CHAPTER 5

The Discrete Fourier Transform

In Chapters 3 and 4 we studied transform-domain representations of discrete signals. The discrete-time Fourier transform provided the frequency-domain (ω) representation for absolutely summable sequences. The z-transform provided a generalized frequency-domain (z) representation for arbitrary sequences. These transforms have two features in common. First, the transforms are defined for infinite-length sequences. Second, and the most important, they are functions of continuous variables (ω or z). From the numerical computation viewpoint (or from MATLAB's viewpoint), these two features are troublesome because one has to evaluate *infinite sums* at *uncountably infinite* frequencies. To use MATLAB, we have to truncate sequences and then evaluate the expressions at finitely many points. This is what we did in many examples in the two previous chapters. The evaluations were obviously approximations to the exact calculations. In other words, the discrete-time Fourier transform and the z-transform are not *numerically computable* transforms.

Therefore we turn our attention to a numerically computable transform. It is obtained by sampling the discrete-time Fourier transform in the frequency domain (or the z-transform on the unit circle). We develop this transform by first analyzing periodic sequences. From Fourier analysis we know that a periodic function (or sequence) can always be represented by a linear combination of harmonically related complex exponentials (which is a form of sampling). This gives us the *discrete Fourier series* (DFS) representation. Since the sampling is in the frequency domain, we study the effects of sampling in the time domain and the issue of reconstruction in the z-domain. We then extend the DFS to *finite-duration sequences*, which leads to a new transform, called the *discrete Fourier transform* (DFT). The DFT avoids the two problems mentioned and is a numerically computable transform that is suitable for computer implementation. We study its properties and its use in system analysis in detail. The numerical computation of the DFT for long sequences is prohibitively time-consuming. Therefore several algorithms have been developed to efficiently compute the DFT. These are collectively called fast Fourier transform (or FFT) algorithms. We will study two such algorithms in detail.

5.1 THE DISCRETE FOURIER SERIES

In Chapter 2 we defined the periodic sequence by $\tilde{x}(n)$, satisfying the condition

$$\tilde{x}(n) = \tilde{x}(n + kN), \quad \forall n, k \tag{5.1}$$

where N is the fundamental period of the sequence. From Fourier analysis we know that the periodic functions can be synthesized as a linear combination of complex exponentials whose frequencies are multiples (or harmonics) of the fundamental frequency (which in our case is $2\pi/N$). From the frequency-domain periodicity of the discrete-time Fourier transform, we conclude that there are a finite number of harmonics; the frequencies are $\{\frac{2\pi}{N}k, \quad k = 0, 1, \ldots, N-1\}$. Therefore a periodic sequence $\tilde{x}(n)$ can be expressed as

$$\tilde{x}(n) = \frac{1}{N}\sum_{k=0}^{N-1} \tilde{X}(k) e^{j\frac{2\pi}{N}kn}, \quad n = 0, \pm 1, \ldots, \tag{5.2}$$

where $\{\tilde{X}(k), \quad k = 0, \pm 1, \ldots,\}$ are called the discrete Fourier series coefficients, which are given by

$$\tilde{X}(k) = \sum_{n=0}^{N-1} \tilde{x}(n) e^{-j\frac{2\pi}{N}nk}, \quad k = 0, \pm 1, \ldots, \tag{5.3}$$

Note that $\tilde{X}(k)$ is itself a (complex-valued) periodic sequence with fundamental period equal to N, that is,

$$\tilde{X}(k + N) = \tilde{X}(k) \tag{5.4}$$

The pair of equations (5.3) and (5.2), taken together, is called the *discrete Fourier series* representation of periodic sequences. Using $W_N \overset{\triangle}{=} e^{-j\frac{2\pi}{N}}$ to denote the complex exponential term, we express (5.3) and (5.2) as

$$\begin{aligned}
\tilde{X}(k) &\stackrel{\triangle}{=} \text{DFS}[\tilde{x}(n)] = \sum_{n=0}^{N-1} \tilde{x}(n) W_N^{nk} && : \text{Analysis or a DFS equation} \\
\tilde{x}(n) &\stackrel{\triangle}{=} \text{IDFS}[\tilde{X}(k)] = \frac{1}{N} \sum_{k=0}^{N-1} \tilde{X}(k) W_N^{-nk} && : \text{Synthesis or an inverse DFS equation}
\end{aligned} \tag{5.5}$$

□ **EXAMPLE 5.1** Find DFS representation of the periodic sequence

$$\tilde{x}(n) = \{\dots, 0, 1, 2, 3, \underset{\uparrow}{0}, 1, 2, 3, 0, 1, 2, 3, \dots\}$$

Solution The fundamental period of this sequence is $N = 4$. Hence $W_4 = e^{-j\frac{2\pi}{4}} = -j$. Now

$$\tilde{X}(k) = \sum_{n=0}^{3} \tilde{x}(n) W_4^{nk}, \quad k = 0, \pm 1, \pm 2, \dots$$

Hence

$$\tilde{X}(0) = \sum_{0}^{3} \tilde{x}(n) W_4^{0 \cdot n} = \sum_{0}^{3} \tilde{x}(n) = \tilde{x}(0) + \tilde{x}(1) + \tilde{x}(2) + \tilde{x}(3) = 6$$

Similarly,

$$\tilde{X}(1) = \sum_{0}^{3} \tilde{x}(n) W_4^{n} = \sum_{0}^{3} \tilde{x}(n)(-j)^n = (-2 + 2j)$$

$$\tilde{X}(2) = \sum_{0}^{3} \tilde{x}(n) W_4^{2n} = \sum_{0}^{3} \tilde{x}(n)(-j)^{2n} = 2$$

$$\tilde{X}(3) = \sum_{0}^{3} \tilde{x}(n) W_4^{3n} = \sum_{0}^{3} \tilde{x}(n)(-j)^{3n} = (-2 - 2j)$$

□

5.1.1 MATLAB IMPLEMENTATION

A careful look at (5.5) reveals that the DFS is a numerically computable representation. It can be implemented in many ways. To compute each sample $\tilde{X}(k)$, we can implement the summation as a `for...end` loop. To compute all DFS coefficients would require another `for...end` loop. This will result in a nested two `for...end` loop implementation. This is clearly inefficient in MATLAB. An efficient implementation in MATLAB would be to use a matrix-vector multiplication for each of the relations in (5.5). We have used this approach earlier in implementing a numerical approximation to the discrete-time Fourier transform. Let $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{X}}$ denote column vectors corresponding to the primary periods of sequences $\tilde{x}(n)$ and $\tilde{X}(k)$, respectively. Then (5.5) is given by

$$\begin{aligned}\tilde{\mathbf{X}} &= \mathbf{W}_N \tilde{\mathbf{x}} \\ \tilde{\mathbf{x}} &= \frac{1}{N}\mathbf{W}_N^* \tilde{\mathbf{X}}\end{aligned} \tag{5.6}$$

where the matrix $\mathbf{W}_N$ is given by

$$\mathbf{W}_N \triangleq \left[W_N^{kn}{}_{0\le k,n\le N-1} \right] = \begin{array}{cc} & n \longrightarrow \\ \begin{array}{c} k \\ \downarrow \end{array} & \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & W_N^1 & \cdots & W_N^{(N-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & W_N^{(N-1)} & \cdots & W_N^{(N-1)^2} \end{bmatrix} \end{array} \tag{5.7}$$

The matrix $\mathbf{W}_N$ is a square matrix and is called a *DFS matrix*. The following MATLAB function `dfs` implements this procedure.

```
function [Xk] = dfs(xn,N)
% Computes Discrete Fourier Series Coefficients
% ---------------------------------------------
% [Xk] = dfs(xn,N)
% Xk = DFS coeff. array over 0 <= k <= N-1
% xn = One period of periodic signal over 0 <= n <= N-1
%  N = Fundamental period of xn
%
n = [0:1:N-1];                 % row vector for n
k = [0:1:N-1];                 % row vecor for k
WN = exp(-j*2*pi/N);           % Wn factor
nk = n'*k;                     % creates a N by N matrix of nk values
WNnk = WN .^ nk;               % DFS matrix
Xk = xn * WNnk;                % row vector for DFS coefficients
```

The DFS in Example 5.1 can be computed using MATLAB as

```
>> xn = [0,1,2,3]; N = 4;  Xk = dfs(xn,N)
Xk =
   6.0000            -2.0000 + 2.0000i  -2.0000 - 0.0000i  -2.0000 - 2.0000i
```

The following `idfs` function implements the synthesis equation.

```
function [xn] = idfs(Xk,N)
% Computes Inverse Discrete Fourier Series
% ----------------------------------------
% [xn] = idfs(Xk,N)
% xn = One period of periodic signal over 0 <= n <= N-1
% Xk = DFS coeff. array over 0 <= k <= N-1
%  N = Fundamental period of Xk
%
n = [0:1:N-1];                 % row vector for n
k = [0:1:N-1];                 % row vecor for k
WN = exp(-j*2*pi/N);           % Wn factor
nk = n'*k;                     % creates a N by N matrix of nk values
WNnk = WN .^ (-nk);            % IDFS matrix
xn = (Xk * WNnk)/N;            % row vector for IDFS values
```

Caution: These functions are efficient approaches of implementing (5.5) in MATLAB. They are not computationally efficient, especially for large N. We will deal with this problem later in this chapter.

□ **EXAMPLE 5.2** A periodic "square wave" sequence is given by

$$\tilde{x}(n) = \begin{cases} 1, & mN \le n \le mN + L - 1 \\ 0, & mN + L \le n \le (m+1)N - 1 \end{cases}; \quad m = 0, \pm1, \pm2, \ldots$$

where N is the fundamental period and L/N is the duty cycle.

a. Determine an expression for $|\tilde{X}(k)|$ in terms of L and N.
b. Plot the magnitude $|\tilde{X}(k)|$ for $L = 5$, $N = 20$; $L = 5$, $N = 40$; $L = 5$, $N = 60$; and $L = 7$, $N = 60$.
c. Comment on the results.

Solution A plot of this sequence for $L = 5$ and $N = 20$ is shown in Figure 5.1.

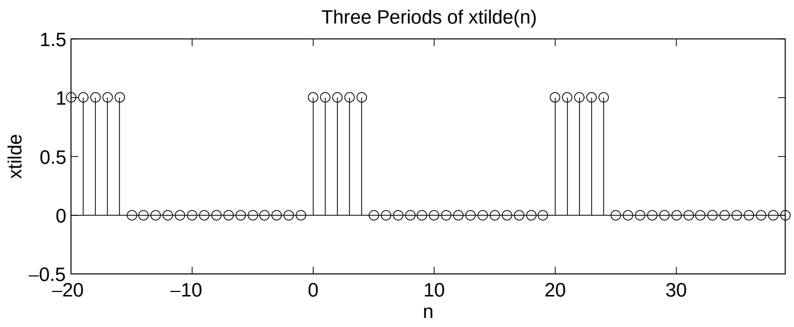


FIGURE 5.1 *Periodic square wave sequence*

a. By applying the analysis equation (5.3),

$$\begin{aligned}\tilde{X}(k) &= \sum_{n=0}^{N-1} \tilde{x}(n)e^{-j\frac{2\pi}{N}nk} = \sum_{n=0}^{L-1} e^{-j\frac{2\pi}{N}nk} = \sum_{n=0}^{L-1}\left(e^{-j\frac{2\pi}{N}k}\right)^n \\ &= \begin{cases} L, & k = 0, \pm N, \pm 2N, \ldots \\ \dfrac{1-e^{-j2\pi Lk/N}}{1-e^{-j2\pi k/N}}, & \text{otherwise}\end{cases}\end{aligned}$$

The last step follows from the sum of the geometric terms formula (2.7) in Chapter 2. The last expression can be simplified to

$$\begin{aligned}\frac{1-e^{-j2\pi Lk/N}}{1-e^{-j2\pi k/N}} &= \frac{e^{-j\pi Lk/N}}{e^{-j\pi k/N}} \frac{e^{j\pi Lk/N}-e^{-j\pi Lk/N}}{e^{j\pi k/N}-e^{-j\pi k/N}} \\ &= e^{-j\pi(L-1)k/N}\frac{\sin(\pi kL/N)}{\sin(\pi k/N)}\end{aligned}$$

or the magnitude of $\tilde{X}(k)$ is given by

$$\left|\tilde{X}(k)\right| = \begin{cases} L, & k = 0, \pm N, \pm 2N, \ldots \\ \left|\dfrac{\sin(\pi kL/N)}{\sin(\pi k/N)}\right|, & \text{otherwise}\end{cases}$$

b. The MATLAB script for $L = 5$ and $N = 20$:

```
>> L = 5; N = 20; k = [-N/2:N/2];            % Sq wave parameters
>> xn = [ones(1,L), zeros(1,N-L)];           % Sq wave x(n)
>> Xk = dfs(xn,N);                           % DFS
>> magXk = abs([Xk(N/2+1:N) Xk(1:N/2+1)]); % DFS magnitude
>> subplot(2,2,1); stem(k,magXk); axis([-N/2,N/2,-0.5,5.5])
>> xlabel('k'); ylabel('Xtilde(k)')
>> title('DFS of SQ. wave: L=5, N=20')
```

The plots for this and all other cases are shown in Figure 5.2. Note that since $\tilde{X}(k)$ is periodic, the plots are shown from $-N/2$ to $N/2$.

c. Several interesting observations can be made from plots in Figure 5.2. The envelopes of the DFS coefficients of square waves look like "sinc" functions. The amplitude at $k = 0$ is equal to L, while the zeros of the functions are at multiples of N/L, which is the reciprocal of the duty cycle. We will study these functions later in this chapter. □

5.1.2 RELATION TO THE z-TRANSFORM

Let $x(n)$ be a finite-duration sequence of duration N such that

$$x(n) = \begin{cases}\text{Nonzero}, & 0 \le n \le N-1 \\ 0, & \text{Elsewhere}\end{cases} \tag{5.8}$$

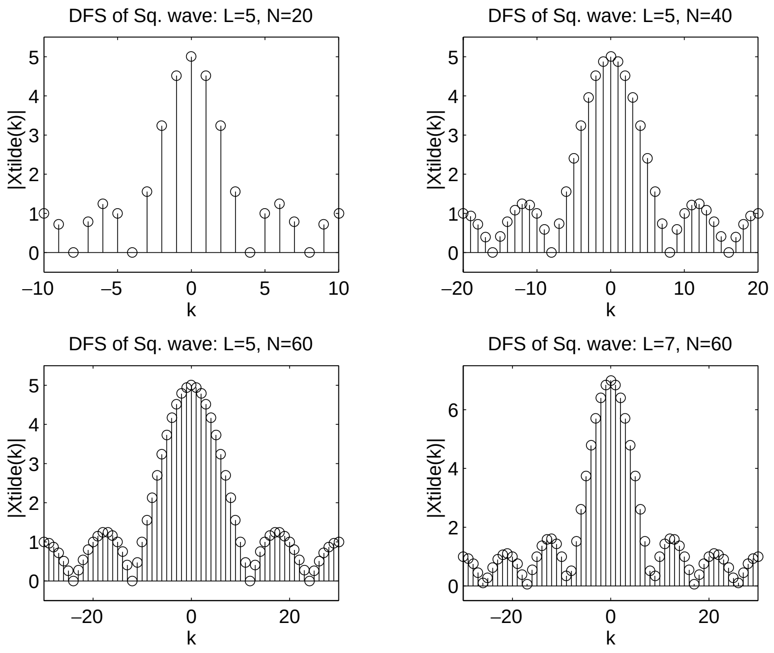


FIGURE 5.2 *The DFS plots of a periodic square wave for various L and N*

Then we can compute its z-transform:

$$X(z) = \sum_{n=0}^{N-1} x(n) z^{-n} \tag{5.9}$$

Now we construct a periodic sequence $\tilde{x}(n)$ by periodically repeating $x(n)$ with period N, that is,

$$x(n) = \begin{cases} \tilde{x}(n), \ 0 \le n \le N-1 \\ 0, \qquad \text{Elsewhere} \end{cases} \tag{5.10}$$

The DFS of $\tilde{x}(n)$ is given by

$$\tilde{X}(k) = \sum_{n=0}^{N-1} \tilde{x}(n) e^{-j\frac{2\pi}{N}nk} = \sum_{n=0}^{N-1} x(n) \left[e^{j\frac{2\pi}{N}k} \right]^{-n} \tag{5.11}$$

Comparing it with (5.9), we have

$$\tilde{X}(k) = X(z)\big|_{z=e^{j\frac{2\pi}{N}k}} \tag{5.12}$$

which means that the DFS $\tilde{X}(k)$ represents N evenly spaced samples of the z-transform $X(z)$ around the unit circle.

5.1.3 RELATION TO THE DTFT

Since $x(n)$ in (5.8) is of finite duration of length N, it is also absolutely summable. Hence its DTFT exists and is given by

$$X(e^{j\omega}) = \sum_{n=0}^{N-1} x(n)e^{-j\omega n} = \sum_{n=0}^{N-1} \tilde{x}(n)e^{-j\omega n} \tag{5.13}$$

Comparing (5.13) with (5.11), we have

$$\tilde{X}(k) = X(e^{j\omega})\big|_{\omega=\frac{2\pi}{N}k} \tag{5.14}$$

Let

$$\omega_1 \triangleq \frac{2\pi}{N} \qquad \text{and} \qquad \omega_k \triangleq \frac{2\pi}{N}k = k\omega_1$$

Then the DFS $X(k) = X(e^{j\omega_k}) = X(e^{jk\omega_1})$, which means that the DFS is obtained by *evenly sampling* the DTFT at $\omega_1 = \frac{2\pi}{N}$ intervals. From (5.12) and (5.14) we observe that the DFS representation gives us a sampling mechanism in the frequency domain that, in principle, is similar to sampling in the time domain. The interval $\omega_1 = \frac{2\pi}{N}$ is the *sampling interval* in the frequency domain. It is also called the *frequency resolution* because it tells us how close the frequency samples (or measurements) are.

□ **EXAMPLE 5.3** Let $x(n) = \{\underset{\uparrow}{0}, 1, 2, 3\}$.

a. Compute its discrete-time Fourier transform $X(e^{j\omega})$.
b. Sample $X(e^{j\omega})$ at $k\omega_1 = \frac{2\pi}{4}k, \quad k = 0, 1, 2, 3$ and show that it is equal to $\tilde{X}(k)$ in Example 5.1.

Solution The sequence $x(n)$ is not periodic but is of finite duration.

a. The discrete-time Fourier transform is given by

$$X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x(n)e^{-j\omega n} = e^{-j\omega} + 2e^{-j2\omega} + 3e^{-j3\omega}$$

b. Sampling at $k\omega_1 = \frac{2\pi}{4}k, \quad k = 0, 1, 2, 3$, we obtain

$$\begin{aligned}
X(e^{j0}) &= 1 + 2 + 3 = 6 = \tilde{X}(0) \\
X(e^{j2\pi/4}) &= e^{-j2\pi/4} + 2e^{-j4\pi/4} + 3e^{-j6\pi/4} = -2 + 2j = \tilde{X}(1) \\
X(e^{j4\pi/4}) &= e^{-j4\pi/4} + 2e^{-j8\pi/4} + 3e^{-j12\pi/4} = 2 = \tilde{X}(2) \\
X(e^{j6\pi/4}) &= e^{-j6\pi/4} + 2e^{-j12\pi/4} + 3e^{-j18\pi/4} = -2 - 2j = \tilde{X}(3)
\end{aligned}$$

as expected. □

5.2 SAMPLING AND RECONSTRUCTION IN THE z-DOMAIN

Let $x(n)$ be an arbitrary absolutely summable sequence, which may be of infinite duration. Its z-transform is given by

$$X(z) = \sum_{m=-\infty}^{\infty} x(m) z^{-m}$$

and we assume that the ROC of $X(z)$ includes the unit circle. We sample $X(z)$ on the unit circle at equispaced points separated in angle by $\omega_1 = 2\pi/N$ and call it a DFS sequence,

$$\begin{aligned}\tilde{X}(k) &\overset{\triangle}{=} X(z)\big|_{z=e^{j\frac{2\pi}{N}k}}, \qquad k = 0, \pm 1, \pm 2, \ldots \\ &= \sum_{m=-\infty}^{\infty} x(m) e^{-j\frac{2\pi}{N}km} = \sum_{m=-\infty}^{\infty} x(m) W_N^{km} \end{aligned} \tag{5.15}$$

which is periodic with period N. Finally, we compute the IDFS of $\tilde{X}(k)$,

$$\tilde{x}(n) = \text{IDFS}\big[\tilde{X}(k)\big]$$

which is also periodic with period N. Clearly, there must be a relationship between the arbitrary $x(n)$ and the periodic $\tilde{x}(n)$. This is an important issue. In order to compute the inverse DTFT or the inverse z-transform numerically, we must deal with a finite number of samples of $X(z)$ around the unit circle. Therefore we must know the effect of such sampling on the time-domain sequence. This relationship is easy to obtain.

$$\begin{aligned}\tilde{x}(n) &= \frac{1}{N} \sum_{k=0}^{N-1} \tilde{X}(k) W_N^{-kn} && \text{[from (5.2)]} \\ &= \frac{1}{N} \sum_{k=0}^{N-1} \left\{ \sum_{m=-\infty}^{\infty} x(m) W_N^{km} \right\} W_N^{-kn} && \text{[from (5.15)]}\end{aligned}$$

or

$$\begin{aligned}\tilde{x}(n) &= \sum_{m=-\infty}^{\infty} x(m) \underbrace{\frac{1}{N} \sum_{0}^{N-1} W_N^{-k(n-m)}}_{=\begin{cases}1, & n-m=rN \\ 0, & \text{elsewhere}\end{cases}} = \sum_{m=-\infty}^{\infty} x(m) \sum_{r=-\infty}^{\infty} \delta(n-m-rN) \\ &= \sum_{r=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} x(m) \delta(n-m-rN)\end{aligned}$$

or

$$\tilde{x}(n) = \sum_{r=-\infty}^{\infty} x(n-rN) = \cdots + x(n+N) + x(n) + x(n-N) + \cdots \quad \textbf{(5.16)}$$

which means that when we sample $X(z)$ on the unit circle, we obtain a periodic sequence in the time domain. This sequence is a linear combination of the original $x(n)$ and its infinite replicas, each shifted by multiples of $\pm N$. This is illustrated in Example 5.5. From (5.16), we observe that if $x(n) = 0$ for $n < 0$ and $n \geq N$, then there will be no overlap or aliasing in the time domain. Hence we should be able to recognize and recover $x(n)$ from $\tilde{x}(n)$, that is,

$$x(n) = \tilde{x}(n) \text{ for } 0 \leq n \leq (N-1)$$

or

$$x(n) = \tilde{x}(n)\mathcal{R}_N(n) = \begin{cases} \tilde{x}(n), & 0 \leq n \leq N-1 \\ 0, & \text{else} \end{cases}$$

where $\mathcal{R}_N(n)$ is called a *rectangular window* of length N. Therefore we have the following theorem.

■ **THEOREM 1** ***Frequency Sampling***

If $x(n)$ is time-limited (i.e., of finite duration) to $[0, N-1]$, then N samples of $X(z)$ on the unit circle determine $X(z)$ for all z.

□ **EXAMPLE 5.4** Let $x_1(n) = \{\underset{\uparrow}{6}, 5, 4, 3, 2, 1\}$. Its DTFT $X_1(e^{j\omega})$ is sampled at

$$\omega_k = \frac{2\pi k}{4}, \quad k = 0, \pm 1, \pm 2, \pm 3, \ldots$$

to obtain a DFS sequence $\tilde{X}_2(k)$. Determine the sequence $\tilde{x}_2(n)$, which is the inverse DFS of $\tilde{X}_2(k)$.

Solution Without computing the DTFT, the DFS, or the inverse DFS, we can evaluate $\tilde{x}_2(n)$ by using the aliasing formula (5.16).

$$\tilde{x}_2(n) = \sum_{r=-\infty}^{\infty} x_1(n-4r)$$

Thus $x(4)$ is aliased into $x(0)$, and $x(5)$ is aliased into $x(1)$. Hence

$$\tilde{x}_2(n) = \{\ldots, 8, 6, 4, 3, \underset{\uparrow}{8}, 6, 4, 3, 8, 6, 4, 3, \ldots\}$$ □

□ **EXAMPLE 5.5** Let $x(n) = (0.7)^n\, u(n)$. Sample its z-transform on the unit circle with $N = 5$, 10, 20, 50 and study its effect in the time domain.

Solution From Table 4.1 the z-transform of $x(n)$ is

$$X(z) = \frac{1}{1-0.7z^{-1}} = \frac{z}{z-0.7}, \quad |z| > 0.7$$

We can now use MATLAB to implement the sampling operation

$$\tilde{X}(k) = X(z)|_{z=e^{j2\pi k/N}}, \quad k = 0, \pm 1, \pm 2, \ldots$$

and the inverse DFS computation to determine the corresponding time-domain sequence. The MATLAB script for $N = 5$ is as follows.

```
>> N = 5; k = 0:1:N-1;                                    % sample index
>> wk = 2*pi*k/N; zk = exp(j*wk);                         % samples of z
>> Xk = (zk)./(zk-0.7);                                   % DFS as samples of X(z)
>> xn = real(idfs(Xk,N));                                 % IDFS
>> xtilde = xn'* ones(1,8); xtilde = (xtilde(:))'; % Periodic sequence
>> subplot(2,2,1); stem(0:39,xtilde);axis([0,40,-0.1,1.5])
>> xlabel('n'); ylabel('xtilde(n)'); title('N=5')
```

The plots in Figure 5.3 clearly demonstrate the aliasing in the time domain, especially for $N = 5$ and $N = 10$. For large values of N the tail end of $x(n)$

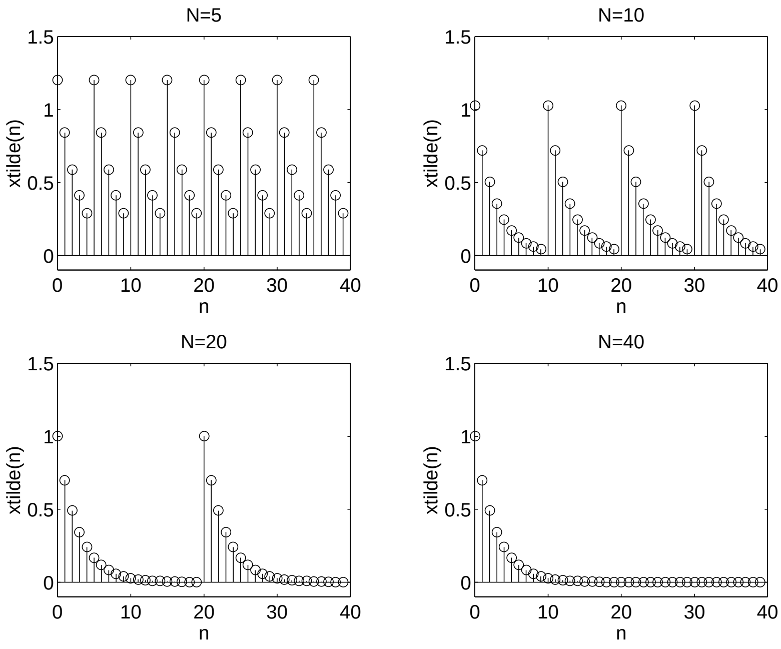


FIGURE 5.3 *Plots in Example 5.5*

is sufficiently small to result in any appreciable amount of aliasing in practice. Such information is useful in effectively truncating an infinite-duration sequence prior to taking its transform. □

5.2.1 THE z-TRANSFORM RECONSTRUCTION FORMULA

Let $x(n)$ be time-limited to $[0, N-1]$. Then from Theorem 1 we should be able to recover the z-transform $X(z)$ using its samples $\tilde{X}(k)$. This is given by

$$\begin{aligned} X(z) &= \mathcal{Z}\left[x(n)\right] = \mathcal{Z}\left[\tilde{x}(n)\mathcal{R}_N(n)\right] \\ &= \mathcal{Z}[\,\text{IDFS}\{\quad \underbrace{\tilde{X}(k)}_{\text{Samples of } X(z)} \quad\}\mathcal{R}_N(n)] \end{aligned}$$

This approach results in the z-domain reconstruction formula.

$$\begin{aligned} X(z) &= \sum_{0}^{N-1} x(n)z^{-n} = \sum_{0}^{N-1} \tilde{x}(n)z^{-n} \\ &= \sum_{0}^{N-1} \left\{\frac{1}{N}\sum_{0}^{N-1} \tilde{X}(k)W_N^{-kn}\right\} z^{-n} \\ &= \frac{1}{N}\sum_{k=0}^{N-1} \tilde{X}(k)\left\{\sum_{0}^{N-1} W_N^{-kn}z^{-n}\right\} \\ &= \frac{1}{N}\sum_{k=0}^{N-1} \tilde{X}(k)\left\{\sum_{0}^{N-1} \left(W_N^{-k}z^{-1}\right)^n\right\} \\ &= \frac{1}{N}\sum_{k=0}^{N-1} \tilde{X}(k)\left\{\frac{1-W_N^{-kN}z^{-N}}{1-W_N^{-k}z^{-1}}\right\} \end{aligned}$$

Since $W_N^{-kN} = 1$, we have

$$X(z) = \frac{1-z^{-N}}{N}\sum_{k=0}^{N-1} \frac{\tilde{X}(k)}{1-W_N^{-k}z^{-1}} \tag{5.17}$$

5.2.2 THE DTFT INTERPOLATION FORMULA

The reconstruction formula (5.17) can be specialized for the discrete-time Fourier transform by evaluating it on the unit circle $z = e^{j\omega}$. Then

$$\begin{aligned} X(e^{j\omega}) &= \frac{1-e^{-j\omega N}}{N}\sum_{k=0}^{N-1} \frac{\tilde{X}(k)}{1-e^{j2\pi k/N}e^{-j\omega}} \\ &= \sum_{k=0}^{N-1} \tilde{X}(k)\frac{1-e^{-j\omega N}}{N\left\{1-e^{j2\pi k/N}e^{-j\omega}\right\}} \end{aligned}$$

Consider

$$\frac{1-e^{-j\omega N}}{N\left\{1-e^{j2\pi k/N}e^{-j\omega}\right\}} = \frac{1-e^{-j(\omega-\frac{2\pi k}{N})N}}{N\left\{1-e^{-j(\omega-\frac{2\pi k}{N})}\right\}}$$
$$= \frac{e^{-j\frac{N}{2}(\omega-\frac{2\pi k}{N})}}{e^{-\frac{1}{2}j(\omega-\frac{2\pi k}{N})}}\left\{\frac{\sin\left[(\omega-\frac{2\pi k}{N})\frac{N}{2}\right]}{N\sin\left[(\omega-\frac{2\pi k}{N})\frac{1}{2}\right]}\right\}$$

Let

$$\Phi(\omega) \triangleq \frac{\sin(\frac{\omega N}{2})}{N\sin(\frac{\omega}{2})}e^{-j\omega(\frac{N-1}{2})} : \text{an interpolating function} \tag{5.18}$$

Then

$$X(e^{j\omega}) = \sum_{k=0}^{N-1}\tilde{X}(k)\Phi\left(\omega-\frac{2\pi k}{N}\right) \tag{5.19}$$

This is the DTFT interpolation formula to reconstruct $X(e^{j\omega})$ from its samples $\tilde{X}(k)$. Since $\Phi(0)=1$, we have that $X(e^{j2\pi k/N})=\tilde{X}(k)$, which means that the interpolation is exact at sampling points. Recall the time-domain interpolation formula (3.33) for analog signals:

$$x_a(t) = \sum_{n=-\infty}^{\infty} x(n)\,\text{sinc}\left[F_s(t-nT_s)\right] \tag{5.20}$$

The DTFT interpolating formula (5.19) looks similar.

However, there are some differences. First, the time-domain formula (5.20) reconstructs an arbitrary *nonperiodic* analog signal, while the frequency-domain formula (5.19) gives us a periodic waveform. Second, in (5.19) we use a $\frac{\sin(Nx)}{N\sin x}$ interpolation function instead of our more familiar $\frac{\sin x}{x}$ (sinc) function. The $\Phi(\omega)$ function is a periodic function and hence is known as a *periodic-sinc* function. It is also known as the Dirichlet function. This is the function we observed in Example 5.2.

5.2.3 MATLAB IMPLEMENTATION

The interpolation formula (5.19) suffers the same fate as that of (5.20) while trying to implement it in practice. One has to generate several interpolating functions (5.18) and perform their linear combinations to obtain the discrete-time Fourier transform $X(e^{j\omega})$ from its computed samples $\tilde{X}(k)$. Furthermore, in MATLAB we have to evaluate (5.19) on a finer grid over $0 \le \omega \le 2\pi$. This is clearly an inefficient approach. Another approach is to use the cubic spline interpolation function as an efficient approximation to (5.19). This is what we did to implement (5.20) in Chapter 3. However, there is an alternate and efficient approach based on the DFT, which we will study in the next section.

5.3 THE DISCRETE FOURIER TRANSFORM

The discrete Fourier series provides a mechanism for numerically computing the discrete-time Fourier transform. It also alerted us to a potential problem of aliasing in the time domain. Mathematics dictates that the sampling of the discrete-time Fourier transform result in a periodic sequence $\tilde{x}(n)$. But most of the signals in practice are not periodic. They are likely to be of finite duration. How can we develop a numerically computable Fourier representation for such signals? Theoretically, we can take care of this problem by defining a periodic signal whose primary shape is that of the finite-duration signal and then using the DFS on this periodic signal. Practically, we define a new transform called the *discrete Fourier transform* (DFT), which is the primary period of the DFS. This DFT is the ultimate numerically computable Fourier transform for arbitrary finite-duration sequences.

First we define a finite-duration sequence $x(n)$ that has N samples over $0 \le n \le N-1$ as an N*-point sequence*. Let $\tilde{x}(n)$ be a periodic signal of period N, created using the N-point sequence $x(n)$; that is, from (5.19)

$$\tilde{x}(n) = \sum_{r=-\infty}^{\infty} x(n - rN)$$

This is a somewhat cumbersome representation. Using the modulo-N operation on the argument we can simplify it to

$$\tilde{x}(n) = x(n \bmod N) \tag{5.21}$$

A simple way to interpret this operation is the following: if the argument n is between 0 and $N-1$, then leave it as it is; otherwise add or subtract multiples of N from n until the result is between 0 and $N-1$. Note carefully that (5.21) is valid only if the length of $x(n)$ is N or less. Furthermore, we use the following convenient notation to denote the modulo-N operation.

$$x((n))_N \overset{\triangle}{=} x(n \bmod N) \tag{5.22}$$

Then the compact relationships between $x(n)$ and $\tilde{x}(n)$ are

$$\begin{aligned} \tilde{x}(n) &= x((n))_N \qquad &\text{(Periodic extension)} \\ x(n) &= \tilde{x}(n)\mathcal{R}_N(n) \qquad &\text{(Window operation)} \end{aligned} \tag{5.23}$$

The `rem(n,N)` function in MATLAB determines the remainder after dividing n by N. This function can be used to implement our modulo-N

operation when $n \geq 0$. When $n < 0$, we need to modify the result to obtain correct values. This is shown below in the m=mod(n,N) function.

```
function m = mod(n,N)
% Computes m = (n mod N) index
% ----------------------------
% m = mod(n,N)
m = rem(n,N);  m = m+N;  m = rem(m,N);
```

In this function n can be any integer array, and the array m contains the corresponding modulo-N values.

From the frequency sampling theorem we conclude that N equispaced samples of the discrete-time Fourier transform $X(e^{j\omega})$ of the N-point sequence $x(n)$ can uniquely reconstruct $X(e^{j\omega})$. These N samples around the unit circle are called the discrete Fourier transform coefficients. Let $\tilde{X}(k) = \text{DFS}\,\tilde{x}(n)$, which is a periodic (and hence of infinite duration) sequence. Its primary interval then is the discrete Fourier transform, which is of finite duration. These notions are made clear in the following definitions. The Discrete Fourier Transform of an N-point sequence is given by

$$X(k) \stackrel{\triangle}{=} \text{DFT}\,[x(n)] = \begin{cases} \tilde{X}(k), & 0 \leq k \leq N-1 \\ 0, & \text{elsewhere} \end{cases} = \tilde{X}(k)\mathcal{R}_N(k)$$

or

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk}, \quad 0 \leq k \leq N-1 \tag{5.24}$$

Note that the DFT $X(k)$ is also an N-point sequence, that is, it is not defined outside of $0 \leq k \leq N-1$. From (5.23) $\tilde{X}(k) = X((k))_N$; that is, outside the $0 \leq k \leq N-1$ interval only the DFS $\tilde{X}(k)$ is defined, which of course is the periodic extension of $X(k)$. Finally, $X(k) = \tilde{X}(k)\mathcal{R}_N(k)$ means that the DFT $X(k)$ is the primary interval of $\tilde{X}(k)$.

The inverse discrete Fourier transform of an N-point DFT $X(k)$ is given by

$$x(n) \stackrel{\triangle}{=} \text{IDFT}\,[X(k)] = \tilde{x}(n)\mathcal{R}_N(n)$$

or

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) W_N^{-kn}, \quad 0 \leq n \leq N-1 \tag{5.25}$$

Once again $x(n)$ is not defined outside $0 \leq n \leq N-1$. The extension of $x(n)$ outside this range is $\tilde{x}(n)$.

5.3.1 MATLAB IMPLEMENTATION

It is clear from the discussions at the top of this section that the DFS is practically equivalent to the DFT when $0 \le n \le N-1$. Therefore the implementation of the DFT can be done in a similar fashion. If $x(n)$ and $X(k)$ are arranged as column vectors $\mathbf{x}$ and $\mathbf{X}$, respectively, then from (5.24) and (5.25) we have

$$\begin{aligned} \mathbf{X} &= \mathbf{W}_N \mathbf{x} \\ \mathbf{x} &= \frac{1}{N}\mathbf{W}_N^* \mathbf{X} \end{aligned} \tag{5.26}$$

where $\mathbf{W}_N$ is the matrix defined in (5.7) and will now be called a *DFT matrix*. Hence the earlier `dfs` and `idfs` MATLAB functions can be renamed as the `dft` and `idft` functions to implement the discrete Fourier transform computations.

```
function [Xk] = dft(xn,N)
% Computes Discrete Fourier Transform
% -----------------------------------
% [Xk] = dft(xn,N)
% Xk = DFT coeff. array over 0 <= k <= N-1
% xn = N-point finite-duration sequence
%  N = Length of DFT
%
n = [0:1:N-1];               % row vector for n
k = [0:1:N-1];               % row vecor for k
WN = exp(-j*2*pi/N);         % Wn factor
nk = n'*k;                   % creates a N by N matrix of nk values
WNnk = WN .^ nk;             % DFT matrix
Xk = xn * WNnk;              % row vector for DFT coefficients

function [xn] = idft(Xk,N)
% Computes Inverse Discrete Transform
% -----------------------------------
% [xn] = idft(Xk,N)
% xn = N-point sequence over 0 <= n <= N-1
% Xk = DFT coeff. array over 0 <= k <= N-1
%  N = length of DFT
%
n = [0:1:N-1];               % row vector for n
k = [0:1:N-1];               % row vecor for k
WN = exp(-j*2*pi/N);         % Wn factor
nk = n'*k;                   % creates a N by N matrix of nk values
WNnk = WN .^ (-nk);          % IDFT matrix
xn = (Xk * WNnk)/N;          % row vector for IDFT values
```

□ **EXAMPLE 5.6** Let $x(n)$ be a 4-point sequence:

$$x(n) = \begin{cases} 1, & 0 \le n \le 3 \\ 0, & \text{otherwise} \end{cases}$$

a. Compute the discrete-time Fourier transform $X(e^{j\omega})$ and plot its magnitude and phase.
b. Compute the 4-point DFT of $x(n)$.

Solution **a.** The discrete-time Fourier transform is given by

$$\begin{aligned} X(e^{j\omega}) &= \sum_{0}^{3} x(n) e^{-j\omega n} = 1 + e^{-j\omega} + e^{-j2\omega} + e^{-j3\omega} \\ &= \frac{1 - e^{-j4\omega}}{1 - e^{-j\omega}} = \frac{\sin(2\omega)}{\sin(\omega/2)} e^{-j3\omega/2} \end{aligned}$$

Hence

$$\left| X(e^{j\omega}) \right| = \left| \frac{\sin(2\omega)}{\sin(\omega/2)} \right|$$

and

$$\angle X(e^{j\omega}) = \begin{cases} -\dfrac{3\omega}{2}, & \text{when } \dfrac{\sin(2\omega)}{\sin(\omega/2)} > 0 \\ -\dfrac{3\omega}{2} \pm \pi, & \text{when } \dfrac{\sin(2\omega)}{\sin(\omega/2)} < 0 \end{cases}$$

The plots are shown in Figure 5.4.

b. Let us denote the 4-point DFT by $X_4(k)$. Then

$$X_4(k) = \sum_{n=0}^{3} x(n) W_4^{nk}; \quad k = 0, 1, 2, 3; \; W_4 = e^{-j2\pi/4} = -j$$

These calculations are similar to those in Example 5.1. We can also use MATLAB to compute this DFT.

```
>> x = [1,1,1,1]; N = 4;  X = dft(x,N);
>> magX = abs(X), phaX = angle(X)*180/pi
magX =
    4.0000    0.0000    0.0000    0.0000
phaX =
         0 -134.9810  -90.0000  -44.9979
```

Hence

$$X_4(k) = \{\underset{\uparrow}{4}, 0, 0, 0\}$$

Note that when the magnitude sample is zero, the corresponding angle is not zero. This is due to a particular algorithm used by MATLAB to compute the

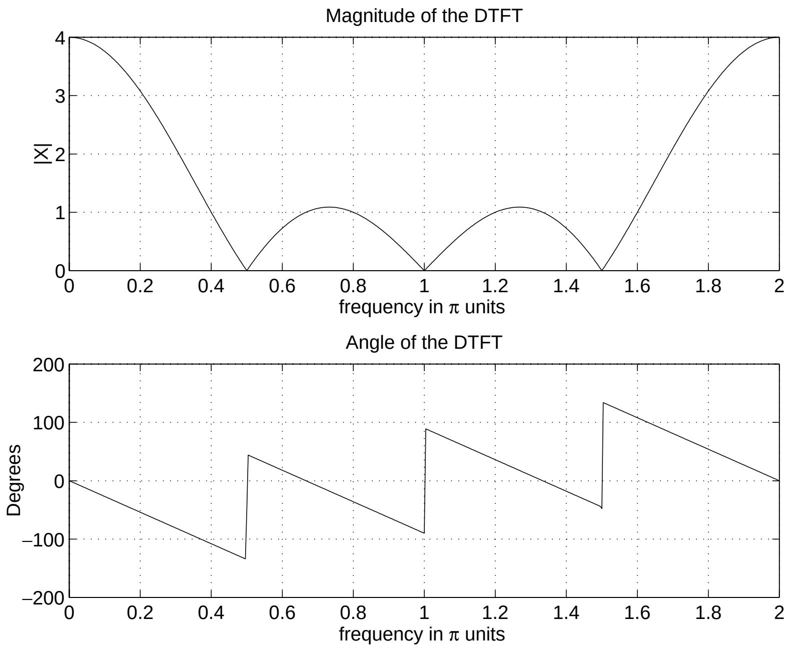


FIGURE 5.4 *The DTFT plots in Example 5.6*

angle part. Generally these angles should be ignored. The plot of DFT values is shown in Figure 5.5. The plot of $X(e^{j\omega})$ is also shown as a dashed line for comparison. From the plot in Figure 5.5 we observe that X_4 correctly gives 4 samples of $X(e^{j\omega})$, but it has only one nonzero sample. Is this surprising? By looking at the 4-point $x(n)$, which contains all 1's, one must conclude that its periodic extension is

$$\tilde{x}(n) = 1, \ \forall n$$

which is a constant (or a DC) signal. This is what is predicted by the DFT $X_4(k)$, which has a nonzero sample at $k = 0$ (or $\omega = 0$) and has no values at other frequencies. □

□ **EXAMPLE 5.7** How can we obtain other samples of the DTFT $X(e^{j\omega})$?

Solution It is clear that we should sample at dense (or finer) frequencies; that is, we should increase N. Suppose we take twice the number of points, or $N = 8$ instead of 4. This we can achieve by treating $x(n)$ as an 8-point sequence by *appending* 4 zeros.

$$x(n) = \{\underset{\uparrow}{1}, 1, 1, 1, 0, 0, 0, 0\}$$

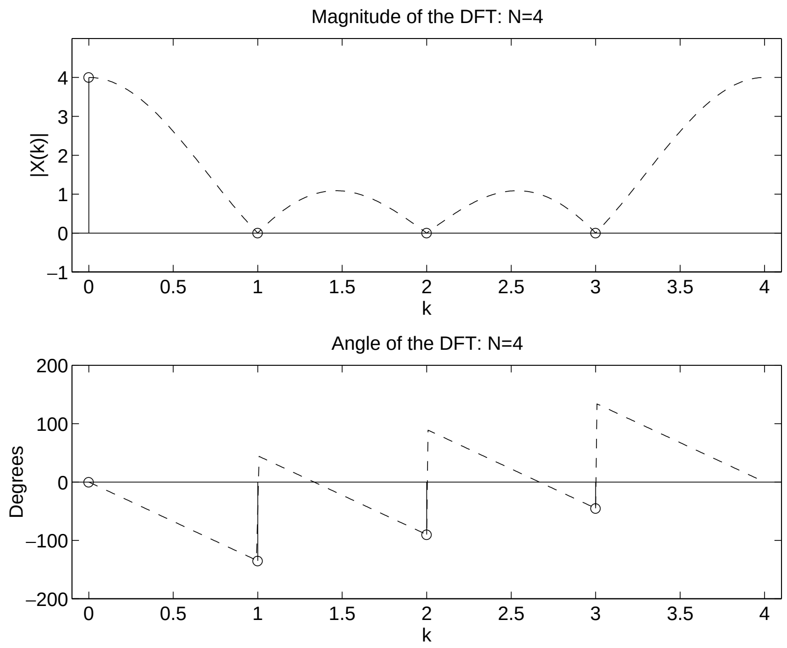


FIGURE 5.5 *The DFT plots of Example 5.6*

This is a very important operation called a *zero-padding operation.* This operation is necessary in practice to obtain a *dense spectrum* of signals as we shall see. Let $X_8(k)$ be an 8-point DFT, then

$$X_8(k) = \sum_{n=0}^{7} x(n) W_8^{nk}; \quad k = 0, 1, \ldots, 7; \; W_8 = e^{-j\pi/4}$$

In this case the frequency resolution is $\omega_1 = 2\pi/8 = \pi/4$.

MATLAB script:

```
>> x = [1,1,1,1, zeros(1,4)]; N = 8;  X = dft(x,N);
>> magX = abs(X), phaX = angle(X)*180/pi
magX =
    4.0000    2.6131    0.0000    1.0824    0.0000    1.0824    0.0000    2.6131
phaX =
         0  -67.5000 -134.9810  -22.5000  -90.0000   22.5000  -44.9979   67.5000
```

Hence

$$X_8(k) = \{\underset{\uparrow}{4}, \; 2.6131e^{-j67.5^\circ}, \; 0, \; 1.0824e^{-j22.5^\circ}, \; 0, \; 1.0824e^{j22.5^\circ}, \; 0, \; 2.6131e^{j67.5^\circ}\}$$

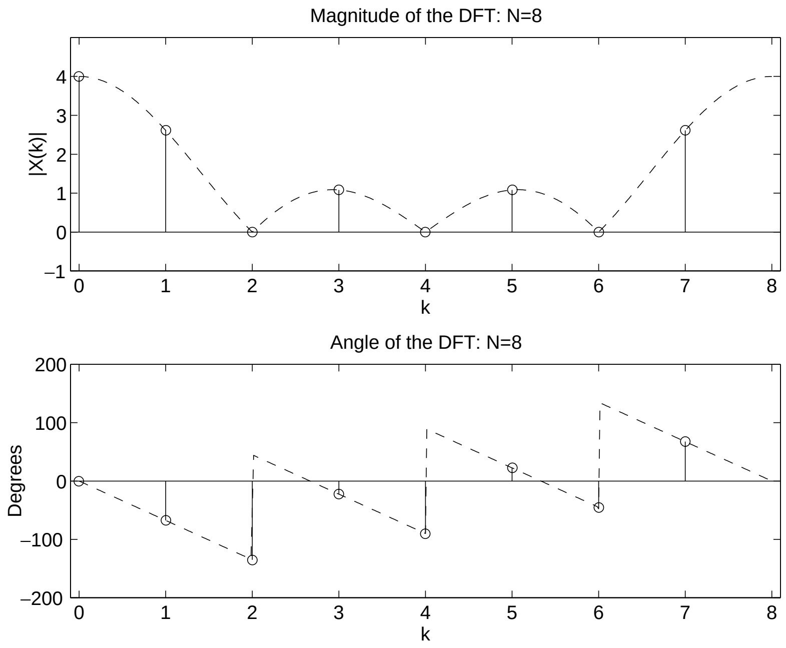


FIGURE 5.6 *The DFT plots of Example 5.7: $N = 8$*

which is shown in Figure 5.6. Continuing further, if we treat $x(n)$ as a 16-point sequence by padding 12 zeros, such that

$$x(n) = \{\underset{\uparrow}{1}, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\}$$

then the frequency resolution is $\omega_1 = 2\pi/16 = \pi/8$ and $W_{16} = e^{-j\pi/8}$. Therefore we get a more dense spectrum with spectral samples separated by $\pi/8$. The sketch of $X_{16}(k)$ is shown in Figure 5.7.

It should be clear then that if we obtain many more spectral samples by choosing a large N value then the resulting DFT samples will be very close to each other and we will obtain plot values similar to those in Figure 5.4. However, the displayed stem-plots will be dense. In this situation a better approach to display samples is to either show them using dots or join the sample values using the `plot` command (that is, using the FOH studied in Chapter 3). Figure 5.8 shows the magnitude and phase of the 128-point DFT $x_{128}(k)$ obtained by padding 120 zeros. The DFT magnitude plot overlaps the DTFT magnitude plot shown as dotted-line while the phase plot shows discrepancy at discontinuities due to finite N value, which should be expected. □

Comments: Based on the last two examples there are several comments that we can make.

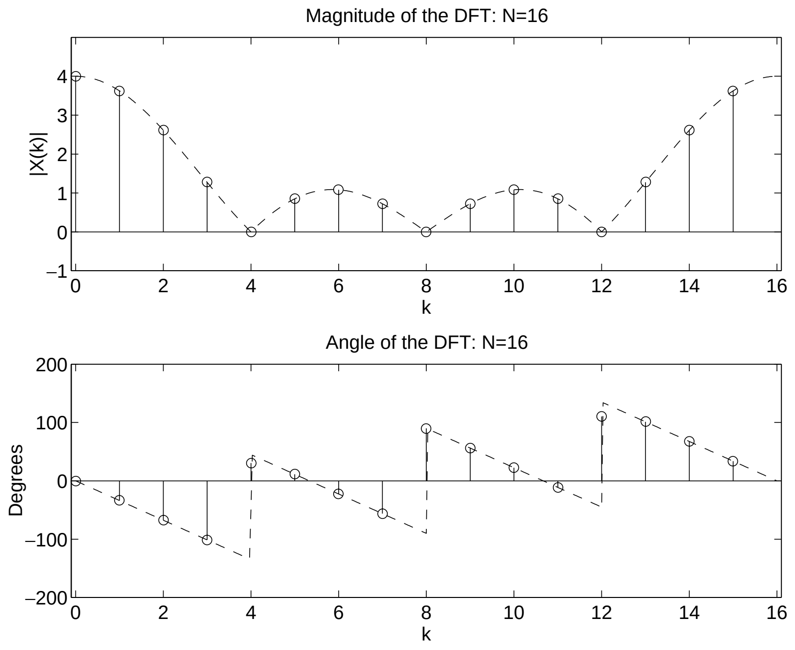


FIGURE 5.7 *The DFT plots of Example 5.7:* $N = 16$

1. Zero-padding is an operation in which more zeros are appended to the original sequence. The resulting longer DFT provides closely spaced samples of the discrete-time Fourier transform of the original sequence. In MATLAB zero-padding is implemented using the `zeros` function.
2. In Example 5.6 all we needed to accurately plot the discrete-time Fourier transform $X(e^{j\omega})$ of $x(n)$ was $X_4(k)$, the 4-point DFT. This is because $x(n)$ had only 4 nonzero samples, so we could have used the interpolation formula (5.19) on $X_4(k)$ to obtain $X(e^{j\omega})$. However, in practice, it is easier to obtain $X_8(k)$ and $X_{16}(k)$, and so on, to *fill in* the values of $X(e^{j\omega})$ rather than using the interpolation formula. This approach can be made even more efficient using fast Fourier transform algorithms to compute the DFT.
3. The zero-padding gives us a *high-density spectrum* and provides a better displayed version for plotting. But it does not give us a *high-resolution spectrum* because no new information is added to the signal; only additional zeros are added in the data.
4. To get a high-resolution spectrum, one has to obtain more data from the experiment or observations (see Example 5.8 below). There are also other advanced methods that use additional side information or nonlinear techniques.

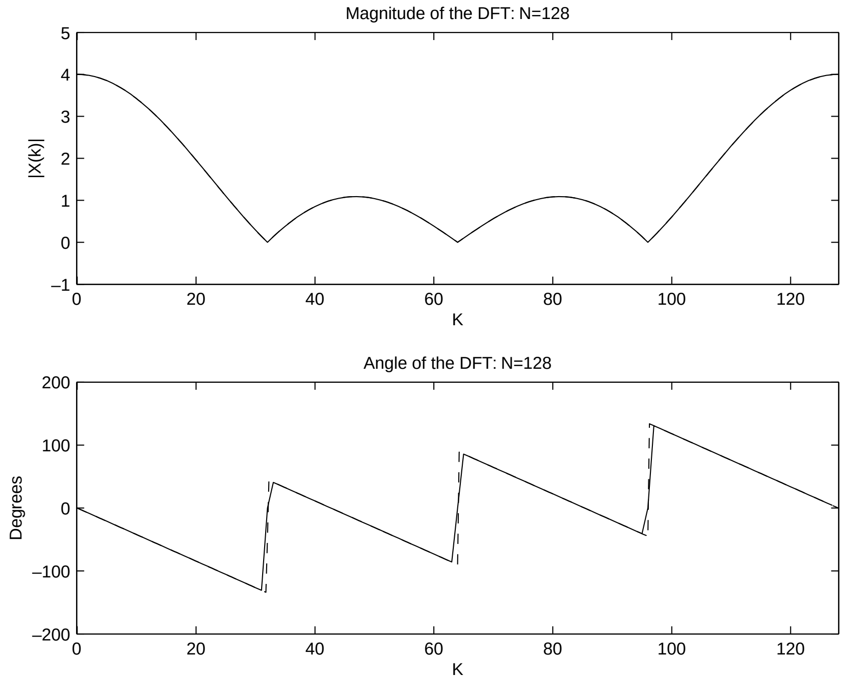


FIGURE 5.8 *The DFT plots of Example 5.7 for $N = 128$ are shown as line plots*

□ **EXAMPLE 5.8** To illustrate the difference between the high-density spectrum and the high-resolution spectrum, consider the sequence

$$x(n) = \cos(0.48\pi n) + \cos(0.52\pi n)$$

We want to determine its spectrum based on the finite number of samples.

a. Determine and plot the discrete-time Fourier transform of $x(n)$, $0 \le n \le 10$.
b. Determine and plot the discrete-time Fourier transform of $x(n)$, $0 \le n \le 100$.

Solution We could determine analytically the discrete-time Fourier transform in each case, but MATLAB is a good vehicle to study these problems.

a. We can first determine the 10-point DFT of $x(n)$ to obtain an estimate of its discrete-time Fourier transform.
MATLAB Script:

```
>> n = [0:1:99]; x = cos(0.48*pi*n)+cos(0.52*pi*n);
>> n1 = [0:1:9] ;y1 = x(1:1:10);
>> subplot(2,1,1) ;stem(n1,y1); title('signal x(n), 0 <= n <= 9');xlabel('n')
```

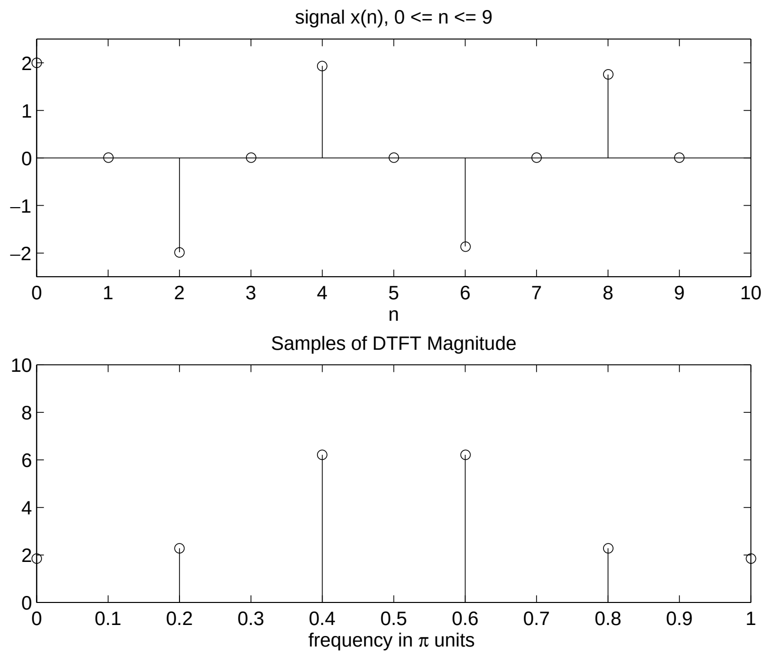


FIGURE 5.9 *Signal and its spectrum in Example 5.8a: $N = 10$*

```
>> Y1 = dft(y1,10); magY1 = abs(Y1(1:1:6));
>> k1 = 0:1:5 ;w1 = 2*pi/10*k1;
>> subplot(2,1,2);stem(w1/pi,magY1);title('Samples of DTFT Magnitude');
>> xlabel('frequency in pi units')
```

The plots in Figure 5.9 show there aren't enough samples to draw any conclusions. Therefore we will pad 90 zeros to obtain a dense spectrum. As explained in Example 5.7, this spectrum is plotted using the `plot` command.

MATLAB Script:

```
>> n2 = [0:1:99]; y2 = [x(1:1:10) zeros(1,90)];
>> subplot(2,1,1) ;stem(n2,y2) ;title('signal x(n), 0 <= n <= 9 + 90 zeros');
>> xlabel('n')
>> Y2 =dft(y2,100); magY2 = abs(Y2(1:1:51));
>> k2 = 0:1:50; w2 = 2*pi/100*k2;
>> subplot(2,1,2); plot(w3/pi,magY3); title('DTFT Magnitude');
>> xlabel('frequency in pi units')
```

Now the plot in Figure 5.10 shows that the sequence has a dominant frequency at $\omega = 0.5\pi$. This fact is not supported by the original sequence, which has two

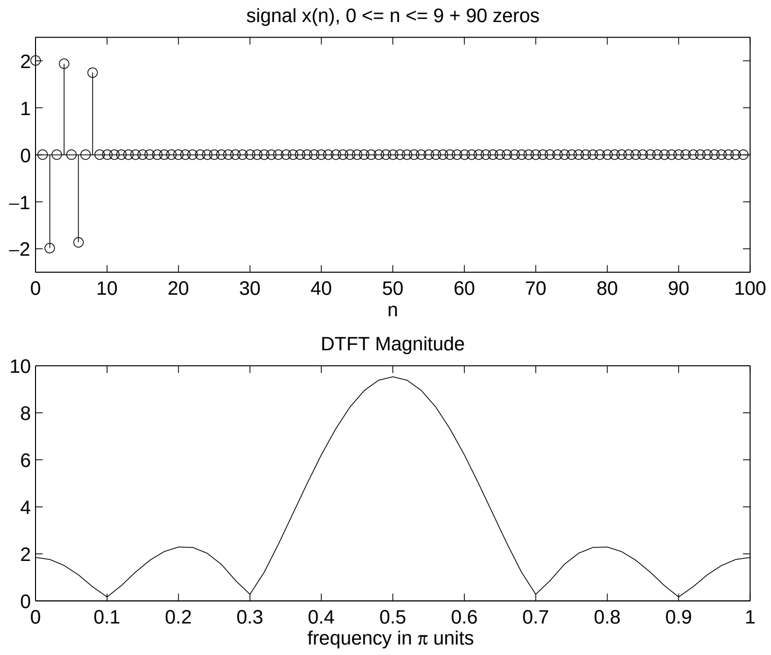


FIGURE 5.10 *Signal and its spectrum in Example 5.8a:* $N = 100$

frequencies. The zero-padding provided a smoother version of the spectrum in Figure 5.9.

b. To get better spectral information, we will take the first 100 samples of $x(n)$ and determine its discrete-time Fourier transform.

MATLAB Script:

```
>> subplot(2,1,1); stem(n,x);
>> title('signal x(n), 0 <= n <= 99'); xlabel('n')
>> X = dft(x,100); magX = abs(X(1:1:51));
>> k = 0:1:50; w = 2*pi/100*k;
>> subplot(2,1,2); plot(w/pi,magX); title('DTFT Magnitude');
>> xlabel('frequency in pi units')
```

Now the discrete-time Fourier transform plot in Figure 5.11 clearly shows two frequencies, which are very close to each other. This is the high-resolution spectrum of $x(n)$. Note that padding more zeros to the 100-point sequence will result in a smoother rendition of the spectrum in Figure 5.11 but will not reveal any new information. Readers are encouraged to verify this. □

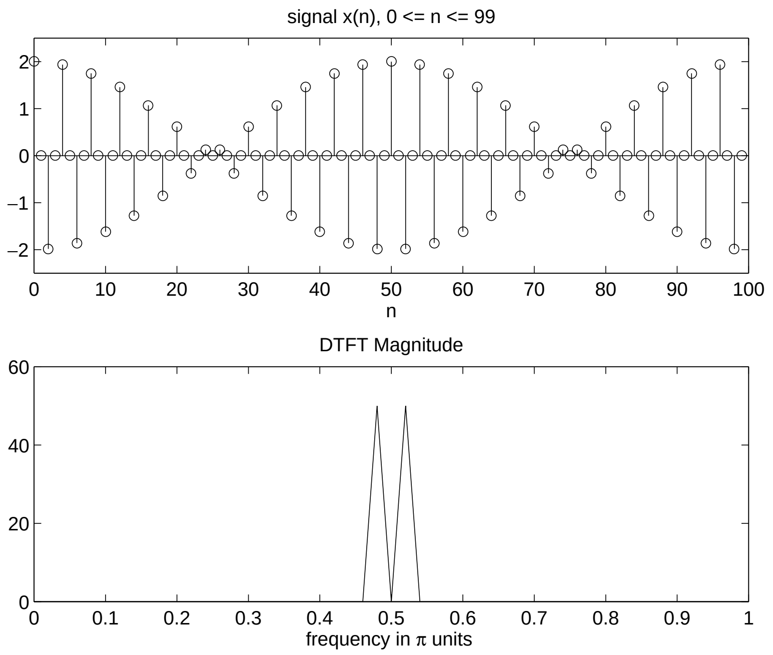


FIGURE 5.11 *Signal and its spectrum in Example 5.8b: $N = 100$*

5.4 PROPERTIES OF THE DISCRETE FOURIER TRANSFORM

The DFT properties are derived from those of the DFS because mathematically DFS is the valid representation. We discuss several useful properties, which are given without proof. These properties also apply to the DFS with necessary changes. Let $X(k)$ be an N-point DFT of the sequence $x(n)$. Unless otherwise stated, the N-point DFTs will be used in these properties.

1. **Linearity:** The DFT is a linear transform

$$\text{DFT}\left[ax_1(n) + bx_2(n)\right] = a\,\text{DFT}\left[x_1(n)\right] + b\,\text{DFT}\left[x_2(n)\right] \tag{5.27}$$

   Note: If $x_1(n)$ and $x_2(n)$ have different durations—that is, they are N_1-point and N_2-point sequences, respectively—then choose $N_3 = \max(N_1, N_2)$ and proceed by taking N_3-point DFTs.

2. **Circular folding:** If an N-point sequence is folded, then the result $x(-n)$ would not be an N-point sequence, and it would not be possible

to compute its DFT. Therefore we use the modulo-N operation on the argument $(-n)$ and define folding by

$$x\left((-n)\right)_N = \begin{cases} x(0), & n = 0 \\ x(N-n), & 1 \le n \le N-1 \end{cases} \tag{5.28}$$

This is called a *circular folding.* To visualize it, imagine that the sequence $x(n)$ is wrapped around a circle in the counterclockwise direction so that indices $n = 0$ and $n = N$ overlap. Then $x((-n))_N$ can be viewed as a clockwise wrapping of $x(n)$ around the circle; hence the name circular folding. In MATLAB the circular folding can be achieved by `x=x(mod(-n,N)+1)`. Note that the arguments in MATLAB begin with 1. The DFT of a circular folding is given by

$$\text{DFT}\left[x\left((-n)\right)_N\right] = X\left((-k)\right)_N = \begin{cases} X(0), & k = 0 \\ X(N-k), & 1 \le k \le N-1 \end{cases} \tag{5.29}$$

□ **EXAMPLE 5.9** Let $x(n) = 10\,(0.8)^n\,, \quad 0 \le n \le 10.$

a. Determine and plot $x\left((-n)\right)_{11}$.
b. Verify the circular folding property.

Solution **a.** MATLAB script:

```
>> n = 0:100; x = 10*(0.8) .^ n;  y = x(mod(-n,11)+1);
>> subplot(2,1,1); stem(n,x); title('Original sequence')
>> xlabel('n'); ylabel('x(n)');
>> subplot(2,1,2); stem(n,y); title('Circularly folded sequence')
>> xlabel('n'); ylabel('x(-n mod 10)');
```

The plots in Figure 5.12 show the effect of circular folding.

b. MATLAB script:

```
>> X = dft(x,11); Y = dft(y,11);
>> subplot(2,2,1); stem(n,real(X));
>> title('Real{DFT[x(n)]}'); xlabel('k');
>> subplot(2,2,2); stem(n,imag(X));
>> title('Imag{DFT[x(n)]}'); xlabel('k');
>> subplot(2,2,3); stem(n,real(Y));
>> title('Real{DFT[x((-n))11]}'); xlabel('k');
>> subplot(2,2,4); stem(n,imag(Y));
>> title('Imag{DFT[x((-n))11]}'); xlabel('k');
```

The plots in Figure 5.13 verify the property. □

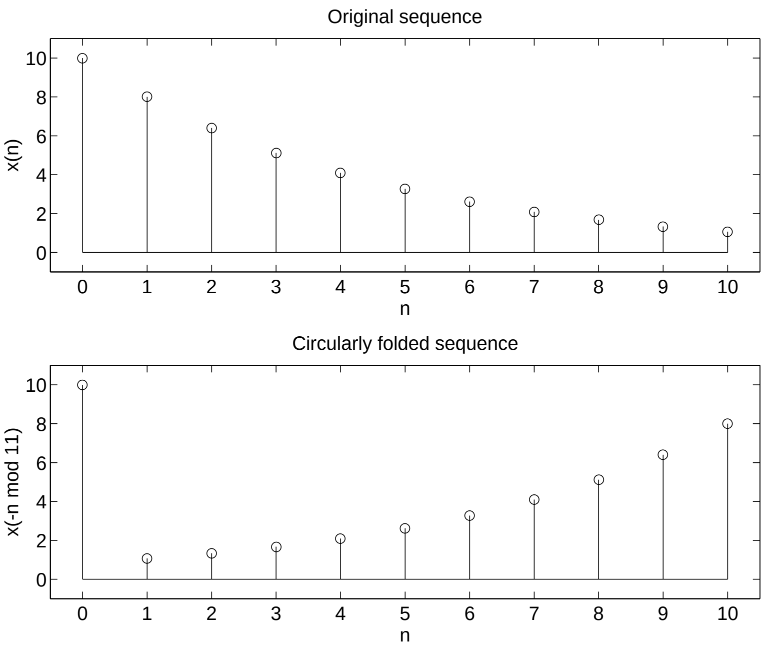


FIGURE 5.12 *Circular folding in Example 5.9a*

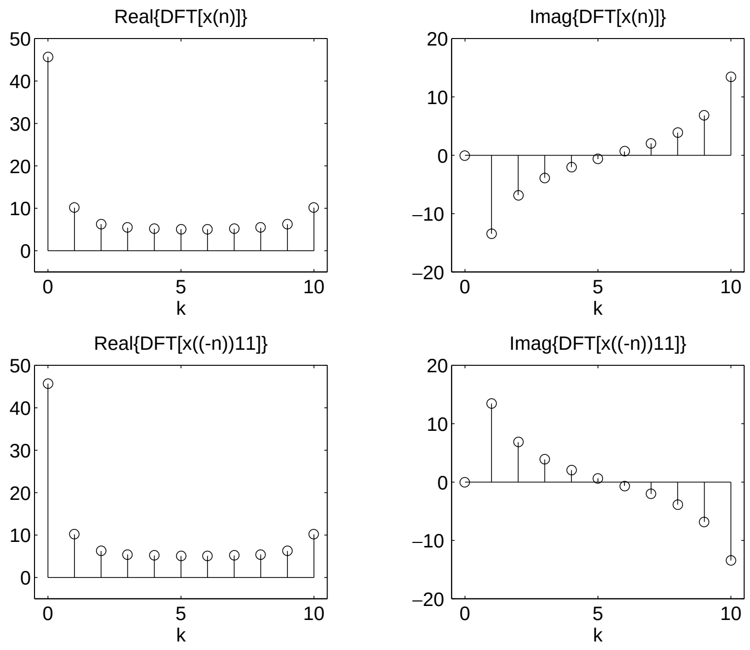


FIGURE 5.13 *Circular folding property in Example 5.9b*

3. **Conjugation:** Similar to the above property we have to introduce the circular folding in the frequency domain.

$$\text{DFT}\left[x^*(n)\right] = X^*\left((-k)\right)_N \tag{5.30}$$

4. **Symmetry properties for real sequences:** Let $x(n)$ be a real-valued N-point sequence. Then $x(n) = x^*(n)$. Using (5.30)

$$X(k) = X^*\left((-k)\right)_N \tag{5.31}$$

This symmetry is called a *circular conjugate symmetry*. It further implies that

$$\begin{aligned}
\text{Re}\left[X(k)\right] &= \text{Re}\left[X\left((-k)\right)_N\right] && \Longrightarrow \text{Circular-even sequence}\\
\text{Im}\left[X(k)\right] &= -\,\text{Im}\left[X\left((N-k)\right)_N\right] && \Longrightarrow \text{Circular-odd sequence}\\
\left|X(k)\right| &= \left|X\left((-k)\right)_N\right| && \Longrightarrow \text{Circular-even sequence}\\
\angle X(k) &= -\angle X\left((-k)\right)_N && \Longrightarrow \text{Circular-odd sequence}
\end{aligned} \tag{5.32}$$

Comments:

1. Observe the magnitudes and angles of the various DFTs in Examples 5.6 and 5.7. They do satisfy the above circular symmetries. These symmetries are different than the usual even and odd symmetries. To visualize this, imagine that the DFT samples are arranged around a circle so that the indices $k = 0$ and $k = N$ overlap; then the samples will be symmetric with respect to $k = 0$, which justifies the name circular symmetry.
2. The corresponding symmetry for the DFS coefficients is called the *periodic conjugate symmetry.*
3. Since these DFTs have symmetry, one needs to compute $X(k)$ only for

$$k = 0, 1, \ldots, \frac{N}{2}; \quad N \text{ even}$$

or for

$$k = 0, 1, \ldots, \frac{N-1}{2}; \quad N \text{ odd}$$

This results in about 50% savings in computation as well as in storage.

4. From (5.30)

$$X(0) = X^*((-0))_N = X^*(0)$$

which means that the DFT coefficient at $k = 0$ must be a real number. But $k = 0$ means that the frequency $\omega_k = k\omega_1 = 0$, which is the DC frequency. Hence the DC coefficient for a real-valued $x(n)$ must be a real number. In addition, if N is even, then $N/2$ is also an integer. Then from (5.32)

$$X(N/2) = X^*\left((-N/2)\right)_N = X^*(N/2)$$

which means that even the $k = N/2$ component is also real-valued. This component is called the *Nyquist component* since $k = N/2$ means that the frequency $\omega_{N/2} = (N/2)(2\pi/N) = \pi$, which is the digital Nyquist frequency.

The real-valued signals can also be decomposed into their even and odd components, $x_e(n)$ and $x_o(n)$, respectively, as discussed in Chapter 2. However, these components are not N-point sequences and therefore we cannot take their N-point DFTs. Hence we define a new set of components using the circular folding discussed above. These are called *circular-even* and *circular-odd* components defined by

$$x_{ec}(n) \triangleq \tfrac{1}{2}\left[x(n) + x\left((-n)\right)_N\right] = \begin{cases} x(0), & n = 0 \\ \tfrac{1}{2}\left[x(n) + x(N-n)\right], & 1 \le n \le N-1 \end{cases}$$
$$x_{oc}(n) \triangleq \tfrac{1}{2}\left[x(n) - x\left((-n)\right)_N\right] = \begin{cases} 0, & n = 0 \\ \tfrac{1}{2}\left[x(n) - x(N-n)\right], & 1 \le n \le N-1 \end{cases} \tag{5.33}$$

Then

$$\begin{aligned} \text{DFT}\left[x_{ec}(n)\right] &= \text{Re}\left[X(k)\right] = \text{Re}\left[X\left((-k)\right)_N\right] \\ \text{DFT}\left[x_{oc}(n)\right] &= \text{Im}\left[X(k)\right] = \text{Im}\left[X\left((-k)\right)_N\right] \end{aligned} \tag{5.34}$$

Implication: If $x(n)$ is real and circular-even, then its DFT is also real and circular-even. Hence only the first $0 \le n \le N/2$ coefficients are necessary for complete representation.

Using (5.33), it is easy to develop a function to decompose an N-point sequence into its circular-even and circular-odd components. The following `circevod` function uses the `mod` function given earlier to implement the modulo-N operation.

```
function [xec, xoc] = circevod(x)
% signal decomposition into circular-even and circular-odd parts
% ---------------------------------------------------------------
% [xec, xoc] = circevod(x)
%
if any(imag(x) ~= 0)
      error('x is not a real sequence')
```

```
end
N = length(x); n = 0:(N-1);
xec = 0.5*(x + x(mod(-n,N)+1));  xoc = 0.5*(x - x(mod(-n,N)+1));
```

□ **EXAMPLE 5.10** Let $x(n) = 10\,(0.8)^n\,, \quad 0 \le n \le 10$ as in Example 5.9.

a. Decompose and plot the $x_{ec}(n)$ and $x_{oc}(n)$ components of $x(n)$.
b. Verify the property in (5.34).

Solution **a.** MATLAB script:

```
>> n = 0:10; x = 10*(0.8) .^ n;
>> [xec,xoc] = circevod(x);
>> subplot(2,1,1); stem(n,xec); title('Circular-even component')
>> xlabel('n'); ylabel('xec(n)'); axis([-0.5,10.5,-1,11])
>> subplot(2,1,2); stem(n,xoc); title('Circular-odd component')
>> xlabel('n'); ylabel('xoc(n)'); axis([-0.5,10.5,-4,4])
```

The plots in Figure 5.14 show the circularly symmetric components of $x(n)$.

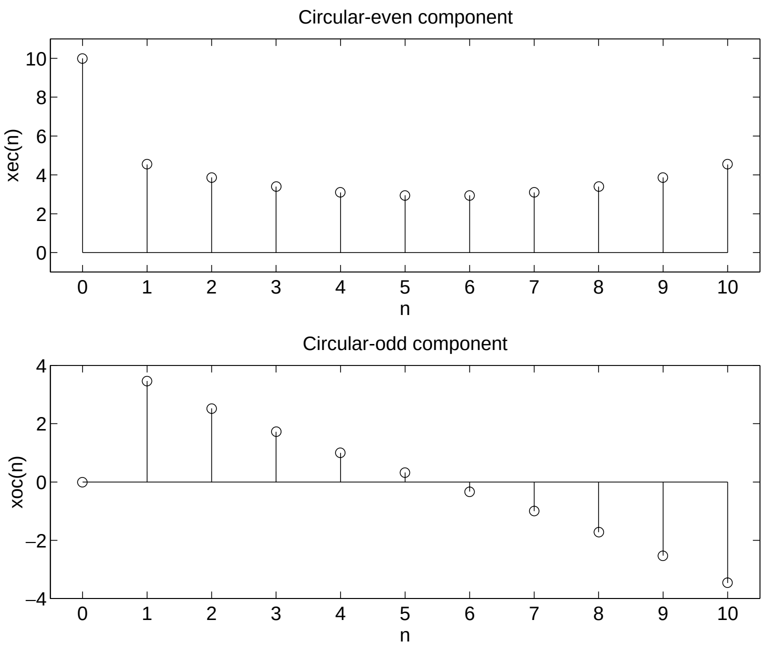


FIGURE 5.14 *Circular-even and circular-odd components of the sequence in Example 5.10a*

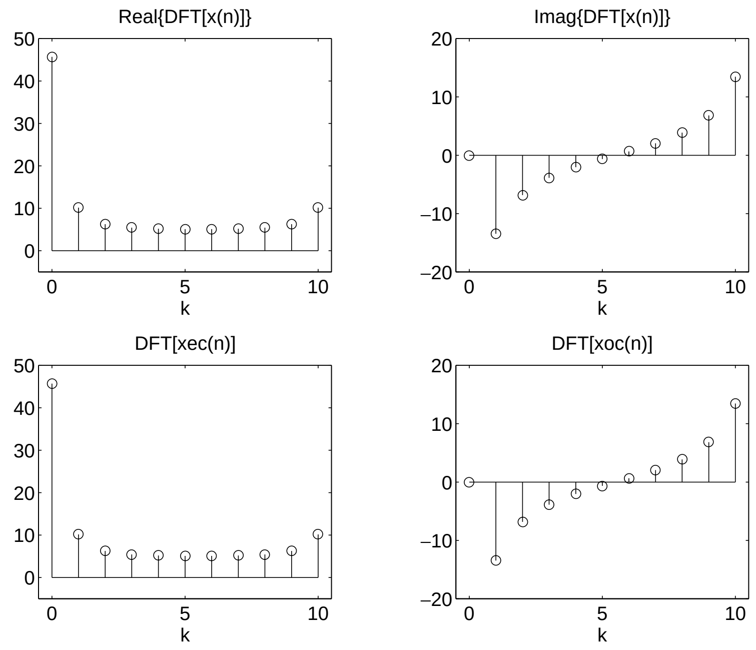


FIGURE 5.15 *Plots of DFT symmetry properties in Example 5.10b*

b. MATLAB script:

```
>> X = dft(x,11); Xec = dft(xec,11); Xoc = dft(xoc,11);
>> subplot(2,2,1); stem(n,real(X)); axis([-0.5,10.5,-5,50])
>> title('Real{DFT[x(n)]}'); xlabel('k');
>> subplot(2,2,2); stem(n,imag(X)); axis([-0.5,10.5,-20,20])
>> title('Imag{DFT[x(n)]}'); xlabel('k');
>> subplot(2,2,3); stem(n,real(Xec)); axis([-0.5,10.5,-5,50])
>> title('DFT[xec(n)]'); xlabel('k');
>> subplot(2,2,4); stem(n,imag(Xoc)); axis([-0.5,10.5,-20,20])
>> title('DFT[xoc(n)]'); xlabel('k');
```

From the plots in Figure 5.15 we observe that the DFT of $x_{ec}(n)$ is the same as the real part of $X(k)$ and that the DFT of $x_{oc}(n)$ is the same as the imaginary part of $X(k)$. □

A similar property for complex-valued sequences is explored in Problem P5.18.

5. **Circular shift of a sequence:** If an N-point sequence is shifted in either direction, then the result is no longer between $0 \leq n \leq N-1$.

Therefore we first convert $x(n)$ into its periodic extension $\tilde{x}(n)$, and then shift it by m samples to obtain

$$\tilde{x}(n-m) = x\left((n-m)\right)_N \tag{5.35}$$

This is called a *periodic shift* of $\tilde{x}(n)$. The periodic shift is then converted into an N-point sequence. The resulting sequence

$$\tilde{x}(n-m)\mathcal{R}_N(n) = x\left((n-m)\right)_N \mathcal{R}_N(n) \tag{5.36}$$

is called the *circular shift* of $x(n)$. Once again to visualize this, imagine that the sequence $x(n)$ is wrapped around a circle. Now rotate the circle by k samples and unwrap the sequence from $0 \le n \le N-1$. Its DFT is given by

$$\text{DFT}\left[x\left((n-m)\right)_N \mathcal{R}_N(n)\right] = W_N^{km} X(k) \tag{5.37}$$

□ **EXAMPLE 5.11** Let $x(n) = 10\,(0.8)^n\,, \quad 0 \le n \le 10$ be an 11-point sequence.

a. Sketch $x((n+4))_{11}R_{11}(n)$, that is, a circular shift by 4 samples toward the left.

b. Sketch $x((n-3))_{15}R_{15}(n)$, that is, a circular shift by 3 samples toward the right, where $x(n)$ is assumed to be a 15-point sequence.

Solution We will use a step-by-step graphical approach to illustrate the circular shifting operation. This approach shows the periodic extension $\tilde{x}(n) = x((n))_N$ of $x(n)$, followed by a linear shift in $\tilde{x}(n)$ to obtain $\tilde{x}(n-m) = x((n-m))_N$, and finally truncating $\tilde{x}(n-m)$ to obtain the circular shift.

a. Figure 5.16 shows four sequences. The top-left shows $x(n)$, the bottom-left shows $\tilde{x}(n)$, the top-right shows $\tilde{x}(n+4)$, and finally the bottom-right shows $x((n+4))_{11}R_{11}(n)$. Note carefully that as samples move out of the $[0, N-1]$ window in one direction, they reappear from the opposite direction. This is the meaning of the circular shift, and it is different from the linear shift.

b. In this case the sequence $x(n)$ is treated as a 15-point sequence by padding 4 zeros. Now the circular shift will be different than when $N = 11$. This is shown in Figure 5.17. In fact the circular shift $x\left((n-3)\right)_{15}$ looks like a linear shift $x(n-3)$. □

To implement a circular shift, we do not have to go through the periodic shift as shown in Example 5.11. It can be implemented directly in two ways. In the first approach, the modulo-N operation can be used on the argument $(n-m)$ in the time domain. This is shown below in the `cirshftt` function.

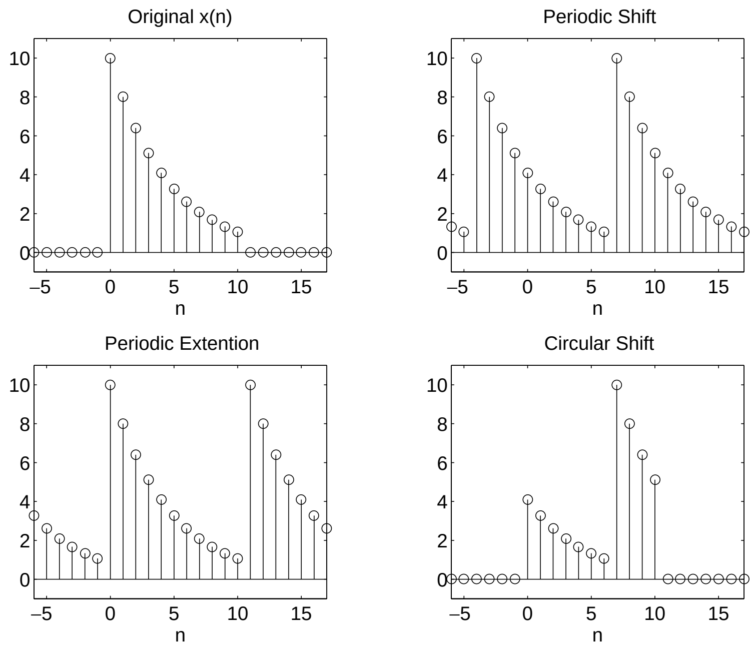


FIGURE 5.16 *Graphical interpretation of circular shift, $N = 11$*

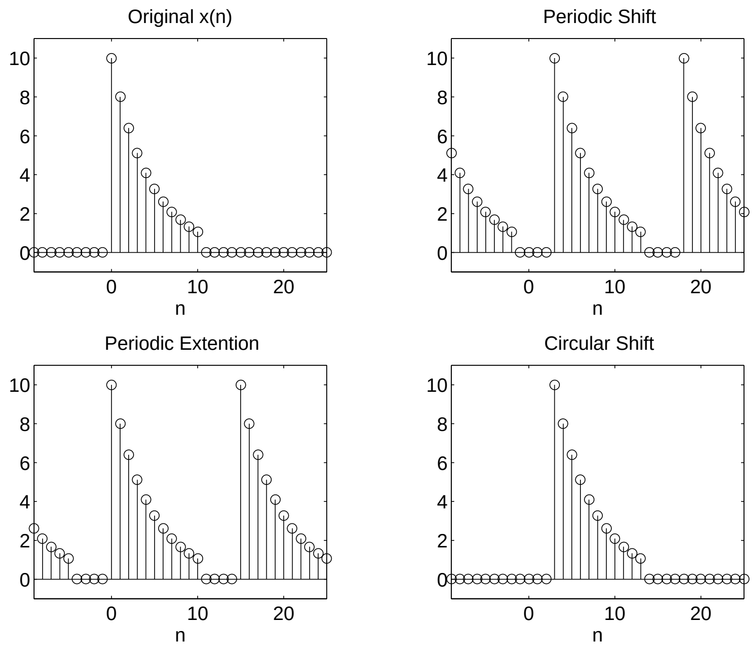


FIGURE 5.17 *Graphical interpretation of circular shift, $N = 15$*

```
function y = cirshftt(x,m,N)
% Circular shift of m samples wrt size N in sequence x: (time domain)
% -------------------------------------------------------------------
% [y] = cirshftt(x,m,N)
% y = output sequence containing the circular shift
% x = input sequence of length <= N
% m = sample shift
% N = size of circular buffer
%  Method: y(n) = x((n-m) mod N)
% Check for length of x
if length(x) > N
      error('N must be >= the length of x')
end
x = [x zeros(1,N-length(x))];
n = [0:1:N-1];  n = mod(n-m,N);  y = x(n+1);
```

In the second approach, the property (5.37) can be used in the frequency domain. This is explored in Problem P5.20.

□ **EXAMPLE 5.12** Given an 11-point sequence $x(n) = 10\,(0.8)^n\,, \quad 0 \le n \le 10$, determine and plot $x\left((n-6)\right)_{15}$.

Solution MATLAB script:

```
>> n = 0:10; x = 10*(0.8) .^ n;  y = cirshftt(x,6,15);
>> n = 0:14; x = [x, zeros(1,4)];
>> subplot(2,1,1); stem(n,x); title('Original sequence')
>> xlabel('n'); ylabel('x(n)');
>> subplot(2,1,2); stem(n,y);
>> title('Circularly shifted sequence, N=15')
>> xlabel('n'); ylabel('x((n-6) mod 15)');
```

The results are shown in Figure 5.18. □

6. **Circular shift in the frequency domain:** This property is a dual of the preceding property given by

$$\text{DFT}\left[W_N^{-\ell n} x(n)\right] = X\left((k-\ell)\right)_N R_N(k) \tag{5.38}$$

7. **Circular convolution:** A linear convolution between two N-point sequences will result in a longer sequence. Once again we have to restrict our interval to $0 \le n \le N-1$. Therefore instead of linear shift, we should consider the circular shift. A convolution operation

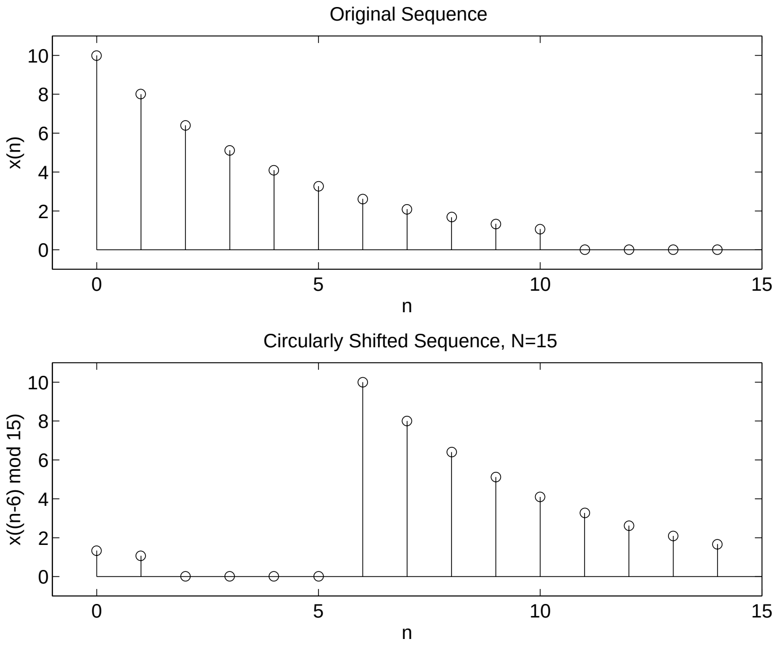


FIGURE 5.18 *Circularly shifted sequence in Example 5.12*

that contains a circular shift is called the *circular convolution* and is given by

$$x_1(n) \text{ Ⓝ } x_2(n) = \sum_{m=0}^{N-1} x_1(m) x_2\left((n-m)\right)_N, \quad 0 \le n \le N-1 \qquad \textbf{(5.39)}$$

Note that the circular convolution is also an N-point sequence. It has a structure similar to that of a linear convolution. The differences are in the summation limits and in the N-point circular shift. Hence it depends on N and is also called an N-point circular convolution. Therefore the use of the notation Ⓝ is appropriate. The DFT property for the circular convolution is

$$\text{DFT}\left[x_1(n) \text{ Ⓝ } x_2(n)\right] = X_1(k) \cdot X_2(k) \qquad \textbf{(5.40)}$$

An alternate interpretation of this property is that when we multiply two N-point DFTs in the frequency domain, we get the circular convolution (and not the usual linear convolution) in the time domain.

□ **EXAMPLE 5.13** Let $x_1(n) = \{1, 2, 2\}$ and $x_2(n) = \{1, 2, 3, 4\}$. Compute the 4-point circular convolution $x_1(n) \text{ ④ } x_2(n)$.

Solution Note that $x_1(n)$ is a 3-point sequence, hence we will have to pad one zero to make it a 4-point sequence before we perform the circular convolution. We will compute this convolution in the time domain as well as in the frequency domain. In the time domain we will use the mechanism of circular convolution, while in the frequency domain we will use the DFTs.

- *Time-domain approach:* The 4-point circular convolution is given by

$$x_1(n) \textcircled{4} \, x_2(n) = \sum_{m=0}^{3} x_1(m)\, x_2\left((n-m)\right)_4$$

Thus we have to create a circularly folded and shifted sequence $x_2((n-m))_N$ for each value of n, multiply it sample by sample with $x_1(m)$, add the samples to obtain the circular convolution value for that n, and then repeat the procedure for $0 \le n \le 3$. Consider

$$x_1(m) = \{1,\ 2,\ 2,\ 0\} \qquad \text{and} \qquad x_2(m) = \{1,\ 2,\ 3,\ 4\}$$

<u>for $n = 0$</u>

$$\begin{aligned}\sum_{m=0}^{3} x_1(m) \cdot x_2\left((0-m)\right)_5 &= \sum_{m=0}^{3} [\{1,\ 2,\ 2,\ 0\} \cdot \{1,\ 4,\ 3,\ 2\}] \\ &= \sum_{m=0}^{3} \{1,\ 8,\ 6,\ 0\} = 15\end{aligned}$$

<u>for $n = 1$</u>

$$\begin{aligned}\sum_{m=0}^{3} x_1(m) \cdot x_2\left((1-m)\right)_5 &= \sum_{m=0}^{3} [\{1,\ 2,\ 2,\ 0\} \cdot \{2,\ 1,\ 4,\ 3\}] \\ &= \sum_{m=0}^{3} \{2,\ 2,\ 8,\ 0\} = 12\end{aligned}$$

<u>for $n = 2$</u>

$$\begin{aligned}\sum_{m=0}^{3} x_1(m) \cdot x_2\left((2-m)\right)_5 &= \sum_{m=0}^{3} [\{1,\ 2,\ 2,\ 0\} \cdot \{3,\ 2,\ 1,\ 4\}] \\ &= \sum_{m=0}^{3} \{3,\ 4,\ 2,\ 0\} = 9\end{aligned}$$

<u>for $n = 3$</u>

$$\begin{aligned}\sum_{m=0}^{3} x_1(m) \cdot x_2\left((3-m)\right)_5 &= \sum_{m=0}^{3} [\{1,\ 2,\ 2,\ 0\} \cdot \{4,\ 3,\ 2,\ 1\}] \\ &= \sum_{m=0}^{3} \{4,\ 6,\ 4,\ 0\} = 14\end{aligned}$$

Hence

$$x_1(n) \textcircled{4} x_2(n) = \{15,\ 12,\ 9,\ 14\}$$

- *Frequency-domain approach:* In this approach we first compute 4-point DFTs of $x_1(n)$ and $x_2(n)$, multiply them sample by sample, and then take the inverse DFT of the result to obtain the circular convolution.

DFT of $x_1(n)$

$$x_1(n) = \{1, 2, 2, 0\} \Longrightarrow X_1(k) = \{5,\ -1 - j2,\ 1,\ -1 + j2\}$$

DFT of $x_2(n)$

$$x_2(n) = \{1, 2, 3, 4\} \Longrightarrow X_2(k) = \{10,\ -2 + j2,\ -2,\ -2 - j2\}$$

Now

$$X_1(k) \cdot X_2(k) = \{50,\ 6 + j2,\ -2,\ 6 - j2\}$$

Finally after IDFT,

$$x_1(n) \textcircled{4} x_2(n) = \{15,\ 12,\ 9,\ 14\}$$

which is the same as before. □

Similar to the circular shift implementation, we can implement the circular convolution in a number of different ways. The simplest approach would be to implement (5.39) literally by using the `cirshftt` function and requiring two nested `for...end` loops. Obviously, this is not efficient. Another approach is to generate a sequence $x\left((n-m)\right)_N$ for each n in $[0, N-1]$ as rows of a matrix and then implement (5.39) as a matrix-vector multiplication similar to our `dft` function. This would require one `for...end` loop. The following `circonvt` function incorporates these steps.

```
function y = circonvt(x1,x2,N)
% N-point circular convolution between x1 and x2: (time-domain)
% ---------------------------------------------------------------
% [y] = circonvt(x1,x2,N)
%  y = output sequence containing the circular convolution
% x1 = input sequence of length N1 <= N
% x2 = input sequence of length N2 <= N
%  N = size of circular buffer
%  Method: y(n) = sum (x1(m)*x2((n-m) mod N))
% Check for length of x1
if length(x1) > N
        error('N must be >= the length of x1')
end
```

```
% Check for length of x2
if length(x2) > N
        error('N must be >= the length of x2')
end
x1=[x1 zeros(1,N-length(x1))];
x2=[x2 zeros(1,N-length(x2))];
m = [0:1:N-1];  x2 = x2(mod(-m,N)+1);  H = zeros(N,N);
for n = 1:1:N
H(n,:) = cirshftt(x2,n-1,N);
end
y = x1*conj(H');
```

Problems P5.24 and P5.25 explore an approach to eliminate the `for... end` loop in the `circonvt` function. The third approach would be to implement the frequency-domain operation (5.40) using the `dft` function. This is explored in Problem P5.26.

□ **EXAMPLE 5.14** Let us use MATLAB to perform the circular convolution in Example 5.13.

Solution The sequences are $x_1(n) = \{1, 2, 2\}$ and $x_2(n) = \{1, 2, 3, 4\}$.

MATLAB script:

```
>> x1 = [1,2,2]; x2 = [1,2,3,4];  y = circonvt(x1, x2, 4)
y =
    15    12     9    14
```

Hence

$$x_1(n) \textcircled{4} x_2(n) = \{15,\ 12,\ 9,\ 14\}$$

as before. □

□ **EXAMPLE 5.15** In this example we will study the effect of N on the circular convolution. Obviously, $N \geq 4$; otherwise there will be a time-domain aliasing for $x_2(n)$. We will use the same two sequences from Example 5.13.

a. Compute $x_1(n) \textcircled{5} x_2(n)$.

b. Compute $x_1(n) \textcircled{6} x_2(n)$.

c. Comment on the results.

Solution The sequences are $x_1(n) = \{1, 2, 2\}$ and $x_2(n) = \{1, 2, 3, 4\}$. Even though the sequences are the same as in Example 5.14, we should expect different results for different values of N. This is not the case with the linear convolution, which is unique, given two sequences.

a. MATLAB Script for 5-point circular convolution:

```
>> x1 = [1,2,2]; x2 = [1,2,3,4];  y = circonvt(x1, x2, 5)
y =
      9     4     9    14    14
```

Hence

$$x_1(n) \textcircled{5}\, x_2(n) = \{9,\ 4,\ 9,\ 14,\ 14\}$$

b. MATLAB Script for 6-point circular convolution:

```
>> x1 = [1,2,2]; x2 = [1,2,3,4];  y = circonvt(x1, x2, 6)
y =
      1     4     9    14    14     8
```

Hence

$$x_1(n) \textcircled{6}\, x_2(n) = \{1,\ 4,\ 9,\ 14,\ 14,\ 8\}$$

c. A careful observation of 4-, 5-, and 6-point circular convolutions from this and the previous example indicates some unique features. Clearly, an N-point circular convolution is an N-point sequence. However, some samples in these convolutions have the same values, while other values can be obtained as a sum of samples in other convolutions. For example, the first sample in the 5-point convolution is a sum of the first and the last samples of the 6-point convolution. The linear convolution between $x_1(n)$ and $x_2(n)$ is given by

$$x_1(n) * x_2(n) = \{1,\ 4,\ 9,\ 14,\ 14,\ 8\}$$

which is equivalent to the 6-point circular convolution. These and other issues are explored in the next section. □

8. **Multiplication:** This is the dual of the circular convolution property. It is given by

$$\text{DFT}\,[x_1(n) \cdot x_2(n)] = \frac{1}{N} X_1(k) \textcircled{N}\, X_2(k) \tag{5.41}$$

in which the circular convolution is performed in the frequency domain. The MATLAB functions developed for circular convolution can also be used here since $X_1(k)$ and $X_2(k)$ are also N-point sequences.

9. **Parseval's relation:** This relation computes the energy in the frequency domain.

$$\text{E}_x = \sum_{n=0}^{N-1} |x(n)|^2 = \frac{1}{N} \sum_{k=0}^{N-1} |X(k)|^2 \tag{5.42}$$

The quantity $\frac{|X(k)|^2}{N}$ is called the *energy spectrum* of finite-duration sequences. Similarly, for periodic sequences, the quantity $|\frac{\tilde{X}(k)}{N}|^2$ is called the *power spectrum.*

5.5 LINEAR CONVOLUTION USING THE DFT

One of the most important operations in linear systems is the linear convolution. In fact, FIR filters are generally implemented in practice using this linear convolution. On the other hand, the DFT is a practical approach for implementing linear system operations in the frequency domain. As we shall see later, it is also an efficient operation in terms of computations. However, there is one problem. The DFT operations result in a circular convolution (something that we do not desire), not in a linear convolution that we want. Now we shall see how to use the DFT to perform a linear convolution (or equivalently, how to make a circular convolution identical to the linear convolution). We alluded to this problem in Example 5.15.

Let $x_1(n)$ be an N_1-point sequence and let $x_2(n)$ be an N_2-point sequence. Define the linear convolution of $x_1(n)$ and $x_2(n)$ by $x_3(n)$, that is,

$$\begin{aligned} x_3(n) &= x_1(n) * x_2(n) \\ &= \sum_{k=-\infty}^{\infty} x_1(k)x_2(n-k) = \sum_{0}^{N_1-1} x_1(k)x_2(n-k) \end{aligned} \tag{5.43}$$

Then $x_3(n)$ is a $(N_1 + N_2 - 1)$-point sequence. If we choose $N = \max(N_1, N_2)$ and compute an N-point circular convolution $x_1(n)$ Ⓝ $x_2(n)$, then we get an N-point sequence, which obviously is different from $x_3(n)$. This observation also gives us a clue. Why not choose $N = N_1 + N_2 - 1$ and perform an $(N_1 + N_2 - 1)$-point circular convolution? Then at least both of these convolutions will have an equal number of samples.

Therefore let $N = N_1 + N_2 - 1$ and let us treat $x_1(n)$ and $x_2(n)$ as N-point sequences. Define the N-point circular convolution by $x_4(n)$.

$$\begin{aligned} x_4(n) &= x_1(n) \text{ Ⓝ } x_2(n) \qquad (5.44) \\ &= \left[\sum_{m=0}^{N-1} x_1(m)x_2((n-m))_N\right] \mathcal{R}_N(n) \\ &= \left[\sum_{m=0}^{N-1} x_1(m) \sum_{r=-\infty}^{\infty} x_2(n-m-rN)\right] \mathcal{R}_N(n) \end{aligned}$$

$$= \left[\sum_{r=-\infty}^{\infty} \underbrace{\sum_{m=0}^{N_1-1} x_1(m)x_2(n-m-rN)}_{x_3(n-rN)} \right] \mathcal{R}_N(n)$$

$$= \left[\sum_{r=-\infty}^{\infty} x_3(n-rN) \right] \mathcal{R}_N(n) \qquad \text{using (5.43)}$$

This analysis shows that, in general, the circular convolution is an aliased version of the linear convolution. We observed this fact in Example 5.15. Now since $x_3(n)$ is an $N = (N_1 + N_2 - 1)$-point sequence, we have

$$x_4(n) = x_3(n); \quad 0 \le n \le (N-1)$$

which means that there is no aliasing in the time domain.

Conclusion: If we make both $x_1(n)$ and $x_2(n)$ $N = N_1 + N_2 - 1$ point sequences by padding an appropriate number of zeros, then the circular convolution is identical to the linear convolution.

□ **EXAMPLE 5.16** Let $x_1(n)$ and $x_2(n)$ be the following two 4-point sequences.

$$x_1(n) = \{1,\ 2,\ 2,\ 1\}, \quad x_2(n) = \{1,\ -1,\ -1,\ 1\}$$

a. Determine their linear convolution $x_3(n)$.
b. Compute the circular convolution $x_4(n)$ so that it is equal to $x_3(n)$.

