i,:) = (1/N)*abs(fft(y,N)).^2;

    % Blackman-Tukey
    r = xcorr(y, 'biased');
    r = r(N - M + 1:N + M - 1);
    r = r.*bartlett(2*M - 1);
    % set r = r.*ones(2*M - 1, 1); %and M = N for the correlogram
    r = circshift(r, -M+1);
    phi_BT(i,:) = real(fft(r,2*M - 1));

    % plot individual realizations
%     if LINEAR == 1
%         plot(w_p, phi_p(i,1:N/2),'g', 'LineWidth', 1)
%         hold on
%         plot(w_BT, phi_BT(i,1:M),'b', 'LineWidth', 1)
%         plot(w, (abs(H).^2), 'r', 'LineWidth', 2);
%         ylim([0, 10^6]);
%         hold off
%         pause;
%     elseif LINEAR == 0
%         plot(w_p, 10*log10(phi_p(i,1:N/2)),'g', 'LineWidth', 1)
%         hold on
%         plot(w_BT, 10*log10(phi_BT(i,1:M)),'b', 'LineWidth', 1)
%         plot(w, 10*log10((abs(H).^2)), 'r', 'LineWidth', 2);
%         ylim([-80, 60]);
%         hold off
%         pause;
%     end


end

phi_p_mean = mean(phi_p)';
phi_p_var = var(phi_p)';

phi_BT_mean = mean(phi_BT)';
phi_BT_var = var(phi_BT)';

% plot expected values across realizations: E[phi_p] and E[phi_BT]
if LINEAR == 0
    plot(w_p, 10*log10(phi_p_mean(1:N/2)),'g', 'LineWidth', 2)
    hold on
    plot(w_p, 10*log10(phi_p_mean(1:N/2) + sqrt(phi_p_var(1:N/2))),'g--')
    plot(w_p, 10*log10( max(phi_p_mean(1:N/2) - sqrt(phi_p_var(1:N/2)), 10e-3) ),'g--')

    plot(w_BT, 10*log10(phi_BT_mean(1:M)),'b', 'LineWidth', 2)
    plot(w_BT, 10*log10(phi_BT_mean(1:M) + sqrt(phi_BT_var(1:M))),'b--')
    plot(w_BT, 10*log10( max(phi_BT_mean(1:M) - sqrt(phi_BT_var(1:M)), 10e-3) ),'b--')
    plot(w, 10*log10(abs(H).^2), 'r', 'LineWidth', 2);

    hold off
elseif LINEAR == 1
    plot(w_p, phi_p_mean(1:N/2),'g', 'LineWidth', 2)
    hold on
    plot(w_p, phi_p_mean(1:N/2) + sqrt(phi_p_var(1:N/2)),'g--')
    plot(w_p, max(phi_p_mean(1:N/2) - sqrt(phi_p_var(1:N/2)),10e-3),'g--')

    plot(w_BT, phi_BT_mean(1:M),'b', 'LineWidth', 2)
    plot(w_BT, phi_BT_mean(1:M) + sqrt(phi_BT_var(1:M)),'b--')
    plot(w_BT, max(phi_BT_mean(1:M) - sqrt(phi_BT_var(1:M)), 10e-3),'b--')
    plot(w, (abs(H).^2), 'r', 'LineWidth', 2);

    hold off
end

legend('E[\phi_{P}(\omega)]', 'E[\phi_{P}(\omega)] + \sigma', 'E[\phi_{P}(\omega)] - \sigma', ...
    'E[\phi_{BT}(\omega)]', 'E[\phi_{BT}(\omega)] + \sigma', 'E[\phi_{BT}(\omega)] - \sigma', ...
    '\phi(\omega)');
```

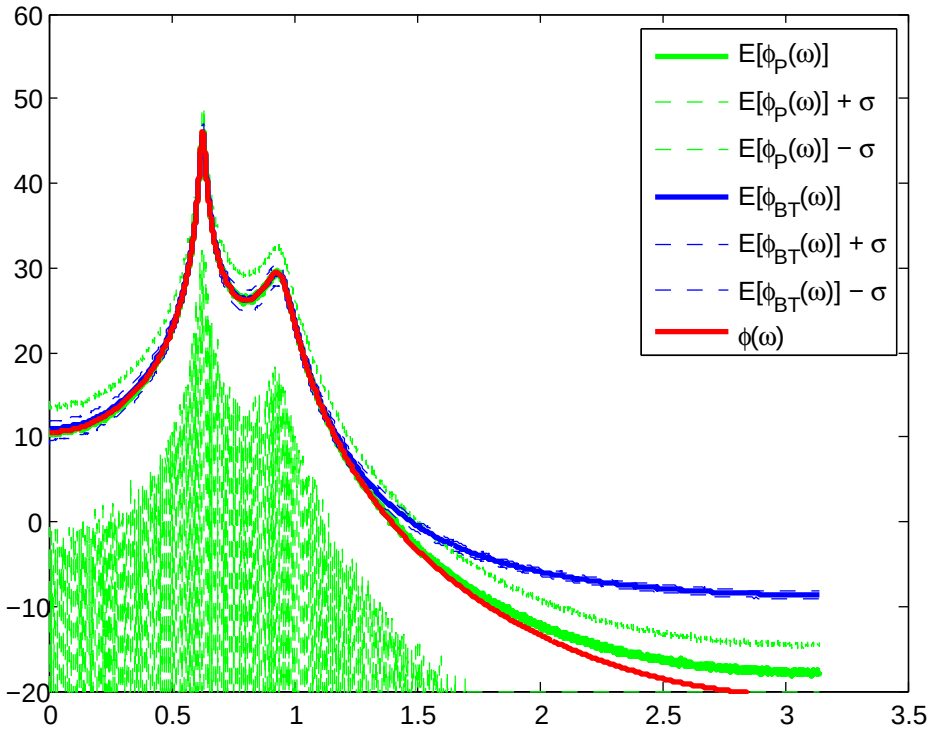


Fig. 7.5. Effect of bias and variance for the case of periodogram-based and Blackman-Tukey spectral estimation.

The Blackman-Tukey method

Basic idea: weighted correlogram, with small weight applied to covariances $\hat{r}(k)$ with large k

$$\hat{\phi}_{BT}(\omega) = \sum_{k=-(M-1)}^{M-1} w(k)\hat{r}(k)e^{-j\omega k} \tag{7.57}$$

where $\{w(k)\}$ is the lag window.

Let $W(\omega)$ (also called spectral window) denote the DTFT of $w(k)$. Making use of the DTFT property, we can write

$$\hat{\phi}_{BT}(\omega) = \frac{1}{2\pi}\int_{-\pi}^{+\pi} \hat{\phi}_p(\omega)W(\omega-\zeta)d\zeta \tag{7.58}$$

Conclusion: $\hat{\phi}_{BT}(\omega)$ is a locally smoothed periodogram.

- Variance decreases substantially (of the order of M/N)
- Bias increases slightly (of the order $1/M$)

The choice of the window is constrained by the fact that we want $\hat{\phi}_{BT}(\omega) \geq 0$. If the lag window $\{w(k)\}$ is positive semidefinite (i.e. $W(\omega) \geq 0$), then the windowed ACS $\{w(k)\hat{r}(k)\}$ is positive semidefinite too; which implies that $\hat{\phi}_{BT}(\omega) \geq 0$ for all ω.

Listing 7.5. Matlab

```
Fs = 1;      %sampling frequency
T = 1/Fs;    %sampling period
f1 = 1/8;  %normalized frequency
f2 = 1/16;  %normalized frequency
N = 1000;    %number of samples
n = [0:N-1]';

y = sin(2*pi*f1*n*T) + sin(2*pi*f2*n*T) + 2*randn(N,1);;

M = floor(N/10);

r = xcorr(y, 'biased');
r = r(N - M + 1:N + M - 1);
r = r.*bartlett(2*M - 1);
r = circshift(r, M);

phi_BT = fft(r,M);
w = 2*pi*[0:M-1]/M;
plot(w, phi_BT);
max_val = 1.1*max(phi_BT);
axis([0 2*pi 0 max_val]);
```

Time-bandwidth product

Define

$$N_e = \frac{\sum_{k=-(M-1)}^{M-1} w(k)}{w(0)} = \text{equiv. time width} \tag{7.59}$$

$$\beta_e = \frac{\frac{1}{2\pi}\int_{-\pi}^{+\pi} W(\omega)d\omega}{W(0)} = \text{equiv. bandwith} \tag{7.60}$$

It is possible to prove that

$$N_e\beta_e = 1 \tag{7.61}$$

This means that the more slowly the window decays to zero in one domain, the more concentrated it is in the other domain.

The equivalent temporal width, N_e is determined by the window length ($N_e \simeq 2M$ for rectangular window, $N_e \simeq M$ for triangular window). Since $N_e\beta_e = 1$ also the bandwidth β_e is determined by the window length, more precisely $\beta_e = O(1/M)$.

- As M increases, bias decreases and variance increases $\Rightarrow$ choose M as a tradeoff between variance and bias. As a rule of thumb, we should choose $M \leq N/10$ in order to reduce the standard deviation of the estimated spectrum at least three times, compared with the periodogram.
- Choose window shape to compromise between smearing (main lobe width) and leakage (sidelobe level). The energy in the main lobe and in the sidelobes cannot be reduced simultaneously, once M is given.

The Bartlett method

Basic idea: split up the available sample of N observations into $L = N/M$ subsamples of M observations each, then average the periodograms obtained from the subsamples for each value of ω.

Mathematically:

$$y_i(t) = y((i-1)M + t) \quad t = 1, \ldots, M \quad \text{the } i\text{th subsequence} \tag{7.62}$$

$$i = 1, \ldots, L \triangleq [N/M] \tag{7.63}$$

$$\hat{\phi}_i(\omega) = \frac{1}{M} \left| \sum_{t=1}^{M} y_i(t) e^{-j\omega t} \right|^2 \tag{7.64}$$

$$\hat{\phi}_B(\omega) = \frac{1}{L} \sum_{i=1}^{L} \hat{\phi}_i(\omega) \tag{7.65}$$

Remarks

- $\hat{\phi}_B(\omega) \simeq \hat{\phi}_{BT}(\omega)$ with a rectangular lag window $w_R(k)$. Since the Bartlett method implicitly uses $w_R(k)$, it has
  - high resolution (little smearing)
  - large leakage and relatively large variance

Listing 7.6. Matlab