Solution We will use MATLAB to do this problem.

a. MATLAB Script:

```
>> x1 = [1,2,2,1]; x2 = [1,-1,-1,1];  x3 = conv(x1,x2)
x3 =    1     1    -1    -2    -1     1     1
```

Hence the linear convolution $x_3(n)$ is a 7-point sequence given by

$$x_3(n) = \{1, 1, -1, -2, -1, 1, 1\}$$

b. We will have to use $N \ge 7$. Choosing $N = 7$, we have

```
>> x4 = circonvt(x1,x2,7)
x4 =   1     1    -1    -2    -1     1     1
```

Hence

$$x_4 = \{1, 1, -1, -2, -1, 1, 1\} = x_3(n)$$ □

5.5.1 ERROR ANALYSIS

To use the DFT for linear convolution, we must choose N properly. However, in practice it may not be possible to do so, especially when N is very large and there is a limit on memory. Then an error will be introduced when N is chosen less than the required value to perform the circular convolution. We want to compute this error, which is useful in practice. Obviously, $N \geq \max(N_1, N_2)$. Therefore let

$$\max(N_1, N_2) \leq N < (N_1 + N_2 - 1)$$

Then, from our previous analysis (5.44)

$$x_4(n) = \left[\sum_{r=-\infty}^{\infty} x_3(n - rN)\right] \mathcal{R}_N(n)$$

Let an error $e(n)$ be given by

$$\begin{aligned} e(n) &\overset{\triangle}{=} x_4(n) - x_3(n) \\ &= \left[\sum_{r\neq 0} x_3(n - rN)\right] \mathcal{R}_N(n) \end{aligned}$$

Since $N \geq \max(N_1, N_2)$, only two terms corresponding to $r = \pm 1$ remain in the above summation. Hence

$$e(n) = [x_3(n - N) + x_3(n + N)]\, \mathcal{R}_N(n)$$

Generally, $x_1(n)$ and $x_2(n)$ are causal sequences. Then $x_3(n)$ is also causal, which means that

$$x_3(n - N) = 0; \quad 0 \leq n \leq N - 1$$

Therefore

$$e(n) = x_3(n + N), \quad 0 \leq n \leq N - 1 \tag{5.45}$$

This is a simple yet important relation. It implies that when $\max(N_1, N_2) \leq N < (N_1 + N_2 - 1)$ the error value at n is the same as the linear convolution value computed N samples away. Now the linear convolution will be zero after $(N_1 + N_2 - 1)$ samples. This means that the first few samples of the circular convolution are in error, while the remaining ones are the correct linear convolution values.

□ **EXAMPLE 5.17** Consider the sequences $x_1(n)$ and $x_2(n)$ from the previous example. Evaluate circular convolutions for $N = 6$, 5, and 4. Verify the error relations in each case.

Solution Clearly, the linear convolution $x_3(n)$ is still the same.

$$x_3(n) = \{1, 1, -1, -2, -1, 1, 1\}$$

When $N = 6$, we obtain a 6-point sequence.

$$x_4(n) = x_1(n) \; \textcircled{6} \; x_2(n) = \{2, 1, -1, -2, -1, 1\}$$

Therefore

$$\begin{aligned} e(n) &= \{2, 1, -1, -2, -1, 1\} - \{1, 1, -1, -2, -1, 1\}, \quad 0 \le n \le 5 \\ &= \{1, 0, 0, 0, 0, 0\} \\ &= x_3(n + 6) \end{aligned}$$

as expected. When $N = 5$, we obtain a 5-point sequence,

$$x_4(n) = x_1(n) \; \textcircled{5} \; x_2(n) = \{2, 2, -1, -2, -1\}$$

and

$$\begin{aligned} e(n) &= \{2, 2, -1, -2, -1\} - \{1, 1, -1, -2, -1\}, \quad 0 \le n \le 4 \\ &= \{1, 1, 0, 0, 0\} \\ &= x_3(n + 5) \end{aligned}$$

Finally, when $N = 4$, we obtain a 4-point sequence,

$$x_4(n) = x_1(n) \; \textcircled{4} \; x_2(n) = \{0, 2, 0, -2\}$$

and

$$\begin{aligned} e(n) &= \{0, 2, 0, -2\} - \{1, 1, -1, -2\}, \quad 0 \le n \le 3 \\ &= \{-1, 1, 1, 0\} \\ &= x_3(n + 4) \end{aligned}$$

The last case of $N = 4$ also provides the following useful observation.

Observation: When $N = \max(N_1, N_2)$ is chosen for circular convolution, then the first $(M - 1)$ samples are in error (i.e., different from the linear convolution), where $M = \min(N_1, N_2)$. This result is useful in implementing long convolutions in the form of block processing. □

5.5.2 BLOCK CONVOLUTIONS

When we want to filter an input sequence that is being received continuously, such as a speech signal from a microphone, then for practical purposes we can think of this sequence as an infinite-length sequence. If we want to implement this filtering operation as an FIR filter in which the linear convolution is computed using the DFT, then we experience some practical problems. We will have to compute a large DFT, which is generally impractical. Furthermore, output samples are not available until all input samples are processed. This introduces an unacceptably large amount of delay. Therefore we have to segment the infinite-length input sequence into smaller sections (or blocks), process each section using the DFT, and finally assemble the output sequence from the outputs of each section. This procedure is called a *block convolution* (or block processing) operation.

Let us assume that the sequence $x(n)$ is sectioned into N-point sequences and that the impulse response of the filter is an M-point sequence, where $M < N$. Then from the observation in Example 5.17 we note that the N-point circular convolution between the input block and the impulse response will yield a block output sequence in which the first $(M-1)$ samples are not the correct output values. If we simply partition $x(n)$ into nonoverlapping sections, then the resulting output sequence will have intervals of incorrect samples. To correct this problem, we can partition $x(n)$ into sections, each overlapping with the previous one by exactly $(M-1)$ samples, save the last $(N-M+1)$ output samples, and finally concatenate these outputs into a sequence. To correct for the first $(M-1)$ samples in the first output block, we set the first $(M-1)$ samples in the first input block to zero. This procedure is called an *overlap-save* method of block convolutions. Clearly, when $N \gg M$, this method is more efficient. We illustrate it using a simple example.

□ **EXAMPLE 5.18** Let $x(n) = (n+1)\,, \quad 0 \le n \le 9$ and $h(n) = \{\underset{\uparrow}{1}, 0, -1\}$. Implement the overlap-save method using $N = 6$ to compute $y(n) = x(n) * h(n)$.

Solution Since $M = 3$, we will have to overlap each section with the previous one by two samples. Now $x(n)$ is a 10-point sequence, and we will need $(M-1) = 2$ zeros in the beginning. Since $N = 6$, we will need 3 sections. Let the sections be

$$\begin{aligned} x_1(n) &= \{0, 0, 1, 2, 3, 4\} \\ x_2(n) &= \{3, 4, 5, 6, 7, 8\} \\ x_3(n) &= \{7, 8, 9, 10, 0, 0\} \end{aligned}$$

Note that we have to pad $x_3(n)$ by two zeros since $x(n)$ runs out of values at $n = 9$. Now we will compute the 6-point circular convolution of each section with $h(n)$.

$$\begin{aligned} y_1 = x_1(n) \textcircled{6}\, h(n) &= \{-3, -4, 1, 2, 2, 2\} \\ y_2 = x_2(n) \textcircled{6}\, h(n) &= \{-4, -4, 2, 2, 2, 2\} \\ y_3 = x_3(n) \textcircled{6}\, h(n) &= \{7, 8, 2, 2, -9, -10\} \end{aligned}$$

Noting that the first two samples in each section are to be discarded, we assemble the output $y(n)$ as

$$y(n) = \{\underset{\uparrow}{1}, 2, 2, 2, 2, 2, 2, 2, 2, 2, -9, -10\}$$

The linear convolution is given by

$$x(n) * h(n) = \{\underset{\uparrow}{1}, 2, 2, 2, 2, 2, 2, 2, 2, 2, -9, -10\}$$

which agrees with the overlap-save method. □

5.5.3 MATLAB IMPLEMENTATION

Using this example as a guide, we can develop a MATLAB function to implement the overlap-save method for a very long input sequence $x(n)$. The key step in this function is to obtain a proper indexing for the segmentation. Given $x(n)$ for $n \geq 0$, we have to set the first $(M-1)$ samples to zero to begin the block processing. Let this augmented sequence be

$$\hat{x}(n) \overset{\triangle}{=} \{\underbrace{0, 0, \ldots, 0}_{(M-1)\text{ zeros}}, x(n)\}, \quad n \geq 0$$

and let $L = N - M + 1$, then the kth block $x_k(n), \quad 0 \leq n \leq N-1$, is given by

$$x_k(n) = \hat{x}(m); \quad kL \leq m \leq kL + N - 1,\ k \geq 0,\ 0 \leq n \leq N-1$$

The total number of blocks is given by

$$K = \left\lfloor \frac{N_x + M - 2}{L} \right\rfloor + 1$$

where N_x is the length of $x(n)$ and $\lfloor \cdot \rfloor$ is the truncation operation. Now each block can be circularly convolved with $h(n)$ using the `circonvt` function developed earlier to obtain

$$y_k(n) = x_k(n) \; \textcircled{N} \; h(n)$$

Finally, discarding the first $(M-1)$ samples from each $y_k(n)$ and concatenating the remaining samples, we obtain the linear convolution $y(n)$. This procedure is incorporated in the following `ovrlpsav` function.

```
%%\leftskip12pt
function [y] = ovrlpsav(x,h,N)
% Overlap-Save method of block convolution
% ----------------------------------------
% [y] = ovrlpsav(x,h,N)
% y = output sequence
% x = input sequence
% h = impulse response
% N = block length
%
Lenx = length(x); M = length(h);  M1 = M-1; L = N-M1;
   h = [h zeros(1,N-M)];
   %
   x = [zeros(1,M1), x, zeros(1,N-1)]; % preappend (M-1) zeros
   K = floor((Lenx+M1-1)/(L));         % # of blocks
   Y = zeros(K+1,N);
   % convolution with succesive blocks
   for k=0:K
    xk = x(k*L+1:k*L+N);
    Y(k+1,:) = circonvt(xk,h,N);
   end
   Y = Y(:,M:N)';                      % discard the first (M-1) samples
   y = (Y(:))';                        % assemble output
```

Note: The `ovrlpsav` function as developed here is not the most efficient approach. We will come back to this issue when we discuss the fast Fourier transform.

□ **EXAMPLE 5.19** To verify the operation of the `ovrlpsav` function, let us consider the sequences given in Example 5.18.

Solution MATLAB script:

```
>> n = 0:9; x = n+1; h = [1,0,-1]; N = 6;  y = ovrlpsav(x,h,N)
y =
     1     2     2     2     2     2     2     2     2     2    -9   -10
```

This is the correct linear convolution as expected. □

There is an alternate method called an *overlap-add* method of block convolutions. In this method the input sequence $x(n)$ is partitioned into nonoverlapping blocks and convolved with the impulse response. The resulting output blocks are overlapped with the subsequent sections and added to form the overall output. This is explored in Problem P5.32.

5.6 THE FAST FOURIER TRANSFORM

The DFT (5.24) introduced earlier is the only transform that is discrete in both the time and the frequency domains, and is defined for finite-duration sequences. Although it is a computable transform, the straightforward implementation of (5.24) is very inefficient, especially when the sequence length N is large. In 1965 Cooley and Tukey [1] showed a procedure to substantially reduce the amount of computations involved in the DFT. This led to the explosion of applications of the DFT, including in the digital signal processing area. Furthermore, it also led to the development of other efficient algorithms. All these efficient algorithms are collectively known as fast Fourier transform (FFT) algorithms.

Consider an N-point sequence $x(n)$. Its N-point DFT is given by (5.24) and reproduced here

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk}, \quad 0 \le k \le N-1 \tag{5.46}$$

where $W_N = e^{-j2\pi/N}$. To obtain one sample of $X(k)$, we need N complex multiplications and $(N-1)$ complex additions. Hence to obtain a complete set of DFT coefficients, we need N^2 complex multiplications and $N(N-1) \simeq N^2$ complex additions. Also one has to store N^2 complex coefficients $\{W_N^{nk}\}$ (or generate internally at an extra cost). Clearly, the number of DFT computations for an N-point sequence depends quadratically on N, which will be denoted by the notation

$$C_N = o\left(N^2\right)$$

For large N, $o\left(N^2\right)$ is unacceptable in practice. Generally, the processing time for one addition is much less than that for one multiplication. Hence from now on we will concentrate on the number of complex multiplications, which itself requires 4 real multiplications and 2 real additions.

Goal of an Efficient Computation In an efficiently designed algorithm the number of computations should be constant per data sample, and therefore the total number of computations should be linear with respect to N.

The quadratic dependence on N can be reduced by realizing that most of the computations (which are done again and again) can be eliminated using the periodicity property

$$W_N^{kn} = W_N^{k(n+N)} = W_N^{(k+N)n}$$

and the symmetry property

$$W_N^{kn+N/2} = -W_N^{kn}$$

of the factor $\{W_N^{nk}\}$.

One algorithm that considers only the periodicity of W_N^{nk} is the Goertzel algorithm. This algorithm still requires $C_N = o(N^2)$ multiplications, but it has certain advantages. This algorithm is described in Chapter 12. We first begin with an example to illustrate the advantages of the symmetry and periodicity properties in reducing the number of computations. We then describe and analyze two specific FFT algorithms that require $C_N = o(N \log N)$ operations. They are the *decimation-in-time* (DIT-FFT) and *decimation-in-frequency* (DIF-FFT) algorithms.

□ **EXAMPLE 5.20** Let us discuss the computations of a 4-point DFT and develop an efficient algorithm for its computation.

$$X(k) = \sum_{n=0}^{3} x(n) W_4^{nk}, \quad 0 \le k \le 3; \quad W_4 = e^{-j2\pi/4} = -j$$

Solution These computations can be done in the matrix form

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} W_4^0 & W_4^0 & W_4^0 & W_4^0 \\ W_4^0 & W_4^1 & W_4^2 & W_4^3 \\ W_4^0 & W_4^2 & W_4^4 & W_4^6 \\ W_4^0 & W_4^3 & W_4^6 & W_4^9 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix}$$

which requires 16 complex multiplications.

Efficient Approach Using periodicity,

$$\begin{aligned} W_4^0 &= W_4^4 = 1 \quad ; \; W_4^1 = W_4^9 = -j \\ W_4^2 &= W_4^6 = -1 \; ; \; W_4^3 = j \end{aligned}$$

and substituting in the above matrix form, we get

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix}$$

Using symmetry, we obtain

$$\begin{aligned}
X(0) &= x(0)+x(1)+x(2)+x(3) = [\underbrace{x(0)+x(2)}_{g_1}] + [\underbrace{x(1)+x(3)}_{g_2}]\\
X(1) &= x(0)-jx(1)-x(2)+jx(3) = [\underbrace{x(0)-x(2)}_{h_1}] - j[\underbrace{x(1)-x(3)}_{h_2}]\\
X(2) &= x(0)-x(1)+x(2)-x(3) = [\underbrace{x(0)+x(2)}_{g_1}] - [\underbrace{x(1)+x(3)}_{g_2}]\\
X(3) &= x(0)+jx(1)-x(2)-jx(3) = [\underbrace{x(0)-x(2)}_{h_1}] + j[\underbrace{x(1)-x(3)}_{h_2}]
\end{aligned}$$

Hence an efficient algorithm is

$$\begin{array}{r||l}
\text{Step 1} & \text{Step 2}\\
g_1 = x(0)+x(2) & X(0) = g_1 + g_2\\
g_2 = x(1)+x(3) & X(1) = h_1 - jh_2\\
h_1 = x(0)-x(2) & X(2) = g_1 - g_2\\
h_2 = x(1)-x(3) & X(3) = h_1 + jh_2
\end{array} \tag{5.47}$$

which requires only 2 complex multiplications, which is a considerably smaller number, even for this simple example. A signal flowgraph structure for this algorithm is given in Figure 5.19.

An Interpretation This efficient algorithm (5.47) can be interpreted differently. First, a 4-point sequence $x(n)$ is divided into two 2-point sequences, which are arranged into column vectors as shown here.

$$\left[\begin{bmatrix} x(0)\\ x(2)\end{bmatrix}, \begin{bmatrix} x(1)\\ x(3)\end{bmatrix}\right] = \begin{bmatrix} x(0) & x(1)\\ x(2) & x(3)\end{bmatrix}$$

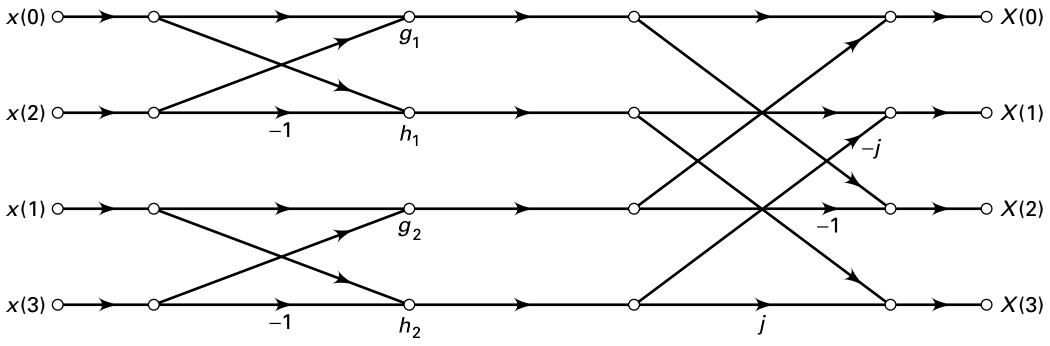


FIGURE 5.19 *Signal flowgraph in Example 5.20*

Second, a smaller 2-point DFT of each column is taken.

$$\begin{aligned}\mathbf{W}_2 \begin{bmatrix} x(0) & x(1) \\ x(2) & x(3) \end{bmatrix} &= \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x(0) & x(1) \\ x(2) & x(3) \end{bmatrix} \\ &= \begin{bmatrix} x(0)+x(2) & x(1)+x(3) \\ x(0)-x(2) & x(1)-x(3) \end{bmatrix} = \begin{bmatrix} g_1 & g_2 \\ h_1 & h_2 \end{bmatrix}\end{aligned}$$

Then each element of the resultant matrix is multiplied by $\{W_4^{pq}\}$, where p is the row index and q is the column index; that is, the following *dot-product* is performed:

$$\begin{bmatrix} 1 & 1 \\ 1 & -j \end{bmatrix} \cdot * \begin{bmatrix} g_1 & g_2 \\ h_1 & h_2 \end{bmatrix} = \begin{bmatrix} g_1 & g_2 \\ h_1 & -jh_2 \end{bmatrix}$$

Finally, two more smaller 2-point DFTs are taken of *row vectors*.

$$\begin{aligned}\begin{bmatrix} g_1 & g_2 \\ h_1 & -jh_2 \end{bmatrix} \mathbf{W}_2 &= \begin{bmatrix} g_1 & g_2 \\ h_1 & -jh_2 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} g_1+g_2 & g_1-g_2 \\ h_1-jh_2 & h_1+jh_2 \end{bmatrix} \\ &= \begin{bmatrix} X(0) & X(2) \\ X(1) & X(3) \end{bmatrix}\end{aligned}$$

Although this interpretation seems to have more multiplications than the efficient algorithm, it does suggest a systematic approach of computing a larger DFT based on smaller DFTs. □

5.6.1 DIVIDE-AND-COMBINE APPROACH

To reduce the DFT computation's quadratic dependence on N, one must choose a composite number $N = LM$ since

$$L^2 + M^2 \ll N^2 \quad \text{for large } N$$

Now divide the sequence into M smaller sequences of length L, compute M smaller L-point DFTs, and then combine these into a larger DFT using L smaller M-point DFTs. This is the essence of the divide-and-combine approach. Let $N = LM$, then the indices n and k in (5.46) can be written as

$$\begin{aligned} n &= \ell + Lm,\ 0 \le \ell \le L-1,\ 0 \le m \le M-1 \\ k &= q + Mp,\ 0 \le p \le L-1,\ 0 \le q \le M-1 \end{aligned} \tag{5.48}$$

and write sequences $x(n)$ and $X(k)$ as arrays $x(\ell,m)$ and $X(p,q)$, respectively. Then (5.46) can be written as

$$\begin{aligned}
X(p,q) &= \sum_{\ell=0}^{L-1}\sum_{m=0}^{M-1} x(\ell,m) W_N^{(\ell+Lm)(q+Mp)} \\
&= \sum_{\ell=0}^{L-1}\left\{W_N^{\ell q}\left[\sum_{m=0}^{M-1} x(\ell,m) W_N^{Lmq}\right]\right\} W_N^{M\ell p} \\
&= \underbrace{\sum_{\ell=0}^{L-1}\left\{W_N^{\ell q}\underbrace{\left[\sum_{m=0}^{M-1} x(\ell,m) W_M^{mq}\right]}_{M\text{-point DFT}}\right\} W_L^{\ell p}}_{L\text{-point DFT}}
\end{aligned} \tag{5.49}$$

Hence (5.49) can be implemented as a three-step procedure:

1. First, we compute the M-point DFT array

$$F(\ell,q) \triangle \sum_{m=0}^{M-1} x(\ell,m) W_M^{mq}; \quad 0 \le q \le M-1 \tag{5.50}$$

for each of the rows $\ell = 0,\ldots,L-1$.

2. Second, we modify $F(\ell,q)$ to obtain another array.

$$G(\ell,q) = W_N^{\ell q} F(\ell,q), \quad \begin{matrix} 0 \le \ell \le L-1 \\ 0 \le q \le M-1 \end{matrix} \tag{5.51}$$

The factor $W_N^{\ell q}$ is called a *twiddle* factor.

3. Finally, we compute the L-point DFTs

$$X(p,q) = \sum_{\ell=0}^{L-1} G(\ell,q) W_L^{\ell p} \quad 0 \le p \le L-1 \tag{5.52}$$

for each of the columns $q = 0,\ldots,M-1$.

The total number of complex multiplications for this approach can now be given by

$$C_N = LM^2 + N + ML^2 < o\left(N^2\right) \tag{5.53}$$

We illustrate this approach in the following example.

□ **EXAMPLE 5.21** Develop the divide-and-combine FFT algorithm for $N = 15$.

Solution Let $L = 3$ and $M = 5$. Then, from (5.48), we have

$$\begin{aligned} n &= \ell + 3M, && 0 \le \ell \le 2, && 0 \le m \le 4 \\ k &= q + 5p, && 0 \le p \le 2, && 0 \le q \le 4 \end{aligned} \tag{5.54}$$

Hence (5.49) becomes

$$X(p,q) = \sum_{\ell=0}^{2} \left\{ W_{15}^{\ell q} \left[\sum_{m=0}^{4} x(\ell,m) W_5^{mq} \right] \right\} W_3^{\ell p} \tag{5.55}$$

To implement (5.55), we arrange the given sequence $x(n)$ in the form of an array $\{x(\ell,m)\}$ using a column-wise ordering as

$$\begin{array}{ccccc} x(0) & x(3) & x(6) & x(9) & x(12) \\ x(1) & x(4) & x(7) & x(10) & x(13) \\ x(2) & x(5) & x(8) & x(11) & x(14) \end{array} \tag{5.56}$$

The first step is to compute 5-point DFTs $F(\ell,q)$ for each of the three rows and arrange them back in the same array formation

$$\begin{array}{ccccc} F(0,0) & F(0,1) & F(0,2) & F(0,3) & F(0,4) \\ F(1,0) & F(1,1) & F(1,2) & F(1,3) & F(1,4) \\ F(2,0) & F(2,1) & F(2,2) & F(2,3) & F(2,4) \end{array} \tag{5.57}$$

which requires a total of $3 \times 5^2 = 75$ complex operations. The second step is to modify $F(\ell,q)$ to obtain the array $G(\ell,q)$ using the twiddle factors $W_{15}^{\ell q}$

$$\begin{array}{ccccc} G(0,0) & G(0,1) & G(0,2) & G(0,3) & G(0,4) \\ G(1,0) & G(1,1) & G(1,2) & G(1,3) & G(1,4) \\ G(2,0) & G(2,1) & G(2,2) & G(2,3) & G(2,4) \end{array} \tag{5.58}$$

which requires 15 complex operations. The last step is to perform 3-point DFTs $X(p,q)$ for each of the five columns to obtain

$$\begin{array}{ccccc} X(0,0) & X(0,1) & X(0,2) & X(0,3) & X(0,4) \\ X(1,0) & X(1,1) & X(1,2) & X(1,3) & X(1,4) \\ X(2,0) & X(2,1) & X(2,2) & X(2,3) & X(2,4) \end{array} \tag{5.59}$$

using a total of $5 \times 3^2 = 45$ complex operations. According to (5.54) the array in (5.59) is a rearrangement of $X(k)$ as

$$\begin{array}{ccccc} X(0) & X(1) & X(2) & X(3) & X(4) \\ X(5) & X(6) & X(7) & X(8) & X(9) \\ X(10) & X(11) & X(12) & X(13) & X(14) \end{array} \tag{5.60}$$

Finally, after "unwinding" this array in the row-wise fashion, we obtain the required 15-point DFT $X(k)$. The total number of complex operations required for this divide-and-combine approach is 135, whereas the direct approach for the 15-point DFT requires 225 complex operations. Thus the divide-and-combine approach is clearly efficient. □

The divide-and-combine procedure can be further repeated if M or L are composite numbers. Clearly, the most efficient algorithm is obtained when N is a highly composite number, that is, $N = R^{\nu}$. Such algorithms are called *radix-R* FFT algorithms. When $N = R_1^{\nu_1} R_2^{\nu_2} \ldots$, then such decompositions are called *mixed-radix* FFT algorithms. The one most popular and easily programmable algorithm is the radix-2 FFT algorithm.

5.6.2 RADIX-2 FFT ALGORITHM

Let $N = 2^{\nu}$; then we choose $L = 2$ and $M = N/2$ and divide $x(n)$ into two $N/2$-point sequences according to (5.48) as

$$\begin{matrix} g_1(n) = x(2n) \\ g_2(n) = x(2n+1) \end{matrix}; \quad 0 \le n \le \frac{N}{2} - 1$$

The sequence $g_1(n)$ contains even-ordered samples of $x(n)$, while $g_2(n)$ contains odd-ordered samples of $x(n)$. Let $G_1(k)$ and $G_2(k)$ be $N/2$-point DFTs of $g_1(n)$ and $g_2(n)$, respectively. Then (5.49) reduces to

$$X(k) = G_1(k) + W_N^k G_2(k), \quad 0 \le k \le N-1 \tag{5.61}$$

This is called a *merging formula*, which combines two $N/2$-point DFTs into one N-point DFT. The total number of complex multiplications reduces to

$$C_N = \frac{N^2}{2} + N = o\left(N^2/2\right)$$

This procedure can be repeated again and again. At each stage the sequences are decimated and the smaller DFTs combined. This decimation ends after ν stages when we have N one-point sequences, which are also one-point DFTs. The resulting procedure is called the *decimation-in-time* FFT (DIT-FFT) algorithm, for which the total number of complex multiplications is

$$C_N = N\nu = N \log_2 N$$

Clearly, if N is large, then C_N is approximately linear in N, which was the goal of our efficient algorithm. Using additional symmetries, C_N can be reduced to $\frac{N}{2}\log_2 N$. The signal flowgraph for this algorithm is shown in Figure 5.20 for $N = 8$.

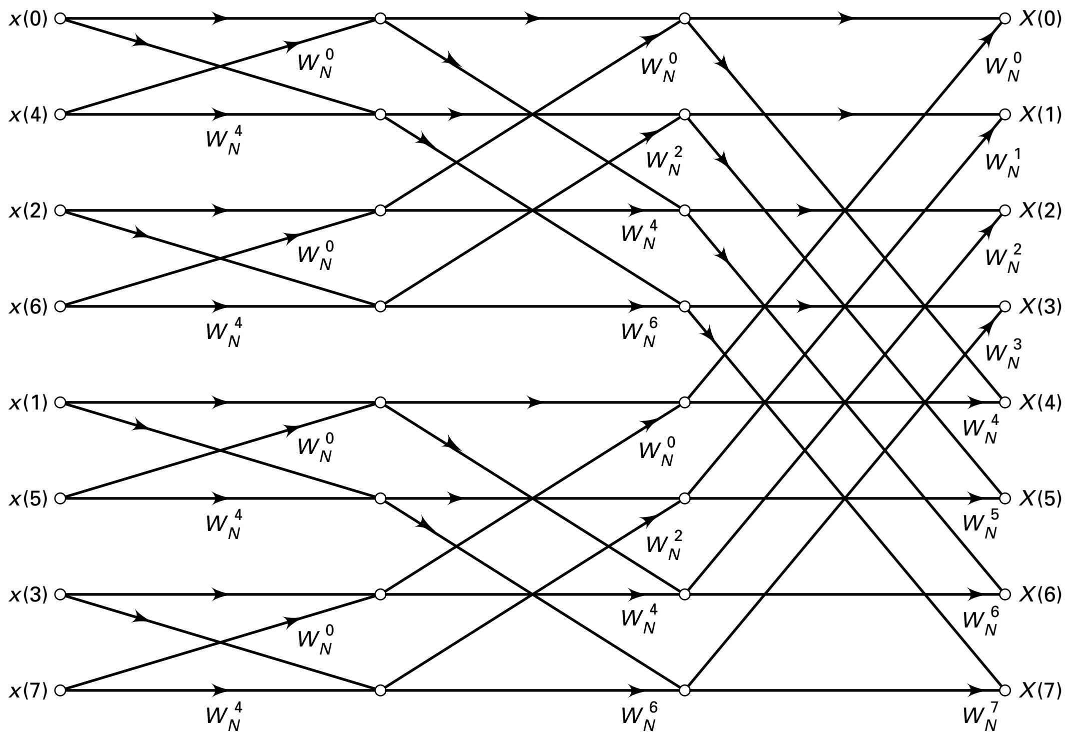


FIGURE 5.20 *Decimation-in-time FFT structure for $N = 8$*

In an alternate approach we choose $M = 2$, $L = N/2$ and follow the steps in (5.49). Note that the initial DFTs are 2-point DFTs, which contain no complex multiplications. From (5.50)

$$\begin{aligned} F(0,m) &= x(0,m) + x(1,m)W_2^0 \\ &= x(n) + x(n+N/2), \quad 0 \le n \le N/2 \\ F(1,m) &= x(0,m) + x(1,m)W_2^1 \\ &= x(n) - x(n+N/2), \quad 0 \le n \le N/2 \end{aligned}$$

and from (5.51)

$$\begin{aligned} G(0,m) &= F(0,m)W_N^0 \\ &= x(n) + x(n+N/2), \quad 0 \le n \le N/2 \\ G(1,m) &= F(1,m)W_N^m \\ &= [x(n) - x(n+N/2)]\, W_N^n, \quad 0 \le n \le N/2 \end{aligned} \tag{5.62}$$

Let $G(0,m) = d_1(n)$ and $G(1,m) = d_2(n)$ for $0 \le n \le N/2 - 1$ (since they can be considered as time-domain sequences); then from (5.52) we have

$$\begin{aligned} X(0,q) &= X(2q) &&= D_1(q) \\ X(1,q) &= X(2q+1) &&= D_2(q) \end{aligned} \tag{5.63}$$

This implies that the DFT values $X(k)$ are computed in a decimated fashion. Therefore this approach is called a *decimation-in-frequency* FFT (DIF-FFT) algorithm. Its signal flowgraph is a transposed structure of the DIT-FFT structure, and its computational complexity is also equal to $\frac{N}{2}\log_2 N$.

5.6.3 MATLAB IMPLEMENTATION

MATLAB provides a function called `fft` to compute the DFT of a vector `x`. It is invoked by `X = fft(x,N)`, which computes the N-point DFT. If the length of `x` is less than `N`, then `x` is padded with zeros. If the argument `N` is omitted, then the length of the DFT is the length of `x`. If `x` is a matrix, then `fft(x,N)` computes the N-point DFT of each column of `x`.

This `fft` function is written in machine language and not using MATLAB commands (i.e., it is not available as a `.m` file). Therefore it executes very fast. It is written as a mixed-radix algorithm. If N is a power of two, then a high-speed radix-2 FFT algorithm is employed. If N is not a power of two, then N is decomposed into prime factors and a slower mixed-radix FFT algorithm is used. Finally, if N is a prime number, then the `fft` function is reduced to the raw DFT algorithm.

The inverse DFT is computed using the `ifft` function, which has the same characteristics as `fft`.

□ **EXAMPLE 5.22** In this example we will study the execution time of the `fft` function for $1 \le N \le 2048$. This will reveal the divide-and-combine strategy for various values of N. One note of caution. The results obtained in this example are valid only for MATLAB Versions 5 and earlier. Beginning in Version 6, MATLAB is using a new numerical computing core called LAPACK. It is optimized for memory references and cache usage and not for individual floating-point operations. Therefore, results for Version 6 and later are difficult to interpret. Also the execution times given here are for a specific computer and may vary on different computers.

Solution To determine the execution time, MATLAB provides two functions. The `clock` function provides the instantaneous clock reading, while the `etime(t1,t2)` function computes the elapsed time between two time marks `t1` and `t2`. To determine the execution time, we will generate random vectors from length 1 through 2048, compute their FFTs, and save the computation time in an array. Finally, we will plot this execution time versus N.

MATLAB script:

```
>> Nmax = 2048;  fft_time=zeros(1,Nmax);
>> for n=1:1:Nmax
>>    x=rand(1,n);
>>    t=clock;fft(x);fft_time(n)=etime(clock,t);
>> end
>> n=[1:1:Nmax];   plot(n,fft_time,'.')
>> xlabel('N');ylabel('Time in Sec.')  title('FFT execution times')
```

The plot of the execution times is shown in Figure 5.21. This plot is very informative. The points in the plot do not show one clear function but appear to group themselves into various trends. The uppermost group depicts a $o(N^2)$ dependence on N, which means that these values must be prime numbers between 1 and 2048 for which the FFT algorithm defaults to the DFT algorithm. Similarly, there are groups corresponding to the $o\left(N^2/2\right)$, $o\left(N^2/3\right)$, $o\left(N^2/4\right)$, and so on, dependencies for which the number N has fewer decompositions. The last group shows the (almost linear) $o\left(N \log N\right)$ dependence, which is for

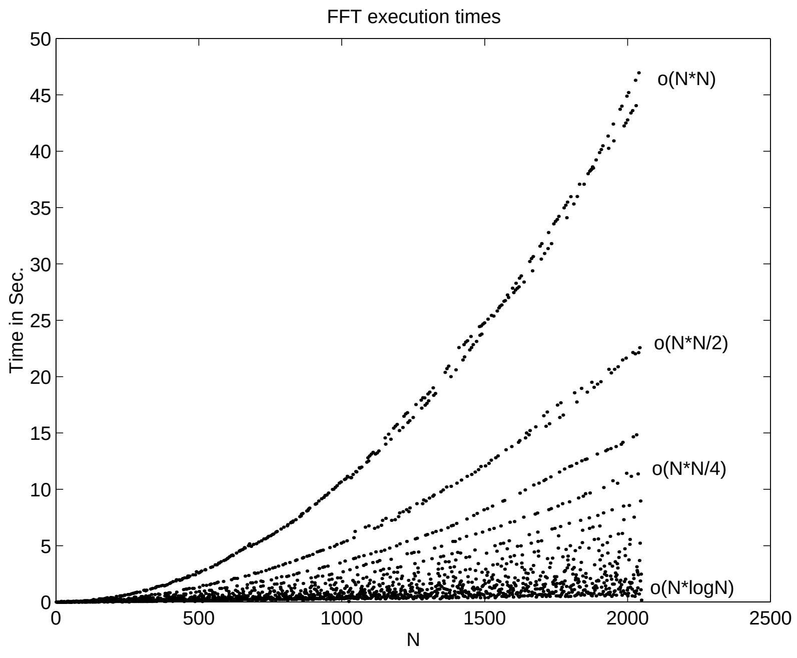


FIGURE 5.21 *FFT execution times for* $1 <= N <= 2048$

$N = 2^{\nu}, 0 \le \nu \le 11$. For these values of N, the radix-2 FFT algorithm is used. For all other values, a mixed-radix FFT algorithm is employed. This shows that the divide-and-combine strategy is very effective when N is highly composite. For example, the execution time is 0.16 sec for $N = 2048$, 2.48 sec for $N = 2047$, and 46.96 sec for $N = 2039$. □

The MATLAB functions developed previously in this chapter should now be modified by substituting the `fft` function in place of the `dft` function. From the preceding example care must be taken to use a highly composite N. A good practice is to choose $N = 2^{\nu}$ unless a specific situation demands otherwise.

5.6.4 FAST CONVOLUTIONS

The `conv` function in MATLAB is implemented using the `filter` function (which is written in C) and is very efficient for smaller values of N (< 50). For larger values of N it is possible to speed up the convolution using the FFT algorithm. This approach uses the circular convolution to implement the linear convolution, and the FFT to implement the circular convolution. The resulting algorithm is called a *fast convolution* algorithm. In addition, if we choose $N = 2^{\nu}$ and implement the radix-2 FFT, then the algorithm is called a *high-speed convolution*. Let $x_1(n)$ be a N_1-point sequence and $x_2(n)$ be a N_2-point sequence; then for high-speed convolution N is chosen to be

$$N = 2^{\lceil \log_2 (N_1 + N_2 - 1) \rceil} \tag{5.64}$$

where $\lceil x \rceil$ is the smallest integer greater than x (also called a *ceiling* function). The linear convolution $x_1(n) * x_2(n)$ can now be implemented by two N-point FFTs, one N-point IFFT, and one N-point dot-product.

$$x_1(n) * x_2(n) = \text{IFFT}\left[\text{FFT}\left[x_1(n)\right] \cdot \text{FFT}\left[x_2(n)\right]\right] \tag{5.65}$$

For large values of N, (5.65) is faster than the time-domain convolution, as we see in the following example.

□ **EXAMPLE 5.23** To demonstrate the effectiveness of the high-speed convolution, let us compare the execution times of two approaches. Let $x_1(n)$ be an L-point uniformly distributed random number between $[0, 1]$, and let $x_2(n)$ be an L-point Gaussian random sequence with mean 0 and variance 1. We will determine the average execution times for $1 \le L \le 150$, in which the average is computed over the 100 realizations of random sequences. (Please see the cautionary note given in Example 5.22.)

Solution MATLAB script:

```
conv_time = zeros(1,150); fft_time = zeros(1,150);
%
for L = 1:150
    tc = 0; tf=0;
   N = 2*L-1; nu = ceil(log10(NI)/log10(2)); N = 2^nu;
    for I=1:100
       h = randn(1,L);  x = rand(1,L);
      t0 = clock; y1 = conv(h,x); t1=etime(clock,t0);  tc = tc+t1;
      t0 = clock; y2 = ifft(fft(h,N).*fft(x,N)); t2=etime(clock,t0);
      tf = tf+t2;
    end
%
    conv_time(L)=tc/100;  fft_time(L)=tf/100;
end
%
n = 1:150; subplot(1,1,1);
plot(n(25:150),conv_time(25:150),n(25:150),fft_time(25:150))
```

Figure 5.22 shows the linear convolution and the high-speed convolution times for $25 \leq L \leq 150$. It should be noted that these times are affected by the

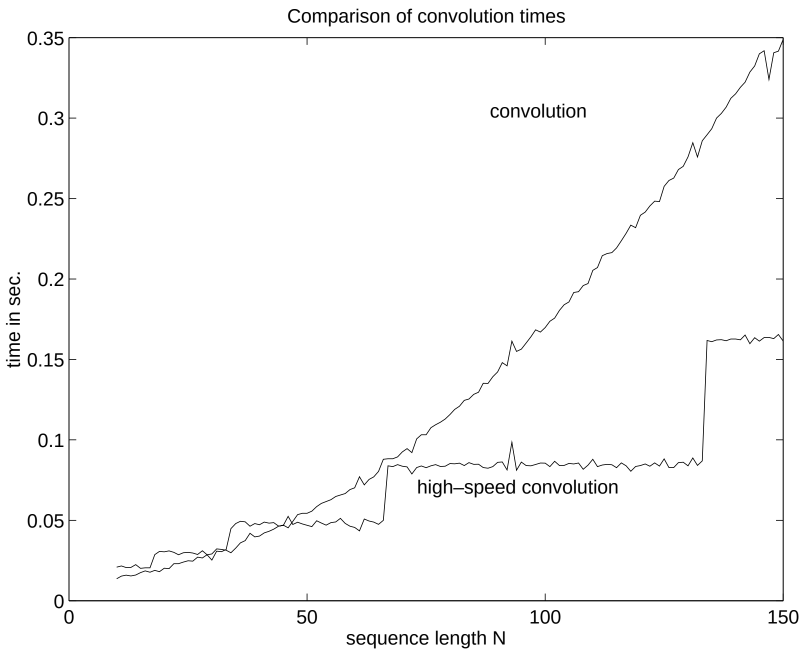


FIGURE 5.22 *Comparison of linear and high-speed convolution times*

computing platform used to execute the MATLAB script. The plot in Figure 5.22 was obtained on a 33 MHz 486 computer. It shows that for low values of L the linear convolution is faster. The crossover point appears to be $L = 50$, beyond which the linear convolution time increases exponentially, while the high-speed convolution time increases fairly linearly. Note that since $N = 2^{\nu}$, the high-speed convolution time is constant over a range on L. □

5.6.5 HIGH-SPEED BLOCK CONVOLUTIONS

Earlier we discussed a block convolution algorithm called the overlap-and-save method (and its companion the overlap-and-add method), which is used to convolve a very large sequence with a relatively smaller sequence. The MATLAB function `ovrlpsav` developed in that section uses the DFT to implement the linear convolution. We can now replace the DFT by the radix-2 FFT algorithm to obtain a *high-speed* overlap-and-save algorithm. To further reduce the computations, the FFT of the shorter (fixed) sequence can be computed only once. The following `hsolpsav` function shows this algorithm.

```
function [y] =  hsolpsav(x,h,N)
% High-speed Overlap-Save method of block convolutions using FFT
% ----------------------------------------------------------------
% [y] = hsolpsav(x,h,N)
% y = output sequence
% x = input sequence
% h = impulse response
% N = block length (must be a power of two)
%
N = 2^(ceil(log10(N)/log10(2));
Lenx = length(x); M = length(h);
M1 = M-1; L = N-M1;  h = fft(h,N);
%
x = [zeros(1,M1), x, zeros(1,N-1)];
K = floor((Lenx+M1-1)/(L)); % # of blocks
Y = zeros(K+1,N);
for k=0:K
xk = fft(x(k*L+1:k*L+N));
Y(k+1,:) = real(ifft(xk.*h));
end
Y = Y(:,M:N)'; y = (Y(:))';
```

A similar modification can be done to the overlap-and-add algorithm. MATLAB also provides the function `fftfilt` to implement the overlap-and-add algorithm.

5.7 PROBLEMS

P5.1 Compute the DFS coefficients of the following periodic sequences using the DFS definition, and then verify your answers using MATLAB.

1. $\tilde{x}_1(n) = \{4, 1, -1, 1\}$, $N = 4$
2. $\tilde{x}_2(n) = \{2, 0, 0, 0, -1, 0, 0, 0\}$, $N = 8$
3. $\tilde{x}_3(n) = \{1, 0, -1, -1, 0\}$, $N = 5$
4. $\tilde{x}_4(n) = \{0, 0, 2j, 0, 2j, 0\}$, $N = 6$
5. $\tilde{x}_5(n) = \{3, 2, 1\}$, $N = 3$

P5.2 Determine the periodic sequences given the following periodic DFS coefficients. First use the IDFS definition and then verify your answers using MATLAB.

1. $\tilde{X}_1(k) = \{4, 3j, -3j\}$, $N = 3$
2. $\tilde{X}_2(k) = \{j, 2j, 3j, 4j\}$, $N = 4$
3. $\tilde{X}_3(k) = \{1, 2 + 3j, 4, 2 - 3j\}$, $N = 4$
4. $\tilde{X}_4(k) = \{0, 0, 2, 0, 0\}$, $N = 5$
5. $\tilde{X}_5(k) = \{3, 0, 0, 0, -3, 0, 0, 0\}$, $N = 8$

P5.3 Let $\tilde{x}_1(n)$ be periodic with fundamental period $N = 40$ where one period is given by

$$\tilde{x}_1(n) = \begin{cases} 5\sin(0.1\pi n), & 0 \le n \le 19 \\ 0, & 20 \le n \le 39 \end{cases}$$

and let $\tilde{x}_2(n)$ be periodic with fundamental period $N = 80$, where one period is given by

$$\tilde{x}_2(n) = \begin{cases} 5\sin(0.1\pi n), & 0 \le n \le 19 \\ 0, & 20 \le n \le 79 \end{cases}$$

These two periodic sequences differ in their periodicity but otherwise have the same nonzero samples.

1. Compute the DFS $\tilde{X}_1(k)$ of $\tilde{x}_1(n)$, and plot samples (using the `stem` function) of its magnitude and angle versus k.
2. Compute the DFS $\tilde{X}_2(k)$ of $\tilde{x}_2(n)$, and plot samples of its magnitude and angle versus k.
3. What is the difference between the two preceding DFS plots?

P5.4 Consider the periodic sequence $\tilde{x}_1(n)$ given in Problem P5.3. Let $\tilde{x}_2(n)$ be periodic with fundamental period $N = 40$, where one period is given by

$$\tilde{x}_2(n) = \begin{cases} \tilde{x}_1(n), & 0 \le n \le 19 \\ -\tilde{x}_1(n-20), & 20 \le n \le 39 \end{cases}$$

1. Determine analytically the DFS $\tilde{X}_2(k)$ in terms of $\tilde{X}_1(k)$.
2. Compute the DFS $\tilde{X}_2(k)$ of $\tilde{x}_2(n)$ and plot samples of its magnitude and angle versus k.
3. Verify your answer in part 1 using the plots of $\tilde{X}_1(k)$ and $\tilde{X}_2(k)$?

P5.5 Consider the periodic sequence $\tilde{x}_1(n)$ given in Problem P5.3. Let $\tilde{x}_3(n)$ be periodic with period 80, obtained by concatenating two periods of $\tilde{x}_1(n)$, i.e.,

$$\tilde{x}_3(n) = [\tilde{x}_1(n), \tilde{x}_1(n)]_{\text{PERIODIC}}$$

Clearly, $\tilde{x}_3(n)$ is different from $\tilde{x}_2(n)$ of Problem P5.3 even though both of them are periodic with period 80.

1. Compute the DFS $\tilde{X}_3(k)$ of $\tilde{x}_3(n)$, and plot samples of its magnitude and angle versus k.
2. What effect does the periodicity doubling have on the DFS?
3. Generalize this result to M-fold periodicity. In particular, show that if

$$\tilde{x}_M(n) = \Big[\underbrace{\tilde{x}_1(n), \tilde{x}_1(n), \ldots, \tilde{x}_1(n)}_{M \text{ times}}\Big]_{\text{PERIODIC}}$$

then

$$\begin{aligned} \tilde{X}_M(Mk) &= M\tilde{X}_1(k), & k &= 0, 1, \ldots, N-1 \\ \tilde{X}_M(k) &= 0, & k &\neq 0, M, \ldots, MN \end{aligned}$$

P5.6 Let $X(e^{j\omega})$ be the DTFT of a finite-length sequence

$$x(n) = \begin{cases} n+1, & 0 \le n \le 49; \\ 100-n, & 50 \le n \le 99; \\ 0, & \text{otherwise.} \end{cases}$$

1. Let

$$y_1(n) = \overset{10\text{-point}}{\text{IDFS}} \left[X(e^{j0}), X(e^{j2\pi/10}), X(e^{j4\pi/10}), \ldots, X(e^{j18\pi/10})\right]$$

Determine $y_1(n)$ using the frequency sampling theorem. Verify your answer using MATLAB.

2. Let

$$y_2(n) = \overset{200\text{-point}}{\text{IDFS}} \left[X(e^{j0}), X(e^{j2\pi/200}), X(e^{j4\pi/200}), \ldots, X(e^{j398\pi/200})\right]$$

Determine $y_2(n)$ using the frequency sampling theorem. Verify your answer using MATLAB.

3. Comment on your results in parts (a) and (b).

P5.7 Let $\tilde{x}(n)$ be a periodic sequence with period N and let

$$\tilde{y}(n) \triangleq \tilde{x}(-n) = \tilde{x}(N-n)$$

that is, $\tilde{y}(n)$ is a periodically folded version of $\tilde{x}(n)$. Let $\tilde{X}(k)$ and $\tilde{Y}(k)$ be the DFS sequences.

1. Show that

$$\tilde{Y}(k) = \tilde{X}(-k) = \tilde{X}(N-k)$$

that is, $\tilde{Y}(k)$ is also a periodically folded version of $\tilde{X}(k)$.

2. Let $\tilde{x}(n) = \{\underset{\uparrow}{2}, 4, 6, 1, 3, 5\}_{\text{PERIODIC}}$ with $N = 6$.

(a) *Sketch* $\tilde{y}(n)$ for $0 \le n \le 5$.
(b) Compute $\tilde{X}(k)$ for $0 \le k \le 5$.
(c) Compute $\tilde{Y}(k)$ for $0 \le k \le 5$.
(d) Verify the relation in part 1.

P5.8 Consider the following finite-length sequence.

$$x(n) = \begin{cases} \text{sinc}^2\{(n-50)/2\}, & 0 \le n \le 100; \\ 0, & \text{else.} \end{cases}$$

1. Determine the DFT $X(k)$ of $x(n)$. Plot (using the `stem` function) its magnitude and phase.
2. Plot the magnitude and phase of the DTFT $X(e^{j\omega})$ of $x(n)$ using MATLAB.
3. Verify that the above DFT is the sampled version of $X(e^{j\omega})$. It might be helpful to combine the above two plots in one graph using the `hold` function.
4. Is it possible to reconstruct the DTFT $X(e^{j\omega})$ from the DFT $X(k)$? If possible, give the necessary interpolation formula for reconstruction. If not possible, state why this reconstruction cannot be done.

P5.9 Let a finite-length sequence be given by

$$x(n) = \begin{cases} 2e^{-0.9|n|}, & -5 \le n \le 5; \\ 0, & \text{otherwise.} \end{cases}$$

Plot the DTFT $X(e^{j\omega})$ of the above sequence using DFT as a computation tool. Choose the length N of the DFT so that this plot appears to be a smooth graph.

P5.10 Plot the DTFT magnitude and angle of each of the following sequences using the DFT as a computation tool. Make an educated guess about the length N so that your plots are meaningful.

1. $x(n) = (0.6)^{|n|}\left[u(n+10) - u(n-11)\right]$.
2. $x(n) = n(0.9)^n\left[u(n) - u(n-21)\right]$.
3. $x(n) = [\cos(0.5\pi n) + j\sin(0.5\pi n)][u(n) - u(n-51)]$.
4. $x(n) = \{1, 2, 3, \underset{\uparrow}{4}, 3, 2, 1\}$.
5. $x(n) = \{-1, -2, -3, \underset{\uparrow}{0}, 3, 2, 1\}$.

P5.11 Let $H(e^{j\omega})$ be the frequency response of a real, causal discrete-time LSI system.

1. If

$$\text{Re}\left\{H\left(e^{j\omega}\right)\right\} = \sum_{k=0}^{5} (0.9)^k \cos(k\omega)$$

determine the impulse response $h(n)$ analytically. Verify your answer using DFT as a computation tool. Choose the length N appropriately.

2. If

$$\text{Im}\left\{H\left(e^{j\omega}\right)\right\} = \sum_{\ell=0}^{5} 2\ell\sin(\ell\omega), \quad \text{and} \quad \int_{-\pi}^{\pi} H(e^{j\omega})d\omega = 0$$

determine the impulse response $h(n)$ analytically. Verify your answer using DFT as a computation tool. Again choose the length N appropriately.

P5.12 Let $X(k)$ denote the N-point DFT of an N-point sequence $x(n)$. The DFT $X(k)$ itself is an N-point sequence.

1. If the DFT of $X(k)$ is computed to obtain another N-point sequence $x_1(n)$, show that
$$x_1(n) = Nx((-n))_N, \quad 0 \le n \le N-1$$
2. Using this property, design a MATLAB function to implement an N-point circular folding operation $x_2(n) = x_1((-n))_N$. The format should be

```
x2 = circfold(x1,N)
% Circular folding using DFT
% x2 = circfold(x1,N)
% x2 = circularly folded output sequence
% x1 = input sequence of length <= N
%  N = circular buffer length
```

3. Determine the circular folding of the following sequence.
$$x_1(n) = \{1, 3, 5, 7, 9, -7, -5, -3, -1\}$$

P5.13 Let $X(k)$ be an N-point DFT of an N-point sequence $x(n)$. Let N be an even integer.

1. If $x(n) = x(n + N/2)$ for all n, then show that $X(k) = 0$ for k odd (i.e., nonzero for k even). Verify this result for $x(n) = \{1, 2, -3, 4, 5, 1, 2, -3, 4, 5\}$.
2. If $x(n) = -x(n + N/2)$ for all n, then show that $X(k) = 0$ for k even (i.e., nonzero for k odd). Verify this result for $x(n) = \{1, 2, -3, 4, 5, -1, -2, 3, -4, -5\}$.

P5.14 Let $X(k)$ be an N-point DFT of an N-point sequence $x(n)$. Let $N = 4\nu$ where ν is an integer.

1. If $x(n) = x(n + \nu)$ for all n, then show that $X(k)$ is nonzero for $k = 4\ell$ for $0 \le \ell \le \nu - 1$. Verify this result for $x(n) = \{1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3\}$.
2. If $x(n) = -x(n + \nu)$ for all n, then show that $X(k)$ is nonzero for $k = 4\ell + 2$ for $0 \le \ell \le \nu - 1$. Verify this result for $x(n) = \{1, 2, 3, -1, -2, -3, 1, 2, 3, -1, -2, -3\}$.

P5.15 Let $X(k)$ be an N-point DFT of an N-point sequence $x(n)$. Let $N = 2\mu\nu$ where μ and ν are integers.

1. If $x(n) = x(n + \nu)$ for all n, then show that $X(k)$ is nonzero for $k = 2(\mu\ell)$ for $0 \le \ell \le \nu - 1$. Verify this result for $x(n) = \{1, -2, 3, 1, -2, 3, 1, -2, 3, 1, -2, 3, 1, -2, 3, 1, -2, 3\}$.
2. If $x(n) = -x(n + \nu)$ for all n, then show that $X(k)$ is nonzero for $k = 2(\mu\ell + 1)$ for $0 \le \ell \le \nu - 1$. Verify this result for $x(n) = \{1, -2, 3, -1, 2, -3, 1, -2, 3, -1, 2, -3, 1, -2, 3, -1, 2, -3\}$.

P5.16 Let $X(k)$ and $Y(k)$ be 10-point DFTs of two 10-point sequences $x(n)$ and $y(n)$, respectively. If
$$X(k) = \exp(j0.2\pi k), \quad 0 \le k \le 9$$
determine $Y(k)$ in each of the following cases without computing the DFT.

1. $y(n) = x((n-5))_{10}$
2. $y(n) = x((n+4))_{10}$
3. $y(n) = x((3-n))_{10}$
4. $y(n) = x(n)e^{j3\pi n/5}$
5. $y(n) = x(n) \textcircled{10} x((-n))_{10}$

Verify your answers using MATLAB.

P5.17 The first six values of the 10-point DFT of a real-valued sequence $x(n)$ are given by

$$\{10, -2+j3, 3+j4, 2-j3, 4+j5, 12\}$$

Determine the DFT of each of the following sequences using DFT properties.

1. $x_1(n) = x((2-n))_{10}$
2. $x_2(n) = x((n+5))_{10}$
3. $x_3(n) = x(n)x((-n))_{10}$
4. $x_4(n) = x(n) \textcircled{10} x((-n))_{10}$
5. $x_5(n) = x(n)e^{-j4\pi n/5}$

P5.18 Complex-valued N-point sequence $x(n)$ can be decomposed into N-point circular-conjugate-symmetric and circular-conjugate-antisymmetric sequences using the following relations

$$x_{\text{ccs}}(n) \overset{\triangle}{=} \frac{1}{2}\left[x(n) + x^*((-n))_N\right]$$

$$x_{\text{cca}}(n) \overset{\triangle}{=} \frac{1}{2}\left[x(n) - x^*((-n))_N\right]$$

If $X_{\text{R}}(k)$ and $X_{\text{I}}(k)$ are the real and imaginary parts of the N-point DFT of $x(n)$, then

$$\text{DFT}\left[x_{\text{ccs}}(n)\right] = X_{\text{R}}(k) \quad \text{and} \quad \text{DFT}\left[x_{\text{cca}}(n)\right] = jX_{\text{I}}(k)$$

1. Prove these relations property analytically.
2. Modify the `circevod` function developed in the chapter so that it can be used for complex-valued sequences.
3. Let $X(k) = [3\cos(0.2\pi k) + j4\sin(0.1\pi k)][u(k) - u(k-20)]$ be a 20-point DFT. Verify this symmetry property using your `circevod` function.

P5.19 If $X(k)$ is the N-point DFT of an N-point complex-valued sequence

$$x(n) = x_{\text{R}}(n) + jx_{\text{I}}(n)$$

where $x_{\text{R}}(n)$ and $x_{\text{I}}(n)$ are the real and imaginary parts of $x(n)$, then

$$\text{DFT}\left[x_{\text{R}}(n)\right] = X_{\text{ccs}}(k) \quad \text{and} \quad \text{DFT}\left[jx_{\text{I}}(n)\right] = X_{\text{cca}}(k)$$

where $X_{\text{ccs}}(k)$ and $X_{\text{cca}}(k)$ are the circular-even and circular-odd components of $X(k)$ as defined in Problem P5.18.

1. Prove this property analytically.
2. This property can be used to compute the DFTs of two real-valued N-point sequences using one N-point DFT operation. Specifically, let $x_1(n)$ and $x_2(n)$ be two N-point sequences. Then we can form a complex-valued sequence

$$x(n) = x_1(n) + jx_2(n)$$

and use this property. Develop a MATLAB function to implement this approach with the following format.

```
function [X1,X2] = real2dft(x1,x2,N)
% DFTs of two real sequences
% [X1,X2] = real2dft(x1,x2,N)
%  X1 = n-point DFT of x1
%  X2 = n-point DFT of x2
%  x1 = sequence of length <= N
%  x2 = sequence of length <= N
%   N = length of DFT
```

3. Compute and plot the DFTs of the following two sequences using this function.

$$x_1(n) = \cos(0.1\pi n),\ x_2(n) = \sin(0.2\pi n); \quad 0 \le n \le 39$$

P5.20 Using the frequency domain approach, devise a MATLAB function to determine a circular shift $x((n-m))_N$, given an N_1-point sequence $x(n)$ where $N_1 \le N$. Your function should have the following format.

```
function y = cirshftf(x,m,N)
%  Circular shift of m samples wrt size N in sequence x: (freq domain)
%  -------------------------------------------------------------------
%  y = cirshftf(x,m,N)
%      y : output sequence containing the circular shift
%      x : input sequence of length <= N
%      m : sample shift
%      N : size of circular buffer
%   Method: y(n) = idft(dft(x(n))*WN^(mk))
%
%   If m is a scalar then y is a sequence (row vector)
%   If m is a vector then y is a matrix, each row is a circular shift
%       in x corresponding to entries in vecor m
%   M and x should not be matrices
```

Verify your function on the following sequence

$$x(n) = \{\underset{\uparrow}{5}, 4, 3, 2, 1, 0, 0, 1, 2, 3, 4\}, \quad 0 \le n \le 10$$

with (a) $m = -5$, $N = 12$ and (b) $m = 8$, $N = 15$.

P5.21 Using the analysis and synthesis equations of the DFT, show that the energy of a sequence satisfies

$$\mathcal{E}_X \triangleq \sum_{n=0}^{N-1} |x(n)|^2 = \frac{1}{N}\sum_{k=0}^{N-1} |X(k)|^2$$

This is commonly referred to as a *Parseval's relation for the DFT*. Verify this relation using MATLAB on the sequence in Problem P5.20.

P5.22 A 512-point DFT $X(k)$ of a real-valued sequence $x(n)$ has the following DFT values:

$$\begin{array}{llll} X(0) = 20 + j\alpha; & X(5) = 20 + j30; & X(k_1) = -10 + j15; & X(152) = 17 + j23; \\ X(k_2) = 20 - j30; & X(k_3) = 17 - j23; & X(480) = -10 - j15; & X(256) = 30 + j\beta \end{array}$$

and all other values are known to be zero.

1. Determine the real-valued coefficients α and β.
2. Determine the values of the integers k_1, k_2, and k_3.
3. Determine the energy of the signal $x(n)$.
4. Express the sequence $x(n)$ in a closed form.

P5.23 Let $x(n)$ be a finite length sequence given by

$$x(n) = \left\{\ldots, 0, 0, 0, \underset{\uparrow}{1}, 2, -3, 4, -5, 0, \ldots\right\}$$

Determine and sketch the sequence $x((-8-n))_7 \mathcal{R}_7(n)$ where

$$\mathcal{R}_7(n) = \begin{cases} 1, & 0 \le n \le 6 \\ 0, & \text{elsewhere} \end{cases}$$

P5.24 The `circonvt` function developed in this chapter implements the circular convolution as a matrix-vector multiplication. The matrix corresponding to the circular shifts $\{x((n-m))_N;\ 0 \le n \le N-1\}$ has an interesting structure. This matrix is called a *circulant* matrix, which is a special case of Toeplitz matrix introduced in Chapter 2.

1. Consider the sequences given in Example 5.13. Express $x_1(n)$ as a column vector $\mathbf{x}_1$ and $x_2((n-m))_N$ as a circulant matrix $\mathbf{X}_2$ with rows corresponding to $n = 0, 1, 2, 3$. Characterize this matrix $\mathbf{X}_2$. Can it completely be described by its first row (or column)?
2. Determine the circular convolution as $\mathbf{X}_2\mathbf{x}_1$ and verify your calculations.

P5.25 Develop a MATLAB function to construct a circulant matrix $\mathbf{C}$ given an N-point sequence $x(n)$. Use the `toeplitz` function to implement matrix $\mathbf{C}$. Your subroutine function should have the following format:

```
function [C] = circulnt(x,N)
% Circulant Matrix from an N-point sequence
% [C] = circulnt(x,N)
% C = circulant matrix of size NxN
% x = sequence of length <= N
% N = size of circulant matrix
```

Using this function, modify the circular convolution function `circonvt` discussed in the chapter so that the `for...end` loop is eliminated. Verify your functions on the sequences in Problem P5.24.

P5.26 Using the frequency domain approach, devise a MATLAB function to implement the circular convolution operation between two sequences. The format of the sequence should be

```
function x3 = circonvf(x1,x2,N)
% Circular convolution in the frequency domain
%  x3 = circonvf(x1,x2,N)
%  x3 = convolution result of length N
%  x1 = sequence of length <= N
%  x2 = sequence of length <= N
%   N = length of circular buffer
```

Using your function, compute the circular convolution $\{4,3,2,1\}$ $\textcircled{4}$ $\{1,2,3,4\}$.

P5.27 The following four sequences are given:

$$x_1(n) = \{\underset{\uparrow}{1},3,2,-1\};\ x_2(n) = \{\underset{\uparrow}{2},1,0,-1\};\ x_3(n) = x_1(n) * x_2(n);\ x_4(n) = x_1(n) \textcircled{5}\ x_2(n)$$

1. Determine and sketch $x_3(n)$.
2. Using $x_3(n)$ alone, determine and sketch $x_4(n)$. Do not directly compute $x_4(n)$.

P5.28 Compute the N-point circular convolution for the following sequences. Plot their samples.

1. $x_1(n) = \sin(\pi n/3)\mathcal{R}_6(n), \quad x_2(n) = \cos(\pi n/4)\mathcal{R}_8(n); \quad N = 10$
2. $x_1(n) = \cos(2\pi n/N)\mathcal{R}_N(n), \quad x_2(n) = \sin(2\pi n/N)\mathcal{R}_N(n); \quad N = 32$
3. $x_1(n) = (0.8)^n\,\mathcal{R}_N(n), \quad x_2(n) = (-0.8)^n\,\mathcal{R}_N(n); \quad N = 20$
4. $x_1(n) = n\mathcal{R}_N(n), \quad x_2(n) = (N-n)\,\mathcal{R}_N(n); \quad N = 10$
5. $x_1(n) = (0.8)^n R_{20}, \quad x_2(n) = u(n) - u(n-40); \quad N = 50$

P5.29 Let $x_1(n)$ and $x_2(n)$ be two N-point sequences.

1. If $y(n) = x_1(n)$ $\textcircled{N}$ $x_2(n)$ show that

$$\sum_{n=0}^{N-1} y(n) = \left(\sum_{n=0}^{N-1} x_1(n)\right)\left(\sum_{n=0}^{N-1} x_2(n)\right)$$

2. Verify this result for the following sequences.

$$x_1(n) = \{9,4,-1,4,-4,-1,8,3\}; \quad x_2(n) = \{-5,6,2,-7,-5,2,2,-2\}$$

P5.30 Let $X(k)$ be the 8-point DFT of a 3-point sequence $x(n) = \{\underset{\uparrow}{5},-4,3\}$. Let $Y(k)$ be the 8-point DFT of a sequence $y(n)$. Determine $y(n)$ when $Y(k) = W_8^{5k}X(-k)_8$.

P5.31 For the following sequences compute (i) the N-point circular convolution $x_3(n) = x_1(n)$ $\textcircled{N}$ $x_2(n)$, (ii) the linear convolution $x_4(n) = x_1(n) * x_2(n)$, and (iii) the error sequence $e(n) = x_3(n) - x_4(n)$.

1. $x_1(n) = \{1,1,1,1\}, \quad x_2(n) = \cos(\pi n/4)\,\mathcal{R}_6(n); \quad N = 8$
2. $x_1(n) = \cos(2\pi n/N)\,\mathcal{R}_{16}(n), \quad x_2(n) = \sin(2\pi n/N)\,\mathcal{R}_{16}(n); \quad N = 32$
3. $x_1(n) = (0.8)^n\,\mathcal{R}_{10}(n), \quad x_2(n) = (-0.8)^n\,\mathcal{R}_{10}(n); \quad N = 15$
4. $x_1(n) = n\mathcal{R}_{10}(n), \quad x_2(n) = (N-n)\,\mathcal{R}_{10}(n); \quad N = 10$
5. $x_1(n) = \{1,-1,1,-1\}, \quad x_2(n) = \{1,0,-1,0\}; \quad N = 5$

In each case verify that $e(n) = x_4(n+N)$.

P5.32 The overlap-add method of block convolution is an alternative to the overlap-save method. Let $x(n)$ be a long sequence of length ML where $M, L \gg 1$. Divide $x(n)$ into M segments $\{x_m(n),\ m = 1, \ldots, M\}$ each of length L

$$x_m(n) = \begin{cases} x(n), & mL \le n \le (m+1)L - 1 \\ 0, & \text{elsewhere} \end{cases} \qquad \text{so that} \quad x(n) = \sum_{m=0}^{M-1} x_m(n)$$

Let $h(n)$ be an L-point impulse response. Then

$$y(n) = x(n) * h(n) = \sum_{m=0}^{M-1} x_m(n) * h(n) = \sum_{m=0}^{M-1} y_m(n); \quad y_m(n) \stackrel{\triangle}{=} x_m(n) * h(n)$$

Clearly, $y_m(n)$ is a $(2L-1)$-point sequence. In this method we have to save the intermediate convolution results and then properly overlap these before adding to form the final result $y(n)$. To use DFT for this operation we have to choose $N \ge (2L-1)$.

1. Develop a MATLAB function to implement the overlap-add method using the circular convolution operation. The format should be

```
function [y] = ovrlpadd(x,h,N)
% Overlap-Add method of block convolution
% [y] = ovrlpadd(x,h,N)
%
% y = output sequence
% x = input sequence
% h = impulse response
% N = block length >= 2*length(h)-1
```

2. Incorporate the radix-2 FFT implementation in this function to obtain a high-speed overlap-add block convolution routine. Remember to choose $N = 2^\nu$.
3. Verify your functions on the following two sequences

$$x(n) = \cos(\pi n/500)\,\mathcal{R}_{4000}(n), \quad h(n) = \{1, -1, 1, -1\}$$

P5.33 Given the following sequences $x_1(n)$ and $x_2(n)$:

$$x_1(n) = \{2, 1, 1, 2\}, \quad x_2(n) = \{1, -1, -1, 1\}$$

1. Compute the circular convolution $x_1(n) \textcircled{N} x_2(n)$ for $N = 4$, 7, and 8.
2. Compute the linear convolution $x_1(n) * x_2(n)$.
3. Using results of calculations, determine the minimum value of N necessary so that linear and circular convolutions are same on the N-point interval.
4. Without performing the actual convolutions, explain how you could have obtained the result of P5.33.3.

P5.34 Let

$$x(n) = \begin{cases} A\cos(2\pi\ell n/N), & 0 \le n \le N-1 \\ 0, & \text{elsewhere} \end{cases} = A\cos(2\pi\ell n/N)\,\mathcal{R}_N(n)$$

where ℓ is an integer. Notice that $x(n)$ contains *exactly* ℓ periods (or cycles) of the cosine waveform in N samples. This is a windowed cosine sequence containing *no leakage*.

1. Show that the DFT $X(k)$ is a real sequence given by

$$X(k) = \frac{AN}{2}\delta\left(k-\ell\right) + \frac{AN}{2}\delta(k-N+\ell);\quad 0 \le k \le (N-1),\quad 0 < \ell < N$$

2. Show that if $\ell = 0$, then the DFT $X(k)$ is given by

$$X(k) = AN\delta(k);\quad 0 \le k \le (N-1)$$

3. Explain clearly how these results should be modified if $\ell < 0$ or $\ell > N$.
4. Verify the results of parts 1, 2, and 3 using the following sequences. Plot the real parts of the DFT sequences using the `stem` function.
   (a) $x_1(n) = 3\cos\left(0.04\pi n\right)\mathcal{R}_{200}(n)$
   (b) $x_2(n) = 5\mathcal{R}_{50}(n)$
   (c) $x_3(n) = \left[1 + 2\cos\left(0.5\pi n\right) + \cos\left(\pi n\right)\right]\mathcal{R}_{100}(n)$
   (d) $x_4(n) = \cos\left(25\pi n/16\right)\mathcal{R}_{64}(n)$
   (e) $x_5(n) = \left[4\cos\left(0.1\pi n\right) - 3\cos\left(1.9\pi n\right)\right]\mathcal{R}_{40}(n)$

P5.35 Let $x(n) = A\cos\left(\omega_0 n\right)\mathcal{R}_N(n)$, where ω_0 is a real number.

1. Using the properties of the DFT, show that the real and the imaginary parts of $X(k)$ are given by

$$\begin{aligned}
X(k) &= X_R(k) + jX_I(k)\\
X_R(k) &= (A/2)\cos\left[\tfrac{\pi(N-1)}{N}\left(k-f_0N\right)\right]\frac{\sin\left[\pi\left(k-f_0N\right)\right]}{\sin\left[\pi(k-f_0N)/N\right]}\\
&\quad + (A/2)\cos\left[\tfrac{\pi(N-1)}{N}\left(k+f_0N\right)\right]\frac{\sin\left[\pi\left(k-N+f_0N\right)\right]}{\sin\left[\pi(k-N+f_0N)/N\right]}\\
X_I(k) &= -\,(A/2)\sin\left[\tfrac{\pi(N-1)}{N}\left(k-f_0N\right)\right]\frac{\sin\left[\pi\left(k-f_0N\right)\right]}{\sin\left[\pi(k-f_0N)/N\right]}\\
&\quad -\,(A/2)\sin\left[\tfrac{\pi(N-1)}{N}\left(k+f_0N\right)\right]\frac{\sin\left[\pi\left(k-N+f_0N\right)\right]}{\sin\left[\pi(k-N+f_0N)/N\right]}
\end{aligned}$$

2. This result implies that the original frequency ω_0 of the cosine waveform has *leaked* into other frequencies that form the harmonics of the time-limited sequence, and hence it is called the leakage property of cosines. It is a natural result due to the fact that bandlimited periodic cosines are sampled over noninteger periods. Explain this result using the periodic extension $\tilde{x}(n)$ of $x(n)$ and the result in Problem P5.34.1.
3. Verify the leakage property using $x(n) = \cos\left(5\pi n/99\right)\mathcal{R}_{200}(n)$. Plot the real and the imaginary parts of $X(k)$ using the `stem` function.

P5.36 Let

$$x(n) = \begin{cases} A\sin\left(2\pi\ell n/N\right), & 0 \le n \le N-1\\ 0, & \text{Elsewhere}\end{cases} = A\sin\left(2\pi\ell n/N\right)\mathcal{R}_N(n)$$

where ℓ is an integer. Notice that $x(n)$ contains *exactly* ℓ periods (or cycles) of the sine waveform in N samples. This is a windowed sine sequence containing *no leakage*.

1. Show that the DFT $X(k)$ is a purely imaginary sequence given by

$$X(k) = \frac{AN}{2j}\delta\,(k-\ell) - \frac{AN}{2j}\delta(k-N+\ell); \quad 0 \le k \le (N-1), \quad 0 < \ell < N$$

2. Show that if $\ell = 0$, then the DFT $X(k)$ is given by

$$X(k) = 0; \quad 0 \le k \le (N-1)$$

3. Explain clearly how these results should be modified if $\ell < 0$ or $\ell > N$.
4. Verify the results of parts 1, 2, and 3 using the following sequences. Plot the imaginary parts of the DFT sequences using the `stem` function.
   (a) $x_1(n) = 3\sin(0.04\pi n)\,\mathcal{R}_{200}(n)$
   (b) $x_2(n) = 5\sin 10\pi n\mathcal{R}_{50}(n)$
   (c) $x_3(n) = [2\sin(0.5\pi n) + \sin(\pi n)]\,\mathcal{R}_{100}(n)$
   (d) $x_4(n) = \sin(25\pi n/16)\,\mathcal{R}_{64}(n)$
   (e) $x_5(n) = [4\sin(0.1\pi n) - 3\sin(1.9\pi n)]\,\mathcal{R}_{20}(n)$

P5.37 Let $x(n) = A\sin(\omega_0 n)\,\mathcal{R}_N(n)$, where ω_0 is a real number.

1. Using the properties of the DFT, show that the real and the imaginary parts of $X(k)$ are given by

$$\begin{aligned}
X(k) &= X_R(k) + jX_I(k) \\
X_R(k) &= -(A/2)\sin\left[\tfrac{\pi(N-1)}{N}(k-f_0N)\right]\frac{\sin[\pi(k-f_0N)]}{\sin[\pi(k-f_0N)/N]} \\
&\quad + (A/2)\sin\left[\tfrac{\pi(N-1)}{N}(k+f_0N)\right]\frac{\sin[\pi(k-N+f_0N)]}{\sin[\pi(k-N+f_0N)/N]} \\
X_I(k) &= -(A/2)\cos\left[\tfrac{\pi(N-1)}{N}(k-f_0N)\right]\frac{\sin[\pi(k-f_0N)]}{\sin[\pi(k-f_0N)/N]} \\
&\quad + (A/2)\cos\left[\tfrac{\pi(N-1)}{N}(k+f_0N)\right]\frac{\sin[\pi(k-N+f_0N)]}{\sin[\pi(k-N+f_0N)/N]}
\end{aligned}$$

2. This result is the leakage property of sines. Explain it using the periodic extension $\tilde{x}(n)$ of $x(n)$ and the result in Problem P5.36.1.
3. Verify the leakage property using $x(n) = \sin(5\pi n/99)\,\mathcal{R}_{100}(n)$. Plot the real and the imaginary parts of $X(k)$ using the `stem` function.

P5.38 An analog signal $x_a(t) = 2\sin(4\pi t) + 5\cos(8\pi t)$ is sampled at $t = 0.01n$ for $n = 0, 1, \ldots, N-1$ to obtain an N-point sequence $x(n)$. An N-point DFT is used to obtain an estimate of the magnitude spectrum of $x_a(t)$.

1. From the following values of N, choose the one that will provide the accurate estimate of the spectrum of $x_a(t)$. Plot the real and imaginary parts of the DFT spectrum $X(k)$.
   (a) $N = 40$, (b) $N = 50$, (c) $N = 60$.

2. From the following values of N, choose the one that will provide the least amount of leakage in the spectrum of $x_a(t)$. Plot the real and imaginary parts of the DFT spectrum $X(k)$. (a) $N = 90$, (b) $N = 95$, (c) $N = 99$.

P5.39 Using (5.49), determine and draw the signal flow graph for the $N = 8$ point, radix-2 decimation-in-frequency FFT algorithm. Using this flow graph, determine the DFT of the sequence

$$x(n) = \cos(\pi n/2), \quad 0 \le n \le 7$$

P5.40 Using (5.49), determine and draw the signal flow graph for the $N = 16$ point, radix-4 decimation-in-time FFT algorithm. Using this flow graph, determine the DFT of the sequence

$$x(n) = \cos(\pi n/2), \quad 0 \le n \le 15$$

P5.41 Let $x(n)$ be a uniformly distributed random number between $[-1, 1]$ for $0 \le n \le 10^6$. Let

$$h(n) = \sin(0.4\pi n), \quad 0 \le n \le 100$$

1. Using the `conv` function, determine the output sequence $y(n) = x(n) * h(n)$.
2. Consider the overlap-and-save method of block convolution along with the FFT algorithm to implement high-speed block convolution. Using this approach, determine $y(n)$ with FFT sizes of 1024, 2048, and 4096.
3. Compare these approaches in terms of the convolution results and their execution times.

CHAPTER 6

Implementation of Discrete-time Filters

In earlier chapters we studied the theory of discrete systems in both the time and frequency domains. We will now use this theory for the processing of digital signals. To process signals, we have to design and implement systems called *filters* (or spectrum analyzers in some contexts). The filter design issue is influenced by such factors as the type of the filter (i.e., IIR or FIR) or the form of its implementation (structures). Hence, before we discuss the design issue, we first concern ourselves with how these filters can be implemented in practice. This is an important concern because different filter structures dictate different design strategies.

IIR filters as designed and used in DSP, can be modeled by rational system functions or, equivalently, by difference equations. Such filters are termed *autoregressive moving average* (*ARMA*) or, more generally, as *recursive* filters. Although ARMA filters include moving average filters that are FIR filters, we will treat FIR filters separately from IIR filters for both design and implementation purposes.

In addition to describing various filter structures, we also begin to consider problems associated with quantization effects when finite-precision arithmetic is used in the implementation. Digital hardware contains processing elements that use finite-precision arithmetic. When filters are implemented either in hardware or in software, filter coefficients as well as filter operations are subjected to the effects of these finite-precision operations. In this chapter, we treat the effects on filter frequency response characteristics due to coefficient quantization. In Chapter 10, we will consider the round-off noise effects in the digital filter implementations.

We begin with a description of basic building blocks that are used to describe filter structures. In the subsequent sections, we briefly describe IIR, FIR, and lattice filter structures, respectively, and provide MATLAB functions to implement these structures. This is followed by a brief treatment of the representation of numbers and the resulting error characteristics, which is then used to analyze coefficient quantization effects.

6.1 BASIC ELEMENTS

Since our filters are LTI systems, we need the following three elements to describe digital filter structures. These elements are shown in Figure 6.1.

1. **Adder:** This element has two inputs and one output and is shown in Figure 6.1a. Note that the addition of three or more signals is implemented by successive two-input adders.
2. **Multiplier (gain):** This is a single-input, single-output element and is shown in Figure 6.1b. Note that the multiplication by 1 is understood and hence not explicitly shown.
3. **Delay element (shifter or memory):** This element delays the signal passing through it by one sample, as shown in Figure 6.1c. It is implemented by using a shift register.

Using these basic elements, we can now describe various structures of both IIR and FIR filters. MATLAB is a convenient tool in the development of these structures that require operations on polynomials.

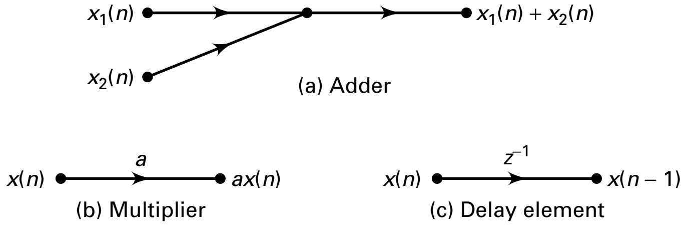


FIGURE 6.1 *Three basic elements*

6.2 IIR FILTER STRUCTURES

The system function of an IIR filter is given by

$$H(z) = \frac{B(z)}{A(z)} = \frac{\sum_{n=0}^{M} b_n z^{-n}}{\sum_{n=0}^{N} a_n z^{-n}} = \frac{b_0 + b_1 z^{-1} + \cdots + b_M z^{-M}}{1 + a_1 z^{-1} + \cdots + a_N z^{-N}}; \; a_0 = 1 \tag{6.1}$$

where b_n and a_n are the coefficients of the filter. We have assumed without loss of generality that $a_0 = 1$. The order of such an IIR filter is called N if $a_N \neq 0$. The difference equation representation of an IIR filter is expressed as

$$y(n) = \sum_{m=0}^{M} b_m x(n-m) - \sum_{m=1}^{N} a_m y(n-m) \tag{6.2}$$

Three different structures can be used to implement an IIR filter:

1. **Direct form:** In this form the difference equation (6.2) is implemented directly as given. There are two parts to this filter, namely the moving average part and the recursive part (or equivalently, the numerator and denominator parts). Therefore this implementation leads to two versions: direct form I and direct form II structures.
2. **Cascade form:** In this form the system function $H(z)$ in equation (6.1) is factored into smaller 2nd-order sections, called *biquads*. The system function is then represented as a *product* of these biquads. Each biquad is implemented in a direct form, and the entire system function is implemented as a *cascade* of biquad sections.
3. **Parallel form:** This is similar to the cascade form, but after factorization, a partial fraction expansion is used to represent $H(z)$ as a *sum* of smaller 2nd-order sections. Each section is again implemented in a direct form, and the entire system function is implemented as a *parallel* network of sections.

We will briefly discuss these forms in this section. IIR filters are generally described using the rational form version (or the direct form structure) of the system function. Hence we will provide MATLAB functions for converting direct form structures to cascade and parallel form structures.

6.2.1 DIRECT FORM

As the name suggests, the difference equation (6.2) is implemented as given using delays, multipliers, and adders. For the purpose of illustration, let $M = N = 4$. Then the difference equation is

$$\begin{aligned} y(n) = \; & b_0 x(n) + b_1 x(n-1) + b_2 x(n-2) + b_3 x(n-3) + b_4 x(n-4) \\ & - a_1 y(n-1) - a_2 y(n-2) - a_3 y(n-3) - a_4 y(n-4) \end{aligned}$$

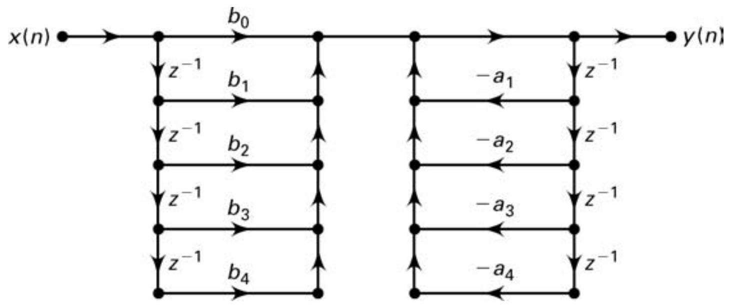


FIGURE 6.2 *Direct form I structure*

which can be implemented as shown in Figure 6.2. This block diagram is called *direct form I* structure.

The direct form I structure implements each part of the rational function $H(z)$ separately with a cascade connection between them. The numerator part is a tapped delay line followed by the denominator part, which is a feedback tapped delay line. Thus there are two separate delay lines in this structure, and hence it requires eight delay elements. We can reduce this delay element count or eliminate one delay line by interchanging the order in which the two parts are connected in the cascade. Now the two delay lines are close to each other, connected by a unity gain branch. Therefore one delay line can be removed, and this reduction leads to a canonical structure called *direct form II* structure, shown in Figure 6.3. It should be noted that both direct forms are equivalent from the input-output point of view. Internally, however, they have different signals.

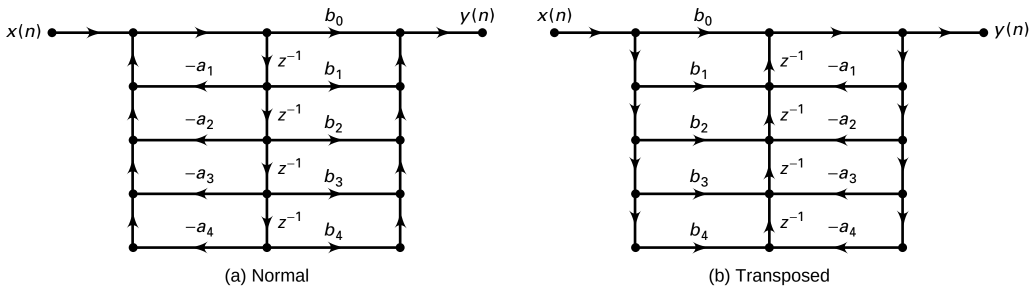


FIGURE 6.3 *Direct form II structure*

6.2.2 TRANSPOSED STRUCTURE

An equivalent structure to the direct form can be obtained using a procedure called *transposition*. In this operation three steps are performed:

1. All path arrow directions are reversed.
2. All branch nodes are replaced by adder nodes, and all adder nodes are replaced by branch nodes.
3. The input and output nodes are interchanged.

The resulting structure is called the *transposed direct form* structure. The transposed direct form II structure is shown in Figure 6.3b. Problem P6.3 explains this equivalent structure.

6.2.3 MATLAB IMPLEMENTATION

In MATLAB the direct form structure is described by two row vectors; `b` containing the $\{b_n\}$ coefficients and `a` containing the $\{a_n\}$ coefficients. The `filter` function, which is discussed in Chapter 2, implements the transposed direct form II structure.

6.2.4 CASCADE FORM

In this form the system function $H(z)$ is written as a product of 2nd-order sections with real coefficients. This is done by factoring the numerator and denominator polynomials into their respective roots and then combining either a complex conjugate root pair or any two real roots into 2nd-order polynomials. In the remainder of this chapter, we assume that N is an even integer. Then

$$\begin{aligned} H(z) &= \frac{b_0 + b_1 z^{-1} + \cdots + b_N z^{-N}}{1 + a_1 z^{-1} + \cdots + a_N z^{-N}} \\ &= b_0 \frac{1 + \frac{b_1}{b_0} z^{-1} + \cdots + \frac{b_N}{b_0} z^{-N}}{1 + a_1 z^{-1} + \cdots + a_N z^{-N}} \\ &= b_0 \prod_{k=1}^{K} \frac{1 + B_{k,1} z^{-1} + B_{k,2} z^{-2}}{1 + A_{k,1} z^{-1} + A_{k,2} z^{-2}} \end{aligned} \tag{6.3}$$

where K is equal to $\frac{N}{2}$, and $B_{k,1}$, $B_{k,2}$, $A_{k,1}$, and $A_{k,2}$ are real numbers representing the coefficients of 2nd-order sections. The 2nd-order section

$$H_k(z) = \frac{Y_{k+1}(z)}{Y_k(z)} = \frac{1 + B_{k,1} z^{-1} + B_{k,2} z^{-2}}{1 + A_{k,1} z^{-1} + A_{k,2} z^{-2}}; \quad k = 1, \ldots, K$$

with

$$Y_1(z) = b_0 X(z); \quad Y_{K+1}(z) = Y(z)$$

is called the kth biquad section. The input to the kth biquad section is the output from the $(k-1)$th biquad section, and the output from the

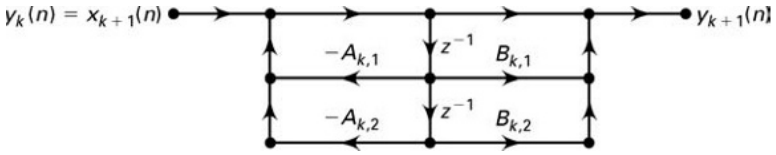


FIGURE 6.4 *Biquad section structure*

kth biquad is the input to the $(k+1)$th biquad. Now each biquad section $H_k(z)$ can be implemented in direct form II, as shown in Figure 6.4. The entire filter is then implemented as a cascade of biquads.

As an example, consider $N = 4$. Figure 6.5 shows a cascade form structure for this 4th-order IIR filter.

6.2.5 MATLAB IMPLEMENTATION

Given the coefficients $\{b_n\}$ and $\{a_n\}$ of the direct form filter, we have to obtain the coefficients b_0, $\{B_{k,i}\}$, and $\{A_{k,i}\}$. This is done by the following function `dir2cas`.

```
function [b0,B,A] = dir2cas(b,a);
% DIRECT-form to CASCADE-form conversion (cplxpair version)
% -----------------------------------------------------------
% [b0,B,A] = dir2cas(b,a)
% b0 = gain coefficient
%  B = K by 3 matrix of real coefficients containing bk's
%  A = K by 3 matrix of real coefficients containing ak's
%  b = numerator polynomial coefficients of DIRECT form
%  a = denominator polynomial coefficients of DIRECT form

% compute gain coefficient b0
b0 = b(1); b = b/b0;  a0 = a(1); a = a/a0;  b0 = b0/a0;
%
M = length(b); N = length(a);
if N > M
b = [b zeros(1,N-M)];
```

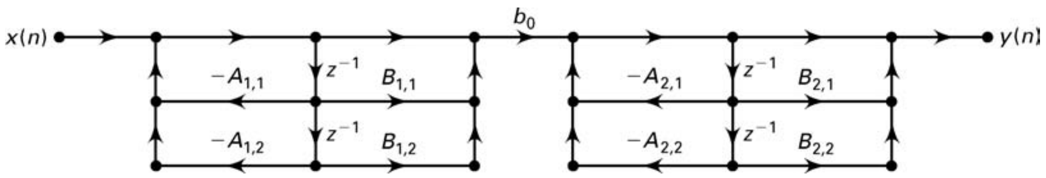


FIGURE 6.5 *Cascade form structure for $N = 4$*

```
elseif M > N
a = [a zeros(1,M-N)]; N = M;
else
NM = 0;
end
%
K = floor(N/2); B = zeros(K,3); A = zeros(K,3);
if K*2 == N;
b = [b 0];  a = [a 0];
end
%
broots = cplxpair(roots(b));  aroots = cplxpair(roots(a));
for i=1:2:2*K
Brow = broots(i:1:i+1,:);   Brow = real(poly(Brow));
B(fix((i+1)/2),:) = Brow;
Arow = aroots(i:1:i+1,:);   Arow = real(poly(Arow));
A(fix((i+1)/2),:) = Arow;
end
```

This function converts the b and a vectors into $K \times 3$ B and A matrices. It begins by computing b_0, which is equal to b_0/a_0 (assuming $a_0 \neq 1$). It then makes the vectors b and a of equal length by zero-padding the shorter vector. This ensures that each biquad has a nonzero numerator and denominator. Next it computes the roots of the $B(z)$ and $A(z)$ polynomials. Using the cplxpair function, these roots are ordered in complex conjugate pairs. Now every pair is converted back into a 2nd-order numerator or denominator polynomial using the poly function. The SP toolbox function, tf2sos (transfer function to 2nd-order section), also performs a similar operation.

The cascade form is implemented using the following casfiltr function.

```
function y = casfiltr(b0,B,A,x);
% CASCADE form realization of IIR and FIR filters
% -----------------------------------------------
% y = casfiltr(b0,B,A,x);
%  y = output sequence
% b0 = gain coefficient of CASCADE form
%  B = K by 3 matrix of real coefficients containing bk's
%  A = K by 3 matrix of real coefficients containing ak's
%  x = input sequence
%
[K,L] = size(B);
N = length(x);  w = zeros(K+1,N);  w(1,:) = x;
```

```
for i = 1:1:K
        w(i+1,:) = filter(B(i,:),A(i,:),w(i,:));
end
y = b0*w(K+1,:);
```

It employs the `filter` function in a loop using the coefficients of each biquad stored in `B` and `A` matrices. The input is scaled by `b0`, and the output of each filter operation is used as an input to the next filter operation. The output of the final filter operation is the overall output.

The following MATLAB function, `cas2dir`, converts a cascade form to a direct form. This is a simple operation that involves multiplication of several 2nd-order polynomials. For this purpose, the MATLAB function `conv` is used in a loop over K factors. The SP toolbox function, `sos2tf` also performs a similar operation.

```
function [b,a] = cas2dir(b0,B,A);
% CASCADE-to-DIRECT form conversion
% ---------------------------------
% [b,a] = cas2dir(b0,B,A)
%  b = numerator polynomial coefficients of DIRECT form
%  a = denominator polynomial coefficients of DIRECT form
% b0 = gain coefficient
%  B = K by 3 matrix of real coefficients containing bk's
%  A = K by 3 matrix of real coefficients containing ak's
%
[K,L] = size(B);
b = [1];  a = [1];
for i=1:1:K
b=conv(b,B(i,:));  a=conv(a,A(i,:));
end
b = b*b0;
```

□ **EXAMPLE 6.1** A filter is described by the following difference equation:

$$\begin{aligned}16y(n) + 12y(n-1) + 2y(n-2) - 4y(n-3) - y(n-4)\\ = x(n) - 3x(n-1) + 11x(n-2) - 27x(n-3) + 18x(n-4)\end{aligned}$$

Determine its cascaded form structure.

Solution MATLAB script:

```
>> b=[1 -3 11 -27 18];  a=[16 12 2 -4 -1];
>> [b0,B,A]=dir2cas(b,a)
 b0 = 0.0625
  B =
      1.0000   -0.0000    9.0000
      1.0000   -3.0000    2.0000
  A =
      1.0000 1.0000    0.5000
      1.0000   -0.2500   -0.1250
```

The resulting structure is shown in Figure 6.6. To check that our cascade structure is correct, let us compute the first 8 samples of the impulse response using both forms.

```
>> delta = impseq(0,0,7)
delta =
      1     0     0     0     0     0     0     0
>> format long
>> hcas=casfiltr(b0,B,A,delta)
hcas =
  Columns 1 through 4
   0.06250000000000  -0.23437500000000   0.85546875000000  -2.28417968750000
  Columns 5 through 8
   2.67651367187500  -1.52264404296875   0.28984069824219   0.49931716918945
>> hdir=filter(b,a,delta)
hdir =
  Columns 1 through 4
   0.06250000000000  -0.23437500000000   0.85546875000000  -2.28417968750000
  Columns 5 through 8
   2.67651367187500  -1.52264404296875   0.28984069824219   0.49931716918945
```

□

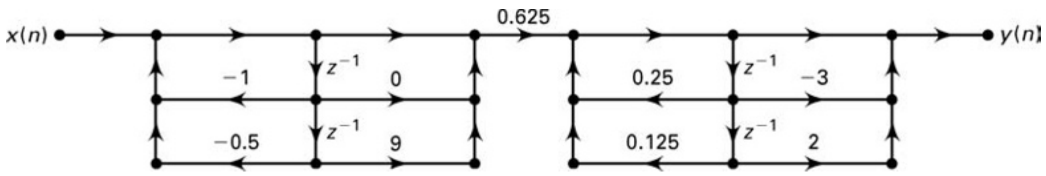


FIGURE 6.6 *Cascade structure in Example 6.1*

6.2.6 PARALLEL FORM

In this form the system function $H(z)$ is written as a sum of 2nd-order sections using partial fraction expansion.

$$\begin{aligned}
H(z) &= \frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + \cdots + b_M z^{-M}}{1 + a_1 z^{-1} + \cdots + a_N z^{-N}} \\
&= \frac{\hat{b}_0 + \hat{b}_1 z^{-1} + \cdots + \hat{b}_{N-1} z^{1-N}}{1 + a_1 z^{-1} + \cdots + a_N z^{-N}} + \underbrace{\sum_{0}^{M-N} C_k z^{-k}}_{\text{only if } M \ge N} \\
&= \sum_{k=1}^{K} \frac{B_{k,0} + B_{k,1} z^{-1}}{1 + A_{k,1} z^{-1} + A_{k,2} z^{-2}} + \underbrace{\sum_{0}^{M-N} C_k z^{-k}}_{\text{only if } M \ge N}
\end{aligned} \tag{6.4}$$

where K is equal to $\frac{N}{2}$, and $B_{k,0}$, $B_{k,1}$, $A_{k,1}$, and $A_{k,2}$ are real numbers representing the coefficients of 2nd-order sections. The 2nd-order section

$$H_k(z) = \frac{Y_{k+1}(z)}{Y_k(z)} = \frac{B_{k,0} + B_{k,1} z^{-1}}{1 + A_{k,1} z^{-1} + A_{k,2} z^{-2}}; \quad k = 1, \ldots, K$$

with

$$Y_k(z) = H_k(z) X(z), \quad Y(z) = \sum Y_k(z), \quad M < N$$

is the kth proper rational biquad section. The filter input is available to all biquad sections as well as to the polynomial section if $M \ge N$ (which is an FIR part). The output from these sections is summed to form the filter output. Now each biquad section $H_k(z)$ can be implemented in direct form II. Due to the summation of subsections, a parallel structure can be built to realize $H(z)$. As an example, consider $M = N = 4$. Figure 6.7 shows a parallel-form structure for this 4th-order IIR filter.

6.2.7 MATLAB IMPLEMENTATION

The following function `dir2par` converts the direct-form coefficients $\{b_n\}$ and $\{a_n\}$ into parallel form coefficients $\{B_{k,i}\}$ and $\{A_{k,i}\}$.

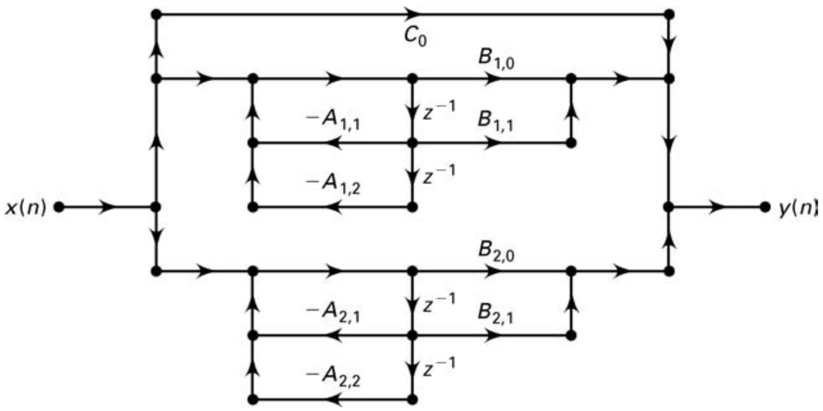


FIGURE 6.7 *Parallel form structure for* $N = 4$

```
function [C,B,A] = dir2par(b,a);
% DIRECT-form to PARALLEL-form conversion
% --------------------------------------
% [C,B,A] = dir2par(b,a)
%  C = Polynomial part when length(b) >= length(a)
%  B = K by 2 matrix of real coefficients containing bk's
%  A = K by 3 matrix of real coefficients containing ak's
%  b = numerator polynomial coefficients of DIRECT form
%  a = denominator polynomial coefficients of DIRECT form
%
M = length(b); N = length(a);

[r1,p1,C] = residuez(b,a);
p = cplxpair(p1,10000000*eps);  I = cplxcomp(p1,p);  r = r1(I);

K = floor(N/2); B = zeros(K,2); A = zeros(K,3);
if K*2 == N; %N even, order of A(z) odd, one factor is first order
 for i=1:2:N-2
 Brow = r(i:1:i+1,:);  Arow = p(i:1:i+1,:);
 [Brow,Arow] = residuez(Brow,Arow,[]);
 B(fix((i+1)/2),:) = real(Brow);  A(fix((i+1)/2),:) = real(Arow);
 end
 [Brow,Arow] = residuez(r(N-1),p(N-1),[]);
 B(K,:) = [real(Brow) 0]; A(K,:) = [real(Arow) 0];
else
for i=1:2:N-1
```

```
Brow = r(i:1:i+1,:);  Arow = p(i:1:i+1,:);
[Brow,Arow] = residuez(Brow,Arow,[]);
B(fix((i+1)/2),:) = real(Brow);  A(fix((i+1)/2),:) = real(Arow);
end
end
```

The `dir2cas` function first computes the z-domain partial fraction expansion using the `residuez` function. We need to arrange pole-and-residue pairs into complex conjugate pole-and-residue pairs followed by real pole-and-residue pairs. To do this, the `cplxpair` function from MATLAB can be used; this sorts a complex array into complex conjugate pairs. However, two consecutive calls to this function, one each for pole and residue arrays, will not guarantee that poles and residues will correspond to each other. Therefore a new `cplxcomp` function is developed, which compares two shuffled complex arrays and returns the index of one array, which can be used to rearrange another array.

```
function I = cplxcomp(p1,p2)
%  I = cplxcomp(p1,p2)
% Compares two complex pairs which contain the same scalar elements
%  but (possibly) at differrent indices.  This routine should be
%  used after CPLXPAIR routine for rearranging pole vector and its
%  corresponding residue vector.
%      p2 = cplxpair(p1)
%
I=[];
for j=1:1:length(p2)
    for i=1:1:length(p1)
         if (abs(p1(i)-p2(j)) < 0.0001)
           I=[I,i];
        end
    end
end
I=I';
```

After collecting these pole-and-residue pairs, the `dir2cas` function computes the numerator and denominator of the biquads by employing the `residuez` function in the reverse fashion.

These parallel-form coefficients are then used in the function `parfiltr`, which implements the parallel form. The `parfiltr` function uses the `filter` function in a loop using the coefficients of each biquad stored in the `B` and `A` matrices. The input is first filtered through the FIR part `C` and stored in the first row of a `w` matrix. Then the outputs of all biquad filters are computed for the same input and stored as subsequent rows in the w matrix. Finally, all the columns of the w matrix are summed to yield the output.

```
function y = parfiltr(C,B,A,x);
% PARALLEL form realization of IIR filters
% ----------------------------------------
%  [y] = parfiltr(C,B,A,x);
%  y = output sequence
%  C = polynomial (FIR) part when M >= N
%  B = K by 2 matrix of real coefficients containing bk's
%  A = K by 3 matrix of real coefficients containing ak's
%  x = input sequence
%
[K,L] = size(B);  N = length(x);  w = zeros(K+1,N);
w(1,:) = filter(C,1,x);
for i = 1:1:K
        w(i+1,:) = filter(B(i,:),A(i,:),x);
end
y = sum(w);
```

To obtain a direct form from a parallel form, the function par2dir can be used. It computes poles and residues of each proper biquad and combines these into system poles and residues. Another call of the residuez function in reverse order computes the numerator and denominator polynomials.

```
function [b,a] = par2dir(C,B,A);
% PARALLEL-to-DIRECT form conversion
% ----------------------------------
% [b,a] = par2dir(C,B,A)
%  b = numerator polynomial coefficients of DIRECT form
%  a = denominator polynomial coefficients of DIRECT form
%  C = Polynomial part of PARALLEL form
%  B = K by 2 matrix of real coefficients containing bk's
%  A = K by 3 matrix of real coefficients containing ak's
%
[K,L] = size(A); R = []; P = [];

for i=1:1:K
[r,p,k]=residuez(B(i,:),A(i,:));  R = [R;r]; P = [P;p];
end
[b,a] = residuez(R,P,C);  b = b(:)'; a = a(:)';
```

□ **EXAMPLE 6.2** Consider the filter given in Example 6.1.

$$\begin{aligned}16y(n) + 12y(n-1) + 2y(n-2) - 4y(n-3) - y(n-4)\\ = x(n) - 3x(n-1) + 11x(n-2) - 27x(n-3) + 18x(n-4)\end{aligned}$$

Now determine its parallel form.

Solution MATLAB script:

```
>> b=[1 -3 11 -27 18];  a=[16 12 2 -4 -1];
>> [C,B,A]=dir2par(b,a)
C =
   -18
B =
  10.0500   -3.9500
  28.1125  -13.3625
A =
   1.0000    1.0000    0.5000
   1.0000   -0.2500   -0.1250
```

The resulting structure is shown in Figure 6.8. To check our parallel structure, let us compute the first 8 samples of the impulse response using both forms.

```
>> format long; delta = impseq(0,0,7);  hpar=parfiltr(C,B,A,delta)
hpar =
  Columns 1 through 4
   0.06250000000000  -0.23437500000000   0.85546875000000  -2.28417968750000
```

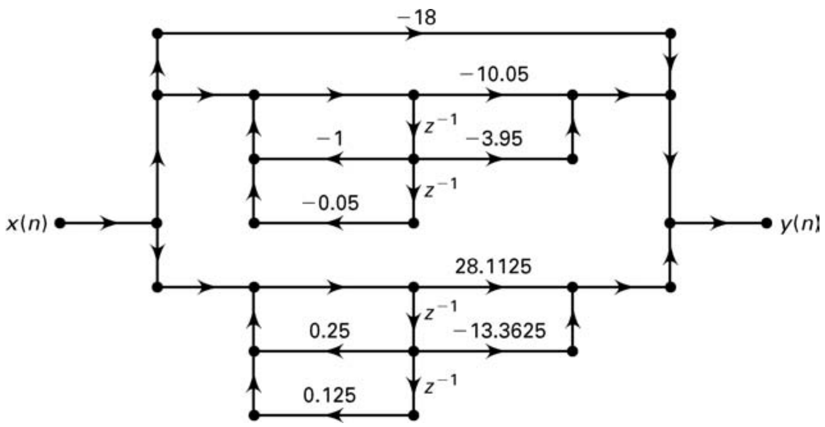


FIGURE 6.8 *Parallel form structure in Example 6.2*

```
  Columns 5 through 8
   2.67651367187500  -1.52264404296875   0.28984069824219   0.49931716918945
>> hdir = filter(b,a,delta)
hdir =
  Columns 1 through 4
   0.06250000000000  -0.23437500000000   0.85546875000000  -2.28417968750000
  Columns 5 through 8
   2.67651367187500  -1.52264404296875   0.28984069824219   0.49931716918945
```

□

□ **EXAMPLE 6.3** What would be the overall direct, cascade, or parallel form if a structure contains a combination of these forms? Consider the block diagram shown in Figure 6.9.

Solution This structure contains a cascade of two parallel sections. The first parallel section contains 2 biquads, while the second one contains 3 biquads. We will have to convert each parallel section into a direct form using the `par2dir` function, giving us a cascade of 2 direct forms. The overall direct form can be computed by convolving the corresponding numerator and denominator polynomials. The overall cascade and parallel forms can now be derived from the direct form.

MATLAB script:

```
>> C0=0; B1=[2 4;3 1]; A1=[1 1 0.9; 1 0.4 -0.4];
>> B2=[0.5 0.7;1.5 2.5;0.8 1]; A2=[1 -1 0.8;1 0.5 0.5;1 0 -0.5];
>> [b1,a1]=par2dir(C0,B1,A1)
b1 =
   5.0000    8.8000    4.5000   -0.7000
```

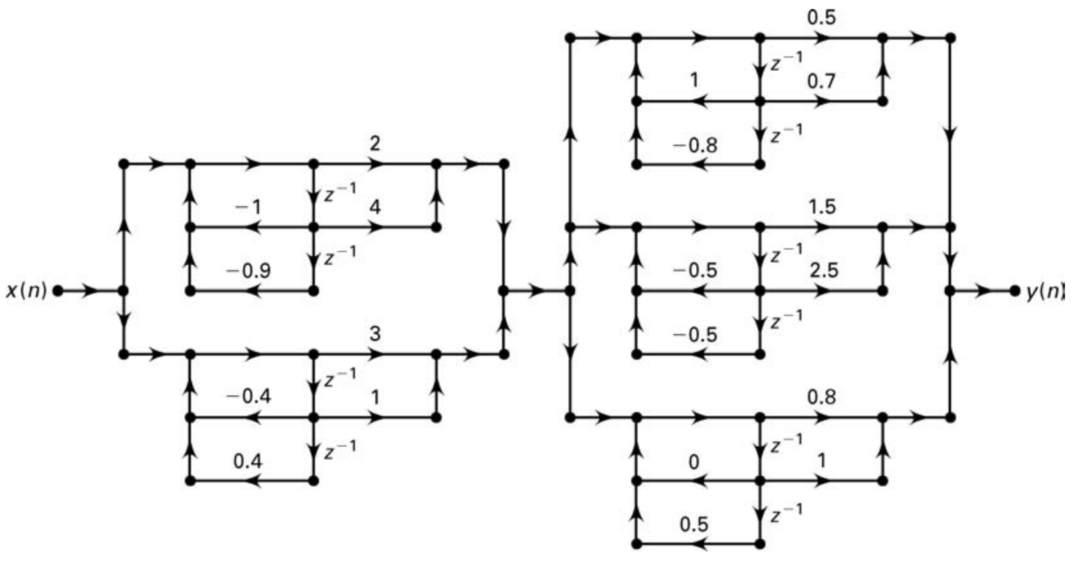


FIGURE 6.9 *Block diagram in Example 6.3*

```
a1 =
   1.0000    1.4000    0.9000   -0.0400   -0.3600
>> [b2,a2]=par2dir(C0,B2,A2)
b2 =
   2.8000    2.5500   -1.5600    2.0950    0.5700   -0.7750
a2 =
   1.0000   -0.5000    0.3000    0.1500    0.0000    0.0500   -0.2000
>> b=conv(b1,b2) % Overall direct form numerator
b =
  Columns 1 through 7
  14.0000   37.3900   27.2400    6.2620   12.4810   11.6605   -5.7215
  Columns 8 through 9
  -3.8865    0.5425
>> a=conv(a1,a2) % Overall direct form denominator
a =
  Columns 1 through 7
   1.0000    0.9000    0.5000    0.0800    0.1400    0.3530   -0.2440
  Columns 8 through 11
  -0.2890   -0.1820   -0.0100    0.0720
>> [b0,Bc,Ac]=dir2cas(b,a) % Overall cascade form
b0 =
  14.0000
Bc =
   1.0000    1.8836    1.1328
   1.0000   -0.6915    0.6719
   1.0000    2.0776    0.8666
   1.0000         0         0
   1.0000   -0.5990    0.0588
Ac =
   1.0000    1.0000    0.9000
   1.0000    0.5000    0.5000
   1.0000   -1.0000    0.8000
   1.0000    1.5704    0.6105
   1.0000   -1.1704    0.3276
>> [C0,Bp,Ap]=dir2par(b,a) % Overall parallel form
C0 = []
Bp =
  -20.4201   -1.6000
   24.1602    5.1448
    2.4570    3.3774
   -0.8101   -0.2382
    8.6129   -4.0439
Ap =
    1.0000    1.0000    0.9000
    1.0000    0.5000    0.5000
    1.0000   -1.0000    0.8000
    1.0000    1.5704    0.6105
    1.0000   -1.1704    0.3276
```

This example shows that by using the MATLAB functions developed in this section, we can probe and construct a wide variety of structures. □

6.3 FIR FILTER STRUCTURES

A finite-duration impulse response filter has a system function of the form

$$H(z) = b_0 + b_1 z^{-1} + \cdots + b_{M-1} z^{1-M} = \sum_{n=0}^{M-1} b_n z^{-n} \tag{6.5}$$

Hence the impulse response $h(n)$ is

$$h(n) = \begin{cases} b_n, & 0 \le n \le M-1 \\ 0, & \text{else} \end{cases} \tag{6.6}$$

and the difference equation representation is

$$y(n) = b_0 x(n) + b_1 x(n-1) + \cdots + b_{M-1} x(n-M+1) \tag{6.7}$$

which is a linear convolution of finite support.

The order of the filter is $M-1$, and the *length* of the filter (which is equal to the number of coefficients) is M. The FIR filter structures are always stable, and they are relatively simple compared to IIR structures. Furthermore, FIR filters can be designed to have a linear-phase response, which is desirable in some applications.

We will consider the following four structures:

1. **Direct form:** In this form the difference equation (6.7) is implemented directly as given.
2. **Cascade form:** In this form the system function $H(z)$ in (6.5) is factored into 2nd-order factors, which are then implemented in a cascade connection.
3. **Linear-phase form:** When an FIR filter has a linear-phase response, its impulse response exhibits certain symmetry conditions. In this form we exploit these symmetry relations to reduce multiplications by about half.
4. **Frequency-sampling form**: This structure is based on the DFT of the impulse response $h(n)$ and leads to a parallel structure. It is also suitable for a design technique based on the sampling of frequency response $H(e^{j\omega})$.

We will briefly describe these four forms along with some examples. The MATLAB function `dir2cas` developed in the previous section is also applicable for the cascade form.

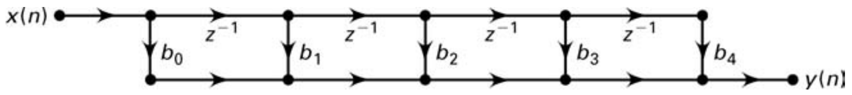


FIGURE 6.10 *Direct form FIR structure*

6.3.1 DIRECT FORM

The difference equation (6.7) is implemented as a tapped delay line since there are no feedback paths. Let $M = 5$ (i.e., a 4th-order FIR filter); then

$$y(n) = b_0x(n) + b_1x(n-1) + b_2x(n-2) + b_3x(n-3) + b_4x(n-4)$$

The direct form structure is given in Figure 6.10. Note that since the denominator is equal to unity, there is only one direct form structure.

6.3.2 MATLAB IMPLEMENTATION

In MATLAB the direct form FIR structure is described by the row vector `b` containing the $\{b_n\}$ coefficients. The structure is implemented by the `filter` function, in which the vector `a` is set to the scalar value 1, as discussed in Chapter 2.

6.3.3 CASCADE FORM

This form is similar to that of the IIR form. The system function $H(z)$ is converted into products of 2nd-order sections with real coefficients. These sections are implemented in direct form and the entire filter as a cascade of 2nd-order sections. From (6.5)

$$\begin{aligned} H(z) &= b_0 + b_1z^{-1} + \cdots + b_{M-1}z^{-M+1} \\ &= b_0\left(1 + \frac{b_1}{b_0}z^{-1} + \cdots + \frac{b_{M-1}}{b_0}z^{-M+1}\right) \\ &= b_0\prod_{k=1}^{K}\left(1 + B_{k,1}z^{-1} + B_{k,2}z^{-2}\right) \end{aligned} \tag{6.8}$$

where K is equal to $\lfloor \frac{M}{2} \rfloor$, and $B_{k,1}$ and $B_{k,2}$ are real numbers representing the coefficients of 2nd-order sections. For $M = 7$ the cascade form is shown in Figure 6.11.

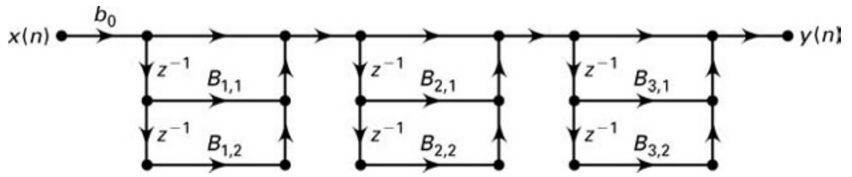


FIGURE 6.11 *Cascade form FIR structure*

6.3.4 MATLAB IMPLEMENTATION

Although it is possible to develop a new MATLAB function for the FIR cascade form, we will use our `dir2cas` function by setting the denominator vector `a` equal to 1. Similarly, `cas2dir` can be used to obtain the direct form from the cascade form.

6.3.5 LINEAR-PHASE FORM

For frequency-selective filters (e.g., lowpass filters) it is generally desirable to have a phase response that is a linear function of frequency; that is, we want

$$\angle H(e^{j\omega}) = \beta - \alpha\omega, \quad -\pi < \omega \leq \pi \tag{6.9}$$

where $\beta = 0$ or $\pm\pi/2$ and α is a constant. For a causal FIR filter with impulse response over $[0,\ M-1]$ interval, the linear-phase condition (6.9) imposes the following symmetry conditions on the impulse response $h(n)$ (see Problem P6.16):

$$h(n) = h(M-1-n); \quad \beta = 0, \alpha = \frac{M-1}{2}, \; 0 \leq n \leq M-1 \tag{6.10}$$

$$h(n) = -h(M-1-n); \quad \beta = \pm\pi/2, \alpha = \frac{M-1}{2}, \; 0 \leq n \leq M-1 \tag{6.11}$$

An impulse response that satisfies (6.10) is called a *symmetric impulse response*, and that in (6.11) is called an *antisymmetric impulse response*. These symmetry conditions can now be exploited in a structure called the linear-phase form.

Consider the difference equation given in (6.7) with a symmetric impulse response in (6.10). We have

$$\begin{aligned} y(n) &= b_0 x(n) + b_1 x(n-1) + \cdots + b_1 x(n-M+2) + b_0 x(n-M+1) \\ &= b_0[x(n) + x(n-M+1)] + b_1[x(n-1) + x(n-M+2)] + \cdots \end{aligned}$$

The block diagram implementation of these difference equation is shown in Figure 6.12 for both odd and even M.

Clearly, this structure requires 50% fewer multiplications than the direct form. A similar structure can be derived for an antisymmetric impulse response.

6.3.6 MATLAB IMPLEMENTATION

The linear-phase structure is essentially a direct form drawn differently to save on multiplications. Hence in a MATLAB representation of the linear-phase structure is equivalent to the direct form.

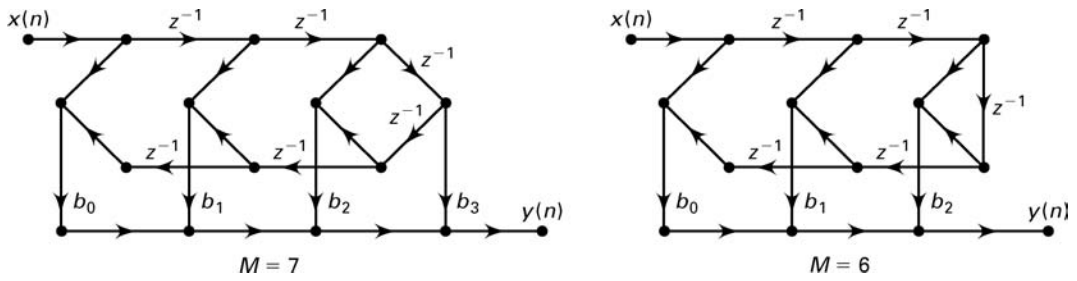


FIGURE 6.12 *Linear-phase form FIR structures (symmetric impulse response)*

□ **EXAMPLE 6.4** An FIR filter is given by the system function

$$H(z) = 1 + 16\frac{1}{16}z^{-4} + z^{-8}$$

Determine and draw the direct, linear-phase, and cascade form structures.

a. Direct form: The difference equation is given by

$$y(n) = x(n) + 16.0625x(n-4) + x(n-8)$$

and the direct form structure is shown in Figure 6.13(a).

b. Linear-phase form: The difference equation can be written in the form

$$y(n) = [x(n) + x(n-8)] + 16.0625x(n-4)$$

and the resulting structure is shown in Figure 6.13b.

c. Cascade form: We use the following MATLAB Script.

```
>> b=[1,0,0,0,16+1/16,0,0,0,1];  [b0,B,A] = dir2cas(b,1)
```

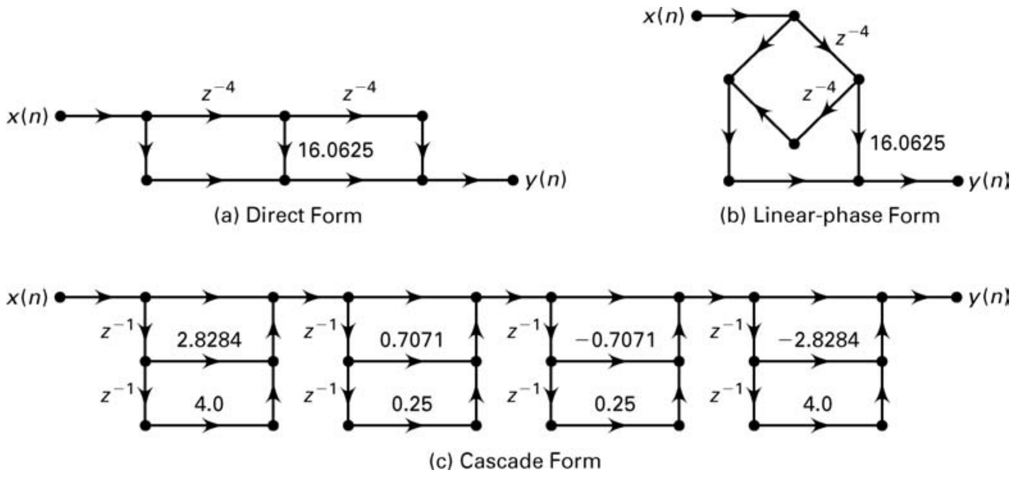


FIGURE 6.13 *FIR filter structures in Example 6.4*

```
b0 = 1
B =
    1.0000    2.8284    4.0000
    1.0000    0.7071    0.2500
    1.0000   -0.7071    0.2500
    1.0000   -2.8284    4.0000
A =
    1     0     0
    1     0     0
    1     0     0
    1     0     0
```

The cascade form structure is shown in Figure 6.13c. □

□ **EXAMPLE 6.5** For the filter in Example 6.4, what would be the structure if we desire a cascade form containing linear-phase components with real coefficients?

Solution We are interested in cascade sections that have symmetry and real coefficients. From the properties of linear-phase FIR filters (see Chapter 7), if such a filter has an arbitrary zero at $z = r\angle\theta$, then there must be 3 other zeros at $(1/r)\angle\theta$, $r\angle-\theta$, and $(1/r)\angle-\theta$ to have real filter coefficients. We can now make use of this property. First we will determine the zero locations of the given 8th-order polynomial. Then we will group 4 zeros that satisfy this property to obtain one (4th-order) linear-phase section. There are two such sections, which we will connect in cascade.

MATLAB script:

```
>> b=[1,0,0,0,16+1/16,0,0,0,1];  broots=roots(b)
broots =
  -1.4142 + 1.4142i
  -1.4142 - 1.4142i
   1.4142 + 1.4142i
   1.4142 - 1.4142i
  -0.3536 + 0.3536i
  -0.3536 - 0.3536i
   0.3536 + 0.3536i
   0.3536 - 0.3536i
>> B1=real(poly([broots(1),broots(2),broots(5),broots(6)]))
B1 =
    1.0000    3.5355    6.2500    3.5355    1.0000
>> B2=real(poly([broots(3),broots(4),broots(7),broots(8)]))
B2 =
    1.0000   -3.5355    6.2500   -3.5355    1.0000
```

The structure is shown in Figure 6.14. □

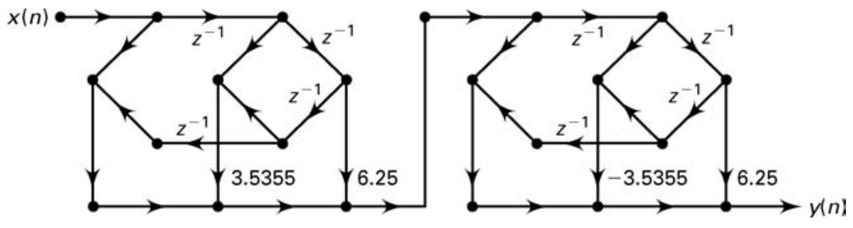


FIGURE 6.14 *Cascade of FIR linear-phase elements*

6.3.7 FREQUENCY SAMPLING FORM

In this form we use the fact that the system function $H(z)$ of an FIR filter can be reconstructed from its samples on the unit circle. From our discussions on the DFT in Chapter 5, we recall that these samples are in fact the M-point DFT values $\{H(k),\ 0 \le k \le M-1\}$ of the M-point impulse response $h(n)$. Therefore we have

$$H(z) = \mathcal{Z}[h(n)] = \mathcal{Z}[\text{IDFT}\{H(k)\}]$$

Using this procedure, we obtain [see (5.17) in Chapter 5]

$$H(z) = \left(\frac{1-z^{-M}}{M}\right)\sum_{k=0}^{M-1}\frac{H(k)}{1-W_M^{-k}z^{-1}} \tag{6.12}$$

This shows that the DFT $H(k)$, rather than the impulse response $h(n)$ (or the difference equation), is used in this structure. Also note that the FIR filter described by (6.12) has a recursive form similar to an IIR filter because (6.12) contains both poles and zeros. The resulting filter is an FIR filter since the poles at W_M^{-k} are canceled by the roots of

$$1 - z^{-M} = 0$$

The system function in (6.12) leads to a parallel structure, as shown in Figure 6.15 for $M = 4$.

One problem with the structure in Figure 6.15 is that it requires a complex arithmetic implementation. Since an FIR filter is almost always a real-valued filter, it is possible to obtain an alternate realization in which only real arithmetic is used. This realization is derived using the symmetry properties of the DFT and the W_M^{-k} factor. Then (6.12) can be expressed as (see Problem P6.19)

$$H(z) = \frac{1-z^{-M}}{M}\left\{\sum_{k=1}^{L}2\left|H(k)\right|H_k(z) + \frac{H(0)}{1-z^{-1}} + \frac{H(M/2)}{1+z^{-1}}\right\} \tag{6.13}$$

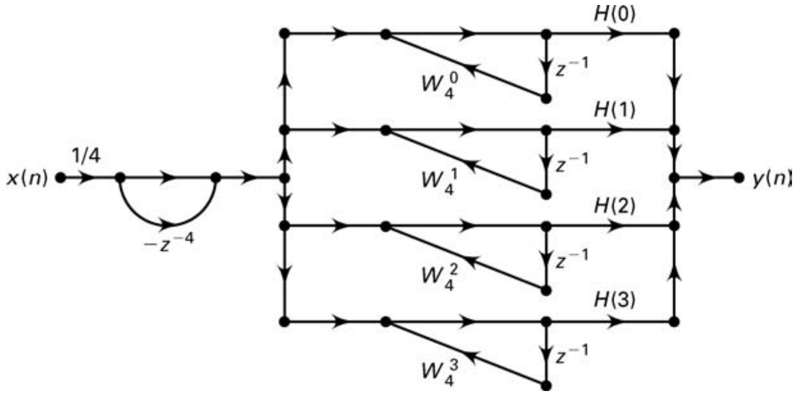


FIGURE 6.15 *Frequency sampling structure for $M = 4$*

where $L = \frac{M-1}{2}$ for M odd, $L = \frac{M}{2} - 1$ for M even, and $\{H_k(z),\ k = 1, \ldots, L\}$ are 2nd-order sections given by

$$H_k(z) = \frac{\cos[\angle H(k)] - z^{-1}\cos\left[\angle H(k) - \frac{2\pi k}{M}\right]}{1 - 2z^{-1}\cos\left(\frac{2\pi k}{M}\right) + z^{-2}} \tag{6.14}$$

Note that the DFT samples $H(0)$ and $H(M/2)$ are real-valued and that the third term on the right-hand side of (6.13) is absent if M is odd. Using (6.13) and (6.14), we show a frequency sampling structure in Figure 6.16 for $M = 4$ containing real coefficients.

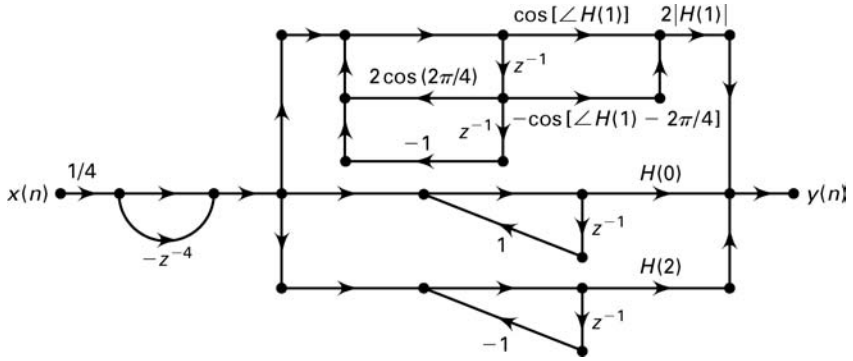


FIGURE 6.16 *Frequency sampling structure for $M = 4$ with real coefficients*

6.3.8 MATLAB IMPLEMENTATION

Given the impulse response $h(n)$ or the DFT $H(k)$, we have to determine the coefficients in (6.13) and (6.14). The following MATLAB function, `dir2fs`, converts a direct form [$h(n)$ values] to the frequency sampling form by directly implementing (6.13) and (6.14).

```
function [C,B,A] = dir2fs(h)
% Direct form to Frequency Sampling form conversion
% -------------------------------------------------
% [C,B,A] = dir2fs(h)
% C = Row vector containing gains for parallel sections
% B = Matrix containing numerator coefficients arranged in rows
% A = Matrix containing denominator coefficients arranged in rows
% h = impulse response vector of an FIR filter
%
M = length(h);  H = fft(h,M);
magH = abs(H); phaH = angle(H)';
% check even or odd M
if (M == 2*floor(M/2))
       L = M/2-1;   % M is even
      A1 = [1,-1,0;1,1,0];  C1 = [real(H(1)),real(H(L+2))];
else
       L = (M-1)/2; % M is odd
      A1 = [1,-1,0];  C1 = [real(H(1))];
end
k = [1:L]';
% initialize B and A arrays
B = zeros(L,2); A = ones(L,3);
% compute denominator coefficients
A(1:L,2) = -2*cos(2*pi*k/M); A = [A;A1];
% compute numerator coefficients
B(1:L,1) = cos(phaH(2:L+1));
B(1:L,2) = -cos(phaH(2:L+1)-(2*pi*k/M));
% compute gain coefficients
C = [2*magH(2:L+1),C1]';
```

In this function, the impulse response values are supplied through the `h` array. After conversion, the `C` array contains the gain values for each parallel section. The gain values for the 2nd-order parallel sections are given first, followed by $H(0)$ and $H(M/2)$ (if M is even). The `B` matrix contains the numerator coefficients, which are arranged in length-2 row vectors for each 2nd-order section. The `A` matrix contains the denominator coefficients, which are arranged in length-3 row vectors for the 2nd-order sections corresponding to those in `B`, followed by the coefficients for the 1st-order sections.

A practical problem with the structure in Figure 6.16 is that it has poles on the unit circle, which makes this filter critically unstable. If the filter is not excited by one of the pole frequencies, then the output is bounded. We can avoid this problem by sampling $H(z)$ on a circle $|z| = r$, where the radius r is very close to 1 but is less than 1 (e.g., $r = 0.99$), which results in

$$H(z) = \frac{1 - r^M z^{-M}}{M} \sum_{k=0}^{M-1} \frac{H(k)}{1 - rW_M^{-k} z^{-k}}; \quad H(k) = H\left(re^{j2\pi k/M}\right) \tag{6.15}$$

Now approximating $H\left(re^{j2\pi k/M}\right) \approx H\left(e^{j2\pi k/M}\right)$ for $r \approx 1$, we can obtain a stable structure similar to the one in Figure 6.16 containing real values. This is explored in Problem P6.20.

□ **EXAMPLE 6.6** Let $h(n) = \frac{1}{9}\{\underset{\uparrow}{1}, 2, 3, 2, 1\}$. Determine and draw the frequency sampling form.

Solution MATLAB script:

```
>> h = [1,2,3,2,1]/9;  [C,B,A] = dir2fs(h)
C =
    0.5818
    0.0849
    1.0000
B =
   -0.8090    0.8090
    0.3090   -0.3090
A =
    1.0000   -0.6180    1.0000
    1.0000    1.6180    1.0000
    1.0000   -1.0000         0
```

Since $M = 5$ is odd, there is only one 1st-order section. Hence

$$H(z) = \frac{1 - z^{-5}}{5} \left[0.5818 \frac{-0.809 + 0.809z^{-1}}{1 - 0.618z^{-1} + z^{-2}} + 0.0848 \frac{0.309 - 0.309z^{-1}}{1 + 1.618z^{-1} + z^{-2}} + \frac{1}{1 - z^{-1}} \right]$$

The frequency-sampling form is shown in Figure 6.17. □

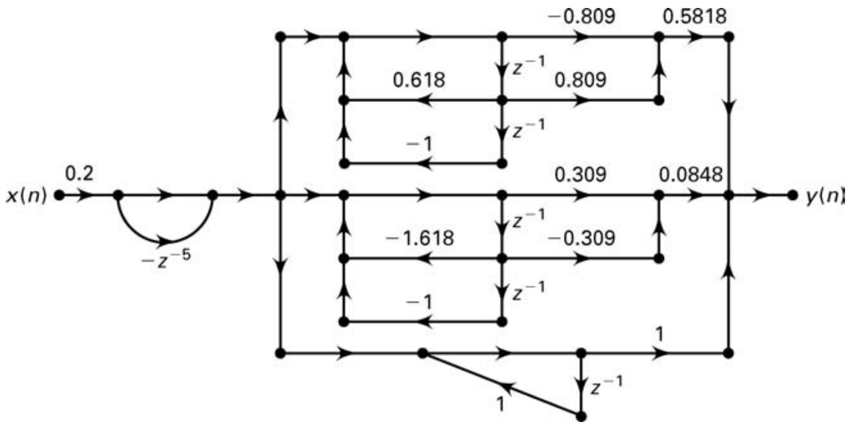


FIGURE 6.17 *Frequency sampling structure in Example 6.6*

□ **EXAMPLE 6.7** The frequency samples of a 32-point linear-phase FIR filter are given by

$$|H(k)| = \begin{cases} 1, & k = 0, 1, 2 \\ 0.5, & k = 3 \\ 0, & k = 4, 5, \ldots, 15 \end{cases}$$

Determine its frequency sampling form, and compare its computational complexity with the linear-phase form.

Solution In this example since the samples of the DFT $H(k)$ are given, we could use (6.13) and (6.14) directly to determine the structure. However, we will use the `dir2fs` function for which we will have to determine the impulse response $h(n)$. Using the symmetry property and the linear-phase constraint, we assemble the DFT $H(k)$ as

$$\begin{aligned} H(k) &= |H(k)|\, e^{j\angle H(k)}, \quad k = 0, 1, \ldots, 31 \\ |H(k)| &= |H(32-k)|, \quad k = 1, 2, \ldots, 31;\ H(0) = 1 \\ \angle H(k) &= -\frac{31}{2}\frac{2\pi}{32}k = -\angle H(32-k), \quad k = 0, 1, \ldots, 31 \end{aligned}$$

Now the IDFT of $H(k)$ will result in the desired impulse response.

MATLAB script:

```
>> M = 32; alpha = (M-1)/2;
>> magHk = [1,1,1,0.5,zeros(1,25),0.5,1,1];
>> k1 = 0:15; k2 = 16:M-1;
>> angHk = [-alpha*(2*pi)/M*k1, alpha*(2*pi)/M*(M-k2)];
>> H = magHk.*exp(j*angHk);  h = real(ifft(H,M));  [C,B,A] = dir2fs(h)
```

```
C =
    2.0000
    2.0000
    1.0000
    0.0000
    0.0000
    0.0000
    0.0000
         0
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    1.0000
         0
B =
   -0.9952    0.9952
    0.9808   -0.9808
   -0.9569    0.9569
   -0.8944    0.3162
    0.9794   -0.7121
    0.8265    0.2038
   -0.6754    0.8551
    1.0000    0.0000
    0.6866   -0.5792
    0.5191    0.9883
   -0.4430    0.4993
   -0.8944   -0.3162
   -0.2766    0.3039
    0.9343    0.9996
   -0.9077   -0.8084
A =
    1.0000   -1.9616    1.0000
    1.0000   -1.8478    1.0000
    1.0000   -1.6629    1.0000
    1.0000   -1.4142    1.0000
    1.0000   -1.1111    1.0000
    1.0000   -0.7654    1.0000
    1.0000   -0.3902    1.0000
    1.0000    0.0000    1.0000
    1.0000    0.3902    1.0000
    1.0000    0.7654    1.0000
    1.0000    1.1111    1.0000
    1.0000    1.4142    1.0000
    1.0000    1.6629    1.0000
    1.0000    1.8478    1.0000
    1.0000    1.9616    1.0000
    1.0000   -1.0000         0
    1.0000    1.0000         0
```

Note that only 4 gain coefficients are nonzero. Hence the frequency sampling form is

$$H(z) = \frac{1 - z^{-32}}{32} \left[\begin{array}{c} 2\dfrac{-0.9952 + 0.9952z^{-1}}{1 - 1.9616z^{-1} + z^{-2}} + 2\dfrac{0.9808 - 0.9808z^{-1}}{1 - 1.8478z^{-1} + z^{-2}} + \\ \dfrac{-0.9569 + 0.9569z^{-1}}{1 - 1.6629z^{-1} + z^{-2}} + \dfrac{1}{1 - z^{-1}} \end{array} \right]$$

To determine the computational complexity, note that since $H(0) = 1$, the 1st-order section requires no multiplication, whereas the three 2nd-order sections require 3 multiplications each for a total of 9 multiplications per output sample. The total number of additions is 13. To implement the linear-phase structure would require 16 multiplications and 31 additions per output sample. Therefore the frequency sampling structure of this FIR filter is more efficient than the linear-phase structure. □

6.4 LATTICE FILTER STRUCTURES

The lattice filter is extensively used in digital speech processing and in the implementation of adaptive filters. It is a preferred form of realization over other FIR or IIR filter structures because in speech analysis and in speech synthesis the small number of coefficients allows a large number of *formants* to be modeled in real time. The all-zero lattice is the FIR filter representation of the lattice filter, while the lattice ladder is the IIR filter representation.

6.4.1 ALL-ZERO LATTICE FILTERS

An FIR filter of length M (or order $M-1$) has a lattice structure with $M-1$ stages as shown in Figure 6.18. Each stage of the filter has an input and output that are related by the order-recursive equations [23]:

$$\begin{aligned} f_m(n) &= f_{m-1}(n) + K_m g_{m-1}(n-1), \quad m = 1, 2, \ldots, M-1 \\ g_m(n) &= K_m f_{m-1}(n) + g_{m-1}(n-1), \quad m = 1, 2, \ldots, M-1 \end{aligned} \tag{6.16}$$

where the parameters K_m, $m = 1, 2, \ldots, M-1$, called the *reflection coefficients*, are the lattice filter coefficients. If the initial values of $f_m(n)$ and $g_m(m)$ are both the scaled value (scaled by K_0) of the filter input

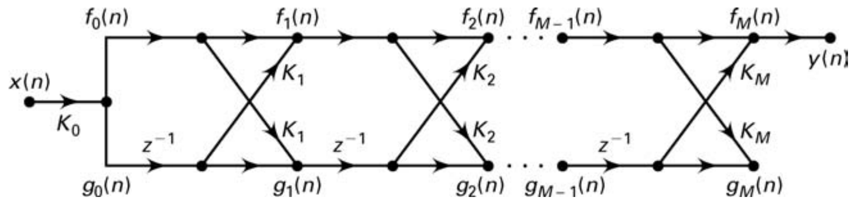


FIGURE 6.18 *All-zero lattice filter*

$x(n)$, then the output of the $(M-1)$ stage lattice filter corresponds to the output of an $(M-1)$ order FIR filter; that is,

$$\begin{aligned} f_0(n) &= g_0(n) = K_0 x(n) \\ y(n) &= f_{M-1}(n) \end{aligned} \tag{6.17}$$

If the FIR filter is given by the direct form

$$H(z) = \sum_{m=0}^{M-1} b_m z^{-m} = b_0 \left(1 + \sum_{m=1}^{M-1} \frac{b_m}{b_0} z^{-m}\right) \tag{6.18}$$

and if we denote the polynomial $A_{M-1}(z)$ by

$$\begin{aligned} A_{M-1}(z) &= \left(1 + \sum_{m=1}^{M-1} \alpha_{M-1}(m) z^{-m}\right); \\ \alpha_{M-1}(m) &= \frac{b_m}{b_0},\ m = 1, \ldots, M-1 \end{aligned} \tag{6.19}$$

then the lattice filter coefficients $\{K_m\}$ can be obtained by the following recursive algorithm [23]:

$$\begin{aligned} K_0 &= b_0 \\ K_{M-1} &= \alpha_{M-1}(M-1) \\ J_m(z) &= z^{-m} A_m\left(z^{-1}\right); && m = M-1, \ldots, 1 \\ A_{m-1}(z) &= \frac{A_m(z) - K_m J_m(z)}{1 - K_m^2}, && m = M-1, \ldots, 1 \\ K_m &= \alpha_m(m), && m = M-2, \ldots, 1 \end{aligned} \tag{6.20}$$

Note that this algorithm will fail if $|K_m| = 1$ for any $m = 1, \ldots, M-1$. Clearly, this condition is satisfied by linear-phase FIR filters since

$$b_0 = |b_{M-1}| \Rightarrow |K_{M-1}| = |\alpha_{M-1}(M-1)| = \left|\frac{b_{M-1}}{b_0}\right| = 1$$

Therefore, linear-phase FIR filters cannot be implemented using lattice structures.

6.4.2 MATLAB IMPLEMENTATION

Given the coefficients $\{b_n\}$ of the direct form, we can obtain the lattice filter coefficients $\{K_m\}$ using (6.20). This is done by the following MATLAB function `dir2latc`. Note that the equation to compute $J_m(z)$ implies that the polynomial $J_m(z)$ is a `fliplr` operation on the $A_m(z)$ polynomial.

```
function [K] = dir2latc(b)
% FIR Direct form to All-Zero Lattice form Conversion
% -----------------------------------------------------
% [K] = dir2latc(b)
%  K = Lattice filter coefficients (reflection coefficients)
%  b = FIR direct form coefficients (impulse response)
%
M = length(b);  K = zeros(1,M);  b1 = b(1);
if b1 == 0
error('b(1) is equal to zero')
end
K(1) = b1; A = b/b1;
for m=M:-1:2
K(m) = A(m);  J = fliplr(A);
A = (A-K(m)*J)/(1-K(m)*K(m));  A = A(1:m-1);
end
```

The lattice filter is implemented using (6.16) and (6.17), which is done by a `latcfilt` function, as shown here.

```
function [y] = latcfilt(K,x)
% LATTICE form realization of FIR filters
% ---------------------------------------
% y = latcfilt(K,x)
%   y = output sequence
%   K = LATTICE filter (reflection) coefficient array
%   x = input sequence
%
Nx = length(x)-1;  x = K(1)*x;
M = length(K)-1; K = K(2:M+1);  fg = [x; [0 x(1:Nx)]];
for m = 1:M
    fg = [1,K(m);K(m),1]*fg;
    fg(2,:) = [0 fg(2,1:Nx)];
end
y = fg(1,:);
```

The equations (6.20) can also be used to determine the direct, form coefficients $\{b_m\}$ from the lattice filter coefficients $\{K_m\}$ using a recursive procedure [22]:

$$\begin{aligned}
A_0(z) &= J_0(z) = 1 \\
A_m(z) &= A_{m-1}(z) + K_m z^{-1} J_{m-1}(z), && m = 1, 2, \ldots, M-1 \\
J_m(z) &= z^{-m} A_m\left(z^{-1}\right), && m = 1, 2, \ldots, M-1 \\
b_m &= K_0 \alpha_{M-1}(m), && m = 0, 1, \ldots, M-1
\end{aligned} \tag{6.21}$$

The following MATLAB function `latc2dir` implements (6.21). Note that the product $K_m z^{-1} J_{m-1}(z)$ is obtained by convolving the 2 corresponding arrays, whereas the polynomial $J_m(z)$ is obtained by using a `fliplr` operation on the $A_m(z)$ polynomial.

```
function [b] = latc2dir(K)
% All-Zero Lattice form to FIR Direct form Conversion
% ---------------------------------------------------
% [b] = latc2dir(K)
%  b = FIR direct form coefficients (impulse response)
%  K = Lattice filter coefficients (reflection coefficients)
%
M = length(K);  J = 1; A = 1;
for m=2:1:M
A = [A,0]+conv([0,K(m)],J);  J = fliplr(A);
end
b=A*K(1);
```

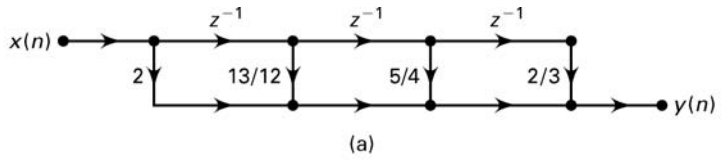
□ **EXAMPLE 6.8** An FIR filter is given by the difference equation

$$y(n) = 2x(n) + \frac{13}{12}x(n-1) + \frac{5}{4}x(n-2) + \frac{2}{3}x(n-3)$$

Determine its lattice form.

Solution MATLAB script:

```
>> b=[2, 13/12, 5/4, 2/3];  K=dir2latc(b)
K =
    2.0000    0.2500    0.5000    0.3333
```



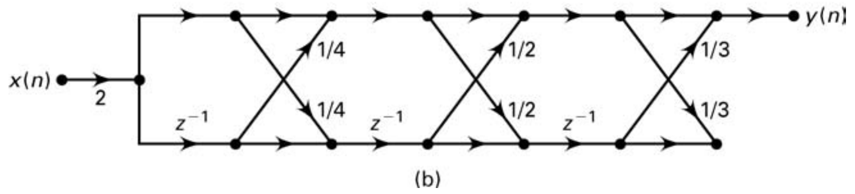


FIGURE 6.19 *FIR filter structures in Example 6.8: (a) direct form (b) lattice form*

Hence

$$K_0 = 2,\ K_1 = \frac{1}{4},\ K_2 = \frac{1}{2},\ K_3 = \frac{1}{3}$$

The direct form and the lattice form structures are shown in Figure 6.19. To check that our lattice structure is correct, let us compute the impulse response of the filter using both forms.

```
>> [x,n] = impseq(0,0,3];  format long  hdirect=filter(b,1,delta)
hdirect =
   2.00000000000000   1.08333333333333   1.25000000000000   0.66666666666667
>> hlattice=latcfilt(K,delta)
hlattice =
   2.00000000000000   1.08333333333333   1.25000000000000   0.66666666666667
```

□

6.4.3 ALL-POLE LATTICE FILTERS

A lattice structure for an IIR filter is restricted to an all-pole system function. It can be developed from an FIR lattice structure. Let an all-pole system function be given by

$$H(z) = \frac{1}{1 + \sum_{m=1}^{N} a_N(m) z^{-m}} \tag{6.22}$$

which from (6.19) is equal to $H(z) = 1/A_N(z)$. Clearly, it is an *inverse* system to the FIR lattice of Figure 6.18 (except for factor b_0). This IIR filter of order N has a lattice structure with N stages, as shown in Figure 6.20.

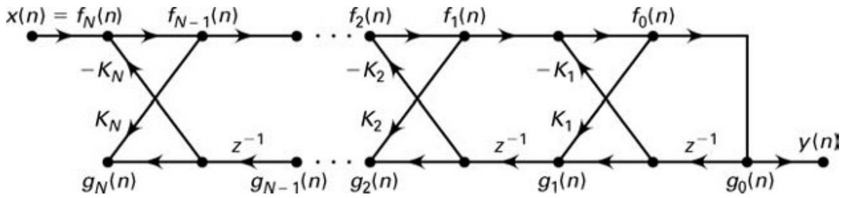


FIGURE 6.20 *All-pole lattice filter*

Each stage of the filter has an input and output that are related by the order-recursive equations [23]:

$$\begin{aligned} f_N(n) &= x(n) \\ f_{m-1}(n) &= f_m(n) - K_m g_{m-1}(n-1), \qquad m = N, N-1, \dots, 1 \\ g_m(n) &= K_m f_{m-1}(n) + g_{m-1}(n-1), \quad m = N, N-1, \dots, 1 \\ y(n) &= f_0(n) = g_0(n) \end{aligned} \tag{6.23}$$

where the parameters $K_m,\ m = 1, 2, \dots, M-1$, are the reflection coefficients of the all-pole lattice and are obtained from (6.20) except for K_0, which is equal to 1.

6.4.4 MATLAB IMPLEMENTATION

Since the IIR lattice coefficients are derived from the same (6.20) procedure used for an FIR lattice filter, we can use the `dir2latc` function in MATLAB. Care must be taken to ignore the K_0 coefficient in the `K` array. Similarly, the `latc2dir` function can be used to convert the lattice $\{K_m\}$ coefficients into the direct form $\{a_N(m)\}$, provided that $K_0 = 1$ is used as the first element of the `K` array. The implementation of an IIR lattice is given by (6.23), and we will discuss it in the next section.

□ **EXAMPLE 6.9** Consider an all-pole IIR filter given by

$$H(z) = \frac{1}{1 + \frac{13}{24}z^{-1} + \frac{5}{8}z^{-2} + \frac{1}{3}z^{-3}}$$

Determine its lattice structure.

Solution MATLAB script:

```
>> a=[1, 13/24, 5/8, 1/3];  K=dir2latc(a)
K =
    1.0000    0.2500    0.5000    0.3333
```

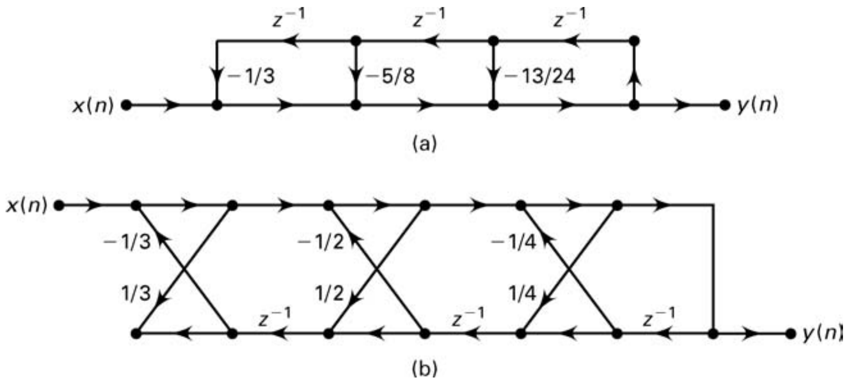


FIGURE 6.21 *IIR filter structures in Example 6.9: (a) direct form (b) lattice form*

Hence

$$K_1 = \frac{1}{4}, \qquad K_2 = \frac{1}{2}, \qquad \text{and} \qquad K_3 = \frac{1}{3}$$

The direct form and the lattice form structures of this IIR filter are shown in Figure 6.21. □

6.4.5 LATTICE-LADDER FILTERS

A general IIR filter containing both poles and zeros can be realized as a lattice-type structure by using an all-pole lattice as the basic building block. Consider an IIR filter with system function

$$H(z) = \frac{\sum_{k=0}^{M} b_M(k) z^{-k}}{1 + \sum_{k=1}^{N} a_N(k) z^{-k}} = \frac{B_M(z)}{A_N(z)} \tag{6.24}$$

where, without loss of generality, we assume that $N \geq M$. A lattice-type structure can be constructed by first realizing an all-pole lattice with coefficients K_m, $1 \leq m \leq N$ for the denominator of (6.24), and then adding a *ladder* part by taking the output as a weighted linear combination of $\{g_m(n)\}$, as shown in Figure 6.22 for $M = N$. The result is a pole-zero IIR filter that has the *lattice-ladder* structure. Its output is given by

$$y(n) = \sum_{m=0}^{M} C_m g_m(n) \tag{6.25}$$

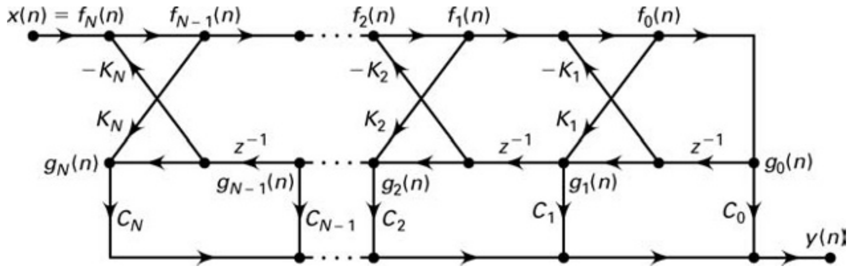


FIGURE 6.22 *Lattice-ladder structure for realizing a pole-zero IIR filter*

where $\{C_m\}$ are called the *ladder coefficients* that determine the zeros of the system function $H(z)$. It can be shown [23] that $\{C_m\}$ are given by

$$B_M(z) = \sum_{m=0}^{M} C_m J_m(z) \tag{6.26}$$

where $J_m(z)$ is the polynomial in (6.20). From (6.26) one can obtain a recursive relation

$$B_m(z) = B_{m-1}(z) + C_m J_m(z); \quad m = 1, 2, \ldots, M$$

or equivalently,

$$C_m = b_m + \sum_{i=m+1}^{M} C_i \alpha_i(i-m); \quad m = M,\ M-1, \ldots, 0 \tag{6.27}$$

from the definitions of $B_m(z)$ and $A_m(z)$.

6.4.6 MATLAB IMPLEMENTATION

To obtain a lattice-ladder structure for a general rational IIR filter, we can first obtain the lattice coefficients $\{K_m\}$ from $A_N(z)$ using the recursion (6.20). Then we can solve (6.27) recursively for the ladder coefficients $\{C_m\}$ to realize the numerator $B_M(z)$. This is done in the following MATLAB function `dir2ladr`. It can also be used to determine the all-pole lattice parameters when the array `b` is set to `b=[1]`.

```
function [K,C] = dir2ladr(b,a)
% IIR Direct form to pole-zero Lattice/Ladder form Conversion
% -----------------------------------------------------------
% [K,C] = dir2ladr(b,a)
%  K = Lattice coefficients (reflection coefficients), [K1,...,KN]
%  C = Ladder Coefficients, [C0,...,CN]
%  b = Numerator polynomial coefficients (deg <= Num deg)
%  a = Denominator polynomial coefficients
%
a1 = a(1); a = a/a1; b = b/a1;
M = length(b); N = length(a);
if M > N
     error('   *** length of b must be <= length of a ***')
end
b = [b, zeros(1,N-M)]; K = zeros(1,N-1);
A = zeros(N-1,N-1); C = b;
for m = N-1:-1:1
     A(m,1:m) = -a(2:m+1)*C(m+1);
     K(m) = a(m+1);  J = fliplr(a);
     a = (a-K(m)*J)/(1-K(m)*K(m));  a = a(1:m);
     C(m) = b(m) + sum(diag(A(m:N-1,1:N-m)));
end
```

Note: To use this function, $N \geq M$. If $M > N$, the numerator $A_N(z)$ should be divided into the denominator $B_M(z)$ using the `deconv` function to obtain a proper rational part and a polynomial part. The proper rational part can be implemented using a lattice-ladder structure, while the polynomial part is implemented using a direct structure.

To convert a lattice-ladder form into a direct form, we first use the recursive procedure in (6.21) on $\{K_m\}$ coefficients to determine $\{a_N(k)\}$ and then solve (6.27) recursively to obtain $\{b_M(k)\}$. This is done in the following MATLAB function `ladr2dir`.

```
function [b,a] = ladr2dir(K,C)
% Lattice/Ladder form to IIR Direct form Conversion
% -------------------------------------------------
% [b,a] = ladr2dir(K,C)
%  b = numerator polynomial coefficients
%  a = denominator polymonial coefficients
%  K = Lattice coefficients (reflection coefficients)
%  C = Ladder coefficients
%
N = length(K); M = length(C);
C = [C, zeros(1,N-M+1)];
```

```
J = 1; a = 1; A = zeros(N,N);
for m=1:1:N
     a = [a,0]+conv([0,K(m)],J);
     A(m,1:m) = -a(2:m+1);  J = fliplr(a);
end
b(N+1) = C(N+1);
for m = N:-1:1
     A(m,1:m) = A(m,1:m)*C(m+1);
     b(m) = C(m) - sum(diag(A(m:N,1:N-m+1)));
end
```

The lattice-ladder filter is implemented using (6.23) and (6.25). This is done in the following MATLAB function `ladrfilt`. It should be noted that, due to the recursive nature of this implementation along with the feedback loops, this MATLAB function is neither an elegant nor an efficient method of implementation. It is not possible to exploit MATLAB's inherent parallel processing capabilities in implementing this lattice-ladder structure.

```
function [y] = ladrfilt(K,C,x)
% LATTICE/LADDER form realization of IIR filters
% -----------------------------------------------
% [y] = ladrfilt(K,C,x)
% y = output sequence
% K = LATTICE (reflection) coefficient array
% C = LADDER coefficient array
% x = input sequence
%
Nx = length(x); y = zeros(1,Nx);
N = length(C); f = zeros(N,Nx); g = zeros(N,Nx+1);
f(N,:) = x;
for n = 2:1:Nx+1
    for m = N:-1:2
        f(m-1,n-1) = f(m,n-1) - K(m-1)*g(m-1,n-1);
        g(m,n) = K(m-1)*f(m-1,n-1) + g(m-1,n-1);
    end
    g(1,n) = f(1,n-1);
end
y = C*g(:,2:Nx+1);
```

□ **EXAMPLE 6.10** Convert the following pole-zero IIR filter into a lattice-ladder structure.

$$H(z) = \frac{1 + 2z^{-1} + 2z^{-2} + z^{-3}}{1 + \frac{13}{24}z^{-1} + \frac{5}{8}z^{-2} + \frac{1}{3}z^{-3}}$$

Solution MATLAB script:

```
>> b = [1,2,2,1] a = [1, 13/24, 5/8, 1/3];  [K,C] = dir2ladrc(b)
K =
    0.2500    0.5000    0.3333
C =
   -0.2695    0.8281    1.4583    1.0000
```

Hence

$$K_1 = \frac{1}{4},\ K_2 = \frac{1}{2},\ K_3 = \frac{1}{3};$$

and

$$C_0 = -0.2695,\ C_1 = 0.8281,\ C_2 = 1.4583,\ C_3 = 1$$

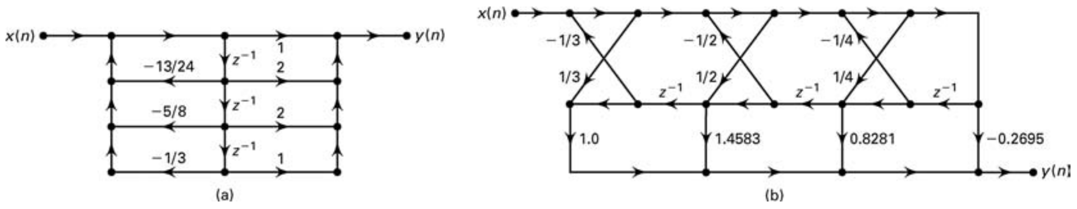


FIGURE 6.23 *IIR filter structures in Example 6.10: (a) direct form (b) lattice-ladder form*

The resulting direct form and the lattice-ladder form structures are shown in Figure 6.23. To check that our lattice-ladder structure is correct, let us compute the first 8 samples of its impulse response using both forms.

```
>> [x,n]=impseq(0,0,7)  format long  hdirect = filter(b,a,x)
hdirect =
  Columns 1 through 4
   1.00000000000000   1.45833333333333   0.58506944444444  -0.56170428240741
  Columns 5 through 8
  -0.54752302758488   0.45261700163162   0.28426911049255  -0.25435705167494
>> hladder = ladrfilt(K,C,x)
hladder =
  Columns 1 through 4
   1.00000000000000   1.45833333333333   0.58506944444444  -0.56170428240741
  Columns 5 through 8
  -0.54752302758488   0.45261700163162   0.28426911049255  -0.25435705167494
```

□

Finally, we note that the SP toolbox also provides functions similar to the ones discussed in this section—the complementary functions, `tf2latc` and `latc2tf`, compute all-pole lattice, all-zero lattice, and lattice-ladder structure coefficients, and vice versa. Similarly, the function `latcfilt` (the same name as the book function) implements the all-zero lattice structure. The SP toolbox does not provide a function to implement the lattice-ladder structure.

6.5 OVERVIEW OF FINITE-PRECISION NUMERICAL EFFECTS

Until now we have considered digital filter designs and implementations in which both the filter coefficients and the filter operations such as additions and multiplications were expressed using infinite-precision numbers. When discrete-time systems are implemented in hardware or in software, all parameters and arithmetic operations are implemented using finite-precision numbers and hence their effect is unavoidable.

Consider a typical digital filter implemented as a direct-form II structure, which is shown in Figure 6.24a. When finite-precision representation is used in its implementation, there are three possible considerations that affect the overall quality of its output. We have to

1. quantize filter coefficients, $\{a_k, b_k\}$, to obtain their finite word-length representations, $\{\hat{a}_k, \hat{b}_k\}$,
2. quantize the input sequence, $x(n)$ to obtain $\hat{x}(n)$, and
3. consider all internal arithmetic that must be converted to their next best representations.

Thus, the output, $y(n)$, is also a quantized value $\hat{y}(n)$. This gives us a new filter realization, $\hat{H}(z)$, which is shown in Figure 6.24b. We hope that this

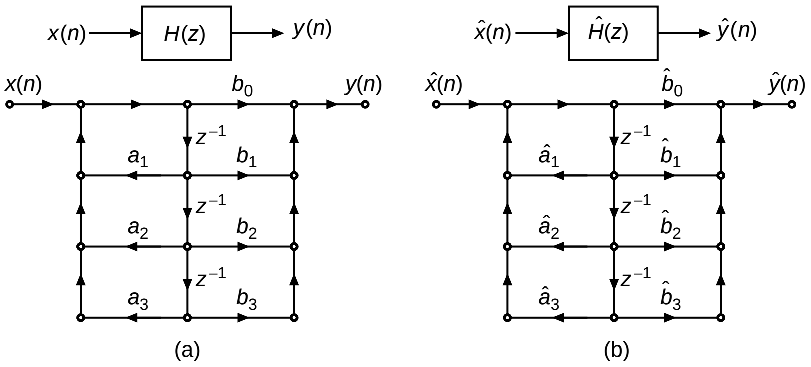


FIGURE 6.24 *Direct-form II digital filter implementation: (a) Infinite precision, (b) Finite precision*

new filter $\hat{H}(z)$ and its output $\hat{y}(n)$ are as close as possible to the original filter $H(z)$ and the original output $y(n)$.

Since the quantization operation is a nonlinear operation, the overall analysis that takes into account all three effects described above is very difficult and tedious. Therefore, we will study each of these effects separately as though it were the only one acting at the time. This makes the analysis easier and the results more interpretable.

We begin by discussing the number representation in a computer—more accurately, a central processing unit (CPU). This leads to the process of number quantization and the resulting error characterization. We then analyze the effects of filter coefficient quantization on digital filter frequency responses. The effects of multiplication and addition quantization (collectively known as arithmetic round-off errors) on filter output are discussed in Chapter 10.

6.6 REPRESENTATION OF NUMBERS

In computers, numbers (real-valued or complex-valued, integers or fractions) are represented using binary digits (*bits*), which take the value of either a 0 or a 1. The finite word-length arithmetic needed for processing these numbers is implemented using two different approaches, depending on the ease of implementation and the accuracy as well as dynamic range needed in processing. The *fixed-point* arithmetic is easy to implement but has only a fixed dynamic range and accuracy (i.e., very large numbers or very small numbers). The *floating-point* arithmetic, on the other hand, has a wide dynamic range and a variable accuracy (relative to the magnitude of a number) but is more complicated to implement and analyze.

Since a computer can operate only on a binary variable (e.g., a 1 or a 0), positive numbers can straightforwardly be represented using binary numbers. The problem arises as to how to represent the negative numbers. There are three different formats used in each of these arithmetics: *sign-magnitude* format, *one's-complement* format, and *two's-complement* format. In discussing and analyzing these representations, we will mostly consider a binary number system containing bits. However, this discussion and analysis is also valid for any radix numbering system—for example, the hexadecimal, octal, or decimal system.

In the following discussion, we will first begin with fixed-point signed integer arithmetic. A B-bit binary representation of an integer x is given by[1]

$$x \equiv b_{B-1}\, b_{B-2}\, \ldots\, b_0 = b_{B-1} \times 2^{B-1} + b_{B-2} \times 2^{B-2} + \cdots + b_0 \times 2^0 \qquad \textbf{(6.28)}$$

[1]Here the letter b is used to represent a binary bit. It is also used for filter coefficients $\{b_k\}$. Its use in the text should be clear from the context.

where each bit b_i represents either a 0 or a 1. This representation will help us to understand the advantages and disadvantages of each signed format and to develop simple MATLAB functions. We will then extend these concepts to fractional real numbers for both fixed-point and floating-point arithmetic.

6.6.1 FIXED-POINT SIGNED INTEGER ARITHMETIC

In this arithmetic, positive numbers are coded using their binary representation. For example, using 3 bits, we can represent numbers from 0 to 7 as

```
 0    1    2    3    4    5    6    7
-+----+----+----+----+----+----+----+-
000  001  010  011  100  101  110  111
```

Thus, with 8 bits the numbers represented can be 0 to 255, with 10 bits we can represent the numbers from 0 to 1023, and with 16 bits the range covered is 0 to 65535. For negative numbers, the following three formats are used: sign-magnitude, one's-complement, and two's-complement.

Sign-magnitude format In this format, positive numbers are represented using bits as before. However, the leftmost bit (also known as the most-significant bit or MSB) is used as the sign bit (0 is +, and 1 is −), and the remaining bits hold the absolute magnitude of the number as shown here:

```
Sign Bit
  -+    Absolute Magnitude
 +---+----------------------+
 |   |                      |
 +---+----------------------+
```

This system has thus two different codes for 0, one for the positive 0, the other one for the negative 0. For example, using 3 bits, we can represent numbers from −3 to 3 as

```
-3   -2   -1   -0    0    1    2    3
-+----+----+----+----+----+----+----+-
111  110  101  100  000  001  010  011
```

Thus, 8 bits cover the interval $[-127, +127]$, while 16 bits cover $[-32,767, +32,767]$. If we use B bits in the sign-magnitude format, then we can represent integers from $-(2^{B-1}-1)$ to $+(2^{B-1}-1)$ only.

This format has two drawbacks. First, there are two representations for 0. Second, the arithmetic using sign-magnitude format requires one rule to compute addition, another rule to compute subtraction, and a way to compare two magnitudes to determine their relative value before subtraction.

MATLAB Implementation MATLAB is a 64-bit floating-point computation engine that provides results in decimal numbers. Therefore, fixed-point binary operations must be simulated in MATLAB. It provides a function, `dec2bin`, to convert a positive decimal integer into a `B`-bit representation, which is a symbol (or a code) and not a number. Hence it cannot be used in computation. Similarly, the function `bin2dec` converts a `B`-bit binary character code into a decimal integer. For example, `dec2bin(3,3)` gives `011` and `bin2dec('111')` results in `7`. To obtain a sign-magnitude format, a sign bit must be prefixed. Similarly, to convert a sign-magnitude format, the leading bit must be used to impart a positive or negative value. These functions are explored in Problem P9.1.

One's-complement format In this format, the negation (or complementation) of an integer x is obtained by complementing every bit (i.e., a 0 is replaced by 1 and a 1 by 0) in the binary representation of x. Suppose the B-bit binary representation of x is $b_{B-1}\,b_{B-2}\,\cdots\,b_0$; then the B-bit *one's-complement*, $\bar{x}$, of x is given by

$$\bar{x} \triangleq \bar{b}_{B-1}\,\bar{b}_{B-2}\,\cdots\,\bar{b}_0$$

where each bit $\bar{b}_i$ is a complement of bit b_i. Clearly then

$$x + \bar{x} \equiv 1\,1\,\ldots\,1 = 2^B - 1 \tag{6.29}$$

The MSB of the representation once again represents the sign bit, because the positive integer has the MSB of 0 so that its negation (or a negative integer) has the MSB of 1. The remaining bits represent either the number x (if positive) or its one's-complement (if negative). Thus, using (6.29) the one's-complement format representation[2] is given by

$$x_{(1)} \triangleq \begin{cases} x, & x \geq 0 \\ |\overline{x}|, & x < 0 \end{cases} = \begin{cases} x, & x \geq 0 \\ 2^B - 1 - |x|, & x < 0 \end{cases} = \begin{cases} x, & x \geq 0 \\ 2^B - 1 + x, & x < 0 \end{cases} \tag{6.30}$$

Clearly, if B bits are available, then we can represent only integers from $(-2^{B-1}+1)$ to $(+2^{B-1}-1)$, which is similar to the sign-magnitude format.

[2]The one's-complement format refers to the representation of positive and negative numbers, whereas the one's-complement of a number refers to the negation of that number.

For example, using 3 bits, we can represent numbers from -3 to 3 as

```
-3   -2   -1   -0    0    1    2    3
-+----+----+----+----+----+----+----+-
100  101  110  111  000  001  010  011
```

which is a different bit arrangement for negative numbers compared to the sign-magnitude format.

The advantage of this format is that subtraction can be achieved by adding the complement, which is very easy to obtain by simply complementing a number's bits. However, there are many drawbacks. There are still two different codes for 0. The addition is a bit tricky to implement, and overflow management requires addition of the overflow bit to the least significant bit (or 2^0).

MATLAB Implementation The 1s-complement of a positive integer x using B bits can be obtained by using the built-in function `bitcmp(x,B)`, which complements the number's bits. The result is a decimal number between 0 and $2^B - 1$. As before, the `dec2bin` can be used to obtain the binary code. Using (6.30), we can develop the MATLAB function, `OnesComplement`, which obtains the one's-complement format representation. It uses the sign of a number to determine when to use one's-complement and can use scalar as well as vector values. The result is a decimal equivalent of the representation.

```
function y = OnesComplement(x,B)
% y = OnesComplement(x,B)
% ---------------
% Decimal equivalent of
%  Sign-Magnitude format integer to b-bit Ones'-Complement format conversion
%
%    x: integer between -2^(b-1) <  x <  2^(b-1) (sign-magnitude)
%    y: integer between        0 <= y <= 2^b-1   (1's-complement)

if any((x <= -2^(B-1) | (x >= 2^(B-1))))
    error('Numbers must satisfy -2^(B-1) < x <  2^(B-1)')
end
s = sign(x);  % sign of x (-1 if x<0, 0 if x=0, 1 if x>0)
sb = (s < 0); % sign-bit  (0 if x>=0, 1 if x<0));
y = (1-sb).*x + sb.*bitcmp(abs(x),B);
```

□ **EXAMPLE 6.11** Using the function `OnesComplement`, obtain one's-complement format representation of integers from -7 to 7 using 4 bits.

Solution MATLAB script:

```
>> x = -7:7
x =
   -7   -6   -5   -4   -3   -2   -1    0    1    2    3    4    5    6    7
>> y = OnesComplement(x,4)
y =
    8    9   10   11   12   13   14    0    1    2    3    4    5    6    7
```

Note that the number 15 is missing since we do not have -0 in our original array. □

Two's-complement format The disadvantage of having two codes for the number 0 is eliminated in this format. Positive numbers are coded as usual. The B-bit two's-complement, $\tilde{x}$, of a positive integer x is given by

$$\tilde{x} = \bar{x} + 1 = 2^B - x \quad \text{or} \quad x + \tilde{x} = 2^B \tag{6.31}$$

where the second equality is obtained from (6.30). Once again, the MSB of the representation provides the sign bit. Thus, using (6.31) the two's-complement format representation[3] is given by

$$x_{(2)} = \begin{cases} x, & x \geq 0 \\ |\tilde{x}|, & x < 0 \end{cases} = \begin{cases} x, & x \geq 0 \\ 2^B - |x|, & x < 0 \end{cases} = \begin{cases} x, & x \geq 0 \\ 2^B + x, & x < 0 \end{cases} \tag{6.32}$$

Thus, in B-bit two's-complement format negative numbers are obtained by adding 2^B to them. Clearly, if B bits are available, then we can represent 2^B integers from (-2^{B-1}) to $(+2^{B-1} - 1)$. For example, using 3 bits, we can represent numbers from -4 to 3 as

```
-4   -3   -2   -1    0    1    2    3
-+----+----+----+----+----+----+----+-
100  101  110  111  000  001  010  011
```

This format, by shifting to the right (e.g., by incrementing) the code of the negative numbers, straightforwardly removes the problem of having 2 codes for 0 and gives access to an additional negative number at the left of the line. Thus, 4 bits go from -8 to $+7$, 8 bits cover the interval $[-127, +127]$ and 16 bits cover $[-32768, +32767]$.

[3]Again, the two's-complement format refers to the representation of positive and negative numbers, whereas the two's-complement of a number refers to the negation of that number.

MATLAB Implementation Using (6.32), we can develop the MATLAB function, `TwosComplement`, which obtains the two's-complement format representation. We can use the `bitcmp` function and then add one to the result to obtain the two's-complement. However, we will use the last equality in (6.32) to obtain the two's-complement since this approach will also be useful for fractional numbers. The function can use scalar as well as vector values. The result is a decimal equivalent of the two's-complement representation. As before, the `dec2bin` can be used to obtain the binary code.

```
function y = TwosComplement(x,b)
% y = TwosComplement(x,b)
% ---------------
% Decimal equivalent of
%  Sign-Magnitude format integer to b-bit Ones'-Complement format conversion
%
%    x: integer between -2^(b-1) <= x <  2^(b-1) (sign-magnitude)
%    y: integer between        0 <= y <= 2^b-1   (2's-complement)
if any((x < -2^(b-1) | (x >= 2^(b-1))))
    error('Numbers must satisfy -2^(b-1) <= x <  2^(b-1)')
end
s = sign(x);  % sign of x (-1 if x<0, 0 if x=0, 1 if x>0)
sb = (s < 0); % sign-bit  (0 if x>=0, 1 if x<0));
y = (1-sb).*x + sb.*(2^b+x); % or y = (1-sb).*x + sb.*(bitcmp(abs(x),b)+1);
```

☐ **EXAMPLE 6.12** Using the function `TwosComplement`, obtain the two's-complement format representation of integers from -8 to 7 using 4 bits.

Solution MATLAB script:

```
>> x = -8:7
x =
   -8   -7   -6   -5   -4   -3   -2   -1    0    1    2    3    4    5    6    7
>> y = TwosComplement(x,4)
y =
    8    9   10   11   12   13   14   15    0    1    2    3    4    5    6    7
>> y = dec2bin(y,4); disp(sprintf('%s',[y';char(ones(1,16)*32)]))
1000 1001 1010 1011 1100 1101 1110 1111 0000 0001 0010 0011 0100 0101 0110 0111
```

☐

The two's-complement format has many interesting characteristics and advantages. These will be given after we discuss the next format, namely the ten's-complement.

Ten's-complement format This is a representation for decimal integers. We will describe it so that we can explore characteristics of two's-complement through decimal integers, which is much easier to understand. Following (6.31), the N-digit ten's-complement of a positive integer x is given by

$$\tilde{x} = 10^N - x \quad or \quad x + \tilde{x} = 10^N \tag{6.33}$$

Using (6.33), the N-digit ten's-complement format representation is given by

$$x_{(10^N)} \triangleq \begin{cases} x, & x \geq 0 \\ |\tilde{x}|, & x < 0 \end{cases} = \begin{cases} x, & x \geq 0 \\ 10^N - |x|, & x < 0 \end{cases} = \begin{cases} x, & x \geq 0 \\ 10^N + x, & x < 0 \end{cases} \tag{6.34}$$

Thus, in N-digit ten's-complement format (which is sometimes referred to as 10^N-complement format), negative numbers are obtained by adding 10^N to them. Clearly, when N digits are available, we can represent 10^N integers from $(-\frac{10^N}{2})$ to $(+\frac{10^N}{2} - 1)$. For example, using 1 digit, we can represent numbers from -5 to 4 as

```
-5   -4   -3   -2   -1    0    1    2    3    4
-+----+----+----+----+----+----+----+----+----+
 5    6    7    8    9    0    1    2    3    4
```

☐ **EXAMPLE 6.13** Using the 2-digit ten's-complement, i.e., 100s-complement format, perform the following operations:
1. $16 - 32$, 2. $32 - 16$, 3. $-30 - 40$, 4. $40 + 20 - 30$, 5. $-40 - 20 + 30$.

Solution

1. $16 - 32$
   First we note that $16 - 32 = -16$. If we use the usual subtraction rule to proceed from right to left generating carries in the process, we cannot complete the operation. To use the 100s-complement format, we first note that in the 100s-complement format we have

   $$16_{(100)} = 16, \quad -16_{(100)} = 100 - 16 = 84, \quad \text{and} \quad -32_{(100)} = 100 - 32 = 68$$

   Hence $16 - 32 \equiv 16 + 68 = 84 \equiv -16$ in the sign-magnitude format as expected.
2. $32 - 16$
   In this case the 100s-complement format gives

   $$32 + 84 = 116 \equiv 16$$

   in the sign-magnitude format by ignoring the generated carry digit. This is because the sign bits were different; therefore, the operation cannot generate an overflow. Hence, we check for overflow only if the sign bits are same.
3. $-30 - 40$
   In this case the 100s-complement format gives

   $$(100 - 30) + (100 - 40) = 70 + 60 = 130$$

Since the sign bits were the same, an overflow is generated and the result is invalid.

4. $40 + 20 - 30$
   This is an example of more than one addition or subtraction. Since the final result is well within the range, the overflow can be ignored—that is,

$$40 + 20 + (100 - 30) = 40 + 20 + 70 = 130 \equiv 30$$

   which is a correct result.
5. $-40 - 20 + 30$
   In this case, we have

$$(100 - 40) + (100 - 20) + 30 = 60 + 80 + 30 = 170 \equiv -30$$

   in the sign-magnitude format, which is, again, a correct result. □

MATLAB Implementation Using (6.34), one can develop the MATLAB function, `TensComplement`, which obtains ten's-complement format representation. It is similar to the `TwosComplement` function and is explored in Problem P6.25.

Advantages of two's-complement format Using the results of the Example 6.13, we now state the benefits of the two's-complement format. These also hold (with obvious modifications) for the ten's-complement format.

1. It provides for all 2^{B+1} distinct representations for a B-bit fractional representation. There is only one representation for zero.
2. This complement is compatible with our notion of negation: the complement of a complement is the number itself.
3. It unifies the subtraction and addition operations (subtractions are essentially additions).
4. In a sum of more than two numbers, the internal overflows do not affect the final result so long as the result is within the range (i.e., adding two positive numbers gives a positive result, and adding two negative numbers gives a negative result).

Hence in most A/D converters and processors, negative numbers are represented using two's-complement format. Almost all current processors implement signed arithmetic using this format and provide special functions (e.g., an overflow flag) to support it.

Excess-2^{B-1} format This format is used in describing the exponent of floating-point arithmetic; hence it is briefly discussed here. In excess-2^{B-1} signed format (also known as a *biased* format), all positive and negative integers between -2^{B-1} and $2^{B-1}-1$ are given by

$$x_{(e)} \triangleq 2^{B-1} + x \tag{6.35}$$

For example, using 3 bits, we can represent the numbers from -4 to 3 as

```
-4   -3   -2   -1    0    1    2    3
-+----+----+----+----+----+----+----+-
000  001  010  011  100  101  110  111
```

Notice that this format is very similar to the two's-complement format, but the sign bit is complemented. The arithmetic for this format is similar to that of the two's-complement format. It is used in the exponent of floating-point number representation.

6.6.2 GENERAL FIXED-POINT ARITHMETIC

Using the discussion of integer arithmetic from the last section as a guide, we can extend the fixed-point representation to arbitrary real (integer and fractional) numbers. We assume that a given infinite-precision real number, x, is approximated by a binary number, $\hat{x}$, with the following bit arrangement:

$$\hat{x} = \underset{\substack{\uparrow \\ \text{Sign bit}}}{\pm} \quad \underbrace{\mathtt{xx\cdots x}}_{\substack{\text{“}L\text{”} \\ \text{Integer bits}}} \blacktriangle \underbrace{\mathtt{xx\cdots x}}_{\substack{\text{“}B\text{”} \\ \text{Fraction bits}}} \tag{6.36}$$

where the sign bit $\pm$ is 0 for positive numbers and 1 for negative numbers, x represents either a 0 or a 1, and $\blacktriangle$ represents the *binary point.* This representation is in fact the sign-magnitude format for real numbers, as we will see. The total *word length* of the number $\hat{x}$ is then equal to $L+B+1$ bits.

□ **EXAMPLE 6.14** Let $L = 4$ and $B = 5$, which means $\hat{x}$ is a 10-bit number. Represent $11010\blacktriangle01110$ in decimal.

Solution

$$\begin{aligned}\hat{x} &= -(1\times 2^3 + 0\times 2^2 + 1\times 2^1 + 0\times 2^0 + 0\times 2^{-1} + 1\times 2^{-2} + 1\times 2^{-3} + 1\times 2^{-4} + 0\times 2^{-5}) \\ &= -10.4375\end{aligned}$$

in decimal. □

In many A/D converters and processors, the real numbers are scaled so that the fixed-point representation is in the $(-1, 1)$ range. This has the advantage that the multiplication of two fractions is always a fraction and, as such, there is no overflow. Hence we will consider the following representation:

$$\hat{x} = A(\pm\blacktriangle \underbrace{\mathtt{xxxxxx}\cdots\mathtt{x}}_{B\,\text{fraction bits}}) \tag{6.37}$$

where A is a positive *scaling factor*.

□ **EXAMPLE 6.15** Represent the number $\hat{x} = -10.4375$ in Example 6.14 using a fraction-only arrangement.

Solution Choose $A = 2^4 = 16$ and $B = 9$. Then

$$\hat{x} = -10.4375 = 16\,(1_\blacktriangle 101001110)$$

Hence by properly choosing A and B, one can obtain any fraction-only representation.

Note: The scalar A need not be a power of 2. In fact, by choosing any real number A we can obtain an arbitrary range. The power-of-2 choice for A, however, makes hardware implementation a little easier. □

As discussed in the previous section, there are three main formats for fixed-point arithmetic, depending on how negative numbers are obtained. For all these formats, positive numbers have exactly the same representation. In the following we assume the fraction-only arrangement.

Sign-magnitude format As the name suggests, the magnitude is given by the B-bit fraction, and the sign is given by the MSB. Thus,

$$\hat{x} = \begin{cases} \mathtt{0}_\blacktriangle \mathtt{x}_1\mathtt{x}_2\cdots\mathtt{x}_B & \text{if } x \geq 0 \\ \mathtt{1}_\blacktriangle \mathtt{x}_1\mathtt{x}_2\cdots\mathtt{x}_B & \text{if } x < 0 \end{cases} \tag{6.38}$$

For example, when $B = 2$, $\hat{x} = +1/4$ is represented by $\hat{x} = 0_\blacktriangle 01$, and $\hat{x} = -1/4$ is represented by $\hat{x} = 1_\blacktriangle 01$.

One's-complement format In this format, the positive numbers have the same representation as the sign-magnitude format. When the number is negative, then its magnitude is given by its bit-complement arrangement. Thus,

$$\hat{x} = \begin{cases} \mathtt{0}_\blacktriangle \mathtt{x}_1\mathtt{x}_2\cdots\mathtt{x}_B & \text{if } x \geq 0 \\ \mathtt{1}_\blacktriangle \bar{\mathtt{x}}_1\bar{\mathtt{x}}_2\cdots\bar{\mathtt{x}}_B & \text{if } x < 0 \end{cases} \tag{6.39}$$

For example, when $B = 2$, $\hat{x} = +1/4$ is represented by $\hat{x} = 0_\blacktriangle 01$, and $\hat{x} = -1/4$ is represented by $\hat{x} = 1_\blacktriangle 10$.

Two's-complement format Once again, the positive numbers have the same representation. Negative numbers are obtained by first complementing the magnitude and then modulo-2 adding one to the last bit or the *least-significant bit* (LSB). Stated differently, two's-complement is formed by subtracting the magnitude of the number from 2. Thus

$$\hat{x} = \begin{cases} 0_\blacktriangle \mathtt{x}_1\mathtt{x}_2\cdots\mathtt{x}_B & \text{if } x \ge 0 \\ 2-|x| = 1_\blacktriangle \bar{\mathtt{x}}_1\bar{\mathtt{x}}_2\cdots\bar{\mathtt{x}}_B \oplus 0_\blacktriangle 00\cdots 1 = 1_\blacktriangle \mathtt{y}_1\mathtt{y}_2\cdots\mathtt{y}_B & \text{if } x < 0 \end{cases} \tag{6.40}$$

where $\oplus$ represents modulo-2 addition and bit $\mathtt{y}$ is, in general, different from bit $\bar{\mathtt{x}}$. For example, when $B = 2$, $\hat{x} = +1/4$ is represented by $\hat{x} = 0_\blacktriangle 01$, and $\hat{x} = -1/4$ is represented by $\hat{x} = 1_\blacktriangle 10 \oplus 0_\blacktriangle 01 = 1_\blacktriangle 11$.

□ **EXAMPLE 6.16** Let $B = 3$; then $\hat{x}$ is a 4-bit number (sign plus 3 bits). Provide all possible values that $\hat{x}$ can take in each of the three formats.

Solution There are $2^4 = 16$ possible values that $\hat{x}$ can take for each of the three formats, as shown in the following table.

Binary	*Sign-Magnitude*	*one's*	*two's*
$0_\blacktriangle 111$	7/8	7/8	7/8
$0_\blacktriangle 110$	6/8	6/8	6/8
$0_\blacktriangle 101$	5/8	5/8	5/8
$0_\blacktriangle 100$	4/8	4/8	4/8
$0_\blacktriangle 011$	3/8	3/8	3/8
$0_\blacktriangle 010$	2/8	2/8	2/8
$0_\blacktriangle 001$	1/8	1/8	1/8
$0_\blacktriangle 000$	0	0	0
$1_\blacktriangle 000$	−0	−7/8	−1
$1_\blacktriangle 001$	−1/8	−6/8	−7/8
$1_\blacktriangle 010$	−2/8	−5/8	−6/8
$1_\blacktriangle 011$	−3/8	−4/8	−5/8
$1_\blacktriangle 100$	−4/8	−3/8	−4/8
$1_\blacktriangle 101$	−5/8	−2/8	−3/8
$1_\blacktriangle 110$	−6/8	−1/8	−2/8
$1_\blacktriangle 111$	−7/8	−0	−1/8

□

In the Example, observe that the bit arrangement is exactly the same as in the integer case for 4 bits. The only difference is in the position of the binary point. Thus the MATLAB programs developed in the previous section can easily be used with proper modifications. The MATLAB function `sm2oc` converts a decimal sign-magnitude fraction into its one's-complement format, while the function `oc2sm` performs the inverse operation. These functions are explored in Problem P6.24. Similarly, MATLAB functions `sm2tc` and `tc2sm` convert a decimal sign-magnitude fraction into its two's-complement format and vice versa, respectively; they are explored in Problem P6.25.

6.6.3 FLOATING-POINT ARITHMETIC

In many applications, the range of numbers needed is very large. For example, in physics one might need, at the same time, the mass of the sun (e.g., 2.10^{30}kg) and the mass of the electron (e.g., 9.10^{-31}kg). These two numbers cover a range of over 10^{60}. For fixed-point arithmetic, we would need 62-digit numbers (or 62-digit precision). However, even the mass of the sun is not accurately known with a precision of 5 digits, and there is almost no measurement in physics that could be made with a precision of 62 digits. One could then imagine making all calculations with a precision of 62 digits and throwing away 50 or 60 of them before printing out the final results. This would be wasteful of both CPU time and memory space. So what is needed is a system for representing numbers in which the range of expressible numbers is independent of the number of significant digits.

Decimal numbers The floating-point representation for a decimal number x is based on expressing the number in the scientific notation:

$$x = \pm M \times 10^{\pm E}$$

where M is called the *mantissa* and E is the *exponent.* However, there are different possible representations of the same number, depending on the actual position of the decimal point—for example,

$$1234 = 0.1234 \times 10^4 = 1.234 \times 10^3 = 12.34 \times 10^2 = \cdots$$

To fix this problem, a floating-point number is always stored using a unique representation, which has only one nonzero digit to the left of the decimal point. This representation of a floating-point number is called a *normalized form.* The normalized form of the preceding number is 1.234×10^3, because it is the only representation resulting in a unique nonzero digit to the left of the decimal point. The digit arrangement for the normalized form is given by

$$\hat{x} = \overset{\text{sign of } M}{\overset{\downarrow}{\pm}} \quad \mathtt{x}_{\blacktriangle}\underbrace{\mathtt{xx\cdots x}}_{N\text{-bit } M} \quad \overset{\text{sign of } E}{\overset{\downarrow}{\pm}} \quad \underbrace{\mathtt{xx\cdots x}}_{L\text{-bit } E} \tag{6.41}$$

For the negative numbers we have the same formats as the fixed-point representations, including the 10s-complement format.

The number of digits used in the exponent determine the range of representable numbers, whereas the number of digits used in the mantissa determine the precision of the numbers. For example, if the mantissa is expressed using 2 digits plus the sign, and the exponent is expressed using 2 digits plus the sign, then the real number line will be covered as:

```
                    99            -99                    -99         99
        -9.99x10      -1.0x10       0         1.0x10      9.99x10
---------+-------------+-----------+-----------+-------------+----------->
         |  accessible |           0           |  accessible |
negative |   negative  | negative  | positive  |   positive  | positive
overflow |    numbers  | underflow | underflow |    numbers  | overflow
```

The range of accessible floating-point numbers with a given representation can be large, but it is still finite. In the preceding example (e.g., with 2 digits for the mantissa and 2 digits for the exponent), there are only $9 \times 10 \times 10 \times 199 = 179,100$ positive numbers, and as many negative numbers, plus the number zero, for a total of 358,201 numbers that can be represented.

Binary numbers Although the fraction-only fixed-point arithmetic does not have any overflow problems when two numbers are multiplied, it does suffer from overflow problems when two numbers are added. Also, the fixed-point numbers have limited dynamic range. Both of these aspects are unacceptable for an intensive computational job. These limitations can be removed by making the binary point $\blacktriangle$ floating rather than fixed.

The floating-point bit arrangement for binary-number representation is similar to that for the decimal numbers. In practice, however, two exceptions are made. The exponent is expressed using L-bit excess-2^{L-1} format, and the B-bit normalized mantissa is a fractional number with a 1 following the binary point. Note that the sign bit is the MSB of the bit pattern. Thus the B-bit mantissa and L-bit exponent (for a total of $B + L + 1$ word length) bit pattern is given by (note the reversal of the mantissa exponent places)

$$\hat{x} = \overset{\substack{\text{Sign of M} \\ \downarrow}}{\pm} \quad \underbrace{\mathtt{xx\cdots x}}_{L\text{-bit } E} \blacktriangle \underbrace{\mathtt{1x\cdots x}}_{B\text{-bit } M} \tag{6.42}$$

where exponent E is adjusted so that we have a normalized mantissa—that is, $1/2 \leq M < 1$. Hence the first bit after the binary point is always 1. The decimal equivalent of $\hat{x}$ is given by

$$\hat{x} = \pm M \times 2^E \tag{6.43}$$

For the negative numbers we can have the same formats as the fixed-point representations for the mantissa including two's-complement format.

However, the most widely used format for the mantissa is the sign-magnitude one.

□ **EXAMPLE 6.17** Consider a 32-bit floating-point word with the following arrangement:

$$\hat{x} = \pm \underbrace{\texttt{xx}\cdots\texttt{x}}_{\text{8-bit } E} \blacktriangle \underbrace{\texttt{1x}\cdots\texttt{x}}_{\text{23-bit } M}$$

Determine the decimal equivalent of

`01000001111000000000000000000000`

Solution Since the exponent is 8-bit, it is expressed in excess-2^7 or in excess-128 format. Then the bit pattern can be partitioned into

$$\hat{x} = \overset{\substack{\text{Sign}\\ \downarrow}}{\texttt{0}} \; \underbrace{\texttt{10000011}}_{E=131} \blacktriangle \underbrace{\texttt{11000000000000000000000}}_{M=2^{-1}+2^{-2}}$$

The sign bit is 0, which means that the number is positive. The exponent code is 131, which means that its decimal value is $131 - 128 = 3$. Thus, the bit pattern represents the decimal number $\hat{x} = +\left(2^{-1} + 2^{-2}\right)(2^3) = 2^2 + 2^1 = 6$. □

□ **EXAMPLE 6.18** Let $\hat{x} = -0.1875$. Represent $\hat{x}$ using the format given in (6.42), in which $B = 11$, $L = 4$ (for a total of 16 bits), and sign-magnitude format is used for the mantissa.

Solution We can write

$$\hat{x} = -0.1875 = -0.75 \times 2^{-2}$$

Hence the exponent is -2, the mantissa is 0.75, and the sign is negative. The 4-bit exponent, in excess-8 format, is expressed as $8 - 2 = 6$ or with bit pattern `0110`. The mantissa is expressed as `11000000000`. Since $\hat{x}$ is negative, the bit pattern is

$$\hat{x} \equiv \texttt{1011011000000000}$$

□

The advantages of the floating-point representation are that it has a large dynamic range and that its resolution, defined as the interval between two consecutive representable levels, is proportional to the magnitude. The disadvantages include no representation for the number 0 and the fact that the arithmetic operations are more complicated than their fixed-point representations.

IEEE 754 standard In the early days of the digital computer revolution, each processor design had its own internal representation for floating-point numbers. Since floating-point arithmetic is more complicated to implement, some of these designs did incorrect arithmetic. Therefore, in 1985 IEEE issued a standard (IEEE standard 754-1985 or IEEE-754 for short) to allow floating-point data exchange among different computers and to provide hardware designers with a model known to be correct. Currently, almost all manufacturers design main processors or a dedicated coprocessor for floating-point operations using the IEEE-754 standard representation.

The IEEE 754 standard defines three formats for binary numbers: a 32-bit single precision format, a 64-bit double precision format, and an 80-bit temporary format (which is used internally by the processors or arithmetic coprocessors to minimize rounding errors).

We will briefly describe the 32-bit single precision standard. This standard has many similarities with the floating-point representation discussed above, but there are also differences. Remember, this is another model advocated by IEEE. The form of this model is

$$\hat{x} = \overset{\text{sign of M}}{\overset{\downarrow}{\pm}} \quad \underbrace{\mathtt{xx\cdots x}}_{8-\text{bit E}} \blacktriangle \underbrace{\mathtt{xx\cdots x}}_{23-\text{bit M}} \tag{6.44}$$

The mantissa's value is called the *significand* in this standard. Features of this model are as follows:

- If the sign bit is 0, the number is positive; if the sign bit is 1, the number is negative.
- The exponent is coded in 8-bit excess-127 (and not 128) format. Hence the uncoded exponents are between -127 and 128.
- The mantissa is in 23-bit binary. A normalized mantissa always starts with a bit 1, followed by the binary point, followed by the rest of the 23-bit mantissa. However, the leading bit 1, which is always present in a normalized mantissa, is hidden (not stored) and needs to be restored for computation. Again, note that this is different from the usual definition of the normalized mantissa. If all the 23 bits representing the mantissa are set to 0, the significand is 1 (remember the implicit leading 1). If all 23 bits are set to 1, the significand is almost 2 (in fact $2-2^{-23}$). All IEEE 754 normalized numbers have a significand that is in the interval $1 \leq M < 2$.
- The smallest normalized number is 2^{-126}, and the greatest normalized number is almost 2^{128}. The resulting positive decimal range is roughly 10^{-38} to 10^{38} with a similar negative range.
- If $E = 0$ and $M = 0$, then the representation is interpreted as a *denormalized* number (i.e., the hidden bit is 0) and is assigned a value of

± 0, depending on the sign bit (called the *soft zero*). Thus 0 has two representations.

- If $E = 255$ and $M \neq 0$, then the representation is interpreted as a *not-a-number* (abbreviated as NaN). MATLAB assigns a variable `NaN` when this happens—e.g., 0/0.
- If $E = 255$ and $M = 0$, then the representation is interpreted as $\pm\infty$. MATLAB assigns a variable `inf` when this happens—e.g., 1/0.

□ **EXAMPLE 6.19** Consider the bit pattern given in Example 6.17. Assuming IEEE-754 format, determine its decimal equivalent.

Solution The sign bit is 0 and the exponent code is 131, which means that the exponent is $131 - 127 = 4$. The significand is $1 + 2^{-1} + 2^{-2} = 1.75$. Hence the bit pattern represents

$$\hat{x} = +(1 + 2^{-1} + 2^{-2})(2^4) = 2^4 + 2^3 + 2^2 = 28$$

which is different from the number in Example 6.17. □

MATLAB employs the 64-bit double-precision IEEE-754 format for all its number representations and the 80-bit temporary format for its internal computations. Hence all calculations that we perform in MATLAB are in fact floating-point computations. Simulating a different floating-point format in MATLAB would be much more complicated and would not add any more insight to our understanding than the native format. Hence we will not consider a MATLAB simulation of floating-point arithmetic as we did for fixed-point.

6.7 THE PROCESS OF QUANTIZATION AND ERROR CHARACTERIZATIONS

From the discussion of number representations in the previous section, it should be clear that a general infinite-precision real number must be assigned to one of the finite representable number, given a specific structure for the finite-length register (that is, the arithmetic as well as the format). Usually in practice, there are two different operations by which this assignment is made to the nearest number or level: the *truncation* operation and the *rounding* operation. These operations affect the accuracy as well as general characteristics of digital filters and DSP operations.

We assume, without loss of generality, that there are $B + 1$ bits in the fixed-point (fractional) arithmetic or in the mantissa of floating-point arithmetic including the sign bit. Then the resolution (Δ) is given by

$$\Delta = 2^{-B} \begin{cases} \text{absolute in the case of fixed-point arithmetic} \\ \text{relative in the case of floating-point arithmetic} \end{cases} \tag{6.45}$$

6.7.1 FIXED-POINT ARITHMETIC

The quantizer block diagram in this case is given by

$$\begin{tikzcd} \underset{\text{Infinite}-\text{precision}}{x} \arrow[r] & \boxed{\underset{B,\ \Delta}{\text{Quantizer}\ \ \mathcal{Q}[\cdot]}} \arrow[r] & \underset{\text{Finite}-\text{precision}}{\mathcal{Q}[x]} \end{tikzcd}$$

where B, the number of fractional bits, and Δ, the resolution, are the parameters of the quantizer. We will denote the finite word-length number, after quantization, by $\mathcal{Q}[x]$ for an input number x. Let the quantization error be given by

$$e \overset{\triangle}{=} \mathcal{Q}[x] - x \tag{6.46}$$

We will analyze this error for both the truncation and the rounding operations.

Truncation operation In this operation, the number x is truncated beyond B significant bits (that is, the rest of the bits are eliminated) to obtain $\mathcal{Q}_{\text{T}}[x]$. In MATLAB, to obtain a B-bit truncation, we have to first scale the number x upward by 2^B, then use the `fix` function on the scaled number, and finally scale the result down by 2^{-B}. Thus, the MATLAB statement `xhat = fix(x*2^B)/2^B;` implements the desired operation. We will now consider each of the 3 formats.

Sign-magnitude format If the number x is positive, then after truncation $\mathcal{Q}_{\text{T}}[x] \le x$ since some value in x is lost. Hence quantizer error for truncation denoted by e_{T} is less than or equal to 0 or $e_{\text{T}} \le 0$. However, since there are B bits in the quantizer, the maximum error in terms of magnitude is

$$|e_{\text{T}}| = 0{\blacktriangle}\,\underbrace{00\cdots0}_{B\,\text{bits}}\,111\cdots = 2^{-B}\ \text{(decimal)} \tag{6.47}$$

or

$$-2^{-B} \le e_{\text{T}} \le 0, \quad \text{for } x \ge 0 \tag{6.48}$$

Similarly, if the $x < 0$ then after truncation $\mathcal{Q}_{\text{T}}[x] \ge x$ since $\mathcal{Q}_{\text{T}}[x]$ is less negative, or $e_{\text{T}} \ge 0$. The largest magnitude of this error is again 2^{-B} or

$$0 \le e_{\text{T}} \le 2^{-B}, \quad \text{for } x < 0 \tag{6.49}$$

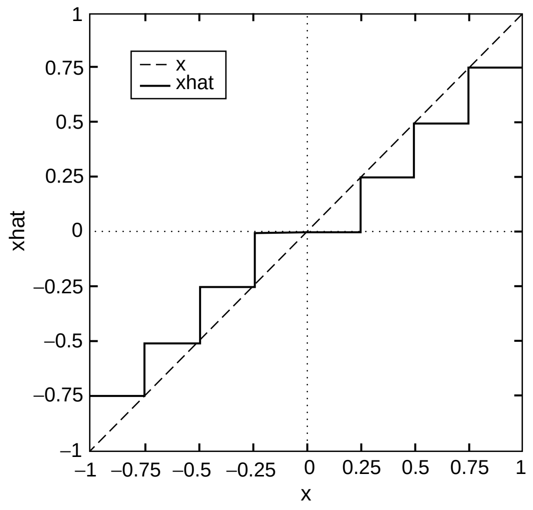


FIGURE 6.25 *Truncation error characteristics in the sign-magnitude format*

□ **EXAMPLE 6.20** Let $-1 < x < 1$ and $B = 2$. Using MATLAB, verify the truncation error characteristics.

Solution The resolution is $\Delta = 2^{-2} = 0.25$. Using the following MATLAB script, we can verify the truncation error e_{T} relations given in (6.48) and (6.49).

```
x = [-1+2^(-10):2^(-10):1-2^(-10)];     % Sign-Mag numbers between -1 and 1
B = 2;                                  % Number of bits for Truncation
xhat = fix(x*2^B)/2^B                   % Truncation
plot(x,x,'g',x,xhat,'r','linewidth',1); % Plot
```

The resulting plots of x and $\hat{x}$ are shown in Figure 6.25. Note that the plot of $\hat{x}$ has a staircase shape and that it satisfies (6.48) and (6.49). □

One's-complement format For $x \geq 0$, we have the same characteristics for e_{T} as in sign-magnitude format—that is,

$$-2^{-B} \leq e_{\text{T}} \leq 0, \quad \text{for } x \geq 0 \tag{6.50}$$

For $x < 0$, the representation is obtained by complementing all bits including sign bit. To compute maximum error, let

$$x = 1_\blacktriangle b_1 b_2 \cdots b_B 000 \cdots = -\left\{{}_\blacktriangle\, (1-b_1)\,(1-b_2)\cdots(1-b_B)\,111\cdots\right\}$$

After truncation, we obtain

$$\mathcal{Q}_{\text{T}}[x] = 1_\blacktriangle b_1 b_2 \cdots b_B = -\left\{{}_\blacktriangle\, (1-b_1)\,(1-b_2)\cdots(1-b_B)\right\}$$

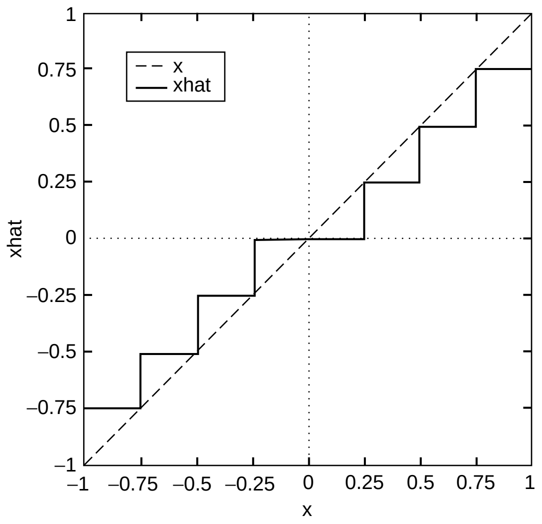


FIGURE 6.26 *Truncation error characteristics in the one's-complement format*

Clearly, x is more negative than $\mathcal{Q}_\text{T}[x]$ or $x \le \mathcal{Q}_\text{T}[x]$ or $e_\text{T} \ge 0$. In fact, the maximum truncation error is

$$e_{\text{Tmax}} = 0_{\blacktriangle}00\cdots0111\cdots = 2^{-B} \text{ (decimal)}$$

Hence

$$0 \le e_\text{T} \le 2^{-B}, \quad \text{for } x < 0 \tag{6.51}$$

□ **EXAMPLE 6.21** Again let $-1 < x < 1$ and $B = 2$ with the resolution $\Delta = 2^{-2} = 0.25$. Using MATLAB script, verify the truncation error e_T relations given in (6.50) and (6.51).

Solution The MATLAB script uses functions `sm2oc` and `oc2sm`, which are explored in Problem P6.25.

```
x = [-1+2^(-10):2^(-10):1-2^(-10)];     % Sign-Magnitude numbers between -1 and 1
B = 2;                                  % Select bits for Truncation
y = sm2oc(x,B);                         % Sign-Mag to One's Complement
yhat = fix(y*2^B)/2^B;                  % Truncation
xhat = oc2sm(yhat,B);                   % Ones'-Complement to Sign-Mag
plot(x,x,'g',x,xhat,'r','linewidth',1); % Plot
```

The resulting plots of x and $\hat{x}$ are shown in Figure 6.26. Note that the plot of $\hat{x}$ is identical to the plot in Figure 6.25 and that it satisfies (6.50) and (6.51). □

Two's-complement format Once again, for $x \geq 0$, we have

$$-2^{-B} \leq e_{\mathrm{T}} \leq 0, \quad \text{for } x \geq 0 \tag{6.52}$$

For $x < 0$, the representation is given by $2 - |x|$ where $|x|$ is the magnitude. Hence the magnitude of x is given by

$$|x| = 2 - x \tag{6.53}$$

with $x = 1_\blacktriangle b_1 b_2 \cdots b_B b_{B+1} \cdots$. After truncation to B bits, we obtain $\mathcal{Q}_{\mathrm{T}}[x] = 1_\blacktriangle b_1 b_2 \cdots b_B$ the magnitude of which is

$$|\mathcal{Q}_{\mathrm{T}}[x]| = 2 - \mathcal{Q}_{\mathrm{T}}[x] \tag{6.54}$$

From (6.53) and (6.54)

$$\begin{aligned}|\mathcal{Q}_{\mathrm{T}}[x]| - |x| &= x - \mathcal{Q}_{\mathrm{T}}[x] = 1_\blacktriangle b_1 b_2 \cdots b_B b_{B+1} \cdots - 1_\blacktriangle b_1 b_2 \cdots b_B \\ &= 0_\blacktriangle 00 \cdots 0 b_{B+1} \cdots \end{aligned} \tag{6.55}$$

The largest change in magnitude from (6.55) is

$$0_\blacktriangle 00 \cdots 0111 \cdots = 2^{-B} \text{ (decimal)} \tag{6.56}$$

Since the change in the magnitude is positive, then after truncation $\mathcal{Q}_{\mathrm{T}}[x]$ becomes more negative, which means that $\mathcal{Q}_{\mathrm{T}}[x] \leq x$. Hence

$$-2^{-B} \leq e_{\mathrm{T}} \leq 0, \quad \text{for } x < 0 \tag{6.57}$$

□ **EXAMPLE 6.22** Again consider $-1 < x < 1$ and $B = 2$ with the resolution $\Delta = 2^{-2} = 0.25$. Using MATLAB, verify the truncation error e_{T} relations given in (6.52) and (6.57).

Solution The MATLAB script uses functions `sm2tc` and `tc2sm`, which are explored in Problem P9.4.

```
x = [-1+2^(-10):2^(-10):1-2^(-10)]; % Sign-Magnitude numbers between -1 and 1
B = 2;                              % Select bits for Truncation
y = sm2tc(x);                       % Sign-Mag to Two's Complement
yhat = fix(y*2^B)/2^B;              % Truncation
xq = tc2sm(yq );                    % Two's-Complement to Sign-Mag
plot(x,x,'g',x,xhat,'r','linewidth',1); % Plot
```

The resulting plots of x and $\hat{x}$ are shown in Figure 6.27. Note that the plot of $\hat{x}$ is also a staircase graph but is below the x graph and that it satisfies (6.52) and (6.57). □

Collecting results (6.48)–(6.52), and (6.57) along with in Figures 6.25–6.27, we conclude that the truncation characteristics for fixed-point arithmetic are the same for the sign-magnitude and the one's-complement formats but are different for the two's-complement format.