```
Fs = 1;       %sampling frequency
T = 1/Fs;     %sampling period
f1 = 1/8;     %normalized frequency
f2 = 1/16;    %normalized frequency
L = 10;       %number of observations
M = 100;      %number of samples / observation
N = L*M;      %total number of samples
n = [0:N-1]';

y = sin(2*pi*f1*n*T) + sin(2*pi*f2*n*T) + 2*randn(N,1);;

phi_l = zeros(M,1); for l = 0:L-1

    y_l = y(1 + l*M:M + l*M);

    phi_l = phi_l + (1/M)*abs(fft(y_l)).^2;

end
phi_B = phi_l / L;

w = 2*pi*[0:M-1]/M;
plot(w, phi_B);
max_val = 1.1*max(phi_B);
axis([0 2*pi 0 max_val])
```

Welch method

Similar to Bartlett method, but

- allow overlap of subsequences (gives more subsequences, thus better averaging)
- use data window for each periodogram; gives mainlobe-sidelobe tradeoff capability

Let

$$y_i(t) = y((i-1)K + t) \quad t = 1, \ldots, M \quad i = 1, \ldots, S \tag{7.66}$$

denote the ith data segment.

- if $K = M$, no overlap as in Bartlett method
- if $K = M/2$, 50% overlap, $S \simeq 2M/N$ data segments

$$\hat{\phi}_i(\omega) = \frac{1}{MP} \left| \sum_{t=1}^{M} v(t) y_i(t) e^{-j\omega t} \right|^2 \tag{7.67}$$

where P denotes the power of the temporal window $\{v(t)\}$

$$P = \frac{1}{M} \sum_{t=1}^{M} |v(t)|^2 \tag{7.68}$$

The Welch estimate is determined by averaging the windowed periodograms

$$\hat{\phi}_W(\omega) = \frac{1}{S} \sum_{i=1}^{S} \hat{\phi}_i(\omega) \tag{7.69}$$

The Welch method is approximately equal to Blackman-Tuckey with a non-rectangular lag window.

Listing 7.7. Matlab

```
Fs = 1;      %sampling frequency
T = 1/Fs;    %sampling period
f1 = 1/8;    %normalized frequency
f2 = 1/16;   %normalized frequency
M = 200;     %number of samples / observation
K = M/4;     %new samples / observation
N = 1000;    %number of samples
S = N/K - (M - K)/K;  %number of observations
n = [0:N-1]';

y = sin(2*pi*f1*n*T) + sin(2*pi*f2*n*T) + 2*randn(N,1);

phi_s = zeros(M,1);
v = hamming(M);
P = (1/M)*sum(v.^2);
for s = 0:S-1

    y_s = y(1 + s*K:M + s*K);
```

```
    phi_s = phi_s + (1/(M*P))*abs(fft(v.*y_s)).^2;

end
phi_W = phi_s / S;

w = 2*pi*[0:M-1]/M;
plot(w, phi_W);
max_val = 1.1*max(phi_W);
axis([0 2*pi 0 max_val])
```

Daniell

It can be proved that, for $N >> 1$, $\{\hat{\phi}(\omega_i)\}$ are nearly uncorrelated random variables for

$$\left\{\omega_i = \frac{2\pi}{N} i\right\}_{j=0}^{N-1} \tag{7.70}$$

The basic idea of the Daniel method is to perform local averaging of $2J+1$ samples in the frequency domain to reduce the variance by about $2J+1$

$$\hat{\phi}_D(\omega) = \frac{1}{2J+1} \sum_{i=k-J}^{k+J} \hat{\phi}_p(\omega_i) \tag{7.71}$$

As J increases:

- bias increases (more smoothing)
- variance decreases (more averaging)

The Daniell method is approximately equal to Blackman-Tuckey with a rectangular spectral window.

7.6 Parametric spectral estimation

The parametric of model-based methods of spectral estimation assume that the signal satisfies a generating model with known functional form, and then proceed in estimating the parameters in the assumed model (see Figure 7.6).

Typically, the functional form of the PSD is a rational function in $e^{-j\omega}$.

$$\phi(\omega) = \frac{\sum_{|k|\leq m} \gamma_k e^{-j\omega k}}{\sum_{|k|\leq n} \rho_k e^{-j\omega k}} \tag{7.72}$$

where $\gamma_{-k} = \gamma_k^*$ and $\rho_{-k} = \rho_k^*$

$\phi(\omega)$ can approximate arbitrarily well any continuous PSD, provided m and n are chosen sufficiently large.

Note, however

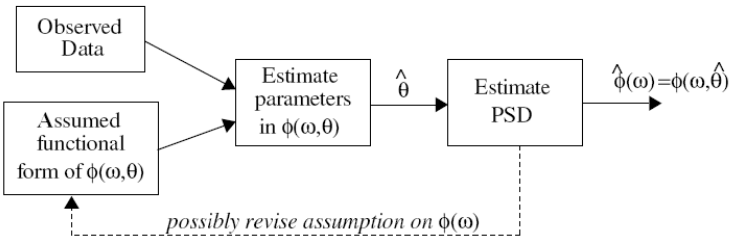


Fig. 7.6. Parametric spectral estimation

- choice of m and n is not simple
- some PSDs are not continuous

By Spectral Factorization theorem, a rational $\phi(\omega)$ can be factored as

$$\phi(\omega) = \left|\frac{B(\omega)}{A(\omega)}\right|^2 \sigma^2 \tag{7.73}$$

$$A(z) = 1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_n z^{-n} \tag{7.74}$$

$$B(z) = 1 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_m z^{-m} \tag{7.75}$$

and $A(\omega) = A(z)|_{z=e^{j\omega}}$.

The arbitrary rational PSD in (7.73) can be associated with a signal obtained by filtering white noise $\{e(n)\}$ of power σ^2 through the rational filter with transfer function $H(\omega) = B(\omega)/A(\omega)$. In the z-domain we can write

$$Y(z) = \frac{B(z)}{A(z)} E(z) \tag{7.76}$$

or, alternatively

$$A(z)Y(z) = B(z)E(z) \tag{7.77}$$

Depending on the values assumed by m and n we can have the following cases

- if $m = 0$ and $n \neq 0$, autoregressive model (AR), $A(z)Y(z) = E(z)$
- if $m \neq 0$ and $n = 0$, moving average model (MA), $Y(z) = B(z)E(z)$
- if $m \neq 0$ and $n \neq 0$, ARMA model (autoregressive, moving average), $A(z)Y(z) = B(z)E(z)$

By assumption, $\phi(\omega)$ is finite for all ω values. Therefore $A(z)$ cannot have any root exactly on the unit circle. Furthermore, since poles and zeros of $\phi(\omega)$ are in reciprocal pairs, it is always possible to choose $A(z)$ to have all its roots strictly inside the unit circle.

Covariance structure of ARMA processes

We derive an expression for the covariances of an ARMA process in terms of the parameters $\{a_i\}_{i=1}^n$, $\{b_i\}_{i=1}^m$ and σ^2.

Equation (7.77) can be rewritten as

$$y(t) + \sum_{i=1}^{n} a_i y(t-i) = \sum_{j=0}^{m} b_j e(t-j) \tag{7.78}$$

Multiplying by $y^*(t-k)$ and taking the expectation gives

$$\begin{aligned} r(k) + \sum_{i=1}^{n} a_i r(k-i) = \sum_{j=0}^{m} b_j E\{e(t-j)y^*(t-k)\} &= \sigma^2 \sum_{j=0}^{m} b_j h_{j-k}^* \\ &= 0 \quad \text{for} \quad k > m \end{aligned} \tag{7.79}$$

In fact

$$H(z) = \frac{B(z)}{A(z)} = \sum_{k=0}^{\infty} h_k z^{-k} \quad (h_0 = 1) \tag{7.80}$$

which gives

$$y(t) = \sum_{k=0}^{\infty} h_k e(t-k) \tag{7.81}$$

Then the second term becomes

$$E\{e(t-j)y^*(t-k)\} = E\left\{ e(t-j) \sum_{s=0}^{\infty} h_s^* e^*(t-k-s) \right\} = \sigma^2 \sum_{s=0}^{\infty} h_s^* \delta_{j,k+s} = \sigma^2 h_{j-k}^* \tag{7.82}$$

AR

In the ARMA class, the autoregressive or all-pole signals constitute the type that is most frequently used in applications. The AR equation may model spectra with narrow peaks by placing zeros of the A-polynomial close to the unit circle.

For AR signals, $m = 0$ and $B(z) = 1$. Thus, equation (7.79) holds for $k > 0$. Also, for $k = 0$, we get

$$r(0) + \sum_{i=1}^{n} a_i r(-i) = \sigma^2 \sum_{j=0}^{m} b_j h_j^* = \sigma^2 \tag{7.83}$$

Combining this equation with the following set of equations for $k > 0$

$$r(k) + \sum_{i=1}^{n} a_i r(k-i) = 0 \quad k > 0 \tag{7.84}$$

we form a system of linear equations

$$\begin{bmatrix} r(0) & r(-1) & \cdots & r(-n) \\ r(1) & r(0) & \cdots & \vdots \\ \vdots & \vdots & \ddots & r(-1) \\ r(n) & \cdots & \cdots & r(0) \end{bmatrix} \begin{bmatrix} 1 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} \sigma^2 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{7.85}$$

The above equations are called the Yule-Walker equations. If the $\{r(k)\}_{k=0}^{n}$ were known, we could solve for

$$\theta = [a_1, a_2, \ldots, a_n] \tag{7.86}$$

by using all but the first row.

$$\begin{bmatrix} r(1) \\ \vdots \\ r(n) \end{bmatrix} + \begin{bmatrix} r(0) & \cdots & r(-n+1) \\ \vdots & \ddots & \vdots \\ r(n-1) & \cdots & r(0) \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \tag{7.87}$$

or, with obvious definition

$$r_n + R_n \theta = 0 \tag{7.88}$$

The solution is $\theta = -R_n^{-1} r_n$. Once θ is found, σ^2 can be obtained from (7.83).

If the true ACS is unknown, we can replace $r(k)$ with $\hat{r}(k)$ and solve for $\{\hat{a}_i\}$ and $\hat{\sigma}^2$. Then, the PSD estimate is

$$\hat{\phi}(\omega) = \frac{\hat{\sigma^2}}{|\hat{A}(\omega)|^2} \tag{7.89}$$

Listing 7.8. Matlab