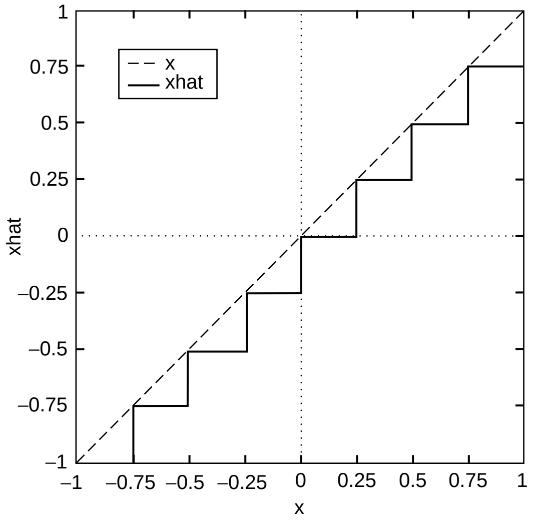


FIGURE 6.27 *Truncation error characteristics in the two's-complement format*

Rounding operation In this operation, the real number x is rounded to the *nearest* representable level, which we will refer to as $\mathcal{Q}_{\mathrm{R}}[x]$. In MATLAB, to obtain a B-bit rounding approximation, we have to first scale the number x up by 2^B, then use the `round` function on the scaled number, and finally scale the result down by 2^{-B}. Thus the MATLAB statement `xhat = round(x*2^B)/2^B;` implements the desired operation.

Since the quantization step or resolution is $\Delta = 2^{-B}$, the magnitude of the maximum error is

$$|e_{\mathrm{R}}|_{\max} = \frac{\Delta}{2} = \frac{1}{2}2^{-B} \tag{6.58}$$

Hence for all three formats, the quantizer error due to rounding, denoted by e_{R}, satisfies

$$-\frac{1}{2}2^{-B} \le e_{\mathrm{R}} \le \frac{1}{2}2^{-B} \tag{6.59}$$

□ **EXAMPLE 6.23** Demonstrate the rounding operations and the corresponding error characteristics on the signal of Examples 6.20–6.22 using the three formats.

Solution Since the rounding operation assigns values that can be larger than the unquantized values, which can create problems for the two's- and one's-complement format, we will restrict the signal over the interval $[-1, 1-2^{-B-1}]$. The following MATLAB script shows the two's-complement format rounding, but other scripts are similar (readers are encouraged to verify).

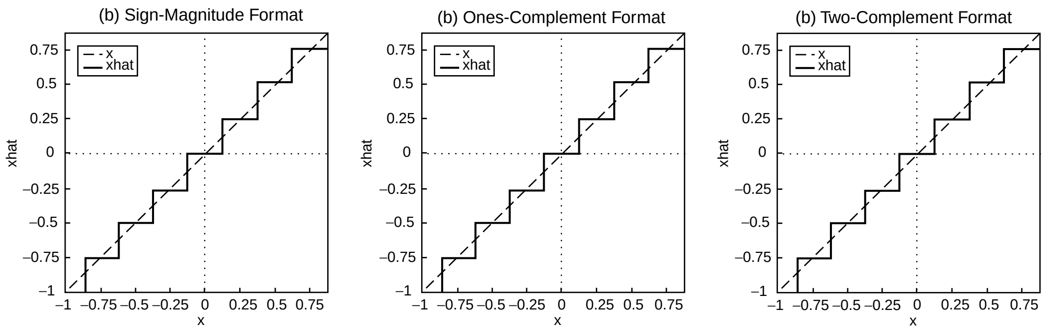


FIGURE 6.28 *Rounding error characteristics in the fixed-point representation*

```
B = 2;                            % Select bits for Rounding
x = [-1:2^(-10):1-2^(-B-1)];      % Sign-Magnitude numbers between -1 and 1
y = sm2tc(x);                     % Sign-Mag to Two's Complement
yq = round(y*2^B)/2^B;            % Rounding
xq = tc2sm(yq);                   % Two'-Complement to Sign-Mag
```

The resulting plots for the sign-magnitude, ones-, and two's-complement formats are shown in Figure 6.28. These plots do satisfy (6.59). □

Comparing the error characteristics of the truncation and rounding operations given in Figures 6.25 through 6.28, it is clear that the rounding operation is a superior one for the quantization error. This is because the error is symmetric with respect to zero (or equal positive and negative distribution) and because the error is the same across all three formats. Hence we will mostly consider the rounding operation for the floating-point arithmetic as well as for further analysis.

6.7.2 FLOATING-POINT ARITHMETIC

In this arithmetic, the quantizer affects only the mantissa M. However, the number x is represented by $M \times 2^E$ where E is the exponent. Hence the quantizer errors are multiplicative and depend on the magnitude of x. Therefore, the more appropriate measure of error is the relative error rather than the absolute error, $(\mathcal{Q}[x] - x)$. Let us define the relative error, ε, as

$$\varepsilon \triangleq \frac{\mathcal{Q}[x] - x}{x} \tag{6.60}$$

Then the quantized value $\mathcal{Q}[x]$ can be written as

$$\mathcal{Q}[x] = x + \varepsilon x = x\,(1+\varepsilon) \tag{6.61}$$

When $\mathcal{Q}[x]$ is due to the rounding operation, then the *error in the mantissa* is between $[-\frac{1}{2}2^{-B}, \frac{1}{2}2^{-B}]$. In this case we will denote the relative error by ε_{R}. Then from (6.43), the absolute error, $\mathcal{Q}_{\mathrm{R}}[x] - x = \varepsilon_{\mathrm{R}}x$, is between

$$\left(-\frac{1}{2}2^{-B}\right)2^{E} \le \varepsilon_{\mathrm{R}}x \le \left(\frac{1}{2}2^{-B}\right)2^{E} \tag{6.62}$$

Now for a given E, and since the mantissa is between $\frac{1}{2} \le M < 1$ (this is not the IEEE-754 model), the number x is between

$$2^{E-1} \le x < 2^{E} \tag{6.63}$$

Hence from (6.62) and using the smallest value in (6.63), we obtain

$$-2^{-B} \le \varepsilon_{\mathrm{R}} \le 2^{-B} \tag{6.64}$$

This relative error relation, (6.64), will be used in subsequent analysis.

6.8 QUANTIZATION OF FILTER COEFFICIENTS

We now study the finite word-length effects on the filter responses, pole-zero locations, and stability when the filter coefficients are quantized. We will separately discuss the issues relating to IIR and FIR filters since we can obtain simpler results for FIR filters. We begin with the case of IIR filters.

6.8.1 IIR FILTERS

Consider a general IIR filter described by

$$H(z) = \frac{\sum_{k=0}^{M} b_k z^{-k}}{1 + \sum_{k=1}^{N} a_k z^{-k}} \tag{6.65}$$

where a_ks and b_ks are the filter coefficients. Now assume that these coefficients are represented by their finite precision numbers $\hat{a}_k$s and $\hat{b}_k$s. Then we get a new filter system function

$$H(z) \to \hat{H}(z) \triangleq \frac{\sum_{k=0}^{M} \hat{b}_k z^{-k}}{1 + \sum_{k=1}^{N} \hat{a}_k z^{-k}} \tag{6.66}$$

Since this is a new filter, we want to know how "different" this filter is from the original one $H(z)$. Various aspects can be compared; for example, we may want to compare their magnitude responses, or phase responses, or change in their pole-zero locations, and so on. A general analytical expression to compute this change in all these aspects is difficult to derive. This is where MATLAB can be used to investigate this change and its overall effect on the usability of the filter.

6.8.2 EFFECT ON POLE-ZERO LOCATIONS

One aspect can be reasonably analyzed, which is the movement of filter poles when a_k is changed to $\hat{a}_k$. This can be used to check the stability of IIR filters. A similar movement of zeros to changes in numerator coefficients can also be analyzed.

To evaluate this movement, consider the denominator polynomial of $H(z)$ in (6.65)

$$D(z) \overset{\triangle}{=} 1 + \sum_{k=1}^{N} a_k z^{-k} = \prod_{\ell=1}^{N} \left(1 - p_\ell \, z^{-1}\right) \tag{6.67}$$

where $\{p_\ell\}$s are the poles of $H(z)$. We will regard $D(z)$ as a function $D(p_1, \ldots, p_N)$ of poles $\{p_1, \ldots, p_N\}$ where each pole p_ℓ is a function of the filter coefficients $\{a_1, \ldots, a_N\}$—that is, $p_\ell = f(a_1, \ldots, a_N)$, $\ell = 1, \ldots N$. Then the change in the denominator $D(z)$ due to a change in the kth coefficient a_k is given by

$$\left(\frac{\partial D(z)}{\partial a_k}\right) = \left(\frac{\partial D(z)}{\partial p_1}\right)\left(\frac{\partial p_1}{\partial a_k}\right) + \left(\frac{\partial D(z)}{\partial p_2}\right)\left(\frac{\partial p_2}{\partial a_k}\right) + \cdots + \left(\frac{\partial D(z)}{\partial p_N}\right)\left(\frac{\partial p_N}{\partial a_k}\right) \tag{6.68}$$

where from (6.67)

$$\left(\frac{\partial D(z)}{\partial p_i}\right) = \frac{\partial}{\partial p_i}\left[\prod_{\ell=1}^{N} \left(1 - p_\ell \, z^{-1}\right)\right] = -z^{-1} \prod_{\ell \neq i} \left(1 - p_\ell \, z^{-1}\right) \tag{6.69}$$

From (6.69), note that $\left(\frac{\partial D(z)}{\partial p_i}\right)\Big|_{z=p_\ell} = 0$ for $\ell \neq i$. Hence from (6.68) we obtain

$$\left(\frac{\partial D(z)}{\partial a_k}\right)\Bigg|_{z=p_\ell} = \left(\frac{\partial D(z)}{\partial p_\ell}\right)\Bigg|_{z=p_\ell} \left(\frac{\partial p_\ell}{\partial a_k}\right) \quad \text{or} \quad \left(\frac{\partial p_\ell}{\partial a_k}\right) = \frac{\left(\frac{\partial D(z)}{\partial a_k}\right)\Big|_{z=p_\ell}}{\left(\frac{\partial D(z)}{\partial p_\ell}\right)\Big|_{z=p_\ell}} \tag{6.70}$$

Now

$$\left(\frac{\partial D(z)}{\partial a_k}\right)\Bigg|_{z=p_\ell} = \frac{\partial}{\partial a_k}\left(1 + \sum_{i=1}^{N} a_i z^{-i}\right)\Bigg|_{z=p_\ell} = z^{-k}\big|_{z=p_\ell} = p_\ell^{-k} \tag{6.71}$$

From (6.69), (6.70) and (6.71), we obtain

$$\left(\frac{\partial p_\ell}{\partial a_k}\right) = \frac{p_\ell^{-k}}{-z^{-1} \prod_{i \neq \ell} \left(1 - p_i \, z^{-1}\right)\big|_{z=p_\ell}} = -\frac{p_\ell^{N-k}}{\prod_{i \neq \ell} \left(p_\ell - p_i\right)} \tag{6.72}$$

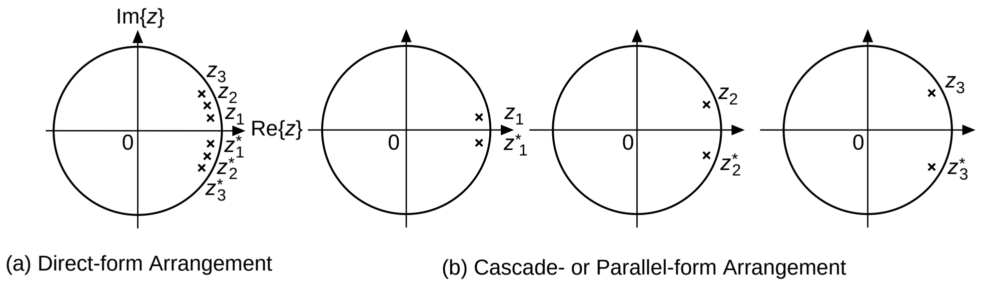

Im{z}
Re{z}
0
z3
z2
z1
z1*
z2*
z3*
(a) Direct-form Arrangement
(b) Cascade- or Parallel-form Arrangement

FIGURE 6.29 *z-plane plots of tightly clustered poles of a digital filter*

Finally, the total perturbation error $\triangle p_\ell$ can be expressed as

$$\triangle p_\ell = \sum_{k=1}^{N} \frac{\partial p_\ell}{\partial a_k} \triangle a_k \tag{6.73}$$

This formula measures the movement of the ℓth pole, p_ℓ, to changes in each of the coefficient $\{a_k\}$; hence it is known as a *sensitivity* formula. It shows that if the coefficients $\{a_k\}$ are such that if the poles p_ℓ and p_i are very close for some ℓ, i, then $(p_\ell - p_i)$ is very small and as a result the filter is very sensitive to the changes in filter coefficients. A similar result can be obtained for the sensitivity of zeros to changes in the parameters $\{b_k\}$.

To investigate this further in the light of various filter realizations, consider the z-plane plot shown in Figure 6.29(a) where poles are tightly clustered. This situation arises in wideband frequency selective filters such as lowpass or highpass filters. Now if we were to realize this filter using the direct form (either I or II), then the filter has all these tightly clustered poles, which makes the direct-form realization very sensitive to coefficient changes due to finite word length. Thus, the direct form realizations will suffer severely from coefficient quantization effects.

On the other hand, if we were to use either the cascade or the parallel forms, then we would realize the filter using 2nd-order sections containing widely separated poles, as shown in Figure 6.29(b). Thus, each 2nd-order section will have low sensitivity in that its pole locations will be perturbed only slightly. Consequently, we expect that the overall system function $H(z)$ will be perturbed only slightly. Thus, the cascade or the parallel forms, when realized properly, will have low sensitivity to the changes or errors in filter coefficients.

□ **EXAMPLE 6.24** Consider a digital resonator that is a 2nd-order IIR filter given by

$$H(z) = \frac{1}{1 - (2r\cos\theta)\, z^{-1} + r^2 z^{-2}} \tag{6.74}$$

Analyze its sensitivity to pole locations when a 3-bit sign-magnitude format is used for the coefficient representation.

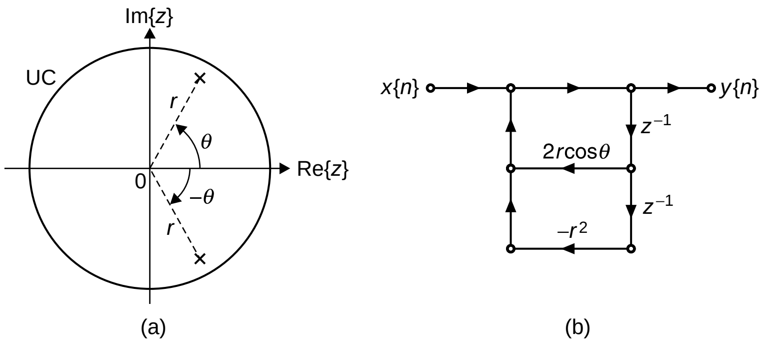


FIGURE 6.30 *Digital filter in Example 6.24 (a) pole-zero plot, (b) filter realization*

Solution The filter has two complex-conjugate poles at

$$p_1 = re^{j\theta} \quad \text{and} \quad p_2 = re^{-j\theta} = p_1^*$$

For a proper operation as a resonator, the poles must be close to the unit circle—that is, $r \simeq 1$ (but $r < 1$). Then the resonant frequency $\omega_{\text{r}} \simeq \theta$. The zero-pole diagram is shown in Figure 6.30 along with the filter realization. Let $r = 0.9$ and $\theta = \pi/3$. Then from (6.74),

$$a_1 = -2r\cos\theta = -0.9 \quad \text{and} \quad a_2 = r^2 = 0.81$$

We now represent a_1 and a_2, each using 3-bit sign-magnitude format representation—that is,

$$a_k = \pm \blacktriangle b_1\, b_2\, b_3 = \pm\left(b_1 2^{-1} + b_2 2^{-2} + b_3 2^{-3}\right), \quad k = 1, 2$$

where b_j represents the jth bit and $\blacktriangle$ represents the binary point. Then for the closest representation, we must have

$$\hat{a}_1 = 1\blacktriangle 1\,1\,1 = -0.875 \qquad \text{and} \qquad \hat{a}_2 = 0\blacktriangle 1\,1\,0 = +0.75$$

Hence $|\triangle a_1| = 0.025$ and $|\triangle a_2| = 0.06$. Consider the sensitivity formula (6.73) in which

$$\frac{\partial p_1}{\partial a_1} = -\frac{p_1^{2-1}}{(p_1 - p_1^*)} = \frac{-p_1}{2\,\text{Im}\{p_1\}} = \frac{-re^{j\theta}}{2r(\sin\theta)} = \frac{e^{j\pi/3}}{\sqrt{3}}, \text{ and}$$
$$\frac{\partial p_1}{\partial a_2} = -\frac{p_1^{2-2}}{(p_1 - p_1^*)} = \frac{-1}{2\,\text{Im}\{p_1\}} = \frac{1}{0.9\sqrt{3}}$$

Using (6.73), we obtain

$$\begin{aligned}|\triangle p_1| &\le \left|\frac{\partial p_1}{\partial a_1}\right| |\triangle a_1| + \left|\frac{\partial p_1}{\partial a_2}\right| |\triangle a_2| \\ &= \frac{1}{\sqrt{3}}(0.025) + \frac{1}{0.9\sqrt{3}}(0.06) = 0.0529 \qquad \textbf{(6.75)}\end{aligned}$$

To determine the exact locations of the changed poles, consider the changed denominator

$$\hat{D}(z) = 1-0.875z^{-1}+0.75z^{-2} = \left(1-0.866e^{j0.331\pi}z^{-1}\right)\left(1-0.866e^{-j0.331\pi}z^{-1}\right)$$

Thus, the changed pole locations are $\hat{p}_1 = 0.866e^{j0.331\pi} = \hat{p}_2^*$. Then $|\triangle p_1| = \left|0.9e^{i\pi/3} - 0.866e^{i0.331\pi}\right| = 0.0344$, which agrees with (6.75). □

Analysis using MATLAB To investigate the effect of coefficient quantization on filter behavior, MATLAB is an ideal vehicle. Using functions developed in previous sections, we can obtain quantized coefficients and then study such aspects as pole-zero movements, frequency response, or impulse response. We will have to represent all filter coefficients using the same number of integer and fraction bits. Hence instead of quantizing each coefficient separately, we will develop the function, `QCoeff`, for coefficient quantization. This function implements quantization using rounding operation on sign-magnitude format. Although similar functions can be written for truncation as well as for other formats, we will analyze the effects using the `Qcoeff` function as explained previously.

```
function [y,L,B] = QCoeff(x,N)
%  [y,L,B] = QCoeff(x,N)
%      Coefficient Quantization using N=1+L+B bit Representation
%        with Rounding operation
%      y: quantized array (same dim as x)
%      L: number of integer bits
%      B: number of fractional bits
%      x: a scalar, vector, or matrix
%      N: total number of bits

xm = abs(x);
 L = max(max(0,fix(log2(xm(:)+eps)+1))); % Integer bits
if (L > N)
     errmsg = [' *** N must be at least ',num2str(L),' ***']; error(errmsg);
end
B = N-L;                               % Fractional bits
y = xm./(2^L); y = round(y.*(2^N)); % Rounding to N bits
y = sign(x).*y*(2^(-B));            % L+B+1 bit representation
```

The `Qcoeff` function represents each coefficient in the `x` array using `N+1`-bit (including the sign bit) representation. First, it determines the number of bits `L` needed for integer representation for the magnitude-wise largest coefficient, and then it assigns `N-L` bits to the fraction part. The resulting number is returned in `B`. Thus all coefficients have the same bit pattern `L+B+1`. Clearly, `N` $\geq$ `L`.

□ **EXAMPLE 6.25** Consider the digital resonator in Example 6.24. Determine the change in the pole locations using MATLAB.

Solution The filter coefficients, $a_1 = -0.9$ and $a_2 = 0.81$ can be quantized using

```
>> x = [-0.9,0.81]; [y,L,B] = Qcoeff(x,3)
y = -0.8750    0.7500
L =  0
B =  3
```

as expected. Now using the following MATLAB script, we can determine the change in the location of the poles:

```
% Unquantized parameters
r = 0.9; theta = pi/3; a1 = -2*r*cos(theta); a2 = r*r;
p1 = r*exp(j*theta); p2 = p1';
% Quantized parameters: N = 3;
[ahat,L,B] = Qcoeff([a1,a2],3); rhat = sqrt(ahat(2));
thetahat = acos(-ahat(1)/(2*rhat)); p1hat = rhat*exp(j*thetahat); p2 = p1';
% Changes in pole locations
Dp1 = abs(p1-p1hat)
Dp1 = 0.0344
```

This is the same as before. □

□ **EXAMPLE 6.26** Consider the following IIR filter with 10 poles closely packed at a radius of $r = 0.9$ around angles $\pm 45^\circ$ with a separation of 5°. Due to large number of poles, the denominator coefficients have values that require 6 bits for the integer part. Using 9 bits for the fractional part for a total of 16-bit representation, we compute and plot the new locations of poles:

```
r = 0.9; theta = (pi/180)*[-55:5:-35,35:5:55]';
p = r*exp(j*theta); a = poly(p); b = 1;

% Direct form: quantized coefficients
N = 15; [ahat,L,B] = Qcoeff(a,N);
TITLE = sprintf('%i-bit (1+%i+%i) Precision',N+1,L,B);

% Comparison of Pole-Zero Plots
subplot(1,2,1); [HZ,HP,Hl] = zplane(1,a);
set(HZ,'color','g','linewidth',1); set(HP,'color','g','linewidth',1);
set(Hl,'color','w'); axis([-1.1,1.1,-1.1,1.1]);
title('Infinite Precision','fontsize',10,'fontweight','bold');
```

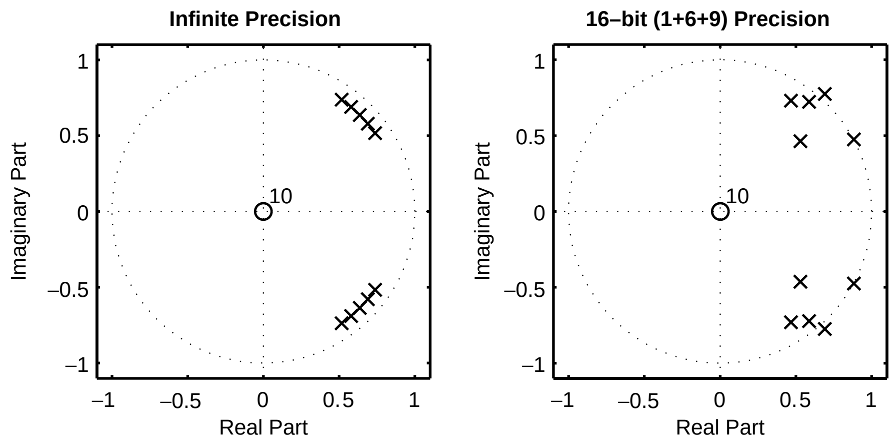


FIGURE 6.31 *Pole-zero plots for direct-form structure in Example 6.26*

```
subplot(1,2,2); [HZhat,HPhat,Hlhat] = zplane(1,ahat);
set(HZhat,'color','r','linewidth',1); set(HPhat,'color','r','linewidth',1);
set(Hlhat,'color','w'); title(TITLE,'fontsize',10,'fontweight','bold');
axis([-1.1,1.1,-1.1,1.1]);
```

Figure 6.31 shows the pole-zero plots for filters with both infinite and 16-bit precision coefficients. Clearly, with 16-bit word length, the resulting filter is completely different from the original one and is unstable. To investigate finite word-length effect on the cascade-form structure, we first converted the direct-form coefficients into the cascade-form coefficients using the `dir2cas` function, quantized the resulting set of coefficients, and then converted back to the direct-form for pole-zero plotting. We show results for two different word lengths. In the first case, we used the same 16-bit word length. Since the cascade coefficients have smaller integer parts that require only one integer bit, the number of fractional bits is 14. In the second case we used 9 fractional bits (same as those in the direct form) for a total word length of 11 bits.

```
% Cascade form: quantized coefficients: Same N
[b0,B0,A0] = dir2cas(b,a);  [BAhat1,L1,B1] = Qcoeff([B0,A0],N);
TITLE1 = sprintf('%i-bit (1+%i+%i) Precision',N+1,L1,B1);
Bhat1 = BAhat1(:,1:3); Ahat1 = BAhat1(:,4:6);
[bhat1,ahat1] = cas2dir(b0,Bhat1,Ahat1);

subplot(1,2,1);  [HZhat1,HPhat1,Hlhat1] = zplane(bhat1,ahat1);
set(HZhat1,'color','g','linewidth',1); set(HPhat1,'color','g','linewidth',1);
set(Hlhat1,'color','w'); axis([-1.1,1.1,-1.1,1.1]);
title(TITLE1,'fontsize',10,'fontweight','bold');
```

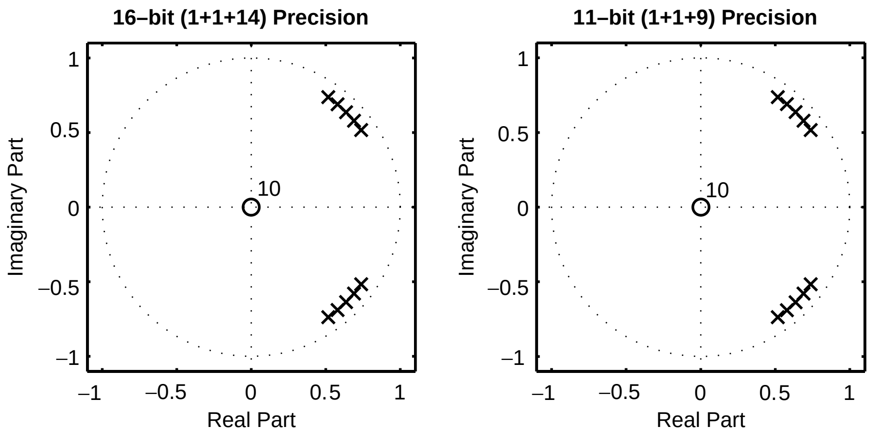


FIGURE 6.32 *Pole-zero plots for cascade-form structure in Example 6.26*

```
% Cascade form: quantized coefficients: Same B (N=L1+B)
N1 = L1+B; [BAhat2,L2,B2] = Qcoeff([B0,A0],N1);
TITLE2 = sprintf('%i-bit (1+%i+%i) Precision',N1+1,L2,B2);
Bhat2 = BAhat2(:,1:3); Ahat2 = BAhat2(:,4:6);
[bhat2,ahat2] = cas2dir(b0,Bhat2,Ahat2);

subplot(1,2,2);  [HZhat2,HPhat2,Hlhat2] = zplane(bhat2,ahat2);
set(HZhat2,'color','r','linewidth',1); set(HPhat2,'color','r','linewidth',1);
set(Hlhat2,'color','w');title(TITLE2,'fontsize',10,'fontweight','bold');
axis([-1.1,1.1,-1.1,1.1]);
```

The results are shown in Figure 6.32. We observe that not only for 16-bit representation but also for 11-bit representation, the resulting filter is essentially the same as the original one and is stable. Clearly, the cascade form structure has better finite word-length properties than the direct form structure. □

6.8.3 EFFECTS ON FREQUENCY RESPONSE

The frequency response of the IIR filter in (6.50) is given by

$$H(e^{\jmath\omega}) = \frac{\sum_{k=0}^{M} b_k\, e^{-\jmath\omega k}}{1 + \sum_{k=1}^{N} a_k\, e^{-\jmath\omega k}} \tag{6.76}$$

When the coefficients $\{a_k\}$ and $\{b_k\}$ are quantized to $\{\hat{a}_k\}$ and $\{\hat{b}_k\}$, respectively, the new frequency response is given by

$$\hat{H}(e^{\jmath\omega}) = \frac{\sum_{k=0}^{M} \hat{b}_k\, e^{-\jmath\omega k}}{1 + \sum_{k=1}^{N} \hat{a}_k\, e^{-\jmath\omega k}} \tag{6.77}$$

One can perform analysis similar to that for the movement of poles to obtain maximum change in the magnitude or phase responses due to changes in filter coefficients. However, such an analysis is very complicated and may not add any new insight. Hence we will study these effects using MATLAB. We provide the following two examples.

□ **EXAMPLE 6.27** Compute and plot magnitude responses of filter structures given for the filter in Example 6.26.

Solution The filter is a bandpass filter with 10 tightly clustered poles implemented using the direct and the cascade forms. For the direct-form structure, we compute the magnitude response for infinite precision as well as for 16-bit quantization. For the cascade-form structure, we use 16-bit and 11-bit representations.

```
r = 0.9; theta = (pi/180)*[-55:5:-35,35:5:55]';
p = r*exp(j*theta); a = poly(p); b = 1;
w = [0:500]*pi/500; H = freqz(b*1e-4,a,w);
magH = abs(H); magHdb = 20*log10(magH);

% Direct form: quantized coefficients
N = 15; [ahat,L,B] = Qcoeff(a,N);
TITLE = sprintf('%i-bit (1+%i+%i) Precision (DF)',N+1,L,B);
Hhat = freqz(b*1e-4,ahat,w); magHhat = abs(Hhat);

% Cascade form: quantized coefficients: Same N
[b0,B0,A0] = dir2cas(b,a);
[BAhat1,L1,B1] = Qcoeff([B0,A0],N);
TITLE1 = sprintf('%i-bit (1+%i+%i) Precision (CF)',N+1,L1,B1);
Bhat1 = BAhat1(:,1:3); Ahat1 = BAhat1(:,4:6);
[bhat1,ahat1] = cas2dir(b0,Bhat1,Ahat1);
Hhat1 = freqz(b*1e-4,ahat1,w); magHhat1 = abs(Hhat1);

% Cascade form: quantized coefficients: Same B (N=L1+B)
N1 = L1+B; [BAhat2,L2,B2] = Qcoeff([B0,A0],N1);
TITLE2 = sprintf('%i-bit (1+%i+%i) Precision (CF)',N1+1,L2,B2);
Bhat2 = BAhat2(:,1:3); Ahat2 = BAhat2(:,4:6);
[bhat2,ahat2] = cas2dir(b0,Bhat2,Ahat2);
Hhat2 = freqz(b*1e-4,ahat2,w); magHhat2 = abs(Hhat2);

% Comparison of Magnitude Plots
Hf_1 = figure('paperunits','inches','paperposition',[0,0,6,4]);
subplot(2,2,1); plot(w/pi,magH,'g','linewidth',2); axis([0,1,0,0.7]);
%xlabel('Digital Frequency in \pi units','fontsize',10);
ylabel('Magnitude Response','fontsize',10);
title('Infinite Precision (DF)','fontsize',10,'fontweight','bold');
subplot(2,2,2); plot(w/pi,magHhat,'r','linewidth',2); axis([0,1,0,0.7]);
```

```
%xlabel('Digital Frequency in \pi units','fontsize',10);
ylabel('Magnitude Response','fontsize',10);
title(TITLE,'fontsize',10,'fontweight','bold');
subplot(2,2,3); plot(w/pi,magHhat1,'r','linewidth',2); axis([0,1,0,0.7]);
xlabel('Digital Frequency in \pi units','fontsize',10);
ylabel('Magnitude Response','fontsize',10);
title(TITLE1,'fontsize',10,'fontweight','bold');
subplot(2,2,4); plot(w/pi,magHhat2,'r','linewidth',2); axis([0,1,0,0.7]);
xlabel('Digital Frequency in \pi units','fontsize',10);
ylabel('Magnitude Response','fontsize',10);
title(TITLE2,'fontsize',10,'fontweight','bold');
```

The plots are shown in Figure 6.33. The top row shows plots for the direct form, and the bottom row shows those for the cascade form. As expected, the magnitude plot of the direct form is severely distorted for 16-bit representation, while those for the cascade form are preserved even for 11-bit word length. □

□ **EXAMPLE 6.28** An 8th-order bandpass filter was obtained using the elliptic filter design approach. This and other design methods will be discussed in Chapter 8. The MATLAB functions needed for this design are shown in the following script. This design produces direct-form filter coefficients b_k and a_k, using 64-bit floating-point arithmetic, which gives the precision of 15 decimals and hence can be considered as *unquantized* coefficients. Table 6.1 shows these filter coefficients.

Represent the unquantized filter coefficients using 16-bit and 8-bit word lengths. Plot the filter log-magnitude responses and pole-zero locations for both the infinite and finite word-length coefficients.

TABLE 6.1 *Unquantized IIR filter coefficients used in Example 6.28*

k	b_k	a_k
0	0.021985541264351	1.000000000000000
1	0.000000000000000	−0.000000000000004
2	−0.032498273955222	2.344233276056572
3	0.000000000000000	−0.000000000000003
4	0.046424673058794	2.689868616770005
5	0.000000000000000	0.000000000000001
6	−0.032498273955221	1.584557559015230
7	0.000000000000000	0.000000000000001
8	0.021985541264351	0.413275250482975

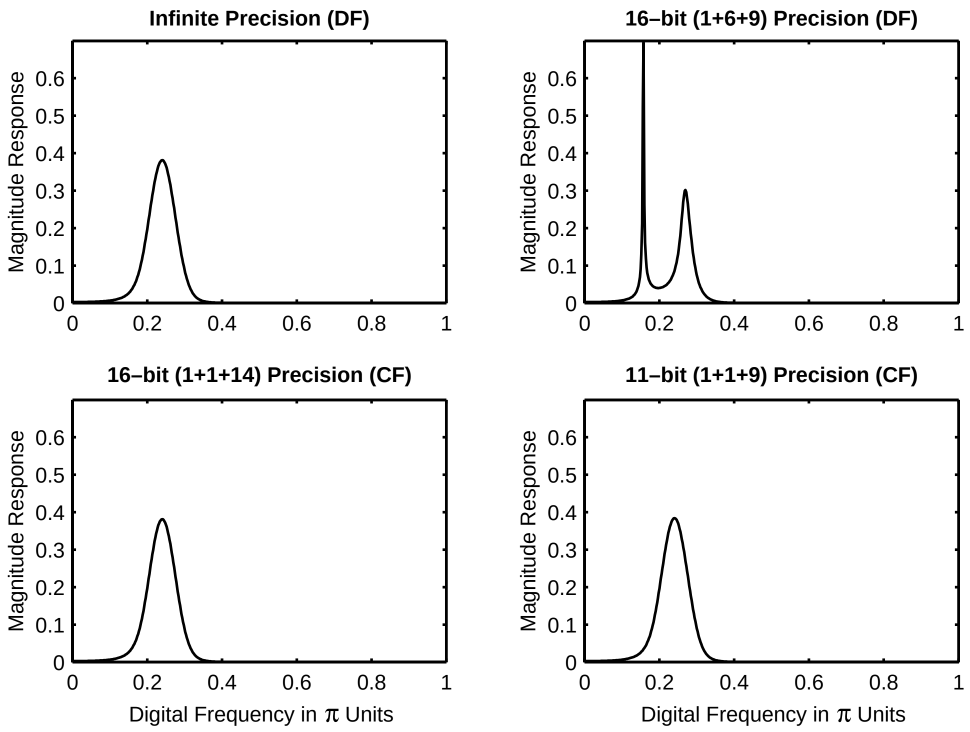


FIGURE 6.33 *Magnitude plots for direct- and cascade-form structures in Example 6.27*

Solution Unlike the previous example, some of the filter coefficient values (specifically those of the autoregressive part) are greater than one and hence require bits for the integer part. This assignment is done for all coefficients since in practice, the same bit-pattern is used for the filter representation. These and other steps are given in the following MATLAB script.

```
% The following 3 lines produce filter coefficients shown in Table 6.1.
wp = [0.35,0.65]; ws = [0.25,0.75]; Rp = 1; As = 50;
[N, wn] = ellipord(wp, ws, Rp, As);
[b,a] = ellip(N,Rp,As,wn);
w = [0:500]*pi/500; H = freqz(b,a,w); magH = abs(H);
magHdb = 20*log10(magH);

% 16-bit word-length quantization
N1 = 15; [bahat,L1,B1] = QCoeff([b;a],N1);
TITLE1 = sprintf('%i-bits (1+%i+%i)',N1+1,L1,B1);
bhat1 = bahat(1,:); ahat1 = bahat(2,:);
Hhat1 = freqz(bhat1,ahat1,w); magHhat1 = abs(Hhat1);
magHhat1db = 20*log10(magHhat1); zhat1 = roots(bhat1);
```

```
% 8-bit word-length quantization
N2 = 7; [bahat,L2,B2] = QCoeff([b;a],N2);
TITLE2 = sprintf('%i-bits (1+%i+%i)',N2+1,L2,B2);
bhat2 = bahat(1,:); ahat2 = bahat(2,:);
Hhat2 = freqz(bhat2,ahat2,w); magHhat2 = abs(Hhat2);
magHhat2db = 20*log10(magHhat2); zhat2 = roots(bhat2);

% Plots
Hf_1 = figure('paperunits','inches','paperposition',[0,0,6,5]);

% Comparison of Log-Magnitude Responses: 16 bits
subplot(2,2,1); plot(w/pi,magHdb,'g','linewidth',1.5); axis([0,1,-80,5]);
hold on; plot(w/pi,magHhat1db,'r','linewidth',1); hold off;
xlabel('Digital Frequency in \pi units','fontsize',10);
ylabel('Decibels','fontsize',10);
title(['Log-Mag plot: ',TITLE1],'fontsize',10,'fontweight','bold');

% Comparison of Pole-Zero Plots: 16 bits
subplot(2,2,3); [HZ,HP,Hl] = zplane([b],[a]); axis([-2,2,-2,2]); hold on;
set(HZ,'color','g','linewidth',1,'markersize',4);
set(HP,'color','g','linewidth',1,'markersize',4);
plot(real(zhat1),imag(zhat1),'r+','linewidth',1);
title(['PZ Plot: ',TITLE1],'fontsize',10,'fontweight','bold');
hold off;

% Comparison of Log-Magnitude Responses: 8 bits
subplot(2,2,2); plot(w/pi,magHdb,'g','linewidth',1.5); axis([0,1,-80,5]);
hold on; plot(w/pi,magHhat2db,'r','linewidth',1); hold off;
xlabel('Digital Frequency in \pi units','fontsize',10);
ylabel('Decibels','fontsize',10);
title(['Log-Mag plot: ',TITLE2],'fontsize',10,'fontweight','bold');

% Comparison of Pole-Zero Plots: 8 bits
subplot(2,2,4); [HZ,HP,Hl] = zplane([b],[a]); axis([-2,2,-2,2]); hold on;
set(HZ,'color','g','linewidth',1,'markersize',4);
set(HP,'color','g','linewidth',1,'markersize',4);
plot(real(zhat2),imag(zhat2),'r+','linewidth',1);
title(['PZ Plot: ',TITLE2],'fontsize',10,'fontweight','bold');
hold off;
```

The log-magnitude responses and zero-pole locations of the resulting filters are plotted in Figure 6.34 along with those of the original filter. When 16 bits are used, the resulting filter is virtually indistinguishable from the original one. However, when 8 bits are used, the filter behavior is severely distorted. The filter is still stable, but it does not satisfy the design specifications. □

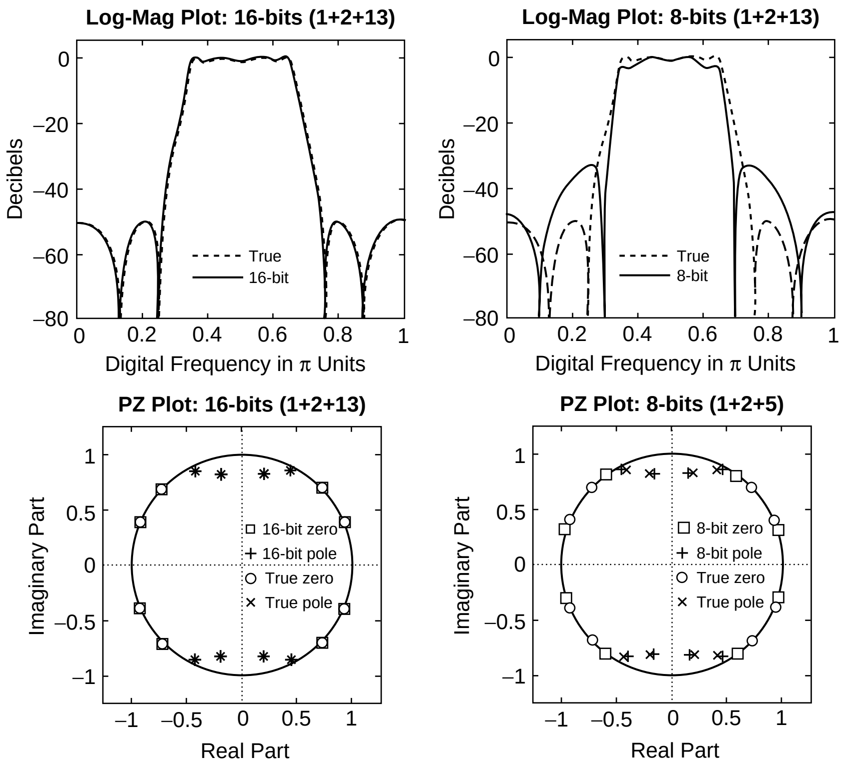


FIGURE 6.34 *Plots for the IIR filter in Example 6.28*

6.8.4 FIR FILTERS

A similar analysis can be done for FIR filters. Let the impulse response of an FIR filter be $h(n)$ with system response

$$H(z) = \sum_{n=0}^{M-1} h(n)z^{-n} \tag{6.78}$$

Then,

$$\Delta H(z) = \sum_{n=0}^{M-1} \Delta h(n)z^{-n} \tag{6.79}$$

where $\Delta H(z)$ is the change due to change in the impulse response $h(n)$. Hence

$$\Delta H\left(e^{\jmath\omega}\right) = \sum_{n=0}^{M-1} \Delta h(n)\, e^{-\jmath\omega n} \quad \text{or} \quad |\Delta H(e^{\jmath\omega})| \le \sum_{n=0}^{M-1} |\Delta h(n)| \tag{6.80}$$

Now, if each coefficient is quantized to B fraction bits (i.e., total register length is $B+1$), then,

$$|\Delta h(n)| \le \frac{1}{2}2^{-B}$$

Therefore,

$$|\Delta H(e^{\jmath\omega})| \le \frac{1}{2}2^{-B}M = \frac{M}{2}2^{-B} \tag{6.81}$$

Thus, the change in frequency response depends not only on the number of bits used but also on the length M of the filter. For large M and small b, this difference can be significant and can destroy the desirable behavior of the filter, as we see in the following example.

□ **EXAMPLE 6.29** An order-30 lowpass FIR filter is designed using the `firpm` function. This and other FIR filter design functions will be discussed in Chapter 7. The resulting filter coefficients are symmetric and are shown in Table 6.2. We will consider these coefficients as essentially unquantized. The coefficients are quantized to 16 bits (15 fractional plus 1 sign bit) and to 8 bits (7 fractional and 1 sign bit). The resulting filter frequency responses and pole-zero plots are determined and compared. These and other relevant steps are shown in the following MATLAB script.

TABLE 6.2 *Unquantized FIR filter coefficients used in Example 6.29*

k	b_k	k
0	0.000199512328641	30
1	−0.002708453461401	29
2	−0.002400461099957	28
3	0.003546543555809	27
4	0.008266607456720	26
5	0.000012109690648	25
6	−0.015608300819736	24
7	−0.012905580320708	23
8	0.017047710292001	22
9	0.036435951059014	21
10	0.000019292305776	20
11	−0.065652005307521	19
12	−0.057621325403582	18
13	0.090301607282890	17
14	0.300096964940136	16
15	0.400022084144842	15

```
% The following function computes the filter
% coefficients given in Table 6.2.
  b = firpm(30,[0,0.3,0.5,1],[1,1,0,0]);
  w = [0:500]*pi/500; H = freqz(b,1,w); magH = abs(H);
```

```
magHdb = 20*log10(magH);
N1 = 15; [bhat1,L1,B1] = Qcoeff(b,N1);
TITLE1 = sprintf('%i-bits (1+%i+%i)',N1+1,L1,B1);
Hhat1 = freqz(bhat1,1,w); magHhat1 = abs(Hhat1);
magHhat1db = 20*log10(magHhat1);
zhat1 = roots(bhat1);

N2 = 7; [bhat2,L2,B2] = Qcoeff(b,N2);
TITLE2 = sprintf('%i-bits (1+%i+%i)',N2+1,L2,B2);
Hhat2 = freqz(bhat2,1,w); magHhat2 = abs(Hhat2);
magHhat2db = 20*log10(magHhat2);
zhat2 = roots(bhat2);

% Plots
Hf_1 = figure('paperunits','inches','paperposition',[0,0,6,5]);

% Comparison of Log-Magnitude Responses: 16 bits
subplot(2,2,1); plot(w/pi,magHdb,'g','linewidth',1.5); axis([0,1,-80,5]);
hold on; plot(w/pi,magHhat1db,'r','linewidth',1); hold off;
xlabel('Digital Frequency in \pi units','fontsize',10);
ylabel('Decibels','fontsize',10);
title(['Log-Mag plot: ',TITLE1],'fontsize',10,'fontweight','bold');

% Comparison of Pole-Zero Plots: 16 bits
subplot(2,2,3); [HZ,HP,Hl] = zplane([b],[1]); axis([-2,2,-2,2]); hold on;
set(HZ,'color','g','linewidth',1,'markersize',4);
set(HP,'color','g','linewidth',1,'markersize',4);
plot(real(zhat1),imag(zhat1),'r+','linewidth',1);
title(['PZ Plot: ',TITLE1],'fontsize',10,'fontweight','bold');
hold off;

% Comparison of Log-Magnitude Responses: 8 bits
subplot(2,2,2); plot(w/pi,magHdb,'g','linewidth',1.5); axis([0,1,-80,5]);
hold on; plot(w/pi,magHhat2db,'r','linewidth',1); hold off;
xlabel('Digital Frequency in \pi units','fontsize',10);
ylabel('Decibels','fontsize',10);
title(['Log-Mag plot: ',TITLE2],'fontsize',10,'fontweight','bold');

% Comparison of Pole-Zero Plots: 8 bits
subplot(2,2,4); [HZ,HP,Hl] = zplane([b],[1]); axis([-2,2,-2,2]); hold on;
set(HZ,'color','g','linewidth',1,'markersize',4);
set(HP,'color','g','linewidth',1,'markersize',4);
plot(real(zhat2),imag(zhat2),'r+','linewidth',1);
title(['PZ Plot: ',TITLE2],'fontsize',10,'fontweight','bold');
hold off;
```

The log-magnitude responses and zero-pole locations of the resulting filters are computed and plotted in Figure 6.35 along with those of the original filter.

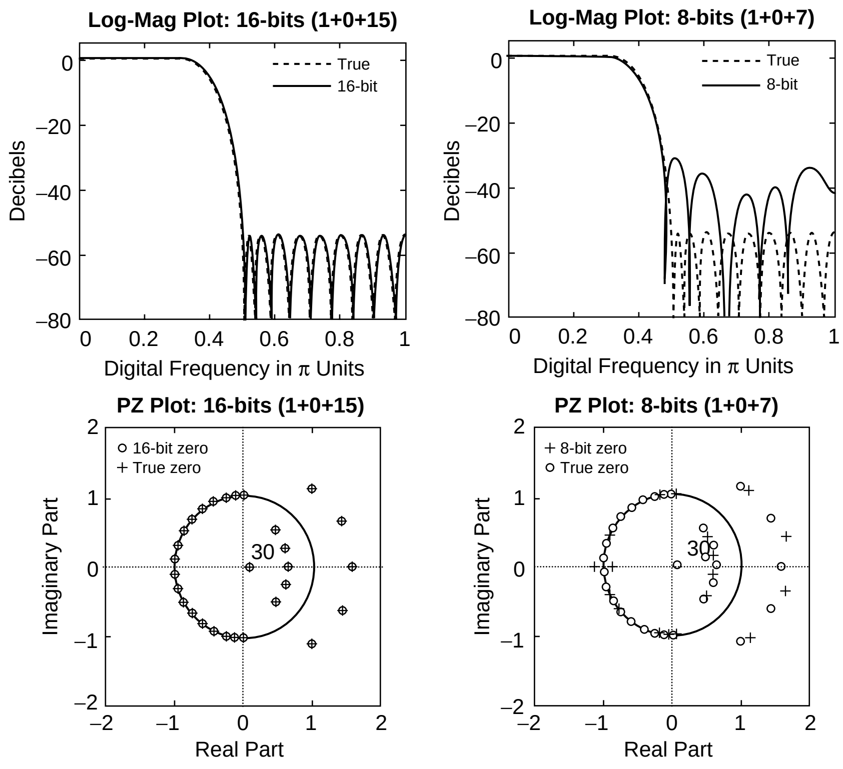


FIGURE 6.35 *Plots for the FIR filter in Example 6.29*

When 16 bits are used, the resulting filter is virtually indistinguishable from the original one. However, when 8 bits are used, the filter behavior is severely distorted and the filter does not satisfy the design specifications. □

6.9 PROBLEMS

P6.1 Draw direct form I block diagram structures for each of the following LTI systems with input node $x(n)$ and output node $y(n)$.

1. $y(n) = x(n) + 2\,x(n-1) + 3x(n-2)$
2. $H(z) = \dfrac{1}{1 - 1.7z^{-1} + 1.53z^{-2} - 0.648z^{-3}}$
3. $y(n) = 1.7\,y(n-1) - 1.36\,y(n-2) + 0.576\,y(n-3) + x(n)$
4. $y(n) = 1.6\,y(n-1) + 0.64\,y(n-2) + x(n) + 2\,x(n-1) + x(n-2)$
5. $H(z) = \dfrac{1 - 3z^{-1} + 3z^{-2} + z^{-3}}{1 + 0.2z^{-1} - 0.14z^{-2} + 0.44z^{-3}}$

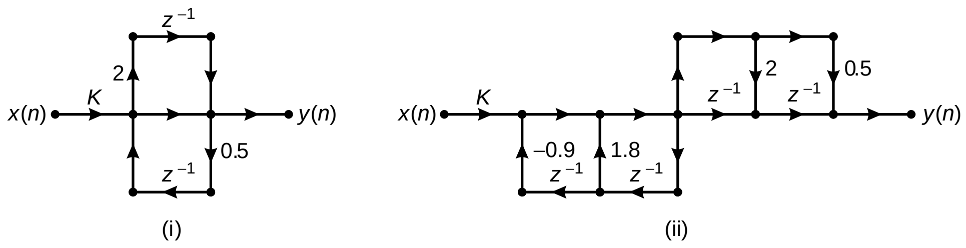


FIGURE P6.1 *Block diagrams for Problem 6.2*

P6.2 Two block diagrams are shown in Figure P6.1. Answer the following for each structure.

1. Determine the system function $H(z) = Y(z)/X(z)$.
2. Is the structure canonical (i.e., with the least number of delays)? If not, draw a canonical structure.
3. Determine the value of K so that $H(e^{j\,0}) = 1$.

P6.3 Consider the LTI system described by

$$y(n) = a\,y(n-1) + b\,x(n) \tag{6.82}$$

1. Draw a block diagram of this system with input node $x(n)$ and output node $y(n)$.
2. Now perform the following two operations on the structure drawn in part 1: (i) reverse all arrow directions and (ii) interchange the input node with the output node. Notice that the branch node becomes the adder node and vice versa. Redraw the block diagram so that input node is on the left side and the output node is on the right side. This is the *transposed* block diagram.
3. Determine the difference equation representation of your transposed structure in part 2, and verify that it is the same equation as (6.82).

P6.4 Consider the LTI system given by

$$H(z) = \frac{1 - 2.818z^{-1} + 3.97z^{-2} - 2.8180z^{-3} + z^{-4}}{1 - 2.536z^{-1} + 3.215z^{-2} - 2.054z^{-3} + 0.6560z^{-4}} \tag{6.83}$$

1. Draw the normal direct form I structure block diagram.
2. Draw the transposed direct form I structure block diagram.
3. Draw the normal direct form II structure block diagram. Observe that it looks very similar to that in part 2.
4. Draw the transposed direct form II structure block diagram. Observe that it looks very similar to that in part 1.

P6.5 Consider the LTI system given in Problem P6.4.

1. Draw a cascade structure containing 2nd-order normal direct-form-II sections.
2. Draw a cascade structure containing 2nd-order transposed direct-form-II sections.
3. Draw a parallel structure containing 2nd-order normal direct-form-II sections.
4. Draw a parallel structure containing 2nd-order transposed direct-form-II sections.

P6.6 A causal linear time-invariant system is described by

$$y(n) = \sum_{k=0}^{4} \cos(0.1\pi k)x(n-k) - \sum_{k=1}^{5} (0.8)^k \sin(0.1\pi k)y(n-k)$$

Determine and draw the block diagrams of the following structures. Compute the response of the system to

$$x(n) = [1 + 2(-1)^n], \quad 0 \le n \le 50$$

in each case, using the following structures.

1. Normal direct form I
2. Transposed direct form II
3. Cascade form containing 2nd-order normal direct-form-II sections
4. Parallel form containing 2nd-order transposed direct-form-II sections
5. Lattice-ladder form

P6.7 An IIR filter is described by the following system function

$$H(z) = 2\left(\frac{1 + 0z^{-1} + z^{-2}}{1 - 0.8z^{-1} + 0.64z^{-2}}\right) + \left(\frac{2 - z^{-1}}{1 - 0.75z^{-1}}\right) + \left(\frac{1 + 2z^{-1} + z^{-2}}{1 + 0.81z^{-2}}\right)$$

Determine and draw the following structures.

1. Transposed direct form I
2. Normal direct form II
3. Cascade form containing transposed 2nd-order direct-form-II sections
4. Parallel form containing normal 2nd-order direct-form-II sections
5. Lattice-ladder form

P6.8 An IIR filter is described by the following system function

$$H(z) = \left(\frac{-14.75 - 12.9z^{-1}}{1 - \frac{7}{8}z^{-1} + \frac{3}{32}z^{-2}}\right) + \left(\frac{24.5 + 26.82z^{-1}}{1 - z^{-1} + \frac{1}{2}z^{-2}}\right)\left(\frac{1 + 2z^{-1} + z^{-2}}{1 + 0.81z^{-2}}\right)$$

Determine and draw the following structures:

1. Normal direct form I
2. Normal direct form II
3. Cascade form containing transposed 2nd-order direct-form-II sections
4. Parallel form containing transposed 2nd-order direct-form-II sections
5. Lattice-ladder form

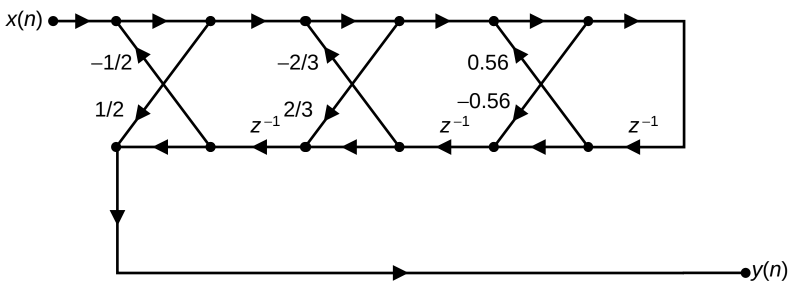


FIGURE P6.2 *Structure for Problem 6.9*

P6.9 Figure P6.2 describes a causal linear time-invariant system. Determine and draw the following structures:

1. Direct form I
2. Direct form II
3. Cascade form containing second-order direct-form-II sections
4. Parallel form containing second-order direct-form-II sections

P6.10 A linear time-invariant system with system function

$$H(z) = \frac{0.05 - 0.01z^{-1} - 0.13z^{-2} + 0.13z^{-4} + 0.01z^{-5} - 0.05z^{-6}}{1 - 0.77z^{-1} + 1.59z^{-2} - 0.88z^{-3} + 1.2z^{-4} - 0.35z^{-5} + 0.31z^{-6}}$$

is to be implemented using a flow graph of the form shown in Figure P6.3.

1. Fill in all the coefficients in the diagram.
2. Is your solution unique? Explain.

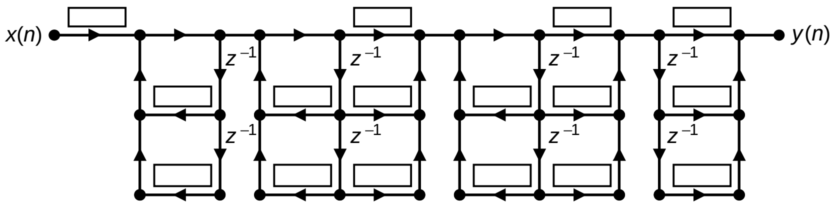


FIGURE P6.3 *Structure for Problem 6.10*

P6.11 A linear time-invariant system with system function

$$H(z) = \frac{0.051 + 0.088z^{-1} + 0.06z^{-2} - 0.029z^{-3} - 0.069z^{-4} - 0.046z^{-5}}{1 - 1.34z^{-1} + 1.478z^{-2} - 0.789z^{-3} + 0.232z^{-4}}$$

is to be implemented using a flow graph of the form shown in Figure P6.4. Fill in all the coefficients in the diagram.

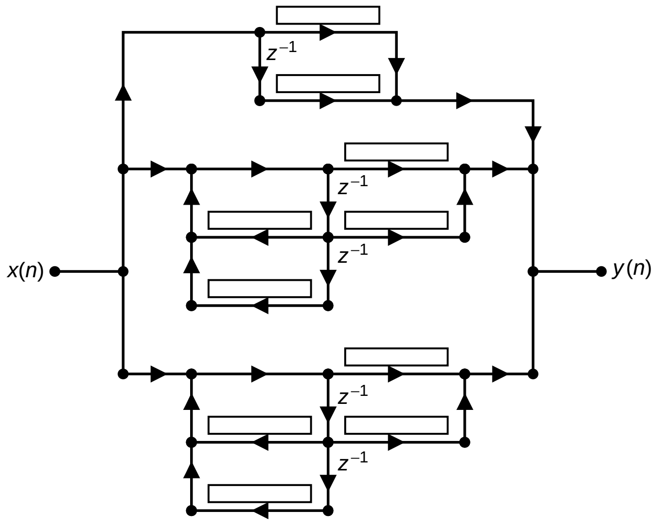


FIGURE P6.4 *Problem for Problem 6.11*

P6.12 Consider the linear time-invariant system given in Problem P6.10.

$$H(z) = \frac{0.05 - 0.01z^{-1} - 0.13z^{-2} + 0.13z^{-4} + 0.01z^{-5} - 0.05z^{-6}}{1 - 0.77z^{-1} + 1.59z^{-2} - 0.88z^{-3} + 1.2z^{-4} - 0.35z^{-5} + 0.31z^{-6}}$$

It is to be implemented using a flow graph of the form shown in Figure P6.5.

1. Fill in all the coefficients in the diagram.
2. Is your solution unique? Explain.

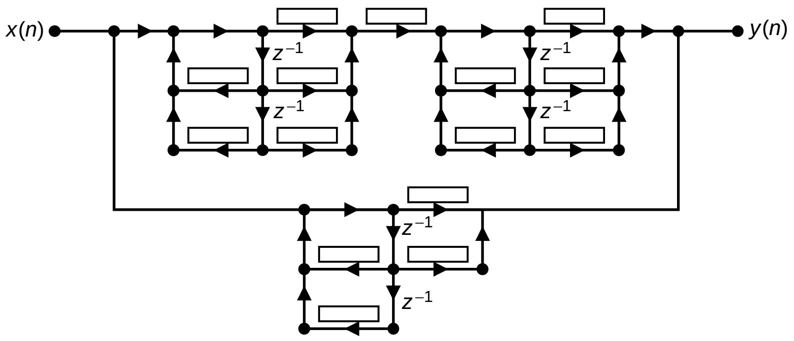


FIGURE P6.5 *Structure for Problem 6.12*

P6.13 The filter structure shown in Figure P6.6 contains a parallel connection of cascade sections. Determine and draw the overall

1. direct form (normal) structure,
2. direct form (transposed) structure,

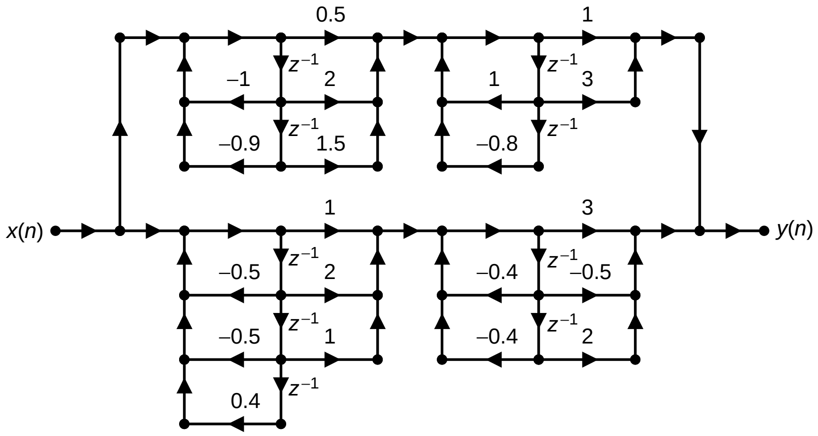


FIGURE P6.6 *Structure for Problem 6.13*

3. cascade form structure containing 2nd-order sections,
4. parallel form structure containing 2nd-order sections.

P6.14 In filter structure shown in Figure P6.7, systems $H_1(z)$ and $H_2(z)$ are subcomponents of a larger system $H(z)$. The system function $H_1(z)$ is given in the parallel form

$$H_1(z) = 2 + \frac{0.2 - 0.3z^{-1}}{1 + 0.9z^{-1} + 0.9z^{-2}} + \frac{0.4 + 0.5z^{-1}}{1 - 0.8z^{-1} + 0.8z^{-2}}$$

and the system function $H_2(z)$ is given in the cascade form

$$H_2(z) = \left(\frac{2 + z^{-1} - z^{-2}}{1 + 1.7z^{-1} + 0.72z^{-2}}\right)\left(\frac{3 + 4z^{-1} + 5z^{-2}}{1 - 1.5z^{-1} + 0.56z^{-2}}\right)$$

1. Express $H(z)$ as a rational function.
2. Draw the block diagram of $H(z)$ as a cascade-form structure.
3. Draw the block diagram of $H(z)$ as a parallel-form structure.

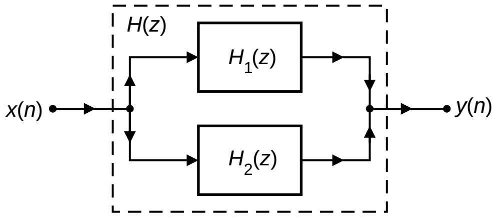


FIGURE P6.7 *Structure for Problem 6.14*

P6.15 The digital filter structure shown in Figure P6.8 is a cascade of 2 parallel sections and corresponds to a 10th-order IIR digital filter system function

$$H(z) = \frac{1 - 2.2z^{-2} + 1.6368z^{-4} - 0.48928z^{-6} + 5395456 \times 10^{-8}z^{-8} - 147456 \times 10^{-8}z^{-10}}{1 - 1.65z^{-2} + 0.8778z^{-4} - 0.17281z^{-6} + 1057221 \times 10^{-8}z^{-8} - 893025 \times 10^{-10}z^{-10}}$$

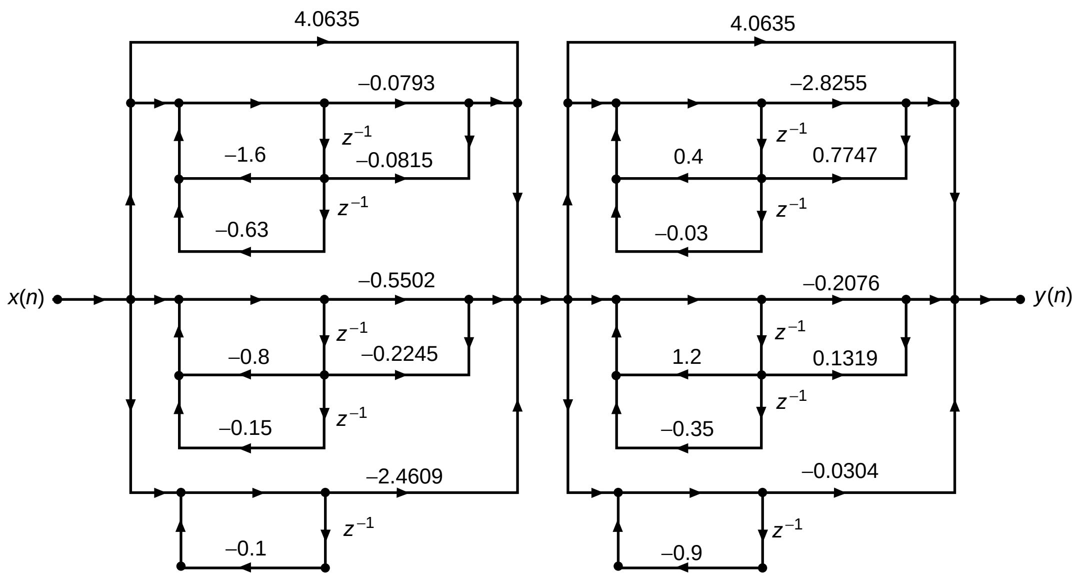


FIGURE P6.8 *Structure for Problem 6.15*

1. Due to an error in labeling, two of the multiplier coefficients (rounded to 4 decimals) in this structure have incorrect values. Locate these 2 multipliers and determine their correct values.
2. Determine and draw an overall cascade structure containing 2nd-order section and which contains the least number of multipliers.

P6.16 As described in this chapter, a linear-phase FIR filter is obtained by requiring certain symmetry conditions on its impulse responses.

1. In the case of symmetrical impulse response, we have $h(n) = h(M-1-n)$, $0 \le n \le M-1$. Show that the resulting phase response is linear in ω and is given by
$$\angle H\left(e^{j\omega}\right) = -\left(\frac{M-1}{2}\right)\omega, \quad -\pi < \omega \le \pi$$
2. Draw the linear-phase structures for this form when $M = 5$ and $M = 6$.
3. In the case of antisymmetrical impulse response, we have $h(n) = -h(M-1-n)$, $0 \le n \le M-1$. Show that the resulting phase response is given by
$$\angle H\left(e^{j\omega}\right) = \pm\frac{\pi}{2} - \left(\frac{M-1}{2}\right)\omega, \quad -\pi < \omega \le \pi$$
4. Draw the linear-phase structures for this form when $M = 5$ and $M = 6$.

P6.17 An FIR filter is described by the difference equation
$$y(n) = \sum_{k=0}^{6} \mathrm{e}^{-0.9|k-3|}x(n-k)$$

Determine and draw the block diagrams of the following structures.

1. Direct form
2. Linear-phase form
3. Cascade form
4. Frequency sampling form

P6.18 A linear time-invariant system is given by the system function

$$H(z) = 2 + 3z^{-1} + 5z^{-2} - 3z^{-3} + 4z^{-5} + 8z^{-7} - 7z^{-8} + 4z^{-9}$$

Determine and draw the block diagrams of the following structures.

1. Direct form
2. Cascade form
3. Lattice form
4. Frequency sampling form

P6.19 Using the conjugate symmetry property of the DFT

$$H(k) = \begin{cases} H(0), & k = 0 \\ H^*(M-k), & k = 1, \ldots, M-1 \end{cases}$$

and the conjugate symmetry property of the W_M^{-k} factor, show that (6.12) can be put in the form (6.13) and (6.14) for real FIR filters.

P6.20 To avoid poles on the unit circle in the frequency sampling structure, one samples $H(z)$ at $z_k = re^{j2\pi k/M}$, $k = 0, \ldots, M-1$ where $r \approx 1$(but < 1), as discussed in Section 6.3.

1. Using

$$H\left(re^{j2\pi k/M}\right) \approx H(k),$$

show that the frequency-sampling structure is given by

$$H(z) = \frac{1-(rz)^{-M}}{M}\left\{\sum_{k=1}^{L} 2\left|H(k)\right| H_k(z) + \frac{H(0)}{1-rz^{-1}} + \frac{H(M/2)}{1+rz^{-1}}\right\}$$

where

$$H_k(z) = \frac{\cos\left[\angle H(k)\right] - rz^{-1}\cos\left[\angle H(k) - \frac{2\pi k}{M}\right]}{1 - 2rz^{-1}\cos\left(\frac{2\pi k}{M}\right) + r^2 z^{-2}}, \quad k = 1, \ldots, L$$

and M is even.

2. Modify the MATLAB function `dir2fs` (which was developed in Section 6.3) to implement this frequency-sampling form. The format of this function should be

```
[C,B,A,rM] = dir2fs(h,r)
% Direct form to Frequency Sampling form conversion
% -------------------------------------------------
% [C,B,A,rM] = dir2fs(h,r)
%  C = Row vector containing gains for parallel sections
%  B = Matrix containing numerator coefficients arranged in rows
%  A = Matrix containing denominator coefficients arranged in rows
% rM = r^M factor needed in the feedforward loop
%  h = impulse response vector of an FIR filter
%  r = radius of the circle over which samples are taken (r<1)
%
```

3. Determine the frequency sampling structure for the impulse response given in Example 6.6 using this function.

P6.21 Determine the impulse response of an FIR filter with lattice parameters

$$K_0 = 2, \quad K_1 = 0.6, \quad K_2 = 0.3, \quad K_3 = 0.5, \quad K_4 = 0.9$$

Draw the direct form and lattice form structures of this filter.

P6.22 Consider the following system function of an FIR filter

$$H(z) = 1 - 4z^{-1} + 6.4z^{-2} - 5.12z^{-3} + 2.048z^{-4} - 0.32768z^{-5}$$

1. Provide block diagram structures in the following forms:
   (a) Normal and transposed direct forms
   (b) Cascade of five 1st-order sections
   (c) Cascade of one 1st-order section and two 2nd-order sections
   (d) Cascade of one 2nd-order section and one 3rd-order section
   (e) Frequency-sampling structure with real coefficients
2. The computational complexity of a digital filter structure can be given by the total number of multiplications and the total number of 2-input additions that are required per output point. Assume that $x(n)$ is real and that multiplication by 1 is not counted as a multiplication. Compare the computational complexity of each of these structures.

P6.23 A causal digital filter is described by the following zeros:

$$z_1 = 0.5\,e^{j60^\circ}, \quad z_2 = 0.5\,e^{-j60^\circ}, \quad z_3 = 2\,e^{j60^\circ}, \quad z_4 = 2\,e^{-j60^\circ},$$
$$z_5 = 0.25\,e^{j30^\circ}, \quad z_6 = 0.25\,e^{-j30^\circ}, \quad z_7 = 4\,e^{j30^\circ}, \quad z_8 = 4\,e^{-j30^\circ},$$

and poles: $\{p_i\}_{i=1}^{8} = 0$.

1. Determine the phase response of this filter, and show that it is a linear-phase FIR filter.
2. Determine the impulse response of the filter.
3. Draw a block diagram of the filter structure in the direct form.
4. Draw a block diagram of the filter structure in the linear-phase form.

P6.24 MATLAB provides the built-in functions `dec2bin` and `bin2dec` to convert non-negative decimal integers into binary codes and vice versa, respectively.

1. Develop a function `B = sm2bin(D)` to convert a sign-magnitude format decimal integer D into its binary representation B. Verify your function on the following numbers:

   (a) $D = 1001$ (b) $D = -63$ (c) $D = -449$ (d) $D = 978$ (e) $D = -205$

2. Develop a function `D = bin2sm(B)` to convert a binary representation B into its sign-magnitude format decimal integer D. Verify your function on the following representations:

   (a) $B =$ `1010` (b) $B =$ `011011011` (c) $B =$ `11001`
   (d) $B =$ `1010101` (e) $B =$ `011011`

P6.25 Using the function `TwosComplement` as a model, develop a function `y = TensComplement (x,N)` that converts a sign-magnitude format integer x into the N-digit ten's-complement integer y.

1. Verify your function using the following integers:

   (a) $x = 1234$, $N = 6$ (b) $x = -603$, $N = 4$ (c) $x = -843$, $N = 5$
   (d) $x = -1978$, $N = 6$ (e) $x = 50$, $N = 3$

2. Using the ten's-complement format, perform the following arithmetic operations. In each case, choose an appropriate value on N for the meaningful result.

   (a) $123 + 456 - 789$ (b) $648 + 836 - 452$ (c) $2001 + 3756$
   (d) $-968 + 4539$ (e) $888 - 666 + 777$

   Verify your results using decimal operations.

P6.26 The function `OnesComplement` developed in this chapter converts signed integers into one's-complement format decimal representations. In this problem we will develop functions that will operate on fractional numbers.

1. Develop a MATLAB function y = sm2oc(x, B) that converts the sign-magnitude format fraction x into the B-bit 1s-complement format decimal equivalent number y. Verify your function on the following numbers. In each case the numbers to be considered are both positive and negative. Also, in each case select the appropriate number of bits B.

   (a) $x = \pm 0.5625$ (b) $x = \pm 0.40625$ (c) $x = \pm 0.953125$
   (d) $x = \pm 0.1328125$ (e) $x = \pm 0.7314453125$

2. Develop a MATLAB function x = oc2sm(y, B) that converts the B-bit one's-complement format decimal equivalent number y into the sign-magnitude format fraction x. Verify your function on the following fractional binary representations:

   (a) $y =$ `1▲10110` (b) $y =$ `0.▲011001` (c) $y =$ `1▲00110011`
   (d) $y =$ `1▲11101110` (e) $y =$ `0▲00010001`

P6.27 The function `TwosComplement` developed in this chapter converts signed integers into two's-complement format decimal representations. In this problem we will develop functions that will operate on fractional numbers.

1. Develop a MATLAB function y = sm2tc(x, B) that converts the sign-magnitude format fraction x into the B-bit two's-complement format decimal equivalent number y. Verify

your function on the following numbers. In each case the numbers to be considered are both positive and negative. Also, in each case select the appropriate number of bits B.

(a) $x = \pm 0.5625$ (b) $x = \pm 0.40625$ (c) $x = \pm 0.953125$
(d) $x = \pm 0.1328125$ (f) $x = \pm 0.7314453125$

Compare your representations with those in Problem P6.26, part 1.

2. Develop a MATLAB function x = tc2sm(y, B) that converts the B-bit two's-complement format decimal equivalent number y into the sign-magnitude format fraction x. Verify your function on the following fractional binary representations:

(a) $y =$ `1▲10110` (b) $y =$ `0.▲011001` (c) $y =$ `1▲00110011`
(d) $y =$ `1▲11101110` (e) $y =$ `0▲00010001`

Compare your representations with those in Problem P6.26, part 2.

P6.28 Determine the 10-bit sign-magnitude, one's-complement, and two's-complement representation of the following decimal numbers:

(a) 0.12345 (b) -0.56789 (c) 0.38452386 (d) -0.762349 (e) -0.90625

P6.29 Consider a 32-bit floating-point number representation with a 6-bit exponent and a 25-bit mantissa.

1. Determine the value of the smallest number that can be represented.
2. Determine the value of the largest number that can be represented.
3. Determine the dynamic range of this floating-point representation and compare it with the dynamic range of a 32-bit fixed-point signed integer representation.

P6.30 Show that the magnitudes of floating-point numbers in a 32-bit IEEE standard range from 1.18×10^{-38} to 3.4×10^{38}.

P6.31 Compute and plot the truncation error characteristics when $B = 4$ for the sign-magnitude, one's-complement, and two's-complement formats.

P6.32 Consider the 3rd-order elliptic lowpass filter:

$$H(z) = \frac{0.1214\left(1 - 1.4211z^{-1} + z^{-2}\right)\left(1 + z^{-1}\right)}{\left(1 - 1.4928z^{-1} + 0.8612z^{-2}\right)\left(1 - 0.6183z^{-1}\right)}$$

1. If the filter is realized using a direct-form structure, determine its pole sensitivity.
2. If the filter is realized using a cascade-form structure, determine its pole sensitivity.

P6.33 Consider the filter described by the difference equation

$$y(n) = \frac{1}{\sqrt{2}}y(n-1) - x(n) + \sqrt{2}x(n-1) \tag{6.84}$$

1. Show that this filter is an all-pass filter (i.e., $|H(e^{j\omega})|$) is a constant over the entire frequency range $-\pi \le \omega \le \pi$. Verify your answer by plotting the magnitude response $|H(e^{j\omega})|$ over the normalized frequency range $0 \le \omega/\pi \le 1$. Use `subplot(3,1,1)`.
2. Round the coefficients of the difference equation in (6.84) to 3 decimals. Is the filter still all-pass? Verify your answer by plotting the resulting magnitude response, $|\hat{H}_1(e^{j\omega})|$, over the normalized frequency range $0 \le \omega/\pi \le 1$. Use `subplot(3,1,2)`.
3. Round the coefficients of the difference equation in (6.84) to 2 decimals. Is the filter still all-pass? Verify your answer by plotting the resulting magnitude response, $|\hat{H}_2(e^{j\omega})|$, over the normalized frequency range $0 \le \omega/\pi \le 1$. Use `subplot(3,1,3)`.

4. Explain why the magnitude $|\hat{H}_1(e^{j\omega})|$ is "different" from the magnitude $|\hat{H}_2(e^{j\omega})|$.

P6.34 An IIR lowpass filter designed to meet the specifications of 0.5 dB ripple in the passband, 60 dB ripple in the stopband, a passband edge frequency $\omega_{\mathrm{p}} = 0.25\pi$, and a stopband edge frequency $\omega_{\mathrm{s}} = 0.3\pi$ is obtained using the following MATLAB script:

```
wp = 0.25*pi; ws = 0.3*pi; Rp = 0.5; As = 60;
[N, Wn] = ellipord(wp/pi, ws/pi, Rp, As);
[b,a] = ellip(N,Rp,As,Wn);
```

The filter coefficients b_k and a_k are in the arrays `b` and `a`, respectively, and can be considered to have infinite precision.

1. Using infinite precision, plot the log-magnitude and phase responses of the designed filter. Use two rows and one column of subplots.
2. Quantize the direct-form coefficients to 4 decimals (by rounding). Now plot the log-magnitude and phase responses of the resulting filter. Use 2 rows and 1 column of subplots.
3. Quantize the direct form coefficients to 3 decimals (by rounding). Now plot the log-magnitude and phase responses of the resulting filter. Use 2 rows and 1 column of subplots.
4. Comment on the plots in parts 1, 2, and 3.

P6.35 Consider the digital lowpass filter used in Problem P6.34.

1. Using infinite precision and cascade-form realization, plot the log-magnitude and phase responses of the designed filter. Use two rows and one column of subplots.
2. Quantize the cascade-form coefficients to 4 decimals (by rounding). Now plot the log-magnitude and phase responses of the resulting filter. Use two rows and one column of subplots.
3. Quantize the cascade-form coefficients to 3 decimals (by rounding). Now plot the log-magnitude and phase responses of the resulting filter. Use two rows and one column of subplots.
4. Comment on the plots in the above three parts and compare them with the similar plots in Problem P6.34.

P6.36 A length-32 linear-phase FIR bandpass filter that satisfies the requirements of 60 dB stopband attenuation, lower stopband edge frequency $\omega_{\mathrm{s}_1} = 0.2\pi$, and upper stopband edge frequency $\omega_{\mathrm{s}_2} = 0.8\pi$ is obtained using the following MATLAB script.

```
ws1 = 0.2*pi; ws2 = 0.8*pi; As = 60;
M = 32; Df = 0.2115;
fp1 = ws1/pi+Df; fp2 = ws2/pi-Df;
h = firpm(M-1,[0,ws1/pi,fp1,fp2,ws2/pi,1],[0,0,1,1,0,0]);
```

The filter impulse response $h(n)$ is in the array `h` and can be considered to have infinite precision.

1. Using infinite precision, plot the log-magnitude and amplitude responses of the designed filter. Use 2 rows and 1 column of subplots.

2. Quantize the direct-form coefficients to 4 decimals (by rounding). Now plot the log-magnitude and amplitude responses of the resulting filter. Use 2 rows and 1 column of subplots.
3. Quantize the direct-form coefficients to 3 decimals (by rounding). Now plot the log-magnitude and amplitude responses of the resulting filter. Use 2 rows and 1 column of subplots.
4. Comment on the plots in parts 1, 2, and 3.
5. Based on the results of this problem, determine how many significant *bits* (and not decimals) are needed in practice to represent FIR direct form realizations.

P6.37 The digital filter structure shown in Figure P6.9 is a cascade of 2 parallel sections and corresponds to a 10th-order IIR digital filter system function

$$H(z) = \frac{1 - 2.2z^{-2} + 1.6368z^{-4} - 0.48928z^{-6} + 5395456 \times 10^{-8}z^{-8} - 147456 \times 10^{-8}z^{-10}}{1 - 1.65z^{-2} + 0.8778z^{-4} - 0.17281z^{-6} + 1057221 \times 10^{-8}z^{-8} - 893025 \times 10^{-10}z^{-10}}$$

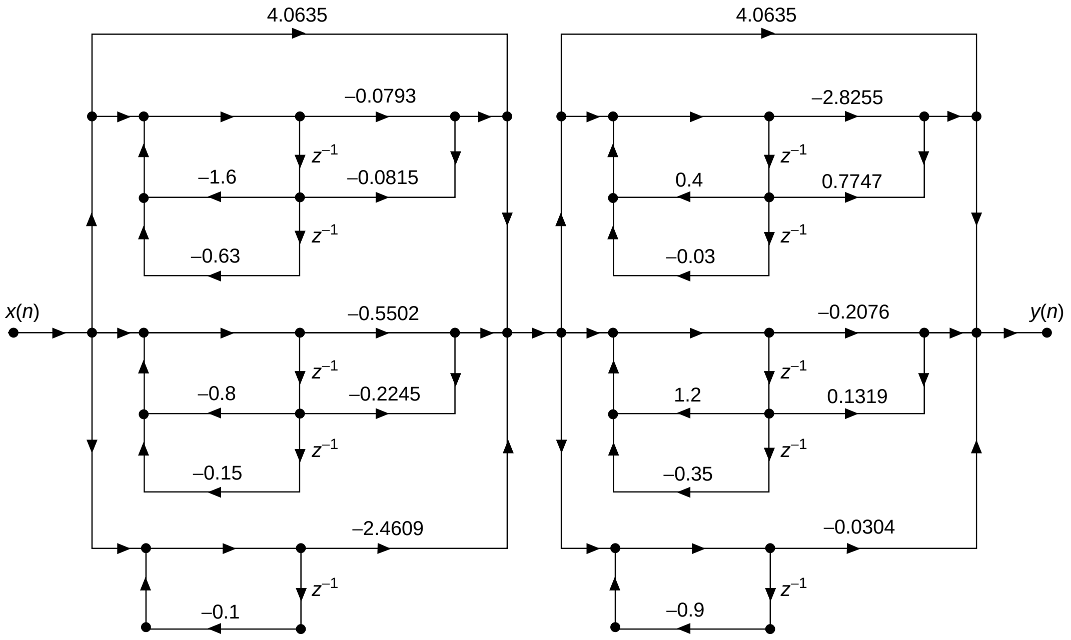


FIGURE P6.9 *Structure for Problem P6.37*

1. Due to an error in labeling, two of the multiplier coefficients (rounded to 4 decimals) in this structure have incorrect values. Locate these 2 multipliers and determine their correct values.
2. By inspecting the pole locations of the system function $H(z)$, you should realize that this structure is sensitive to the coefficient quantization. Suggest, with justification, an alternative structure that in *your opinion* is least sensitive to coefficient quantization.

P6.38 An IIR bandstop digital filter that satisfies the requirements:

$$\begin{aligned} 0.95 \le |H(e^{j\omega})| &\le 1.05, & 0 \le |\omega| &\le 0.25\pi \\ 0 \le |H(e^{j\omega})| &\le 0.01, & 0.35\pi \le |\omega| &\le 0.65\pi \\ 0.95 \le |H(e^{j\omega})| &\le 1.05, & 0.75\pi \le |\omega| &\le \pi \end{aligned}$$

can be obtained using the following MATLAB script:

```
wp = [0.25,0.75]; ws = [0.35,0.65]; delta1 = 0.05; delta2 = 0.01;
[Rp,As] = delta2db(delta1,delta2);
[N, wn] = cheb2ord(wp, ws, Rp, As);
[b,a] = cheby2(N,As,wn,'stop');
```

The filter coefficients b_k and a_k are in the arrays `b` and `a`, respectively, and can be considered to have infinite precision.

1. Using infinite precision, provide the log-magnitude response plot and the pole-zero plot of the designed filter.
2. Assuming direct-form structure and a 12-bit representation for filter coefficients, provide the log-magnitude response plot and the pole-zero plot of the designed filter. Use the `Qcoeff` function.
3. Assuming cascade-form structure and a 12-bit representation for filter coefficients, provide the log-magnitude response plot and the pole-zero plot of the designed filter. Use the `Qcoeff` function.

P6.39 An IIR lowpass digital filter that satisfies the specifications:

$$\begin{aligned} &\text{passband edge:} \quad 0.4\pi, \quad R_p = 0.5 \text{ dB} \\ &\text{stopband edge:} \quad 0.6\pi, \quad As = 50 \text{ dB} \end{aligned}$$

can be obtained using the following MATLAB script:

```
wp = 0.4; ws = 0.6; Rp = 0.5; As = 50;
[N, wn] = buttord(wp, ws, Rp, As);
[b,a] = butter(N,wn);
```

The filter coefficients b_k and a_k are in the arrays `b` and `a`, respectively, and can be considered to have infinite precision.

1. Using infinite precision, provide the magnitude response plot and the pole-zero plot of the designed filter.
2. Assuming direct-form structure and a 10-bit representation for filter coefficients, provide the magnitude response plot and the pole-zero plot of the designed filter. Use the `Qcoeff` function.
3. Assuming cascade-form structure and a 10-bit representation for filter coefficients, provide the magnitude response plot and the pole-zero plot of the designed filter. Use the `Qcoeff` function.

P6.40 An IIR highpass digital filter that satisfies the specifications:

$$\begin{aligned} &\text{stopband edge:} \quad 0.4\pi, \quad A_s = 60 \text{ dB} \\ &\text{passband edge:} \quad 0.6\pi, \quad R_p = 0.5 \text{ dB} \end{aligned}$$

can be obtained using the following MATLAB script:

```
wp = 0.6; ws = 0.4; Rp = 0.5; As = 60;
[N,wn] = ellipord(wp, ws, Rp, As);
[b,a] = ellip(N,Rp,As,wn,'high');
```

The filter coefficients b_k and a_k are in the arrays b and a, respectively, and can be considered to have infinite precision.

1. Using infinite precision, provide the magnitude response plot and the pole-zero plot of the designed filter.
2. Assuming direct-form structure and a 10-bit representation for filter coefficients, provide the magnitude response plot and the pole-zero plot of the designed filter. Use the Qcoeff function.
3. Assuming parallel-form structure and a 10-bit representation for filter coefficients, provide the magnitude response plot and the pole-zero plot of the designed filter. Use the Qcoeff function.

P6.41 A bandstop linear-phase FIR filter that satisfies the specifications:

lower stopband edge:	0.4π	$A_s = 50$ dB
upper stopband edge:	0.6π	
lower passband edge:	0.3π	$R_p = 0.2$ dB
upper passband edge:	0.7π	

can be obtained using the following MATLAB script:

```
wp1 = 0.3; ws1 = 0.4; ws2 = 0.6; wp2 = 0.7; Rp = 0.2; As = 50;
[delta1,delta2] = db2delta(Rp,As);
b = firpm(44,[0,wp1,ws1,ws2,wp2,1],[1,1,0,0,1,1],...
    [delta2/delta1,1,delta2/delta1]);
```

The filter impulse response $h(n)$ is in the array b and can be considered to have infinite precision.

P6.42 A bandpass linear-phase FIR filter that satisfies the specifications:

$$\begin{array}{rcccl} 0 & \le & |H(e^{j\omega})| & \le & 0.01, \quad 0 \le \omega \le 0.25\pi \\ 0.95 & \le & |H(e^{j\omega})| & \le & 1.05, \; 0.35\pi \le \omega \le 0.65\pi \\ 0 & \le & |H(e^{j\omega})| & \le & 0.01, \; 0.75\pi \le \omega \le \pi \end{array}$$

can be obtained using the following MATLAB script:

```
ws1 = 0.25; wp1 = 0.35; wp2 = 0.65; ws2 = 0.75;
delta1 = 0.05; delta2 = 0.01;
b = firpm(40,[0,ws1,wp1,wp2,ws2,1],[0,0,1,1,0,0],...
    [1,delta2/delta1,1]);
```

The filter impulse response $h(n)$ is in the array b and can be considered to have infinite precision.

CHAPTER 7

FIR Filter Design

We now turn our attention to the inverse problem of designing systems from the given specifications. It is an important as well as a difficult problem. In digital signal processing there are two important types of systems. The first type of systems perform signal filtering in the time domain and hence are called *digital filters*. The second type of systems provide signal representation in the frequency domain and are called *spectrum analyzers*. In Chapter 5 we described signal representations using the DFT. In this and the next chapter we will study several basic design algorithms for both FIR and IIR filters. These designs are mostly of the *frequency selective* type; that is, we will design primarily multiband lowpass, highpass, bandpass, and bandstop filters. In FIR filter design we will also consider systems like differentiators or Hilbert transformers, which, although not frequency-selective filters, nevertheless follow the design techniques being considered. More sophisticated filter designs are based on arbitrary frequency-domain specifications and require tools that are beyond the scope of this book.

We first begin with some preliminary issues related to design philosophy and design specifications. These issues are applicable to both FIR and IIR filter designs. We will then study FIR filter design algorithms in the rest of this chapter. In Chapter 8 we will provide a similar treatment for IIR filters.

7.1 PRELIMINARIES

The design of a digital filter is carried out in three steps:

- **Specifications:** Before we can design a filter, we must have some specifications. These specifications are determined by the applications.
- **Approximations:** Once the specifications are defined, we use various concepts and mathematics that we studied so far to come up with a filter description that approximates the given set of specifications. This step is the topic of filter design.
- **Implementation:** The product of the above step is a filter description in the form of either a difference equation, or a system function $H(z)$, or an impulse response $h(n)$. From this description we implement the filter in hardware or through software on a computer as we discussed in Chapter 6.

In this and the next chapter we will discuss in detail only the second step, which is the conversion of specifications into a filter description.

In many applications like speech or audio signal processing, digital filters are used to implement frequency-selective operations. Therefore, specifications are required in the frequency-domain in terms of the desired magnitude and phase response of the filter. Generally a linear phase response in the passband is desirable. In the case of FIR filters, it is possible to have exact linear phase as we have seen in Chapter 6. In the case of IIR filters a linear phase in the passband is not achievable. Hence we will consider *magnitude-only* specifications.

The magnitude specifications are given in one of two ways. The first approach is called *absolute specifications*, which provide a set of requirements on the magnitude response function $|H(e^{j\omega})|$. These specifications are generally used for FIR filters. IIR filters are specified in a somewhat different way, which we will discuss in Chapter 8. The second approach is called *relative specifications*, which provide requirements in *decibels* (dB), given by

$$\text{dB scale} = -20\log_{10}\frac{|H(e^{j\omega})|}{|H(e^{j\omega})|_{\max}} \geq 0$$

This approach is the most popular one in practice and is used for both FIR and IIR filters. To illustrate these specifications, we will consider a lowpass filter design as an example.

7.1.1 ABSOLUTE SPECIFICATIONS

A typical absolute specification of a lowpass filter is shown in Figure 7.1a, in which

- band $[0, \omega_p]$ is called the *passband*, and δ_1 is the tolerance (or ripple) that we are willing to accept in the ideal passband response,

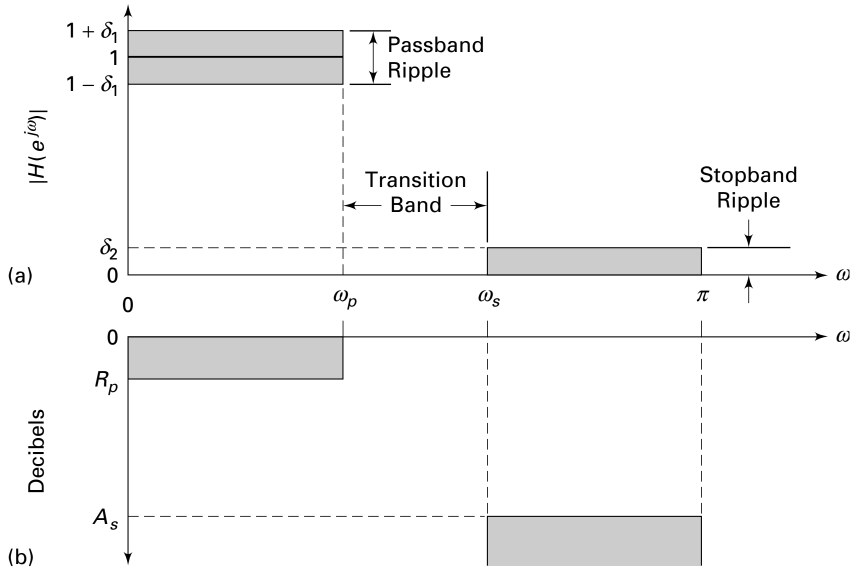


FIGURE 7.1 *FIR filter specifications: (a) absolute (b) relative*

- band $[\omega_s, \pi]$ is called the *stopband*, and δ_2 is the corresponding tolerance (or ripple), and
- band $[\omega_p, \omega_s]$ is called the *transition band*, and there are no restrictions on the magnitude response in this band.

7.1.2 RELATIVE (DB) SPECIFICATIONS

A typical absolute specification of a lowpass filter is shown in Figure 7.1b, in which

- R_p is the passband ripple in dB, and
- A_s is the stopband attenuation in dB.

The parameters given in these two specifications are obviously related. Since $|H(e^{j\omega})|_{\max}$ in absolute specifications is equal to $(1+\delta_1)$, we have

$$R_p = -20\log_{10}\frac{1-\delta_1}{1+\delta_1} > 0 \ (\approx 0) \tag{7.1}$$

and

$$A_s = -20\log_{10}\frac{\delta_2}{1+\delta_1} > 0 \ (\gg 1) \tag{7.2}$$

□ **EXAMPLE 7.1** In a certain filter's specifications the passband ripple is 0.25 dB, and the stopband attenuation is 50 dB. Determine δ_1 and δ_2.

Solution Using (7.1), we obtain

$$R_p = 0.25 = -20\log_{10}\frac{1-\delta_1}{1+\delta_1} \Rightarrow \delta_1 = 0.0144$$

Using (7.2), we obtain

$$A_s = 50 = -20\log_{10}\frac{\delta_2}{1+\delta_1} = -20\log_{10}\frac{\delta_2}{1+0.0144} \Rightarrow \delta_2 = 0.0032 \qquad \square$$

□ **EXAMPLE 7.2** Given the passband tolerance $\delta_1 = 0.01$ and the stopband tolerance $\delta_2 = 0.001$, determine the passband ripple R_p and the stopband attenuation A_s.

Solution From (7.1) the passband ripple is

$$R_p = -20\log_{10}\frac{1-\delta_1}{1+\delta_1} = 0.1737 \text{ dB}$$

and from (7.2) the stopband attenuation is

$$A_s = -20\log_{10}\frac{\delta_2}{1+\delta_1} = 60 \text{ dB} \qquad \square$$

Problem P7.1 develops MATLAB functions to convert one set of specifications into another.

These specifications were given for a lowpass filter. Similar specifications can also be given for other types of frequency-selective filters, such as highpass or bandpass. However, the most important design parameters are *frequency-band tolerances* (or ripples) and *band-edge frequencies*. Whether the given band is a passband or a stopband is a relatively minor issue. Therefore in describing design techniques, we will concentrate on a lowpass filter. In the next chapter we will discuss how to transform a lowpass filter into other types of frequency-selective filters. Hence it makes more sense to develop techniques for a lowpass filter so that we can compare these techniques. However, we will also provide examples of other types of filters. In light of this discussion our design goal is the following.

Problem statement Design a lowpass filter (i.e., obtain its system function $H(z)$ or its difference equation) that has a passband $[0, \omega_p]$ with tolerance δ_1 (or R_p in dB) and a stopband $[\omega_s, \pi]$ with tolerance δ_2 (or A_s in dB).

In this chapter we turn our attention to the design and approximation of FIR digital filters. These filters have several design and implementational advantages:

- The phase response can be exactly linear.
- They are relatively easy to design since there are no stability problems.
- They are efficient to implement.
- The DFT can be used in their implementation.

As we discussed in Chapter 6, we are generally interested in linear-phase frequency-selective FIR filters. Advantages of a linear-phase response are:

- design problem contains only real arithmetic and not complex arithmetic
- linear-phase filters provide no delay distortion and only a fixed amount of delay
- for the filter of length M (or order $M - 1$) the number of operations are of the order of $M/2$ as we discussed in the linear-phase filter implementation

We first begin with a discussion of the properties of the linear-phase FIR filters, which are required in design algorithms. Then we will discuss three design techniques, namely the window design, the frequency sampling design, and the optimal equiripple design techniques for linear-phase FIR filters.

7.2 PROPERTIES OF LINEAR-PHASE FIR FILTERS

In this section we discuss shapes of impulse and frequency responses and locations of system function zeros of linear-phase FIR filters. Let $h(n)$, $0 \leq n \leq M - 1$ be the impulse response of length (or duration) M. Then the system function is

$$H(z) = \sum_{n=0}^{M-1} h(n)z^{-n} = z^{-(M-1)} \sum_{n=0}^{M-1} h(n)z^{M-1-n}$$

which has $(M - 1)$ poles at the origin $z = 0$ (trivial poles) and $(M - 1)$ zeros located anywhere in the z-plane. The frequency response function is

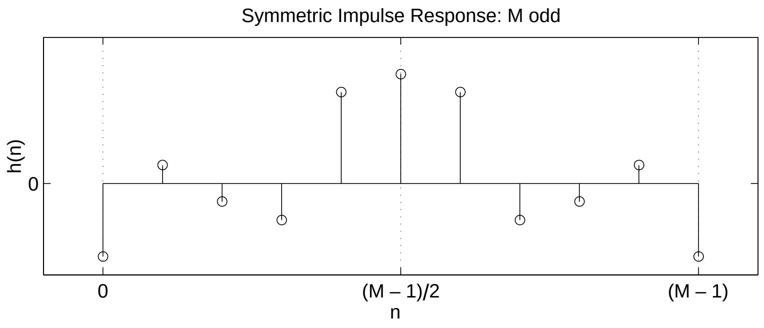
$$H(e^{j\omega}) = \sum_{n=0}^{M-1} h(n)e^{-j\omega n}, \quad -\pi < \omega \leq \pi$$

Now we will discuss specific requirements on the forms of $h(n)$ and $H(e^{j\omega})$ as well as requirements on the specific locations of $(M - 1)$ zeros that the linear-phase constraint imposes.

7.2.1 IMPULSE RESPONSE $h(n)$

We impose a linear-phase constraint

$$\angle H(e^{j\omega}) = -\alpha\omega, \quad -\pi < \omega \leq \pi$$

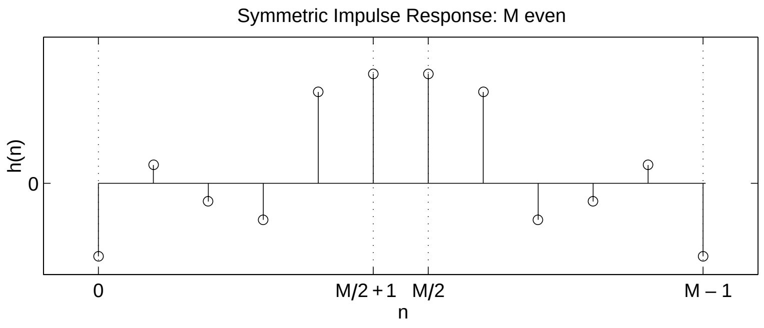


where α is a *constant phase delay.* Then we know from Chapter 6 that $h(n)$ must be symmetric, that is,

$$h\,(n) = h(M-1-n), \quad 0 \le n \le (M-1) \text{ with } \alpha = \frac{M-1}{2} \tag{7.3}$$

Hence $h(n)$ is symmetric about α, which is the index of symmetry. There are two possible types of symmetry:

- *M odd:* In this case $\alpha = (M-1)/2$ is an integer. The impulse response is as shown below.
- *M even:* In this case $\alpha = (M-1)/2$ is not an integer. The impulse response is as shown here.



We also have a second type of "linear-phase" FIR filter if we require that the phase response $\angle H(e^{j\omega})$ satisfy the condition

$$\angle H(e^{j\omega}) = \beta - \alpha\omega$$

which is a straight line but not through the origin. In this case α is not a constant phase delay, but

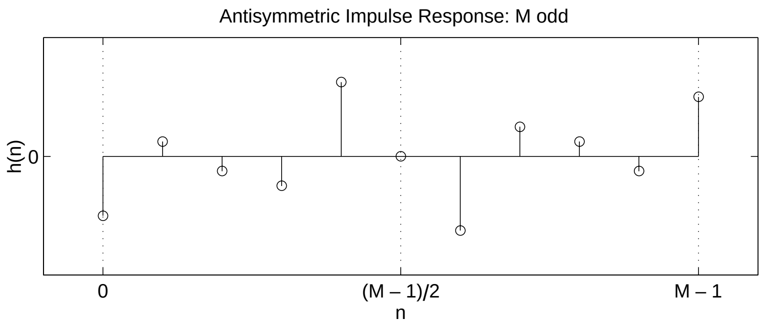
$$\frac{d\angle H(e^{j\omega})}{d\omega} = -\alpha$$

is constant, which is the group delay. Therefore α is called a *constant group delay*. In this case, as a group, frequencies are delayed at a constant rate. But some frequencies may get delayed more and others delayed less. For this type of linear phase one can show that

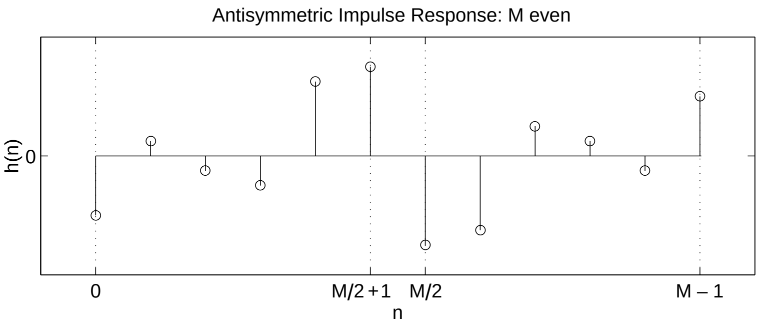
$$h(n) = -h(M-1-n), \quad 0 \le n \le (M-1); \; \alpha = \frac{M-1}{2}, \; \beta = \pm\frac{\pi}{2} \tag{7.4}$$

This means that the impulse response $h(n)$ is *antisymmetric*. The index of symmetry is still $\alpha = (M-1)/2$. Once again we have two possible types, one for M odd and one for M even.

- *M odd:* In this case $\alpha = (M-1)/2$ is an integer and the impulse response is as shown.



  Note that the sample $h(\alpha)$ at $\alpha = (M-1)/2$ must necessarily be equal to zero, i.e., $h((M-1)/2) = 0$.
- *M even:* In this case $\alpha = (M-1)/2$ is not an integer and the impulse response is as shown.