```
Nfft = 1024; N = 1000; sigma_e = 10; n = 3;

a = poly([0.99 0.99*exp(j*pi/4) 0.99*exp(-j*pi/4)]);
b = 1;

[H, w] = freqz(b, a, Nfft);

%generate signal according to parametric model
z = sigma_e*randn(N,1);
y = filter(b,a,z);

%estimate AR coefficients
r = xcorr(y,'biased');

Rx = toeplitz(r(N:N+n-1), r(N:-1:N-n+1));
rz = r(N+1:N+n);
theta =-Rx^(-1)*rz;
var_z = r(N) +sum(theta.*r(N-1:-1:N-n));

%visualize estimated and true power specrum
plot(w, 10*log10(sigma_e^2*abs(H).^2))
```

```
hold on
[He, w] = freqz(1,[1; theta], Nfft);
plot(w, 10*log10(var_z*abs(He).^2),'r')
legend('true','estimated');
hold off
```

MA

There are two main ways to estimate $\phi(\omega)$ when $n = 0$.

1. Estimate $\{b_k\}$ and σ^2 and insert them in

$$\phi(\omega) = |B(\omega)|^2 \sigma^2 \tag{7.90}$$

   - It is a nonlinear estimation problem
   - $\hat{\phi}(\omega)$ is guaranteed to be ≥ 0

2. Insert sample covariances $\{\hat{r}(k)\}$ in

$$\phi(\omega) = \sum_{k=-m}^{m} r(k) e^{-j\omega k} \tag{7.91}$$

   - this is $\hat{\phi}_{BT}(\omega)$ with a rectangular window of length $2m + 1$
   - $\hat{\phi}(\omega)$ is not guaranteed to be ≥ 0

References

- Stoica P., Moses R.L., Introduction to spectral analysis, Prentice Hall, 1st edition, January 1997, ISBN 0132584190

8

Linear prediction

8.1 Basic principles of LPC

Motivation and overview

One of the most powerful analysis techniques for voice and audio signals is the method of Linear Predictive Coding (LPC).

Basic idea: A sample of a discrete-time signal can be approximated (predicted) as a linear combination of past samples.

Motivation: Why Linear Predictive coding?

- LPC provides a parsimonious source-filter model for the human voice and other signals
- LPC is good for low-bitrate coding of speech, as in Codebook Excited LP (CELP)
- LPC provides a spectral envelope in the form of an all-pole digital filter
- LPC spectral envelopes are well suited for audio work such as when estimating vocal formants
- The LPC voice model has a loose physical interpretation
- LPC is analytically tractable: mathematically precise, simple and easy to implement
- Variations on LPC show up in other kinds of audio signal analysis

Introduction

A signal sample $s(n)$ at time n can be approximated by a linear combination of its own past p samples:

$$s(n) \approx a_1 s(n-1) + a_2 s(n-2) + \cdots + a_p s(n-p) = \sum_{k=1}^{p} a_k s(n-k) \tag{8.1}$$

where the coefficients a_k are assumed to be constant over the duration of the analysis window. If we assume that the signal can be modeled as an autoregressive (AR) stochastic process, then $s(n)$ can be expressed as

$$s(n) = \sum_{k=1}^{p} a_k s(n-k) + Gu(n) \tag{8.2}$$

where G is the gain parameter and $u(n)$ is a white noise excitation signal.

We'll see that $u(n)$ can also be taken to be a single impulse (e.g. a glottal pitch pulse for a voice model). When $u(n)$ is not white noise we may simply call $s(n)$ as an all-pole filtered version of the excitation signal $u(n)$. Calling $s(n)$ an AR process implies it is a stochastic process and thus is an all-pole filter driven by white noise.

Example: voice production can be modeled as above with $u(n)$ being the source excitation at the glottis and $s(n)$ being the output voice signal.

Z-transforming the previous equation gives:

$$S(z) = \sum_{k=1}^{p} a_k z^{-k} S(z) + GU(z) \tag{8.3}$$

leading to the transfer function $H(z)$:

$$H(z) = \frac{S(z)}{GU(z)} = \frac{1}{1 - \sum_{k=1}^{p} a_k z^{-k}} = \frac{1}{A(z)} \tag{8.4}$$

where $A(z) = 1 - \sum_{k=1}^{p} a_k z^{-k}$.

The source-filter interpretation of the above equation is provided in Figure 8.1 below which shows the excitation source $u(n)$ being scaled by the gain G and fed to the all-pole system $H(z) = 1/A(z)$ to produce the voice signal $s(n)$.

Now let's look at LPC from the viewpoint of estimating a signal sample based on its past. We consider the linear combination of past samples as the linearly predicted estimate $\hat{s}(n)$, defined by

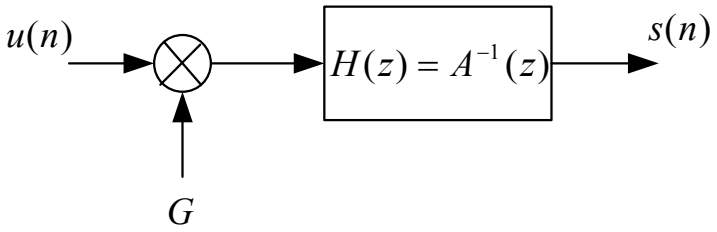


Fig. 8.1. Model of linear prediction

$$\hat{s}(n) = \sum_{k=1}^{p} a_k s(n-k) \tag{8.5}$$

The prediction error $e(n)$ is then defined as

$$e(n) = s(n) - \hat{s}(n) = s(n) - \sum_{k=1}^{p} a_k s(n-k) \tag{8.6}$$

Z-transforming the above equation, we obtain $E(z)$:

$$E(z) = (1 - \sum_{k=1}^{p} a_k z^{-k}) S(z) = A(z)S(z) = \frac{S(z)}{H(z)} \tag{8.7}$$

We can see that we can obtain $E(z)$ by applying the filter $A(z)$ to the signal $s(n)$. $A(z)$ is called inverse filter, since $A(z) = H^{-1}(z)$, where $H(z)$ is the forward filter used in the source-filter model $S(z) = H(z)E(z)$.

From Figure 8.1 we can deduce that the prediction error $e(n)$ equals $Gu(n)$, the scaled white noise process.

Goal of linear prediction

Goal: Find the set of predictor coefficients $\{a_k\}$ that minimizes the mean-squared prediction error over a short segment of the signal.

To formulate the problem let s define the short-time data and error signals at time n

$$s_n(m) = s(n+m), \quad e_n(m) = e(n+m), \quad m = 0, 1, 2, \ldots, M-1 \tag{8.8}$$

where $s_n(m)$ denotes the m-th sample of the length M signal segment starting at time n.

We seek to minimize the energy of the short-time error signal at time n

$$\min_{a_i} E_n \tag{8.9}$$

where

$$E_n = \sum_m e_n^2(m) = \sum_m \left[s_n(m) - \sum_{k=1}^{p} a_k s_n(m-k) \right]^2 \tag{8.10}$$

To solve equation (8.10) for the predictor coefficients a_i, we differentiate E_n with respect to each a_i and set the result to zero:

$$\frac{\partial E_n}{\partial a_i} = 0, \quad i = 0, 1, \ldots, p \tag{8.11}$$

$$\begin{aligned}
\frac{\partial E_n}{\partial a_i} &= 2 \sum_m \left[s_n(m) - \sum_{k=1}^{p} a_k s_n(m-k) \right] \cdot \frac{\partial}{\partial a_i} \left[s_n(m) - \sum_{k=1}^{p} a_k s_n(m-k) \right] \\
&= -2 \sum_m \left[s_n(m) - \sum_{k=1}^{p} a_k s_n(m-k) \right] s_n(m-i) \\
&= -2 \left[\sum_m s_n(m) s_n(m-i) - \sum_m \sum_{k=1}^{p} a_k s_n(m-k) s_n(m-i) \right] \\
&= 0
\end{aligned} \tag{8.12}$$

giving

$$\sum_m s_n(m) s_n(m-i) = \sum_{k=1}^{p} \hat{a}_k \sum_m s_n(m-k) s_n(m-i) \tag{8.13}$$

where $\{\hat{a}_k\}$ is the set of values of a_k that minimizes E_n.

Defining the 'deterministic' short-time covariance matrix of $s_n(m)$ as a matrix having the (i,k)-th element as

$$\psi_n(i,k) = \sum_m s_n(m-k) s_n(m-i) \tag{8.14}$$

we can express equation (8.13) as:

$$\psi_n(i,0) = \sum_{k=1}^{p} a_k \psi_n(i,k), \quad i = 1, 2, \ldots, p \tag{8.15}$$

This is the general form of the Wiener-Hopf equation (also called Yule-Walker equation or the normal equation). It is a set of p linear equations in p unknowns.

The resulting MMSE (Minimum Mean-Squared Error) E_n can also be expressed in terms of $\hat{a}_k$ and $\psi_n(i,k)$:

$$
\begin{aligned}
E_n &= \sum_m \left[s_n(m) - \sum_{k=1}^{p} \hat{a}_k s_n(m-k) \right]^2 \\
&= \sum_m s_n^2(m) - 2 \sum_m \sum_{k=1}^{p} \hat{a}_k s_n(m) s_n(m-k) + \sum_m \sum_{k=1}^{p} \sum_{l=1}^{p} \hat{a}_k \hat{a}_l s_n(m-k) s_n(m-l) \\
&= \sum_m s_n^2(m) - 2 \sum_{k=1}^{p} \hat{a}_k \sum_m s_n(m) s_n(m-k) + \sum_{l=1}^{p} \hat{a}_l \left[\sum_{k=1}^{p} \hat{a}_k \sum_m s_n(m-k) s_n(m-l) \right] \\
&= \sum_m s_n^2(m) - 2 \sum_{k=1}^{p} \hat{a}_k \sum_m s_n(m) s_n(m-k) + \sum_{l=1}^{p} \hat{a}_l \sum_m s_n(m) s_n(m-l) \\
&= \sum_m s_n^2(m) - \sum_{k=1}^{p} \hat{a}_k \sum_m s_n(m) s_n(m-k) \\
&= \psi_n(0,0) - \sum_{k=1}^{p} \hat{a}_k \psi_n(0,k)
\end{aligned}
\tag{8.16}
$$

Notes and discussions

- to solve the Wiener-Hopf equations (8.15), we simply compute the values of $\psi_n(i,k)$ for $1 \leq i \leq p$ and $0 \leq k \leq p$, and then we solve the set of p linear equations
- Note that we did not define precisely the range of m for computing $\psi_n(i,k)$. There are two main methods that are used to define the limits of summation in $\psi_n(i,k)$, namely
  - the **autocorrelation** method, in which the signal segment is windowed and zero-padded so that m effectively ranges over $(-\infty, +\infty)$. Thus we cannot access data outside of the window, so that the number of terms in the sum depends on $|i-k|$
  - the **covariance method**, in which the summation of equation (8.14) is carried out from $m = 0$ to $m = M - 1$, such that we are accessing data outside the window
- The computational complexity and the numerical stability in solving the Wiener-Hopf equations depend on the prediction order p and the window length M

8.2 Statistical interpretation of LPC

Up to now we have been treating $s(n)$ as a deterministic signal (e.g. digitized samples of a real, recorded voice signal). Now let's assume that $\{s(n)\}$ is a real, discrete-time,

zero-mean stationary random process with autocorrelation $r_s(m) = E[s(n)s(n-m)]$ (which is real and even).

Let $\{\hat{s}(n)\}$ be a random process that predicts $\{s(n)\}$ as a linear combination of past samples of the process.

$$\hat{s}(n) = \sum_{k=1}^{p} a_k s(n-k) \tag{8.17}$$

Note that this is exactly same we had for the deterministic case in the previous section. Similarly to the deterministic case, we define the error (or residual) $e(n)$ by

$$e(n) = s(n) - \hat{s}(n) = s(n) - \sum_{k=1}^{p} a_k s(n-k) \tag{8.18}$$

Define $P(z) = \sum_{k=1}^{p} a_k z^{-k}$ as the prediction filter (the inverse filter $A(z) = 1 - P(z)$). We have, in the z-domain,

$$E(z) = S(z) - \hat{S}(z) = S(z) - P(z)S(z) = A(z)S(z) \tag{8.19}$$

Definition: the optimal (minimum) mean-square prediction error variance for linear prediction of order p is defined as

$$D_p = E[e(n)^2] \tag{8.20}$$

and the resulting prediction gain G_p is

$$G_p = \frac{\sigma_s^2}{D_p} \tag{8.21}$$

where σ_s^2 is the average power (variance) of the signal $s(n)$. One can show that in general $G_p \geq 1$. The worse case ($G_p = 1$) occurs when $s(n)$ is a zero-mean white noise, in which case $\hat{s}(n) = 0$ and $e(n) = s(n) - \hat{s}(n) = s(n)$

Solution to the MMSE Linear Prediction Problem

Goal: Find the set of linear prediction coefficients $\{a_k\}_{k=1}^{p}$ which minimize the prediction error variance

$$D_p = E[e(n)^2] = E[(s(n) - \hat{s}(n))^2] = E\left[(s(n) - \sum_{k=1}^{p} a_k s(n-k))^2\right] \tag{8.22}$$

Following the same line of reasoning as the deterministic case above, we differentiate with respect to the prediction coefficients and we set it equal to zero:

$$\frac{\partial D_p}{\partial a_i} = -2E\left[s(n-i)\left(s(n) - \sum_{k=1}^{p} a_k s(n-k)\right)\right] = -2E\left[s(n-i)e(n)\right] = 0 \quad i = 1, 2, \ldots, p \tag{8.23}$$

The above set of equations states the **Orthogonality Principle**, i.e. the optimal error signal (in a mean square sense) is always orthogonal to the predictors:

$$e(n) \perp s(n-i), \quad i = 1, 2, \ldots, p \tag{8.24}$$

or, equivalently

$$E[e(n)s(n-i)] = 0, \quad i = 1, 2, \ldots, p \tag{8.25}$$

Substituting $e(n) = s(n) - \hat{s}(n)$ yields:

$$\begin{aligned} E[e(n)s(n-i)] &= E[\{s(n) - \hat{s}(n)\}s(n-i)] \\ &= E[\{s(n) - \sum_{k=1}^{p} a_k s(n-k)\}s(n-i)] \\ &= E[s(n)s(n-i)] - E[\sum_{k=1}^{p} a_k s(n-k)s(n-i)] \end{aligned} \tag{8.26}$$

which implies

$$r_s(i) = \sum_{k=1}^{p} a_k r_s(i-k), \quad i = 1, 2, \ldots, p \tag{8.27}$$

Comparing this Wiener-Hopf equation with the one in the previous deterministic case we see that they are essentially the same (except for the definition of the autocorrelation function r_s vs the covariance matrix ψ we had before).

Prediction error variance

As in the deterministic case, the prediction error variance D_p for a given prediction order p can be expressed in terms of a_k and $R_s(i)$:

$$D_p = r_s(0) - \sum_{k=1}^{p} a_k r_s(k) \tag{8.28}$$

and the corresponding prediction gain is:

$$G_p = \frac{\sigma_s^2}{D_p} \tag{8.29}$$

8.3 Infinite memory linear prediction

Let's assume that we are predicting $s(n)$ using the entire set of past samples from $n = -\infty$ to $n-1$

$$\hat{s}(n) = \sum_{k=1}^{\infty} a_k s(n-k) \tag{8.30}$$

In this case, the following orthogonality relations hold:

$$E[e(n)s(n-i)] = 0, \quad i = 1, 2, \ldots, \infty \tag{8.31}$$

Computing the autocorrelation of the optimal error $r_e(i)$ gives us

$$\begin{aligned} r_e(i) &= E[e(n)e(n-i)] \\ &= E[e(n)\{s(n-i) - \hat{s}(n-i)\}] \\ &= E[e(n)\{s(n-i) - \sum_{k=1}^{\infty} a_k s(n-k-i)\}] \\ &= E[e(n)s(n-i)] - \sum_{k=1}^{\infty} a_k E[e(n)s(n-k-i)] \\ &= 0 \quad \forall i > 0 \end{aligned} \tag{8.32}$$

Since $r_e(i)$ is even, the above also holds for $i < 0$.

$$r_e(i) = D_p \delta(i) \tag{8.33}$$

Thus we see that $e(n)$ is a white noise process. Since $e(n)$ is purely random in some sense, we see that infinite-memory LP extracts all the information of $s(n)$ into the inverse filter $A(z)$, and the residual of the prediction process $e(n)$ is thus left with no sample-spanning information about the signal.

Assuming $s(n)$ is filtered white noise, LP divides out the filter in the frequency domain to produce a (flat) white noise spectrum.

Infinite memory LP can be characterized as a whitening process on the signal $s(n)$.

Remarks:

- After obtaining the prediction error $e(n)$ from the prediction, we can recover the original signal back from $e(n)$ and $A(z)$. Indeed, we can get $s(n)$ by feeding $e(n)$ into the inverse filter $H(z) = \frac{1}{A(z)}$. Thus we can say that $e(n)$ and $A(z)$ fully characterize $s(n)$