7.2.2 FREQUENCY RESPONSE $H(e^{j\omega})$

When the cases of symmetry and antisymmetry are combined with odd and even M, we obtain four types of linear-phase FIR filters. Frequency response functions for each of these types have some peculiar expressions and shapes. To study these responses, we write $H(e^{j\omega})$ as

$$H(e^{j\omega}) = H_r(\omega)e^{j(\beta-\alpha\omega)}; \quad \beta = \pm\frac{\pi}{2},\ \alpha = \frac{M-1}{2} \tag{7.5}$$

where $H_r(\omega)$ is an *amplitude response* function and not a magnitude response function. The amplitude response is a real function, but unlike the magnitude response, which is always positive, the amplitude response may be both positive and negative. The phase response associated with the magnitude response is a *discontinuous* function, while that associated with the amplitude response is a *continuous linear* function. To illustrate the difference between these two types of responses, consider the following example.

□ **EXAMPLE 7.3** Let the impulse response be $h(n) = \{\underset{\uparrow}{1}, 1, 1\}$. Determine and draw frequency responses.

Solution The frequency response function is

$$\begin{aligned} H(e^{j\omega}) &= \sum_0^2 h(n)e^{j\omega n} = 1 + 1e^{-j\omega} + e^{-j2\omega} = \left\{e^{j\omega} + 1 + e^{-j\omega}\right\} e^{-j\omega} \\ &= \{1 + 2\cos\omega\}\, e^{-j\omega} \end{aligned}$$

From this the magnitude and the phase responses are

$$\begin{aligned} |H(e^{j\omega})| &= |1 + 2\cos\omega|\,, \quad 0 < \omega \le \pi \\ \angle H(e^{j\omega}) &= \begin{cases} -\omega, & 0 < \omega < 2\pi/3 \\ \pi - \omega, & 2\pi/3 < \omega < \pi \end{cases} \end{aligned}$$

since $\cos\omega$ can be both positive and negative. In this case the phase response is *piecewise linear*. On the other hand, the amplitude and the corresponding phase responses are

$$\left.\begin{aligned} H_r(\omega) &= 1 + 2\cos\omega, \\ \angle H(e^{j\omega}) &= -\omega, \end{aligned}\right. \qquad -\pi < \omega \le \pi$$

In this case the phase response is *truly linear*. These responses are shown in Figure 7.2. From this example, the difference between the magnitude and the amplitude (or between the piecewise linear and the linear-phase) responses should be clear. □

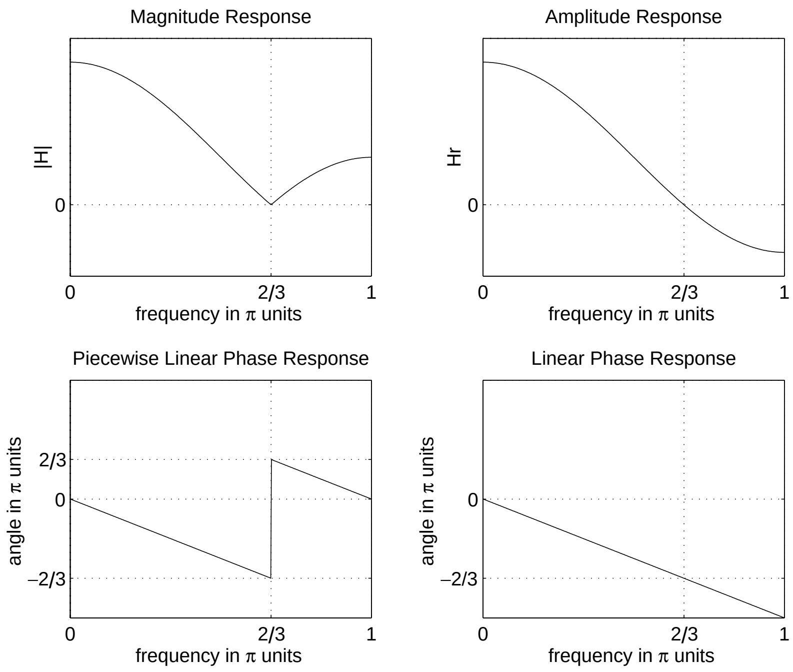


FIGURE 7.2 *Frequency responses in Example 7.3*

Type-1 linear-phase FIR filter: Symmetrical impulse response, M odd In this case $\beta = 0$, $\alpha = (M-1)/2$ is an integer, and $h(n) = h(M-1-n), \quad 0 \le n \le M-1$. Then we can show (see Problem P7.2) that

$$H(e^{j\omega}) = \left[\sum_{n=0}^{(M-1)/2} a(n)\cos\omega n\right] e^{-j\omega(M-1)/2} \tag{7.6}$$

where sequence $a(n)$ is obtained from $h(n)$ as

$$\begin{aligned} a(0) &= h\left(\frac{M-1}{2}\right) \quad : \quad \text{the middle sample} \\ a(n) &= 2h\left(\frac{M-1}{2} - n\right), \quad 1 \le n \le \frac{M-3}{2} \end{aligned} \tag{7.7}$$

Comparing (7.5) with (7.6), we have

$$H_r(\omega) = \sum_{n=0}^{(M-1)/2} a(n)\cos\omega n \tag{7.8}$$

Type-2 linear-phase FIR filter: Symmetrical impulse response, M even In this case again $\beta = 0$, $h(n) = h(M-1-n), \quad 0 \le n \le M-1$, but $\alpha = (M-1)/2$ is not an integer. Then we can show (see Problem P7.3) that

$$H(e^{j\omega}) = \left[\sum_{n=1}^{M/2} b(n) \cos\left\{\omega\left(n - \frac{1}{2}\right)\right\}\right] e^{-j\omega(M-1)/2} \tag{7.9}$$

where

$$b(n) = 2h\left(\frac{M}{2} - n\right), \quad n = 1, 2, \ldots, \frac{M}{2} \tag{7.10}$$

Hence

$$H_r(\omega) = \sum_{n=1}^{M/2} b(n) \cos\left\{\omega\left(n - \frac{1}{2}\right)\right\} \tag{7.11}$$

Note: At $\omega = \pi$ we get

$$H_r(\pi) = \sum_{n=1}^{M/2} b(n) \cos\left\{\pi\left(n - \frac{1}{2}\right)\right\} = 0$$

regardless of $b(n)$ or $h(n)$. Hence we cannot use this type (i.e., symmetric $h(n)$, M even) for highpass or bandstop filters.

Type-3 linear-phase FIR filter: Antisymmetric impulse response, M odd In this case $\beta = \pi/2$, $\alpha = (M-1)/2$ is an integer, $h(n) = -h(M-1-n), \quad 0 \le n \le M-1$, and $h((M-1)/2) = 0$. Then we can show (see Problem P7.4) that

$$H(e^{j\omega}) = \left[\sum_{n=1}^{(M-1)/2} c(n) \sin \omega n\right] e^{j\left[\frac{\pi}{2} - \left(\frac{M-1}{2}\right)\omega\right]} \tag{7.12}$$

where

$$c(n) = 2h\left(\frac{M-1}{2} - n\right), \quad n = 1, 2, \ldots, \frac{M-1}{2} \tag{7.13}$$

and

$$H_r(\omega) = \sum_{n=1}^{(M-1)/2} c(n) \sin \omega n \tag{7.14}$$

Note: At $\omega = 0$ and $\omega = \pi$ we have $H_r(\omega) = 0$, regardless of $c(n)$ or $h(n)$. Furthermore, $e^{j\pi/2} = j$, which means that $jH_r(\omega)$ is purely imaginary. Hence this type of filter is not suitable for designing a lowpass filter or a highpass filter. However, this behavior is suitable for approximating ideal digital Hilbert transformers and differentiators. An ideal Hilbert transformer [23] is an all-pass filter that imparts a 90° phase shift on the input signal. It is frequently used in communication systems for modulation purposes. Differentiators are used in many analog and digital systems to take the derivative of a signal.

Type-4 linear-phase FIR filter: Antisymmetric impulse response, M even This case is similar to Type-2. We have (see Problem P7.5)

$$H(e^{j\omega}) = \left[\sum_{n=1}^{M/2} d(n)\sin\left\{\omega\left(n-\frac{1}{2}\right)\right\}\right] e^{j[\frac{\pi}{2}-\omega(M-1)/2]} \tag{7.15}$$

where

$$d(n) = 2h\left(\frac{M}{2}-n\right), \quad n = 1, 2, \ldots, \frac{M}{2} \tag{7.16}$$

and

$$H_r(\omega) = \sum_{n=1}^{M/2} d(n)\sin\left\{\omega\left(n-\frac{1}{2}\right)\right\} \tag{7.17}$$

Note: At $\omega = 0$, $H_r(0) = 0$ and $e^{j\pi/2} = j$. Hence this type is also suitable for designing digital Hilbert transformers and differentiators.

7.2.3 MATLAB IMPLEMENTATION

The MATLAB function `freqz` computes the frequency response from which we can determine the magnitude response but not the amplitude response. The SP toolbox now provides the function `zerophase` that can compute the amplitude response. However, it easy to write simple functions to compute amplitude responses for each of the four types. We provide four functions to do this.

1. `Hr_type1`:

```
function [Hr,w,a,L] = Hr_Type1(h);
% Computes Amplitude response Hr(w) of a Type-1 LP FIR filter
% -----------------------------------------------------------
% [Hr,w,a,L] = Hr_Type1(h)
% Hr = Amplitude Response
%  w = 500 frequencies between [0 pi] over which Hr is computed
%  a = Type-1 LP filter coefficients
%  L = Order of Hr
%  h = Type-1 LP filter impulse response
%
 M = length(h);  L = (M-1)/2;
 a = [h(L+1) 2*h(L:-1:1)]; % 1x(L+1) row vector
 n = [0:1:L];              % (L+1)x1 column vector
 w = [0:1:500]'*pi/500;  Hr = cos(w*n)*a';
```

2. Hr_type2:

```
function [Hr,w,b,L] = Hr_Type2(h);
% Computes Amplitude response of a Type-2 LP FIR filter
% -------------------------------------------------------
% [Hr,w,b,L] = Hr_Type2(h)
% Hr = Amplitude Response
%  w = frequencies between [0 pi] over which Hr is computed
%  b = Type-2 LP filter coefficients
%  L = Order of Hr
%  h = Type-2 LP impulse response
%
 M = length(h);  L = M/2;
 b = 2*[h(L:-1:1)];  n = [1:1:L]; n = n-0.5;
 w = [0:1:500]'*pi/500;  Hr = cos(w*n)*b';
```

3. Hr_type3:

```
function [Hr,w,c,L] = Hr_Type3(h);
% Computes Amplitude response Hr(w) of a Type-3 LP FIR filter
% -------------------------------------------------------------
% [Hr,w,c,L] = Hr_Type3(h)
% Hr = Amplitude Response
%  w = frequencies between [0 pi] over which Hr is computed
%  c = Type-3 LP filter coefficients
%  L = Order of Hr
%  h = Type-3 LP impulse response
%
 M = length(h);  L = (M-1)/2;
 c = [2*h(L+1:-1:1)];  n = [0:1:L];
 w = [0:1:500]'*pi/500;  Hr = sin(w*n)*c';
```

4. Hr_type4:

```
function [Hr,w,d,L] = Hr_Type4(h);
% Computes Amplitude response of a Type-4 LP FIR filter
% -------------------------------------------------------
% [Hr,w,d,L] = Hr_Type4(h)
% Hr = Amplitude Response
%  w = frequencies between [0 pi] over which Hr is computed
%  d = Type-4 LP filter coefficients
%  L = Order of d
%  h = Type-4 LP impulse response
%
 M = length(h);  L = M/2;
 d = 2*[h(L:-1:1)];  n = [1:1:L]; n = n-0.5;
 w = [0:1:500]'*pi/500;  Hr = sin(w*n)*d';
```

These four functions can be combined into one function, `called ampl-res, that` can be written to determine the type of the linear-phase filter and to implement the appropriate amplitude response expression. This is explored in Problem P7.6. The use of these functions is described in Examples 7.4 through 7.7.

The `zerophase` function from the SP toolbox is similar in use to the `freqz` function. The invocation `[Hr,w, phi] = zerophase(b,a)` returns the amplitude response in `Hr`, evaluated at 512 values around the top half of the unit circle in the array `w` and the continuous phase response in `phi`. Thus, this function can be used for both FIR and IIR filters. Other invocations are also available.

7.2.4 ZERO LOCATIONS

Recall that for an FIR filter there are $(M-1)$ (trivial) poles at the origin and $(M-1)$ zeros located somewhere in the z-plane. For linear-phase FIR filters, these zeros possess certain symmetries that are due to the symmetry constraints on $h(n)$. It can be shown (see [23] and Problem P7.7) that if $H(z)$ has a zero at

$$z = z_1 = re^{j\theta}$$

then for linear phase there must be a zero at

$$z = \frac{1}{z_1} = \frac{1}{r}e^{-j\theta}$$

For a real-valued filter we also know that if z_1 is complex, then there must be a conjugate zero at $z_1^* = re^{-j\theta}$, which implies that there must be a zero at $1/z_1^* = (1/r)\,e^{j\theta}$. Thus a general *zero constellation* is a quadruplet

$$re^{j\theta} \qquad \frac{1}{r}e^{j\theta} \qquad re^{-j\theta} \qquad \text{and} \qquad \frac{1}{r}e^{-j\theta}$$

as shown in Figure 7.3. Clearly, if $r = 1$, then $1/r = 1$, and hence the zeros are on the unit circle and occur in pairs

$$e^{j\theta} \qquad \text{and} \qquad e^{-j\theta}$$

If $\theta = 0$ or $\theta = \pi$, then the zeros are on the real line and occur in pairs

$$r \qquad \text{and} \qquad \frac{1}{r}$$

Finally, if $r = 1$ and $\theta = 0$ or $\theta = \pi$, the zeros are either at $z = 1$ or $z = -1$. These symmetries can be used to implement cascade forms with linear-phase sections.

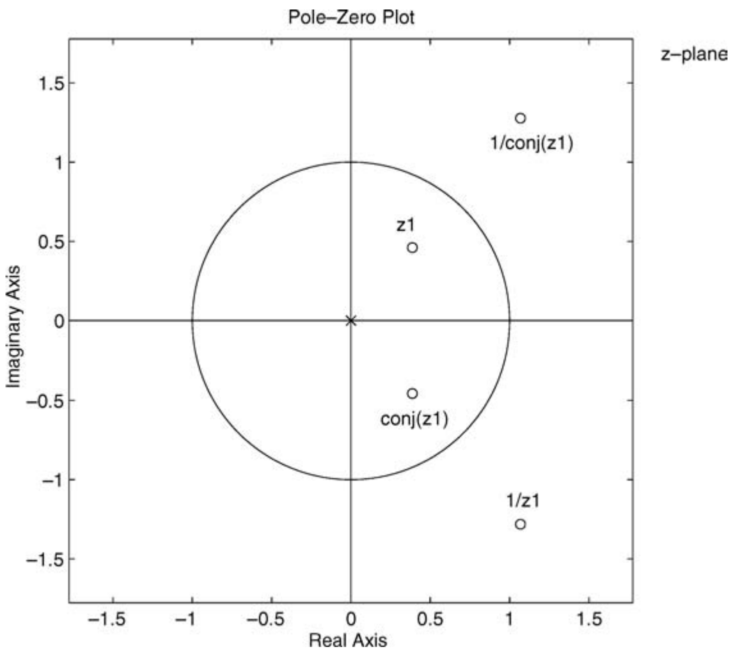


FIGURE 7.3 *A general zero constellation*

In the following examples, we illustrate the preceding properties of linear-phase FIR filters.

☐ **EXAMPLE 7.4** Let $h(n) = \{-4, 1, -1, -2, \underset{\uparrow}{5}, 6, 5, -2, -1, 1, -4\}$. Determine the amplitude response $H_r(\omega)$ and the locations of the zeros of $H(z)$.

Solution Since $M = 11$, which is odd, and since $h(n)$ is symmetric about $\alpha = (11-1)/2 = 5$, this is a Type-1 linear-phase FIR filter. From (7.7) we have

$$a(0) = h(\alpha) = h(5) = 6,\ a(1) = 2h(5-1) = 10,\ a(2) = 2h(5-2) = -4$$
$$a(3) = 2h(5-3) = -2,\ a(4) = 2h(5-4) = 2,\ a(5) = 2h(5-5) = -8$$

From (7.8), we obtain

$$\begin{aligned} H_r(\omega) &= a(0) + a(1)\cos\omega + a(2)\cos 2\omega + a(3)\cos 3\omega + a(4)\cos 4\omega + a(5)\cos 5\omega \\ &= 6 + 10\cos\omega - 4\cos 2\omega - 2\cos 3\omega + 2\cos 4\omega - 8\cos 5\omega \end{aligned}$$

MATLAB script:

```
>> h = [-4,1,-1,-2,5,6,5,-2,-1,1,-4];
>> M = length(h); n = 0:M-1;
>> [Hr,w,a,L] = Hr_Type1(h);
>> a,L
a = 6    10    -4    -2    2    -8
L = 5
>> amax = max(a)+1; amin = min(a)-1;
>> subplot(2,2,1); stem(n,h); axis([-1 2*L+1 amin amax])
>> xlabel('n'); ylabel('h(n)'); title('Impulse Response')
>> subplot(2,2,3); stem(0:L,a); axis([-1 2*L+1 amin amax])
>> xlabel('n'); ylabel('a(n)'); title('a(n) coefficients')
>> subplot(2,2,2); plot(w/pi,Hr);grid
>> xlabel('frequency in pi units'); ylabel('Hr')
>> title('Type-1 Amplitude Response')
>> subplot(2,2,4); pzplotz(h,1)
```

The plots and the zero locations are shown in Figure 7.4. From these plots, we observe that there are no restrictions on $H_r(\omega)$ either at $\omega = 0$ or at $\omega = \pi$. There is one zero-quadruplet constellation and three zero pairs. □

□ **EXAMPLE 7.5** Let $h(n) = \{\underset{\uparrow}{-4}, 1, -1, -2, 5, 6, 6, 5, -2, -1, 1, -4\}$. Determine the amplitude response $H_r(\omega)$ and the locations of the zeros of $H(z)$.

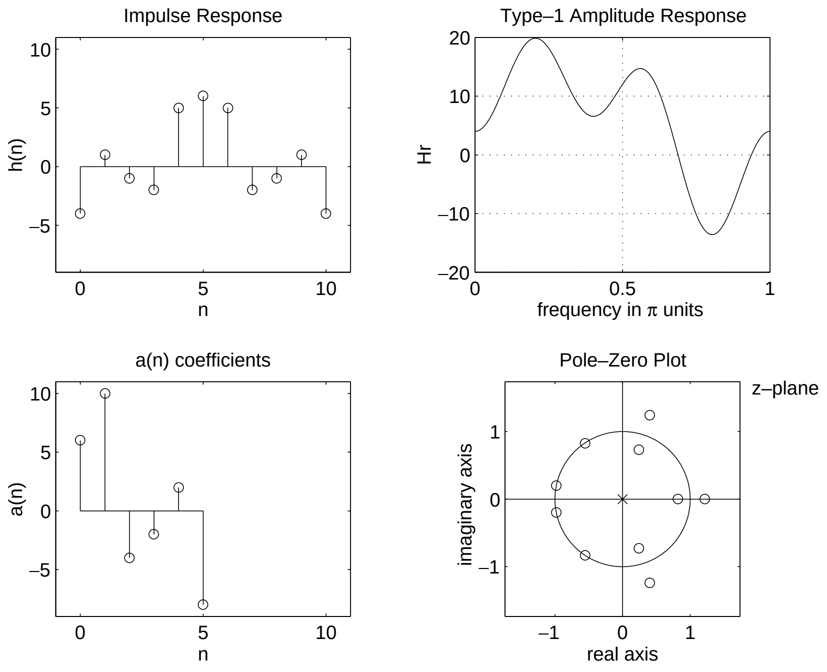


FIGURE 7.4 *Plots in Example 7.4*

Solution This is a Type-2 linear-phase FIR filter since $M = 12$ and since $h(n)$ is symmetric with respect to $\alpha = (12-1)/2 = 5.5$. From (7.10) we have

$$b(1) = 2h\left(\tfrac{12}{2} - 1\right) = 12,\ \ b(2) = 2h\left(\tfrac{12}{2} - 2\right) = 10,\ b(3) = 2h\left(\tfrac{12}{2} - 3\right) = -4$$
$$b(4) = 2h\left(\tfrac{12}{2} - 4\right) = -2,\ b(5) = 2h\left(\tfrac{12}{2} - 5\right) = 2,\ \ b(6) = 2h\left(\tfrac{12}{2} - 6\right) = -8$$

Hence from (7.11) we obtain

$$\begin{aligned} H_r(\omega) &= b(1)\cos\left[\omega\left(1-\tfrac{1}{2}\right)\right] + b(2)\cos\left[\omega\left(2-\tfrac{1}{2}\right)\right] + b(3)\cos\left[\omega\left(3-\tfrac{1}{2}\right)\right] \\ &\quad + b(4)\cos\left[\omega\left(4-\tfrac{1}{2}\right)\right] + b(5)\cos\left[\omega\left(5-\tfrac{1}{2}\right)\right] + b(6)\cos\left[\omega\left(6-\tfrac{1}{2}\right)\right] \\ &= 12\cos\left(\frac{\omega}{2}\right) + 10\cos\left(\frac{3\omega}{2}\right) - 4\cos\left(\frac{5\omega}{2}\right) - 2\cos\left(\frac{7\omega}{2}\right) \\ &\quad + 2\cos\left(\frac{9\omega}{2}\right) - 8\cos\left(\frac{11\omega}{2}\right) \end{aligned}$$

MATLAB script:

```
>> h = [-4,1,-1,-2,5,6,6,5,-2,-1,1,-4];
>> M = length(h); n = 0:M-1;  [Hr,w,a,L] = Hr_Type2(h);
>> b,L
b = 12    10    -4    -2    2    -8
L = 6
>> bmax = max(b)+1; bmin = min(b)-1;
>> subplot(2,2,1); stem(n,h); axis([-1 2*L+1 bmin bmax])
>> xlabel('n'); ylabel('h(n)'); title('Impulse Response')
>> subplot(2,2,3); stem(1:L,b); axis([-1 2*L+1 bmin bmax])
>> xlabel('n'); ylabel('b(n)'); title('b(n) coefficients')
>> subplot(2,2,2); plot(w/pi,Hr);grid
>> xlabel('frequency in pi units'); ylabel('Hr')
>> title('Type-1 Amplitude Response')
>> subplot(2,2,4); pzplotz(h,1)
```

The plots and the zero locations are shown in Figure 7.5. From these plots, we observe that $H_r(\omega)$ is zero at $\omega = \pi$. There is one zero-quadruplet constellation, three zero pairs, and one zero at $\omega = \pi$ as expected. □

□ **EXAMPLE 7.6** Let $h(n) = \{\underset{\uparrow}{-4}, 1, -1, -2, 5, 0, -5, 2, 1, -1, 4\}$. Determine the amplitude response $H_r(\omega)$ and the locations of the zeros of $H(z)$.

Solution Since $M = 11$, which is odd, and since $h(n)$ is antisymmetric about $\alpha = (11-1)/2 = 5$, this is a Type-3 linear-phase FIR filter. From (7.13) we have

$$c(0) = h(\alpha) = h(5) = 0,\ c(1) = 2h(5-1) = 10,\ c(2) = 2h(2-2) = -4$$
$$c(3) = 2h(5-3) = -2,\ \ c(4) = 2h(5-4) = 2,\ \ c(5) = 2h(5-5) = -8$$

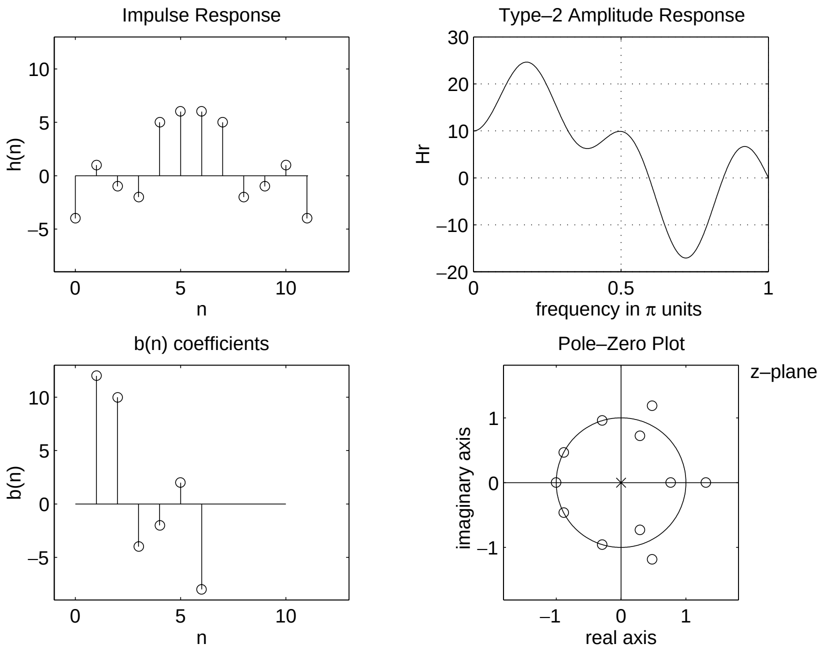


FIGURE 7.5 *Plots in Example 7.5*

From (7.14) we obtain

$$\begin{aligned} H_r(\omega) &= c(0) + c(1)\sin\omega + c(2)\sin 2\omega + c(3)\sin 3\omega + c(4)\sin 4\omega + c(5)\sin 5\omega \\ &= 0 + 10\sin\omega - 4\sin 2\omega - 2\sin 3\omega + 2\sin 4\omega - 8\sin 5\omega \end{aligned}$$

MATLAB script:

```
>> h = [-4,1,-1,-2,5,0,-5,2,1,-1,4];
>> M = length(h); n = 0:M-1; [Hr,w,c,L] = Hr_Type3(h);
>> c,L
a = 0    10    -4    -2    2    -8
L = 5
>> cmax = max(c)+1; cmin = min(c)-1;
>> subplot(2,2,1); stem(n,h); axis([-1 2*L+1 cmin cmax])
>> xlabel('n'); ylabel('h(n)'); title('Impulse Response')
>> subplot(2,2,3); stem(0:L,c); axis([-1 2*L+1 cmin cmax])
>> xlabel('n'); ylabel('c(n)'); title('c(n) coefficients')
>> subplot(2,2,2); plot(w/pi,Hr);grid
>> xlabel('frequency in pi units'); ylabel('Hr')
>> title('Type-1 Amplitude Response')
>> subplot(2,2,4); pzplotz(h,1)
```

The plots and the zero locations are shown in Figure 7.6. From these plots, we observe that $H_r(\omega) = 0$ at $\omega = 0$ and at $\omega = \pi$. There is one zero-quadruplet constellation, two zero pairs, and zeros at $\omega = 0$ and $\omega = \pi$ as expected. □

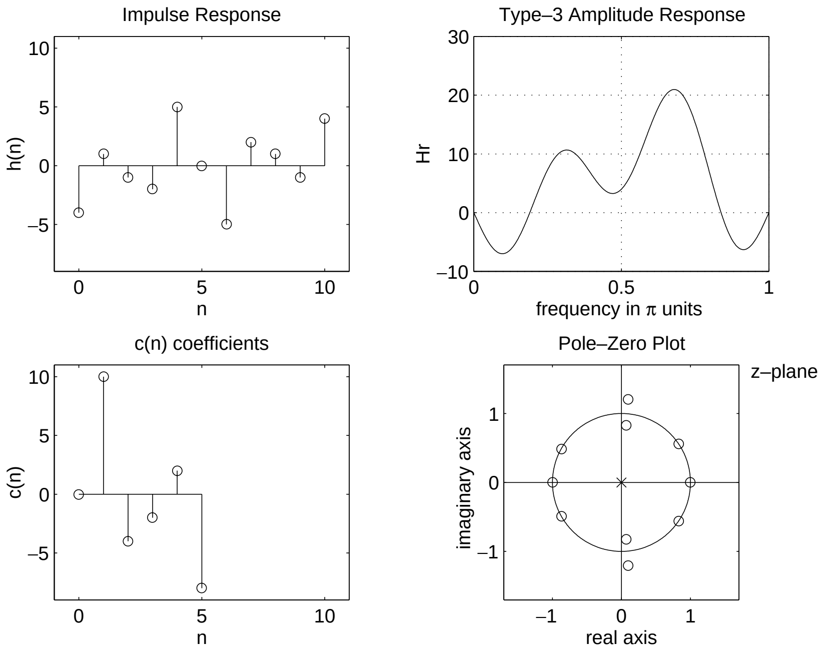


FIGURE 7.6 *Plots in Example 7.6*

□ **EXAMPLE 7.7** Let $h(n) = \{-4, \underset{\uparrow}{1}, -1, -2, 5, 6, -6, -5, 2, 1, -1, 4\}$. Determine the amplitude response $H_r(\omega)$ and the locations of the zeros of $H(z)$.

Solution This is a Type-4 linear-phase FIR filter since $M = 12$ and since $h(n)$ is antisymmetric with respect to $\alpha = (12-1)/2 = 5.5$. From (7.16) we have

$$d(1) = 2h\left(\tfrac{12}{2}-1\right) = 12,\ d(2) = 2h\left(\tfrac{12}{2}-2\right) = 10,\ d(3) = 2h\left(\tfrac{12}{2}-3\right) = -4$$
$$d(4) = 2h\left(\tfrac{12}{2}-4\right) = -2,\ d(5) = 2h\left(\tfrac{12}{2}-5\right) = 2,\ d(6) = 2h\left(\tfrac{12}{2}-6\right) = -8$$

Hence from (7.17) we obtain

$$\begin{aligned} H_r(\omega) &= d(1)\sin\left[\omega\left(1-\frac{1}{2}\right)\right] + d(2)\sin\left[\omega\left(2-\frac{1}{2}\right)\right] + d(3)\sin\left[\omega\left(3-\frac{1}{2}\right)\right] \\ &\quad + d(4)\sin\left[\omega\left(4-\frac{1}{2}\right)\right] + d(5)\sin\left[\omega\left(5-\frac{1}{2}\right)\right] + d(6)\sin\left[\omega\left(6-\frac{1}{2}\right)\right] \\ &= 12\sin\left(\frac{\omega}{2}\right) + 10\sin\left(\frac{3\omega}{2}\right) - 4\sin\left(\frac{5\omega}{2}\right) - 2\sin\left(\frac{7\omega}{2}\right) \\ &\quad + 2\sin\left(\frac{9\omega}{2}\right) - 8\sin\left(\frac{11\omega}{2}\right) \end{aligned}$$

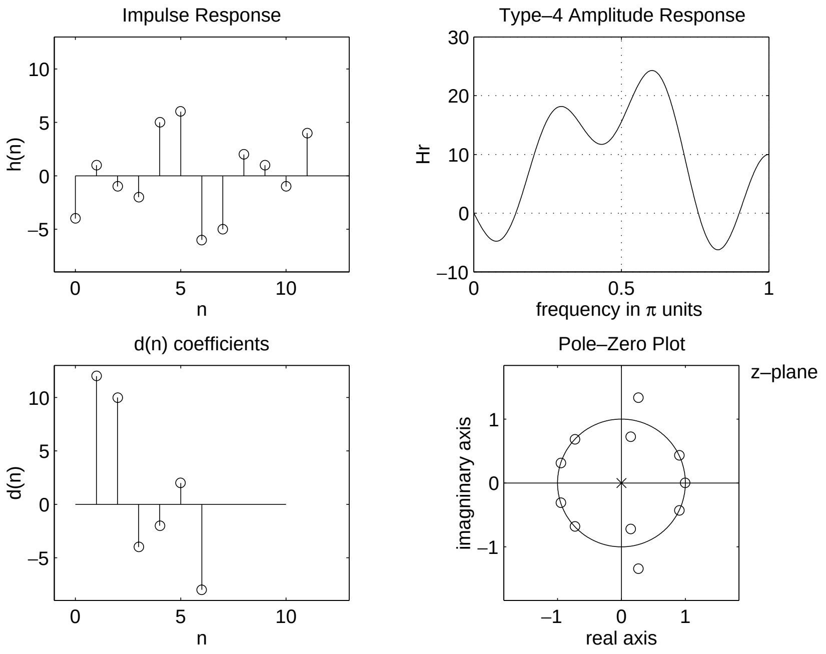


FIGURE 7.7 *Plots in Example 7.7*

MATLAB script:

```
>> h = [-4,1,-1,-2,5,6,-6,-5,2,1,-1,4];
>> M = length(h); n = 0:M-1;  [Hr,w,d,L] = Hr_Type4(h);
>> b,L
d = 12    10    -4    -2    2    -8
L = 6
>> dmax = max(d)+1; dmin = min(d)-1;
>> subplot(2,2,1); stem(n,h); axis([-1 2*L+1 dmin dmax])
>> xlabel('n'); ylabel('h(n)'); title('Impulse Response')
>> subplot(2,2,3); stem(1:L,d); axis([-1 2*L+1 dmin dmax])
>> xlabel('n'); ylabel('d(n)'); title('d(n) coefficients')
>> subplot(2,2,2); plot(w/pi,Hr);grid
>> xlabel('frequency in pi units'); ylabel('Hr')
>> title('Type-1 Amplitude Response')
>> subplot(2,2,4); pzplotz(h,1)
```

The plots and the zero locations are shown in Figure 7.7. From these plots, we observe that $H_r(\omega)$ is zero at $\omega = 0$. There is one zero-quadruplet constellation, three zero pairs, and one zero at $\omega = 0$ as expected. □

7.3 WINDOW DESIGN TECHNIQUES

The basic idea behind the window design is to choose a proper ideal frequency-selective filter (which always has a noncausal, infinite-duration impulse response) and then to truncate (or window) its impulse response to obtain a linear-phase and causal FIR filter. Therefore the emphasis in this method is on selecting an appropriate *windowing* function and an appropriate *ideal* filter. We will denote an ideal frequency-selective filter by $H_d(e^{j\omega})$, which has a unity magnitude gain and linear-phase characteristics over its passband, and zero response over its stopband. An ideal LPF of bandwidth $\omega_c < \pi$ is given by

$$H_d(e^{j\omega}) = \begin{cases} 1 \cdot e^{-j\alpha\omega}, & |\omega| \le \omega_c \\ 0, & \omega_c < |\omega| \le \pi \end{cases} \tag{7.18}$$

where ω_c is also called the *cutoff* frequency, and α is called the sample *delay*. (Note that from the DTFT properties, $e^{-j\alpha\omega}$ implies shift in the positive n direction or delay.) The impulse response of this filter is of infinite duration and is given by

$$\begin{aligned} h_d(n) &= \mathcal{F}^{-1}\left[H_d(e^{j\omega})\right] = \frac{1}{2\pi}\int_{-\pi}^{\pi} H_d(e^{j\omega})e^{j\omega n}d\omega \\ &= \frac{1}{2\pi}\int_{-\omega_c}^{\omega_c} 1\cdot e^{-j\alpha\omega}e^{j\omega n}d\omega \\ &= \frac{\sin\left[\omega_c(n-\alpha)\right]}{\pi(n-\alpha)} \end{aligned} \tag{7.19}$$

Note that $h_d(n)$ is symmetric with respect to α, a fact useful for linear-phase FIR filters.

To obtain an FIR filter from $h_d(n)$, one has to truncate $h_d(n)$ on both sides. To obtain a causal and linear-phase FIR filter $h(n)$ of length M, we must have

$$h(n) = \begin{cases} h_d(n), & 0 \le n \le M-1 \\ 0, & \text{elsewhere} \end{cases} \quad \text{and} \quad \alpha = \frac{M-1}{2} \tag{7.20}$$

This operation is called "windowing." In general, $h(n)$ can be thought of as being formed by the product of $h_d(n)$ and a window function $w(n)$ as follows:

$$h(n) = h_d(n)w(n) \tag{7.21}$$

where

$$w(n) = \begin{cases} \text{some symmetric function with respect to} \\ \alpha \text{ over } 0 \le n \le M-1 \\ 0, \text{ otherwise} \end{cases}$$

Depending on how we define $w(n)$, we obtain different window designs. For example, in (7.20)

$$w(n) = \begin{cases} 1, \ 0 \le n \le M-1 \\ 0, \quad \text{otherwise} \end{cases} = \mathcal{R}_M(n)$$

which is the *rectangular window* defined earlier.

In the frequency domain the causal FIR filter response $H(e^{j\omega})$ is given by the periodic convolution of $H_d(e^{j\omega})$ and the window response $W(e^{j\omega})$; that is,

$$H(e^{j\omega}) = H_d(e^{j\omega}) \circledast W(e^{j\omega}) = \frac{1}{2\pi}\int_{-\pi}^{\pi} W\left(e^{j\lambda}\right) H_d\left(e^{j(\omega-\lambda)}\right) d\lambda \tag{7.22}$$

This is shown pictorially in Figure 7.8 for a typical window response, from which we have the following observations:

1. Since the window $w(n)$ has a finite length equal to M, its response has a *peaky main lobe* whose width is proportional to $1/M$, and has side lobes of smaller heights.

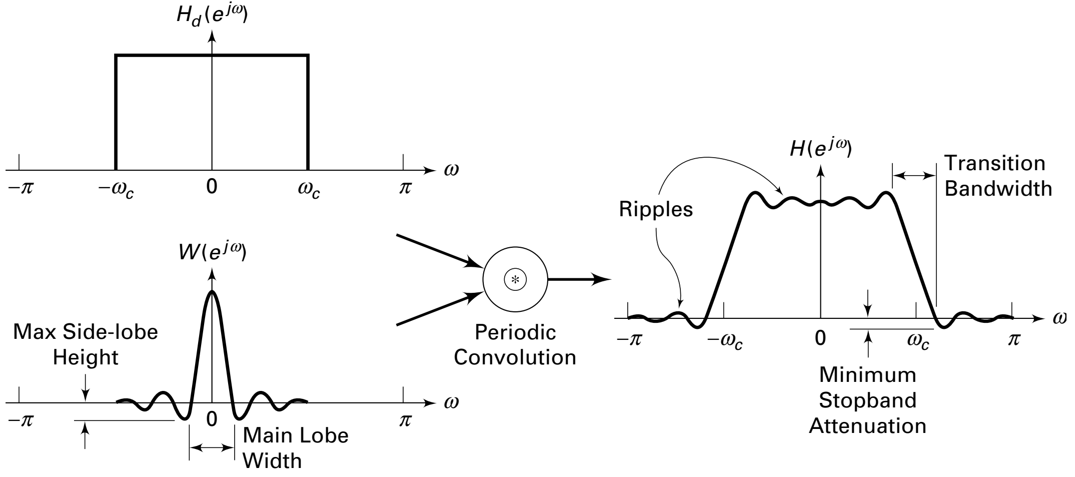


FIGURE 7.8 *Windowing operation in the frequency domain*

2. The periodic convolution (7.22) produces a smeared version of the ideal response $H_d(e^{j\omega})$.
3. The main lobe produces a transition band in $H(e^{j\omega})$ whose width is responsible for the transition width. This width is then proportional to $1/M$. The wider the main lobe, the wider will be the transition width.
4. The side lobes produce ripples that have similar shapes in both the passband and stopband.

Basic window design idea For the given filter specifications, choose the filter length M and a window function $w(n)$ for the narrowest main lobe width and the smallest side lobe attenuation possible.

From observation 4, we note that the passband tolerance δ_1 and the stopband tolerance δ_2 cannot be specified independently. We generally take care of δ_2 alone, which results in $\delta_2 = \delta_1$. We now briefly describe various well-known window functions. We will use the rectangular window as an example to study their performances in the frequency domain.

7.3.1 RECTANGULAR WINDOW

This is the simplest window function but provides the worst performance from the viewpoint of stopband attenuation. It was defined earlier by

$$w(n) = \begin{cases} 1, & 0 \le n \le M-1 \\ 0, & \text{otherwise} \end{cases} \tag{7.23}$$

Its frequency response function is

$$W(e^{j\omega}) = \left[\frac{\sin\left(\frac{\omega M}{2}\right)}{\sin\left(\frac{\omega}{2}\right)}\right] e^{-j\omega\frac{M-1}{2}} \Rightarrow W_r(\omega) = \frac{\sin\left(\frac{\omega M}{2}\right)}{\sin\left(\frac{\omega}{2}\right)}$$

which is the amplitude response. From (7.22) the actual amplitude response $H_r(\omega)$ is given by

$$H_r(\omega) \simeq \frac{1}{2\pi}\int_{-\pi}^{\omega+\omega_c} W_r(\lambda)\, d\lambda = \frac{1}{2\pi}\int_{-\pi}^{\omega+\omega_c} \frac{\sin\left(\frac{\omega M}{2}\right)}{\sin\left(\frac{\omega}{2}\right)} d\lambda, \quad M \gg 1 \tag{7.24}$$

This implies that the running integral of the window amplitude response (or *accumulated* amplitude response) is necessary in the accurate analysis of the transition bandwidth and the stopband attenuation. Figure 7.9 shows the rectangular window function $w(n)$, its amplitude response $W(\omega)$, the amplitude response in dB, and the accumulated amplitude response (7.24) in dB. From the observation of plots in Figure 7.9, we can make several observations.

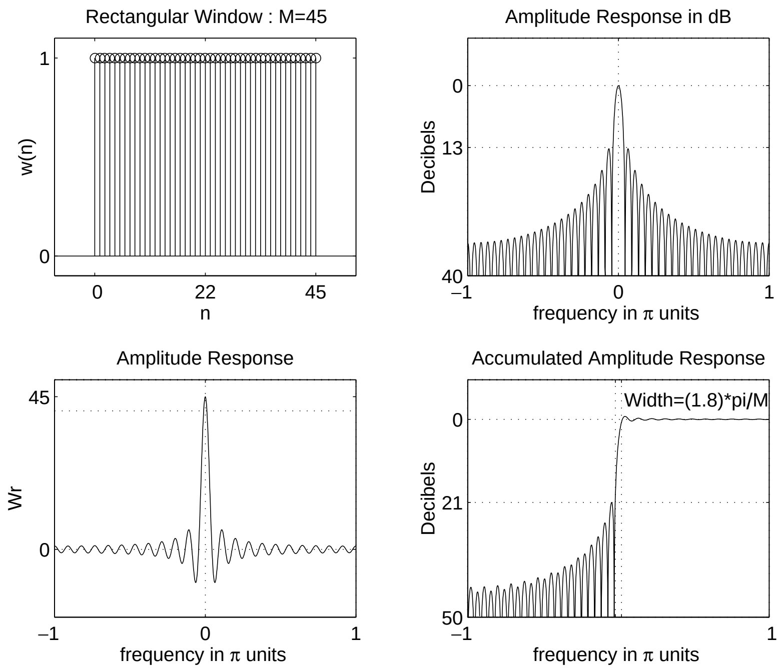


FIGURE 7.9 *Rectangular window: $M = 45$*

1. The amplitude response $W_r(\omega)$ has the first zero at $\omega = \omega_1$, where

$$\frac{\omega_1 M}{2} = \pi \qquad \text{or} \qquad \omega_1 = \frac{2\pi}{M}$$

   Hence the width of the main lobe is $2\omega_1 = 4\pi/M$. Therefore the *approximate transition bandwidth* is $4\pi/M$.

2. The magnitude of the first side lobe (which is also the peak side lobe magnitude) is approximately at $\omega = 3\pi/M$ and is given by

$$\left|W_r\left(\omega = \frac{3\pi}{M}\right)\right| = \left|\frac{\sin\left(\frac{3\pi}{2}\right)}{\sin\left(\frac{3\pi}{2M}\right)}\right| \simeq \frac{2M}{3\pi} \qquad \text{for } M \gg 1$$

   Comparing this with the main lobe amplitude, which is equal to M, the *peak side lobe magnitude* is

$$\frac{2}{3\pi} = 21.22\% \equiv 13 \text{ dB}$$

   of the main lobe amplitude.

3. The accumulated amplitude response has the first side lobe magnitude at 21 dB. This results in the *minimum stopband attenuation* of 21 dB irrespective of the window length M.

4. Using the minimum stopband attenuation, the transition bandwidth can be accurately computed. It is shown in the accumulated amplitude response plot in Figure 7.9. This computed *exact transition bandwidth* is

$$\omega_s - \omega_p = \frac{1.8\pi}{M}$$

which is less than half the approximate bandwidth of $4\pi/M$.

Clearly, this is a simple window operation in the time domain and an easy function to analyze in the frequency domain. However, there are two main problems. First, the minimum stopband attenuation of 21 dB is insufficient in practical applications. Second, the rectangular windowing being a direct truncation of the infinite length $h_d(n)$, it suffers from the *Gibbs phenomenon*. If we increase M, the width of each side lobe will decrease, but the area under each lobe will remain constant. Therefore, the *relative amplitudes* of side lobes will remain constant, and the minimum stopband attenuation will remain at 21 dB. This implies that all ripples will bunch up near the band edges. It is shown in Figure 7.10.

Since the rectangular window is impractical in many applications, we consider other fixed window functions that provide a fixed amount

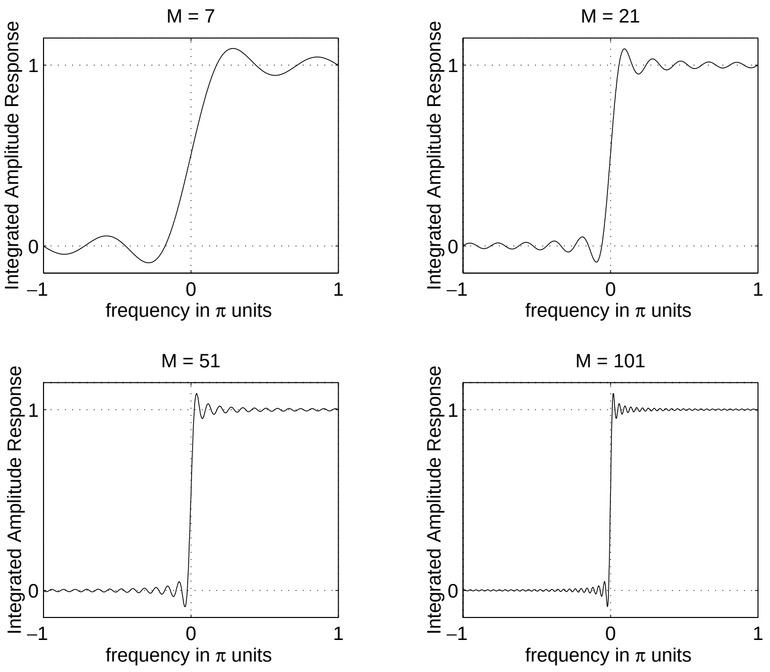


FIGURE 7.10 *Gibbs phenomenon*

of attenuation. These window functions bear the names of the people who first proposed them. Although these window functions can also be analyzed similar to the rectangular window, we present only their results.

7.3.2 BARTLETT WINDOW

Since the Gibbs phenomenon results from the fact that the rectangular window has a sudden transition from 0 to 1 (or 1 to 0), Bartlett suggested a more gradual transition in the form of a triangular window, which is given by

$$w(n) = \begin{cases} \dfrac{2n}{M-1}, & 0 \le n \le \dfrac{M-1}{2} \\ 2 - \dfrac{2n}{M-1}, & \dfrac{M-1}{2} \le n \le M-1 \\ 0, & \text{otherwise} \end{cases} \tag{7.25}$$

This window and its frequency-domain responses are shown in Figure 7.11.

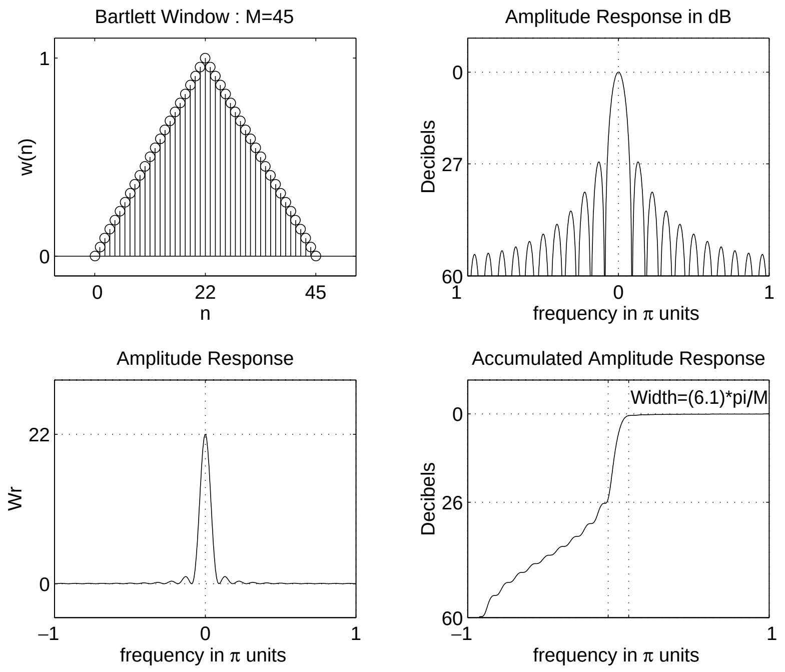


FIGURE 7.11 *Bartlett window:* $M = 45$

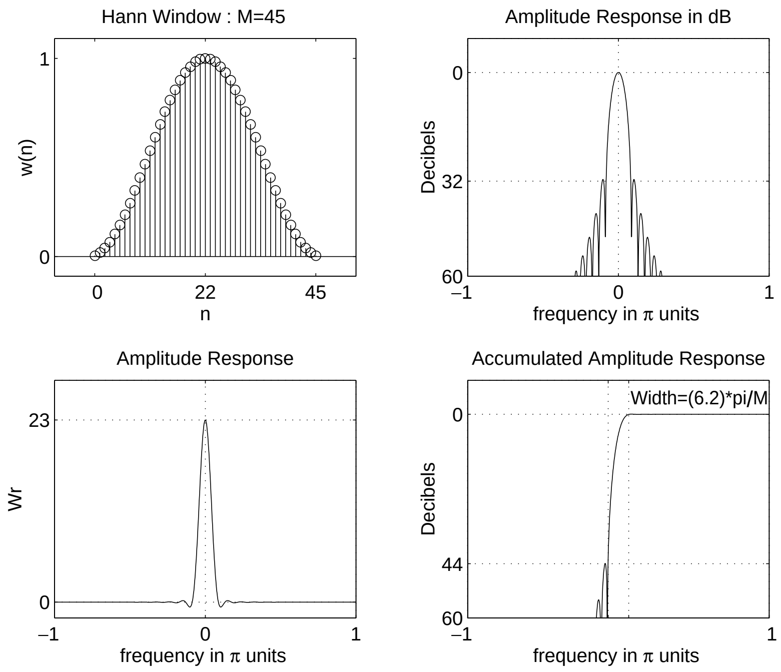


FIGURE 7.12 *Hann window: $M = 45$*

7.3.3 HANN WINDOW

This is a raised cosine window function given by

$$w(n) = \begin{cases} 0.5\left[1 - \cos\left(\frac{2\pi n}{M-1}\right)\right], & 0 \le n \le M-1 \\ 0, & \text{otherwise} \end{cases} \tag{7.26}$$

This window and its frequency-domain responses are shown in Figure 7.12.

7.3.4 HAMMING WINDOW

This window is similar to the Hann window except that it has a small amount of discontinuity and is given by

$$w(n) = \begin{cases} 0.54 - 0.46\cos\left(\frac{2\pi n}{M-1}\right), & 0 \le n \le M-1 \\ 0, & \text{otherwise} \end{cases} \tag{7.27}$$

This window and its frequency-domain responses are shown in Figure 7.13.

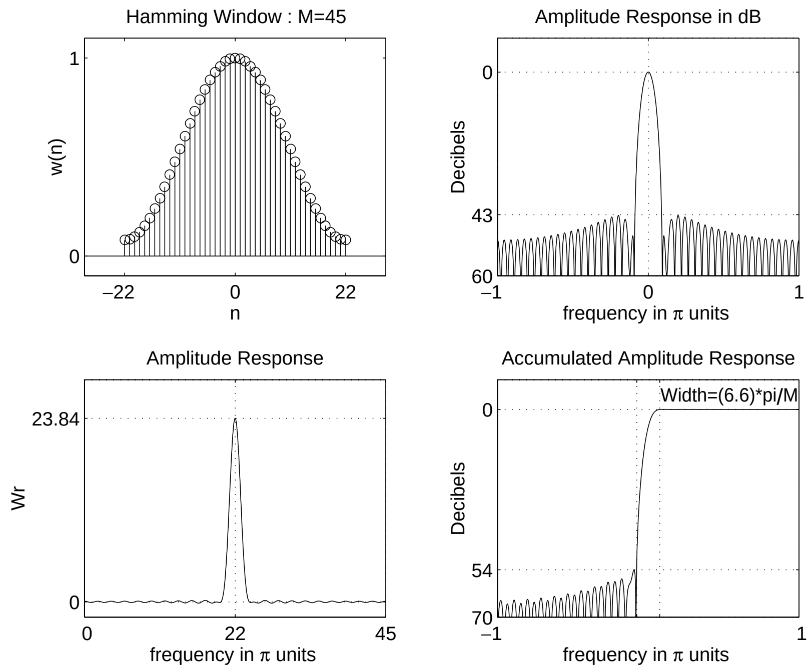


FIGURE 7.13 *Hamming window: $M = 45$*

7.3.5 BLACKMAN WINDOW

This window is also similar to the previous two but contains a second harmonic term and is given by

$$w(n) = \begin{cases} 0.42 - 0.5\cos\left(\frac{2\pi n}{M-1}\right) + 0.08\cos\left(\frac{4\pi n}{M-1}\right), & 0 \le n \le M-1 \\ 0, & \text{otherwise} \end{cases} \tag{7.28}$$

This window and its frequency-domain responses are shown in Figure 7.14.

In Table 7.1 we provide a summary of fixed window function characteristics in terms of their transition widths (as a function of M) and their minimum stopband attenuations in dB. Both the approximate as well as the exact transition bandwidths are given. Note that the transition widths and the stopband attenuations increase as we go down the table. The Hamming window appears to be the best choice for many applications.

7.3.6 KAISER WINDOW

This is an adjustable window function that is widely used in practice. The window function is due to J. F. Kaiser and is given by

$$w(n) = \frac{I_0\left[\beta\sqrt{1-(1-\frac{2n}{M-1})^2}\,\right]}{I_0\left[\beta\right]}, \quad 0 \le n \le M-1 \tag{7.29}$$

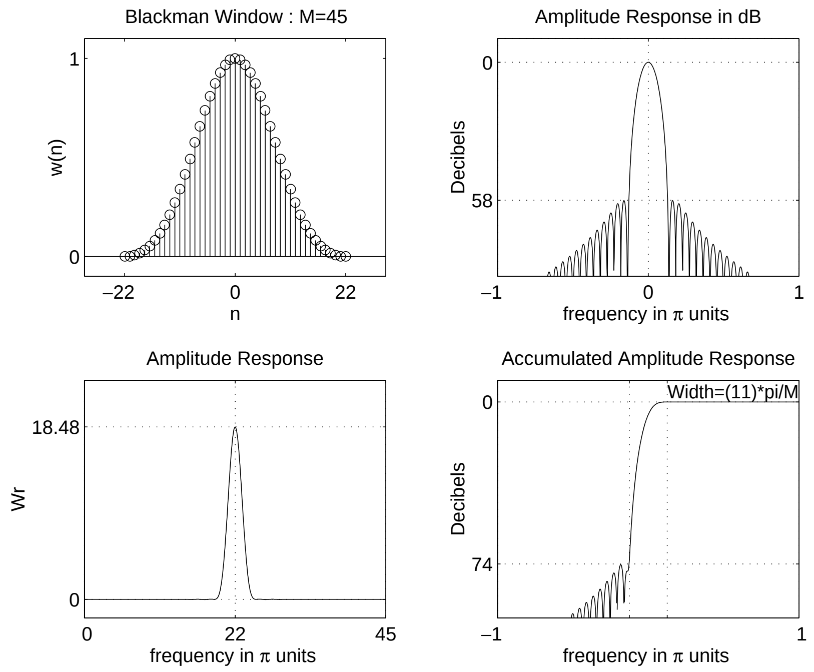


FIGURE 7.14 *Blackman window: $M = 45$*

TABLE 7.1 *Summary of commonly used window function characteristics*

Window Name	*Transition Width $\Delta\omega$* *Approximate*	*Exact Values*	*Min. Stopband Attenuation*
Rectangular	$\frac{4\pi}{M}$	$\frac{1.8\pi}{M}$	21 dB
Bartlett	$\frac{8\pi}{M}$	$\frac{6.1\pi}{M}$	25 dB
Hann	$\frac{8\pi}{M}$	$\frac{6.2\pi}{M}$	44 dB
Hamming	$\frac{8\pi}{M}$	$\frac{6.6\pi}{M}$	53 dB
Blackman	$\frac{12\pi}{M}$	$\frac{11\pi}{M}$	74 dB

where $I_0[\,\cdot\,]$ is the *modified* zero-*order Bessel* function given by

$$I_0(x) = 1 + \sum_{k=0}^{\infty} \left[\frac{(x/2)^k}{k!}\right]^2$$

which is positive for all real values of x. The parameter β controls the minimum stopband attenuation A_s and can be chosen to yield different transition widths for near-optimum A_s. This window can provide different transition widths for the same M, which is something other fixed windows lack. For example,

- if $\beta = 5.658$, then the transition width is equal to $7.8\pi/M$, and the minimum stopband attenuation is equal to 60 dB. This is shown in Figure 7.15.
- if $\beta = 4.538$, then the transition width is equal to $5.8\pi/M$, and the minimum stopband attenuation is equal to 50 dB.

Hence the performance of this window is comparable to that of the Hamming window. In addition, the Kaiser window provides flexible transition bandwidths. Due to the complexity involved in the Bessel functions, the design equations for this window are not easy to derive. Fortunately, Kaiser has developed *empirical* design equations, which we provide here

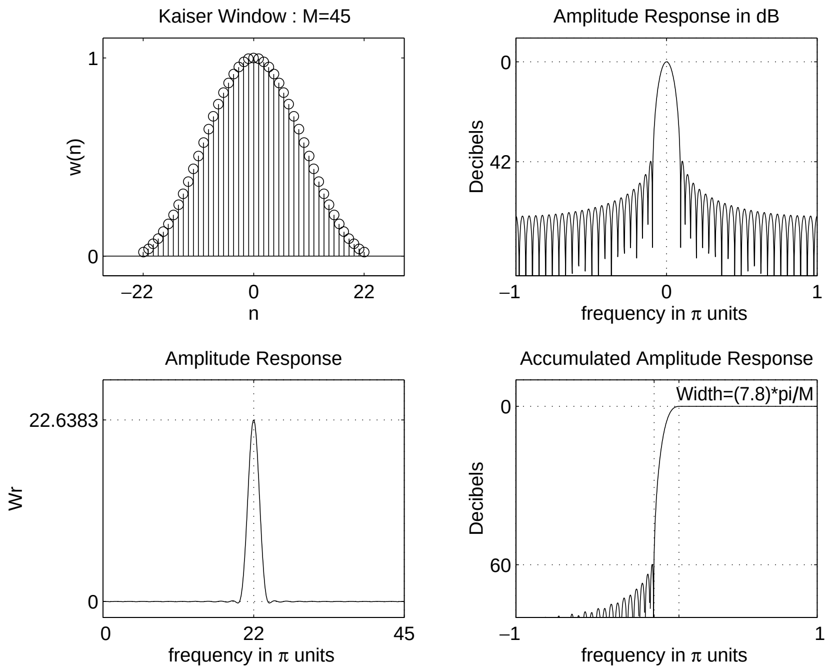


FIGURE 7.15 *Kaiser window:* $M = 45$, $\beta = 5.658$

without proof. Given ω_p, ω_s, R_p, and A_s the parameters M and β are given by

$$\begin{aligned} \text{transition width} &= \Delta\omega = \omega_s - \omega_p \\ \text{Filter length } M &\simeq \frac{A_s - 7.95}{2.285\Delta\omega} + 1 \end{aligned} \tag{7.30}$$

$$\text{Parameter } \beta = \begin{cases} 0.1102(A_s - 8.7), & A_s \geq 50 \\ 0.5842\,(A_s - 21)^{0.4} \\ \quad + 0.07886(A_s - 21), & 21 < A_s < 50 \end{cases}$$

7.3.7 MATLAB IMPLEMENTATION

MATLAB provides several functions to implement window functions discussed in this section. A brief description of these functions follow.

- `w=boxcar(M)` returns the `M`-point rectangular window function in array `w`.
- `w=bartlett(M)` returns the `M`-point Bartlett window function in array `w`.
- `w=hann(M)` returns the `M`-point Hann window function in array `w`.
- `w=hamming(M)` returns the `M`-point Hamming window function in array `w`.
- `w=blackman(M)` returns the `M`-point Blackman window function in array `w`.
- `w=kaiser(M,beta)` returns the `beta`-valued `M`-point rectangular window function in array `w`.

Using these functions, we can use MATLAB to design FIR filters based on the window technique, which also requires an ideal lowpass impulse response $h_d(n)$. Therefore it is convenient to have a simple routine that creates $h_d(n)$ as shown here.

```
function hd = ideal_lp(wc,M);
% Ideal LowPass filter computation
% --------------------------------
% [hd] = ideal_lp(wc,M)
%  hd = ideal impulse response between 0 to M-1
%  wc = cutoff frequency in radians
%   M = length of the ideal filter
%
alpha = (M-1)/2; n = [0:1:(M-1)];
m = n - alpha; fc = wc/pi; hd = fc*sinc(fc*m);
```

To display the frequency-domain plots of digital filters, MATLAB provides the `freqz` function, which we used in earlier chapters. Using this function, we have developed a modified version, called `freqz_m`, which returns the magnitude response in absolute as well as in relative dB scale, the phase response, and the group delay response. We will need the group delay response in the next chapter.

```
function [db,mag,pha,grd,w] = freqz_m(b,a);
% Modified version of freqz subroutine
% ------------------------------------
% [db,mag,pha,grd,w] = freqz_m(b,a);
%  db = Relative magnitude in dB computed over 0 to pi radians
% mag = absolute magnitude computed over 0 to pi radians
% pha = Phase response in radians over 0 to pi radians
% grd = Group delay over 0 to pi radians
%   w = 501 frequency samples between 0 to pi radians
%   b = numerator polynomial of H(z)   (for FIR: b=h)
%   a = denominator polynomial of H(z) (for FIR: a=[1])
%
[H,w] = freqz(b,a,1000,'whole');
    H = (H(1:1:501))'; w = (w(1:1:501))';
  mag = abs(H);  db = 20*log10((mag+eps)/max(mag));
  pha = angle(H);  grd = grpdelay(b,a,w);
```

7.3.8 DESIGN EXAMPLES

We now provide several examples of FIR filter design using window techniques and MATLAB functions.

□ **EXAMPLE 7.8** Design a digital FIR lowpass filter with the following specifications:

$$\omega_p = 0.2\pi, \quad R_p = 0.25 \text{ dB}$$
$$\omega_s = 0.3\pi, \quad A_s = 50 \text{ dB}$$

Choose an appropriate window function from Table 7.1. Determine the impulse response and provide a plot of the frequency response of the designed filter.

Solution Both the Hamming and Blackman windows can provide attenuation of more than 50 dB. Let us choose the Hamming window, which provides the smaller transition band and hence has the smaller order. Although we do not use the passband ripple value of $R_p = 0.25$ dB in the design, we will have to check the actual ripple from the design and verify that it is indeed within the given tolerance. The design steps are given in the following MATLAB script.

```
>> wp = 0.2*pi; ws = 0.3*pi;  tr_width = ws - wp;
>> M = ceil(6.6*pi/tr_width) + 1
M = 67
>> n=[0:1:M-1];
>> wc = (ws+wp)/2, % Ideal LPF cutoff frequency
>> hd = ideal_lp(wc,M);  w_ham = (hamming(M))';  h = hd .* w_ham;
>> [db,mag,pha,grd,w] = freqz_m(h,[1]);  delta_w = 2*pi/1000;
>> Rp = -(min(db(1:1:wp/delta_w+1)));    % Actual Passband Ripple
Rp = 0.0394
>> As = -round(max(db(ws/delta_w+1:1:501))) % Min Stopband attenuation
As = 52
% plots
>> subplot(2,2,1); stem(n,hd); title('Ideal Impulse Response')
>> axis([0 M-1 -0.1 0.3]); xlabel('n'); ylabel('hd(n)')
>> subplot(2,2,2); stem(n,w_ham);title('Hamming Window')
>> axis([0 M-1 0 1.1]); xlabel('n'); ylabel('w(n)')
>> subplot(2,2,3); stem(n,h);title('Actual Impulse Response')
>> axis([0 M-1 -0.1 0.3]); xlabel('n'); ylabel('h(n)')
>> subplot(2,2,4); plot(w/pi,db);title('Magnitude Response in dB');grid
>> axis([0 1 -100 10]); xlabel('frequency in pi units'); ylabel('Decibels')
```

Note that the filter length is $M = 67$, the actual stopband attenuation is 52 dB, and the actual passband ripple is 0.0394 dB. Clearly, the passband ripple is satisfied by this design. This practice of verifying the passband ripple is strongly recommended. The time- and the frequency-domain plots are shown in Figure 7.16. □

□ **EXAMPLE 7.9** For the design specifications given in Example 7.8, choose the Kaiser window to design the necessary lowpass filter.

Solution The design steps are given in the following MATLAB script.

```
>> wp = 0.2*pi; ws = 0.3*pi; As = 50;  tr_width = ws - wp;
>> M = ceil((As-7.95)/(2.285*tr_width/)+1) + 1
M = 61
>> n=[0:1:M-1];  beta = 0.1102*(As-8.7)
beta = 4.5513
>> wc = (ws+wp)/2;  hd = ideal_lp(wc,M);
>> w_kai = (kaiser(M,beta))';  h = hd .* w_kai;
>> [db,mag,pha,grd,w] = freqz_m(h,[1]);  delta_w = 2*pi/1000;
>> As = -round(max(db(ws/delta_w+1:1:501))) % Min Stopband Attenuation
```

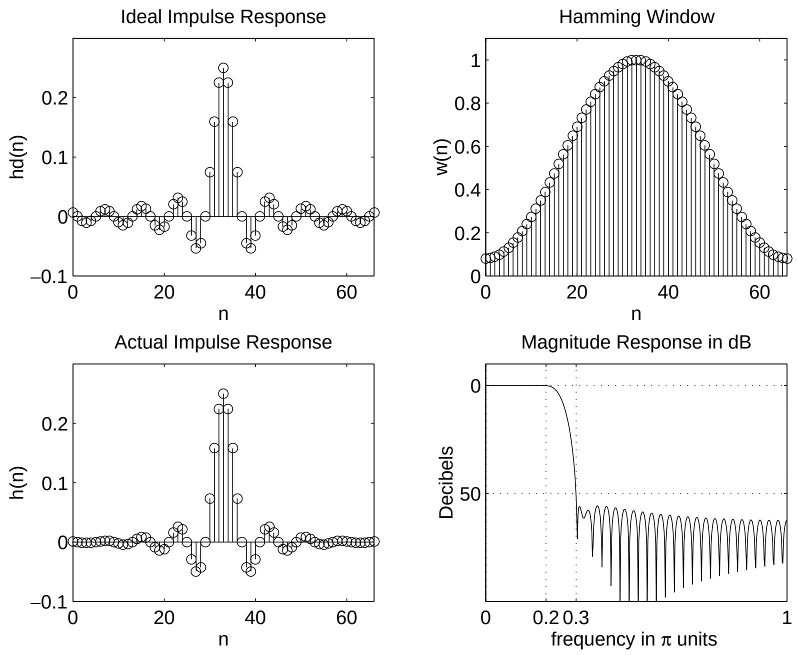


FIGURE 7.16 *Lowpass filter plots for Example 7.8*

```
As = 52
% Plots
>> subplot(2,2,1); stem(n,hd); title('Ideal Impulse Response')
>> axis([0 M-1 -0.1 0.3]); xlabel('n'); ylabel('hd(n)')
>> subplot(2,2,2); stem(n,w_kai);title('Kaiser Window')
>> axis([0 M-1 0 1.1]);  xlabel('n'); ylabel('w(n)')
>> subplot(2,2,3); stem(n,h);title('Actual Impulse Response')
>> axis([0 M-1 -0.1 0.3]); xlabel('n'); ylabel('h(n)')
>> subplot(2,2,4);plot(w/pi,db);title('Magnitude Response in dB');grid
>> axis([0 1 -100 10]); xlabel('frequency in pi units'); ylabel('Decibels')
```

Note that the Kaiser window parameters are $M = 61$ and $\beta = 4.5513$ and that the actual stopband attenuation is 52 dB. The time- and the frequency-domain plots are shown in Figure 7.17. □

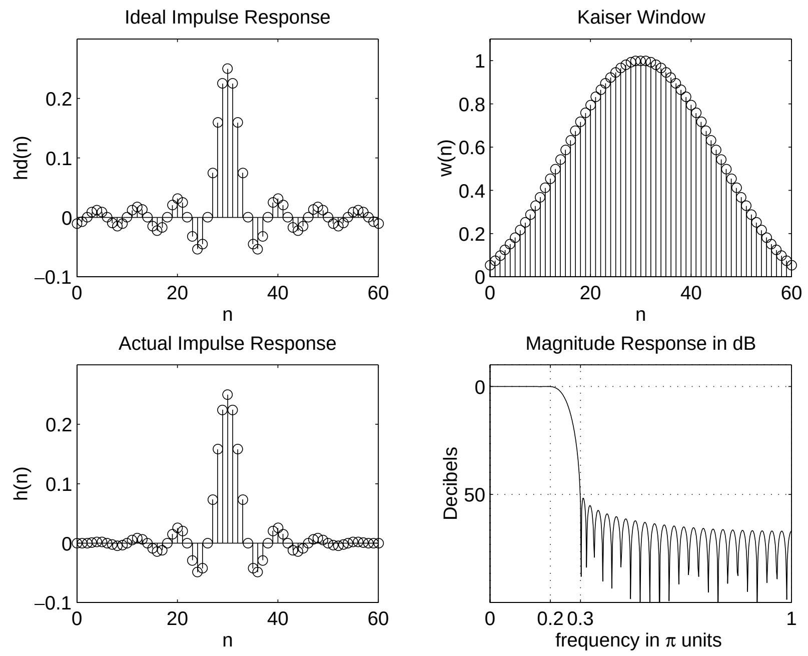


FIGURE 7.17 *Lowpass filter plots for Example 7.9*

□ **EXAMPLE 7.10** Let us design the following digital bandpass filter.

$$\begin{aligned}
\text{lower stopband edge: } & \omega_{1s} = 0.2\pi, & A_s = 60 \text{ dB} \\
\text{lower passband edge: } & \omega_{1p} = 0.35\pi, & R_p = 1 \text{ dB} \\
\text{upper passband edge: } & \omega_{2p} = 0.65\pi & R_p = 1 \text{ dB} \\
\text{upper stopband edge: } & \omega_{2s} = 0.8\pi & A_s = 60 \text{ dB}
\end{aligned}$$

These quantities are shown in Figure 7.18.

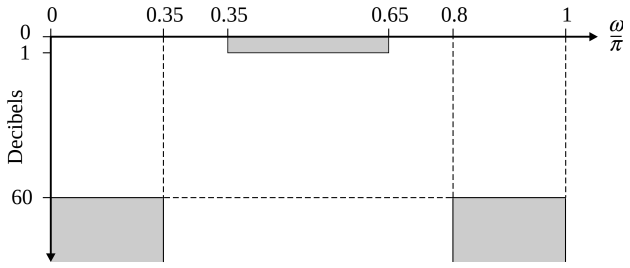


FIGURE 7.18 *Bandpass filter specifications in Example 7.10*

Solution There are two transition bands, namely, $\Delta\omega_1 \overset{\triangle}{=} \omega_{1p} - \omega_{1s}$ and $\Delta\omega_2 \overset{\triangle}{=} \omega_{2s} - \omega_{2p}$. These two bandwidths must be the same in the window design; that is, there is no independent control over $\Delta\omega_1$ and $\Delta\omega_2$. Hence $\Delta\omega_1 = \Delta\omega_2 = \Delta\omega$. For this design we can use either the Kaiser window or the Blackman window. Let us use the Blackman Window. We will also need the ideal bandpass filter impulse response $h_d(n)$. Note that this impulse response can be obtained from two ideal lowpass magnitude responses, provided they have the same phase response. This is shown in Figure 7.19. Therefore the MATLAB routine ideal_lp(wc,M) is sufficient to determine the impulse response of an ideal bandpass filter. The design steps are given in the following MATLAB script.

```
>> ws1 = 0.2*pi; wp1 = 0.35*pi;  wp2 = 0.65*pi; ws2 = 0.8*pi;  As = 60;
>> tr_width = min((wp1-ws1),(ws2-wp2));  M = ceil(11*pi/tr_width) + 1
M = 75
>> n=[0:1:M-1];  wc1 = (ws1+wp1)/2; wc2 = (wp2+ws2)/2;
>> hd = ideal_lp(wc2,M) - ideal_lp(wc1,M);
>> w_bla = (blackman(M))';  h = hd .* w_bla;
>> [db,mag,pha,grd,w] = freqz_m(h,[1]);  delta_w = 2*pi/1000;
>> Rp = -min(db(wp1/delta_w+1:1:wp2/delta_w)) % Actua; Passband Ripple
Rp = 0.0030
>> As = -round(max(db(ws2/delta_w+1:1:501))) % Min Stopband Attenuation
As = 75
%Plots
>> subplot(2,2,1); stem(n,hd); title('Ideal Impulse Response')
>> axis([0 M-1 -0.4 0.5]); xlabel('n'); ylabel('hd(n)')
>> subplot(2,2,2); stem(n,w_bla);title('Blackman Window')
>> axis([0 M-1 0 1.1]); xlabel('n'); ylabel('w(n)')
>> subplot(2,2,3); stem(n,h);title('Actual Impulse Response')
>> axis([0 M-1 -0.4 0.5]); xlabel('n'); ylabel('h(n)')
>> subplot(2,2,4);plot(w/pi,db);axis([0 1 -150 10]);
>> title('Magnitude Response in dB');grid;
>> xlabel('frequency in pi units'); ylabel('Decibels')
```

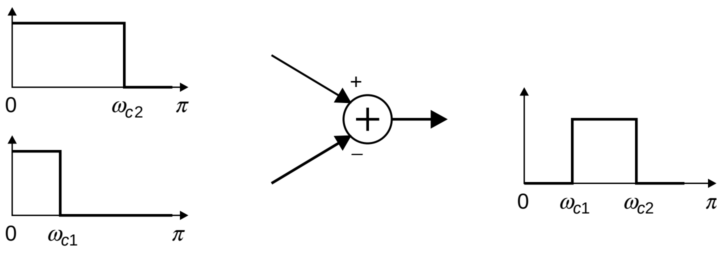


FIGURE 7.19 *Ideal bandpass filter from two lowpass filters*

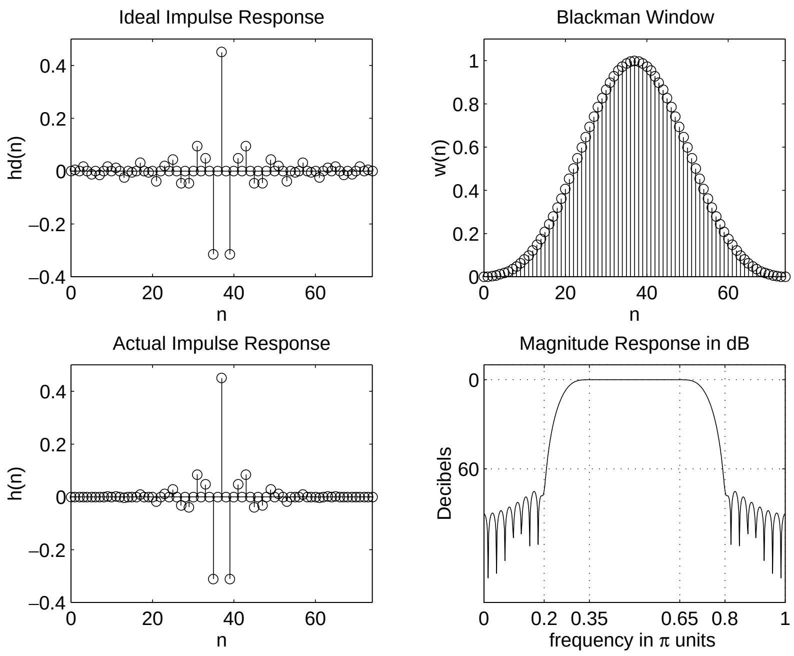


FIGURE 7.20 *Bandpass filter plots in Example 7.10*

Note that the Blackman window length is $M = 61$ and that the actual stopband attenuation is 75 dB. The time- and the frequency-domain plots are shown in Figure 7.20. □

□ **EXAMPLE 7.11** The frequency response of an ideal bandstop filter is given by

$$H_e(e^{j\omega}) = \begin{cases} 1, & 0 \le |\omega| < \pi/3 \\ 0, & \pi/3 \le |\omega| \le 2\pi/3 \\ 1, & 2\pi/3 < |\omega| \le \pi \end{cases}$$

Using a Kaiser window, design a bandstop filter of length 45 with stopband attenuation of 60 dB.

Solution Note that in these design specifications, the transition bandwidth is not given. It will be determined by the length $M = 45$ and the parameter β of the Kaiser window. From the design equations (7.30), we can determine β from A_s; that is,

$$\beta = 0.1102 \times (A_s - 8.7)$$

The ideal bandstop impulse response can also be determined from the ideal lowpass impulse response using a method similar to Figure 7.19. We can now implement the Kaiser window design and check for the minimum stopband attenuation. This is shown in the following MATLAB script.

```
>> M = 45; As = 60; n=[0:1:M-1];
>> beta = 0.1102*(As-8.7)
beta = 5.6533
>> w_kai = (kaiser(M,beta))';  wc1 = pi/3; wc2 = 2*pi/3;
>> hd = ideal_lp(wc1,M) + ideal_lp(pi,M) - ideal_lp(wc2,M);
>> h = hd .* w_kai;  [db,mag,pha,grd,w] = freqz_m(h,[1]);
>> subplot(2,2,1); stem(n,hd); title('Ideal Impulse Response')
>> axis([-1 M -0.2 0.8]); xlabel('n'); ylabel('hd(n)')
>> subplot(2,2,2); stem(n,w_kai);title('Kaiser Window')
>> axis([-1 M 0 1.1]); xlabel('n'); ylabel('w(n)')
>> subplot(2,2,3); stem(n,h);title('Actual Impulse Response')
>> axis([-1 M -0.2 0.8]); xlabel('n'); ylabel('h(n)')
>> subplot(2,2,4);plot(w/pi,db); axis([0 1 -80 10]);
>> title('Magnitude Response in dB');grid;
>> xlabel('frequency in pi units'); ylabel('Decibels')
```

The β parameter is equal to 5.6533, and, from the magnitude plot in Figure 7.21, we observe that the minimum stopband attenuation is smaller than 60 dB. Clearly, we have to increase β to increase the attenuation to 60 dB. The required value was found to be $\beta = 5.9533$.

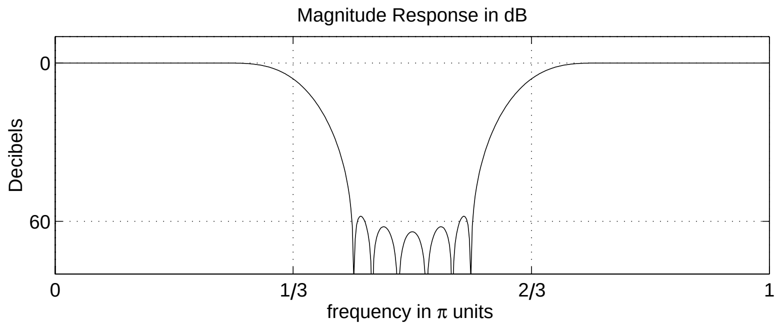


FIGURE 7.21 *Bandstop filter magnitude response in Example 7.11 for* $\beta = 5.6533$

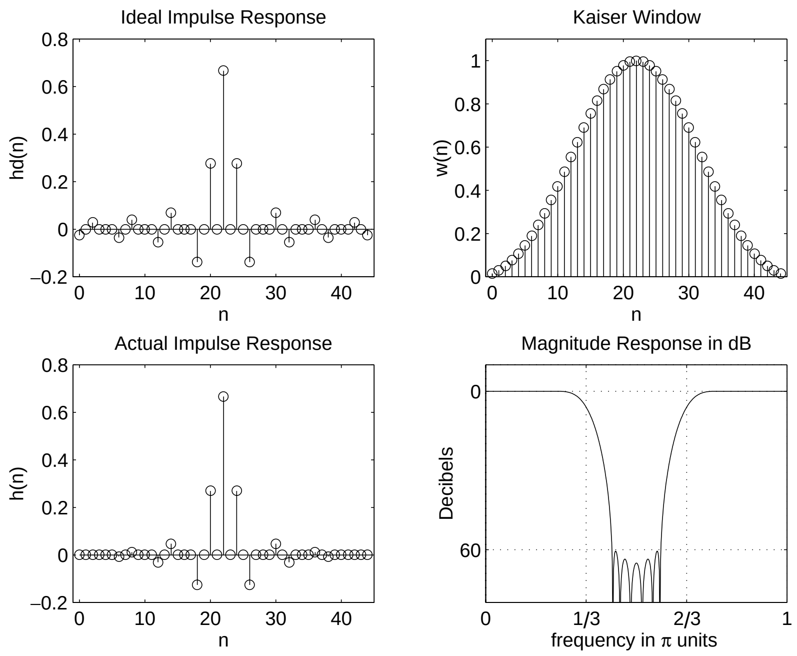


FIGURE 7.22 *Bandstop filter plots in Example 7.11:* $\beta = 5.9533$

```
>> M = 45; As = 60; n=[0:1:M-1];
>> beta = 0.1102*(As-8.7)+0.3
beta = 5.9533
>> w_kai = (kaiser(M,beta))';  wc1 = pi/3; wc2 = 2*pi/3;
>> hd = ideal_lp(wc1,M) + ideal_lp(pi,M) - ideal_lp(wc2,M);
>> h = hd .* w_kai;  [db,mag,pha,grd,w] = freqz_m(h,[1]);
>> subplot(2,2,1); stem(n,hd); title('Ideal Impulse Response')
>> axis([-1 M -0.2 0.8]); xlabel('n'); ylabel('hd(n)')
>> subplot(2,2,2); stem(n,w_kai);title('Kaiser Window')
>> axis([-1 M 0 1.1]); xlabel('n'); ylabel('w(n)')
>> subplot(2,2,3); stem(n,h);title('Actual Impulse Response')
>> axis([-1 M -0.2 0.8]); xlabel('n'); ylabel('h(n)')
>> subplot(2,2,4);plot(w/pi,db); axis([0 1 -80 10]);
>> title('Magnitude Response in dB');grid;
>> xlabel('frequency in pi units'); ylabel('Decibels')
```

The time- and the frequency-domain plots are shown in Figure 7.22, in which the designed filter satisfies the necessary requirements. □

□ **EXAMPLE 7.12** The frequency response of an ideal digital differentiator is given by

$$H_d(e^{j\omega}) = \begin{cases} j\omega, & 0 < \omega \le \pi \\ -j\omega, & -\pi < \omega < 0 \end{cases} \tag{7.31}$$

Using a Hamming window of length 21, design a digital FIR differentiator. Plot the time- and the frequency-domain responses.

Solution The ideal impulse response of a digital differentiator with linear phase is given by

$$\begin{aligned} h_d(n) &= \mathcal{F}\left[H_d(e^{j\omega})e^{-j\alpha\omega}\right] = \frac{1}{2\pi}\int_{-\pi}^{\pi} H_d(e^{j\omega})e^{-j\alpha\omega}e^{j\omega n}d\omega \\ &= \frac{1}{2\pi}\int_{-\pi}^{0} (-j\omega)\, e^{-j\alpha\omega}e^{j\omega n}d\omega + \frac{1}{2\pi}\int_{0}^{\pi} (j\omega)\, e^{-j\alpha\omega}e^{j\omega n}d\omega \\ &= \begin{cases} \dfrac{\cos\pi\,(n-\alpha)}{(n-\alpha)}, & n \ne \alpha \\ 0, & n = \alpha \end{cases} \end{aligned}$$

This impulse response can be implemented in MATLAB, along with the Hamming window to design the required differentiator. Note that if M is an even number, then $\alpha = (M-1)/2$ is not an integer and $h_d(n)$ will be zero for all n. Hence M must be an odd number, and this will be a Type-3 linear-phase FIR filter. However, the filter will not be a full-band differentiator since $H_r(\pi) = 0$ for Type-3 filters.

```
>> M = 21; alpha = (M-1)/2;  n = 0:M-1;
>> hd = (cos(pi*(n-alpha)))./(n-alpha); hd(alpha+1)=0;
>> w_ham = (hamming(M))';  h = hd .* w_ham;  [Hr,w,P,L] = Hr_Type3(h);
% plots
>> subplot(2,2,1); stem(n,hd); title('Ideal Impulse Response')
>> axis([-1 M -1.2 1.2]); xlabel('n'); ylabel('hd(n)')
>> subplot(2,2,2); stem(n,w_ham);title('Hamming Window')
>> axis([-1 M 0 1.2]); xlabel('n'); ylabel('w(n)')
>> subplot(2,2,3); stem(n,h);title('Actual Impulse Response')
>> axis([-1 M -1.2 1.2]); xlabel('n'); ylabel('h(n)')
>> subplot(2,2,4);plot(w/pi,Hr/pi); title('Amplitude Response');grid;
>> xlabel('frequency in pi units'); ylabel('slope in pi units'); axis([0 1 0 1]);
```

The plots are shown in Figure 7.23. □

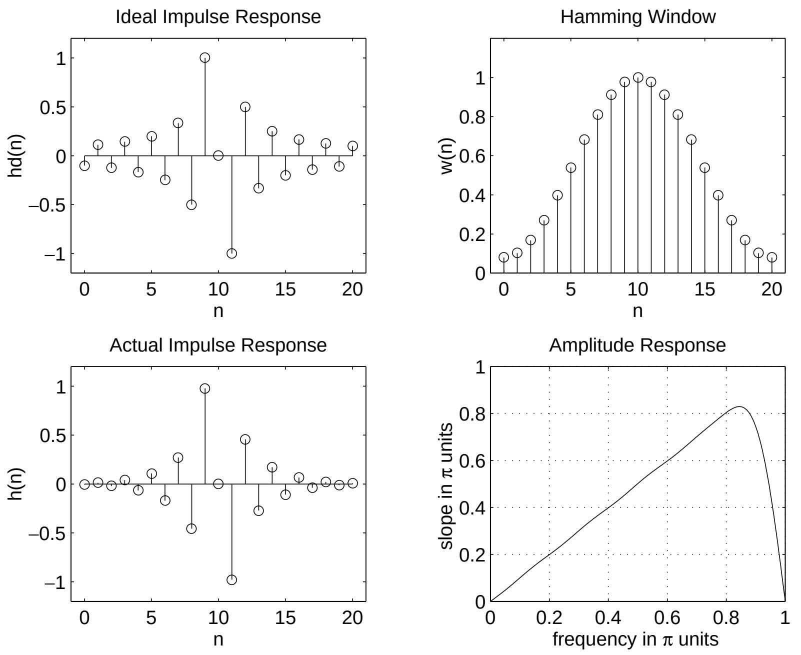


FIGURE 7.23 *FIR differentiator design in Example 7.12*

☐ **EXAMPLE 7.13** Design a length-25 digital Hilbert transformer using a Hann window.

Solution The ideal frequency response of a linear-phase Hilbert transformer is given by

$$H_d(e^{j\omega}) = \begin{cases} -je^{-j\alpha\omega}, & 0 < \omega < \pi \\ +je^{-j\alpha\omega}, & -\pi < \omega < 0 \end{cases} \tag{7.32}$$

After inverse transformation the ideal impulse response is given by

$$h_d(n) = \begin{cases} \dfrac{2}{\pi}\dfrac{\sin^2 \pi (n-\alpha)/2}{n-\alpha}, & n \neq \alpha \\ 0, & n = \alpha \end{cases}$$

which can be easily implemented in MATLAB. Note that since $M = 25$, the designed filter is of Type-3.

MATLAB script:

```
>> M = 25; alpha = (M-1)/2;  n = 0:M-1;
>> hd = (2/pi)*((sin((pi/2)*(n-alpha)).^2)./(n-alpha)); hd(alpha+1)=0;
>> w_han = (hann(M))';  h = hd .* w_han;  [Hr,w,P,L] = Hr_Type3(h);
>> subplot(2,2,1); stem(n,hd); title('Ideal Impulse Response')
>> axis([-1 M -1.2 1.2]); xlabel('n'); ylabel('hd(n)')
```

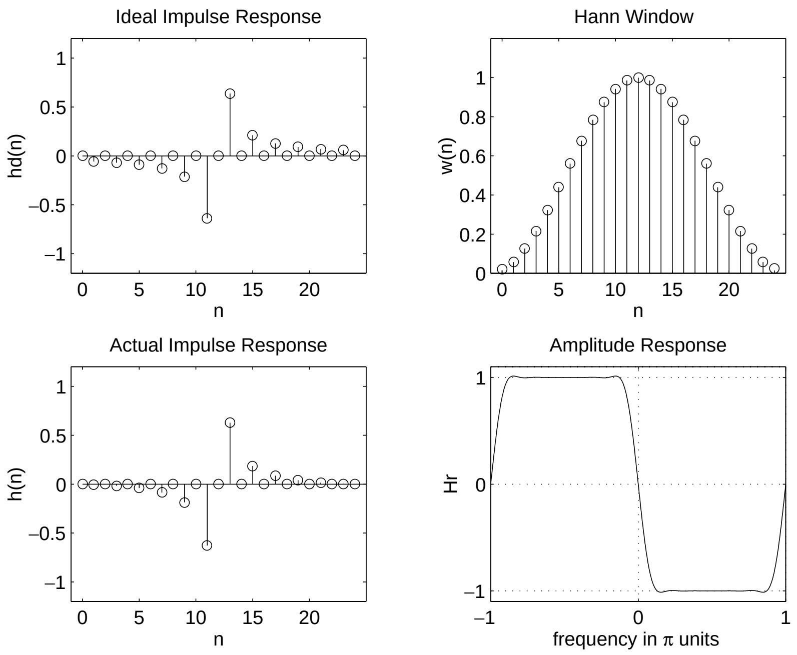


FIGURE 7.24 *FIR Hilbert transformer design in Example 7.13*

```
>> subplot(2,2,2); stem(n,w_han);title('Hann Window')
>> axis([-1 M 0 1.2]); xlabel('n'); ylabel('w(n)')
>> subplot(2,2,3); stem(n,h);title('Actual Impulse Response')
>> axis([-1 M -1.2 1.2]); xlabel('n'); ylabel('h(n)')
>> w = w'; Hr = Hr';
>> w = [-fliplr(w), w(2:501)]; Hr = [-fliplr(Hr), Hr(2:501)];
>> subplot(2,2,4);plot(w/pi,Hr); title('Amplitude Response');grid;
>> xlabel('frequency in pi units'); ylabel('Hr'); axis([-1 1 -1.1 1.1]);
```

The plots are shown in Figure 7.24. Observe that the amplitude response is plotted over $-\pi \leq \omega \leq \pi$. □

The SP toolbox provides a function called `fir1` which designs conventional lowpass, highpass, and other multiband FIR filters using window technique. This function's syntax has several forms, including:

- `h = fir1(N,wc)` designs an Nth-order ($N = M - 1$) lowpass FIR filter and returns the impulse response in vector `h`. By default this is a Hamming-window based, linear-phase design with normalized cutoff frequency in `wc` which is a number between 0 and 1, where 1 corresponds to π rad/sample. If `wc` is a two-element vector, i.e., `wc = [wc1 wc2]`, then `fir1` returns a bandpass filter with passband cutoffs `wc1` and `wc2`. If `wc` is a multi-element (more than two) vector, then `fir1` returns a multiband filter with cutoffs given in `wc`.

- `h = fir1(N,wc,'ftype')` specifies a filter type, where `'ftype'` is:

  a. `'high'` for a highpass filter with cutoff frequency Wn.
  b. `'stop'` for a bandstop filter, if `Wc = [wc1 wc2]`. The stopband frequency range is specified by this interval.
  c. `'DC-1'` to make the first band of a multiband filter a passband.
  d. `'DC-0'` to make the first band of a multiband filter a stopband.

- `h = fir1(N,wc,'ftype',window)` or `h = fir1(N,wc,window)` uses the vector `window` of length `N+1` obtained from one of the specified MATLAB window function. The default window function used is the Hamming window.

To design FIR filters using the Kaiser window, the SP toolbox provides the function `kaiserord`, which estimates window parameters that can be used in the `fir1` function. The basic syntax is

```
[N,wc,beta,ftype] = kaiserord(f,m,ripple);
```

The function computes the window order `N`, the cutoff frequency vector `wc`, parameter β in `beta`, and the filter type `ftype` as discussed. The vector `f` is a vector of normalized band edges and `m` is a vector specifying the desired amplitude on the bands defined by `f`. The length of `f` is twice the length of `m`, minus 2; i.e., `f` does not contain 0 or 1. The vector `ripple` specifies tolerances in each band (not in decibels). Using the estimated parameters, Kaiser window array can be computed and used in the `fir1` function.

To design FIR filters using window technique with arbitrary shaped magnitude response, the SP toolbox provides the function `fir2`, which also incorporates the frequency sampling technique. It is explained in the following section.

7.4 FREQUENCY SAMPLING DESIGN TECHNIQUES

In this design approach we use the fact that the system function $H(z)$ can be obtained from the samples $H(k)$ of the frequency response $H(e^{j\omega})$. Furthermore, this design technique fits nicely with the frequency sampling structure that we discussed in Chapter 6. Let $h(n)$ be the impulse response of an M-point FIR filter, $H(k)$ be its M-point DFT, and $H(z)$ be its system function. Then from (6.12) we have

$$H\left(z\right)=\sum_{n=0}^{M-1}h\left(n\right)z^{-n}=\frac{1-z^{-M}}{M}\sum_{k=0}^{M-1}\frac{H(k)}{1-z^{-1}e^{j2\pi k/M}} \tag{7.33}$$

and

$$H(e^{j\omega})=\frac{1-e^{-j\omega M}}{M}\sum_{k=0}^{M-1}\frac{H(k)}{1-e^{-j\omega}e^{j2\pi k/M}} \tag{7.34}$$

with

$$H\left(k\right)=H\left(e^{j2\pi k/M}\right)=\begin{cases}H\left(0\right), & k=0\\ H^{*}\left(M-k\right), & k=1,\ldots,M-1\end{cases}$$

For a linear-phase FIR filter we have

$$h(n)=\pm h(M-1-n),\quad n=0,1,\ldots,M-1$$

where the positive sign is for the Type-1 and Type-2 linear-phase filters, while the negative sign is for the Type-3 and Type-4 linear-phase filters. Then $H\left(k\right)$ is given by

$$H\left(k\right)=H_{r}\left(\frac{2\pi k}{M}\right)e^{j\angle H(k)} \tag{7.35}$$

where

$$H_{r}\left(\frac{2\pi k}{M}\right)=\begin{cases}H_{r}\left(0\right), & k=0\\ H_{r}\left(\frac{2\pi(M-k)}{M}\right), & k=1,\ldots,M-1\end{cases} \tag{7.36}$$

and

$$\angle H\left(k\right)=\begin{cases}-\left(\frac{M-1}{2}\right)\left(\frac{2\pi k}{M}\right), & k=0,\ldots,\left\lfloor\frac{M-1}{2}\right\rfloor\\ +\left(\frac{M-1}{2}\right)\frac{2\pi}{M}\left(M-k\right), & k=\left\lfloor\frac{M-1}{2}\right\rfloor+1,\ldots,M-1\end{cases},\ \text{(Type-1 \& 2)} \tag{7.37}$$

or

$$\angle H\left(k\right)=\begin{cases}\left(\pm\frac{\pi}{2}\right)-\left(\frac{M-1}{2}\right)\left(\frac{2\pi k}{M}\right), & k=0,\ldots,\left\lfloor\frac{M-1}{2}\right\rfloor\\ -\left(\pm\frac{\pi}{2}\right)+\left(\frac{M-1}{2}\right)\frac{2\pi}{M}\left(M-k\right), & \\ & k=\left\lfloor\frac{M-1}{2}\right\rfloor+1,\ldots,M-1\end{cases},\ \text{(Type-3 \& 4)} \tag{7.38}$$

Finally, we have

$$h(n) = \text{IDFT}\,[H(k)] \tag{7.39}$$

Note that several textbooks (e.g., [18, 23, 24]) provide explicit formulas to compute $h(n)$, given $H(k)$. We will use MATLAB's `ifft` function to compute $h(n)$ from (7.39).

Basic idea Given the ideal lowpass filter $H_d(e^{j\omega})$, choose the filter length M and then sample $H_d(e^{j\omega})$ at M equispaced frequencies between 0 and 2π. The actual response $H(e^{j\omega})$ is the interpolation of the samples $H(k)$ given by (7.34). This is shown in Figure 7.25. The impulse response is given by (7.39). Similar steps apply to other frequency-selective filters. Furthermore, this idea can also be extended for approximating arbitrary frequency-domain specifications.

From Figure 7.25, we observe the following:

1. The approximation error—that is, the difference between the ideal and the actual response—is zero at the sampled frequencies.
2. The approximation error at all other frequencies depends on the shape of the ideal response; that is, the sharper the ideal response, the larger the approximation error.
3. The error is larger near the band edges and smaller within the band.

There are two design approaches. In the first approach, we use the basic idea literally and provide no constraints on the approximation error; that is, we accept whatever error we get from the design. This approach is called a *naive design* method. In the second approach, we try to minimize error in the stopband by varying values of the transition band samples. It results in a much better design called an *optimum design* method.

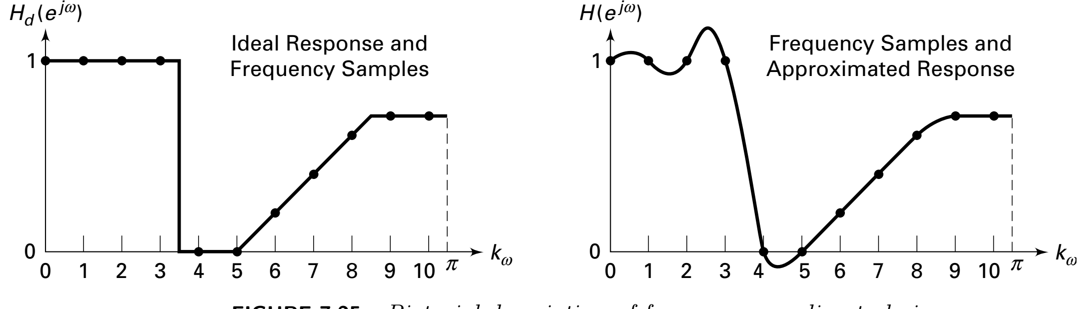


FIGURE 7.25 *Pictorial description of frequency sampling technique*

7.4.1 NAIVE DESIGN METHOD

In this method we set $H(k) = H_d(e^{j2\pi k/M}), \quad k = 0, \ldots, M-1$ and use (7.35) through (7.39) to obtain the impulse response $h(n)$.

□ **EXAMPLE 7.14** Consider the lowpass filter specifications from Example 7.8.

$$\omega_p = 0.2\pi,\ R_p = 0.25 \text{ dB}$$

$$\omega_s = 0.3\pi,\ A_s = 50 \text{ dB}$$

Design an FIR filter using the frequency sampling approach.

Solution Let us choose $M = 20$ so that we have a frequency sample at ω_p, that is, at $k = 2$:

$$\omega_p = 0.2\pi = \frac{2\pi}{20}2$$

and the next sample at ω_s, that is, at $k = 3$:

$$\omega_s = 0.3\pi = \frac{2\pi}{20}3$$

Thus we have 3 samples in the passband $[0 \le \omega \le \omega_p]$ and 7 samples in the stopband $[\omega_s \le \omega \le \pi]$. From (7.36) we have

$$H_r(k) = [1, 1, 1, \underbrace{0, \ldots, 0}_{15 \text{ zeros}}, 1, 1]$$

Since $M = 20$, $\alpha = \frac{20-1}{2} = 9.5$ and since this is a Type-2 linear-phase filter, from (7.37) we have

$$\angle H(k) = \begin{cases} -9.5\dfrac{2\pi}{20}k = -0.95\pi k, & 0 \le k \le 9 \\ +0.95\pi(20-k), & 10 \le k \le 19 \end{cases}$$

Now from (7.35) we assemble $H(k)$ and from (7.39) determine the impulse response $h(n)$. The MATLAB script follows:

```
>> M = 20; alpha = (M-1)/2; l = 0:M-1; wl = (2*pi/M)*l;
>> Hrs = [1,1,1,zeros(1,15),1,1]; %Ideal Amp Res sampled
>> Hdr = [1,1,0,0]; wdl = [0,0.25,0.25,1]; %Ideal Amp Res for plotting
>> k1 = 0:floor((M-1)/2); k2 = floor((M-1)/2)+1:M-1;
>> angH = [-alpha*(2*pi)/M*k1, alpha*(2*pi)/M*(M-k2)];
>> H = Hrs.*exp(j*angH);  h = real(ifft(H,M));
>> [db,mag,pha,grd,w] = freqz_m(h,1);  [Hr,ww,a,L] = Hr_Type2(h);
>> subplot(2,2,1);plot(wl(1:11)/pi,Hrs(1:11),'o',wdl,Hdr);
>> axis([0,1,-0.1,1.1]); title('Frequency Samples: M=20')
>> xlabel('frequency in pi units'); ylabel('Hr(k)')
```

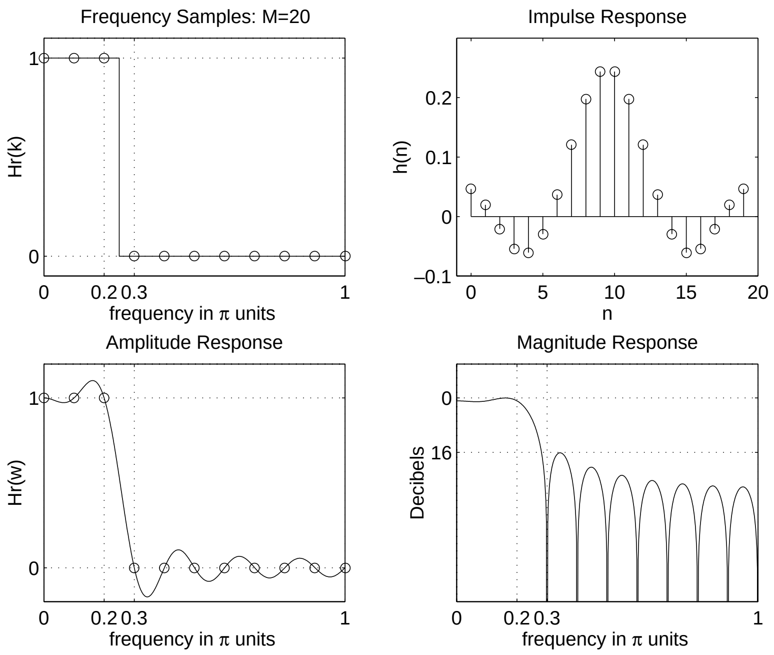


FIGURE 7.26 *Naive frequency sampling design method*

```
>> subplot(2,2,2); stem(l,h); axis([-1,M,-0.1,0.3])
>> title('Impulse Response'); xlabel('n'); ylabel('h(n)');
>> subplot(2,2,3); plot(ww/pi,Hr,wl(1:11)/pi,Hrs(1:11),'o');
>> axis([0,1,-0.2,1.2]); title('Amplitude Response')
>> xlabel('frequency in pi units'); ylabel('Hr(w)')
>> subplot(2,2,4);plot(w/pi,db); axis([0,1,-60,10]); grid
>> title('Magnitude Response'); xlabel('frequency in pi units');
>> ylabel('Decibels');
```

The time- and the frequency-domain plots are shown in Figure 7.26. Observe that the minimum stopband attenuation is about 16 dB, which is clearly unacceptable. If we increase M, then there will be samples in the transition band, for which we do not precisely know the frequency response. Therefore the naive design method is seldom used in practice. □

7.4.2 OPTIMUM DESIGN METHOD

To obtain more attenuation, we will have to increase M and make the transition band samples free samples—that is, we vary their values to obtain the largest attenuation for the given M and the transition width.