- Whitening filter: A byproduct of the prediction is that we end up with a statistically uncorrelated error process (white noise). The reason is that if $e(n)$ is white, then all the correlation information of $s(n)$ is contained in the inverse filter $A(z)$ and we can use any white noise with the same variance in the reconstruction process.

  Let's use any arbitrary white noise $e'(n)$ instead of $e(n)$ and let $s'(n)$ be the output of the filter $H(z)$ with $e'(n)$ as the input. Also assume that $e'(n)$ and $e(n)$ have the same power spectrum:

  $$S_{e'}(\omega) = S_e(\omega) = D_p \text{ const.} \tag{8.34}$$

  Since the power spectrum of the output only depends on the power spectrum of the input and the filter, we can deduce that although $s'(n)$ is not identical to $s(n)$ it would have the same power spectrum $S_s(\omega)$. Figure 8.2 illustrates the procedure of whitening/shaping.

  Note that in general, only the infinite-memory LP results in a true whitening filter $A(z)$. But by convention we call $A(z)$ the whitening filter even if the prediction order is finite. In the finite case, the spectrum of the prediction error is flattened, but not white

- Shaping filter:
  - $H(z) = 1/A(z)$ is called the shaping filter
  - in order for us to get back $s(n)$ from $e(n)$ we need a stable $H(z)$ (or a minimum phase $A(z)$)

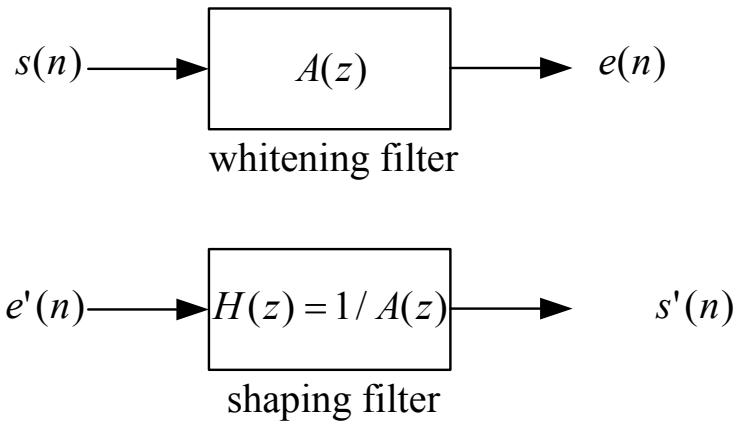


Fig. 8.2. Whitening and shaping filters

8.4 Computation of the LPC parameters

As described in the previous section, there are various ways to obtain the predictor coefficients a_k from the Wiener-Hopf Equation. This is because there are many ways of defining the limits of summation in computing

$$\psi_n(i,k) = \sum_m s_n(m-i)s_n(m-k) \tag{8.35}$$

- **Autocorrelation method**: We assume that the interval for computing $\psi_n(i,k)$ is $[0, M+p-1]$, where p is the prediction order, and define

$$\psi_n^A(i,k) = \hat{r}_n(|i-k|) = \sum_{m=0}^{M-1-(i-k)} s_n(m)s_n(m+i-k) \quad \text{for } 1 \le i \le p, 0 \le k \le p, \text{ and } i \ge k \tag{8.36}$$

  Note that $\psi_n^A(i,k) = \psi_n^A(k,i)$, and $s_n(m)$ is accessed over the range $[0, M-1]$. We shall see that this is the true autocorrelation of the zero padded short time signal $s_n(m)$, and hence the name autocorrelation method is appropriate. To obtain a_k using the autocorrelation method we use what is called the **Levinson-Durbin** algorithm

- Covariance method: Define

$$\psi_n^C(i,k) = \sum_{m=0}^{M-1} s_n(m-i)s_n(m-k) \quad \text{for } 1 \le i \le p, \quad 0 \le k \le p \tag{8.37}$$

  Note that $\psi_n^C(i,k) = \psi_n^C(k,i)$, and $s_n(m)$ is accessed in the range $[-p, M-1]$ ($M+p$ samples). We shall see that $\psi_n^C(i,k)$ is not the true autocorrelation, but the cross-correlation between two very similar (but not identical) finite segments of the signal $s(n)$. In order to solve the values of a_k's we use a method known under the name of **Cholesky decomposition**.

  In the following we describe the autocorrelation method only.

Autocorrelation Method

Consider a short-time average prediction error defined as

$$E_n = \sum_m e_n^2(m) = \sum_m [s_n(m) - \sum_{k=1}^{p} a_k s_n(m-k)]^2 \tag{8.38}$$

One approach to defining the limits in the above is to assume that the short-time signal $s_n(m)$ is zero outside the interval $0 \leq m \leq M-1$. This is equivalent to defining $s_n(m)$ to be the original signal windowed by a length M rectangular window, starting at time index n:

$$s_n(m) = s(n+m)w(m) \tag{8.39}$$

where $w(m)$ is a causal, rectangular window defined on the interval $m \in [0, M-1]$.

The above definition of $s_n(m)$ results in the prediction error $e_n(m) = s_n(m) - \sum_{k=1}^{p} a_k s_n(m-k)$ being nonzero over the range $m \in [0, M-1+p]$. Thus, the limit of the summation would be $[0, M-1+p]$. The corresponding short-time covariance $\psi_n(i,k)$ has the identical limit of summation:

$$\psi_n^A(i,k) = \sum_{m=0}^{M+p-1} s_n(m-i)s_n(m-k) \quad (1 \leq i \leq p, \quad 0 \leq k \leq p) \tag{8.40}$$

Since $s_n(m)$ is zero outside $[0, M-1]$ we can show that

$$\begin{aligned} \psi_n^A(i,k) &= \sum_{m=0}^{M+p-1} s_n(m-i)s_n(m-k) \\ &= \sum_{m=-i}^{M+p-1-i} s_n(m)s_n(m+i-k) \\ &= \sum_{m=0}^{M-1-(i-k)} s_n(m)s_n(m+i-k) \end{aligned} \tag{8.41}$$

Since the sample (unnormalized) autocorrelation of $s_n(m)$ is defined as

$$\hat{r}_n(k) = \sum_{-\infty}^{+\infty} s_n(m)s_n(m+k) = \sum_{m=0}^{M-1-k} s_n(m)s_n(m+k) \tag{8.42}$$

we can see that ψ_n^A is indeed equal to the true sample autocorrelation of the rectangular-windowed signal frame $s_n(m)$.

$$\psi_n^A = \hat{r}_n(|i-k|) \quad (1 \leq i \leq p, 0 \leq k \leq p) \tag{8.43}$$

(Note $\hat{r}_n(k) = \hat{r}_n(-k)$ for real signals)

We thus have the following Wiener-Hopf equations (also called the normal equations) for the autocorrelation method:

$$\sum_{k=1}^{p} a_k \hat{r}_n(|i-k|) = \hat{r}_n(i), \quad i = 1, 2, \ldots, p \tag{8.44}$$

or, in matrix form,

$$\begin{bmatrix} \hat{r}_n(0) & \hat{r}_n(1) & \hat{r}_n(2) & \cdots & \hat{r}_n(p-1) \\ \hat{r}_n(1) & \hat{r}_n(0) & \hat{r}_n(1) & \cdots & \hat{r}_n(p-2) \\ \hat{r}_n(2) & \hat{r}_n(1) & \hat{r}_n(0) & \cdots & \hat{r}_n(p-3) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \hat{r}_n(p-1) & \hat{r}_n(p-2) & \hat{r}_n(p-3) & \cdots & \hat{r}_n(0) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_p \end{bmatrix} = \begin{bmatrix} \hat{r}_n(1) \\ \hat{r}_n(2) \\ \hat{r}_n(3) \\ \vdots \\ \hat{r}_n(p) \end{bmatrix} \tag{8.45}$$

Remarks:

- Edge effect on the prediction error
  - The prediction error $e_n(m)$ can be large at the beginning of the interval ($0 \leq m \leq p-1$) because we are trying to predict $s_n(m)$ from some past samples $s_n(m-k)$ that are not included in the windowed frame, and thus are zero from our definition of $s_n(m)$
  - Likewise, the error can also be large at the end of the interval ($M \leq m \leq M+p-1$) because we are trying to predict out-of-frame samples (which are zero) from samples that are nonzero
  - Solution: We use a tapered windows (i.e. Hamming window) to make the samples at the edge negligible
- The $p \times p$ matrix R_n is a symmetric Toeplitz matrix, i.e. all the element along a given diagonal are equal, and it is symmetric. It is this a special property that allows us to use efficient algorithms to solve the normal equations and obtain the predictor coefficients $\{a_k\}$. One of the most widely used algorithms for solving equations involving a Toeplitz matrix is the Levinson-Durbin algorithm which will be discussed in the next section

Listing 8.1. Matlab

```
M = 1000; p = 3;

a = [0.8; 0.1; 0.05];

randn('state',0)
e = randn(M,1);

s = filter(1, [1; -a], e);

subplot(2,1,1) plot(s);
subplot(2,1,2) plot(e)

r = zeros(p+1, 1);
for t = 0:p
```

```
    for m = 0:M-1-t
    r(t+1) = r(t+1) + s(m+1)*s(m+t+1);
    end
end

% this is equivelent to
% [r2] = xcorr(s, p);
% r = r2(p+1:end);

R = toeplitz(r(1:p));
a_est = R^(-1)*r(2:p+1);

% this is equivalent to
% a_est = levinson(r, p);
% a_est = -a_est(2:end);

e_res = filter([1; -a_est], 1, s);
var(e_res)
```

Levinson-Durbin algorithm

For the autocorrelation method, the normal equation for obtaining the predictor coefficients is

$$\begin{bmatrix} \hat{r}_n(0) & \hat{r}_n(1) & \hat{r}_n(2) & \cdots & \hat{r}_n(p-1) \\ \hat{r}_n(1) & \hat{r}_n(0) & \hat{r}_n(1) & \cdots & \hat{r}_n(p-2) \\ \hat{r}_n(2) & \hat{r}_n(1) & \hat{r}_n(0) & \cdots & \hat{r}_n(p-3) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \hat{r}_n(p-1) & \hat{r}_n(p-2) & \hat{r}_n(p-3) & \cdots & \hat{r}_n(0) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_p \end{bmatrix} = \begin{bmatrix} \hat{r}_n(1) \\ \hat{r}_n(2) \\ \hat{r}_n(3) \\ \vdots \\ \hat{r}_n(p) \end{bmatrix} \tag{8.46}$$

By exploiting the Toeplitz nature of the matrix, we can devise a recursive algorithm for solving the equation. One of the most popular procedures is the Levinson-Durbin algorithm.

The objective of the algorithm is to find the LP (linear predictor) coefficients $a_1^{(i)}, a_2^{(i)}, \ldots, a_p^{(i)}$ and the corresponding predictor error variance E_n^i recursively, for $i = 1, 2, \ldots, p$.

For $i = 1, 2, \ldots, p$, we carry out the following recursion:

1. Initialization

$$E^{(0)} = \hat{r}(0) \tag{8.47}$$

2. Compute 'partial correlation' (PARCOR) coefficients

$$k_i = \left[\hat{r}(i) - \sum_{j=1}^{i-1} a_j^{i-1} \hat{r}(i-j) \right] / E^{(i-1)}, \quad 1 \leq i \leq p \tag{8.48}$$

3. Calculate the predictor coefficients for order i

$$a_i^{(i)} = k_i \tag{8.49}$$

$$a_j^{(i)} = a_j^{(i-1)} - k_i a_{i-j}^{(i-1)}, \quad 1 \leq j \leq i-1 \tag{8.50}$$

4. Update the predictor error

$$E^{(i)} = (1 - k_i^2) E^{(i-1)} \tag{8.51}$$

5. Obtain the final solution

$$a_j = a_j^{(p)} \tag{8.52}$$

Remarks:

- In carrying out the recursion, the LP coefficients and the corresponding prediction error for all orders less than p are automatically obtained

$$a_j^{(i)} \text{ (j-th predictor coefficient for prediction order } i) \tag{8.53}$$

$$E^{(i)} \text{ (Optimal prediction error for prediction order } i) \tag{8.54}$$

- The k_i's are called PARCOR coefficients. The name PARCOR stems from the word 'partial correlation'. PARCOR coefficients have been studied extensively in the context of lattice filters. It can be shown that the whitening filter $A(z)$ can be constructed by cascading single lattice structures with coefficients equal to $-k_i$ as shown in the figure below.

- Recall that $A(z)$ has to have all roots inside the unit circle in order for the shaping filter $H(z)$ to be stable. There is a necessary and sufficient condition for stability in terms of PARCOR coefficients

$$|k_i| \leq 1, \quad \forall i \Longleftrightarrow H(z) \text{ is stable} \tag{8.55}$$

- It can be shown that k_i's obtained using the Levinson-Durbin algorithm always have magnitude less or equal to unity, guaranteeing the stability of the shaping filter.

- The following conditions on the k_i's give more insight into the problem of prediction:

  1. $0 < |k_i| < 1$: the error will always decrease as we increase the prediction order from $i-1$ to i, since $E^{(i)} = (a - k_i^2) E^{(i-1)} < E^{(i-1)}$

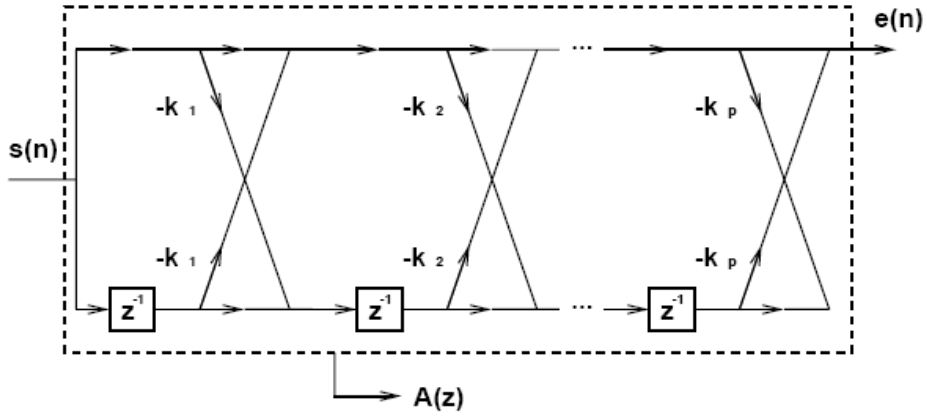


Fig. 8.3. Lattice filter implementation of linear prediction

2. $k_i = \pm 1$: In this case, $E^{(i)} = 0$. This happens when the signal is fully deterministic, i.e. all the samples can be expressed exactly as a linear combination of past i samples

3. $k_i = 0 : E^{(i)} = E^{(i-1)}$, which implies that there is no benefit in terms of reducing the error by increasing the prediction order from $i-1$ to i

Listing 8.2. Matlab

```
M = 100000; p = 6;

a = [0.8; 0.1; 0.05];

randn('state',0) e = randn(M,1);

s = filter(1, [1; -a], e);

r = xcorr(s, p);
r = r(p+1:end);

% what follows is equivalent to
% [A,E,K] = LEVINSON(r, i)
% for i = 1:p

E_est = zeros(p+1, 1);
k_est = zeros(p, 1);
a_est = zeros(p, p);

E_est(1) = r(1);

for i = 1:p

    temp = 0;
    for j = 1:i - 1
        temp = temp + a_est(j, i - 1)*r(i - j + 1);
    end
    k_est(i) = (r(i + 1) - temp)/E_est(i);

    a_est(i,i) = k_est(i);

    for j=1:i-1
        a_est(j, i) = a_est(j, i - 1) - k_est(i)*a_est(i-j, i-1);
    end

    E_est(i + 1) = (1 - k_est(i)^2)*E_est(i);
```

```
end
plot(E_est(2:end))
```

8.5 Frequency domain interpretation of LPC analysis

LPC analysis is basically a correlation type of analysis which can be approached either from the time or from the frequency domain. We can see this by examining the autocorrelation theorem:

$$x * x \longleftrightarrow |X|^2 \tag{8.56}$$

Since the solution of the normal equations depends on the autocorrelation $r(k)$ of the signal (in the case of autocorrelation method) we see that the solution is also a function of the power spectrum of the signal.

Given the error in the time domain:

$$e(n) = s(n) - \hat{s}(n) = s(n) - \sum_{k=1}^{p} a_k s(n-k) \tag{8.57}$$

Applying the DTFT to the previous equation gives us the frequency domain representation of the error signal $e(n)$

$$E(e^{j\omega}) = \left[1 - \sum_{k=1}^{p} a_k e^{-j\omega k}\right] S(e^{j\omega}) = A(e^{j\omega}) S(e^{j\omega}) \tag{8.58}$$

Since the energy of the prediction error can be expressed in the time domain as

$$E_n = \sum_{m=0}^{M-1+p} e_n^2(m) = \sum_{m=-\infty}^{+\infty} e_n^2(m) \tag{8.59}$$

we can express the above in the frequency domain using Parseval's Theorem:

$$E_n = \frac{1}{2\pi} \int_{-\pi}^{+\pi} |E_n(e^{j\omega})|^2 d\omega = \frac{1}{2\pi} \int_{-\pi}^{+\pi} |S_n(e^{j\omega})|^2 |A_n(e^{j\omega})|^2 d\omega \tag{8.60}$$

but since

$$H(e^{j\omega}) = \frac{1}{A(e^{j\omega})} \tag{8.61}$$

we can express the previous equation as

$$E_n = \frac{1}{2\pi} \int_{-\pi}^{+\pi} \frac{|S_n(e^{j\omega})|^2}{|H_n(e^{j\omega})|^2} d\omega \tag{8.62}$$

Since the integrand is positive, we conclude the following:

$$\min_{a_i} E_n \Longleftrightarrow \min_{a_i} \frac{|S_n(e^{j\omega})|^2}{|H_n(e^{j\omega})|^2}, \quad \forall\omega \tag{8.63}$$

Perfect Reconstruction of the Power Spectrum

We saw earlier that in the case of infinite memory LP, where $p = \infty$, the power spectrum of the signal can be exactly reconstructed from the shaping filter $H(z)$ and the error variance $D_p = G^2$. This implies that, as $p \longrightarrow \infty$, we can approximate the power spectrum of the signal with arbitrarily small error using the all-pole shaping filter $H(z)$:

$$\lim_{p\to\infty} |\hat{S}_n(e^{j\omega})|^2 = |S_n(e^{j\omega})|^2 \tag{8.64}$$

where $\hat{S}_n(e^{j\omega}) = GH_n(e^{j\omega})$ is the LPC spectrum, and $S_n(e^{j\omega})$ is the true spectrum of the signal ($S_n(e^{j\omega})$ is the DTFT of the length-M signal $s_n(m)$. As such, with respect to Chapter 7, $|S_n(e^{j\omega})|^2$ is a unnormalized periodogram estimate of the true power spectrum $\phi_s(\omega)$).

Power Spectral Matching properties of LPC

As the prediction order p increases, we saw earlier that the resulting energy or the prediction error E_n monotonically decreases. This implies that as we increase the prediction order, the LPC power spectrum $|\hat{S}_n(e^{j\omega})|^2$ will try to match the signal power spectrum $|S_n(e^{j\omega})|^2$ more closely. Figure 8.4 shows a comparison between the log magnitude of the LPC spectrum and that of the speech spectrum for a vowel segment. with various orders of prediction ($p = 4, 8, 16, 32, 64, 128$)

As LPC tries to minimize $\frac{|S_n(e^{j\omega})|^2}{|H_n(e^{j\omega})|^2}$ for $\omega \in [-\pi, +\pi]$ in the integral, there exists an interesting discrepancy in carrying out the minimization of the integral

- Region 1: $|S_n(e^{j\omega})| > |H_n(e^{j\omega})|$

  This corresponds to the region where the magnitude of the signal spectrum is large. In this case we see that

  $$\frac{|S_n(e^{j\omega})|}{|H_n(e^{j\omega})|} > 1 \tag{8.65}$$

  implying that the integrand contributing to the error integral is relatively large (greater than 1).

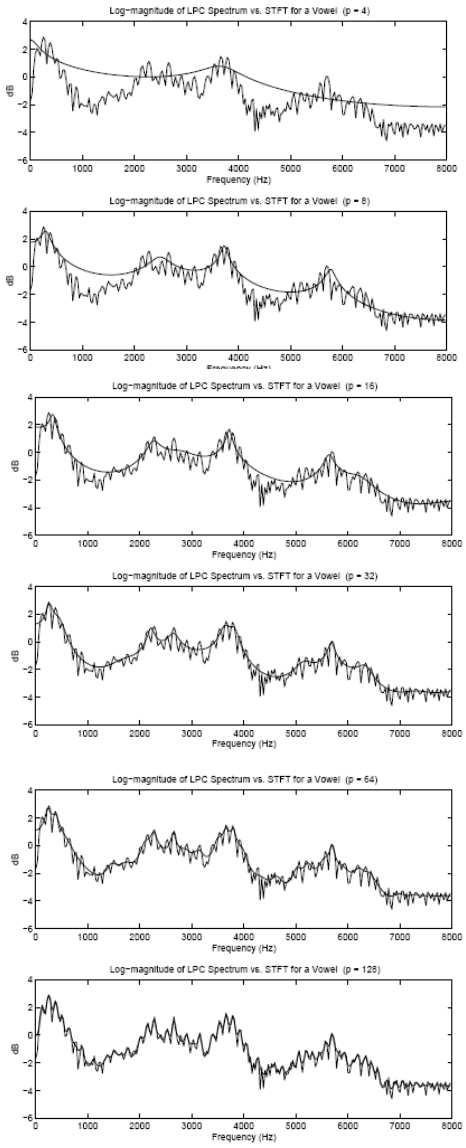


Fig. 8.4. LPC spectral matching for $p = 4, 8, 16, 32, 64, 128$

- Region 2: $|S_n(e^{j\omega})| < |H_n(e^{j\omega})|$

  This corresponds to the region where the magnitude of the signal spectrum is small. In this case we see that

$$\frac{|S_n(e^{j\omega})|}{|H_n(e^{j\omega})|} < 1 \tag{8.66}$$

  implying that the integrand contributing to the error integral is relatively small (less than 1)

From the above argument, it is clear that the LPC spectrum matches the signal spectrum much more closely in region 1 (near the spectrum peaks) than in region 2 (near spectral valleys) because integrands in region 1 contribute more to the total error than those in region 2.

Thus, the LPC spectrum can be considered to be a good spectral envelope estimator since it puts more emphasis on tracking peaks than tracking valleys. Figure 8.4 beautifully illustrates the peak tracking capabilities of the LPC.

Of course, as we increase the order p the approximation to the valleys is going to improve as well as for the peaks since the total error becomes smaller.

Smoothness as a function of prediction order

Now let us consider the effects of prediction order p on the properties of the LPC spectrum. We can see from Figure 8.4 that as we increase the order, the LPC spectrum tries to capture more closely the fine structure of the signal spectrum, and thus becomes less smooth.

Thus, the prediction order p can serve as a control parameter for determining the smoothness of the LPC spectrum. If our objective is to capture the spectral envelope of the spectrum and not the fine structure, then it is essential that we choose an appropriate value of p. If p is too large, the LPC spectrum would too closely match the spectrum and would not be smooth. If p is too small, the necessary peaks (i.e. formants) of the envelope may not be captured.

A good rule of thumb for speech is to use $f_s/1000 \leq p \leq f_s/1000 + 4$, where f_s is the sampling rate in Hz. Thus, for example, if the sampling rate is 16kHz, then using $16 \leq p \leq 20$ would be appropriate.

Listing 8.3. Matlab