This problem is known as an optimization problem, and it is solved using linear programming techniques. We demonstrate the effect of transition band sample variation on the design using the following example.

□ **EXAMPLE 7.15** Using the optimum design method, design a better lowpass filter of Example 7.14.

Solution Let us choose $M = 40$ so that we have one sample in the transition band $0.2\pi < \omega < 0.3\pi$. Since $\omega_1 \triangleq 2\pi/40$, the transition band samples are at $k = 5$ and at $k = 40 - 5 = 35$. Let us denote the value of these samples by T_1, $0 < T_1 < 1$; then the sampled amplitude response is

$$H_r(k) = [1, 1, 1, 1, 1, T_1, \underbrace{0, \ldots, 0}_{\text{29 zeros}}, T_1, 1, 1, 1, 1]$$

Since $\alpha = \frac{40-1}{2} = 19.5$, the samples of the phase response are

$$\angle H(k) = \begin{cases} -19.5\dfrac{2\pi}{40}k = -0.975\pi k, & 0 \le k \le 19 \\ +0.975\pi(40-k), & 20 \le k \le 39 \end{cases}$$

Now we can vary T_1 to get the best minimum stopband attenuation. This will result in the widening of the transition width. We first see what happens when $T_1 = 0.5$ using the following MATLAB script.

```
% T1 = 0.5
>> M = 40; alpha = (M-1)/2;
>> Hrs = [ones(1,5),0.5,zeros(1,29),0.5,ones(1,4)];
>> k1 = 0:floor((M-1)/2); k2 = floor((M-1)/2)+1:M-1;
>> angH = [-alpha*(2*pi)/M*k1, alpha*(2*pi)/M*(M-k2)];
>> H = Hrs.*exp(j*angH);
>> h = real(ifft(H,M));
```

From the plots of this design in Figure 7.27, we observe that the minimum stopband attenuation is now 30 dB, which is better than the naive design attenuation but is still not at the acceptable level of 50 dB. The best value for T_1 was obtained by varying it manually (although more efficient linear programming techniques are available, these were not used in this case), and the near optimum solution was found at $T_1 = 0.39$. The resulting filter is a obtained using the following MATLAB script.

```
% T1 = 0.39
>> M = 40; alpha = (M-1)/2;
>> Hrs = [ones(1,5),0.39,zeros(1,29),0.39,ones(1,4)];
>> k1 = 0:floor((M-1)/2); k2 = floor((M-1)/2)+1:M-1;
>> angH = [-alpha*(2*pi)/M*k1, alpha*(2*pi)/M*(M-k2)];
>> H = Hrs.*exp(j*angH);  h = real(ifft(H,M));
```

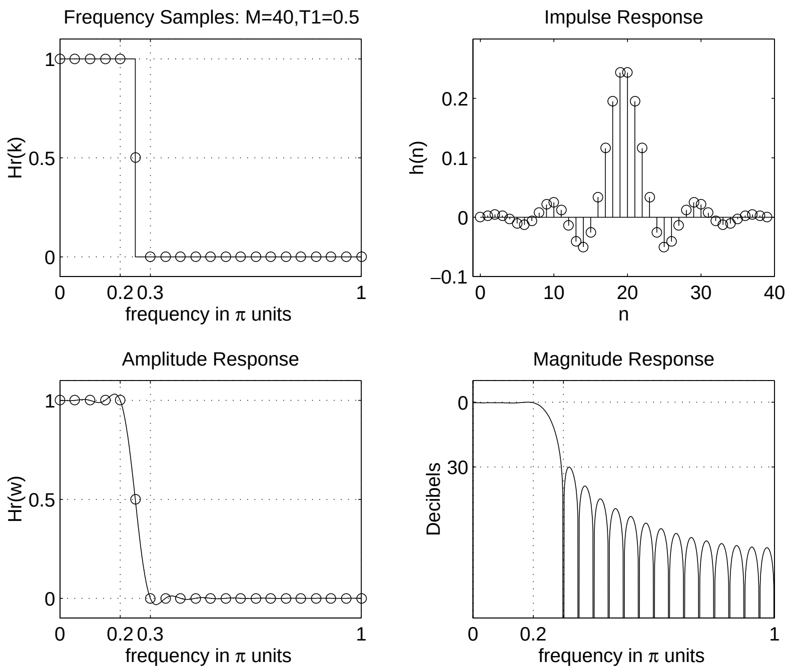


FIGURE 7.27 *Optimum frequency design method:* $T_1 = 0.5$

From the plots in Figure 7.28, we observe that the optimum stopband attenuation is 43 dB. It is obvious that to further increase the attenuation, we will have to vary more than one sample in the transition band. □

Clearly, this method is superior in that by varying one sample we can get a much better design. In practice the transition bandwidth is generally small, containing either one or two samples. Hence we need to optimize at most two samples to obtain the largest minimum stopband attenuation. This is also equivalent to minimizing the maximum side-lobe magnitudes in the absolute sense. Hence this optimization problem is also called a *minimax* problem. This problem is solved by Rabiner et al. [24], and the solution is available in the form of tables of transition values. A selected number of tables are also available in [23, Appendix B]. This problem can also be solved in MATLAB, but it would require the use of the Optimization toolbox. We will consider a more general version of this problem in the next section. We now illustrate the use of these tables in the following examples.

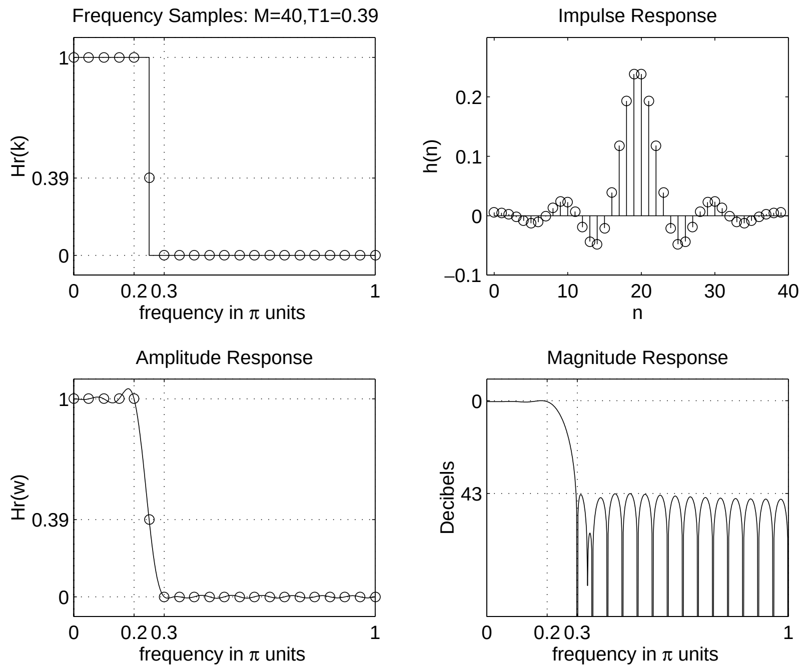


FIGURE 7.28 *Optimum frequency design method:* $T_1 = 0.39$

□ **EXAMPLE 7.16** Let us revisit our lowpass filter design in Example 7.14. We will solve it using two samples in the transition band so that we can get a better stopband attenuation.

Solution Let us choose $M = 60$ so that there are two samples in the transition band. Let the values of these transition band samples be T_1 and T_2. Then $H_r(\omega)$ is given by

$$H(\omega) = [\underbrace{1, \ldots, 1}_{7 \text{ ones}}, T_1, T_2, \underbrace{0, \ldots, 0}_{43 \text{ zeros}}, T_2, T_1, \underbrace{1, \ldots, 1}_{6 \text{ ones}}]$$

From tables in [22, Appendix B] $T_1 = 0.5925$ and $T_2 = 0.1099$. Using these values, we use MATLAB to compute $h(n)$.

```
>> M = 60; alpha = (M-1)/2; l = 0:M-1; wl = (2*pi/M)*l;
>> Hrs = [ones(1,7),0.5925,0.1099,zeros(1,43),0.1099,0.5925,ones(1,6)];
>> Hdr = [1,1,0,0]; wdl = [0,0.2,0.3,1];
>> k1 = 0:floor((M-1)/2); k2 = floor((M-1)/2)+1:M-1;
>> angH = [-alpha*(2*pi)/M*k1, alpha*(2*pi)/M*(M-k2)];
>> H = Hrs.*exp(j*angH);  h = real(ifft(H,M));
>> [db,mag,pha,grd,w] = freqz_m(h,1);  [Hr,ww,a,L] = Hr_Type2(h);
```

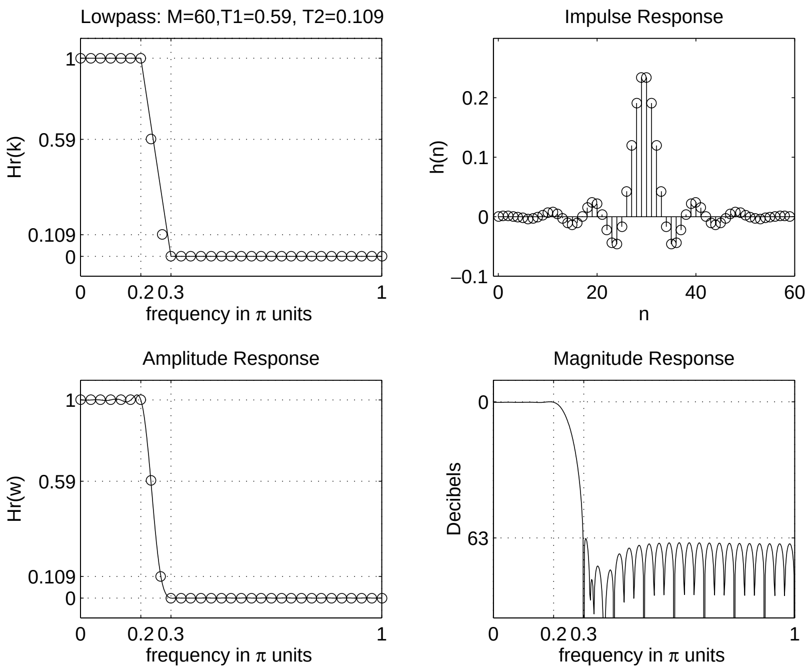


FIGURE 7.29 *Lowpass filter design plots in Example 7.16*

The time- and the frequency-domain plots are shown in Figure 7.29. The minimum stopband attenuation is now at 63 dB, which is acceptable. □

□ **EXAMPLE 7.17** Design the bandpass filter of Example 7.10 using the frequency sampling technique. The design specifications are these:

$$\text{lower stopband edge: } \omega_{1s} = 0.2\pi, \quad A_s = 60 \text{ dB}$$

$$\text{lower passband edge: } \omega_{1p} = 0.35\pi, \; R_p = 1 \text{ dB}$$

$$\text{upper passband edge: } \omega_{2p} = 0.65\pi \quad R_p = 1 \text{ dB}$$

$$\text{upper stopband edge: } \omega_{2s} = 0.8\pi \quad A_s = 60 \text{ dB}$$

Solution Let us choose $M = 40$ so that we have two samples in the transition band. Let the frequency samples in the lower transition band be T_1 and T_2. Then the samples of the amplitude response are

$$H_r(\omega) = [\underbrace{0,\ldots,0}_{5}, T_1, T_2, \underbrace{1,\ldots,1}_{7}, T_2, T_1, \underbrace{0,\ldots,0}_{9}, T_1, T_2, \underbrace{1,\ldots,1}_{7}, T_2, T_1, \underbrace{0,\ldots,0}_{4}]$$

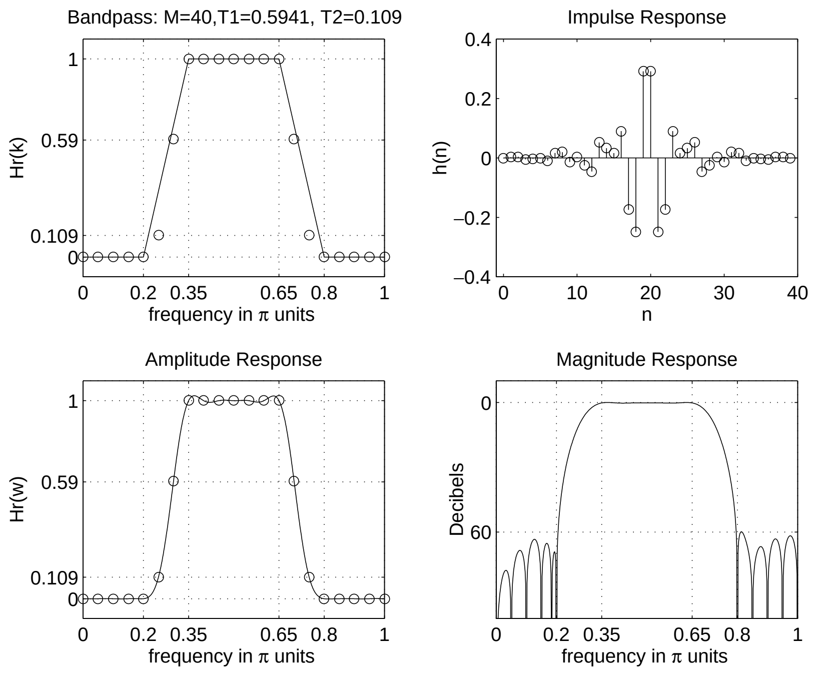


FIGURE 7.30 *Bandpass filter design plots in Example 7.17*

The optimum values of T_1 and T_2 for $M = 40$ and seven samples in the passband [23, Appendix B] are

$$T_1 = 0.109021, \quad T_2 = 0.59417456$$

The MATLAB script is

```
>> M = 40; alpha = (M-1)/2; l = 0:M-1; wl = (2*pi/M)*l;
>> T1 = 0.109021; T2 = 0.59417456;
>> Hrs=[zeros(1,5),T1,T2,ones(1,7),T2,T1,zeros(1,9),T1,T2,ones(1,7),T2,T1,zeros(1,4)];
>> Hdr = [0,0,1,1,0,0]; wdl = [0,0.2,0.35,0.65,0.8,1];
>> k1 = 0:floor((M-1)/2); k2 = floor((M-1)/2)+1:M-1;
>> angH = [-alpha*(2*pi)/M*k1, alpha*(2*pi)/M*(M-k2)];
>> H = Hrs.*exp(j*angH);  h = real(ifft(H,M));
>> [db,mag,pha,grd,w] = freqz_m(h,1);  [Hr,ww,a,L] = Hr_Type2(h);
```

The plots in Figure 7.30 show an acceptable bandpass filter design. □

□ **EXAMPLE 7.18** Design the following highpass filter:

$$\text{Stopband edge: } \omega_s = 0.6\pi \;\; A_s = 50 \text{ dB}$$

$$\text{Passband edge: } \omega_p = 0.8\pi \;\; R_p = 1 \text{ dB}$$

Solution Recall that for a highpass filter M must be odd (or Type-1 filter). Hence we will choose $M = 33$ to get two samples in the transition band. With this choice of M it is not possible to have frequency samples at ω_s and ω_p. The samples of the amplitude response are

$$H_r(k) = [\underbrace{0,\ldots,0}_{11}, T_1, T_2, \underbrace{1,\ldots,1}_{8}, T_2, T_1, \underbrace{0,\ldots,0}_{10}]$$

while the phase response samples are

$$\angle H(k) = \begin{cases} -\dfrac{33-1}{2}\dfrac{2\pi}{33}k = -\dfrac{32}{33}\pi k, & 0 \le k \le 16 \\ +\dfrac{32}{33}\pi(33-k), & 17 \le k \le 32 \end{cases}$$

The optimum values of transition samples are $T_1 = 0.1095$ and $T_2 = 0.598$. Using these values, the MATLAB design is given in the following script.

```
>> M = 33; alpha = (M-1)/2; l = 0:M-1; wl = (2*pi/M)*l;
>> T1 = 0.1095; T2 = 0.598;
>> Hrs = [zeros(1,11),T1,T2,ones(1,8),T2,T1,zeros(1,10)];
>> Hdr = [0,0,1,1]; wdl = [0,0.6,0.8,1];
>> k1 = 0:floor((M-1)/2); k2 = floor((M-1)/2)+1:M-1;
>> angH = [-alpha*(2*pi)/M*k1, alpha*(2*pi)/M*(M-k2)];
>> H = Hrs.*exp(j*angH);  h = real(ifft(H,M));
>> [db,mag,pha,grd,w] = freqz_m(h,1);  [Hr,ww,a,L] = Hr_Type1(h);
```

The time- and the frequency-domain plots of the design are shown in Figure 7.31. □

□ **EXAMPLE 7.19** Design a 33-point digital differentiator based on the ideal differentiator of (7.31) given in Example 7.12.

Solution From (7.31) the samples of the (imaginary-valued) amplitude response are given by

$$jH_r(k) = \begin{cases} +j\dfrac{2\pi}{M}k, & k = 0,\ldots,\left\lfloor\dfrac{M-1}{2}\right\rfloor \\ -j\dfrac{2\pi}{M}(M-k), & k = \left\lfloor\dfrac{M-1}{2}\right\rfloor+1,\ldots,M-1 \end{cases}$$

and for linear phase the phase samples are

$$\angle H(k) = \begin{cases} -\dfrac{M-1}{2}\dfrac{2\pi}{M}k = -\dfrac{M-1}{M}\pi k, & k = 0,\ldots,\left\lfloor\dfrac{M-1}{2}\right\rfloor \\ +\dfrac{M-1}{M}\pi(M-k), & k = \left\lfloor\dfrac{M-1}{2}\right\rfloor+1,\ldots,M-1 \end{cases}$$

Therefore

$$H(k) = jH_r(k)\,e^{j\angle H(k)}, \quad 0 \le k \le M-1 \qquad \text{and} \qquad h(n) = \text{IDFT}[H(k)]$$

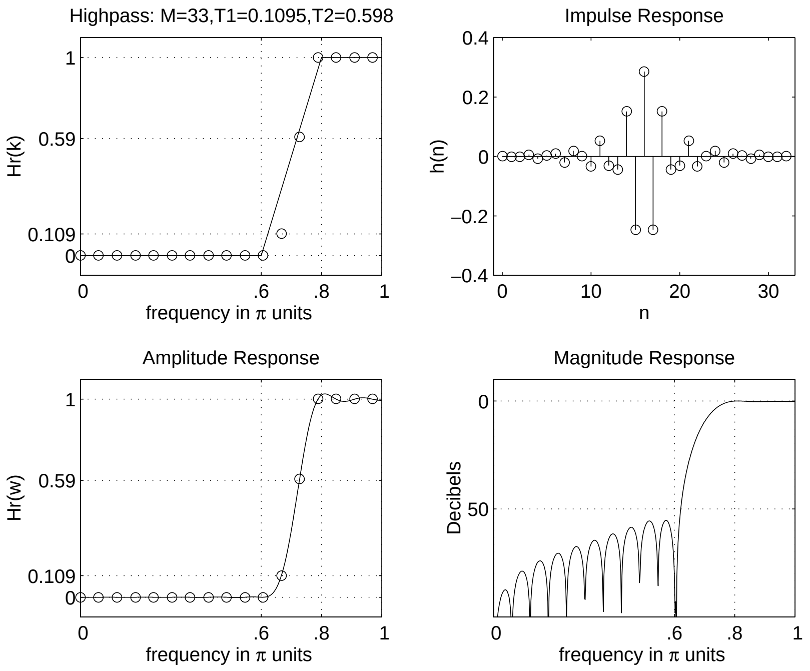


FIGURE 7.31 *Highpass filter design plots in Example 7.18*

MATLAB script:

```
>> M = 33; alpha = (M-1)/2; Dw = 2*pi/M;  l = 0:M-1; wl = Dw*l;
>> k1 = 0:floor((M-1)/2); k2 = floor((M-1)/2)+1:M-1;
>> Hrs = [j*Dw*k1,-j*Dw*(M-k2)];
>> angH = [-alpha*Dw*k1, alpha*Dw*(M-k2)];
>> H = Hrs.*exp(j*angH);  h = real(ifft(H,M));  [Hr,ww,a,P]=Hr_Type3(h);
```

The time- and the frequency-domain plots are shown in Figure 7.32. We observe that the differentiator is not a full-band differentiator. □

□ **EXAMPLE 7.20** Design a 51-point digital Hilbert transformer based on the ideal Hilbert transformer of (7.32).

Solution From (7.32) the samples of the (imaginary-valued) amplitude response are given by

$$jH_r(k) = \begin{cases} -j, & k = 1, \ldots, \left\lfloor \dfrac{M-1}{2} \right\rfloor \\ 0, & k = 0 \\ +j, & k = \left\lfloor \dfrac{M-1}{2} \right\rfloor + 1, \ldots, M-1 \end{cases}$$

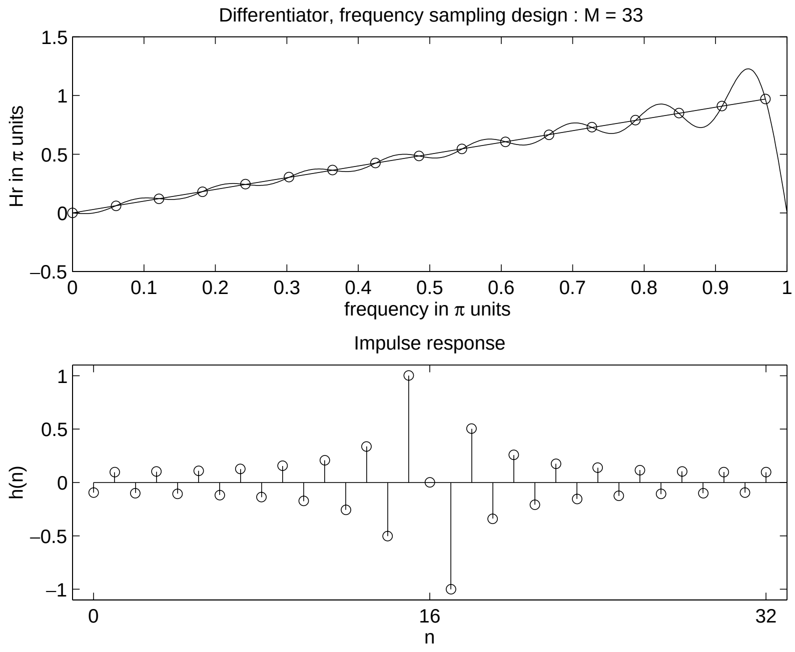


FIGURE 7.32 *Differentiator design plots in Example 7.19*

Since this is a Type-3 linear-phase filter, the amplitude response will be zero at $\omega = \pi$. Hence to reduce the ripples, we should choose the two samples (in transition bands) near $\omega = \pi$ optimally between 0 and j. Using our previous experience, we could select this value as $0.39j$. The samples of the phase response are selected similar to those in Example 7.19.

MATLAB script:

```
>> M = 51; alpha = (M-1)/2; Dw = 2*pi/M;  l = 0:M-1; wl = Dw*l;
>> k1 = 0:floor((M-1)/2); k2 = floor((M-1)/2)+1:M-1;
>> Hrs = [0,-j*ones(1,(M-3)/2),-0.39j,0.39j,j*ones(1,(M-3)/2)];
>> angH = [-alpha*Dw*k1, alpha*Dw*(M-k2)];
>> H = Hrs.*exp(j*angH);  h = real(ifft(H,M));  [Hr,ww,a,P]=Hr_Type3(h);
```

The plots in Figure 7.33 show the effect of the transition band samples. □

The SP toolbox provides a function called `fir2` which combines frequency sampling technique with the window technique to design arbitrary shaped magnitude response FIR filters. After computing filter impulse response using the naive design method, `fir2` then applies a selected

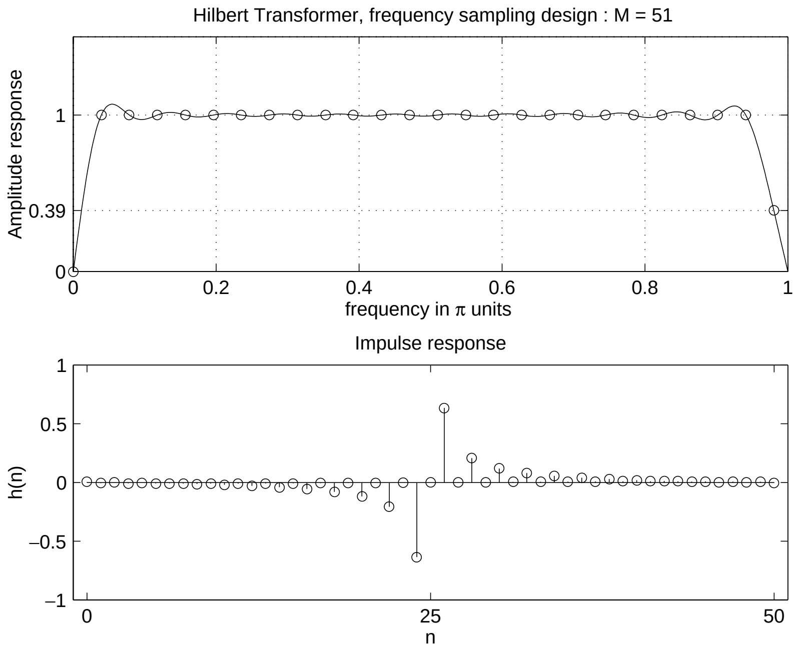


FIGURE 7.33 *Digital Hilbert transformer design plots in Example 7.20*

window to minimize ripples near the band-edge frequencies. This function's syntax also has several forms including:

- `h = fir2(N,f,m)` designs an Nth-order ($N = M-1$) lowpass FIR filter and returns the impulse response in vector `h`. The desired magnitude response of the filter is supplied in vectors `f` and `m`, which must be of the same length. The vector `f` contains normalized frequencies in the range from 0 to 1, where 1 corresponds to π rad/sample. The first value of `f` must be 0 and the last value 1. The vector `m`, contains the samples of the desired magnitude response at the values specified in `f`. The desired frequency response is then interpolated onto a dense, evenly spaced grid of length 512. Thus, this syntax corresponds to the naive design method.
- `h = fir2(N,f,m,window)` uses the vector `window` of length `N+1` obtained from one of the specified MATLAB window function. The default window function used is the Hamming window.
- `h = fir2(N,f,m,npt)` or `h = fir2(N,f,m,npt,window)` specifies the number of points, `npt`, for the grid onto which `fir2` interpolates the frequency response. The default `npt` value is 512.

Note that the `fir2` does not implement the classic optimum frequency sampling method. By incorporating window design, `fir2` has found an alternative (and somewhat clever) approach to do away with the optimum transition band values and the associated tables. By densely sampling values in the entire band, interpolation errors are reduced (but not minimized), and stopband attenuation is increased to an acceptable level. However, the basic design is contaminated by the window operation; hence, the frequency response does not go through the original sampled values. It is more suitable for designing FIR filters with arbitrary shaped frequency responses.

The type of frequency sampling filter that we considered is called a Type-A filter, in which the sampled frequencies are

$$\omega_k = \frac{2\pi}{M}k, \quad 0 \le k \le M-1$$

There is a second set of uniformly spaced samples given by

$$\omega_k = \frac{2\pi\left(k+\frac{1}{2}\right)}{M}, \quad 0 \le k \le M-1$$

This is called a Type-B filter, for which a frequency sampling structure is also available. The expressions for the magnitude response $H(e^{j\omega})$ and the impulse response $h(n)$ are somewhat more complicated and are available in Proakis and Manolakis [23]. Their design can also be done in MATLAB using the approach discussed in this section.

7.5 OPTIMAL EQUIRIPPLE DESIGN TECHNIQUE

The last two techniques—namely, the window design and the frequency sampling design—were easy to understand and implement. However, they have some disadvantages. First, we cannot specify the band frequencies ω_p and ω_s precisely in the design; that is, we have to accept whatever values we obtain after the design. Second, we cannot specify both δ_1 and δ_2 ripple factors simultaneously. Either we have $\delta_1 = \delta_2$ in the window design method, or we can optimize only δ_2 in the frequency sampling method. Finally, the approximation error—that is, the difference between the ideal response and the actual response—is not uniformly distributed over the band intervals. It is higher near the band edges and smaller in the regions away from band edges. By distributing the error uniformly, we can obtain a lower-order filter satisfying the same specifications. Fortunately, a technique exists that can eliminate these three problems. This technique is somewhat difficult to understand and requires a computer for its implementation.

For linear-phase FIR filters, it is possible to derive a set of conditions for which it can be proved that the design solution is optimal in the sense of *minimizing the maximum approximation error* (sometimes called the *minimax* or the *Chebyshev* error). Filters that have this property are called *equiripple* filters because the approximation error is uniformly distributed in both the passband and the stopband. This results in lower-order filters.

In the following we first formulate a minimax optimal FIR design problem and discuss the total number of maxima and minima (collectively called *extrema*) that one can obtain in the amplitude response of a linear-phase FIR filter. Using this, we then discuss a general equiripple FIR filter design algorithm, which uses polynomial interpolation for its solution. This algorithm is known as the Parks-McClellan algorithm, and it incorporates the Remez exchange algorithm for polynomial solution. This algorithm is available as a subroutine on many computing platforms. In this section we will use MATLAB to design equiripple FIR filters.

7.5.1 DEVELOPMENT OF THE MINIMAX PROBLEM

Earlier in this chapter we showed that the frequency response of the four cases of linear-phase FIR filters can be written in the form

$$H(e^{j\omega}) = e^{j\beta}e^{-j\frac{M-1}{2}\omega}H_r(w)$$

where the values for β and the expressions for $H_r(\omega)$ are given in Table 7.2.

TABLE 7.2 *Amplitude response and β-values for linear-phase FIR filters*

Linear-phase FIR Filter Type	β	$H_r(e^{j\omega})$
Type-1: M odd, symmetric $h(n)$	0	$\sum_{0}^{(M-1)/2} a(n)\cos\omega n$
Type-2: M even, symmetric $h(n)$	0	$\sum_{1}^{M/2} b(n)\cos[\omega(n-1/2)]$
Type-3: M odd, antisymmetric $h(n)$	$\frac{\pi}{2}$	$\sum_{1}^{(M-1)/2} c(n)\sin\omega n$
Type-4: M even, antisymmetric $h(n)$	$\frac{\pi}{2}$	$\sum_{1}^{M/2} d(n)\sin[\omega(n-1/2)]$

Using simple trigonometric identities, each expression for $H_r(\omega)$ can be written as a product of a fixed function of ω (call this $Q(\omega)$) and a function that is a sum of cosines (call this $P(\omega)$). For details see Proakis and Manolakis [23] and Problems P7.2–P7.5. Thus

$$H_r(\omega) = Q(\omega)P(\omega) \tag{7.40}$$

where $P(\omega)$ is of the form

$$P(\omega) = \sum_{n=0}^{L} \alpha(n) \cos \omega n \tag{7.41}$$

and $Q(\omega)$, L, $P(\omega)$ for the four cases are given in Table 7.3.

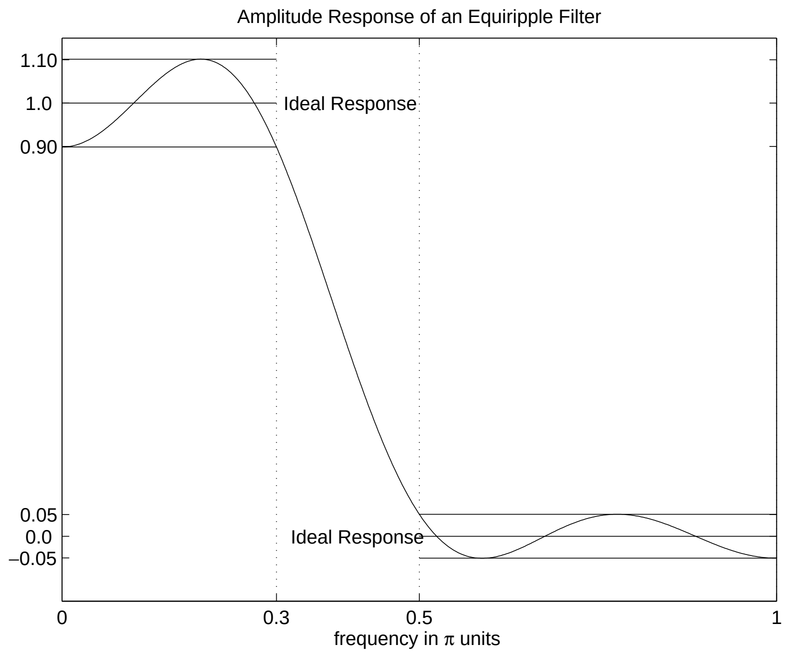
TABLE 7.3 *$Q(\omega)$, L, and $P(\omega)$ for linear-phase FIR filters*

LP FIR Filter Type	$Q(\omega)$	L	$P(\omega)$
Type-1	1	$\frac{M-1}{2}$	$\sum_0^L a(n) \cos \omega n$
Type-2	$\cos \frac{\omega}{2}$	$\frac{M}{2} - 1$	$\sum_0^L \tilde{b}(n) \cos \omega n$
Type-3	$\sin \omega$	$\frac{M-3}{2}$	$\sum_0^L \tilde{c}(n) \cos \omega n$
Type-4	$\sin \frac{\omega}{2}$	$\frac{M}{2} - 1$	$\sum_0^L \tilde{d}(n) \cos \omega n$

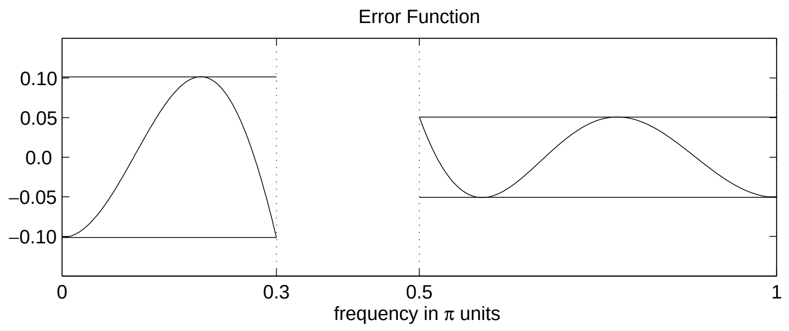
The purpose of the previous analysis was to have a *common form* for $H_r(\omega)$ across all four cases. It makes the problem formulation much easier. To formulate our problem as a Chebyshev approximation problem, we have to define the desired amplitude response $H_{dr}(\omega)$ and a weighting function $W(\omega)$, both defined over passbands and stopbands. The weighting function is necessary so that we can have an independent control over δ_1 and δ_2. The weighted error is defined as

$$E(\omega) \triangleq W(\omega)\left[H_{dr}(\omega) - H_r(\omega)\right], \quad \omega \in \mathcal{S} \triangleq [0, \omega_p] \cup [\omega_s, \pi] \tag{7.42}$$

These concepts are made clear in the following set of figures. It shows a typical equiripple filter response along with its ideal response.



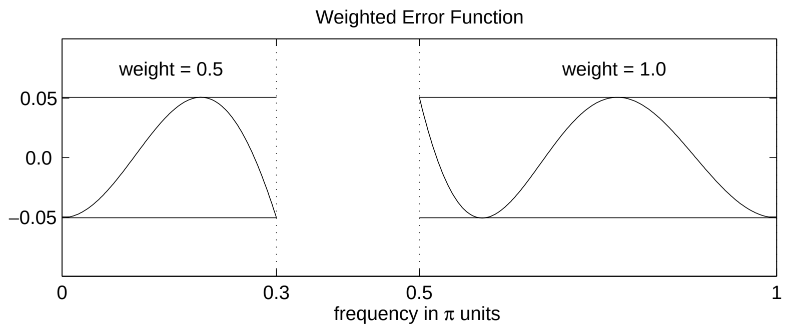
The error $[H_{dr}(\omega) - H_r(\omega)]$ response is shown here.



Now if we choose

$$W(\omega) = \begin{cases} \dfrac{\delta_2}{\delta_1}, & \text{in the passband} \\ 1, & \text{in the stopband} \end{cases} \tag{7.43}$$

Then the weighted error $E(\omega)$ response is



Thus the maximum error in both the passband and stopband is δ_2. Therefore, if we succeed in minimizing the maximum weighted error to δ_2, we automatically also satisfy the specification in the passband to δ_1. Substituting $H_r(\omega)$ from (7.40) into (7.42), we obtain

$$\begin{aligned} E(\omega) &= W(\omega)\left[H_{dr}(\omega) - Q(\omega)P(\omega)\right] \\ &= W(\omega)Q(\omega)\left[\frac{H_{dr}(\omega)}{Q(\omega)} - P(\omega)\right], \quad \omega \in \mathcal{S} \end{aligned}$$

If we define

$$\hat{W}(\omega) \triangleq W(\omega)Q(w) \qquad \text{and} \qquad \hat{H}_{dr}(\omega) \triangleq \frac{H_{dr}(\omega)}{Q(\omega)}$$

then we obtain

$$E(\omega) = \hat{W}(\omega)\left[\hat{H}_{dr}(\omega) - P(\omega)\right], \quad \omega \in \mathcal{S} \tag{7.44}$$

Thus we have a common form of $E(\omega)$ for all four cases.

Problem statement The Chebyshev approximation problem can now be defined as:

Determine the set of coefficients $a(n)$ or $\tilde{b}(n)$ or $\tilde{c}(n)$ or $\tilde{d}(n)$ [or equivalently $a(n)$ or $b(n)$ or $c(n)$ or $d(n)$] to minimize the maximum absolute value of $E(\omega)$ over the passband and stopband, i.e.,

$$\min_{\text{over coeff.}}\left[\max_{\omega\in S}|E(\omega)|\right] \tag{7.45}$$

Now we have succeeded in specifying the exact ω_p, ω_s, δ_1, and δ_2. In addition the error can now be distributed uniformly in both the passband and stopband.

7.5.2 CONSTRAINT ON THE NUMBER OF EXTREMA

Before we give the solution to this above problem, we will first discuss the issue: how many local maxima and minima exist in the error function $E(\omega)$ for a given M-point filter? This information is used by the Parks-McClellan algorithm to obtain the polynomial interpolation. The answer is in the expression $P(\omega)$. From (7.41) $P(\omega)$ is a trigonometric function in ω. Using trigonometric identities of the form

$$\begin{aligned}\cos(2\omega) &= 2\cos^2(\omega) - 1\\ \cos(3\omega) &= 4\cos^3(\omega) - 3\cos(\omega)\\ \vdots\ &= \ \vdots\end{aligned}$$

$P(\omega)$ can be converted to a trigonometric polynomial in $\cos(\omega)$, which we can write (7.41) as

$$P(\omega) = \sum_{n=0}^{L} \beta(n)\cos^n \omega \tag{7.46}$$

□ **EXAMPLE 7.21** Let $h(n) = \frac{1}{15}[1,2,3,4,3,2,1]$. Then $M = 7$ and $h(n)$ is symmetric, which means that we have a Type-1 linear-phase filter. Hence $L = (M-1)/2 = 3$. Now from (7.7)

$$\alpha(n) = a(n) = 2h(3-n), \quad 1 \le n \le 2; \qquad \text{and} \qquad \alpha(0) = a(0) = h(3)$$

or $\alpha(n) = \frac{1}{15}[4,6,4,2]$. Hence

$$\begin{aligned}P(\omega) &= \sum_{0}^{3} \alpha(n)\cos\omega n = \tfrac{1}{15}(4 + 6\cos\omega + 4\cos 2\omega + 2\cos 3\omega)\\ &= \tfrac{1}{15}\left\{4 + 6\cos\omega + 4(2\cos^2\omega - 1) + 2(4\cos^3\omega - 3\cos\omega)\right\}\\ &= 0 + 0 + \tfrac{8}{15}\cos^2\omega + \tfrac{8}{15}\cos^3\omega = \sum_{0}^{3}\beta(n)\cos^n\omega\end{aligned}$$

or $\beta(n) = \left[0, 0, \dfrac{8}{15}, \dfrac{8}{15}\right]$.

From (7.46) we note that $P(\omega)$ is an Lth-order polynomial in $\cos(\omega)$. Since $\cos(\omega)$ is a *monotone* function in the *open* interval $0 < \omega < \pi$, then it follows that the Lth-order polynomial $P(\omega)$ in $\cos(\omega)$ should behave like an ordinary Lth-order polynomial $P(x)$ in x.Therefore $P(\omega)$ has *at most* (i.e., no more than) $(L-1)$ local extrema in the open interval $0 < \omega < \pi$. For example,

$$\cos^2(\omega) = \frac{1 + \cos 2\omega}{2}$$

has only one minimum at $\omega = \pi/2$. However, it has three extrema in the closed interval $0 \le \omega \le \pi$ (i.e., a maximum at $\omega = 0$, a minimum at $\omega = \pi/2$, and

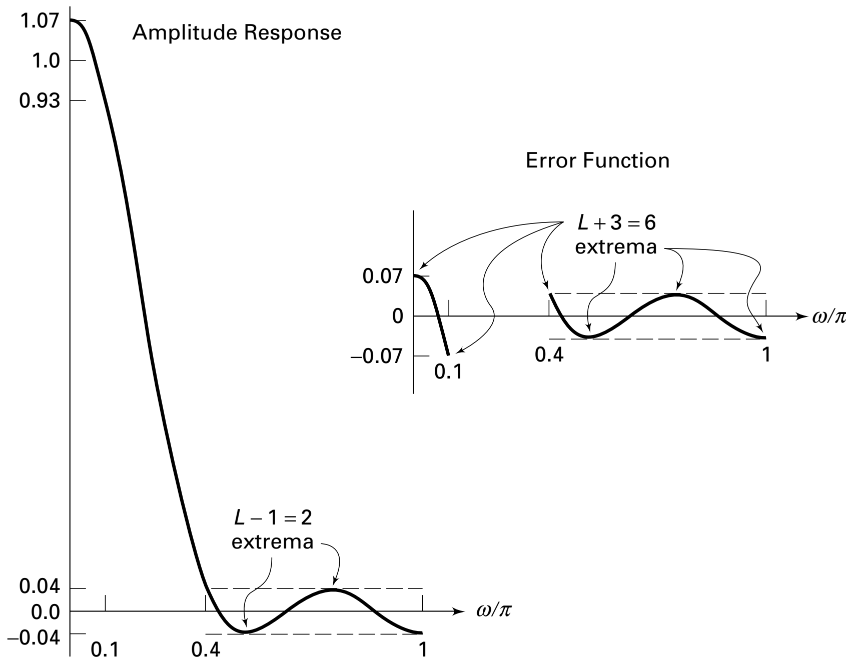


FIGURE 7.34 *Amplitude response and the error function in Example 7.22*

a maximum at $\omega = \pi$). Now if we include the end points $\omega = 0$ and $\omega = \pi$, then $P(\omega)$ has at most $(L + 1)$ local extrema in the closed interval $0 \le \omega \le \pi$. Finally, we would like the filter specifications to be met exactly at band edges ω_p and ω_s. Then the specifications can be met at no more than $(L + 3)$ extremal frequencies in the $0 \le \omega \le \pi$ interval.

Conclusion The error function $E(\omega)$ has at most $(L + 3)$ extrema in $\mathcal{S}$. □

□ **EXAMPLE 7.22** Let us plot the amplitude response of the filter given in Example 7.21 and count the total number of extrema in the corresponding error function.

Solution The impulse response is

$$h(n) = \frac{1}{15}[1, 2, 3, 4, 3, 2, 1], \quad M = 7 \qquad \text{or} \qquad L = 3$$

and $\alpha(n) = \frac{1}{15}[4, 6, 4, 2]$ and $\beta(n) = \left[0, 0, \frac{8}{15}, \frac{8}{15}\right]$ from Example 7.21. Hence

$$P(\omega) = \frac{8}{15}\cos^2\omega + \frac{8}{15}\cos^3\omega$$

which is shown in Figure 7.34. Clearly, $P(\omega)$ has $(L - 1) = 2$ extrema in the open interval $0 < \omega < \pi$. Also shown in Figure 7.34 is the error function, which has $(L + 3) = 6$ extrema. □

Let us now turn our attention to the problem statement and equation (7.45). It is a well-known problem in *approximation theory*, and the solution is given by the following important theorem.

■ THEOREM 1 ***Alternation Theorem***

Let $\mathcal{S}$ be any closed subset of the closed interval $[0, \pi]$. In order that $P(\omega)$ be the unique minimax approximation to $H_{dr}(\omega)$ on $\mathcal{S}$, it is necessary and sufficient that the error function $E(\omega)$ exhibit at least $(L+2)$ "alternations" or extremal frequencies in $\mathcal{S}$; that is, there must exist $(L+2)$ frequencies ω_i in $\mathcal{S}$ such that

$$\begin{aligned} E(\omega_i) = -E(\omega_{i-1}) &= \pm \max_{S} |E(\omega)| \\ &\triangleq \pm\delta, \forall\, \omega_0 < \omega_1 < \cdots < \omega_{L+1} \in \mathcal{S} \end{aligned} \tag{7.47}$$

Combining this theorem with our earlier conclusion, we infer that the optimal equiripple filter has either $(L+2)$ or $(L+3)$ alternations in its error function over $\mathcal{S}$. Most of the equiripple filters have $(L+2)$ alternations. However, for some combinations of ω_p and ω_s, we can get filters with $(L+3)$ alternations. These filters have one extra ripple in their response and hence are called *Extraripple* filters.

7.5.3 PARKS-McCLELLAN ALGORITHM

The alternation theorem ensures that the solution to our minimax approximation problem exists and is unique, but it does not tell us how to obtain this solution. We know neither the order M (or equivalently, L), nor the extremal frequencies ω_i, nor the parameters $\{\alpha(n)\}$, nor the maximum error δ. Parks and McClellan [20] provided an iterative solution using the Remez exchange algorithm. It assumes that the filter length M (or L) and the ratio δ_2/δ_1 are known. If we choose the weighting function as in (7.43), and if we choose the order M correctly, then $\delta = \delta_2$ when the solution is obtained. Clearly, δ and M are related; the larger the M, the smaller the δ. In the filter specifications δ_1, δ_2, ω_p, and ω_s are given. Therefore M has to be assumed. Fortunately, a simple formula, due to Kaiser, exists for approximating M. It is given by

$$\hat{M} = \frac{-20\log_{10}\sqrt{\delta_1\delta_2} - 13}{2.285\Delta\omega} + 1; \quad \Delta\omega = \omega_s - \omega_p \tag{7.48}$$

The Parks-McClellan algorithm begins by guessing $(L+2)$ extremal frequencies $\{\omega_i\}$ and estimating the maximum error δ at these frequencies. It then fits an Lth-order polynomial (7.46) through points given in (7.47).

Local maximum errors are determined over a finer grid, and the extremal frequencies $\{\omega_i\}$ are adjusted at these new extremal values. A new Lth-order polynomial is fit through these new frequencies, and the procedure is repeated. This iteration continues until the optimum set $\{\omega_i\}$ and the global maximum error δ are found. The iterative procedure is guaranteed to converge, yielding the polynomial $P(\omega)$. From (7.46) coefficients $\beta(n)$ are determined. Finally, the coefficients $a(n)$ as well as the impulse response $h(n)$ are computed. This algorithm is available in MATLAB as the `firpm` function, which is described shortly.

Since we approximated M, the maximum error δ may not be equal to δ_2. If this is the case, then we have to increase M (if $\delta > \delta_2$) or decrease M (if $\delta < \delta_2$) and use the `firpm` algorithm again to determine a new δ. We repeat this procedure until $\delta \leq \delta_2$. The optimal equiripple FIR filter, which satisfies all the three requirements discussed earlier, is now determined.

7.5.4 MATLAB IMPLEMENTATION

The Parks-McClellan algorithm is available in MATLAB as a function called `firpm`, the most general syntax of which is

```
[h] = firpm(N,f,m,weights,ftype)
```

There are several versions of this syntax

- `[h] = firpm(N,f,m)` designs an `N`th-order (note that the length of the filter is $M = N + 1$) FIR digital filter whose frequency response is specified by the arrays `f` and `m`. The filter coefficients (or the impulse response) are returned in array `h` of length M. The array `f` contains band-edge frequencies in units of π, that is, $0.0 \leq$ `f` ≤ 1.0. These frequencies must be in increasing order, starting with 0.0 and ending with 1.0. The array `m` contains the desired magnitude response at frequencies specified in `f`. The lengths of `f` and `m` arrays must be the same and must be an even number. The weighting function used in each band is equal to unity, which means that the tolerances (δ_i's) in every band are the same.
- `[h] = firpm(N,f,m,weights)` is similar to the preceding case except that the array `weights` specifies the weighting function in each band.
- `[h] = firpm(N,f,m,ftype)` is similar to the first case except when `ftype` is the string '`differentiator`' or '`hilbert`', it designs digital differentiators or digital Hilbert transformers, respectively. For the digital Hilbert transformer, the lowest frequency in the `f` array should not be 0, and the highest frequency should not be 1. For the digital

differentiator, the m vector does not specify the desired slope in each band but the desired magnitude.

- [h] = firpm(N,f,m,weights,ftype) is similar to the above case except that the array weights specifies the weighting function in each band.

To estimate the filter order N, the SP toolbox provides the function firpmord, which also estimates other parameters that can be used in the firpm function. The basic syntax is

```
[N,f0,m0,weights] = firpmord(f,m,delta);
```

The function computes the window order N, the normalized frequency band edges in f0, amplitude response in a0, and the band weights in weights. The vector f is a vector of normalized band edges and m is a vector specifying the desired amplitude on the bands defined by f. The length of f is two less than twice the length of m; i.e., f does not contain 0 or 1. The vector delta specifies tolerances in each band (not in decibels). The estimated parameters can now be used in the firpm function.

As explained during the description of the Parks-McClellan algorithm, we have to first guess the order of the filter using (7.48) to use the function firpm. After we obtain the filter coefficients in array h, we have to check the minimum stopband attenuation and compare it with the given A_s and then increase (or decrease) the filter order. We have to repeat this procedure until we obtain the desired A_s. We illustrate this procedure in the following several MATLAB examples. These examples also use the ripple conversion function db2delta, which is developed in Problem P7.1.

□ **EXAMPLE 7.23** Let us design the lowpass filter described in Example 7.8 using the Parks-McClellan algorithm. The design parameters are

$$\omega_p = 0.2\pi\ , \quad R_p = 0.25 \text{ dB}$$
$$\omega_s = 0.3\pi\ , \quad A_s = 50 \text{ dB}$$

We provide a MATLAB script to design this filter.

```
>> wp = 0.2*pi; ws = 0.3*pi; Rp = 0.25; As = 50;
>> [delta1,delta2] = db2delta(Rp,As);
>> [N,f,m,weights] = firpmord([wp,ws]/pi,[1,0],[delta1,delta2]);
>> h = firpm(N,f,m,weights);
>> [db,mag,pha,grd,w] = freqz_m(h,[1]);
>> delta_w = 2*pi/1000; wsi=ws/delta_w+1; wpi = wp/delta_w;
>> Asd = -max(db(wsi:1:501))
Asd = 47.8404
```

```
>> N = N+1
N = 43
>> h = firpm(N,f,m,weights); [db,mag,pha,grd,w] = freqz_m(h,[1]);
>> Asd = -max(db(wsi:1:501))
Asd = 48.2131
>> N = N+1
N =  44
>> h = firpm(N,f,m,weights); [db,mag,pha,grd,w] = freqz_m(h,[1]);
>> Asd = -max(db(wsi:1:501))
Asd = 48.8689
>> N = N+1
N = 45
>> h = firpm(N,f,m,weights); [db,mag,pha,grd,w] = freqz_m(h,[1]);
>> Asd = -max(db(wsi:1:501))
Asd = 49.8241
>> N = N+1
N = 46
>> h = firpm(N,f,m,weights); [db,mag,pha,grd,w] = freqz_m(h,[1]);
>> Asd = -max(db(wsi:1:501))
Asd = 51.0857
>> M = N+1
M = 47
```

Note that we stopped this iterative procedure when the computed stopband attenuation exceeded the given stopband attenuation A_s, and the optimal value of M was found to be 47. This value is considerably lower than the window design techniques ($M = 61$ for a Kaiser window) or the frequency sampling technique ($M = 60$). In Figure 7.35 we show the time- and the frequency-domain plots of the designed filter along with the error function in both the passband and the stopband to illustrate the equiripple behavior.

□ **EXAMPLE 7.24** Let us design the bandpass filter described in Example 7.10 using the Parks-McClellan algorithm. The design parameters are:

$$\begin{array}{l} \omega_{1s} = 0.2\pi \\ \omega_{1p} = 0.35\pi \end{array} ; \quad R_p = 1 \text{ dB}$$

$$\begin{array}{l} \omega_{2p} = 0.65\pi \\ \omega_{2s} = 0.8\pi \end{array} ; \quad A_s = 60 \text{ db}$$

Solution The following MATLAB script shows how to design this filter.

```
>> ws1 = 0.2*pi; wp1 = 0.35*pi; wp2 = 0.65*pi; ws2 = 0.8*pi;
>> Rp = 1.0; As = 60;
>> [delta1,delta2] = db2delta(Rp,As);
>> f = [ws1,wp1,wp2,ws2]/pi; m = [0,1,0]; delta = [delta2,delta1,delta2];
>> [N,f,m,weights] = firpmord(f,m,delta); N
N = 26
>> h = firpm(N,f,m,weights);
>> [db,mag,pha,grd,w] = freqz_m(h,[1]);
>> delta_w=2*pi/1000;
>> ws1i=floor(ws1/delta_w)+1; wp1i = floor(wp1/delta_w)+1;
>> ws2i=floor(ws2/delta_w)+1; wp2i = floor(wp2/delta_w)+1;
>> Asd = -max(db(1:1:ws1i))
Asd = 54.7756
>> N = N+1;
>> h = firpm(N,f,m,weights);
>> [db,mag,pha,grd,w] = freqz_m(h,[1]);
>> Asd = -max(db(1:1:ws1i))
Asd = 56.5910
>> N = N+1;
>> h = firpm(N,f,m,weights);
>> [db,mag,pha,grd,w] = freqz_m(h,[1]);
Asd = -max(db(1:1:ws1i))
>> Asd = 61.2843
>> M = N+1
M = 29
```

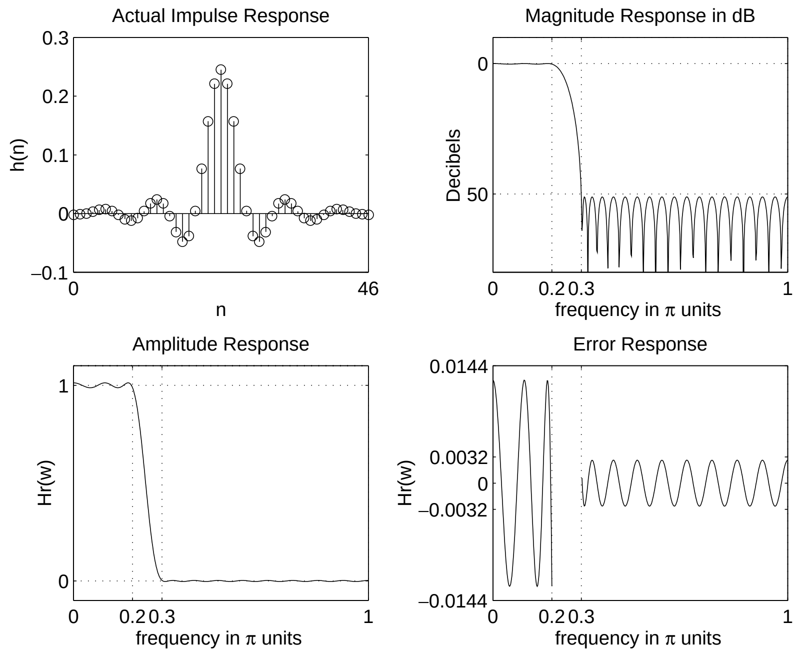


FIGURE 7.35 *Plots for equiripple lowpass FIR filter in Example 7.23*

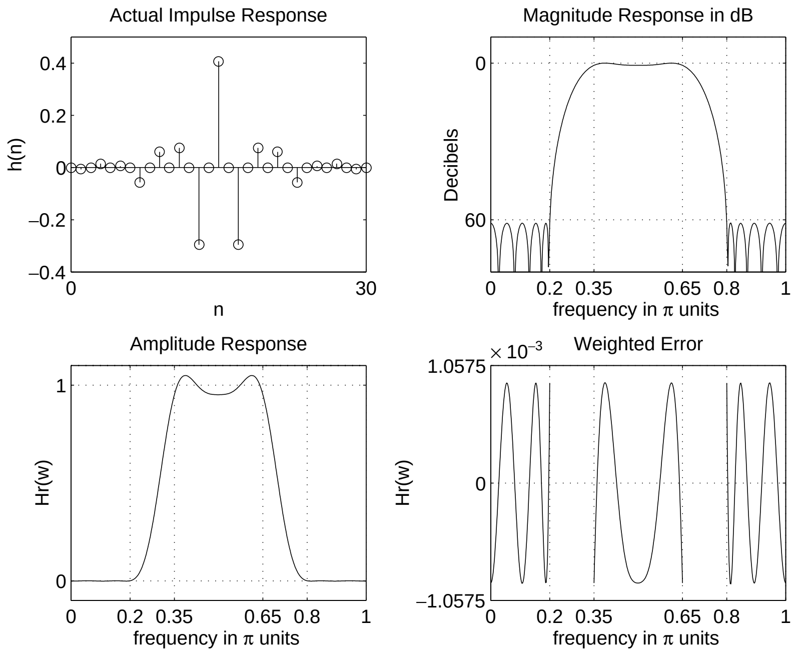


FIGURE 7.36 *Plots for equiripple bandpass FIR filter in Example 7.24*

The optimal value of M was found to be 29. The time- and the frequency-domain plots of the designed filter are shown in Figure 7.36. □

□ **EXAMPLE 7.25** Design a highpass filter that has the following specifications:

$$\omega_s = 0.6\pi, \quad A_s = 50 \text{ dB}$$
$$\omega_p = 0.75\pi, \quad R_p = 0.5 \text{ dB}$$

Solution Since this is a highpass filter, we must ensure that the length M is an odd number. This is shown in the following MATLAB script.

```
>> ws = 0.6*pi; wp = 0.75*pi; Rp = 0.5; As = 50;
>> [delta1,delta2] = db2delta(Rp,As);
>> [N,f,m,weights] = firpmord([ws,wp]/pi,[0,1],[delta2,delta1]); N
N = 26
>> h = firpm(N,f,m,weights);
>> [db,mag,pha,grd,w] = freqz_m(h,[1]);
>> delta_w = 2*pi/1000; wsi=ws/delta_w; wpi = wp/delta_w;
>> Asd = -max(db(1:1:wsi))
Asd = 49.5918
>> N = N+2;
>> h = firpm(N,f,m,weights);
>> [db,mag,pha,grd,w] = freqz_m(h,[1]);
>> Asd = -max(db(1:1:wsi))
>> Asd = 50.2253
>> M = N+1
M = 29
```

Note also that we increased the value of N by two to maintain its even value. The optimum M was found to be 29. The time- and the frequency-domain plots of the designed filter are shown in Figure 7.37. □

□ **EXAMPLE 7.26** In this example we will design a "staircase" filter, which has 3 bands with different ideal responses and different tolerances in each band. The design specifications are

$$\begin{aligned}
&\text{Band-1:} && 0 \le \omega \le 0.3\pi, && \text{Ideal gain} = 1, && \text{Tolerance } \delta_1 = 0.01\\
&\text{Band-2:} && 0.4\pi \le \omega \le 0.7\pi, && \text{Ideal gain} = 0.5, && \text{Tolerance } \delta_2 = 0.005\\
&\text{Band-3:} && 0.8\pi \le \omega \le \pi, && \text{Ideal gain} = 0, && \text{Tolerance } \delta_3 = 0.001
\end{aligned}$$

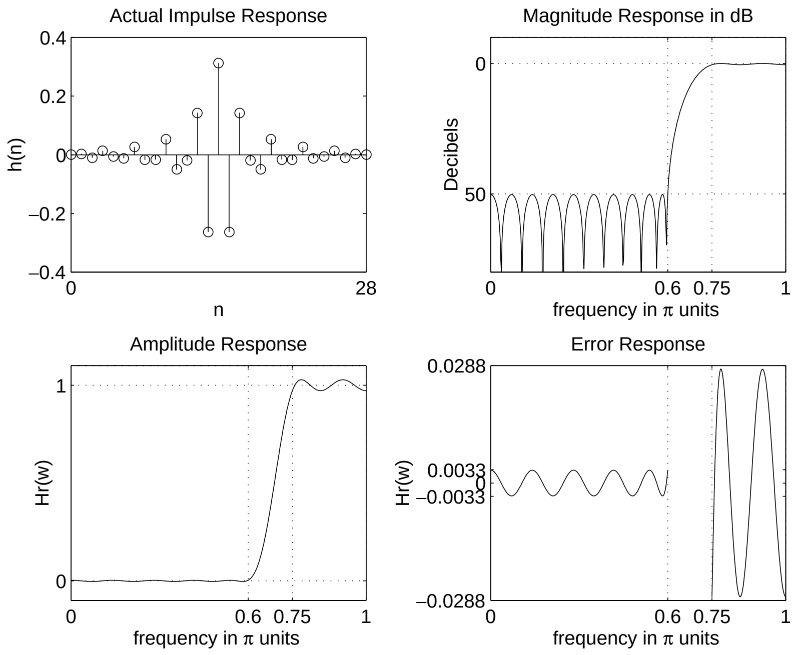


FIGURE 7.37 *Plots for equiripple highpass FIR filter in Example 7.25*

Solution The following MATLAB script describes the design procedure.

```
>> w1 = 0; w2 = 0.3*pi; delta1 = 0.01;
>> w3 = 0.4*pi; w4 = 0.7*pi; delta2 = 0.005;
>> w5 = 0.8*pi; w6 = pi; delta3 = 0.001;
>> weights = [delta3/delta1 delta3/delta2 1];
>> Dw = min((w3-w2), (w5-w3));
>> M = ceil((-20*log10((delta1*delta2*delta3)^(1/3))-13)/(2.285*Dw)+1)
>> M = 51
>> f = [0 w2/pi w3/pi w4/pi w5/pi 1];
>> m = [1 1 0.5 0.5 0 0];
>> h = firpm(M-1,f,m,weights);
>> [db,mag,pha,grd,w] = freqz_m(h,[1]);
>> delta_w = 2*pi/1000;
>> w1i=floor(w1/delta_w)+1; w2i = floor(w2/delta_w)+1;
>> w3i=floor(w3/delta_w)+1; w4i = floor(w4/delta_w)+1;
>> w5i=floor(w5/delta_w)+1; w6i = floor(w6/delta_w)+1;
>> Asd = -max(db(w5i:w6i))
Asd = 62.0745
>> M = M-1; h = firpm(M-1,f,m,weights);
>> [db,mag,pha,grd,w] = freqz_m(h,[1]);
>> Asd = -max(db(w5i:w6i))
Asd = 60.0299
>> M = M-1; h = firpm(M-1,f,m,weights);
>> [db,mag,pha,grd,w] = freqz_m(h,[1]);
>> Asd = -max(db(w5i:w6i))
Asd = 60.6068
>> M
M = 49
```

The time- and the frequency-domain plots of the designed filter are shown in Figure 7.38. □

□ **EXAMPLE 7.27** In this example we will design a digital differentiator with different slopes in each band. The specifications are

$$\begin{aligned}
&\text{Band-1:} && 0 \le \omega \le 0.2\pi, && \text{Slope} = 1 \text{ sam/cycle}\\
&\text{Band-2:} && 0.4\pi \le \omega \le 0.6\pi, && \text{Slope} = 2 \text{ sam/cycle}\\
&\text{Band-3:} && 0.8\pi \le \omega \le \pi, && \text{Slope} = 3 \text{ sam/cycle}
\end{aligned}$$

Solution We need desired magnitude response values in each band. These can be obtained by multiplying band-edge frequencies in cycles/sam by the slope values in sam/cycle

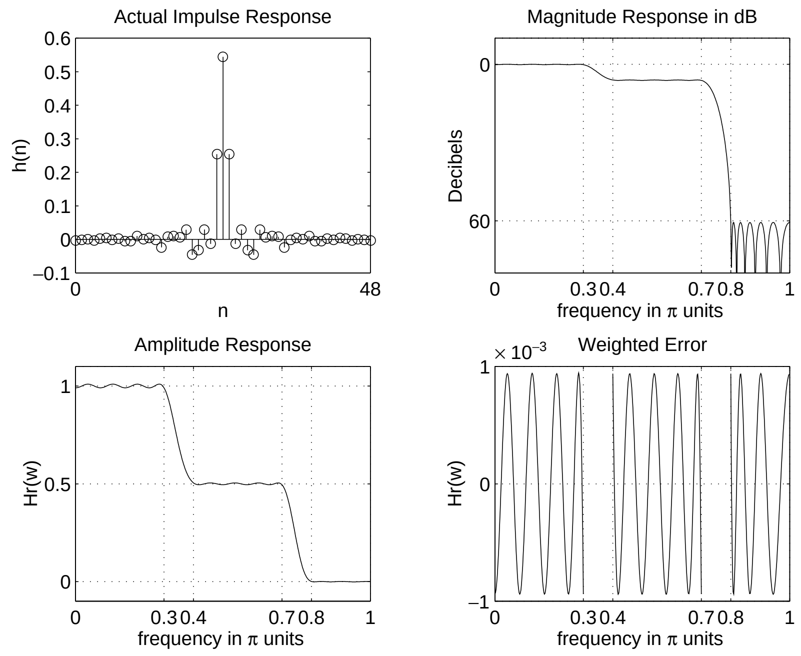


FIGURE 7.38 *Plots for equiripple staircase FIR filter in Example 7.26*

$$\text{Band-1:}\quad 0 \le f \le 0.1,\ \text{Slope} = 1\ \text{sam/cycle} \Rightarrow 0.0 \le |H| \le 0.1$$

$$\text{Band-2:}\ 0.2 \le f \le 0.3,\ \text{Slope} = 2\ \text{sam/cycle} \Rightarrow 0.4 \le |H| \le 0.6$$

$$\text{Band-3:}\ 0.4 \le f \le 0.5,\ \text{Slope} = 3\ \text{sam/cycle} \Rightarrow 1.2 \le |H| \le 1.5$$

Let the weights be equal in all bands. The MATLAB script is:

```
>> f = [0 0.2 0.4 0.6 0.8 1];          % in w/pi unis
>> m = [0,0.1,0.4,0.6,1.2,1.5];        % magnitude values
>> h = firpm(25,f,m,'differentiator');
>> [db,mag,pha,grd,w] = freqz_m(h,[1]);
>> subplot(2,1,1); stem([0:25],h); title('Impulse Response');
>> xlabel('n'); ylabel('h(n)'); axis([0,25,-0.6,0.6])
>> set(gca,'XTickMode','manual','XTick',[0,25])
>> set(gca,'YTickMode','manual','YTick',[-0.6:0.2:0.6]);
>> subplot(2,1,2); plot(w/(2*pi),mag); title('Magnitude Response')
>> xlabel('Normalized frequency f'); ylabel('|H|')
>> set(gca,'XTickMode','manual','XTick',f/2)
>> set(gca,'YTickMode','manual','YTick',[0,0.1,0.4,0.6,1.2,1.5]); grid
```

The frequency-domain response is shown in Figure 7.39. □

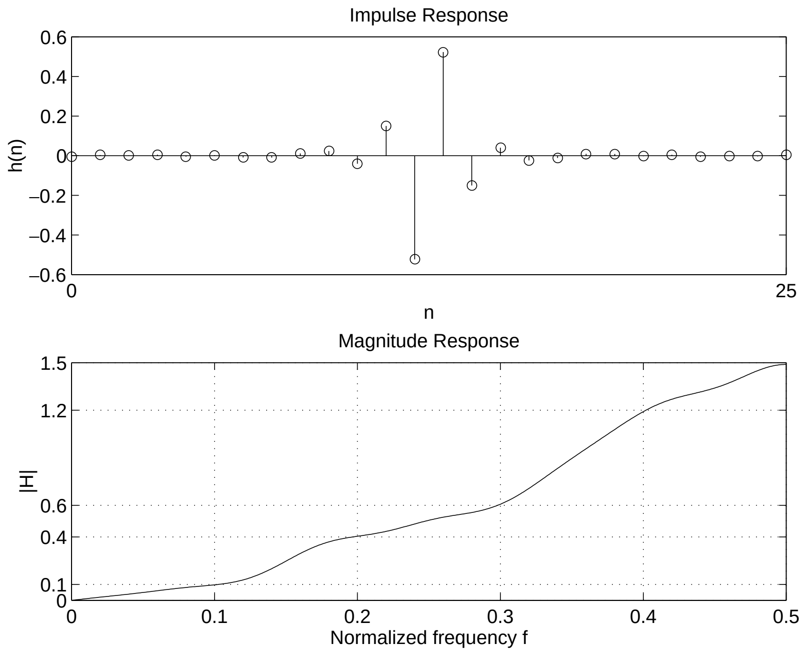


FIGURE 7.39 *Plots of the differentiator in Example 7.27*

☐ **EXAMPLE 7.28** Finally, we design a Hilbert transformer over the band $0.05\pi \le \omega \le 0.95\pi$.

Solution Since this is a wideband Hilbert transformer, we will choose an odd length for our filter (i.e., a Type-3 filter). Let us choose $M = 51$. The MATLAB script is:

```
>> f = [0.05,0.95]; m = [1 1];  h = firpm(50,f,m,'hilbert');
>> [db,mag,pha,grd,w] = freqz_m(h,[1]);
>> subplot(2,1,1); stem([0:50],h); title('Impulse Response');
>> xlabel('n'); ylabel('h(n)'); axis([0,50,-0.8,0.8])
>> set(gca,'XTickMode','manual','XTick',[0,50])
>> set(gca,'YTickMode','manual','YTick',[-0.8:0.2:0.8]);
>> subplot(2,1,2); plot(w/pi,mag); title('Magnitude Response')
>> xlabel('frequency in pi units'); ylabel('|H|')
>> set(gca,'XTickMode','manual','XTick',[0,f,1])
>> set(gca,'YTickMode','manual','YTick',[0,1]);grid
```

The plots of this Hilbert transformer are shown in Figure 7.40. ☐

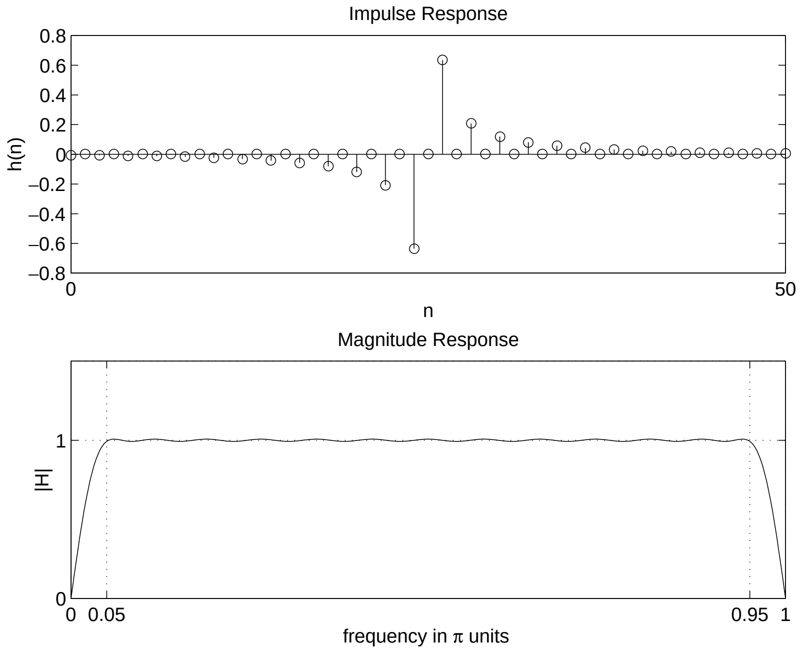


FIGURE 7.40 *Plots of the Hilbert transformer in Example 7.28*

7.6 PROBLEMS

P7.1 The absolute and relative (dB) specifications for a lowpass filter are related by (7.1) and (7.2). In this problem we will develop a simple MATLAB function to convert one set of specifications into another.

1. Write a MATLAB function to convert absolute specifications δ_1 and δ_2 into the relative specifications R_p and A_s in dB. The format of the function should be

```
function [Rp,As] = delta2db(delta1,delta2)
% Converts absolute specs delta1 and delta2 into dB specs Rp and As
% [Rp,As] = delta2db(delta1,delta2)
```

Verify your function using the specifications given in Example 7.2.

2. Write a MATLAB function to convert relative (dB) specifications R_p and A_s into the absolute specifications δ_1 and δ_2. The format of the function should be

```
function [delta1,delta2] = db2delta(Rp,As)
% Converts dB specs Rp and As into absolute specs delta1 and delta2
% [delta1,delta2] = db2delta(Rp,As)
```

Verify your function using the specifications given in Example 7.1.

P7.2 The Type-1 linear-phase FIR filter is characterized by

$$h(n) = h(M-1-n)), \quad 0 \le n \le M-1, \quad M \text{ odd}$$

Show that its amplitude response $H_r(\omega)$ is given by

$$H_r(\omega) = \sum_{n=0}^{L} a(n)\cos(\omega n), \quad L = \frac{M-1}{2}$$

where coefficients $\{a(n)\}$ are obtained as defined in (7.6).

P7.3 The Type-2 linear-phase FIR filter is characterized by

$$h(n) = h(M-1-n), \quad 0 \le n \le M-1, \quad M \text{ even}$$

1. Show that its amplitude response $H_r(\omega)$ is given by

$$H_r(\omega) = \sum_{n=1}^{M/2} b(n)\cos\left\{\omega\left(n-\tfrac{1}{2}\right)\right\}$$

   where coefficients $\{b(n)\}$ are obtained as defined in (7.10).
2. Show that $H_r(\omega)$ can be further expressed as

$$H_r(\omega) = \cos\left(\frac{\omega}{2}\right)\sum_{n=0}^{L} \tilde{b}(n)\cos(\omega n), \quad L = \frac{M}{2}-1$$

   where coefficients $\tilde{b}(n)$ are given by

$$\begin{aligned} b(1) &= \tilde{b}(0) + \tfrac{1}{2}\tilde{b}(1), \\ b(n) &= \frac{1}{2}\left[\tilde{b}(n-1) + \tilde{b}(n)\right], \quad 2 \le n \le \frac{M}{2}-1, \\ b\left(\tfrac{M}{2}\right) &= \frac{1}{2}\tilde{b}\left(\tfrac{M}{2}-1\right). \end{aligned}$$

P7.4 The Type-3 linear-phase FIR filter is characterized by

$$h(n) = -h(M-1-n), \quad 0 \le n \le M-1, \quad M \text{ odd}$$

1. Show that its amplitude response $H_r(\omega)$ is given by

$$H_r(\omega) = \sum_{n=1}^{(M-1)/2} c(n)\sin(\omega n)$$

   where coefficients $\{c(n)\}$ are obtained as defined in (7.13).
2. Show that $H_r(\omega)$ can be further expressed as

$$H_r(\omega) = \sin(\omega)\sum_{n=0}^{L} \tilde{c}(n)\cos(\omega n), \quad L = \frac{M-3}{2}$$

where coefficients $\tilde{c}(n)$ are given by

$$\begin{aligned} c(1) &= \tilde{c}(0) - \tfrac{1}{2}\tilde{c}(1), \\ c(n) &= \frac{1}{2}\left[\tilde{c}(n-1) - \tilde{c}(n)\right], \quad 2 \le n \le \frac{M-3}{2}, \\ c\left(\frac{M-1}{2}\right) &= \frac{1}{2}\tilde{c}\left(\frac{M-3}{2}\right). \end{aligned}$$

P7.5 The Type-4 linear-phase FIR filter is characterized by

$$h(n) = -h(M-1-n), \quad 0 \le n \le M-1, \quad M \text{ even}$$

1. Show that its amplitude response $H_r(\omega)$ is given by

$$H_r(\omega) = \sum_{n=1}^{M/2} d(n) \sin\left\{\omega\left(n - \tfrac{1}{2}\right)\right\}$$

where coefficients $\{d(n)\}$ are obtained as defined in (7.16).

2. Show that the above $H_r(\omega)$ can be further expressed as

$$H_r(\omega) = \sin\left(\frac{\omega}{2}\right) \sum_{n=0}^{L} \tilde{d}(n) \cos(\omega n), \quad L = \frac{M}{2} - 1$$

where coefficients $\tilde{d}(n)$ are given by

$$\begin{aligned} d(1) &= \tilde{d}(0) - \tfrac{1}{2}\tilde{d}(1), \\ d(n) &= \frac{1}{2}\left[\tilde{d}(n-1) - \tilde{d}(n)\right], \quad 2 \le n \le \frac{M}{2} - 1, \\ d\left(\frac{M}{2}\right) &= \frac{1}{2}\tilde{d}\left(\tfrac{M}{2} - 1\right). \end{aligned}$$

P7.6 Write a MATLAB function to compute the amplitude response $H_r(\omega)$ given a linear phase impulse response $h(n)$. The format of this function should be

```
function [Hr,w,P,L] = Ampl_Res(h);
% Computes Amplitude response Hr(w) and its polynomial P of order L,
% given a linear-phase FIR filter impulse response h.
% The type of filter is determined automatically by the subroutine.
%
% [Hr,w,P,L] = Ampl_Res(h)
% Hr = Amplitude Response
% w = frequencies between [0 pi] over which Hr is computed
% P = Polynomial coefficients
% L = Order of P
% h = Linear Phase filter impulse response
```

The function should first determine the type of the linear-phase FIR filter and then use the appropriate `Hr_Type#` function discussed in this chapter. It should also check if the given

$h(n)$ is of a linear-phase type. Verify your function on sequences given here.

$$\begin{aligned}
h_{\rm I}(n) &= (0.9)^{|n-5|}\cos[\pi(n-5)/12]\,[u(n)-u(n-11)]\\
h_{\rm II}(n) &= (0.9)^{|n-4.5|}\cos[\pi(n-4.5)/11]\,[u(n)-u(n-10)]\\
h_{\rm III}(n) &= (0.9)^{|n-5|}\sin[\pi(n-5)/12]\,[u(n)-u(n-11)]\\
h_{\rm IV}(n) &= (0.9)^{|n-4.5|}\sin[\pi(n-4.5)/11]\,[u(n)-u(n-10)]\\
h(n) &= (0.9)^{n}\cos[\pi(n-5)/12]\,[u(n)-u(n-11)]
\end{aligned}$$

P7.7 Prove the following properties of linear-phase FIR filters.

1. If $H(z)$ has four zeros at $z_1 = re^{j\theta}$, $z_2 = \frac{1}{r}e^{-j\theta}$, $z_3 = re^{-j\theta}$, and $z_4 = \frac{1}{r}e^{-j\theta}$ then $H(z)$ represents a linear-phase FIR filter.
2. If $H(z)$ has two zeros at $z_1 = e^{j\theta}$ and $z_2 = e^{-j\theta}$ then $H(z)$ represents a linear-phase FIR filter.
3. If $H(z)$ has two zeros at $z_1 = r$ and $z_2 = \frac{1}{r}$ then $H(z)$ represents a linear-phase FIR filter.
4. If $H(z)$ has a zero at $z_1 = 1$ or a zero at $z_1 = -1$ then $H(z)$ represents a linear-phase FIR filter.
5. For each of the sequences given in Problem P7.6, plot the locations of zeros. Determine which sequences imply linear-phase FIR filters.

P7.8 A notch filter is an LTI system, which is used to eliminate an arbitrary frequency $\omega = \omega_0$. The ideal linear-phase notch filter frequency response is given by

$$H_d\left(e^{j\omega}\right) = \begin{cases} 0, & |\omega| = \omega_0; \\ 1\cdot e^{-j\alpha\omega}, & \text{otherwise.}\end{cases} \qquad (\alpha \text{ is a delay in samples})$$

1. Determine the ideal impulse response, $h_d(n)$, of the ideal notch filter.
2. Using $h_d(n)$, design a linear-phase FIR notch filter using a length 51 rectangular window to eliminate the frequency $\omega_0 = \pi/2$ rad/sample. Plot amplitude the response of the resulting filter.
3. Repeat part 2 using a length 51 Hamming window. Compare your results.

P7.9 Design a linear-phase bandpass filter using the Hann window design technique. The specifications are

lower stopband edge:	0.2π	$A_s = 40$ dB
upper stopband edge:	0.75π	
lower passband edge:	0.35π	$R_p = 0.25$ dB
upper passband edge:	0.55π	

Plot the impulse response and the magnitude response (in dB) of the designed filter. Do not use the `fir1` function.

P7.10 Design a bandstop filter using the Hamming window design technique. The specifications are

lower stopband edge:	0.4π	$A_s = 50$ dB
upper stopband edge:	0.6π	
lower passband edge:	0.3π	$R_p = 0.2$ dB
upper passband edge:	0.7π	

Plot the impulse response and the magnitude response (in dB) of the designed filter. Do not use the `fir1` function.

P7.11 Design a bandpass filter using the Hamming window design technique. The specifications are

$$\begin{array}{rl} \text{lower stopband edge: } 0.3\pi & \multirow{2}{*}{$A_s = 50 \text{ dB}$} \\ \text{upper stopband edge: } 0.6\pi & \\ \text{lower passband edge: } 0.4\pi & \multirow{2}{*}{$R_p = 0.5 \text{ dB}$} \\ \text{upper passband edge: } 0.5\pi & \end{array}$$

Plot the impulse response and the magnitude response (in dB) of the designed filter. Do not use the `fir1` function.

P7.12 Design a highpass filter using one of the fixed window functions. The specifications are

$$\begin{array}{l} \text{stopband edge: } 0.4\pi,\ A_s = 50 \text{ dB} \\ \text{passband edge: } 0.6\pi,\ R_p = 0.004 \text{ dB} \end{array}$$

Plot the zoomed magnitude response (in dB) of the designed filter in the passband to verify the passband ripple R_p. Do not use the `fir1` function.

P7.13 Using the Kaiser window method, design a linear-phase FIR digital filter that meets the following specifications

$$\begin{array}{rl} 0.975 \le |H(e^{j\omega})| \le 1.025, & 0 \le \omega \le 0.25\pi \\ 0 \le |H(e^{j\omega})| \le 0.005, & 0.35\pi \le \omega \le 0.65\pi \\ 0.975 \le |H(e^{j\omega})| \le 1.025, & 0.75\pi \le \omega \le \pi \end{array}$$

Determine the minimum length impulse response $h(n)$ of such a filter. Provide a plot containing subplots of the amplitude response and the magnitude response in dB. Do not use the `fir1` function.

P7.14 We wish to use the Kaiser window method to design a linear-phase FIR digital filter that meets the following specifications:

$$\begin{array}{rl} 0 \le |H(e^{j\omega})| \le 0.01, & 0 \le \omega \le 0.25\pi \\ 0.95 \le |H(e^{j\omega})| \le 1.05, & 0.35\pi \le \omega \le 0.65\pi \\ 0 \le |H(e^{j\omega})| \le 0.01, & 0.75\pi \le \omega \le \pi \end{array}$$

Determine the minimum length impulse response $h(n)$ of such a filter. Provide a plot containing subplots of the amplitude response and the magnitude response in dB. Do not use the `fir1` function.

P7.15 Design the staircase filter of Example 7.26 using the Kaiser window approach. The specifications are

$$\begin{array}{llll} \text{Band-1:} & 0 \le \omega \le 0.3\pi, & \text{Ideal gain } = 1, & \delta_1 = 0.01 \\ \text{Band-2:} & 0.4\pi \le \omega \le 0.7\pi, & \text{Ideal gain } = 0.5, & \delta_2 = 0.005 \\ \text{Band-3:} & 0.8\pi \le \omega \le \pi, & \text{Ideal gain } = 0, & \delta_3 = 0.001 \end{array}$$

Compare the filter length of this design with that of Example 7.26. Provide a plot of the magnitude response in dB. Do not use the `fir1` function.

P7.16 Design a bandpass filter using a fixed window design technique that has the minimum length and that satisfies the following specifications:

$$\left.\begin{array}{r}\text{lower stopband edge} = 0.3\pi\\ \text{upper stopband edge} = 0.6\pi\end{array}\right\} A_s = 40 \text{ dB}$$

$$\left.\begin{array}{r}\text{lower passband edge} = 0.4\pi\\ \text{upper passband edge} = 0.5\pi\end{array}\right\} R_p = 0.5 \text{ dB.}$$

Provide a plot of the log-magnitude response in dB and `stem` plot of the impulse response.

P7.17 Repeat Problem P7.9 using the `fir1` function.

P7.18 Repeat Problem P7.10 using the `fir1` function.

P7.19 Repeat Problem P7.11 using the `fir1` function.

P7.20 Repeat Problem P7.12 using the `fir1` function.

P7.21 Repeat Problem P7.13 using the `fir1` function.

P7.22 Repeat Problem P7.14 using the `fir1` function.

P7.23 Consider an ideal lowpass filter with the cutoff frequency $\omega_c = 0.3\pi$. We want to approximate this filter using a frequency sampling design in which we choose 40 samples.

1. Choose the sample at ω_c equal to 0.5, and use the naive design method to compute $h(n)$. Determine the minimum stopband attenuation.
2. Now vary the sample at ω_c, and determine the optimum value to obtain the largest minimum stopband attenuation.
3. Plot the magnitude responses in dB of the preceding two designs in one plot, and comment on the results.

P7.24 Design the bandstop filter of Problem P7.10 using the frequency sampling method. Choose the order of the filter appropriately so that there are two samples in the transition band. Use optimum values for these samples. Compare your results with those obtained using the `fir2` function.

P7.25 Design the bandpass filter of Problem P7.11 using the frequency sampling method. Choose the order of the filter appropriately so that there are two samples in the transition band. Use optimum values for these samples. Compare your results with those obtained using the `fir2` function.

P7.26 Design the highpass filter of Problem P7.12 using the frequency sampling method. Choose the order of the filter appropriately so that there are two samples in the transition band. Use optimum values. Compare your results with those obtained using the `fir2` function.

P7.27 Consider the filter specifications given in Figure P7.1. Use the `fir2` function and a Hamming window to design a linear-phase FIR filter via the frequency sampling method. Experiment with the filter length to achieve the required design. Plot the amplitude response of the resulting filter.

P7.28 Design a bandpass filter using the frequency sampling method. Choose the order of the filter appropriately so that there is one sample in the transition band. Use optimum value for this sample. The specifications are as follows:

$$\left.\begin{array}{r}\text{lower stopband edge} = 0.3\pi\\ \text{upper stopband edge} = 0.7\pi\end{array}\right\} A_s = 40 \text{ dB}$$

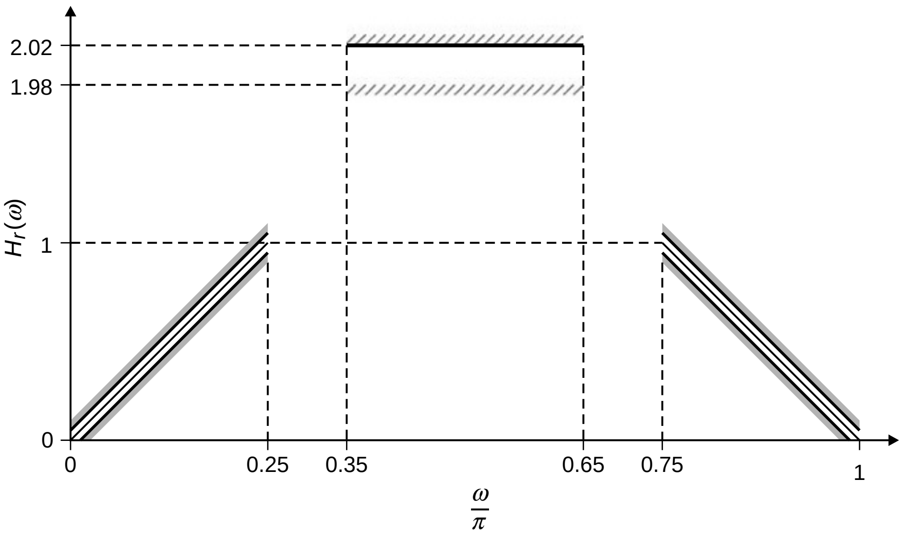


FIGURE P7.1 *Filter Specifications for Problem P7.27*

$$\left.\begin{aligned}\text{lower passband edge} = 0.4\pi\\ \text{upper passband edge} = 0.6\pi\end{aligned}\right\} R_p = 0.5 \text{ dB.}$$

Provide a plot of the log-magnitude response in dB and `stem` plot of the impulse response.

P7.29 The frequency response of an ideal bandpass filter is given by

$$H_d(e^{j\omega}) = \begin{cases} 0, & 0 \le |\omega| \le \pi/3 \\ 1, & \pi/3 \le |\omega| \le 2\pi/3 \\ 0, & 2\pi/3 \le |\omega| \le \pi \end{cases}$$

1. Determine the coefficients of a 25-tap filter based on the Parks-McClellan algorithm with stopband attenuation of 50 dB. The designed filter should have the smallest possible transition width.
2. Plot the amplitude response of the filter using the function developed in Problem P7.6.

P7.30 Consider the bandstop filter given in Problem P7.10.

1. Design a linear-phase bandstop FIR filter using the Parks-McClellan algorithm. Note that the length of the filter must be odd. Provide a plot of the impulse response and the magnitude response in dB of the designed filter.
2. Plot the amplitude response of the designed filter and count the total number of extrema in stopband and passbands. Verify this number with the theoretical estimate of the total number of extrema.
3. Compare the order of this filter with those of the filters in Problems P7.10 and P7.24.
4. Verify the operation of the designed filter on the following signal

$$x(n) = 5 - 5\cos\left(\frac{\pi n}{2}\right); \quad 0 \le n \le 300$$

P7.31 Using the Parks-McClellan algorithm, design a 25-tap FIR differentiator with slope equal to 1 sample/cycle.

1. Choose the frequency band of interest between 0.1π and 0.9π. Plot the impulse response and the amplitude response.
2. Generate 100 samples of the sinusoid

$$x(n) = 3\sin(0.25\pi n), \quad n = 0, ..., 100$$

and process through the preceding FIR differentiator. Compare the result with the theoretical "derivative" of $x(n)$. *Note:* Don't forget to take the 12-sample delay of the FIR filter into account.

P7.32 Design a lowest-order equiripple linear-phase FIR filter to satisfy the specifications given in Figure P7.2. Provide a plot of the amplitude response and a plot of the impulse response.

P7.33 A digital signal $x(n)$ contains a sinusoid of frequency $\pi/2$ and a Gaussian noise $w(n)$ of zero mean and unit variance; i.e.,

$$x(n) = 2\cos\frac{\pi n}{2} + w(n)$$

We want to filter out the noise component using a 50th-order causal and linear-phase FIR filter.

1. Using the Parks-McClellan algorithm, design a narrow bandpass filter with passband width of no more than 0.02π and stopband attenuation of at least 30 dB. Note that no other parameters are given and that you have to choose the remaining parameters for the `firpm` function to satisfy the requirements. Provide a plot of the log-magnitude response in dB of the designed filter.
2. Generate 200 samples of the sequence $x(n)$ and processed through the preceding filter to obtain the output $y(n)$. Provide subplots of $x(n)$ and $y(n)$ for $100 \leq n \leq 200$ on one plot and comment on your results.

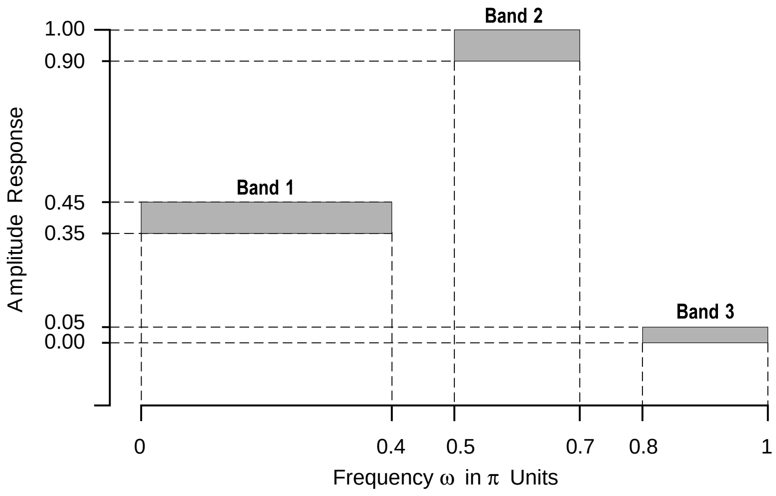


FIGURE P7.2 *Filter Specifications for Problem P7.32*

P7.34 Design a minimum order linear-phase FIR filter, using the Parks-McClellan algorithm, to satisfy the requirements given in Figure P7.1.

1. Provide a plot of the amplitude response with grid-lines and axis labeling as shown in Figure P7.1.
2. Generate the following signals

$$x_1(n) = \cos(0.25\pi n), \quad x_2(n) = \cos(0.5\pi n), \quad x_3(n) = \cos(0.75\pi n); \quad 0 \leq n \leq 100.$$

   Process these signals through this filter to obtain the corresponding output signals $y_1(n)$, $y_2(n)$, and $y_3(n)$. Provide stem plots of all input and output signals in one figure.

P7.35 Design a minimum-order linear-phase FIR filter, using the Parks-McClellan algorithm, to satisfy the requirements given in Figure P7.3. Provide a plot of the amplitude response with grid-lines and axis labeling as shown in Figure P7.3.

P7.36 The specifications on the amplitude response of an FIR filter are given in Figure P7.4.

1. Using a window design approach and a *fixed* window function, design a minimum-length linear-phase FIR filter to satisfy the given requirements. Provide a plot of the amplitude response with grid-lines as shown in Figure P7.4.
2. Using a window design approach and the Kaiser window function, design a minimum-length linear-phase FIR filter to satisfy the given requirements. Provide a plot of the amplitude response with grid-lines as shown in Figure P7.4.
3. Using a frequency-sampling design approach and with no more than two samples in the transition bands, design a minimum-length linear-phase FIR filter to satisfy the given requirements. Provide a plot of the amplitude response with grid-lines as shown in Figure P7.4.
4. Using the Parks-McClellan design approach, design a minimum-length linear-phase FIR filter to satisfy the given requirements. Provide a plot of the amplitude response with grid-lines as shown in Figure P7.4.

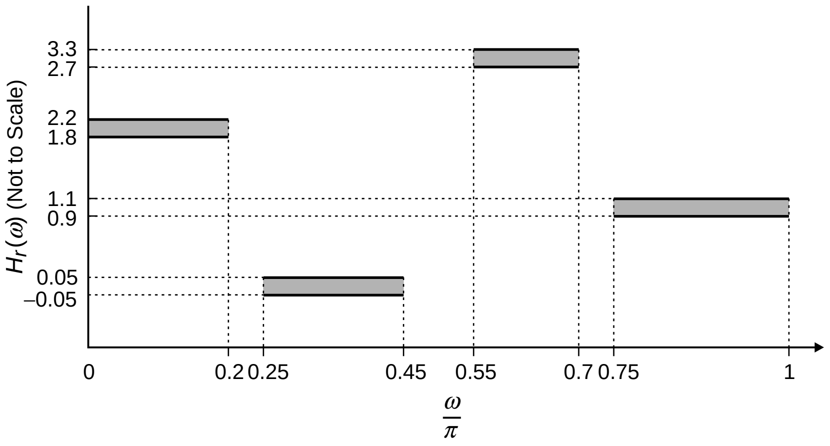


FIGURE P7.3 *Filter Specifications for Problem P7.35*

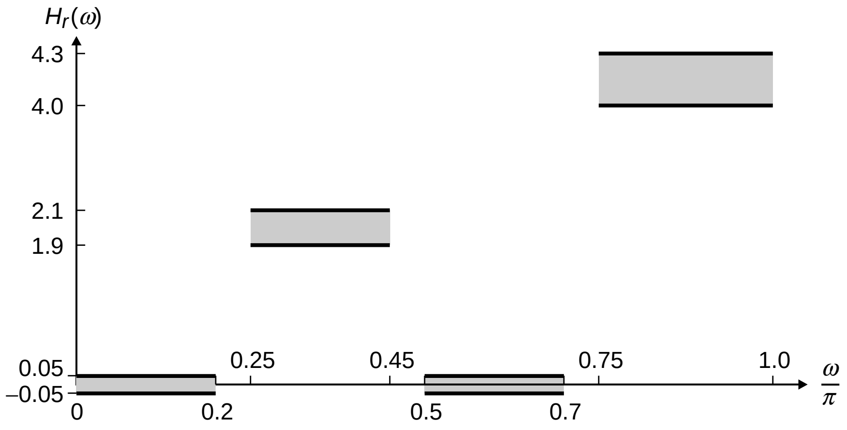


FIGURE P7.4 *Filter Specifications for Problem P7.36*

5. Compare the preceding four design methods in terms of
   - the *order* of the filter
   - the *exact* band-edge frequencies
   - the *exact* tolerances in each band

P7.37 Design a minimum-order linear-phase FIR filter, using the Parks-McClellan algorithm, to satisfy the requirements given in Figure P7.5. Provide a plot of the amplitude response with grid-lines as shown in Figure P7.5.

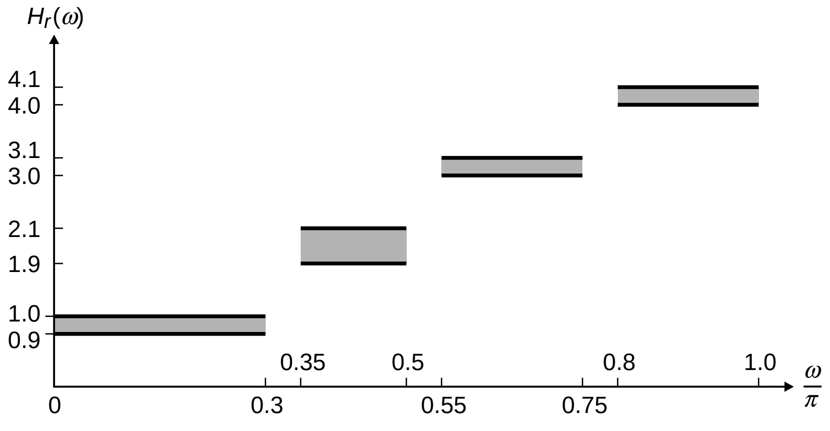


FIGURE P7.5 *Filter Specifications for Problem P7.37*

P7.38 Design a minimum-length linear-phase bandpass filter of Problem P7.9 using the Parks-McClellan algorithm.

1. Plot the impulse response and the magnitude response in dB of the designed filter in one figure plot.
2. Plot the amplitude response of the designed filter and count the total number of extrema in passband and stopbands. Verify this number with the theoretical estimate of the total number of extrema.
3. Compare the order of this filter with that of the filter in Problem P7.9.

CHAPTER 8

IIR Filter Design

IIR filters have infinite-duration impulse responses, hence they can be matched to analog filters, all of which generally have infinitely long impulse responses. Therefore the basic technique of IIR filter design transforms well-known analog filters into digital filters using *complex-valued* mappings. The advantage of this technique lies in the fact that both analog filter design (AFD) tables and the mappings are available extensively in the literature. This basic technique is called the A/D (analog-to-digital) filter transformation. However, the AFD tables are available only for lowpass filters. We also want to design other frequency-selective filters (highpass, bandpass, bandstop, etc.). To do this, we need to apply frequency-band transformations to lowpass filters. These transformations are also complex-valued mappings, and they are also available in the literature. There are two approaches to this basic technique of IIR filter design:

Approach 1:

$$\begin{tikzcd}
\boxed{\begin{array}{c}\text{Design analog}\\ \text{lowpass filter}\end{array}} \arrow[r] & \boxed{\begin{array}{c}\text{Apply freq. band}\\ \text{transformation}\\ s \to s\end{array}} \arrow[r] & \boxed{\begin{array}{c}\text{Apply filter}\\ \text{transformation}\\ s \to z\end{array}} \arrow[r] & \begin{array}{c}\text{Desired IIR}\\ \text{filter}\end{array}
\end{tikzcd}$$

Approach 2:

$$\begin{tikzcd}
\boxed{\begin{array}{c}\text{Design analog}\\ \text{lowpass filter}\end{array}} \arrow[r] & \boxed{\begin{array}{c}\text{Apply filter}\\ \text{transformation}\\ s \to z\end{array}} \arrow[r] & \boxed{\begin{array}{c}\text{Apply freq. band}\\ \text{transformation}\\ z \to z\end{array}} \arrow[r] & \begin{array}{c}\text{Desired IIR}\\ \text{filter}\end{array}
\end{tikzcd}$$

The first approach is used in MATLAB to design IIR filters. A straightforward use of these MATLAB functions does not provide any insight into the design methodology. Therefore we will study the second approach because it involves the frequency-band transformation in the digital domain. Hence in this IIR filter design technique we will follow the following steps:

- Design analog lowpass filters.
- Study and apply filter transformations to obtain digital lowpass filters.
- Study and apply frequency-band transformations to obtain other digital filters from digital lowpass filters.

The main problem with these approaches is that we have no control over the phase characteristics of the IIR filter. Hence IIR filter designs will be treated as *magnitude-only* designs. More sophisticated techniques, which can simultaneously approximate both the magnitude and the phase responses, require advanced optimization tools and hence will not be covered in this book.

We begin with a discussion on the analog filter specifications and the properties of the magnitude-squared response used in specifying analog filters. Next, before we delve into basic techniques for general IIR filters, we consider the design of special types of digital filters—for example, resonators, notch filters, comb filters, etc. This is followed by a brief description of the characteristics of three widely used analog filters: namely. *Butterworth*, *Chebyshev*, and *elliptic* filters. Finally, we will study transformations to convert these prototype analog filters into different frequency-selective digital filters and conclude this chapter with several IIR filter designs using MATLAB.

8.1 SOME PRELIMINARIES

We discuss two preliminary issues in this section. First, we consider the magnitude-squared response specifications, which are more typical of analog (and hence of IIR) filters. These specifications are given on the *relative linear scale*. Second, we study the properties of the magnitude-squared response.

8.1.1 RELATIVE LINEAR SCALE

Let $H_a(j\Omega)$ be the frequency response of an analog filter. Then the lowpass filter specifications on the magnitude-squared response are given by

$$\begin{aligned} \frac{1}{1+\epsilon^2} \le |H_a(j\Omega)|^2 \le\ & 1, \quad |\Omega| \le \Omega_p \\ 0 \le |H_a(j\Omega)|^2 \le\ & \frac{1}{A^2}, \quad \Omega_s \le |\Omega| \end{aligned} \tag{8.1}$$

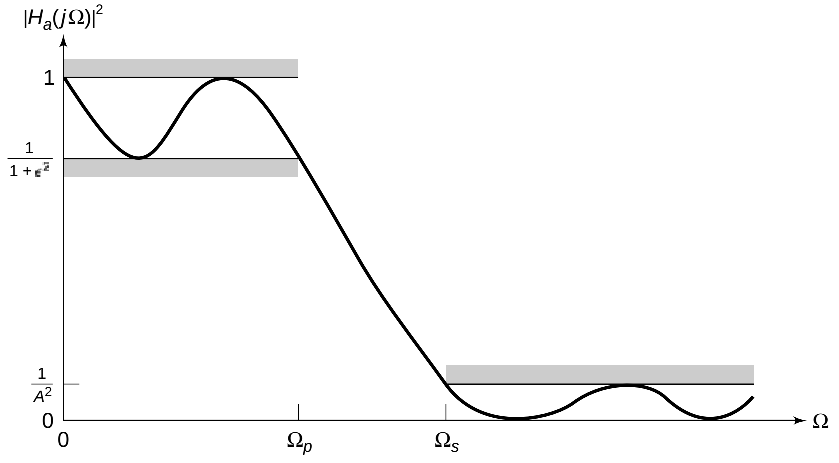


FIGURE 8.1 *Analog lowpass filter specifications*

where ϵ is a passband *ripple parameter*, Ω_p is the passband cutoff frequency in rad/sec, A is a stopband *attenuation parameter*, and Ω_s is the stopband cutoff in rad/sec. These specifications are shown in Figure 8.1, from which we observe that $|H_a(j\Omega)|^2$ must satisfy

$$\begin{aligned} |H_a(j\Omega_p)|^2 &= \frac{1}{1+\epsilon^2} \quad \text{at } \Omega = \Omega_p \\ |H_a(j\Omega_s)|^2 &= \frac{1}{A^2} \quad \text{at } \Omega = \Omega_s \end{aligned} \tag{8.2}$$

The parameters ϵ and A are related to parameters R_p and A_s, respectively, of the dB scale. These relations are given by

$$R_p = -10\log_{10}\frac{1}{1+\epsilon^2} \Longrightarrow \epsilon = \sqrt{10^{R_p/10}-1} \tag{8.3}$$

and

$$A_s = -10\log_{10}\frac{1}{A^2} \Longrightarrow A = 10^{A_s/20} \tag{8.4}$$

The ripples, δ_1 and δ_2, of the absolute scale are related to ϵ and A by

$$\frac{1-\delta_1}{1+\delta_1} = \sqrt{\frac{1}{1+\epsilon^2}} \Longrightarrow \epsilon = \frac{2\sqrt{\delta_1}}{1-\delta_1}$$

and

$$\frac{\delta_2}{1+\delta_1} = \frac{1}{A} \Longrightarrow A = \frac{1+\delta_1}{\delta_2}$$

8.1.2 PROPERTIES OF $|H_a(j\Omega)|^2$

Analog filter specifications (8.1), which are given in terms of the magnitude-squared response, contain no phase information. Now to evaluate the s-domain system function $H_a(s)$, consider

$$H_a(j\Omega) = H_a(s)|_{s=j\Omega}$$

Then we have

$$|H_a(j\Omega)|^2 = H_a(j\Omega)H_a^*(j\Omega) = H_a(j\Omega)H_a(-j\Omega) = H_a(s)H_a(-s)|_{s=j\Omega}$$

or

$$H_a(s)H_a(-s) = |H_a(j\Omega)|^2\Big|_{\Omega=s/j} \tag{8.5}$$

Therefore the poles and zeros of the magnitude-squared function are distributed in a *mirror-image symmetry* with respect to the $j\Omega$ axis. Also for real filters, poles and zeros occur in complex conjugate pairs (or mirror-image symmetry with respect to the real axis). A typical pole-zero pattern of $H_a(s)H_a(-s)$ is shown in Figure 8.2. From this pattern we can construct $H_a(s)$, which is the system function of our analog filter. We want $H_a(s)$ to represent a *causal* and *stable* filter. Then all poles of $H_a(s)$ must lie within the left half-plane. Thus we assign all left-half poles of $H_a(s)H_a(-s)$ to $H_a(s)$. However, zeros of $H_a(s)$ can lie anywhere in the s-plane. Therefore they are not uniquely determined unless they all are on the $j\Omega$ axis. We will choose the zeros of $H_a(s)H_a(-s)$ lying left to or on the $j\Omega$ axis as the zeros of $Ha(s)$. The resulting filter is then called a *minimum-phase* filter.

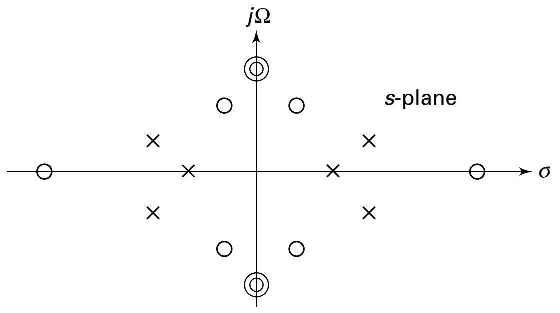


FIGURE 8.2 *Typical pole-zero pattern of $H_a(s)H_a(-s)$*

8.2 SOME SPECIAL FILTER TYPES

In this section we consider the design of several special types of digital filters and describe their frequency response characteristics. We begin by describing the design and characteristics of a digital resonator.

8.2.1 DIGITAL RESONATORS

A digital resonator is a special two-pole bandpass filter with a pair of complex-conjugate poles located very near the unit circle, as shown in Figure 8.3a. The magnitude of the frequency response of the filter is shown in Figure 8.3b. The name *resonator* refers to the fact that the filter has a large magnitude response in the vicinity of the pole position. The angle of the pole location determines the resonant frequency of the filter. Digital resonators are useful in many applications, including simple bandpass filtering and speech generation.

Let us consider the design of a digital resonator with a resonant peak at or near $\omega = \omega_0$. Hence, we select the pole position as

$$p_{1,2} = re^{\pm j\omega_0} \tag{8.6}$$

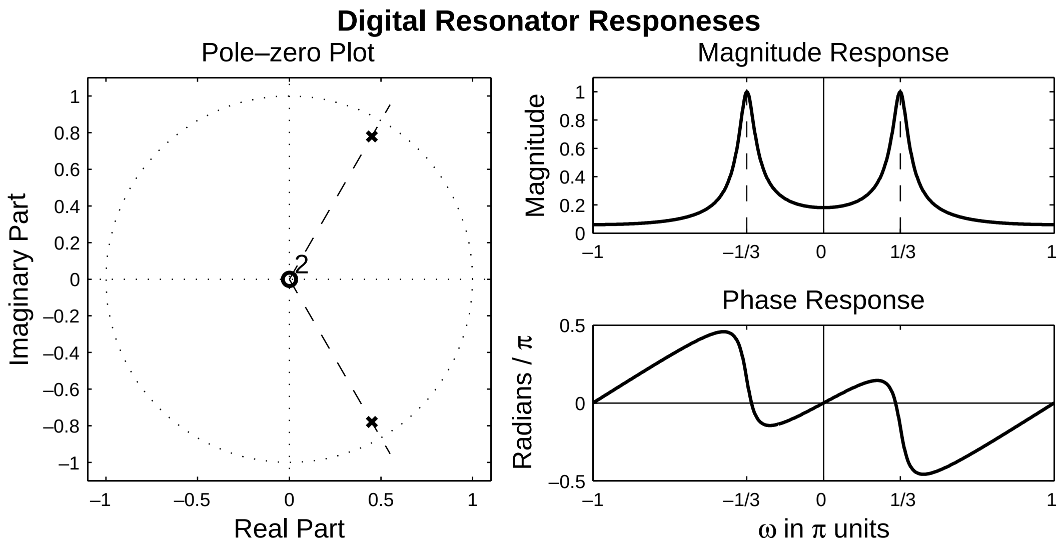


FIGURE 8.3 *Pole positions and frequency response of a digital resonator with $r = 0.9$ and $\omega_0 = \pi/3$*

The corresponding system function is

$$\begin{aligned} H(z) &= \frac{b_0}{(1 - re^{j\omega_0}z^{-1})(1 - re^{-j\omega_0}z^{-1})} \\ &= \frac{b_0}{1 - (2r\cos\omega_0)z^{-1} + r^2 z^{-2}} \end{aligned} \tag{8.7}$$

where b_0 is a gain parameter. The frequency response of the resonator is

$$H(e^{j\omega}) = \frac{b_0}{\left[1 - re^{-j(\omega-\omega_0)}\right]\left[1 - re^{-j(\omega+\omega_0)}\right]} \tag{8.8}$$

Since $\left|H(e^{j\omega})\right|$ has its peak at or near $\omega = \omega_0$, we select the gain parameter b_0 so that $\left|H(e^{j\omega})\right| = 1$. Hence,

$$\begin{aligned} \left|H(e^{j\omega_0})\right| &= \frac{b_0}{|(1-r)(1 - re^{-j2\omega_0})|} \\ &= \frac{b_0}{(1-r)\sqrt{1 + r^2 - 2r\cos 2\omega_0}} \end{aligned} \tag{8.9}$$

Consequently, the desired gain parameter is

$$b_0 = (1-r)\sqrt{1 + r^2 - 2r\cos 2\omega_0} \tag{8.10}$$

The magnitude of the frequency response $H(\omega)$ may be expressed as

$$\left|H(e^{j\omega})\right| = \frac{b_0}{D_1(\omega)D_2(\omega)} \tag{8.11}$$

where $D_1(\omega)$ and $D_2(\omega)$ are given as

$$D_1(\omega) = \sqrt{1 + r^2 - 2r\cos(\omega - \omega_0)} \tag{8.12a}$$

$$D_2(\omega) = \sqrt{1 + r^2 - 2r\cos(\omega + \omega_0)} \tag{8.12b}$$

For a given value of r, $D_1(\omega)$ takes its minimum value $(1-r)$ at $\omega = \omega_0$, and the product $D_1(\omega)D_2(\omega)$ attains a minimum at the frequency

$$\omega_r = \cos^{-1}\left(\frac{1+r^2}{2r}\cos\omega_0\right) \tag{8.13}$$

which defines precisely the resonant frequency of the filter. Note that when r is very close to unity, $\omega_r \approx \omega_0$, which is the angular position of the pole. Furthermore, as r approaches unity, the resonant peak becomes sharper (narrower) because $D_1(\omega)$ changes rapidly in the vicinity of ω_0.

A quantitative measure of the width of the peak is the 3dB bandwidth of the filter, denoted as $\Delta(\omega)$. For values of r close to unity,

$$\Delta\omega \approx 2(1-r) \tag{8.14}$$

Figure 8.3 illustrates the magnitude and phase responses of a digital resonator with $\omega_0 = \pi/3$, $r = 0.90$. Note that the phase response has its greatest rate of change near the resonant frequency $\omega_r \approx \omega_0 = \pi/3$.

This resonator has two zeros at $z = 0$. Instead of placing zeros at the origin, an alternative choice is to locate the zeros at $z = 1$ and $z = -1$. This choice completely eliminates the response of the filter at the frequencies $\omega = 0$ and $\omega = \pi$, which may be desirable in some applications. The corresponding resonator has the system function

$$\begin{aligned} H(z) &= \frac{G(1-z^{-1})(1+z^{-1})}{(1-re^{j\omega_0}z^{-1})(1-re^{-j\omega_0}z^{-1})} \\ &= G\frac{1-z^{-2}}{1-(2r\cos\omega_0)z^{-1}+r^2z^{-2}} \end{aligned} \tag{8.15}$$

and the frequency response characteristic

$$H\left(e^{j\omega}\right) = G\frac{1-e^{-j2\omega}}{[1-re^{j(\omega_0-\omega)}][1-re^{-j(\omega_0+\omega)}]} \tag{8.16}$$

where G is a gain parameter that is selected so that $\left|H\left(e^{j\omega_0}\right)\right| = 1$.

The introduction of zeros at $z = \pm 1$ alters both the magnitude and phase response of the resonator. The magnitude response may be expressed as

$$\left|H\left(e^{j\omega}\right)\right| = G\frac{N(\omega)}{D_1(\omega)D_2(\omega)} \tag{8.17}$$

where $N(\omega)$ is defined as

$$N(\omega) = \sqrt{2(1-\cos 2\omega)} \tag{8.18}$$

Due to the presence of the zeros at $z = \pm 1$, the resonant frequency of the resonator is altered from the expression given by (8.13). The bandwidth of the filter is also altered. Although exact values for these two parameters are rather tedious to derive, we can easily compute the frequency response when the zeros are at $z = \pm 1$ and $z = 0$, and compare the results.

Figure 8.4 illustrates the magnitude and phase responses for the cases $z = \pm 1$ and $z = 0$, for pole location at $\omega = \pi/3$ and $r = 0.90$. We observe that the resonator with $z = \pm 1$ has a slightly smaller bandwidth than the resonator with zeros at $z = 0$. In addition, there appears to be a very small shift in the resonant frequency between the two cases.

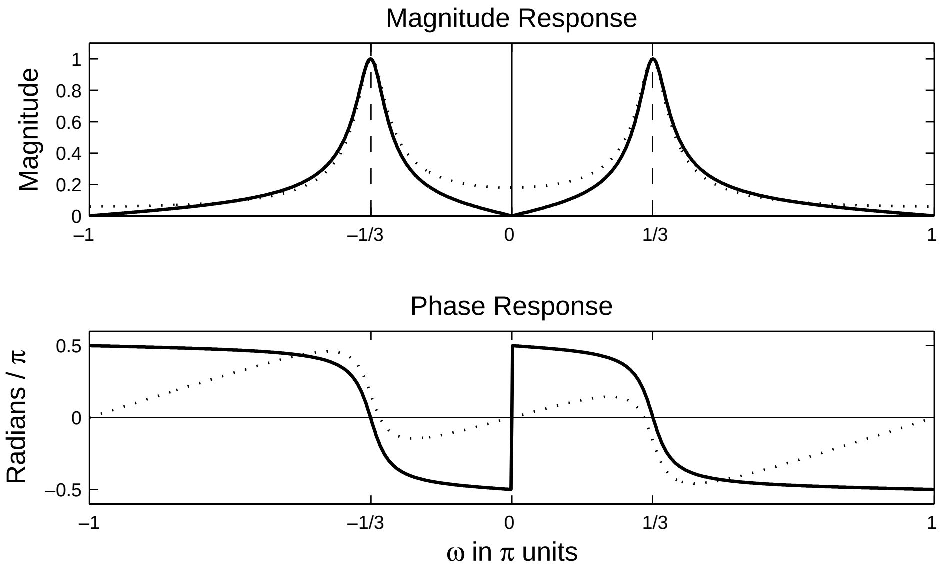


FIGURE 8.4 *Magnitude and phase responses of digital resonator with zeros at $z = \pm 1$ (solid lines) and $z = 0$ (dotted lines) for $r = 0.9$ and $\omega_0 = \pi/3$*

8.2.2 NOTCH FILTERS

A notch filter is a filter that contains one or more deep notches or, ideally, perfect nulls in its frequency response. Figure 8.5 illustrates the frequency response of a notch filter with a null at the frequency $\omega = \omega_0$. Notch filters are useful in many applications where specific frequency components must be eliminated. For example, instrumentation systems require that the power line frequency of 60 Hz and its harmonics be eliminated.

To create a null in the frequency response of a filter at a frequency ω_0, we simply introduce a pair of complex-conjugate zeros on the unit circle at the angle ω_0. Hence, the zeros are selected as

$$z_{1,2} = e^{\pm j\omega_0} \tag{8.19}$$

Then, the system function for the notch filter is

$$\begin{aligned} H(z) &= b_0(1 - e^{j\omega_0}z^{-1})(1 - e^{-j\omega_0}z^{-1}) \\ &= b_0(1 - (2\cos\omega_0)z^{-1} + z^{-2}) \end{aligned} \tag{8.20}$$

where b_0 is a gain factor. Figure 8.6 illustrates the magnitude response of a notch filter having a null at $\omega = \pi/4$.

The major problem with this notch filter is that the notch has a relatively large bandwidth, which means that other frequency components

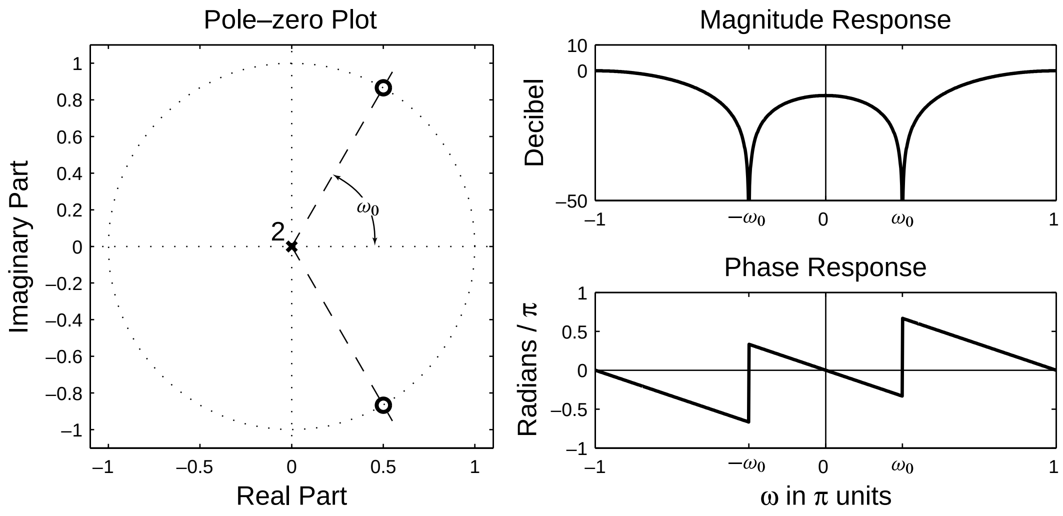


FIGURE 8.5 *Frequency response of a typical notch filter*

around the desired null are severely attenuated. To reduce the bandwidth of the null, we may resort to the more sophisticated, longer FIR filter designed according to the optimum equiripple design method described in Chapter 7. Alternatively, we could attempt to improve the frequency response of the filter by introducing poles in the system function.

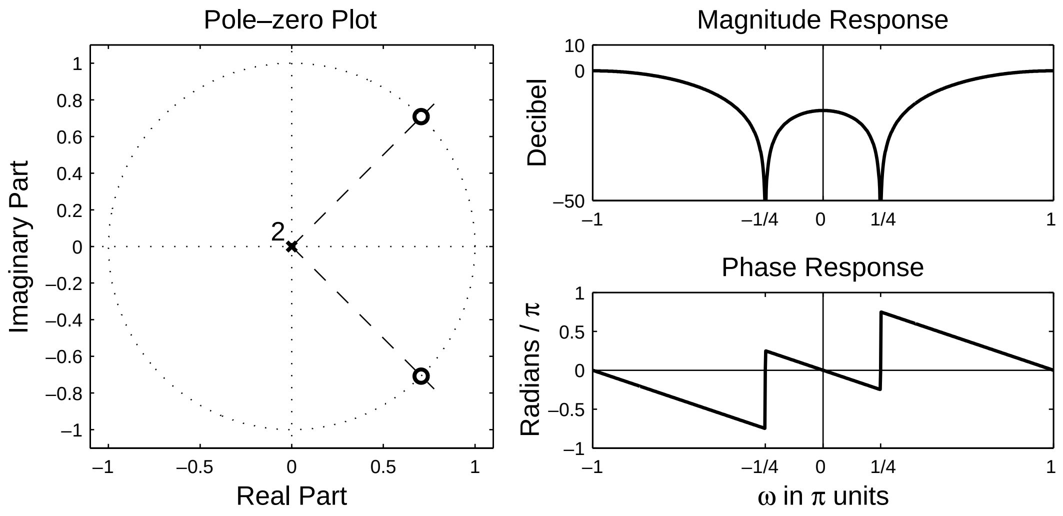


FIGURE 8.6 *Frequency response of a notch filter with $\omega_0 = \pi/4$*

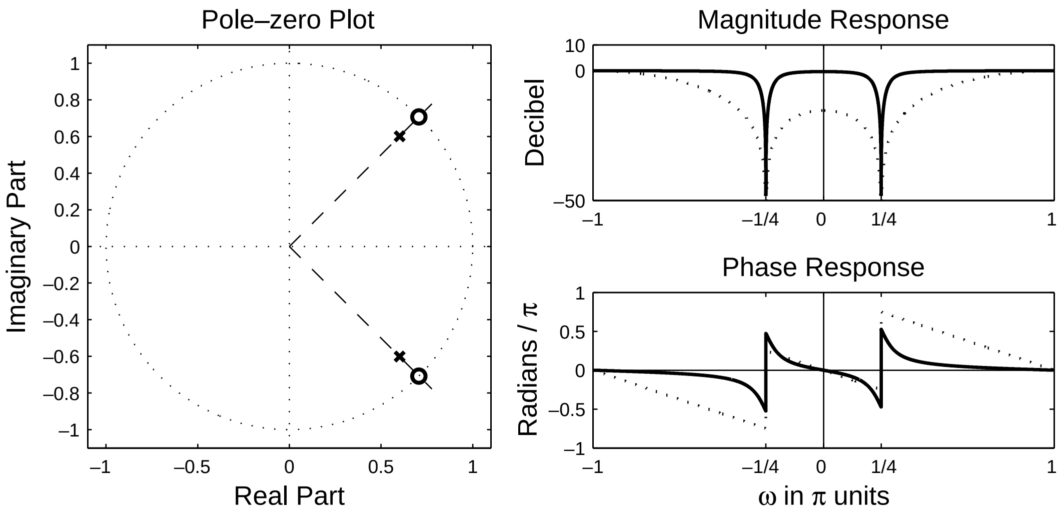


FIGURE 8.7 *Magnitude and phase responses of notch filter with poles (solid lines) and without poles (dotted lines) for $\omega_0 = \pi/4$ and $r = 0.85$*

In particular, suppose that we select the poles at

$$p_{1,2} = re^{\pm j\omega_0} \tag{8.21}$$

Hence, the system function becomes

$$H(z) = b_0 \frac{1 - (2\cos\omega_0)z^{-1} + z^2}{1 - (2r\cos\omega_0)z^{-1} + r^2 z^{-2}} \tag{8.22}$$

The magnitude of the frequency response $|H(e^{j\omega})|$ of this filter is illustrated in Figure 8.7 for $\omega_0 = \pi/4$ and $r = 0.85$. Also plotted in this figure is the frequency response without the poles. We observe that the effect of the pole is to introduce a resonance in the vicinity of the null and, thus, to reduce the bandwidth of the notch. In addition to reducing the bandwidth of the notch, the introduction of a pole in the vicinity of the null may result in a small ripple in the passband of the filter due to the resonance created by the pole.

8.2.3 COMB FILTERS

In its simplest form, a comb filter may be viewed as a notch filter in which the nulls occur periodically across the frequency band, hence the analogy to an ordinary comb that has periodically spaced teeth. Comb filters are used in many practical systems, including the rejections of power-line harmonics, and the suppression of clutter from fixed objects in moving-target indicator (MTI) radars.

We can create a comb filter by taking our FIR filter with system function

$$H(z) = \sum_{k=0}^{M} h(k) z^{-k} \tag{8.23}$$

and replacing z by z^L, where L is a positive integer. Thus, the new FIR filter has the system function

$$H_L(z) = \sum_{k=0}^{M} h(k) z^{-kL} \tag{8.24}$$

If the frequency response of the original FIR filter is $H\big(e^{j\omega}\big)$, the frequency response of the filter given by (8.24) is

$$H_L\big(e^{j\omega}\big) = \sum_{k=0}^{M} h(k) e^{-jkL\omega} = H\big(e^{jL\omega}\big) \tag{8.25}$$

Consequently, the frequency response characteristic $H_L\big(e^{j\omega}\big)$ is an L-order repetition of $H\big(e^{j\omega}\big)$ in the range $0 \le \omega \le 2\pi$. Figure 8.8 illustrates the relationship between $H_L\big(e^{j\omega}\big)$ and $H\big(e^{j\omega}\big)$ for $L = 4$. The introduction of a pole at each notch may be used to narrow the bandwidth of each notch, as just described.

8.2.4 ALLPASS FILTERS

An *allpass filter* is characterized by a system function that has a constant magnitude response for all frequencies, i.e.,

$$\big|H\big(e^{j\omega}\big)\big| = 1, \qquad 0 \le \omega \le \pi \tag{8.26}$$

A simple example of an allpass system is a system that introduces a pure delay to an input signal, i.e.,

$$H(z) = z^{-k} \tag{8.27}$$

This system passes all frequency components of an input signal without any frequency dependent attenuation. It simply delays all frequency components by k samples.

A more general characterization of an allpass filter is one having a system function of the form

$$H(z) = \frac{a_N + a_{N-1}z^{-1} + \cdots + a_1 z^{-N+1} + z^{-N}}{1 + a_1 z^{-1} + \cdots + a_{N-1} z^{-N+1} + a_N z^{-N}} \tag{8.28}$$

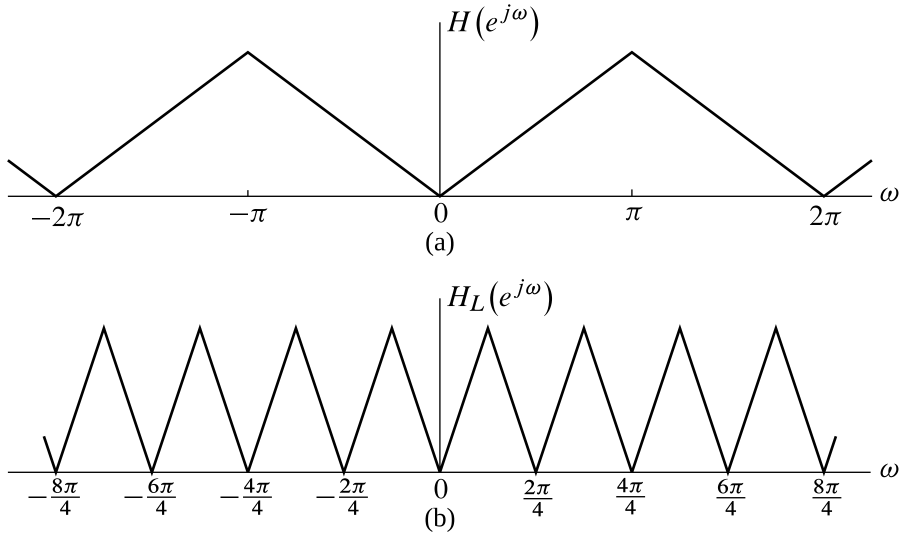


FIGURE 8.8 *Comb filters with frequency response $H_L(e^{j\omega})$ obtained from $H(e^{j\omega})$ for $L = 4$*

which may be expressed in the compact form as

$$H(z) = z^{-N} \frac{A(z^{-1})}{A(z)} \tag{8.29}$$

where

$$A(z) = \sum_{k=0}^{N} a_k z^{-k}, \qquad a_0 = 1 \tag{8.30}$$

We observe that

$$\left|H(e^{j\omega})\right|^2 = H(z)H(z^{-1})|_{z=e^{j\omega}} = 1 \tag{8.31}$$

for all frequencies. Hence, the system is all-pass.

From the form of $H(z)$ given by (8.28), we observe that if z_0 is a pole of $H(z)$, then $1/z_0$ is a zero of $H(z)$. That is, the poles and zeros are reciprocals of one another. Figure 8.9 illustrates the typical pole-zero pattern for a single-pole, single-zero filter and a 2-pole, 2-zero filter. Graphs of the magnitude and phase characteristics of these two filters are shown in Figure 8.10 for $a = 0.6$ and $r = 0.9$, $\omega_0 = \pi/4$, where $A(z)$ for the two filters is, respectively, given as

$$A(z) = 1 + az^{-1} \tag{8.32a}$$

$$A(z) = 1 - (2r\cos\omega_0)z^{-1} + r^2 z^{-2} \tag{8.32b}$$

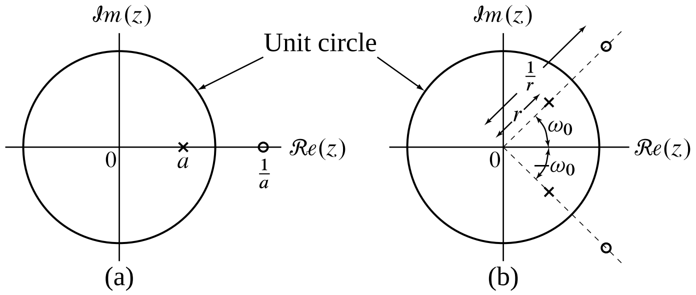


FIGURE 8.9 *Pole-zero locations for (a) one-pole and (b) two-pole allpass filter*

The general form for the system function of an allpass filter with real coefficients may be expressed in factored form as

$$H(z) = \prod_{k=1}^{N_{\text{R}}} \frac{z^{-1} - \alpha_k}{1 - \alpha_k z^{-1}} \prod_{k=1}^{N_{\text{C}}} \frac{(z^{-1} - \beta_k)(z^{-1} - \beta_k^*)}{(1 - \beta_k z^{-1})(1 - \beta_k^* z^{-1})} \qquad \textbf{(8.33)}$$

where N_{R} is the number of real poles and zeros and N_{C} is the number of complex-conjugate pairs of poles and zeros. For a causal and stable system, we require that $|\alpha_k| < 1$ and $|\beta_k| < 1$.

Allpass filters are usually employed as phase equalizers. When placed in cascade with a system that has an undesirable phase response, a phase equalizer is designed to compensate for the poor phase characteristics of the system and thus result in an overall linear phase system.

8.2.5 DIGITAL SINUSOIDAL OSCILLATORS

A digital sinusoidal oscillator can be viewed as a limiting form of a 2-pole resonator for which the complex-conjugate poles are located on the unit

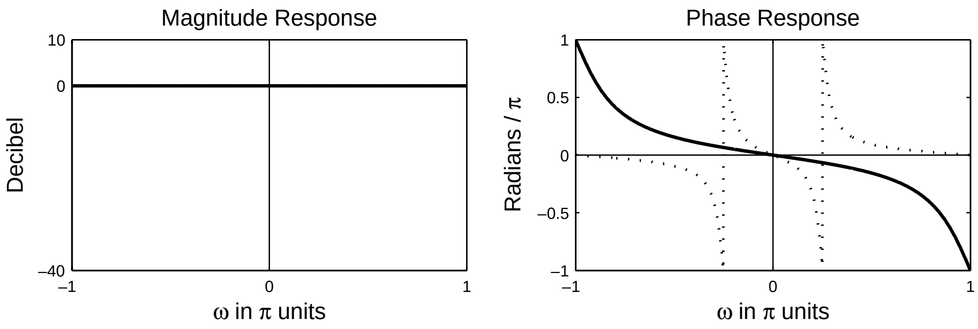


FIGURE 8.10 *Magnitude and phase responses for 1-pole (solid line) and 2-pole (dotted line) allpass filters*

circle. From our previous discussion of resonators, the system function for a resonator with poles at $re\pm^{j\omega_0}$ is

$$H(z) = \frac{b_0}{1-(2r\cos\omega_0)z^{-1}+r^2z^{-2}} \tag{8.34}$$

When we set $r = 1$ and select the gain parameter b_0 as

$$b_0 = A\sin\omega_0 \tag{8.35}$$

The system function becomes

$$H(z) = \frac{A\sin\omega_0}{1-(2\cos\omega_0)z^{-1}+z^{-2}} \tag{8.36}$$

and the corresponding impulse response of the system becomes

$$h(n) = A\sin(n+1)\omega_0 \; u(n) \tag{8.37}$$

Thus, this system generates a sinusoidal signal of frequency ω_0 when excited by an impulse $\delta(n) = 1$.

The block diagram representation of the system function given by (8.36) is illustrated in Figure 8.11. The corresponding difference equation for this system is

$$y(n) = (2\cos\omega_0)\; y(n-1) - y(n-2) + b_0\delta(n) \tag{8.38}$$

where $b_0 = A\sin\omega_0$.

Note that the sinusoidal oscillation obtained from the difference equation in (8.38) can also be obtained by setting the input to zero and setting the initial conditions to $y(-1) = 0,\ y(-2) = -A\sin\omega_0$. Thus, the zero-input response to the 2nd-order system described by the homogeneous difference equation

$$y(n) = (2\cos\omega_0)\; y(n-1) - y(n-2) \tag{8.39}$$

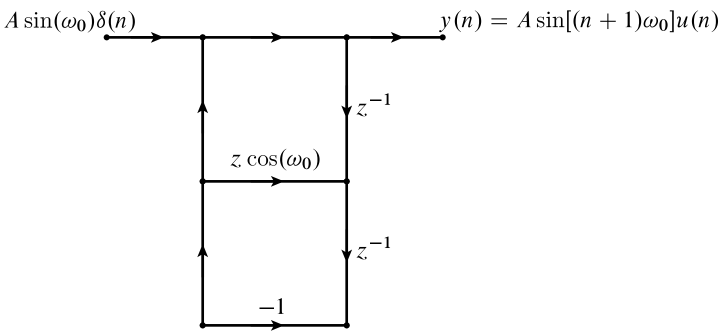


FIGURE 8.11 *Digital sinusoidal oscillator*

with initial conditions $y(-1) = 0$, $y(-2) = -A \sin \omega_0$ is exactly the same as the response of (8.38) to an impulse excitation. In fact, the homogeneous difference equation in (8.39) can be obtained directly from the trigonometric identity

$$\sin \alpha + \sin \beta = 2 \sin \left(\frac{\alpha + \beta}{2} \right) \cos \left(\frac{\alpha - \beta}{2} \right) \tag{8.40}$$

where, by definition, $\alpha = (n+1)\omega_0$, $\beta = (n-1)\omega_0$, and $y(n) = \sin(n+1)\omega_0$.

In practical applications involving modulation of two sinusoidal carrier signals in phase quadrature, there is a need to generate the sinusoids $A \sin \omega_0 n$ and $A \cos \omega_0 n$. These quadrature carrier signals can be generated by the so-called coupled-form oscillator, which can be obtained with the aid of the trigonometric formulas

$$\cos(\alpha + \beta) = \cos \alpha \cos \beta - \sin \alpha \sin \beta \tag{8.41}$$

$$\sin(\alpha + \beta) = \sin \alpha \cos \beta + \cos \alpha \sin \beta \tag{8.42}$$

where by definition, $\alpha = n\omega_0$, $\beta = \omega_0$, $y_c(n) = \cos(n+1)\omega_0$, and $y_s(n) = \sin(n+1)\omega_0$. Thus, with substitution of these quantities into the two trigonometric identities, we obtain the two coupled difference equations.

$$y_c(n) = (\cos \omega_0)\, y_c(n-1) - (\sin \omega_0)\, y_s(n-1) \tag{8.43}$$

$$y_s(n) = (\sin \omega_0)\, y_c(n-1) + (\cos \omega_0)\, y_s(n-1) \tag{8.44}$$

The structure for the realization of the coupled-form oscillator is illustrated in Figure 8.12. Note that this is a 2-output system that does not require any input excitation, but it does require setting the initial conditions $y_c(-1) = A \cos \omega_0$ and $y_s(-1) = -A \sin \omega_0$ in order to begin its self-sustaining oscillations.

8.3 CHARACTERISTICS OF PROTOTYPE ANALOG FILTERS

IIR filter design techniques rely on existing analog filters to obtain digital filters. We designate these analog filters as *prototype* filters. Three prototypes are widely used in practice. In this section we briefly summarize the characteristics of the lowpass versions of these prototypes: Butterworth lowpass, Chebyshev lowpass (Type I and II), and Elliptic lowpass. Although we will use MATLAB functions to design these filters, it is necessary to learn the characteristics of these filters so that we can use proper parameters in MATLAB functions to obtain correct results.

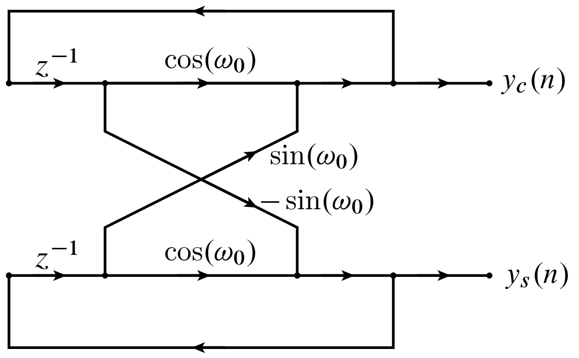


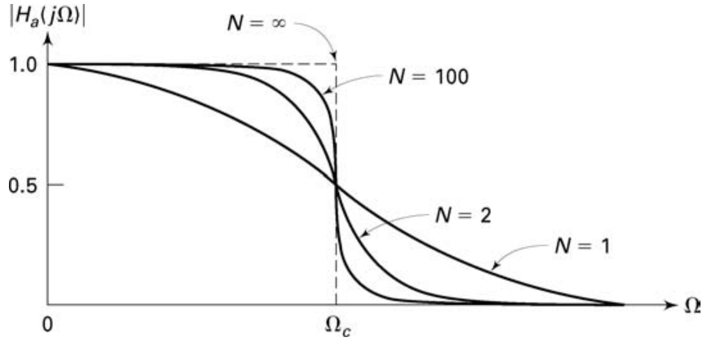
FIGURE 8.12 *Realization of the coupled form oscillator*

8.3.1 BUTTERWORTH LOWPASS FILTERS

This filter is characterized by the property that its magnitude response is flat in both passband and stopband. The magnitude-squared response of an Nth-order lowpass filter is given by

$$|H_a(j\Omega)|^2 = \frac{1}{1+\left(\frac{\Omega}{\Omega_c}\right)^{2N}} \tag{8.45}$$

where N is the order of the filter and Ω_c is the cutoff frequency in rad/sec. The plot of the magnitude-squared response is as follow.



From this plot, we can observe the following properties:

- at $\Omega = 0$, $|H_a(j0)|^2 = 1$ for all N.
- at $\Omega = \Omega_c$, $|H_a(j\Omega_c)|^2 = \frac{1}{2}$ for all N, which implies a 3 dB attenuation at Ω_c.
- $|H_a(j\Omega)|^2$ is a monotonically decreasing function of Ω.

- $|H_a(j\Omega)|^2$ approaches an ideal lowpass filter as $N \to \infty$.
- $|H_a(j\Omega)|^2$ is *maximally flat* at $\Omega = 0$ since derivatives of all orders exist and are equal to zero.

To determine the system function $H_a(s)$, we put (8.45) in the form of (8.5) to obtain

$$H_a(s)H_a(-s) = |H_a(j\Omega)|^2\Big|_{\Omega=s/j} = \frac{1}{1+\left(\dfrac{s}{j\Omega_c}\right)^{2N}} = \frac{(j\Omega)^{2N}}{s^{2N}+(j\Omega_c)^{2N}} \tag{8.46}$$

The roots of the denominator polynomial (or poles of $H_a(s)H_a(-s)$) from (8.46) are given by

$$p_k = (-1)^{\frac{1}{2N}}(j\Omega) = \Omega_c e^{j\frac{\pi}{2N}(2k+N+1)}, \quad k = 0, 1, \ldots, 2N-1 \tag{8.47}$$

An interpretation of (8.47) is that

- there are $2N$ poles of $H_a(s)H_a(-s)$, which are equally distributed on a circle of radius Ω_c with angular spacing of π/N radians
- for N odd the poles are given by $p_k = \Omega_c e^{jk\pi/N}, \quad k = 0, 1, \ldots, 2N-1$
- for N even the poles are given by $p_k = \Omega_c e^{j\left(\frac{\pi}{2N}+\frac{k\pi}{N}\right)}, \quad k = 0, 1, \ldots, 2N-1$
- the poles are symmetrically located with respect to the $j\Omega$ axis
- a pole never falls on the imaginary axis, and falls on the real axis only if N is odd

As an example, poles of 3rd- and 4th-order Butterworth filters are shown in Figure 8.13.

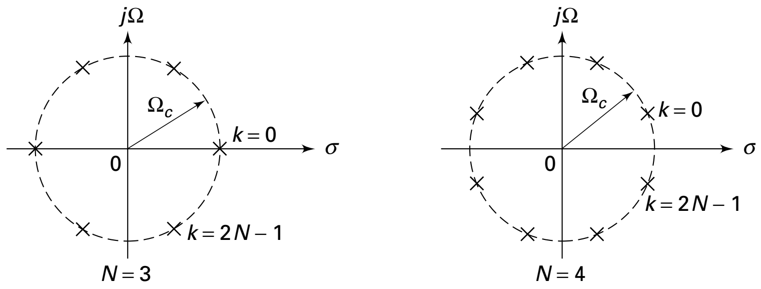


FIGURE 8.13 *Pole plots for Butterworth filters*

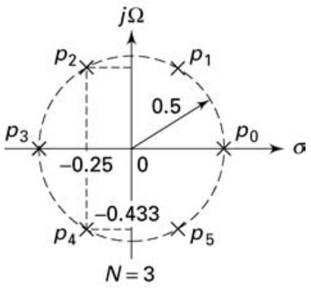


FIGURE 8.14 *Pole plot for Example 8.1*

A stable and causal filter $H_a(s)$ can now be specified by selecting poles in the left half-plane, and $H_a(s)$ can be written in the form

$$H_a(s) = \frac{\Omega_c^N}{\prod\limits_{\text{LHP poles}} (s - p_k)} \tag{8.48}$$

□ **EXAMPLE 8.1** Given that $|H_a(j\Omega)|^2 = \dfrac{1}{1+64\Omega^6}$, determine the analog filter system function $H_a(s)$.

Solution From the given magnitude-squared response,

$$|H_a(j\Omega)|^2 = \frac{1}{1+64\Omega^6} = \frac{1}{1+\left(\dfrac{\Omega}{0.5}\right)^{2(3)}}$$

Comparing this with expression (8.45), we obtain $N = 3$ and $\Omega_c = 0.5$. The poles of $H_a(s)H_a(-s)$ are as shown in Figure 8.14.

Hence

$$\begin{aligned} H_a(j\Omega) &= \frac{\Omega_c^3}{(s-p_2)(s-p_3)(s-p_4)} \\ &= \frac{1/8}{(s+0.25-j0.433)(s+0.5)(s+0.25+j0.433)} \\ &= \frac{0.125}{(s+0.5)(s^2+0.5s+0.25)} \end{aligned}$$

□

8.3.2 MATLAB IMPLEMENTATION

MATLAB provides a function called `[z,p,k]=buttap(N)` to design a *normalized* (i.e., $\Omega_c = 1$) Butterworth analog prototype filter of order N, which returns zeros in `z` array, poles in `p` array, and the gain value `k`. However, we need an unnormalized Butterworth filter with arbitrary Ω_c. From Example 8.1 we observe that there are no zeros and that the poles of the unnormalized filter are on a circle with radius Ω_c instead of on a unit circle. This means that we have to scale the array `p` of the normalized filter by Ω_c and the gain `k` by Ω_c^N. In the following function, called `U_buttap(N,Omegac)`, we design the unnormalized Butterworth analog prototype filter.

```
function [b,a] = u_buttap(N,Omegac);
% Unnormalized Butterworth Analog Lowpass Filter Prototype
% -----------------------------------------------------------
% [b,a] = u_buttap(N,Omegac);
%      b = numerator polynomial coefficients of Ha(s)
%      a = denominator polynomial coefficients of Ha(s)
%      N = Order of the Butterworth Filter
% Omegac = Cutoff frequency in radians/sec
%
[z,p,k] = buttap(N);
      p = p*Omegac;
      k = k*Omegac^N;
      B = real(poly(z));
      b0 = k;  b = k*B;  a = real(poly(p));
```

This function provides a direct form (or numerator-denominator) structure. Often we also need a cascade form structure. In Chapter 6 we have already studied how to convert a direct form into a cascade form. The following `sdir2cas` function describes the procedure that is suitable for analog filters.

```
function [C,B,A] = sdir2cas(b,a);
% DIRECT-form to CASCADE-form conversion in s-plane
% ---------------------------------------------------
% [C,B,A] = sdir2cas(b,a)
%  C = gain coefficient
%  B = K by 3 matrix of real coefficients containing bk's
%  A = K by 3 matrix of real coefficients containing ak's
%  b = numerator polynomial coefficients of DIRECT form
%  a = denominator polynomial coefficients of DIRECT form
%
Na = length(a)-1; Nb = length(b)-1;
```

```
% compute gain coefficient C
b0 = b(1); b = b/b0;  a0 = a(1); a = a/a0;  C = b0/a0;
%
% Denominator second-order sections:
p= cplxpair(roots(a)); K = floor(Na/2);
if K*2 == Na     % Computation when Na is even
   A = zeros(K,3);
   for n=1:2:Na
       Arow = p(n:1:n+1,:);  Arow = poly(Arow);
       A(fix((n+1)/2),:) = real(Arow);
   end

elseif Na == 1   % Computation when Na = 1
       A = [0 real(poly(p))];

else             % Computation when Na is odd and > 1
   A = zeros(K+1,3);
   for n=1:2:2*K
       Arow = p(n:1:n+1,:);  Arow = poly(Arow);
       A(fix((n+1)/2),:) = real(Arow);
       end
       A(K+1,:) = [0 real(poly(p(Na)))];
end

% Numerator second-order sections:
z = cplxpair(roots(b)); K = floor(Nb/2);
if Nb == 0           % Computation when Nb = 0
   B = [0 0 poly(z)];

elseif K*2 == Nb     % Computation when Nb is even
   B = zeros(K,3);
   for n=1:2:Nb
       Brow = z(n:1:n+1,:);  Brow = poly(Brow);
       B(fix((n+1)/2),:) = real(Brow);
   end

elseif Nb == 1       % Computation when Nb = 1
       B = [0 real(poly(z))];

else                 % Computation when Nb is odd and > 1
   B = zeros(K+1,3);
   for n=1:2:2*K
       Brow = z(n:1:n+1,:);  Brow = poly(Brow);
       B(fix((n+1)/2),:) = real(Brow);
   end
   B(K+1,:) = [0 real(poly(z(Nb)))];
end
```

□ **EXAMPLE 8.2** Design a 3rd-order Butterworth analog prototype filter with $\Omega_c = 0.5$ given in Example 8.1.

Solution MATLAB script:

```
>> N = 3; OmegaC = 0.5;  [b,a] = u_buttap(N,OmegaC);
>> [C,B,A] = sdir2cas(b,a)
C = 0.1250
B = 0     0     1
A = 1.0000    0.5000    0.2500
         0    1.0000    0.5000
```

The cascade form coefficients agree with those in Example 8.1. □

8.3.3 DESIGN EQUATIONS

The analog lowpass filter is specified by the parameters Ω_p, R_p, Ω_s, and A_s. Therefore the essence of the design in the case of Butterworth filter is to obtain the order N and the cutoff frequency Ω_c, given these specifications. We want

- at $\Omega = \Omega_p$, $-10\log_{10}|H_a(j\Omega)|^2 = R_p$ or

$$-10\log_{10}\left(\frac{1}{1+\left(\dfrac{\Omega_p}{\Omega_c}\right)^{2N}}\right) = R_p$$

and

- at $\Omega = \Omega_s$, $-10\log_{10}|H_a(j\Omega)|^2 = A_s$ or

$$-10\log_{10}\left(\frac{1}{1+\left(\dfrac{\Omega_s}{\Omega_c}\right)^{2N}}\right) = A_s$$

Solving these two equations for N and Ω_c, we have

$$N = \frac{\log_{10}\left[\left(10^{R_p/10}-1\right)/\left(10^{A_s/10}-1\right)\right]}{2\log_{10}\left(\Omega_p/\Omega_s\right)}$$

In general, N will not be an integer. Since we want N to be an integer, we must choose

$$N = \left\lceil \frac{\log_{10}\left[\left(10^{R_p/10}-1\right)/\left(10^{A_s/10}-1\right)\right]}{2\log_{10}\left(\Omega_p/\Omega_s\right)} \right\rceil \tag{8.49}$$

where the operation $\lceil x \rceil$ means "choose the smallest integer larger than x"—for example, $\lceil 4.5 \rceil = 5$. Since the actual N chosen is larger than required, specifications can be either met or exceeded either at Ω_p or at Ω_s. To satisfy the specifications exactly at Ω_p,

$$\Omega_c = \frac{\Omega_p}{\sqrt[2N]{\left(10^{R_p/10} - 1\right)}} \tag{8.50}$$

or, to satisfy the specifications exactly at Ω_s,

$$\Omega_c = \frac{\Omega_s}{\sqrt[2N]{\left(10^{A_s/10} - 1\right)}} \tag{8.51}$$

□ **EXAMPLE 8.3** Design a lowpass Butterworth filter to satisfy

Passband cutoff: $\Omega_p = 0.2\pi$; Passband ripple: $R_p = 7\text{dB}$

Stopband cutoff: $\Omega_s = 0.3\pi$; Stopband ripple: $A_s = 16\text{dB}$

Solution From (8.49)

$$N = \left\lceil \frac{\log_{10}\left[\left(10^{0.7} - 1\right) / \left(10^{1.6} - 1\right)\right]}{2 \log_{10}\left(0.2\pi / 0.3\pi\right)} \right\rceil = \lceil 2.79 \rceil = 3$$

To satisfy the specifications exactly at Ω_p, from (8.50) we obtain

$$\Omega_c = \frac{0.2\pi}{\sqrt[6]{(10^{0.7} - 1)}} = 0.4985$$

To satisfy specifications exactly at Ω_s, from (8.51) we obtain

$$\Omega_c = \frac{0.3\pi}{\sqrt[6]{(10^{1.6} - 1)}} = 0.5122$$

Now we can choose any Ω_c between the above two numbers. Let us choose $\Omega_c = 0.5$. We have to design a Butterworth filter with $N = 3$ and $\Omega_c = 0.5$, which we did in Example 8.1. Hence

$$H_a(j\Omega) = \frac{0.125}{(s + 0.5)\left(s^2 + 0.5s + 0.25\right)}$$

□

8.3.4 MATLAB IMPLEMENTATION

The preceding design procedure can be implemented in MATLAB as a simple function. Using the `U_buttap` function, we provide the `afd_butt` function to design an analog Butterworth lowpass filter, given its specifications. This function uses (8.50).

```
function [b,a] = afd_butt(Wp,Ws,Rp,As);
% Analog Lowpass Filter Design: Butterworth
% -----------------------------------------
% [b,a] = afd_butt(Wp,Ws,Rp,As);
%  b = Numerator coefficients of Ha(s)
%  a = Denominator coefficients of Ha(s)
% Wp = Passband edge frequency in rad/sec; Wp > 0
% Ws = Stopband edge frequency in rad/sec; Ws > Wp > 0
% Rp = Passband ripple in +dB; (Rp > 0)
% As = Stopband attenuation in +dB; (As > 0)
%
if Wp <= 0
        error('Passband edge must be larger than 0')
end
if Ws <= Wp
        error('Stopband edge must be larger than Passband edge')
end
if (Rp <= 0) | (As < 0)
        error('PB ripple and/or SB attenuation ust be larger than 0')
end

N = ceil((log10((10^(Rp/10)-1)/(10^(As/10)-1)))/(2*log10(Wp/Ws)));
fprintf('\n*** Butterworth Filter Order = %2.0f \n',N)
OmegaC = Wp/((10^(Rp/10)-1)^(1/(2*N)));
[b,a]=u_buttap(N,OmegaC);
```

To display the frequency-domain plots of analog filters, we provide a function called `freqs_m`, which is a modified version of a function `freqs` provided by MATLAB. This function computes the magnitude response in absolute as well as in relative dB scale and the phase response. This function is similar to the `freqz_m` function discussed earlier. One main difference between them is that in the `freqs_m` function the responses are computed up to a maximum frequency $\Omega_{\max}$.

```
function [db,mag,pha,w] = freqs_m(b,a,wmax);
% Computation of s-domain frequency response: Modified version
% ------------------------------------------------------------
% [db,mag,pha,w] = freqs_m(b,a,wmax);
%   db = Relative magnitude in db over [0 to wmax]
%  mag = Absolute magnitude over [0 to wmax]
%  pha = Phase response in radians over [0 to wmax]
%    w = array of 500 frequency samples between [0 to wmax]
%    b = Numerator polynomial coefficents of Ha(s)
%    a = Denominator polynomial coefficents of Ha(s)
% wmax = Maximum frequency in rad/sec over which response is desired
%
w = [0:1:500]*wmax/500;  H = freqs(b,a,w);
mag = abs(H);  db = 20*log10((mag+eps)/max(mag));  pha = angle(H);
```

The impulse response $h_a(t)$ of the analog filter is computed using MATLAB's `impulse` function.

□ **EXAMPLE 8.4** Design the analog Butterworth lowpass filter specified in Example 8.3 using MATLAB.

Solution MATLAB script:

```
>> Wp = 0.2*pi; Ws = 0.3*pi; Rp = 7; As = 16;
>> Ripple = 10 ^ (-Rp/20); Attn = 10 ^ (-As/20);
>> % Analog filter design:
>> [b,a] = afd_butt(Wp,Ws,Rp,As);
*** Butterworth Filter Order =  3
>> % Calculation of second-order sections:
>> [C,B,A] = sdir2cas(b,a)
C = 0.1238
B = 0     0     1
A = 1.0000    0.4985    0.2485
         0    1.0000    0.4985
>> % Calculation of Frequency Response:
>> [db,mag,pha,w] = freqs_m(b,a,0.5*pi);
>> % Calculation of Impulse response:
>> [ha,x,t] = impulse(b,a);
```

The system function is given by

$$H_a(s) = \frac{0.1238}{(s^2 + 0.4985s + 0.2485)\,(s + 0.4985)}$$

This $H_a(s)$ is slightly different from the one in Example 8.3 because in that example we used $\Omega_c = 0.5$, while in the `afd_butt` function Ω_c is chosen to satisfy the specifications at Ω_p. The filter plots are shown in Figure 8.15. □

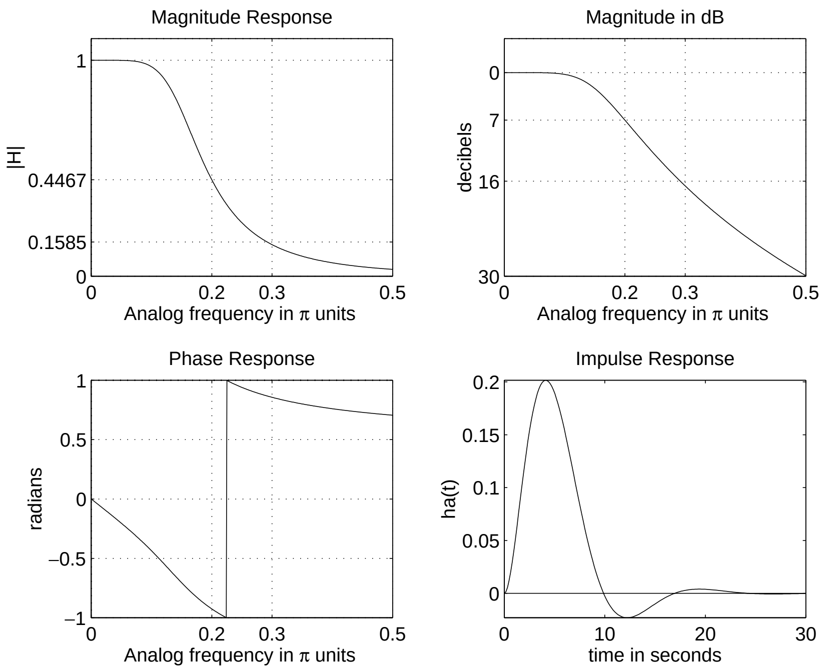


FIGURE 8.15 *Butterworth analog filter in Example 8.4*

8.3.5 CHEBYSHEV LOWPASS FILTERS

There are two types of Chebyshev filters. The Chebyshev-I filters have *equiripple response in the passband*, while the Chebyshev-II filters have *equiripple response in the stopband*. Butterworth filters have monotonic response in both bands. Recall our discussions regarding equiripple FIR filters. We noted that by choosing a filter that has an equiripple rather than a monotonic behavior, we can obtain a lower-order filter. Therefore Chebyshev filters provide lower order than Butterworth filters for the same specifications.

The magnitude-squared response of a Chebyshev-I filter is

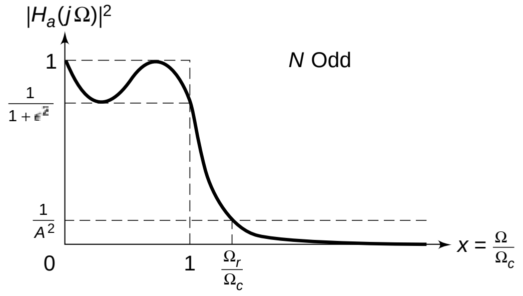
$$|H_a(j\Omega)|^2 = \frac{1}{1+\epsilon^2 T_N^2\left(\dfrac{\Omega}{\Omega_c}\right)} \qquad \textbf{(8.52)}$$

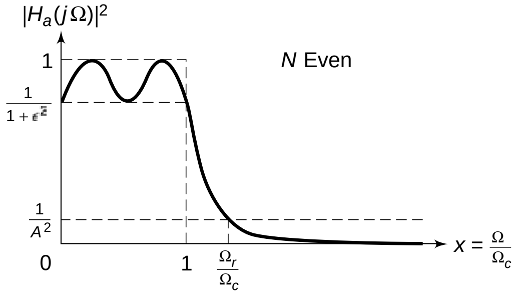
where N is the order of the filter, ϵ is the passband ripple factor, which is related to R_p, and $T_N\left(x\right)$ is the Nth-order Chebyshev polynomial given by

$$T_N(x) = \begin{cases} \cos\left(N\cos^{-1}(x)\right), & 0 \le x \le 1 \\ \cosh\left(\cosh^{-1}(x)\right), & 1 < x < \infty \end{cases} \quad \text{where } x = \frac{\Omega}{\Omega_c}$$

The equiripple response of the Chebyshev filters is due to this polynomial $T_N(x)$. Its key properties are (a) for $0 < x < 1$, $T_N(x)$ oscillates between -1 and 1, and (b) for $1 < x < \infty$, $T_N(x)$ increases monotonically to ∞.

There are two possible shapes of $|H_a(j\Omega)|^2$, one for N odd and one for N even as shown here. Note that $x = \Omega/\Omega_c$ is the normalized frequency.





From these two response plots we observe the following properties:

- At $x = 0$ (or $\Omega = 0$); $|H_a(j0)|^2 = 1$ for N odd.
  $|H_a(j0)|^2 = \dfrac{1}{1+\epsilon^2}$ for N even.
- At $x = 1$ (or $\Omega = \Omega_c$); $|H_a(j1)|^2 = \dfrac{1}{1+\epsilon^2}$ for all N.
- For $0 \le x \le 1$ (or $0 \le \Omega \le \Omega_c$), $|H_a(jx)|^2$ oscillates between 1 and $\dfrac{1}{1+\epsilon^2}$.
- For $x > 1$ (or $\Omega > \Omega_c$), $|H_a(jx)|^2$ decreases monotonically to 0.
- At $x = \Omega_r$, $|H_a(jx)|^2 = \dfrac{1}{A^2}$.

To determine a causal and stable $H_a(s)$, we must find the poles of $H_a(s)H_a(-s)$ and select the left half-plane poles for $H_a(s)$. The poles of $H_a(s)H_a(-s)$ are obtained by finding the roots of

$$1 + \epsilon^2 T_N^2\left(\frac{s}{j\Omega_c}\right)$$

The solution of this equation is tedious if not difficult to obtain. It can be shown that if $p_k = \sigma_k + j\Omega_k, \quad k = 0, \ldots, N-1$ are the (left half-plane) roots of these polynomial, then

$$\begin{aligned}\sigma_k &= (a\Omega_c)\cos\left[\frac{\pi}{2}+\frac{(2k+1)\pi}{2N}\right]\\ \Omega_k &= (b\Omega_c)\sin\left[\frac{\pi}{2}+\frac{(2k+1)\pi}{2N}\right]\end{aligned}\quad k=0,\ldots,N-1 \tag{8.53}$$

where

$$a=\frac{1}{2}\left(\sqrt[N]{\alpha}-\sqrt[N]{1/\alpha}\right),\quad b=\frac{1}{2}\left(\sqrt[N]{\alpha}+\sqrt[N]{1/\alpha}\right),\quad \text{and}\quad \alpha=\frac{1}{\epsilon}+\sqrt{1+\frac{1}{\epsilon^2}} \tag{8.54}$$

These roots fall on an ellipse with major axis $b\Omega_c$ and minor axis $a\Omega_c$. Now the system function is given by

$$H_a(s)=\frac{K}{\prod\limits_k (s-p_k)} \tag{8.55}$$

where K is a normalizing factor chosen to make

$$H_a(j0)=\begin{cases}1, & N \text{ odd}\\ \dfrac{1}{\sqrt{1+\epsilon^2}}, & N \text{ even}\end{cases} \tag{8.56}$$

8.3.6 MATLAB IMPLEMENTATION

MATLAB provides a function called `[z,p,k]=cheb1ap(N,Rp)` to design a *normalized* Chebyshev-I analog prototype filter of order `N` and passband ripple `Rp` and that returns zeros in `z` array, poles in `p` array, and the gain value `k`. We need an unnormalized Chebyshev-I filter with arbitrary Ω_c. This is achieved by scaling the array `p` of the normalized filter by Ω_c. Similar to the Butterworth prototype, this filter has no zeros. The new gain `k` is determined using (8.56), which is achieved by scaling the old `k` by the ratio of the unnormalized to the normalized denominator polynomials evaluated at $s = 0$. In the following function, called `U_chb1ap(N,Rp,Omegac)`, we design an unnormalized Chebyshev-I analog prototype filter that returns $H_a(s)$ in the direct form.

```
function [b,a] = u_chb1ap(N,Rp,Omegac);
% Unnormalized Chebyshev-1 Analog Lowpass Filter Prototype
% -----------------------------------------------------
% [b,a] = u_chb1ap(N,Rp,Omegac);
%      b = numerator polynomial coefficients
%      a = denominator polynomial coefficients
%      N = Order of the Elliptic Filter
%     Rp = Passband Ripple in dB; Rp > 0
% Omegac = Cutoff frequency in radians/sec
%
[z,p,k] = cheb1ap(N,Rp);  a = real(poly(p));  aNn = a(N+1);
      p = p*Omegac;  a = real(poly(p));  aNu = a(N+1);
      k = k*aNu/aNn;
      b0 = k;  B = real(poly(z));  b = k*B;
```

8.3.7 DESIGN EQUATIONS

Given Ω_p, Ω_s, R_p, and A_S, three parameters are required to determine a Chebyshev-I filter: ϵ, Ω_c, and N. From equations (8.3) and (8.4), we obtain

$$\epsilon = \sqrt{10^{0.1R_p} - 1} \qquad \text{and} \qquad A = 10^{A_s/20}$$

From these properties, we have

$$\Omega_c = \Omega_p \qquad \text{and} \qquad \Omega_r = \frac{\Omega_s}{\Omega_p} \tag{8.57}$$

The order N is given by

$$g = \sqrt{(A^2 - 1)/\epsilon^2} \tag{8.58}$$

$$N = \left\lceil \frac{\log_{10}\left[g + \sqrt{g^2 - 1}\right]}{\log_{10}\left[\Omega_r + \sqrt{\Omega_r^2 - 1}\right]} \right\rceil \tag{8.59}$$

Now using (8.54), (8.53), and (8.55), we can determine $H_a(s)$.

□ **EXAMPLE 8.5** Design a lowpass Chebyshev-I filter to satisfy

Passband cutoff: $\Omega_p = 0.2\pi$; Passband ripple: $R_p = 1\text{dB}$

Stopband cutoff: $\Omega_s = 0.3\pi$; Stopband ripple: $A_s = 16\text{dB}$

Solution First compute the necessary parameters.

$$\epsilon = \sqrt{10^{0.1(1)} - 1} = 0.5088 \qquad A = 10^{16/20} = 6.3096$$

$$\Omega_c = \Omega_p = 0.2\pi \qquad \Omega_r = \frac{0.3\pi}{0.2\pi} = 1.5$$

$$g = \sqrt{(A^2 - 1)/\epsilon^2} = 12.2429 \qquad N = 4$$

Now we can determine $H_a(s)$.

$$\alpha = \frac{1}{\epsilon} + \sqrt{1 + \frac{1}{\epsilon^2}} = 4.1702$$

$$a = 0.5\left(\sqrt[N]{\alpha} - \sqrt[N]{1/\alpha}\right) = 0.3646$$

$$b = 0.5\left(\sqrt[N]{\alpha} + \sqrt[N]{1/\alpha}\right) = 1.0644$$

There are four poles for $H_a(s)$:

$$p_{0,3} = (a\Omega_c)\cos\left[\frac{\pi}{2} + \frac{\pi}{8}\right] \pm (b\Omega_c)\sin\left[\frac{\pi}{2} + \frac{\pi}{8}\right] = -0.0877 \pm j0.6179$$

$$p_{1,2} = (a\Omega_c)\cos\left[\frac{\pi}{2} + \frac{3\pi}{8}\right] \pm (b\Omega_c)\sin\left[\frac{\pi}{2} + \frac{3\pi}{8}\right] = -0.2117 \pm j0.2559$$

Hence

$$H_a(s) = \frac{K}{\prod_{k=0}^{3}(s - p_k)} = \frac{\overbrace{0.89125 \times .1103 \times .3895}^{0.03829}}{(s^2 + 0.1754s + 0.3895)\,(s^2 + 0.4234s + 0.1103)}$$

Note that the numerator is such that

$$H_a(j0) = \frac{1}{\sqrt{1 + \epsilon^2}} = 0.89125$$

□

8.3.8 MATLAB IMPLEMENTATION

Using the `U_chb1ap` function, we provide a function called `afd_chb1` to design an analog Chebyshev-II lowpass filter, given its specifications. This is shown below and uses the procedure described in Example 8.5.

```
function [b,a] = afd_chb1(Wp,Ws,Rp,As);
% Analog Lowpass Filter Design: Chebyshev-1
% -----------------------------------------
% [b,a] = afd_chb1(Wp,Ws,Rp,As);
%  b = Numerator coefficients of Ha(s)
%  a = Denominator coefficients of Ha(s)
% Wp = Passband edge frequency in rad/sec; Wp > 0
% Ws = Stopband edge frequency in rad/sec; Ws > Wp > 0
% Rp = Passband ripple in +dB; (Rp > 0)
% As = Stopband attenuation in +dB; (As > 0)
```

```
%
if Wp <= 0
        error('Passband edge must be larger than 0')
end
if Ws <= Wp
        error('Stopband edge must be larger than Passband edge')
end
if (Rp <= 0) | (As < 0)
        error('PB ripple and/or SB attenuation ust be larger than 0')
end

ep = sqrt(10^(Rp/10)-1);  A = 10^(As/20);
OmegaC = Wp;  OmegaR = Ws/Wp;  g = sqrt(A*A-1)/ep;
N = ceil(log10(g+sqrt(g*g-1))/log10(OmegaR+sqrt(OmegaR*OmegaR-1)));
fprintf('\n*** Chebyshev-1 Filter Order = %2.0f \n',N)
[b,a]=u_chb1ap(N,Rp,OmegaC);
```

☐ **EXAMPLE 8.6** Design the analog Chebyshev-I lowpass filter given in Example 8.5 using MATLAB.

Solution MATLAB script:

```
>> Wp = 0.2*pi; Ws = 0.3*pi; Rp = 1; As = 16;
>> Ripple = 10 ^ (-Rp/20); Attn = 10 ^ (-As/20);
>> % Analog filter design:
>> [b,a] = afd_chb1(Wp,Ws,Rp,As);
*** Chebyshev-1 Filter Order =  4
>> % Calculation of second-order sections:
>> [C,B,A] = sdir2cas(b,a)
C = 0.0383
B = 0     0     1
A = 1.0000    0.4233    0.1103
    1.0000    0.1753    0.3895
>> % Calculation of Frequency Response:
>> [db,mag,pha,w] = freqs_m(b,a,0.5*pi);
>> % Calculation of Impulse response:
>> [ha,x,t] = impulse(b,a);
```

The specifications are satisfied by a 4th-order Chebyshev-I filter whose system function is

$$H_a(s) = \frac{0.0383}{(s^2 + 4233s + 0.1103)\,(s^2 + 0.1753s + 0.3895)}$$

The filter plots are shown in Figure 8.16. ☐

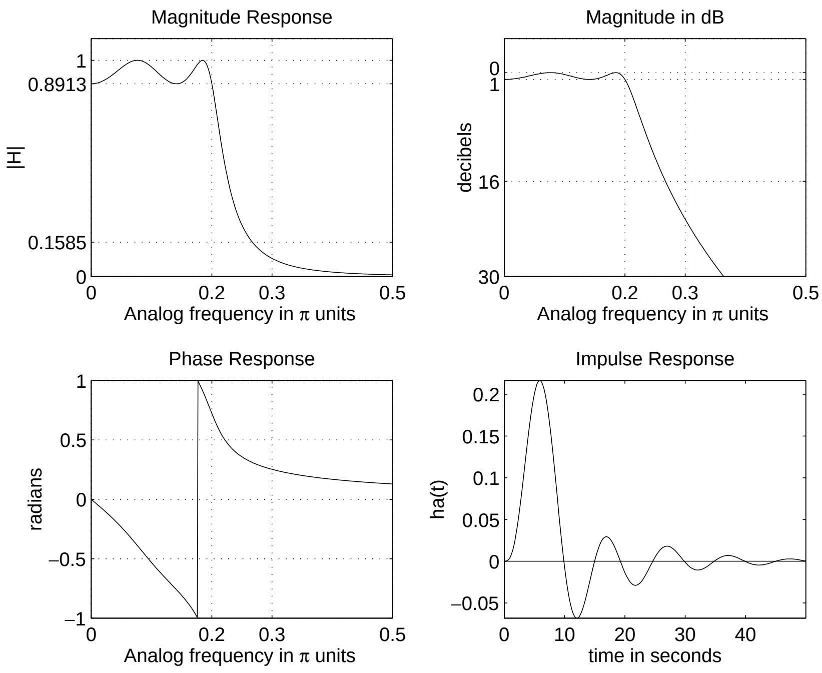


FIGURE 8.16 *Chebyshev-I analog filter in Example 8.6*

A Chebyshev-II filter is related to the Chebyshev-I filter through a simple transformation. It has a monotone passband and an equiripple stopband, which implies that this filter has both poles and zeros in the s-plane. Therefore the group delay characteristics are better (and the phase response more linear) in the passband than the Chebyshev-I prototype. If we replace the term $\epsilon^2 T_N^2(\Omega/\Omega_c)$ in (8.52) by its reciprocal and also the argument $x = \Omega/\Omega_c$ by its reciprocal, we obtain the magnitude-squared response of Chebyshev-II as

$$|H_a(j\Omega)|^2 = \frac{1}{1 + \left[\epsilon^2 T_N^2(\Omega_c/\Omega)\right]^{-1}} \tag{8.60}$$

One approach to designing a Chebyshev-II filter is to design the corresponding Chebyshev-I first and then apply these transformations. We will not discuss the details of this filter but will use a function from MATLAB to design a Chebyshev-II filter.

8.3.9 MATLAB IMPLEMENTATION

MATLAB provides a function called `[z,p,k]=cheb2ap(N,As)` to design a *normalized* Chebyshev-II analog prototype filter of order `N` and passband ripple `As` and that returns zeros in `z` array, poles in `p` array, and the gain value k. We need an unnormalized Chebyshev-I filter with arbitrary Ω_c. This is achieved by scaling the array p of the normalized filter by Ω_c. Since this filter has zeros, we also have to scale the array z by Ω_c. The new gain k is determined using (8.56), which is achieved by scaling the old k by the ratio of the unnormalized to the normalized rational functions evaluated at $s = 0$. In the following function, called U_chb2ap(N,As,Omegac), we design an unnormalized Chebyshev-II analog prototype filter that returns $H_a(s)$ in the direct form.

```
function [b,a] = u_chb2ap(N,As,Omegac);
% Unnormalized Chebyshev-2 Analog Lowpass Filter Prototype
% ---------------------------------------------------------
% [b,a] = u_chb2ap(N,As,Omegac);
%      b = numerator polynomial coefficients
%      a = denominator polynomial coefficients
%      N = Order of the Elliptic Filter
%     As = Stopband Ripple in dB; As > 0
% Omegac = Cutoff frequency in radians/sec
%
[z,p,k] = cheb2ap(N,As);
      a = real(poly(p));  aNn = a(N+1);
      p = p*Omegac;  a = real(poly(p));  aNu = a(N+1);
      b = real(poly(z));  M = length(b);  bNn = b(M);
      z = z*Omegac;  b = real(poly(z));  bNu = b(M);
      k = k*(aNu*bNn)/(aNn*bNu);
     b0 = k;  b = k*b;
```

The design equations for the Chebyshev-II prototype are similar to those of the Chebyshev-I except that $\Omega_c = \Omega_s$ since the ripples are in the stopband. Therefore we can develop a MATLAB function similar to the afd_chb1 function for the Chebyshev-II prototype.

```
function [b,a] = afd_chb2(Wp,Ws,Rp,As);
% Analog Lowpass Filter Design: Chebyshev-2
% -----------------------------------------
% [b,a] = afd_chb2(Wp,Ws,Rp,As);
%  b = Numerator coefficients of Ha(s)
%  a = Denominator coefficients of Ha(s)
% Wp = Passband edge frequency in rad/sec; Wp > 0
% Ws = Stopband edge frequency in rad/sec; Ws > Wp > 0
% Rp = Passband ripple in +dB; (Rp > 0)
% As = Stopband attenuation in +dB; (As > 0)
%
if Wp <= 0
        error('Passband edge must be larger than 0')
end
```

```
if Ws <= Wp
        error('Stopband edge must be larger than Passband edge')
end
if (Rp <= 0) | (As < 0)
        error('PB ripple and/or SB attenuation ust be larger than 0')
end

ep = sqrt(10^(Rp/10)-1);  A = 10^(As/20);
OmegaC = Wp;  OmegaR = Ws/Wp;  g = sqrt(A*A-1)/ep;
N = ceil(log10(g+sqrt(g*g-1))/log10(OmegaR+sqrt(OmegaR*OmegaR-1)));
fprintf('\n*** Chebyshev-2 Filter Order = %2.0f \n',N)
[b,a]=u_chb2ap(N,As,Ws);
```

□ **EXAMPLE 8.7** Design a Chebyshev-II analog lowpass filter to satisfy the specifications given in Example 8.5:

$$\text{Passband cutoff: } \Omega_p = 0.2\pi \text{ ; Passband ripple: } R_p = 1\text{dB}$$
$$\text{Stopband cutoff: } \Omega_s = 0.3\pi \text{ ; Stopband ripple: } A_s = 16\text{dB}$$

Solution MATLAB script:

```
>> Wp = 0.2*pi; Ws = 0.3*pi; Rp = 1; As = 16;
>> Ripple = 10 ^ (-Rp/20); Attn = 10 ^ (-As/20);
>> % Analog filter design:
>> [b,a] = afd_chb2(Wp,Ws,Rp,As);
*** Chebyshev-2 Filter Order =  4
>> % Calculation of second-order sections:
>> [C,B,A] = sdir2cas(b,a)
C = 0.1585
B = 1.0000         0    6.0654
    1.0000         0    1.0407
A = 1.0000    1.9521    1.4747
    1.0000    0.3719    0.6784
>> % Calculation of Frequency Response:
>> [db,mag,pha,w] = freqs_m(b,a,0.5*pi);
>> % Calculation of Impulse response:
>> [ha,x,t] = impulse(b,a);
```

The specifications are satisfied by a 4th-order Chebyshev-II filter whose system function is

$$H_a(s) = \frac{0.1585\left(s^2+6.0654\right)\left(s^2+1.0407\right)}{\left(s^2+1.9521s+1.4747\right)\left(s^2+0.3719s+0.6784\right)}$$

The filter plots are shown in Figure 8.17. □

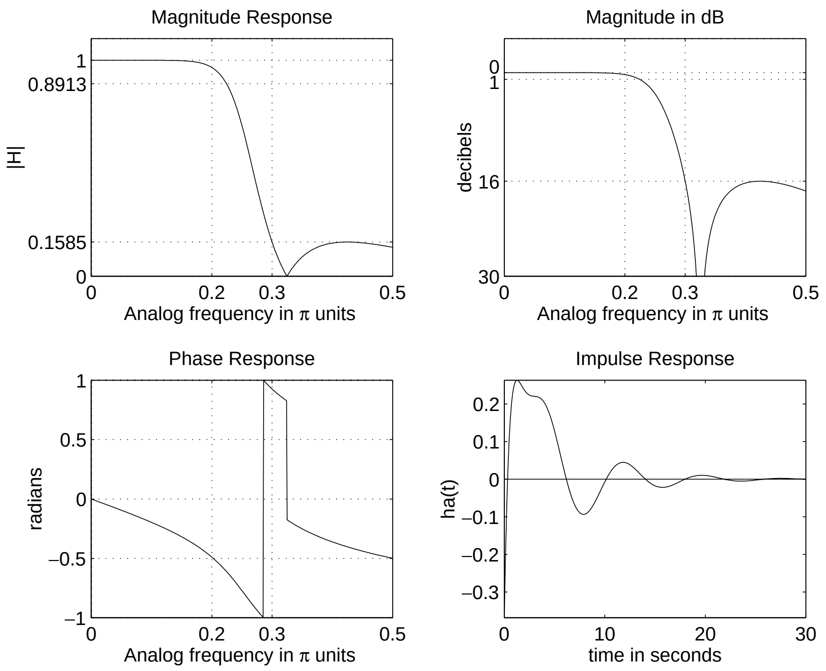


FIGURE 8.17 *Chebyshev-II analog filter in Example 8.7*

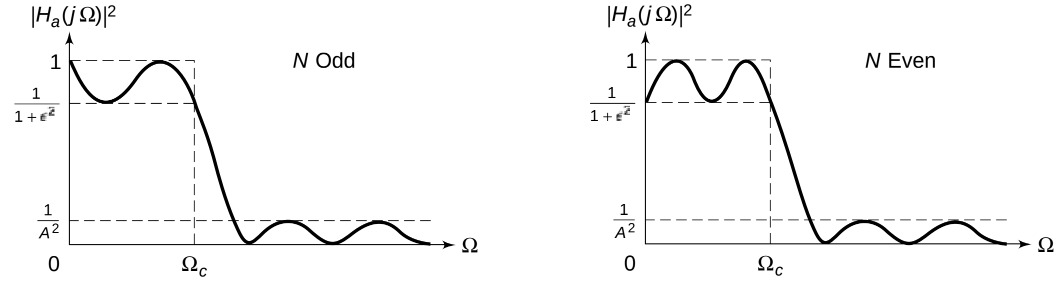
8.3.10 ELLIPTIC LOWPASS FILTERS

These filters exhibit equiripple behavior in the passband as well as in the stopband. They are similar in magnitude response characteristics to the FIR equiripple filters. Therefore elliptic filters are optimum filters in that they achieve the minimum order N for the given specifications (or alternately, achieve the sharpest transition band for the given order N). These filters, for obvious reasons, are very difficult to analyze and, therefore, to design. It is not possible to design them using simple tools, and often programs or tables are needed to design them.

The magnitude-squared response of elliptic filters is given by

$$|H_a(j\Omega)|^2 = \frac{1}{1+\epsilon^2 U_N^2\left(\dfrac{\Omega}{\Omega_c}\right)} \tag{8.61}$$

where N is the order, ϵ is the passband ripple (which is related to R_p), and $U_N(\cdot)$ is the Nth-order Jacobian elliptic function. The analysis of this function, even on a superficial level, is beyond the scope of this book. Note the similarity between the preceding response (8.61) and that of the Chebyshev filters given by (8.52). Typical responses for odd and even N are as follows.



8.3.11 COMPUTATION OF FILTER ORDER N

Even though the analysis of (8.61) is difficult, the order calculation formula is very compact and is available in many textbooks [18, 23, 24]. It is given by

$$N = \frac{K(k)K\left(\sqrt{1-k_1^2}\right)}{K\left(k_1\right)K\left(\sqrt{1-k^2}\right)} \tag{8.62}$$

where

$$k = \frac{\Omega_p}{\Omega_s}, \quad k_1 = \frac{\epsilon}{\sqrt{A^2-1}}$$

and

$$K(x) = \int_0^{\pi/2} \frac{d\theta}{\sqrt{1-x^2\sin^2\theta}}$$

is the complete elliptic integral of the first kind. MATLAB provides the function `ellipke` to numerically compute the above integral, which we will use to compute N and to design elliptic filters.

8.3.12 MATLAB IMPLEMENTATION

MATLAB provides a function called `[z,p,k]=ellipap(N,Rp,As)` to design a *normalized* elliptic analog prototype filter of order `N`, passband ripple `Rp`, and stopband attenuation `As`, and that returns zeros in `z` array, poles in `p` array, and the gain value `k`. We need an unnormalized elliptic filter with arbitrary Ω_c. This is achieved by scaling the arrays `p` and `z` of the normalized filter by Ω_c and the gain `k` by the ratio of the unnormalized to the normalized rational functions evaluated at $s = 0$. In the following function, called `U_elipap(N,Rp,As,Omegac)`, we design an unnormalized elliptic analog prototype filter that returns $H_a(s)$ in the direct form.

```
function [b,a] = u_elipap(N,Rp,As,Omegac);
% Unnormalized Elliptic Analog Lowpass Filter Prototype
% -------------------------------------------------------
% [b,a] = u_elipap(N,Rp,As,Omegac);
%      b = numerator polynomial coefficients
%      a = denominator polynomial coefficients
%      N = Order of the Elliptic Filter
%     Rp = Passband Ripple in dB; Rp > 0
%     As = Stopband Attenuation in dB; As > 0
% Omegac = Cutoff frequency in radians/sec
%
[z,p,k] = ellipap(N,Rp,As);
      a = real(poly(p));  aNn = a(N+1);
      p = p*Omegac;  a = real(poly(p));  aNu = a(N+1);
      b = real(poly(z));  M = length(b);  bNn = b(M);
      z = z*Omegac;  b = real(poly(z));  bNu = b(M);
      k = k*(aNu*bNn)/(aNn*bNu);
     b0 = k;  b = k*b;
```

Using the `U_elipap` function, we provide a function called `afd_elip` to design an analog elliptic lowpass filter, given its specifications. This follows and uses the filter order computation formula given in (8.62).

```
function [b,a] = afd_elip(Wp,Ws,Rp,As);
% Analog Lowpass Filter Design: Elliptic
% -------------------------------------
% [b,a] = afd_elip(Wp,Ws,Rp,As);
%  b = Numerator coefficients of Ha(s)
%  a = Denominator coefficients of Ha(s)
% Wp = Passband edge frequency in rad/sec; Wp > 0
% Ws = Stopband edge frequency in rad/sec; Ws > Wp > 0
% Rp = Passband ripple in +dB; (Rp > 0)
% As = Stopband attenuation in +dB; (As > 0)
%
if Wp <= 0
        error('Passband edge must be larger than 0')
end
if Ws <= Wp
        error('Stopband edge must be larger than Passband edge')
end
if (Rp <= 0) | (As < 0)
        error('PB ripple and/or SB attenuation ust be larger than 0')
end

ep = sqrt(10^(Rp/10)-1);  A = 10^(As/20);
OmegaC = Wp;  k = Wp/Ws;  k1 = ep/sqrt(A*A-1);
```

```
capk = ellipke([k.^2 1-k.^2]); % Version 4.0 code
capk1 = ellipke([(k1 .^2) 1-(k1 .^2)]); % Version 4.0 code
N = ceil(capk(1)*capk1(2)/(capk(2)*capk1(1)));
fprintf('\n*** Elliptic Filter Order = %2.0f \n',N)
[b,a]=u_elipap(N,Rp,As,OmegaC);
```

□ **EXAMPLE 8.8** Design an analog elliptic lowpass filter to satisfy the following specifications of Example 8.5:

$$\Omega_p = 0.2\pi,\ R_p = 1\,\text{dB}$$
$$\Omega_s = 0.3\pi,\ A_s = 16\,\text{db}$$

Solution MATLAB script:

```
>> Wp = 0.2*pi; Ws = 0.3*pi; Rp = 1; As = 16;
>> Ripple = 10 ^ (-Rp/20); Attn = 10 ^ (-As/20);
>> % Analog filter design:
>> [b,a] = afd_elip(Wp,Ws,Rp,As);
*** Elliptic Filter Order =  3
>> % Calculation of second-order sections:
>> [C,B,A] = sdir2cas(b,a)
C = 0.2740
B = 1.0000         0    0.6641
A = 1.0000    0.1696    0.4102
         0    1.0000    0.4435
>> % Calculation of Frequency Response:
>> [db,mag,pha,w] = freqs_m(b,a,0.5*pi);
>> % Calculation of Impulse response:
>> [ha,x,t] = impulse(b,a);
```

The specifications are satisfied by a 3rd-order elliptic filter whose system function is

$$H_a(s) = \frac{0.274\left(s^2 + 0.6641\right)}{\left(s^2 + 0.1696s + 0.4102\right)\left(s + 0.4435\right)}$$

The filter plots are shown in Figure 8.18. □

8.3.13 PHASE RESPONSES OF PROTOTYPE FILTERS

Elliptic filters provide optimal performance in the magnitude-squared response but have highly nonlinear phase response in the passband (which is undesirable in many applications). Even though we decided not to worry

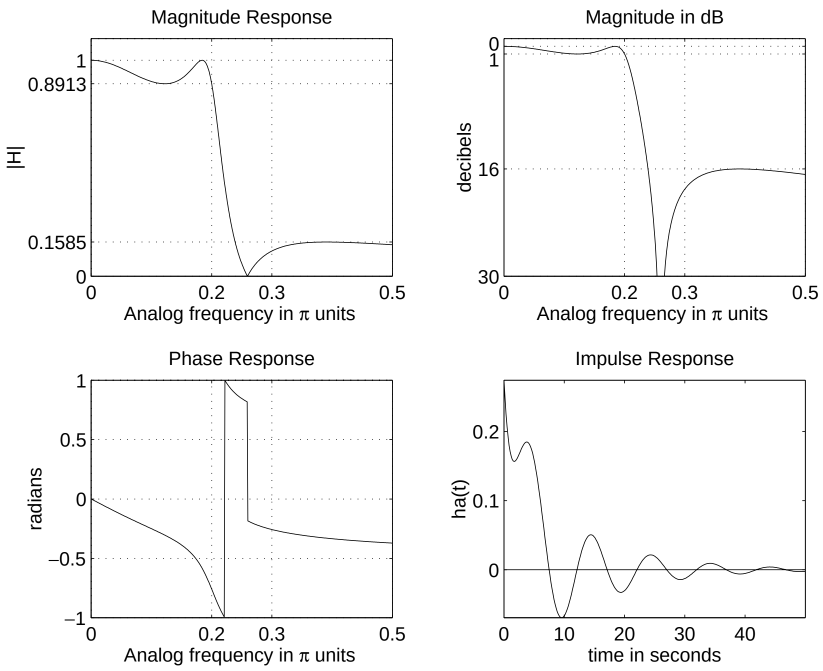


FIGURE 8.18 *Elliptic analog lowpass filter in Example 8.8*

about phase response in our designs, phase is still an important issue in the overall system. At the other end of the performance scale are the Butterworth filters, which have maximally flat magnitude response and require a higher-order N (more poles) to achieve the same stopband specification. However, they exhibit a fairly linear phase response in their passband. The Chebyshev filters have phase characteristics that lie somewhere in between. Therefore in practical applications we do consider Butterworth as well as Chebyshev filters, in addition to elliptic filters. The choice depends on both the filter order (which influences processing speed and implementation complexity) and the phase characteristics (which control the distortion).

8.4 ANALOG-TO-DIGITAL FILTER TRANSFORMATIONS

After discussing different approaches to the design of analog filters, we are now ready to transform them into digital filters. These transformations are complex-valued mappings that are extensively studied in the literature. These transformations are derived by preserving different aspects of analog and digital filters. If we want to preserve the shape of the impulse response from analog to digital filter, then we obtain a technique called *impulse invariance* transformation. If we want to convert a differential equation representation into a corresponding difference equation representation, then we obtain a *finite difference approximation* technique. Numerous other techniques are also possible. One technique, called *step invariance*, preserves the shape of the step response; this is explored in Problem P8.24. Another technique that is similar to the impulse invariance is the matched-z transformation, which matches the pole-zero representation. It is described at the end of this section and is explored in Problem P8.26. The most popular technique used in practice is called a *Bilinear* transformation, which preserves the system function representation from analog to digital domain. In this section we will study in detail impulse invariance and bilinear transformations, both of which can be easily implemented in MATLAB.

8.4.1 IMPULSE INVARIANCE TRANSFORMATION

In this design method we want the digital filter impulse response to look "similar" to that of a frequency-selective analog filter. Hence we sample $h_a(t)$ at some sampling interval T to obtain $h(n)$; that is,

$$h(n) = h_a(nT)$$

The parameter T is chosen so that the shape of $h_a(t)$ is "captured" by the samples. Since this is a sampling operation, the analog and digital frequencies are related by

$$\omega = \Omega T \text{ or } e^{j\omega} = e^{j\Omega T}$$

Since $z = e^{j\omega}$ on the unit circle and $s = j\Omega$ on the imaginary axis, we have the following transformation from the s-plane to the z-plane:

$$z = e^{sT} \tag{8.63}$$

The system functions $H(z)$ and $H_a(s)$ are related through the frequency-domain aliasing formula (3.27):

$$H(z) = \frac{1}{T}\sum_{k=-\infty}^{\infty} H_a\left(s - j\frac{2\pi}{T}k\right)$$

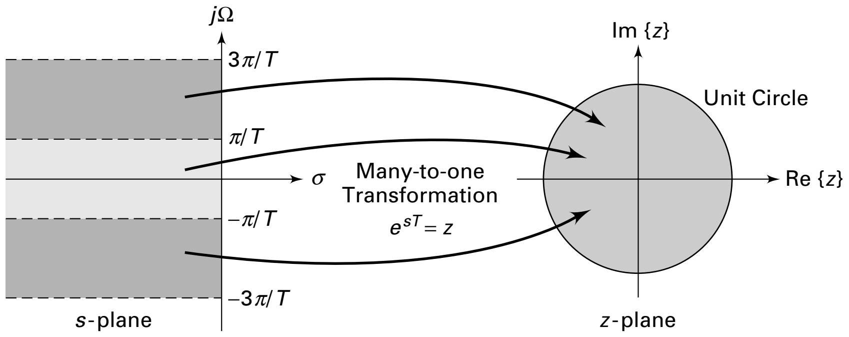


FIGURE 8.19 *Complex-plane mapping in impulse invariance transformation*

The complex plane transformation under the mapping (8.63) is shown in Figure 8.19, from which we have the following observations:

1. Using $\sigma = \text{Re}(s)$, we note that

$$\sigma < 0 \text{ maps into } |z| < 1 \text{ (inside of the UC)}$$
$$\sigma = 0 \text{ maps onto } |z| = 1 \text{ (on the UC)}$$
$$\sigma > 0 \text{ maps into } |z| > 1 \text{ (outside of the UC)}$$

2. All semi-infinite strips (shown above) of width $2\pi/T$ map into $|z| < 1$. Thus this mapping is not unique but a *many-to-one* mapping.
3. Since the entire left half of the s-plane maps into the unit circle, a causal and stable analog filter maps into a causal and stable digital filter.
4. If $H_a(j\Omega) = H_a(j\omega/T) = 0$ for $|\Omega| \geq \pi/T$, then

$$H(e^{j\omega}) = \frac{1}{T} H_a(j\omega/T), \quad |\omega| \leq \pi$$

and there will be no aliasing. However, no analog filter of finite order can be exactly band-limited. Therefore some aliasing error will occur in this design procedure, and hence the sampling interval T plays a minor role in this design method.

8.4.2 DESIGN PROCEDURE

Given the digital lowpass filter specifications ω_p, ω_s, R_p, and A_s, we want to determine $H(z)$ by first designing an equivalent analog filter and then mapping it into the desired digital filter. The steps required for this procedure are

1. Choose T and determine the analog frequencies

$$\Omega_p = \frac{\omega_p}{T_p} \qquad \text{and} \qquad \Omega_s = \frac{\omega_s}{T}$$

2. Design an analog filter $H_a(s)$ using the specifications Ω_p, Ω_s, R_p, and A_s. This can be done using any one of the three (Butterworth, Chebyshev, or elliptic) prototypes of the previous section.
3. Using partial fraction expansion, expand $H_a(s)$ into

$$H_a(s) = \sum_{k=1}^{N} \frac{R_k}{s - p_k}$$

4. Now transform analog poles $\{p_k\}$ into digital poles $\{e^{p_k T}\}$ to obtain the digital filter:

$$H(z) = \sum_{k=1}^{N} \frac{R_k}{1 - e^{p_k T} z^{-1}} \tag{8.64}$$

□ **EXAMPLE 8.9** Transform

$$H_a(s) = \frac{s+1}{s^2 + 5s + 6}$$

into a digital filter $H(z)$ using the impulse invariance technique in which $T = 0.1$.

Solution We first expand $H_a(s)$ using partial fraction expansion:

$$H_a(s) = \frac{s+1}{s^2+5s+6} = \frac{2}{s+3} - \frac{1}{s+2}$$

The poles are at $p_1 = -3$ and $p_2 = -2$. Then from (8.64) and using $T = 0.1$, we obtain

$$H(z) = \frac{2}{1 - e^{-3T}z^{-1}} - \frac{1}{1 - e^{-2T}z^{-1}} = \frac{1 - 0.8966z^{-1}}{1 - 1.5595z^{-1} + 0.6065z^{-2}}$$

It is easy to develop a MATLAB function to implement the impulse invariance mapping. Given a rational function description of $H_a(s)$, we can use the `residue` function to obtain its pole-zero description. Then each analog pole is mapped into a digital pole using (8.63). Finally, the `residuez` function can be used to convert $H(z)$ into rational function form. This procedure is given in the function `imp_invr`.

```
function [b,a] = imp_invr(c,d,T)
% Impulse Invariance Transformation from Analog to Digital Filter
% ---------------------------------------------------------------
% [b,a] = imp_invr(c,d,T)
%  b = Numerator polynomial in z^(-1) of the digital filter
%  a = Denominator polynomial in z^(-1) of the digital filter
%  c = Numerator polynomial in s of the analog filter
%  d = Denominator polynomial in s of the analog filter
%  T = Sampling (transformation) parameter
%
[R,p,k] = residue(c,d);  p = exp(p*T);
[b,a] = residuez(R,p,k);  b = real(b'); a = real(a');
```

A similar function called `impinvar` is available in the SP toolbox of MATLAB. □

□ **EXAMPLE 8.10** We demonstrate the use of the `imp_invr` function on the system function from Example 8.9.

Solution MATLAB script:

```
>> c = [1,1]; d = [1,5,6]; T = 0.1;
>> [b,a] = imp_invr(c,d,T)
b =  1.0000   -0.8966
a =  1.0000   -1.5595    0.6065
```

The digital filter is

$$H(z) = \frac{1 - 0.8966z^{-1}}{1 - 1.5595z^{-1} + 0.6065z^{-2}}$$

as expected. In Figure 8.20 we show the impulse responses and the magnitude responses (plotted up to the sampling frequency $1/T$) of the analog and the resulting digital filter. Clearly, the aliasing in the frequency domain is evident. □

In the next several examples we illustrate the impulse invariance design procedure on all three prototypes.

□ **EXAMPLE 8.11** Design a lowpass digital filter using a Butterworth prototype to satisfy

$$\omega_p = 0.2\pi,\ R_p = 1 \text{ dB}$$
$$\omega_s = 0.3\pi,\ A_s = 15 \text{ dB}$$

Solution The design procedure is described in the following MATLAB script:

```
>> % Digital Filter Specifications:
>> wp = 0.2*pi;                        % digital Passband freq in Hz
>> ws = 0.3*pi;                        % digital Stopband freq in Hz
>> Rp = 1;                             % Passband ripple in dB
>> As = 15;                            % Stopband attenuation in dB
```

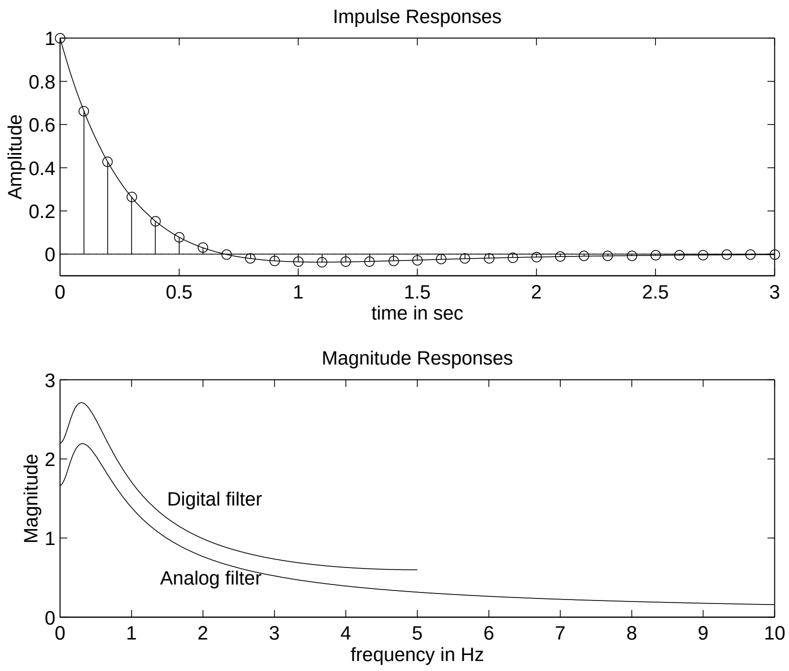


FIGURE 8.20 *Impulse and frequency response plots in Example 8.10*

```
>> % Analog Prototype Specifications: Inverse mapping for frequencies
>> T = 1;                                   % Set T=1
>> OmegaP = wp / T;                         % Prototype Passband freq
>> OmegaS = ws / T;                         % Prototype Stopband freq

>> % Analog Butterworth Prototype Filter Calculation:
>> [cs,ds] = afd_butt(OmegaP,OmegaS,Rp,As);
*** Butterworth Filter Order =  6

>> % Impulse Invariance transformation:
>> [b,a] = imp_invr(cs,ds,T);  [C,B,A] = dir2par(b,a)
C =  []
B = 1.8557   -0.6304
   -2.1428    1.1454
    0.2871   -0.4466
A = 1.0000   -0.9973    0.2570
    1.0000   -1.0691    0.3699
    1.0000   -1.2972    0.6949
```

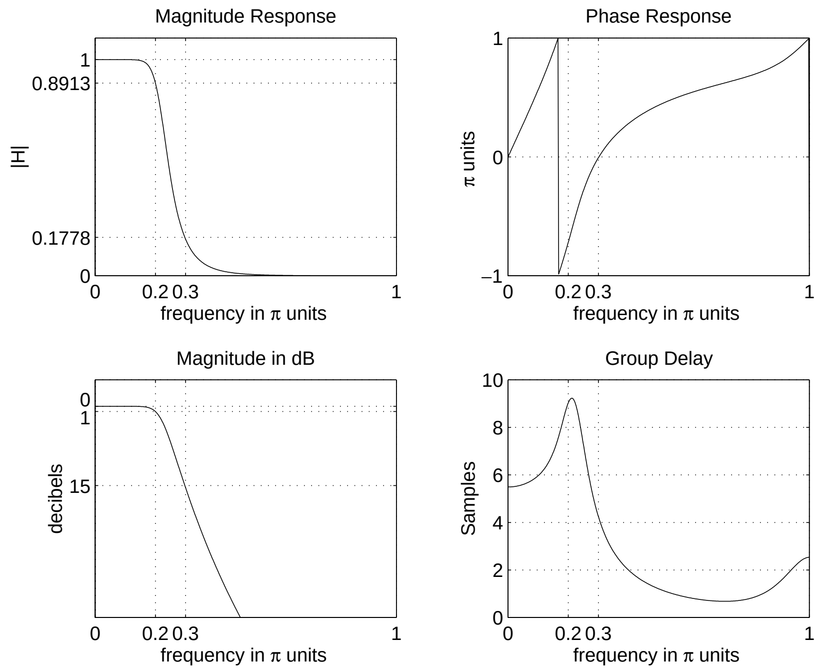


FIGURE 8.21 *Digital Butterworth lowpass filter using impulse invariance design*

The desired filter is a 6th-order Butterworth filter whose system function $H(z)$ is given in the parallel form

$$H(z) = \frac{1.8587 - 0.6304z^{-1}}{1 - 0.9973z^{-1} + 0.257z^{-2}} + \frac{-2.1428 + 1.1454z^{-1}}{1 - 1.0691z^{-1} + 0.3699z^{-2}} + \frac{0.2871 - 0.4463z^{-1}}{1 - 1.2972z^{-1} + 0.6449z^{-2}}$$

The frequency response plots are given in Figure 8.21. □

□ **EXAMPLE 8.12** Design a lowpass digital filter using a Chebyshev-I prototype to satisfy

$$\omega_p = 0.2\pi,\ R_p = 1 \text{ dB}$$
$$\omega_s = 0.3\pi,\ A_s = 15 \text{ dB}$$

Solution The design procedure is described in the following MATLAB script:

```
>> % Digital Filter Specifications:
>> wp = 0.2*pi;                            % digital Passband freq in rad
>> ws = 0.3*pi;                            % digital Stopband freq in rad
>> Rp = 1;                                 % Passband ripple in dB
>> As = 15;                                % Stopband attenuation in dB

>> % Analog Prototype Specifications: Inverse mapping for frequencies
>> T = 1;                                  % Set T=1
>> OmegaP = wp / T;                        % Prototype Passband freq
>> OmegaS = ws / T;                        % Prototype Stopband freq
>> % Analog Chebyshev-1 Prototype Filter Calculation:
>> [cs,ds] = afd_chb1(OmegaP,OmegaS,Rp,As);
*** Chebyshev-1 Filter Order =  4

>> % Impulse Invariance transformation:
>> [b,a] = imp_invr(cs,ds,T);  [C,B,A] = dir2par(b,a)
C =  []
B =-0.0833   -0.0246
    0.0833    0.0239
A = 1.0000   -1.4934    0.8392
    1.0000   -1.5658    0.6549
```

The desired filter is a 4th-order Chebyshev-I filter whose system function $H(z)$ is

$$H(z) = \frac{-0.0833 - 0.0246z^{-1}}{1 - 1.4934z^{-1} + 0.8392z^{-2}} + \frac{-0.0833 + 0.0239z^{-1}}{1 - 1.5658z^{-1} + 0.6549z^{-2}}$$

The frequency response plots are given in Figure 8.22. □

□ **EXAMPLE 8.13** Design a lowpass digital filter using a Chebyshev-II prototype to satisfy

$$\omega_p = 0.2\pi,\ R_p = 1 \text{ dB}$$
$$\omega_s = 0.3\pi,\ A_s = 15 \text{ dB}$$

Solution Recall that the Chebyshev-II filter is equiripple in the stopband. It means that this analog filter has a response that does not go to zero at high frequencies in the stopband. Therefore after impulse invariance transformation, the aliasing effect will be significant; this can degrade the passband response. The MATLAB

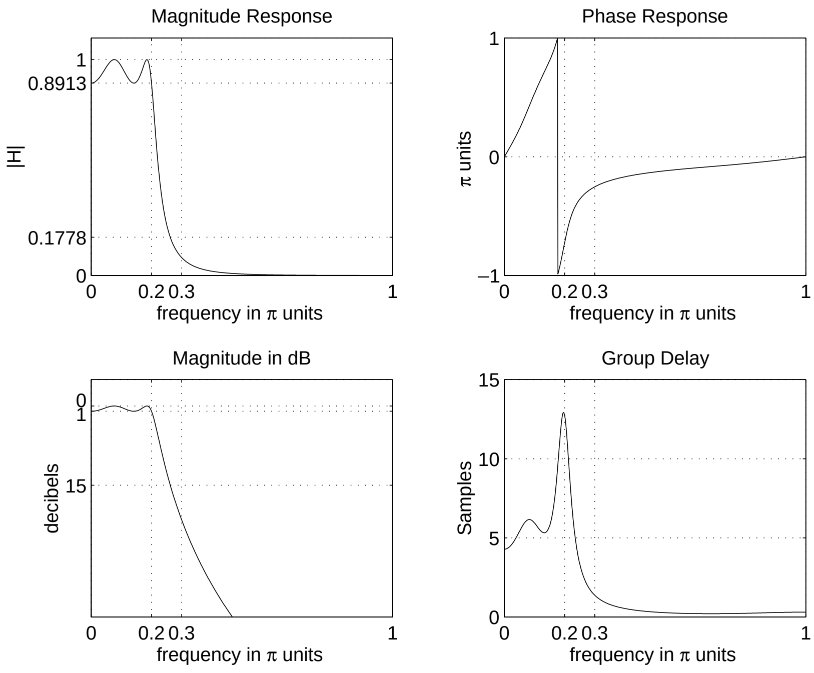


FIGURE 8.22 *Digital Chebyshev-I lowpass filter using impulse invariance design*

script follows:

```
>> % Digital Filter Specifications:
>> wp = 0.2*pi;                          % digital Passband freq in rad
>> ws = 0.3*pi;                          % digital Stopband freq in rad
>> Rp = 1;                               % Passband ripple in dB
>> As = 15;                              % Stopband attenuation in dB

>> % Analog Prototype Specifications: Inverse mapping for frequencies
>> T = 1;                                % Set T=1
>> OmegaP = wp / T;                      % Prototype Passband freq
>> OmegaS = ws / T;                      % Prototype Stopband freq

>> % Analog Chebyshev-1 Prototype Filter Calculation:
>> [cs,ds] = afd_chb2(OmegaP,OmegaS,Rp,As);
*** Chebyshev-2 Filter Order =  4

>> % Impulse Invariance transformation:
>> [b,a] = imp_invr(cs,ds,T);  [C,B,A] = dir2par(b,a);
```

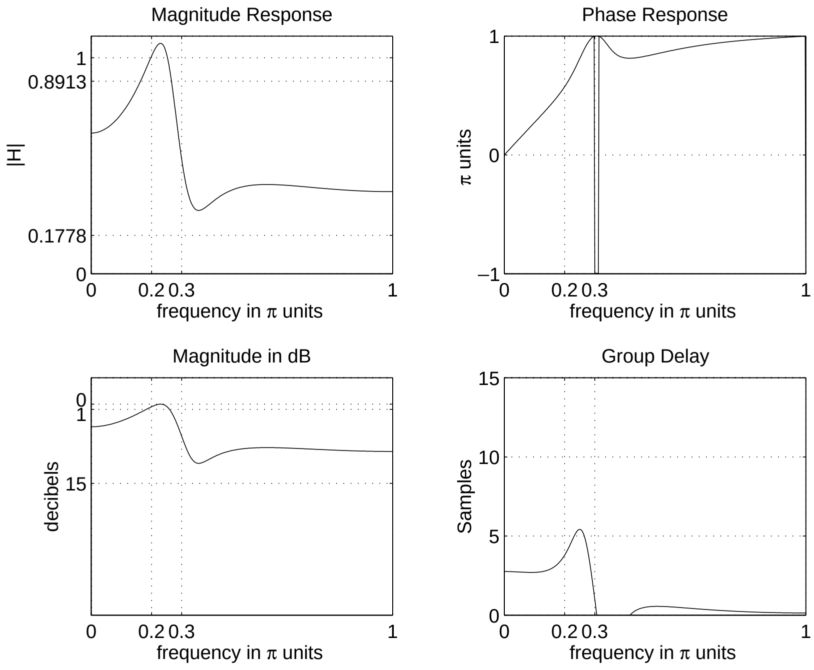


FIGURE 8.23 *Digital Chebyshev-II lowpass filter using impulse invariance design*

From the frequency response plots in Figure 8.23 we clearly observe the passband as well as stopband degradation. Hence the impulse invariance design technique has failed to produce a desired digital filter. □

□ **EXAMPLE 8.14** Design a lowpass digital filter using an elliptic prototype to satisfy

$$\omega_p = 0.2\pi,\ R_p = 1 \text{ dB}$$
$$\omega_s = 0.3\pi,\ A_s = 15 \text{ dB}$$

Solution The elliptic filter is equiripple in both bands. Hence this situation is similar to that of the Chebyshev-II filter, and we should not expect a good digital filter. The MATLAB script follows:

```
>> % Digital Filter Specifications:
>> wp = 0.2*pi;                         % digital Passband freq in rad
>> ws = 0.3*pi;                         % digital Stopband freq in rad
>> Rp = 1;                              % Passband ripple in dB
>> As = 15;                             % Stopband attenuation in dB
```

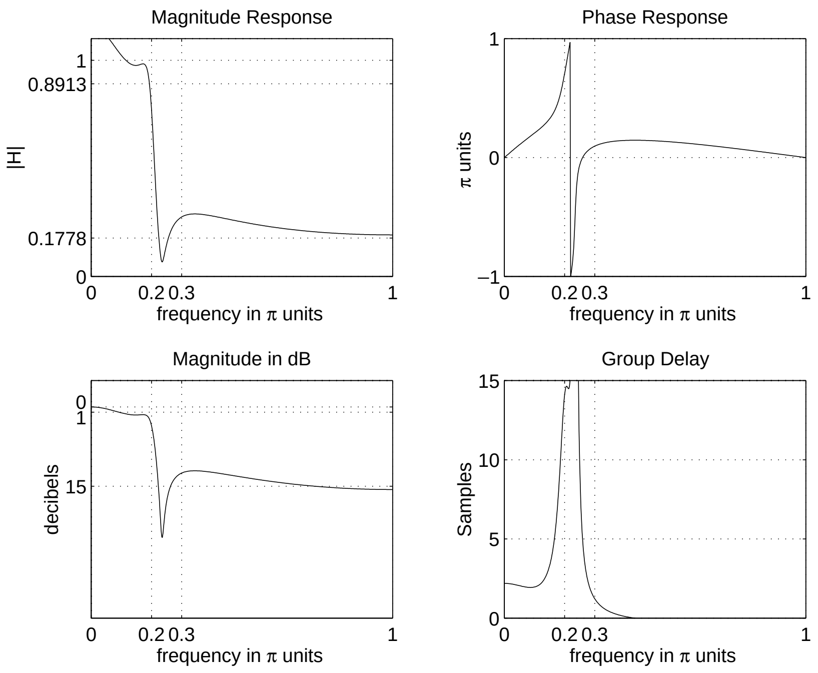


FIGURE 8.24 *Digital elliptic lowpass filter using impulse invariance design*

```
>> % Analog Prototype Specifications: Inverse mapping for frequencies
>> T = 1;                                % Set T=1
>> OmegaP = wp / T;                      % Prototype Passband freq
>> OmegaS = ws / T;                      % Prototype Stopband freq

>> % Analog Elliptic Prototype Filter Calculation:
>> [cs,ds] = afd_elip(OmegaP,OmegaS,Rp,As);
*** Elliptic Filter Order =  3

>> % Impulse Invariance transformation:
>> [b,a] = imp_invr(cs,ds,T);  [C,B,A] = dir2par(b,a);
```

From the frequency response plots in Figure 8.24 we clearly observe that once again the impulse invariance design technique has failed. □

The advantages of the impulse invariance mapping are that it is a stable design and that the frequencies Ω and ω are linearly related. But the disadvantage is that we should expect some aliasing of the analog frequency response, and in some cases this aliasing is intolerable. Consequently, this design method is useful only when the analog filter is essentially band-limited to a lowpass or bandpass filter in which there are no oscillations in the stopband.