```
load mtlb Nfft = 2000;
s = mtlb(701:701 + Nfft);

phi_p = (1/Nfft)*abs(fft(s,Nfft)).^2;
phi_p(1:Nfft/2);
w = 2*pi*[0:Nfft-1]/Nfft;

p = 20;

r = xcorr(s, p);
r = r(p+1:end);

[a, e] = levinson(r, p);
e = e/Nfft;

H = freqz(1, a, w);
plot(w, 10*log10(phi_p));
hold on plot(w, 10*log10(e*abs(H).^2),'r', 'LineWidth', 2);
axis([0 pi -60 20]);
hold off
```

Frequency selective linear prediction

We saw in the previous section that the prediction order p can be adjusted to control the accuracy and the smoothness of the LPC spectrum. In some cases, it would be nice to perform separate LPC analysis for a selected partition of the spectrum.

Example: For voiced speech, such as vowels, we are generally interested in the region from 0 to 4kHz. For unvoiced sounds, such as fricatives, the region from 4 to 8kHz is important.

Motivation: Using frequency selective linear prediction, the spectrum from 0 to 4 kHz can be modeled by a predictor of order p_1, while the region from 4 to 8 kHz can be modeled by a different predictor of order p_2. In most of the cases, we want a smoother fit (smaller p) in the higher octaves.

To model only the region from $f = f_A$ to $f = f_B$, we perform the following:

1. Map to a normalized frequency:

$$f = f_A \Longrightarrow f' = 0 \tag{8.67}$$

$$f = f_B \Longrightarrow f' = \omega'/2\pi = 0.5 \tag{8.68}$$

2. Obtain the new autocorrelation coefficients by IDFT:

$$\hat{r}'(n) = \frac{1}{2\pi} \int_{-\pi}^{+\pi} |S_n(e^{j\omega'})|^2 e^{j\omega'} d\omega' \tag{8.69}$$

3. Solve the new set of normal equations using $\{R'(k)\}$ to get the predictor coefficients for that particular spectral region.

Figure 8.5 illustrates the method of frequency selective linear prediction. The signal is a vowel segment used in Figure 8.4 sampled at 16 kHz. The region from 0 to 4 kHz is modeled by a 16-th order predictor ($p_1 = 16$). while the region from 4 to 8 kHz is modeled by a 4th-order predictor ($p_2 = 4$)

Note that at the boundary between the two regions ($f = 4$ kHz) there is a discontinuity. This arises from the fact that there is no continuity constraint for the boundary conditions.

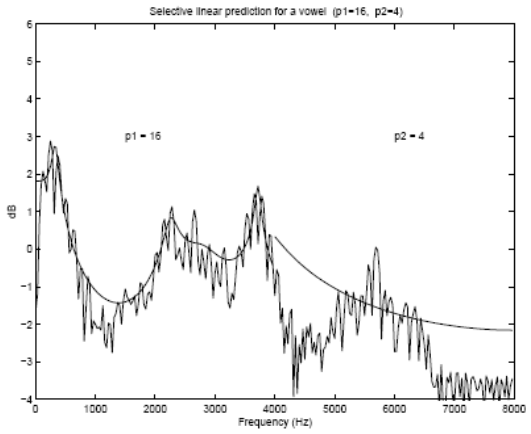


Fig. 8.5. Frequency selective linear prediction of a vowel

8.6 Applications of LPC

1. **Speech Coding/Synthesis**

   We can devise a model of speech production where $e(n)$ is the excitation source at the glottis, and $H(z)$ represents the transfer function of the vocal tract. Moreover, we can encode the LPC parameters to achieve data compression

2. **Pitch prediction**

   We can devise an optimal predictor that predicts the current sample from the past value one pitch period earlier

$$\hat{s}(n) = \beta s(n - P) \tag{8.70}$$

   where P is the pitch period. The goal is to minimize

$$E(\beta, P) = \sum_n [s(n) - \beta s(n - P)]^2 \tag{8.71}$$

   By differentiating, we can get the optimal pitch value P along with the optimal predictor coefficient β. This method is used in CELP (Coded-Excited Linear Prediction) speech coders.

   It is also straightforward to extend the above idea, and predict $s(n)$ from a group of samples centered one period in the past

$$\hat{s}(n) = \beta_{-\frac{L}{2}} s(n - P - \frac{L}{2}) + \cdots + \beta_0 s(n - P) + \cdots + \beta_{\frac{L}{2}} s(n - P + \frac{L}{2}) \tag{8.72}$$

3. **Cross Synthesis in Computer Music - Talking Instruments**

   We may feed any sound (typically that of a musical instrument) into the shaping (synthesis) filter $H(z)$ obtained from LPC analysis of a speech segment (usually vowels). Then, the musical signal input acts as a periodic excitation source $e(n)$ to the shaping filter $H(z)$, and thus the output spectrum will possess vocal formant structure as well as the harmonic and textural qualities of the musical sound.

4. **Spectral Envelope Estimation**

References

- Smith, J.O., Lecture notes,

  `http://www-ccrma.stanford.edu/~jos/`

9

Wiener filtering

9.1 Problem formulation

General Wiener filtering problem: design an LTI filter $f(n)$ that minimizes the expected value of $E[|e(n)|^2]$

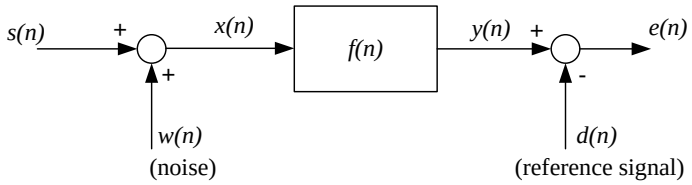


Fig. 9.1. Wiener filtering.

Examples:

- $d(n) = s(n)$: estimation problem (noise removal);
- $d(n) = s(n+m)$, $m > 0$, prediction problem (stock market prediction, LPC, DPCM coding);
- $d(n) = s(n-m)$, $m > 0$: "smoothing" problem.

Assumptions:

- $x(n)$, $s(n)$, $w(n)$ are WSS random sequences
- $x(n)$, $d(n)$ are jointly WSS, i.e. $E[x(n)d^*(n-m)]$ depends only on m

Constraints:

- $f(n)$ is usually, but not always, constrained to be causal.

9.2 Wiener-Hopf equations

Goal: minimize the mean-square error (MSE)

$$\epsilon = E[(d(n) - y(n))^2] \tag{9.1}$$

Expanding this expression gives

$$\begin{aligned}\epsilon &= E[(d(n) - y(n))^2] \\ &= E[d(n)d^*(n)] - \sum_k f^*(k)E[d(n)x^*(n-k)] - \sum_k f(k)E[d^*(n)x(n-k)] + \\ &\quad + \sum_k \sum_m f(m)f^*(k)E[x(n-m)x^*(n-k)]\end{aligned} \tag{9.2}$$

Hence

$$\epsilon = r_d(0) - 2\Re\left\{\sum_k f^*(k)r_{dx}(k)\right\} + \sum_k \sum_m f(m)f^*(k)r_x(k-m) \tag{9.3}$$

This is a real-valued quadratic function of the possibly complex variables $f(i)$ for all i. This suggests that by taking the partial derivative of ϵ with respect to each $f(i)$, and setting this to zero, we can find the values that minimize ϵ. Note that ϵ is a real-valued quadratic function of the real and imaginary parts $f_R(i)$ and $f_I(i)$ of $f(i)$. Hence, we can take the partial derivative with respect to each of these real-valued variables and set those to zero. The values of $f(i)$ that make the partial derivatives zero yield a minimum ϵ is the function is unimodal. We assume for now that it is.

Instead of setting the partial derivatives to zero, we can equivalently set the following complex gradients to zero,

$$\nabla_{f(i)}\epsilon = \frac{1}{2}\left(\frac{\partial}{\partial f_R(i)} - j\frac{\partial}{\partial f_I(i)}\right)\epsilon = 0 \tag{9.4}$$

and

$$\nabla_{f^*(i)}\epsilon = \frac{1}{2}\left(\frac{\partial}{\partial f_R(i)} + j\frac{\partial}{\partial f_I(i)}\right)\epsilon = 0 \tag{9.5}$$

The equivalence follows from the observation that

$$\nabla_{f^*(i)}\epsilon + \nabla_{f^(i)}\epsilon = \frac{\partial \epsilon}{\partial f_R(i)} \tag{9.6}$$

and

$$\nabla_{f^*(i)}\epsilon - \nabla_{f^(i)}\epsilon = j\frac{\partial \epsilon}{\partial f_I(i)} \tag{9.7}$$

With our definition of the complex gradient, it is easy to verify that

$$\nabla_f f^* = 0 \quad \nabla_f f = 1 \quad \nabla_f f f^* = f^* \tag{9.8}$$

and

$$\nabla_{f^*} f^* = 1 \quad \nabla_{f^*} f = 0 \quad \nabla_{f^*} f f^* = f \tag{9.9}$$

i.e. $\nabla_f f = \nabla_f (f_R + j f_I) = \frac{1}{2} + \frac{1}{2} = 1$.

Using these we get

$$\nabla_{f^*(i)} \epsilon = -r_{dx}(i) + \sum_k f(k) r_x(i-k) \tag{9.10}$$

It is easy to show that $(\nabla_{f^*(i)} \epsilon)^* = \nabla_f \epsilon$ (using conjugate symmetry of $r_x(m)$), sot it is sufficient to set only one of the two gradients to zero. Setting $\nabla_{f^*(i)} \epsilon = 0$ we get the *Wiener-Hopf* equations:

$$\boxed{\sum_k f(k) r_x(i-k) = r_{dx}(i) \quad \forall i} \tag{9.11}$$

A solution to these equations will minimize ϵ. We can easily modify these equations to constrain the filter $f(n)$ to be causal by just modifying the limits on the summation.

$$\boxed{\sum_{k=0}^{\infty} f(k) r_x(i-k) = r_{dx}(i) \quad \forall i \geq 0} \tag{9.12}$$

Some examples

- Suppose $d(n) = x(n)$. Then $r_{dx}(i) = r_x(i)$. The Wiener-Hopf equations are satisfied by

$$f(n) = \delta(n) \tag{9.13}$$

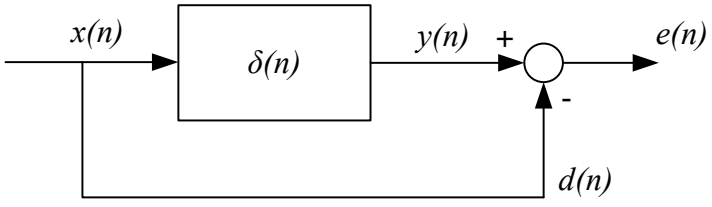


Fig. 9.2. Wiener filtering. $d(n) = x(n)$

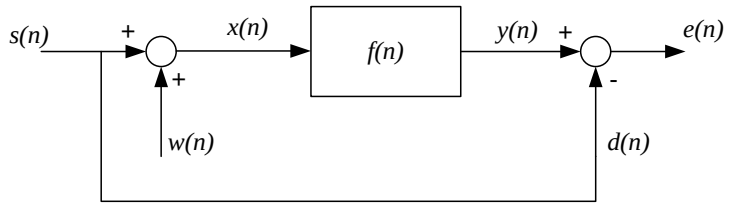


Fig. 9.3. Wiener filtering. $d(n) = s(n)$

- Suppose $d(n) = s(n) = x(n) - w(n)$, where $w(n)$ is white and independent of $x(n)$ (a very strange situation). Again, $r_{dx} = r_x(i)$, so the solution is the same.

  Because $x(n)$ is independent of the noise $w(n)$, there is nothing the filter can do to remove it. So the solution is the same as before. But a more realistic example would be for $w(n)$ to be independent of $s(n)$.

- Suppose $x(n)$ is white, i.e. $r_x(k) = \sigma^2\delta(k)$. Then the Wiener-Hopf equations are satisfied by

$$f(n) = \frac{r_{dx}(n)}{\sigma^2}u(n) \tag{9.14}$$

  where $u(n)$ is the unit step function (so this filter is causal). Knowing the joint statistics of $x(n)$ and $d(n)$, we can find the MMSE filter that estimates $d(n)$ from $x(n)$. The solution is easy when $x(n)$ is white. For other situations, we can use a whitening filter to construct a general solution.

9.3 Non-causal Wiener filter solution

Equation (9.11), which has no causality constraint on the Wiener filter, can be written

$$f(i) * r_x(i) = r_{dx}(i) \tag{9.15}$$

Taking the z transform and solving for $F(z)$ we get

$$F(z) = \frac{S_{dx}(z)}{S_x(z)} \tag{9.16}$$

Thus

$$F(e^{j\omega}) = \frac{S_{dx}(e^{j\omega})}{S_x(e^{j\omega})} \tag{9.17}$$

When $s(n) = d(n)$ and $r_{sw}(m) = E[s(n)w^*(n-m)] = c\delta(m)$ (i.e. $w(n)$ and $s(n)$ are uncorrelated), therefore

$$F(e^{j\omega}) = \frac{S_s(e^{j\omega})}{S_s(e^{j\omega}) + S_w(e^{j\omega})} \tag{9.18}$$

9.4 Causal Wiener filter solution

The causality constraint 9.12 complicates things considerably. Conceptually break the filter into two parts

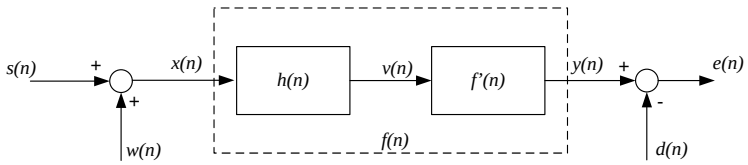


Fig. 9.4. Causal Wiener filtering.

where $h(n)$ is the whitening filter for $x(n)$, $v(n)$ is the innovation process, and $f'(n)$ is the Wiener filter designed for white input, i.e.

$$f'(n) = \frac{r_{dv}(n)}{\sigma_v^2} u(n) \tag{9.19}$$

Denote

$$[R_{dv}(z)]_+ = \sum_{k=0}^{\infty} r_{dv}(k) z^{-k} \tag{9.20}$$

i.e. the z transform of the nonnegative side only. With this,

$$F'(z) = \frac{1}{\sigma_v^2} [R_{dv}(z)]_+ \tag{9.21}$$

A more convenient form would express $F'(z)$ as a function of $R_{dx}(z)$, which is assumed to be known.

Note that since

$$v(n) = \sum_{k=0}^{\infty} h(k) x(n-k) \tag{9.22}$$

$$r_{dv}(m) = E[d(i)v^*(i-m)] = \sum_{k=0}^{\infty} h^*(k) E[d(i)x^*(i-m-k)] = \sum_{k=0}^{\infty} h^*(k) r_{dx}(m+k) \tag{9.23}$$

This relation can be written

$$r_{dv}(-m) = h^*(m) * r_{dx}(-m) \tag{9.24}$$

Taking the z transform on both sides

$$R_{dv}\left(\frac{1}{z}\right) = H^*(z^*) R_{dx}\left(\frac{1}{z}\right) \tag{9.25}$$

or

$$R_{dv}(z) = H^* \left(\frac{1}{z^*} \right) R_{dx}(z)) \tag{9.26}$$

Hence, the overall Wiener filter solution is

$$F(z) = H(z)F'(z) = \frac{1}{\sigma_v^2} H(z) \left[H^*(\frac{1}{z^*}) R_{dx}(z) \right]_+ \tag{9.27}$$

References

- Ed Lee, Lecture Notes, UC Berkeley,

  \http://ptolemy.eecs.berkeley.edu/~eal/ee225a/Supplement.pdf