8.4.3 BILINEAR TRANSFORMATION

This mapping is the best transformation method; it involves a well-known function given by

$$s = \frac{2}{T}\frac{1 - z^{-1}}{1 + z^{-1}} \Longrightarrow z = \frac{1 + sT/2}{1 - sT/2} \tag{8.65}$$

where T is a parameter. Another name for this transformation is the *linear fractional* transformation because when cleared of fractions, we obtain

$$\frac{T}{2}sz + \frac{T}{2}s - z + 1 = 0$$

which is linear in each variable if the other is fixed, or *bilinear* in s and z. The complex plane mapping under (8.65) is shown in Figure 8.25, from which we have the following observations:

1. Using $s = \sigma + j\Omega$ in (8.65), we obtain

$$z = \left(1 + \frac{\sigma T}{2} + j\frac{\Omega T}{2}\right) \Big/ \left(1 - \frac{\sigma T}{2} - j\frac{\Omega T}{2}\right) \tag{8.66}$$

Hence

$$\sigma < 0 \Longrightarrow |z| = \left|\frac{1 + \frac{\sigma T}{2} + j\frac{\Omega T}{2}}{1 - \frac{\sigma T}{2} - j\frac{\Omega T}{2}}\right| < 1$$

$$\sigma = 0 \Longrightarrow |z| = \left|\frac{1 + j\frac{\Omega T}{2}}{1 - j\frac{\Omega T}{2}}\right| = 1$$

$$\sigma > 0 \Longrightarrow |z| = \left|\frac{1 + \frac{\sigma T}{2} + j\frac{\Omega T}{2}}{1 - \frac{\sigma T}{2} - j\frac{\Omega T}{2}}\right| > 1$$

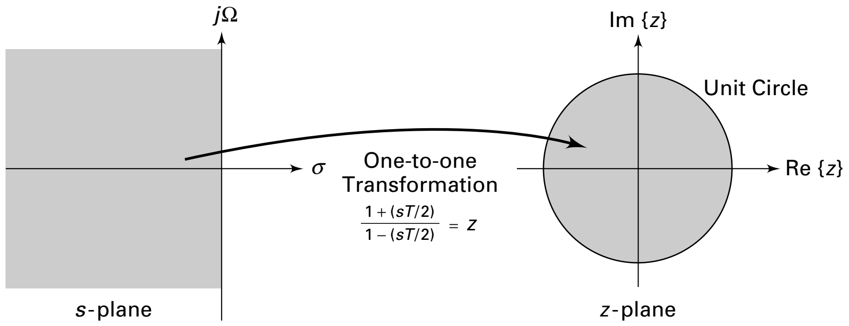


FIGURE 8.25 *Complex-plane mapping in bilinear transformation*

2. The entire left half-plane maps into the inside of the unit circle. Hence this is a stable transformation.
3. The imaginary axis maps onto the unit circle in a one-to-one fashion. Hence there is no aliasing in the frequency domain.

Substituting $\sigma = 0$ in (8.66), we obtain

$$z = \frac{1 + j\frac{\Omega T}{2}}{1 - j\frac{\Omega T}{2}} = e^{j\omega}$$

since the magnitude is 1. Solving for ω as a function of Ω, we obtain

$$\omega = 2\tan^{-1}\left(\frac{\Omega T}{2}\right) \qquad \text{or} \qquad \Omega = \frac{2}{T}\tan\left(\frac{\omega}{2}\right) \tag{8.67}$$

This shows that Ω is nonlinearly related to (or warped into) ω but that there is no aliasing. Hence in (8.67) we will say that ω is prewarped into Ω.

□ **EXAMPLE 8.15** Transform $H_a(s) = \dfrac{s+1}{s^2 + 5s + 6}$ into a digital filter using the bilinear transformation. Choose $T = 1$.

Solution Using (8.65), we obtain

$$\begin{aligned} H(z) &= H_a\left(\frac{2}{T}\,\frac{1 - z^{-1}}{1 + z^{-1}}\bigg|_{T=1}\right) = H_a\left(2\frac{1 - z^{-1}}{1 + z^{-1}}\right) \\ &= \frac{2\dfrac{1 - z^{-1}}{1 + z^{-1}} + 1}{\left(2\dfrac{1 - z^{-1}}{1 + z^{-1}}\right)^2 + 5\left(2\dfrac{1 - z^{-1}}{1 + z^{-1}}\right) + 6} \end{aligned}$$

Simplifying,

$$H(z) = \frac{3 + 2z^{-1} - z^{-2}}{20 + 4z^{-1}} = \frac{0.15 + 0.1z^{-1} - 0.05z^{-2}}{1 + 0.2z^{-1}}$$

□

MATLAB provides a function called `bilinear` to implement this mapping. Its invocation is similar to the `imp_invr` function, but it also takes several forms for different input-output quantities. The SP toolbox manual should be consulted for more details. Its use is shown in the following example.

□ **EXAMPLE 8.16** Transform the system function $H_a(s)$ in Example 8.15 using the `bilinear` function.

Solution MATLAB script:

```
>> c = [1,1]; d = [1,5,6]; T = 1; Fs = 1/T;
>> [b,a] = bilinear(c,d,Fs)
b = 0.1500    0.1000   -0.0500
a = 1.0000    0.2000    0.0000
```

The filter is

$$H(z) = \frac{0.15 + 0.1z^{-1} - 0.05z^{-2}}{1 + 0.2z^{-1}}$$

as before. □

8.4.4 DESIGN PROCEDURE

Given digital filter specifications ω_p, ω_s, R_p, and A_s, we want to determine $H(z)$. The design steps in this procedure are the following:

1. Choose a value for T. This is arbitrary, and we may set $T = 1$.
2. Prewarp the cutoff frequencies ω_p and ω_s; that is, calculate Ω_p and Ω_s using (8.67):

$$\Omega_p = \frac{2}{T}\tan\left(\frac{\omega_p}{2}\right), \quad \Omega_s = \frac{2}{T}\tan\left(\frac{\omega_s}{2}\right) \tag{8.68}$$

3. Design an analog filter $H_a(s)$ to meet the specifications Ω_p, Ω_s, R_p, and A_s. We have already described how to do this in the previous section.
4. Finally, set

$$H(z) = H_a\left(\frac{2}{T}\frac{1 - z^{-1}}{1 + z^{-1}}\right)$$

and simplify to obtain $H(z)$ as a rational function in z^{-1}.

In the next several examples we demonstrate this design procedure on our analog prototype filters.

□ **EXAMPLE 8.17** Design the digital Butterworth filter of Example 8.11. The specifications are

$$\omega_p = 0.2\pi,\ R_p = 1 \text{ dB}$$
$$\omega_s = 0.3\pi,\ A_s = 15 \text{ dB}$$

Solution MATLAB script:

```
>> % Digital Filter Specifications:
>> wp = 0.2*pi;                          % digital Passband freq in rad
>> ws = 0.3*pi;                          % digital Stopband freq in rad
>> Rp = 1;                               % Passband ripple in dB
>> As = 15;                              % Stopband attenuation in dB
>> % Analog Prototype Specifications: Inverse mapping for frequencies
>> T = 1; Fs = 1/T;                      % Set T=1
>> OmegaP = (2/T)*tan(wp/2);             % Prewarp Prototype Passband freq
>> OmegaS = (2/T)*tan(ws/2);             % Prewarp Prototype Stopband freq
>> % Analog Butterworth Prototype Filter Calculation:
>> [cs,ds] = afd_butt(OmegaP,OmegaS,Rp,As);
*** Butterworth Filter Order =  6
>> % Bilinear transformation:
>> [b,a] = bilinear(cs,ds,Fs);  [C,B,A] = dir2cas(b,a)
C = 5.7969e-004
B = 1.0000    2.0183    1.0186
    1.0000    1.9814    0.9817
    1.0000    2.0004    1.0000
A = 1.0000   -0.9459    0.2342
    1.0000   -1.0541    0.3753
    1.0000   -1.3143    0.7149
```

The desired filter is once again a 6th-order filter and has 6 zeros. Since the 6th-order zero of $H_a(s)$ at $s = -\infty$ is mapped to $z = -1$, these zeros should be at $z = -1$. Due to the finite precision of MATLAB these zeros are not exactly at $z = -1$. Hence the system function should be

$$H(z) = \frac{0.00057969\left(1+z^{-1}\right)^6}{(1-0.9459z^{-1}+0.2342z^{-2})\,(1-1.0541z^{-1}+0.3753z^{-2})\,(1-1.3143z^{-1}+0.7149z^{-2})}$$

The frequency response plots are given in Figure 8.26. Comparing these plots with those in Figure 8.21, we observe that these two designs are very similar. □

□ **EXAMPLE 8.18** Design the digital Chebyshev-I filter of Example 8.12. The specifications are

$$\omega_p = 0.2\pi,\ R_p = 1 \text{ dB}$$
$$\omega_s = 0.3\pi,\ A_s = 15 \text{ dB}$$

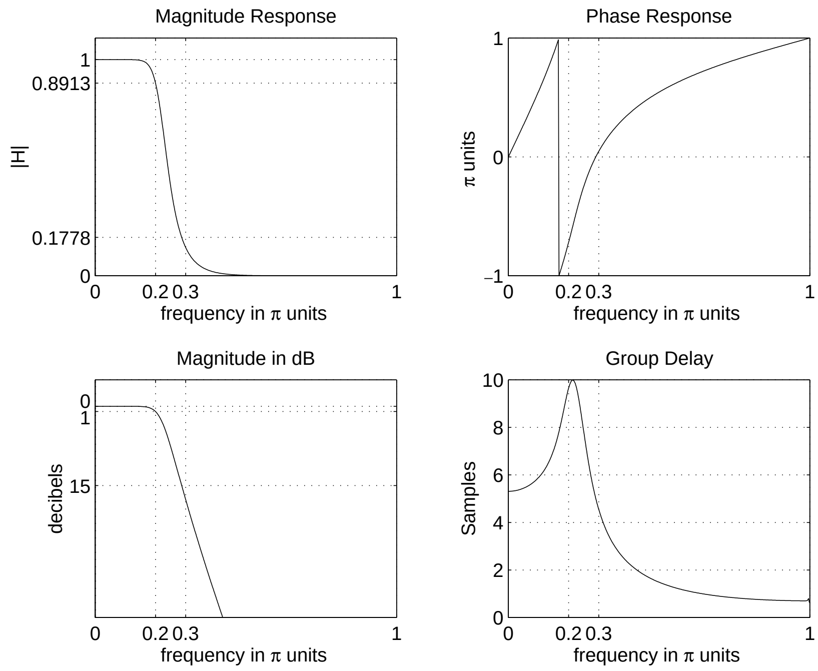


FIGURE 8.26 *Digital Butterworth lowpass filter using bilinear transformation*

Solution MATLAB script:

```
>> % Digital Filter Specifications:
>> wp = 0.2*pi;                          % digital Passband freq in rad
>> ws = 0.3*pi;                          % digital Stopband freq in rad
>> Rp = 1;                               % Passband ripple in dB
>> As = 15;                              % Stopband attenuation in dB
>> % Analog Prototype Specifications: Inverse mapping for frequencies
>> T = 1; Fs = 1/T;                      % Set T=1
>> OmegaP = (2/T)*tan(wp/2);             % Prewarp Prototype Passband freq
>> OmegaS = (2/T)*tan(ws/2);             % Prewarp Prototype Stopband freq
>> % Analog Chebyshev-1 Prototype Filter Calculation:
>> [cs,ds] = afd_chb1(OmegaP,OmegaS,Rp,As);
*** Chebyshev-1 Filter Order =  4
>> % Bilinear transformation:
>> [b,a] = bilinear(cs,ds,Fs);  [C,B,A] = dir2cas(b,a)
C = 0.0018
B = 1.0000    2.0000    1.0000
    1.0000    2.0000    1.0000
A = 1.0000   -1.4996    0.8482
    1.0000   -1.5548    0.6493
```

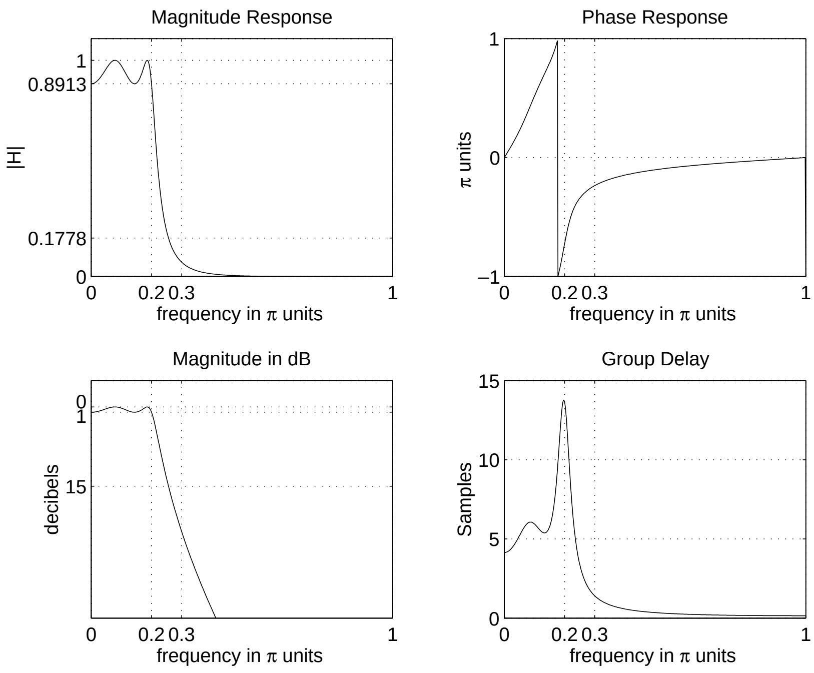


FIGURE 8.27 *Digital Chebyshev-I lowpass filter using bilinear transformation*

The desired filter is a 4th-order filter and has 4 zeros at $z = -1$. The system function is

$$H(z) = \frac{0.0018\left(1 + z^{-1}\right)^4}{\left(1 - 1.4996z^{-1} + 0.8482z^{-2}\right)\left(1 - 1.5548z^{-1} + 0.6493z^{-2}\right)}$$

The frequency response plots are given in Figure 8.27 which are similar to those in Figure 8.22. □

□ **EXAMPLE 8.19** Design the digital Chebyshev-II filter of Example 8.13. The specifications are

$$\omega_p = 0.2\pi,\ R_p = 1 \text{ dB}$$

$$\omega_s = 0.3\pi,\ A_s = 15 \text{ dB}$$

Solution MATLAB script:

```
>> % Digital Filter Specifications:
>> wp = 0.2*pi;                        % digital Passband freq in rad
>> ws = 0.3*pi;                        % digital Stopband freq in rad
>> Rp = 1;                             % Passband ripple in dB
>> As = 15;                            % Stopband attenuation in dB
>> % Analog Prototype Specifications: Inverse mapping for frequencies
>> T = 1; Fs = 1/T;                    % Set T=1
>> OmegaP = (2/T)*tan(wp/2);           % Prewarp Prototype Passband freq
>> OmegaS = (2/T)*tan(ws/2);           % Prewarp Prototype Stopband freq
>> % Analog Chebyshev-2 Prototype Filter Calculation:
>> [cs,ds] = afd_chb2(OmegaP,OmegaS,Rp,As);
*** Chebyshev-2 Filter Order =  4
>> % Bilinear transformation:
>> [b,a] = bilinear(cs,ds,Fs);  [C,B,A] = dir2cas(b,a)
C = 0.1797
B = 1.0000    0.5574    1.0000
    1.0000   -1.0671    1.0000
A = 1.0000   -0.4183    0.1503
    1.0000   -1.1325    0.7183
```

The desired filter is again a 4th-order filter with system function

$$H(z) = \frac{0.1797\left(1 + 0.5574z^{-1} + z^{-2}\right)\left(1 - 1.0671z^{-1} + z^{-2}\right)}{\left(1 - 0.4183z^{-1} + 0.1503z^{-2}\right)\left(1 - 1.1325z^{-1} + 0.7183z^{-2}\right)}$$

The frequency response plots are given in Figure 8.28. Note that the bilinear transformation has properly designed the Chebyshev-II digital filter. □

□ **EXAMPLE 8.20** Design the digital elliptic filter of Example 8.14. The specifications are

$$\omega_p = 0.2\pi,\ R_p = 1 \text{ dB}$$
$$\omega_s = 0.3\pi,\ A_s = 15 \text{ dB}$$

Solution MATLAB script:

```
>> % Digital Filter Specifications:
>> wp = 0.2*pi;                        % digital Passband freq in rad
>> ws = 0.3*pi;                        % digital Stopband freq in rad
>> Rp = 1;                             % Passband ripple in dB
>> As = 15;                            % Stopband attenuation in dB
>> % Analog Prototype Specifications: Inverse mapping for frequencies
>> T = 1; Fs = 1/T;                    % Set T=1
```

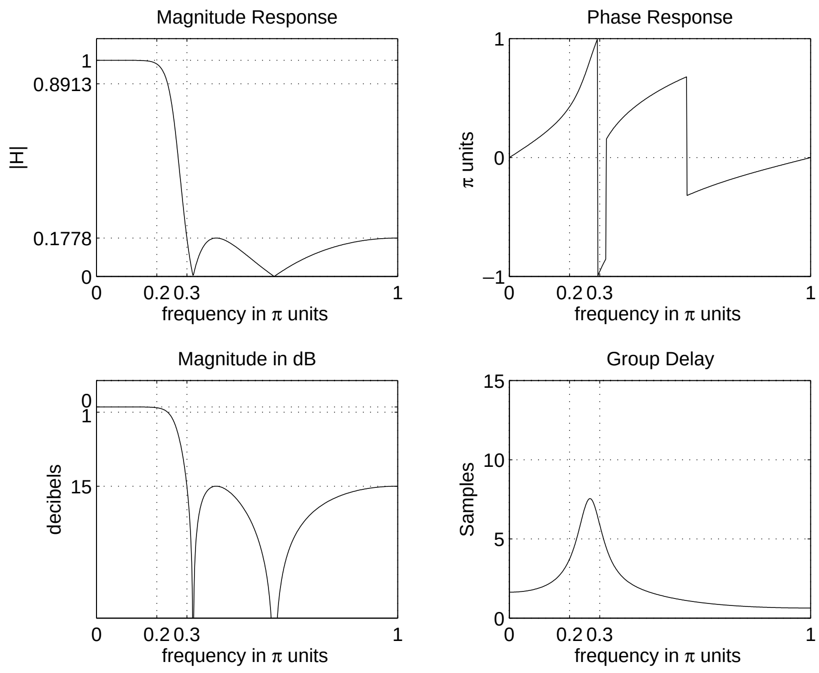


FIGURE 8.28 *Digital Chebyshev-II lowpass filter using bilinear transformation*

```
>> OmegaP = (2/T)*tan(wp/2);           % Prewarp Prototype Passband freq
>> OmegaS = (2/T)*tan(ws/2);           % Prewarp Prototype Stopband freq
>> % Analog Elliptic Prototype Filter Calculation:
>> [cs,ds] = afd_elip(OmegaP,OmegaS,Rp,As);
*** Elliptic Filter Order =  3
>> % Bilinear transformation:
>> [b,a] = bilinear(cs,ds,Fs);  [C,B,A] = dir2cas(b,a)
C = 0.1214
B = 1.0000   -1.4211    1.0000
    1.0000    1.0000         0
A = 1.0000   -1.4928    0.8612
    1.0000   -0.6183         0
```

The desired filter is a 3rd-order filter with system function

$$H(z) = \frac{0.1214\left(1 - 1.4211z^{-1} + z^{-2}\right)\left(1 + z^{-1}\right)}{\left(1 - 1.4928z^{-1} + 0.8612z^{-2}\right)\left(1 - 0.6183z^{-1}\right)}$$

The frequency response plots are given in Figure 8.29. Note that the bilinear transformation has again properly designed the elliptic digital filter. □

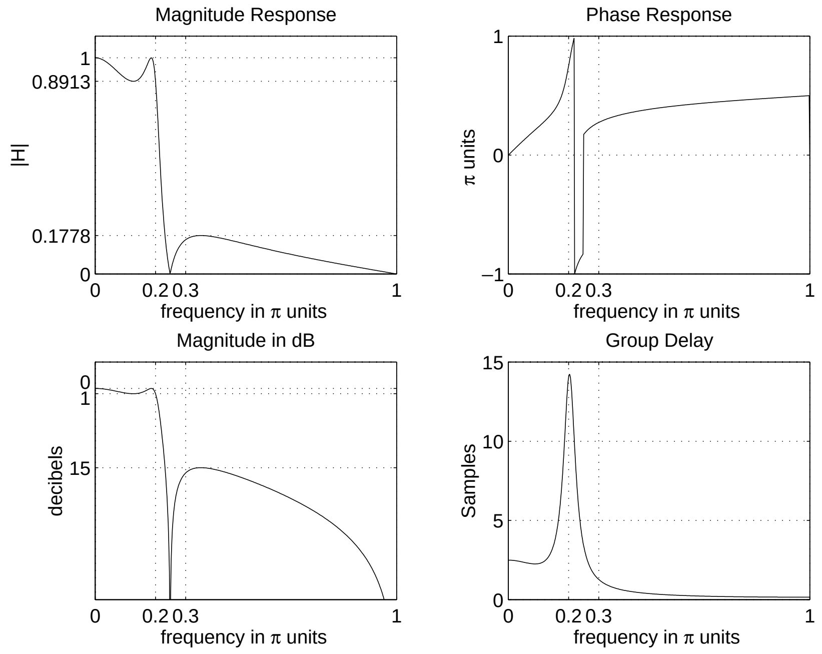


FIGURE 8.29 *Digital elliptic lowpass filter using bilinear transformation*

The advantages of this mapping are that (a) it is a stable design, (b) there is no aliasing, and (c) there is no restriction on the type of filter that can be transformed. Therefore this method is used exclusively in computer programs including MATLAB, as we shall see next.

8.4.5 MATCHED-z TRANSFORMATION

In this method of filter transformation, zeros and poles of $H_a(s)$ are directly mapped into zeros and poles in the z-plane using an exponential function. Given a root (zero or pole) at the location $s = a$ in the s-plane, we map it in the z-plane at $z = e^{aT}$ where T is a sampling interval. Thus, the system function $H_a(s)$ with zeros $\{z_k\}$ and poles $\{p_\ell\}$ is mapped into the digital filter system function $H(z)$ as

$$H_a(s) = \frac{\prod_{k=1}^{M}(s - z_k)}{\prod_{\ell=1}^{N}(s - p_\ell)} \quad \rightarrow \quad H(z) = \frac{\prod_{k=1}^{M}\left(1 - e^{z_k T} z^{-1}\right)}{\prod_{\ell=1}^{N}\left(s - e^{p_\ell T} z^{-1}\right)} \qquad \textbf{(8.69)}$$

Clearly the z-transform system function is "matched" to the s-domain system function.

Note that this technique appears to be similar to the impulse invariance mapping in that the pole locations are identical and aliasing is unavoidable. However, these two techniques differ in zero locations. Also the matched-z transformation does not preserve either the impulse response or the frequency response characteristics. Hence it is suitable when designing using pole-zero placement, but it is generally unsuitable when the frequency-domain specifications are given.

8.5 LOWPASS FILTER DESIGN USING MATLAB

In this section we will demonstrate the use of MATLAB's filter design functions to design digital lowpass filters. These functions use the bilinear transformation because of its desirable advantages as discussed in the previous section. These functions are as follows:

1. `[b,a]=butter(N,wn)`
   This function designs an `N`th-order lowpass digital Butterworth filter and returns the filter coefficients in length $N+1$ vectors `b` and `a`. The filter order is given by (8.49), and the cutoff frequency `wn` is determined by the prewarping formula (8.68). However, in MATLAB all digital frequencies are given in *units of* π. Hence `wn` is computed by using the following relation:

$$\omega_n = \frac{2}{\pi}\tan^{-1}\left(\frac{\Omega_c T}{2}\right)$$

   The use of this function is given in Example 8.21.
2. `[b,a]=cheby1(N,Rp,wn)`
   This function designs an `N`th-order lowpass digital Chebyshev-I filter with `Rp` decibels of ripple in the passband. It returns the filter coefficients in length $N+1$ vectors `b` and `a`. The filter order is given by (8.59), and the cutoff frequency `wn` is the digital passband frequency in units of π; that is,

$$\omega_n = \omega_p/\pi$$

   The use of this function is given in Example 8.22.
3. `[b,a]=cheby2(N,As,wn)`
   This function designs an `N`th-order lowpass digital Chebyshev-II filter with the stopband attenuation `As` decibels. It returns the filter coefficients in length $N+1$ vectors `b` and `a`. The filter order is given by (8.59), and the cutoff frequency `wn` is the digital stopband frequency in units of π; that is,

$$\omega_n = \omega_s/\pi$$

   The use of this function is given in Example 8.23.

4. `[b,a]=ellip(N,Rp,As,wn)`
   This function designs an `N`th-order lowpass digital elliptic filter with the passband ripple of `Rp` decibels and a stopband attenuation of `As` decibels. It returns the filter coefficients in length $N + 1$ vectors `b` and `a`. The filter order is given by (8.62), and the cutoff frequency `wn` is the digital passband frequency in units of π; that is,

$$\omega_n = \omega_p/\pi$$

   The use of this function is given in Example 8.24.

All these above functions can also be used to design other frequency-selective filters, such as highpass and bandpass. We will discuss their additional capabilities in Section 8.6.

There is also another set of filter functions, namely the `buttord`, `cheb1ord`, `cheb2ord`, and `ellipord` functions, which can provide filter order N and filter cutoff frequency ω_n, given the specifications. These functions are available in the Signal Processing toolbox. In the examples to follow we will determine these parameters using the formulas given earlier. We will discuss the filter-order functions in the next section.

In the following examples we will redesign the same lowpass filters of previous examples and compare their results. The specifications of the lowpass digital filter are

$$\omega_p = 0.2\pi, \; R_p = 1 \text{ dB}$$
$$\omega_s = 0.3\pi, \; A_s = 15 \text{ dB}$$

□ **EXAMPLE 8.21** Digital Butterworth lowpass filter design:

```
>> % Digital Filter Specifications:
>> wp = 0.2*pi;                         %digital Passband freq in rad
>> ws = 0.3*pi;                         %digital Stopband freq in rad
>> Rp = 1;                              %Passband ripple in dB
>> As = 15;                             %Stopband attenuation in dB

>> % Analog Prototype Specifications:
>> T = 1;                               %Set T=1
>> OmegaP = (2/T)*tan(wp/2);            %Prewarp Prototype Passband freq
>> OmegaS = (2/T)*tan(ws/2);            %Prewarp Prototype Stopband freq
>> % Analog Prototype Order Calculation:
>> N =ceil((log10((10^(Rp/10)-1)/(10^(As/10)-1)))/(2*log10(OmegaP/OmegaS)));
>> fprintf('\n*** Butterworth Filter Order = %2.0f \n',N)
** Butterworth Filter Order =  6
>> OmegaC = OmegaP/((10^(Rp/10)-1)^(1/(2*N))); %Analog BW prototype cutoff
>> wn = 2*atan((OmegaC*T)/2);            %Digital BW cutoff freq
>> % Digital Butterworth Filter Design:
>> wn = wn/pi;                              %Digital Butter cutoff in pi units
>> [b,a]=butter(N,wn);  [b0,B,A] = dir2cas(b,a)
C = 5.7969e-004
B = 1.0000    2.0297    1.0300
    1.0000    1.9997    1.0000
    1.0000    1.9706    0.9709
A = 1.0000   -0.9459    0.2342
    1.0000   -1.0541    0.3753
    1.0000   -1.3143    0.7149
```

The system function is

$$H(z) = \frac{0.00057969\left(1+z^{-1}\right)^6}{\left(1-0.9459z^{-1}+0.2342z^{-2}\right)\left(1-1.0541z^{-1}+0.3753z^{-2}\right)\left(1-1.3143z^{-1}+0.7149z^{-2}\right)}$$

which is the same as in Example 8.17. The frequency-domain plots were shown in Figure 8.26. □

□ **EXAMPLE 8.22** Digital Chebyshev-I lowpass filter design:

```
>> % Digital Filter Specifications:
>> wp = 0.2*pi;                            %digital Passband freq in rad
>> ws = 0.3*pi;                            %digital Stopband freq in rad
>> Rp = 1;                                 %Passband ripple in dB
>> As = 15;                                %Stopband attenuation in dB

>> % Analog Prototype Specifications:
>> T = 1;                                  %Set T=1
>> OmegaP = (2/T)*tan(wp/2);               %Prewarp Prototype Passband freq
>> OmegaS = (2/T)*tan(ws/2);               %Prewarp Prototype Stopband freq

>> % Analog Prototype Order Calculation:
>> ep = sqrt(10^(Rp/10)-1);                %Passband Ripple Factor
>> A = 10^(As/20);                         %Stopband Attenuation Factor
>> OmegaC = OmegaP;                        %Analog Prototype Cutoff freq
>> OmegaR = OmegaS/OmegaP;                 %Analog Prototype Transition Ratio
>> g = sqrt(A*A-1)/ep;                     %Analog Prototype Intermediate cal.
>> N = ceil(log10(g+sqrt(g*g-1))/log10(OmegaR+sqrt(OmegaR*OmegaR-1)));
>> fprintf('\n*** Chebyshev-1 Filter Order = %2.0f \n',N)
*** Chebyshev-1 Filter Order =  4
>> % Digital Chebyshev-I Filter Design:
>> wn = wp/pi;                             %Digital Passband freq in pi units
>> [b,a]=cheby1(N,Rp,wn);  [b0,B,A] = dir2cas(b,a)
```

```
b0 = 0.0018
B = 1.0000    2.0000    1.0000
    1.0000    2.0000    1.0000
A = 1.0000   -1.4996    0.8482
    1.0000   -1.5548    0.6493
```

The system function is

$$H(z) = \frac{0.0018\left(1+z^{-1}\right)^4}{(1-1.4996z^{-1}+0.8482z^{-2})\,(1-1.5548z^{-1}+0.6493z^{-2})}$$

which is the same as in Example 8.18. The frequency-domain plots were shown in Figure 8.27. □

□ **EXAMPLE 8.23** Digital Chebyshev-II lowpass filter design:

```
>> % Digital Filter Specifications:
>> wp = 0.2*pi;                          %digital Passband freq in rad
>> ws = 0.3*pi;                          %digital Stopband freq in rad
>> Rp = 1;                               %Passband ripple in dB
>> As = 15;                              %Stopband attenuation in dB

>> % Analog Prototype Specifications:
>> T = 1;                                %Set T=1
>> OmegaP = (2/T)*tan(wp/2);             %Prewarp Prototype Passband freq
>> OmegaS = (2/T)*tan(ws/2);             %Prewarp Prototype Stopband freq

>> % Analog Prototype Order Calculation:
>> ep = sqrt(10^(Rp/10)-1);              %Passband Ripple Factor
>> A = 10^(As/20);                       %Stopband Attenuation Factor
>> OmegaC = OmegaP;                      %Analog Prototype Cutoff freq
>> OmegaR = OmegaS/OmegaP;               %Analog Prototype Transition Ratio
>> g = sqrt(A*A-1)/ep;                   %Analog Prototype Intermediate cal.
>> N = ceil(log10(g+sqrt(g*g-1))/log10(OmegaR+sqrt(OmegaR*OmegaR-1)));
>> fprintf('\n*** Chebyshev-2 Filter Order = %2.0f \n',N)
*** Chebyshev-2 Filter Order =  4

>> % Digital Chebyshev-II Filter Design:
>> wn = ws/pi;                           %Digital Stopband freq in pi units
>> [b,a]=cheby2(N,As,wn);  [b0,B,A] = dir2cas(b,a)
b0 = 0.1797
B = 1.0000    0.5574    1.0000
    1.0000   -1.0671    1.0000
A = 1.0000   -0.4183    0.1503
    1.0000   -1.1325    0.7183
```

The system function is

$$H(z) = \frac{0.1797\left(1 + 0.5574z^{-1} + z^{-2}\right)\left(1 - 1.0671z^{-1} + z^{-2}\right)}{\left(1 - 0.4183z^{-1} + 0.1503z^{-2}\right)\left(1 - 1.1325z^{-1} + 0.7183z^{-2}\right)}$$

which is the same as in Example 8.19. The frequency-domain plots were shown in Figure 8.28. □

□ **EXAMPLE 8.24** Digital elliptic lowpass filter design:

```
>> % Digital Filter Specifications:
>> wp = 0.2*pi;                           %digital Passband freq in rad
>> ws = 0.3*pi;                           %digital Stopband freq in rad
>> Rp = 1;                                %Passband ripple in dB
>> As = 15;                               %Stopband attenuation in dB

>> % Analog Prototype Specifications:
>> T = 1;                                 %Set T=1
>> OmegaP = (2/T)*tan(wp/2);              %Prewarp Prototype Passband freq
>> OmegaS = (2/T)*tan(ws/2);              %Prewarp Prototype Stopband freq

>> % Analog Elliptic Filter order calculations:
>> ep = sqrt(10^(Rp/10)-1);               %Passband Ripple Factor
>> A = 10^(As/20);                        %Stopband Attenuation Factor
>> OmegaC = OmegaP;                       %Analog Prototype Cutoff freq
>> k = OmegaP/OmegaS;                     %Analog Prototype Transition Ratio;
>> k1 = ep/sqrt(A*A-1);                   %Analog Prototype Intermediate cal.
>> capk = ellipke([k.^2 1-k.^2]);
>> capk1 = ellipke([(k1 .^2) 1-(k1 .^2)]);
>> N = ceil(capk(1)*capk1(2)/(capk(2)*capk1(1)));
>> fprintf('\n*** Elliptic Filter Order = %2.0f \n',N)
*** Elliptic Filter Order =  3

>> % Digital Elliptic Filter Design:
>> wn = wp/pi;                            %Digital Passband freq in pi units
>> [b,a]=ellip(N,Rp,As,wn);  [b0,B,A] = dir2cas(b,a)
b0 = 0.1214
B = 1.0000   -1.4211    1.0000
    1.0000    1.0000         0
A = 1.0000   -1.4928    0.8612
    1.0000   -0.6183         0
```

The system function is

$$H(z) = \frac{0.1214\left(1 - 1.4211z^{-1} + z^{-2}\right)\left(1 + z^{-1}\right)}{\left(1 - 1.4928z^{-1} + 0.8612z^{-2}\right)\left(1 - 0.6183z^{-1}\right)}$$

which is the same as in Example 8.20. The frequency-domain plots were shown in Figure 8.29. □

TABLE 8.1 *Comparison of three filters*

Prototype	*Order N*	*Stopband Att.*
Butterworth	6	15
Chebyshev-I	4	25
Elliptic	3	27

8.5.1 COMPARISON OF THREE FILTERS

In our examples we designed the same digital filter using four different prototype analog filters. Let us compare their performance. The specifications were $\omega_p = 0.2\pi$, $R_p = 1$ dB, $\omega_s = 0.3\pi$, and $A_s = 15$ dB. This comparison in terms of order N and the minimum stopband attenuations is shown in Table 8.1.

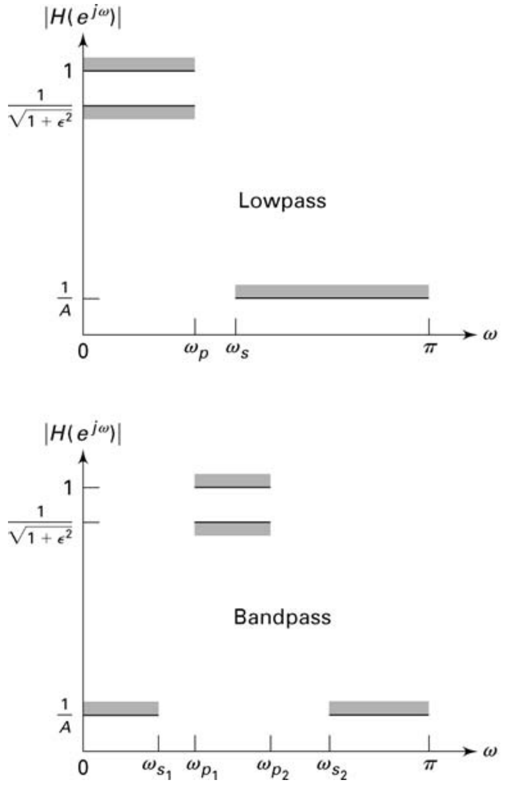
Clearly, the elliptic prototype gives the best design. However, if we compare their phase responses, then the elliptic design has the most nonlinear phase response in the passband.

8.6 FREQUENCY-BAND TRANSFORMATIONS

In the preceding two sections we designed digital lowpass filters from their corresponding analog filters. Certainly, we would like to design other types of frequency-selective filters, such as highpass, bandpass, and bandstop. This is accomplished by transforming the frequency axis (or band) of a lowpass filter so that it behaves as another frequency-selective filter. These transformations on the complex variable z are very similar to bilinear transformations, and the design equations are algebraic. The procedure to design a general frequency-selective filter is to first design a *digital prototype* (of fixed bandwidth, say unit bandwidth) lowpass filter and then to apply these algebraic transformations. In this section we will describe the basic philosophy behind these mappings and illustrate their mechanism through examples. MATLAB provides functions that incorporate frequency-band transformation in the s-plane. We will first demonstrate the use of the z-plane mapping and then illustrate the use of MATLAB functions. Typical specifications for most commonly used types of frequency-selective digital filters are shown in Figure 8.30.

Let $H_{LP}(Z)$ be the given prototype lowpass digital filter, and let $H(z)$ be the desired frequency-selective digital filter. Note that we are using two different frequency variables, Z and z, with H_{LP} and H, respectively. Define a mapping of the form

$$Z^{-1} = G(z^{-1})$$



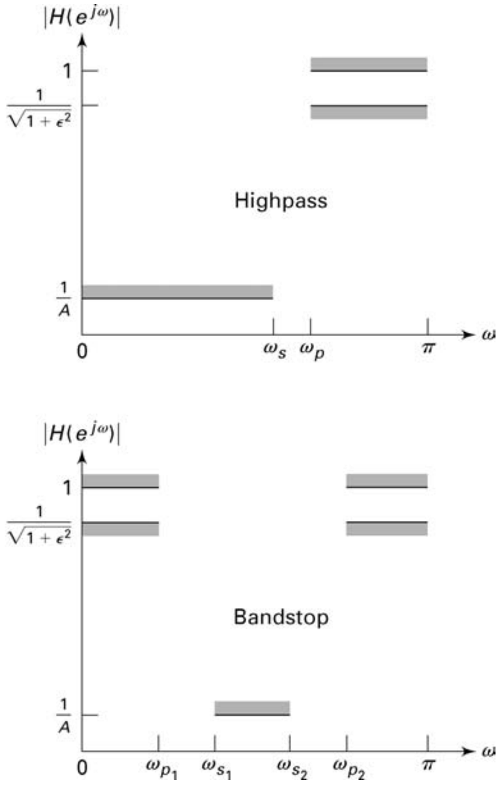


FIGURE 8.30 *Specifications of frequency-selective filters*

such that

$$H(z) = H_{LP}(Z)|_{Z^{-1}=G(z^{-1})}$$

To do this, we simply replace Z^{-1} everywhere in H_{LP} by the function $G(z^{-1})$. Given that $H_{LP}(Z)$ is a stable and causal filter, we also want $H(z)$ to be stable and causal. This imposes the following requirements:

1. $G(\cdot)$ must be a rational function in z^{-1} so that $H(z)$ is implementable.
2. The unit circle of the Z-plane must map onto the unit circle of the z-plane.
3. For stable filters, the inside of the unit circle of the Z-plane must also map onto the inside of the unit circle of the z-plane.

Let ω' and ω be the frequency variables of Z and z, respectively—that is, $Z = e^{j\omega'}$ and $z = e^{j\omega}$ on their respective unit circles. Then requirement 2 above implies that

$$|Z^{-1}| = |G(z^{-1})| = |G(e^{-j\omega})| = 1$$

and

$$e^{-j\omega'} = \left|G(e^{-j\omega})\right| e^{j\angle G(e^{-j\omega})}$$

or

$$-\omega' = \angle G(e^{-j\omega})$$

The general form of the function $G(\cdot)$ that satisfies these requirements is a rational function of the *all-pass* type given by

$$Z^{-1} = G\left(z^{-1}\right) = \pm \prod_{k=1}^{n} \frac{z^{-1} - \alpha_k}{1 - \alpha_k z^{-1}}$$

where $|\alpha_k| < 1$ for stability and to satisfy requirement 3.

Now by choosing an appropriate order n and the coefficients $\{\alpha_k\}$, we can obtain a variety of mappings. The most widely used transformations are given in Table 8.2. We will now illustrate the use of this table for designing a highpass digital filter.

□ **EXAMPLE 8.25** In Example 8.22 we designed a Chebyshev-I lowpass filter with specifications

$$\omega'_p = 0.2\pi, \quad R_p = 1 \text{ dB}$$
$$\omega'_s = 0.3\pi, \quad A_s = 15 \text{ dB}$$

and determined its system function

$$H_{LP}(Z) = \frac{0.001836(1 + Z^{-1})^4}{(1 - 1.4996Z^{-1} + 0.8482Z^{-2})(1 - 1.5548Z^{-1} + 0.6493Z^{-2})}$$

Design a highpass filter with these tolerances but with passband beginning at $\omega_p = 0.6\pi$.

Solution We want to transform the given lowpass filter into a highpass filter such that the cutoff frequency $\omega'_p = 0.2\pi$ is mapped onto the cutoff frequency $\omega_p = 0.6\pi$. From Table 8.2

$$\alpha = -\frac{\cos[(0.2\pi + 0.6\pi)/2]}{\cos[(0.2\pi - 0.6\pi)/2]} = -0.38197 \tag{8.70}$$

Hence

$$\begin{aligned} H_{LP}(z) &= H(Z)\big|_{Z=-\frac{z^{-1}-0.38197}{1-0.38197z^{-1}}} \\ &= \frac{0.02426(1 - z^{-1})^4}{(1 + 0.5661z^{-1} + 0.7657z^{-2})(1 + 1.0416z^{-1} + 0.4019z^{-2})} \end{aligned}$$

which is the desired filter. The frequency response plots of the lowpass filter and the new highpass filter are shown in Figure 8.31. □

TABLE 8.2 *Frequency transformation for digital filters (prototype lowpass filter has cutoff frequency ω_c')*

Type of Transformation	*Transformation*	*Parameters*
Lowpass	$z^{-1} \longrightarrow \dfrac{z^{-1}-\alpha}{1-\alpha z^{-1}}$	$\omega_c =$ cutoff frequency of new filter $\alpha = \dfrac{\sin\left[\left(\omega_c'-\omega_c\right)/2\right]}{\sin\left[\left(\omega_c'+\omega_c\right)/2\right]}$
Highpass	$z^{-1} \longrightarrow -\dfrac{z^{-1}+\alpha}{1+\alpha z^{-1}}$	$\omega_c =$ cutoff frequency of new filter $\alpha = -\dfrac{\cos\left[\left(\omega_c'+\omega_c\right)/2\right]}{\cos\left[\left(\omega_c'-\omega_c\right)/2\right]}$
Bandpass	$z^{-1} \longrightarrow -\dfrac{z^{-2}-\alpha_1 z^{-1}+\alpha_2}{\alpha_2 z^{-2}-\alpha_1 z^{-1}+1}$	$\omega_\ell =$ lower cutoff frequency $\omega_u =$ upper cutoff frequency $\alpha_1 = -2\beta K/(K+1)$ $\alpha_2 = (K-1)/(K+1)$ $\beta = \dfrac{\cos\left[\left(\omega_u+\omega_\ell\right)/2\right]}{\cos\left[\left(\omega_u-\omega_\ell\right)/2\right]}$ $K = \cot\dfrac{\omega_u-\omega_\ell}{2}\tan\dfrac{\omega_c'}{2}$
Bandstop	$z^{-1} \longrightarrow \dfrac{z^{-2}-\alpha_1 z^{-1}+\alpha_2}{\alpha_2 z^{-2}-\alpha_1 z^{-1}+1}$	$\omega_\ell =$ lower cutoff frequency $\omega_u =$ upper cutoff frequency $\alpha_1 = -2\beta/(K+1)$ $\alpha_2 = (K-1)/(K+1)$ $\beta = \dfrac{\cos\left[\left(\omega_u+\omega_\ell\right)/2\right]}{\cos\left[\left(\omega_u-\omega_\ell\right)/2\right]}$ $K = \tan\dfrac{\omega_u-\omega_\ell}{2}\tan\dfrac{\omega_c'}{2}$

From this example it is obvious that to obtain the rational function of a new digital filter from the prototype lowpass digital filter, we should be able to implement rational function substitutions from Table 8.2. This appears to be a difficult task, but since these are algebraic functions, we can use the `conv` function repetitively for this purpose. The following `zmapping` function illustrates this approach.

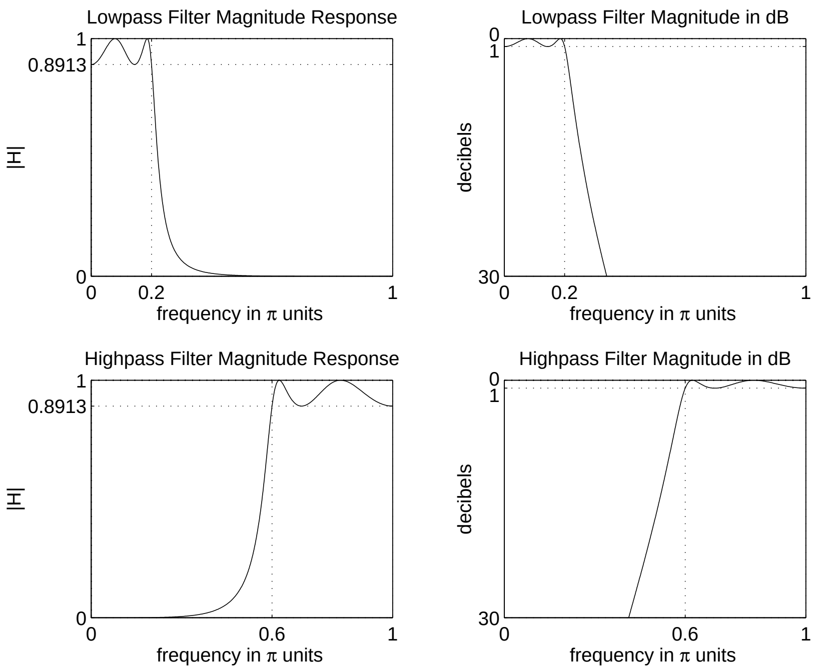


FIGURE 8.31 *Magnitude response plots for Example 8.25*

```
function [bz,az] = zmapping(bZ,aZ,Nz,Dz)
% Frequency band Transformation from Z-domain to z-domain
% -------------------------------------------------------
% [bz,az] = zmapping(bZ,aZ,Nz,Dz)
% performs:
%          b(z)   b(Z)|
%          ---- = ----|     N(z)
%          a(z)   a(Z)|@Z = ----
%                           D(z)
%

bNzord = (length(bZ)-1)*(length(Nz)-1);
aDzord = (length(aZ)-1)*(length(Dz)-1);
bzord = length(bZ)-1; azord = length(aZ)-1;
bz = zeros(1,bNzord+1);
for k = 0:bzord
    pln = [1];
    for l = 0:k-1
        pln = conv(pln,Nz);
    end
    pld = [1];
    for l = 0:bzord-k-1
        pld = conv(pld,Dz);
    end
    bz = bz+bZ(k+1)*conv(pln,pld);
end
az = zeros(1,aDzord+1);
for k = 0:azord
    pln = [1];
    for l = 0:k-1
        pln = conv(pln,Nz);
    end
    pld = [1];
    for l = 0:azord-k-1
        pld = conv(pld,Dz);
    end
    az = az+aZ(k+1)*conv(pln,pld);
end
```

☐ **EXAMPLE 8.26** Use the `zmapping` function to perform the lowpass-to-highpass transformation in Example 8.25.

Solution First we will design the lowpass digital filter in MATLAB using the bilinear transformation procedure and then use the `zmapping` function.

MATLAB script:

```
>> % Digital Lowpass Filter Specifications:
>> wplp = 0.2*pi;                         % digital Passband freq in rad
>> wslp = 0.3*pi;                         % digital Stopband freq in rad
>>   Rp = 1;                              % Passband ripple in dB
>>   As = 15;                             % Stopband attenuation in dB

>> % Analog Prototype Specifications: Inverse mapping for frequencies
>> T = 1; Fs = 1/T;                       % Set T=1
>> OmegaP = (2/T)*tan(wplp/2);            % Prewarp Prototype Passband freq
>> OmegaS = (2/T)*tan(wslp/2);            % Prewarp Prototype Stopband freq

>> % Analog Chebyshev Prototype Filter Calculation:
>> [cs,ds] = afd_chb1(OmegaP,OmegaS,Rp,As);
** Chebyshev-1 Filter Order =  4

>> % Bilinear transformation:
>> [blp,alp] = bilinear(cs,ds,Fs);
>> % Digital Highpass Filter Cutoff frequency:
>> wphp = 0.6*pi;                               % Passband edge frequency

>> % LP-to-HP frequency-band transformation:
>> alpha = -(cos((wplp+wphp)/2))/(cos((wplp-wphp)/2))
alpha = -0.3820

>> Nz = -[alpha,1]; Dz = [1,alpha];
>> [bhp,ahp] = zmapping(blp,alp,Nz,Dz);  [C,B,A] = dir2cas(bhp,ahp)
C = 0.0243
B = 1.0000   -2.0000    1.0000
    1.0000   -2.0000    1.0000
A = 1.0000    1.0416    0.4019
    1.0000    0.5561    0.7647
```

The system function of the highpass filter is

$$H(z) = \frac{0.0243(1 - z^{-1})^4}{(1 + 0.5661z^{-1} + 0.7647z^{-2})(1 + 1.0416z^{-1} + 0.4019z^{-2})}$$

which is essentially identical to that in Example 8.25. □

8.6.1 DESIGN PROCEDURE

In Example 8.26 a lowpass prototype digital filter was available to transform into a highpass filter so that a particular band-edge frequency was properly mapped. In practice we have to first design a prototype lowpass digital filter whose specifications should be obtained from specifications of other frequency-selective filters as given in Figure 8.30. We will now show that the lowpass prototype filter specifications can be obtained from the transformation formulas given in Table 8.2.

Let us use the highpass filter of Example 8.25 as an example. The passband-edge frequencies were transformed using the parameter $\alpha = -0.38197$ in (8.70). What is the stopband-edge frequency of the highpass filter, say ω_s, corresponding to the stopband edge $\omega'_s = 0.3\pi$ of the prototype lowpass filter? This can be answered by (8.70). Since α is fixed for the transformation, we set the equation

$$\alpha = -\frac{\cos[(0.3\pi + \omega_s)/2]}{\cos[(0.3\pi - \omega_s)/2]} = -0.38197$$

This is a transcendental equation whose solution can be obtained iteratively from an initial guess. It can be done using MATLAB, and the solution is

$$\omega_s = 0.4586\pi$$

Now in practice we will know the desired highpass frequencies ω_s and ω_p, and we are required to find the prototype lowpass cutoff frequencies ω'_s and ω'_p. We can choose the passband frequency ω'_p with a reasonable value, say $\omega'_p = 0.2\pi$, and determine α from ω_p using the formula from Table 8.2. Now ω'_s can be determined (for our highpass filter example) from α and

$$Z = -\frac{z^{-1} + \alpha}{1 + \alpha z^{-1}}$$

where $Z = e^{j\omega'_s}$ and $z = e^{j\omega_s}$, or

$$\omega'_s = \angle\left(-\frac{e^{-j\omega_s} + \alpha}{1 + \alpha e^{-j\omega_s}}\right) \tag{8.71}$$

Continuing our highpass filter example, let $\omega_p = 0.6\pi$ and $\omega_s = 0.4586\pi$ be the band-edge frequencies. Let us choose $\omega'_p = 0.2\pi$. Then $\alpha = -0.38197$ from (8.70), and from (8.71)

$$\omega'_s = \angle\left(-\frac{e^{-j0.4586\pi} - 0.38197}{1 - 0.38197e^{-j-0.38197}}\right) = 0.3\pi$$

as expected. Now we can design a digital lowpass filter and transform it into a highpass filter using the zmapping function to complete our design procedure. For designing a highpass Chebyshev-I digital filter, the above procedure can be incorporated into a MATLAB function called the cheb1hpf function shown here.

```
function [b,a] = cheb1hpf(wp,ws,Rp,As)
% IIR Highpass filter design using Chebyshev-1 prototype
% function [b,a] = cheb1hpf(wp,ws,Rp,As)
%   b = Numerator polynomial of the highpass filter
%   a = Denominator polynomial of the highpass filter
%  wp = Passband frequency in radians
%  ws = Stopband frequency in radians
%  Rp = Passband ripple in dB
%  As = Stopband attenuation in dB
%
% Determine the digital lowpass cutoff frequencies:
wplp = 0.2*pi;
alpha = -(cos((wplp+wp)/2))/(cos((wplp-wp)/2));
wslp = angle(-(exp(-j*ws)+alpha)/(1+alpha*exp(-j*ws)));
%
```

```
% Compute Analog lowpass Prototype Specifications:
T = 1; Fs = 1/T;
OmegaP = (2/T)*tan(wplp/2);
OmegaS = (2/T)*tan(wslp/2);

% Design Analog Chebyshev Prototype Lowpass Filter:
[cs,ds] = afd_chb1(OmegaP,OmegaS,Rp,As);

% Perform Bilinear transformation to obtain digital lowpass
[blp,alp] = bilinear(cs,ds,Fs);

% Transform digital lowpass into highpass filter
Nz = -[alpha,1]; Dz = [1,alpha];
[b,a] = zmapping(blp,alp,Nz,Dz);
```

We will demonstrate this procedure in the following example.

□ **EXAMPLE 8.27** Design a highpass digital filter to satisfy

$$\omega_p = 0.6\pi, \qquad R_p = 1 \text{ dB}$$
$$\omega_s = 0.4586\pi, \quad A_s = 15 \text{ dB}$$

Use the Chebyshev-I prototype.

Solution MATLAB script:

```
>> % Digital Highpass Filter Specifications:
>> wp = 0.6*pi;                                  % digital Passband freq in rad
>> ws = 0.4586*pi;                               % digital Stopband freq in rad
>> Rp = 1;                                       % Passband ripple in dB
>> As = 15;                                      % Stopband attenuation in dB

>> [b,a] = cheb1hpf(wp,ws,Rp,As);  [C,B,A] = dir2cas(b,a)
C = 0.0243
B = 1.0000   -2.0000    1.0000
    1.0000   -2.0000    1.0000
A = 1.0000    1.0416    0.4019
    1.0000    0.5561    0.7647
```

The system function is

$$H(z) = \frac{0.0243(1 - z^{-1})^4}{(1 + 0.5661z^{-1} + 0.7647z^{-2})(1 + 1.0416z^{-1} + 0.4019z^{-2})}$$

which is identical to that in Example 8.26. □

This highpass filter design procedure can be easily extended to other frequency-selective filters using the transformation functions in Table 8.2. These design procedures are explored in Problems P8.34, P8.36, P8.38, and P8.40. We now describe MATLAB's filter design functions for designing arbitrary frequency-selective filters.

8.6.2 MATLAB IMPLEMENTATION

In the preceding section we discussed four MATLAB functions to design digital lowpass filters. These same functions can also be used to design highpass, bandpass, and bandstop filters. The frequency-band transformations in these functions are done in the s-plane, that is, they use Approach-1 discussed on page 386. For the purpose of illustration we will use the function `butter`. It can be used with the following variations in its input arguments.

- `[b,a] = BUTTER(N,wn,'high')` designs an `N`th-order *highpass* filter with digital 3-dB cutoff frequency `wn` in units of π.
- `[b,a] = BUTTER(N,wn,)`designs an order 2`N` *bandpass* filter if `wn` is a two-element vector, `wn=[w1 w2]`, with 3-dB passband `w1 < w < w2` in units of π.
- `[b,a] = BUTTER(N,wn,'stop')` is an order 2`N` *bandstop* filter if `wn=[w1 w2]`with 3-dB stopband `w1 < w < w2` in units of π.

To design any frequency-selective Butterworth filter, we need to know the order `N` and the 3-dB cutoff frequency vector `wn`. In this chapter we described how to determine these parameters for lowpass filters. However, these calculations are more complicated for bandpass and bandstop filters. In their SP toolbox, MATLAB provides a function called `buttord` to compute these parameters. Given the specifications, ω_p, ω_s, R_p, and A_s, this function determines the necessary parameters. Its syntax is

```
[N,wn] = buttord(wp,ws,Rp,As)
```

The parameters `wp` and `ws` have some restrictions, depending on the type of filter:

- For lowpass filters `wp < ws`.
- For highpass filters `wp > ws`.
- For bandpass filters `wp` and `ws` are two-element vectors, `wp=[wp1, wp2]` and `ws=[ws1,ws2]`, such that `ws1 < wp1 < wp2 < ws2`.
- For bandstop filters `wp1 < ws1 < ws2 < wp2`.

Now using the `buttord` function in conjunction with the `butter` function, we can design any Butterworth IIR filter. Similar discussions apply for `cheby1`, `cheby2`, and `ellip` functions with appropriate modifications. We illustrate the use of these functions through the following examples.

□ **EXAMPLE 8.28** In this example we will design a Chebyshev-I highpass filter whose specifications were given in Example 8.27.

Solution MATLAB script:

```
>> % Digital Filter Specifications:        % Type: Chebyshev-I highpass
>> ws = 0.4586*pi;                         % Dig. stopband edge frequency
>> wp = 0.6*pi;                            % Dig. passband edge frequency
>> Rp = 1;                                 % Passband ripple in dB
>> As = 15;                                % Stopband attenuation in dB

>> % Calculations of Chebyshev-I Filter Parameters:
>> [N,wn] = cheb1ord(wp/pi,ws/pi,Rp,As);

>> % Digital Chebyshev-I Highpass Filter Design:
>> [b,a] = cheby1(N,Rp,wn,'high');

>> % Cascade Form Realization:
>> [b0,B,A] = dir2cas(b,a)
b0 = 0.0243
B = 1.0000   -1.9991    0.9991
    1.0000   -2.0009    1.0009
A = 1.0000    1.0416    0.4019
    1.0000    0.5561    0.7647
```

The cascade form system function

$$H(z) = \frac{0.0243(1 - z^{-1})^4}{(1 + 0.5661z^{-1} + 0.7647z^{-2})(1 + 1.0416z^{-1} + 0.4019z^{-2})}$$

is identical to the filter designed in Example 8.27, which demonstrates that the two approaches described on page 386 are identical. The frequency-domain plots are shown in Figure 8.32. □

□ **EXAMPLE 8.29** In this example we will design an elliptic bandpass filter whose specifications are given in the following MATLAB script:

```
>> % Digital Filter Specifications:        % Type: Elliptic Bandpass
>> ws = [0.3*pi 0.75*pi];                  % Dig. stopband edge frequency
>> wp = [0.4*pi 0.6*pi];                   % Dig. passband edge frequency
>> Rp = 1;                                 % Passband ripple in dB
>> As = 40;                                % Stopband attenuation in dB
```

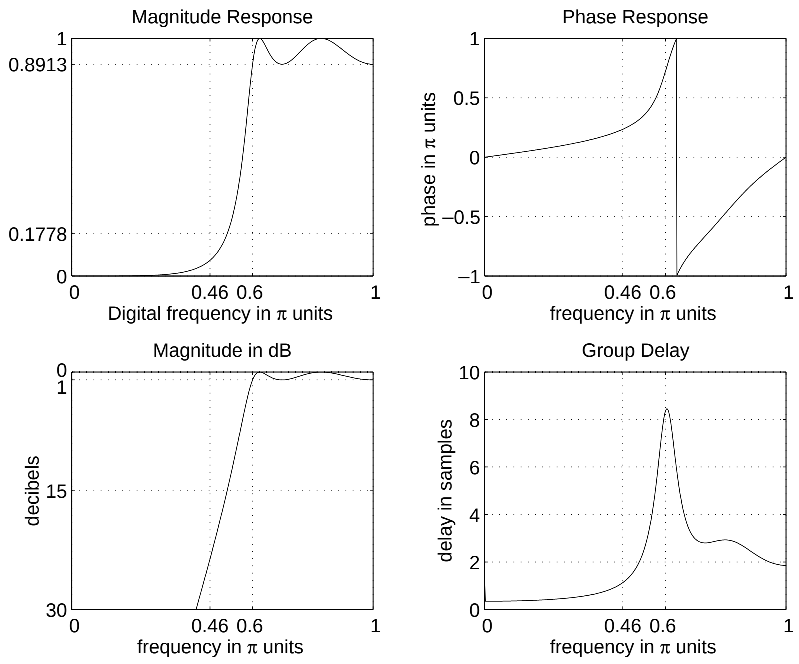


FIGURE 8.32 *Digital Chebyshev-I highpass filter in Example 8.28*

```
>> % Calculations of Elliptic Filter Parameters:
>> [N,wn] = ellipord(wp/pi,ws/pi,Rp,As);

>> % Digital Elliptic Bandpass Filter Design:
>> [b,a] = ellip(N,Rp,As,wn);

>> % Cascade Form Realization:
>> [b0,B,A] = dir2cas(b,a)
b0 = 0.0197
B = 1.0000    1.5066    1.0000
    1.0000    0.9268    1.0000
    1.0000   -0.9268    1.0000
    1.0000   -1.5066    1.0000
A = 1.0000    0.5963    0.9399
    1.0000    0.2774    0.7929
    1.0000   -0.2774    0.7929
    1.0000   -0.5963    0.9399
```

Note that the designed filter is a 10th-order filter. The frequency-domain plots are shown in Figure 8.33. □

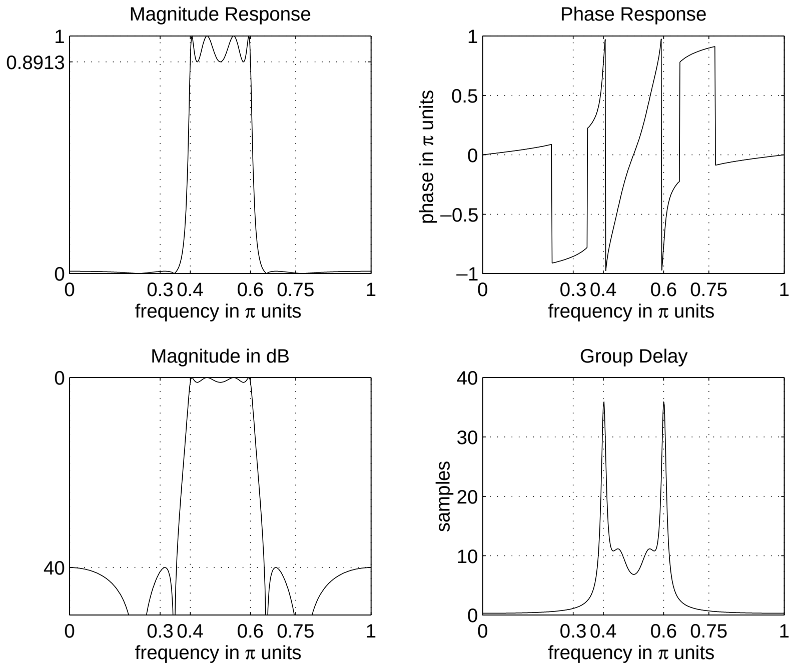


FIGURE 8.33 *Digital elliptic bandpass filter in Example 8.29*

□ **EXAMPLE 8.30** Finally, we will design a Chebyshev-II bandstop filter whose specifications are given in the following MATLAB script.

```
>> % Digital Filter Specifications:       % Type: Chebyshev-II Bandstop
>> ws = [0.4*pi 0.7*pi];                  % Dig. stopband edge frequency
>> wp = [0.25*pi 0.8*pi];                 % Dig. passband edge frequency
>> Rp = 1;                                % Passband ripple in dB
>> As = 40;                               % Stopband attenuation in dB

>> % Calculations of Chebyshev-II Filter Parameters:
>> [N,wn] = cheb2ord(wp/pi,ws/pi,Rp,As);

>> % Digital Chebyshev-II Bandstop Filter Design:
>> [b,a] = cheby2(N,As,ws/pi,'stop');

>> % Cascade Form Realization:
>> [b0,B,A] = dir2cas(b,a)
b0 = 0.1558
B = 1.0000    1.1456    1.0000
    1.0000    0.8879    1.0000
    1.0000    0.3511    1.0000
    1.0000   -0.2434    1.0000
    1.0000   -0.5768    1.0000
```

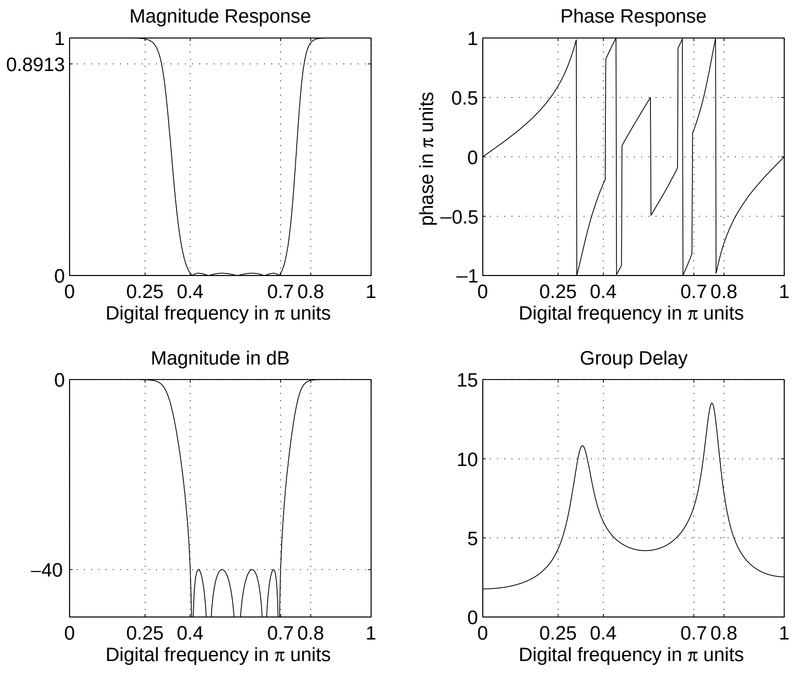


FIGURE 8.34 *Digital Chebyshev-II bandstop filter in Example 8.30*

```
A = 1.0000    1.3041    0.8031
    1.0000    0.8901    0.4614
    1.0000    0.2132    0.2145
    1.0000   -0.4713    0.3916
    1.0000   -0.8936    0.7602
```

This is also a 10th-order filter. The frequency domain plots are shown in Figure 8.34. □

8.7 PROBLEMS

P8.1 A digital resonator is to be designed with $\omega_0 = \pi/4$ that has 2 zeros at $z = 0$.

1. Compute and plot the frequency response of this resonator for $r = 0.8$, 0.9, and 0.99.
2. For each case in part 1, determine the 3 dB bandwidth and the resonant frequency ω_r from your magnitude plots.
3. Check if your results in part 2 are in agreement with the theoretical results.

P8.2 A digital resonator is to be designed with $\omega_0 = \pi/4$ that has 2 zeros at $z = 1$ and $z = -1$.

1. Compute and plot the frequency response of this resonator for $r = 0.8$, 0.9, and 0.99.
2. For each case in part 1 determine the 3 dB bandwidth and the resonant frequency ω_r from your magnitude plots.
3. Compare your results in part 2 with (8.48) and (8.47), respectively.

P8.3 We want to design a digital resonator with the following requirements: a 3 dB bandwidth of 0.05 rad, a resonant frequency of 0.375 cycles/sam, and zeros at $z = 1$ and $z = -1$. Using trial-and-error approach, determine the difference equation of the resonator.

P8.4 A notch filter is to be designed with a null at the frequency $\omega_0 = \pi/2$.

1. Compute and plot the frequency response of this notch filter for $r = 0.7$, 0.9, and 0.99.
2. For each case in part 1, determine the 3 dB bandwidth from your magnitude plots.
3. By trial-and-error approach, determine the value of r if we want the 3 dB bandwidth to be 0.04 radians at the null frequency $\omega_0 = \pi/2$.

P8.5 Repeat Problem P8.4 for a null at $\omega_0 = \pi/6$.

P8.6 A speech signal with bandwidth of 4 kHz is sampled at 8 kHz. The signal is corrupted by sinusoids with frequencies 1 kH, 2 kHz, and 3 kHz.

1. Design an IIR filter using notch filter components that eliminates these sinusoidal signals.
2. Choose the gain of the filter so that the maximum gain is equal to 1, and plot the log-magnitude response of your filter.
3. Load the `handel` sound file in MATLAB, and add the preceding three sinusoidal signals to create a corrupted sound signal. Now filter the corrupted sound signal using your filter and comment on its performance.

P8.7 Consider the system function of an IIR lowpass filter

$$H(z) = K\frac{1+z^{-1}}{1-0.9z^{-1}} \tag{8.72}$$

where K is a constant that can be adjusted to make the maximum gain response equal to 1. We obtain the system function of an Lth-order comb filter $H_L(z)$ using $H_L(z) = H\left(z^L\right)$.

1. Determine the value of K for the system function in (8.72).
2. Using the K value from part 1, determine and plot the log-magnitude response of the comb filter for $L = 6$.
3. Describe the shape of your plot in part 2.

P8.8 Consider the system function of an IIR highpass filter

$$H(z) = K\frac{1-z^{-1}}{1-0.9z^{-1}} \tag{8.73}$$

where K is a constant that can be adjusted to make the maximum gain response equal to 1. We obtain the system function of an Lth-order comb filter $H_L(z)$ using $H_L(z) = H\left(z^L\right)$.

1. Determine the value of K for the system function in (8.73).
2. Using the K value from part 1, determine and plot the log-magnitude response of the comb filter for $L = 6$.
3. Describe the shape of your plot in part 2.

P8.9 (Adapted from [19]) As discussed in Chapter 1, echos and reverberations of a signal $x(n)$ can be obtained by scaling and delaying, that is,

$$y(n) = \sum_{k=0}^{\infty} \alpha_k x(n - kD) \tag{8.74}$$

where D is a positive integer for minimum delay and $\alpha_k > \alpha_{k-1} > 0$.

1. Consider the IIR comb filter given by

$$H(z) = \frac{1}{1 - az^{-D}} \tag{8.75}$$

Determine its impulse response. Explain why this filter can be used as a reverberator.

2. Consider the cascade of three allpass comb filters

$$H(z) = \frac{z^{D_1} - a_1}{1 - a_1 z^{-D_1}} \times \frac{z^{D_2} - a_2}{1 - a_2 z^{-D_2}} \times \frac{z^{D_3} - a_3}{1 - a_3 z^{-D_3}} \tag{8.76}$$

which can be used as a practical digital reverberator. Compute and plot the impulse response of this reverberator for $D_1 = 50$, $a_1 = 0.7$; $D_2 = 41$, $a_2 = 0.665$; and $D_3 = 32$, $a_3 = 0.63175$.

3. Repeat part 2 for $D_1 = 53$, $a_1 = 0.7$; $D_2 = 40$, $a_2 = 0.665$; and $D_3 = 31$, $a_3 = 0.63175$. How does the shape of this reverberator different from the one in part 2? Which is a good reverberator?

P8.10 Consider the 1st-order allpass system function given by

$$H(z) = \frac{a + z^{-1}}{1 + az^{-1}}, \qquad 0 < a < 1 \tag{8.77}$$

The phase-delay of a system is defined as $\Phi(\omega) \triangleq -\angle H\left(e^{j\omega}\right)/\omega$ and is measured in samples.

1. Show that the phase-delay of the system in (8.77) at low frequencies is given by

$$\Phi(\omega) \approx \frac{1 - a}{1 + a} \quad \text{for } a \approx 1 \tag{8.78}$$

2. Plot the phase-delay over $-\pi/2 \le \omega \le \pi/2$ for $a = 0.9$, 0.95, and 0.99 to verify Problem P8.10. Comment on the accuracy of the results.
3. Design a 1st-order allpass system that has phase delay of 0.01 samples. Plot its magnitude and phase-delay responses.

P8.11 Consider the second-order allpass system function given by

$$H(z) = \frac{a_2 + a_1 z^{-1} + z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{8.79}$$

The phase-delay of a system is defined as $\Phi(\omega) \triangleq -\angle H\left(e^{j\omega}\right)/\omega$ and is measured in samples.

It can be shown that if we choose

$$a_1 = 1\left(\frac{2-d}{1+d}\right), \qquad a_2 = \frac{(2-d)(1-d)}{(2+d)(1+d)} \tag{8.80}$$

Then phase-delay $\Phi(\omega)$ at low frequencies is approximated by d in samples. Verify this result by plotting $\Phi(\omega)$ over $-\pi/2 \le \omega \le \pi/2$ for $d = 0.1$, $d = 0.05$, and $d = 0.01$.

P8.12 Design an analog Butterworth lowpass filter that has a 0.25 dB or better ripple at 500 rad/sec and at least 50 dB of attenuation at 2000 rad/sec. Determine the system function in a rational function form. Plot the magnitude response, the log-magnitude response in dB, the phase response, and the impulse response of the filter.

P8.13 Design an analog Butterworth lowpass filter that has a 0.5 dB or better ripple at 10 kHz and at least 45 dB of attenuation at 20 kHz. Determine the system function in a cascade form. Plot the magnitude response, the log-magnitude response in dB, the group-delay, and the impulse response of the filter.

P8.14 Design a lowpass analog Chebyshev-I filter with an acceptable ripple of 1 dB for $|\Omega| \le 10$ and an attenuation of 50 dB or greater beyond $|\Omega| = 15$ rad/sec. Determine the system function in a rational function form. Plot the magnitude response, the log-magnitude response in dB, the group-delay, and the impulse response of the filter.

P8.15 Design a lowpass analog Chebyshev-I filter with the following characteristics:

- A passband ripple of 0.5 dB,
- passband cutoff frequency of 4 kHz, and
- stopband attenuation of 45 dB or greater beyond 20 kHz.

Determine the system function in a cascade form. Plot the magnitude response, the log-magnitude response in dB, the phase response, and the impulse response of the filter.

P8.16 A signal $x_a(t)$ contains two frequencies, 10 kHz and 15 kHz. We want to suppress the 15 kHz component to 50 dB attenuation while passing the 10 kHz component with less than 0.25 dB attenuation. Design a minimum-order Chebyshev-II analog filter to perform this filtering operation. Plot the log-magnitude response, and verify the design.

P8.17 Design an analog Chebyshev-II lowpass filter that has a 0.25 dB or better ripple at 250 Hz and at least 40 dB of attenuation at 400 Hz. Plot the magnitude response, the log-magnitude response in dB, the group-delay, and the impulse response of the filter.

P8.18 A signal $x_a(t)$ contains two frequencies, 10 kHz and 15 kHz. We want to suppress the 15 kHz component to 50 dB attenuation while passing the 10 kHz component with less than 0.25 dB attenuation. Design a minimum-order elliptic analog filter to perform this filtering operation. Plot the log-magnitude response and verify the design. Compare your design with the Chebyshev-II design in Problem P8.16.

P8.19 Design an analog elliptic lowpass filter that has a 0.25 dB or better ripple at 500 rad/sec and at least 50 dB of attenuation at 2000 rad/sec. Determine the system function in a rational function form. Plot the magnitude response, the log-magnitude response in dB, the phase response, and the impulse response of the filter. Compare your design with the Butterworth design in Problem P8.12.

P8.20 Write a MATLAB function to design analog lowpass filters. The format of this function should be

```
function [b,a] =afd(type,Fp,Fs,Rp,As)
%
% function [b,a] =afd(type,Fp,Fs,Rp,As)
%   Designs analog lowpass filters
% type = 'butter' or 'cheby1' or 'cheby2' or 'ellip'
%   Fp = passband cutoff in Hz
%   Fs = stopband cutoff in Hz
%   Rp = passband ripple in dB
%   As = stopband attenuation in dB
```

Use the **afd_butt**, **afd_chb1**, **afd_chb2**, and **afd_elip** functions developed in this chapter. Check your function using specifications given in Problems P8.12 through P8.17.

P8.21 We want to design a Chebyshev-I prototype lowpass digital filter operating at a sampling rate of 8 kHz with a passband edge of 3.2 kHz, a passband ripple of 0.5 dB, and a minimum stopband attenuation of 45 dB at 3.8 kHz.

1. Using the impulse invariance transformation with $T = 1$ sec, design the digital filter. Plot the magnitude and the log-magnitude responses as functions of analog frequency in kHz.
2. Repeat part 1 using $T = 1/8000$ sec.
3. Compare the above two designs in parts 1 and 2 in terms of their frequency responses. Comment on the effect of T on the impulse invariance design.

P8.22 Design a Butterworth digital lowpass filter to satisfy the specifications:

$$\text{passband edge:}\quad 0.4\pi, \quad R_p = 0.5 \text{ dB}$$
$$\text{stopband edge:}\quad 0.6\pi, \quad As = 50 \text{ dB}$$

Use the impulse invariance method with $T = 2$. Determine the system function in the rational form, and plot the log-magnitude response in dB. Plot the impulse response $h(n)$ and the impulse response $h_a(t)$ of the analog prototype and compare their shapes.

P8.23 Write a MATLAB function to design digital lowpass filters based on the impulse invariance transformation. The format of this function should be

```
function [b,a] =dlpfd_ii(type,wp,ws,Rp,As,T)
%
% function [b,a] =dlpfd_ii(type,wp,ws,Rp,As,T)
%   Designs digital lowpass filters using impulse invariance
% type = 'butter' or 'cheby1'
%   wp = passband cutoff in Hz
%   ws = stopband cutoff in Hz
%   Rp = passband ripple in dB
%   As = stopband attenuation in dB
%    T = sampling interval
```

Use the **afd** function developed in Problem P8.20. Check your function on specifications given in Problems P8.21 and P8.22.

P8.24 In this problem we will develop a technique called the *step invariance* transformation. In this technique, the step response of an analog prototype filter is preserved in the resulting digital filter; i.e., if $v_a(t)$ is the step response of the prototype and if $v(n)$ is the step response of the digital filter, then

$$v(n) = v_a(t)|_{t=nT}\,, \quad T: \text{sampling interval}$$

Note that the frequency-domain quantities are related by

$$V_a(s) \overset{\triangle}{=} \mathcal{L}\left[v_a(t)\right] = H_a(s)/s$$

and

$$V(z) \overset{\triangle}{=} \mathcal{Z}\left[v(n)\right] = H(z)\frac{1}{1-z^{-1}}$$

Hence the step invariance transformation steps are: Given $H_a(s)$

- divide $H_a(s)$ by s to obtain $V_a(s)$,
- find residues $\{R_k\}$ and poles $\{p_k\}$ of $V_a(s)$,
- transform analog poles $\{p_k\}$ into digital poles $\left\{e^{p_kT}\right\}$ where T is arbitrary,
- determine $V(z)$ from residues $\{R_k\}$ and poles $\left\{e^{p_kT}\right\}$, and finally
- determine $H(z)$ by multiplying $V(z)$ by $\left(1-z^{-1}\right)$.

Use the above procedure to develop a MATLAB function to implement the step invariance transformation. The format of this function should be

```
function [b,a] =stp_invr(c,d,T)
% Step Invariance Transformation from Analog to Digital Filter
% [b,a] =stp_invr(c,d,T)
%  b = Numerator polynomial in z^(-1) of the digital filter
%  a = Denominator polynomial in z^(-1) of the digital filter
%  c = Numerator polynomial in s of the analog filter
%  d = Denominator polynomial in s of the analog filter
%  T = Sampling (transformation) parameter
```

P8.25 Design the lowpass Butterworth digital filter of Problem P8.22 using the step invariance method. Plot the log-magnitude response in dB and compare it with that in Problem P8.22. Plot the step response $v(n)$ and the impulse response $v_a(t)$ of the analog prototype and compare their shapes.

P8.26 In this chapter we discussed a filter transformation technique called the matched-z transformation. Using (8.69) write a MATLAB function called `mzt` that maps the analog system function $H_a(s)$ into the digital system function $H(z)$. The format of the function should be

```
function [b,a] = mzt(c,d,T)
% Matched-Z Transformation from Analog to Digital Filter
% [b,a] = MZT(c,d,T)
%  b = Numerator polynomial in z^(-1) of the digital filter
%  a = Denominator polynomial in z^(-1) of the digital filter
%  c = Numerator polynomial in s of the analog filter
%  d = Denominator polynomial in s of the analog filter
%  T = Sampling interval (transformation parameter)
```

Using this function, transform

$$H_a(s) = \frac{s+1}{s^2+5s+6}$$

into a digital filter $H(z)$ for the sampling intervals (in seconds): $T = 0.05$, $T = 0.1$, and $T = 0.2$. In each case obtain a plot similar to that in Figure 8.20 and comment on the performance of this technique.

P8.27 Consider an analog Butterworth lowpass filter that has a 1 dB or better ripple at 100 Hz and at least 30 dB of attenuation at 150 Hz. Transform this filter into a digital filter using the matched-z transformation technique in which $F_s = 1000$ Hz. Plot the magnitude and phase response of the resulting digital filter and determine the exact band-edge frequencies for the given dB specifications. Comment on the results.

P8.28 Consider an analog Chebyshev-I lowpass filter that has a 0.5 dB or better ripple at 500 Hz and at least 40 dB of attenuation at 700 Hz. Transform this filter into a digital filter using the matched-z transformation technique in which $F_s = 2000$ Hz. Plot the magnitude and phase response of the resulting digital filter and determine the exact band-edge frequencies for the given dB specifications. Comment on the results.

P8.29 Consider an analog Chebyshev-II lowpass filter that has a 0.25 dB or better ripple at 1500 Hz and at least 80 dB of attenuation at 2000 Hz. Transform this filter into a digital filter using the matched-z transformation technique in which $F_s = 8000$ Hz. Plot the magnitude and phase response of the resulting digital filter, and determine the exact band-edge frequencies for the given dB specifications. Comment on the results. Is this a satisfactory design?

P8.30 Consider the design of the lowpass Butterworth filter of Problem P8.22.

1. Use the bilinear transformation technique outlined in this chapter and the `bilinear` function. Plot the log-magnitude response in dB. Compare the impulse responses of the analog prototype and the digital filter.
2. Use the `butter` function and compare this design with the one in part 1.

P8.31 Consider the design of the digital Chebyshev-1 filter of Problem P8.21.

1. Use the bilinear transformation technique outlined in this chapter and the `bilinear` function. Plot the log-magnitude response in dB. Compare the impulse responses of the analog prototype and the digital filter.
2. Use the `cheby1` function and compare this design with the one above.

P8.32 Design a digital lowpass filter using elliptic prototype to satisfy the requirements:

$$\text{passband edge: } 0.3\pi,\ R_p = 0.25 \text{ dB}$$
$$\text{stopband edge: } 0.4\pi,\ As = 50 \text{ dB}$$

Use the `bilinear` as well as the `ellip` function and compare your designs.

P8.33 Design a digital lowpass filter to satisfy the specifications:

$$\text{passband edge: } 0.45\pi,\ R_p = 0.5 \text{ dB}$$
$$\text{stopband edge: } 0.5\pi,\ As = 60 \text{ dB}$$

1. Use the `butter` function and determine the order N and the actual minimum stopband attenuation in dB.
2. Use the `cheby1` function and determine the order N and the actual minimum stopband attenuation in dB.
3. Use the `cheby2` function and determine the order N and the actual minimum stopband attenuation in dB.
4. Use the `ellip` function and determine the order N and the actual minimum stopband attenuation in dB.
5. Compare the orders, the actual minimum stopband attenuations, and the group delays in each of the above designs.

P8.34 Write a MATLAB function to determine the lowpass prototype digital filter frequencies from an highpass digital filter specifications using the procedure outlined in this chapter. The format of this function should be

```
function [wpLP,wsLP,alpha] = hp2lpfre(wphp,wshp)
% Band-edge frequency conversion from highpass to lowpass digital filter
% [wpLP,wsLP,a] = hp2lpfre(wphp,wshp)
%  wpLP = passband egde for the lowpass prototype
%  wsLP = stopband egde for the lowpass prototype
% alpha = lowpass to highpass transformation parameter
%  wphp = passband egde for the highpass
%  wshp = stopband egde for the highpass
```

Using this function develop a MATLAB function to design a highpass digital filter using the bilinear transformation. The format of this function should be

```
function [b,a] = dhpfd_bl(type,wp,ws,Rp,As)
% IIR Highpass filter design using bilinear transformation
% [b,a] = dhpfd_bl(type,wp,ws,Rp,As)
% type = 'butter' or 'cheby1' or 'chevy2' or 'ellip'
%    b = Numerator polynomial of the highpass filter
%    a = Denominator polynomial of the highpass filter
%   wp = Passband frequency in radians
%   ws = Stopband frequency in radians (wp < ws)
%   Rp = Passband ripple in dB
%   As = Stopband attenuation in dB
```

Verify your function using the specifications in Example 8.27.

P8.35 Design a highpass filter to satisfy the specifications:

$$\begin{aligned}&\text{stopband edge: } 0.4\pi,\ A_s = 60 \text{ dB}\\&\text{passband edge: } 0.6\pi,\ R_p = 0.5 \text{ dB}\end{aligned}$$

1. Use the `dhpfd_bl` function of Problem P8.34 and the Chebyshev-I prototype to design this filter. Plot the log-magnitude response in dB of the designed filter.
2. Use the `cheby1` function for design and plot the log-magnitude response in dB. Compare these two designs.

P8.36 Write a MATLAB function to determine the lowpass prototype digital filter frequencies from an arbitrary lowpass digital filter specifications using the functions given in Table 8.2 and the procedure outlined for highpass filters. The format of this function should be

```
function [wpLP,wsLP,alpha] = lp2lpfre(wplp,wslp)
% Band-edge frequency conversion from lowpass to lowpass digital filter
% [wpLP,wsLP,a] = lp2lpfre(wplp,wslp)
%  wpLP = passband egde for the lowpass prototype
%  wsLP = stopband egde for the lowpass prototype
% alpha = lowpass to highpass transformation parameter
%  wplp = passband egde for the given lowpass
%  wslp = passband egde for the given lowpass
```

Using this function, develop a MATLAB function to design a lowpass filter from a prototype lowpass digital filter using the bilinear transformation. The format of this function should be

```
function [b,a] = dlpfd_bl(type,wp,ws,Rp,As)
% IIR lowpass filter design using bilinear transformation
% [b,a] = dlpfd_bl(type,wp,ws,Rp,As)
% type = 'butter' or 'cheby1' or 'chevy2' or 'ellip'
%    b = Numerator polynomial of the bandpass filter
%    a = Denominator polynomial of the bandpass filter
%   wp = Passband frequency in radians
%   ws = Stopband frequency in radians
%   Rp = Passband ripple in dB
%   As = Stopband attenuation in dB
```

Verify your function using the designs in Problem P8.33.

P8.37 Design a bandpass digital filter using the `Cheby2` function. The specifications are:

$$\begin{array}{rl}\text{lower stopband edge: } 0.3\pi & \multirow{2}{*}{$A_s = 50 \text{ dB}$}\\ \text{upper stopband edge: } 0.6\pi & \\ \text{lower passband edge: } 0.4\pi & \multirow{2}{*}{$R_p = 0.5 \text{ dB}$}\\ \text{upper passband edge: } 0.5\pi & \end{array}$$

Plot the impulse response and the log-magnitude response in dB of the designed filter.

P8.38 Write a MATLAB function to determine the lowpass prototype digital filter frequencies from a bandpass digital filter specifications using the functions given in Table 8.2 and the procedure outlined for highpass filters. The format of this function should be

```
function [wpLP,wsLP,alpha] = bp2lpfre(wpbp,wsblp)
% Band-edge frequency conversion from bandpass to lowpass digital filter
% [wpLP,wsLP,a] = bp2lpfre(wpbp,wsbp)
%  wpLP = passband egde for the lowpass prototype
%  wsLP = stopband egde for the lowpass prototype
% alpha = lowpass to highpass transformation parameter
%  wpbp = passband egde frequency array [wp_lower, wp_upper] for the bandpass
%  wsbp = passband egde frequency array [ws_lower, ws_upper] for the bandpass
```

Using this function, develop a MATLAB function to design a bandpass filter from a prototype lowpass digital filter using the bilinear transformation. The format of this function should be

```
function [b,a] = dbpfd_bl(type,wp,ws,Rp,As)
% IIR bandpass filter design using bilinear transformation
% [b,a] = dbpfd_bl(type,wp,ws,Rp,As)
% type = 'butter' or 'cheby1' or 'chevy2' or 'ellip'
%    b = Numerator polynomial of the bandpass filter
%    a = Denominator polynomial of the bandpass filter
%   wp = Passband frequency array [wp_lower, wp_upper] in radians
%   ws = Stopband frequency array [wp_lower, wp_upper] in radians
%   Rp = Passband ripple in dB
%   As = Stopband attenuation in dB
```

Verify your function using the design in Problem P8.37.

P8.39 We wish to use the Chebyshev-I prototype to design a bandstop digital IIR filter that meets the following specifications:

$$\begin{aligned} 0.95 \le |H(e^{j\omega})| &\le 1.05, & 0 \le |\omega| &\le 0.25\pi \\ 0 \le |H(e^{j\omega})| &\le 0.01, & 0.35\pi \le |\omega| &\le 0.65\pi \\ 0.95 \le |H(e^{j\omega})| &\le 1.05, & 0.75\pi \le |\omega| &\le \pi \end{aligned}$$

Use the `cheby1` function and determine the system function $H(z)$ of such a filter. Provide a plot containing subplots of the log-magnitude response in dB and the impulse response.

P8.40 Write a MATLAB function to determine the lowpass prototype digital filter frequencies from a bandstop digital filter specifications using the functions given in Table 8.2 and the procedure outlined for highpass filters. The format of this function should be

```
function [wpLP,wsLP,alpha] = bs2lpfre(wpbp,wsblp)
% Band-edge frequency conversion from bandstop to lowpass digital filter
% [wpLP,wsLP,a] = bs2lpfre(wpbp,wsbp)
%  wpLP = passband egde for the lowpass prototype
%  wsLP = stopband egde for the lowpass prototype
% alpha = lowpass to highpass transformation parameter
%  wpbp = passband egde frequency array [wp_lower, wp_upper] for the bandstop
%  wsbp = passband egde frequency array [ws_lower, ws_upper] for the bandstop
```

Using this function, develop a MATLAB function to design a bandstop filter from a prototype lowpass digital filter using the bilinear transformation. The format of this function should be

```
function [b,a] = dbsfd_bl(type,wp,ws,Rp,As)
% IIR bandstop filter design using bilinear transformation
% [b,a] = dbsfd_bl(type,wp,ws,Rp,As)
% type = 'butter' or 'cheby1' or 'chevy2' or 'ellip'
%    b = Numerator polynomial of the bandstop filter
%    a = Denominator polynomial of the bandstop filter
%   wp = Passband frequency array [wp_lower, wp_upper] in radians
%   ws = Stopband frequency array [wp_lower, wp_upper] in radians
%   Rp = Passband ripple in dB
%   As = Stopband attenuation in dB
```

Verify your function using the design in Problem P8.39.

P8.41 An analog signal

$$x_a(t) = 3\sin(40\pi t) + 3\cos(50\pi t)$$

is to be processed by a

$$x_a(t) \longrightarrow \boxed{\text{A/D}} \longrightarrow \boxed{H(z)} \longrightarrow \boxed{\text{D/A}} \longrightarrow y_a(t)$$

system in which the sampling frequency is 100 sam/sec

1. Design a minimum order IIR digital filter that will pass the first component of $x_a(t)$ with attenuation of less than 1 dB and suppress the second component to at least 50 dB. The filter should have a monotone passband and an equiripple stopband. Determine the system function in rational function form and plot the log-magnitude response.
2. Generate 500 samples (sampled at 100 sam/sec) of the signal $x_a(t)$ and process through the designed filter to obtain the output sequence. Interpolate this sequence (using any one of the interpolating techniques discussed in Chapter 3) to obtain $y_a(t)$. Plot the input and the output signals and comment on your results.

P8.42 Using the bilinear transformation method, design a 10th-order elliptic bandstop filter to remove the digital frequency $\omega = 0.44\pi$ with bandwidth of 0.08π. Choose a reasonable value for the stopband attenuation. Plot the magnitude response. Generate 201 samples of the sequence

$$x(n) = \sin[0.44\pi n], \quad n = 0, \ldots, 200$$

and process thorough the bandstop filter. Comment on your results.

P8.43 Design a digital highpass filter $H(z)$ to be used in a

$$x_a(t) \longrightarrow \boxed{\text{A/D}} \longrightarrow \boxed{H(z)} \longrightarrow \boxed{\text{D/A}} \longrightarrow y_a(t)$$

structure to satisfy the following requirements:

- sampling rate of 10 Khz
- stopband edge of 1.5 Khz with attenuation of 40 dB
- passband edge of 2 Khz with ripple of 1dB
- equiripple passband and stopband
- bilinear transformation method

1. Plot the magnitude response of the overall analog filter over the $[0, 5 \text{ Khz}]$ interval.
2. Plot the magnitude response of the digital lowpass prototype.
3. What limitations must be placed on the input signals so that the preceding structure truly acts as a highpass filter to them?

P8.44 The filter specifications shown in Figure P8.1 can be considered a combination of a bandpass and a highpass specifications. Design a minimum-order IIR digital filter to satisfy these specifications. Provide a plot of the magnitude response with grid-lines as shown in Figure P8.1. From your design and plot determine the order of the filter and the exact band-edge frequencies.

P8.45 The filter specifications shown in Figure P8.2 can be considered as a combination of a lowpass and a bandpass specifications. Design a minimum-order IIR digital filter to satisfy these specifications. Provide a plot of the magnitude response with grid-lines as shown in Figure P8.2. From your design and plot determine the order of the filter and the exact band-edge frequencies.

P8.46 Design a minimum-order IIR digital filter to satisfy the following specifications:

- a passband over the $[0.35\pi, 0.5\pi]$ interval
- stopbands over the $[0, 0.3\pi]$ and $[0.6\pi, \pi]$ intervals

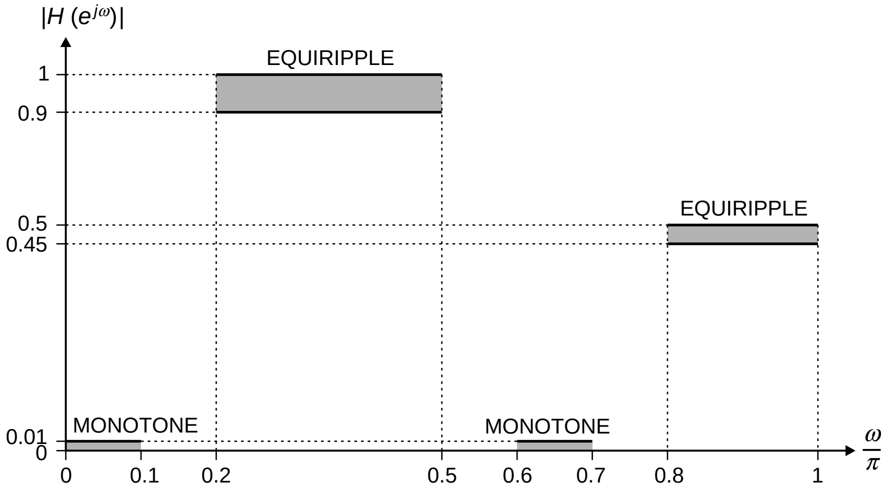


FIGURE P8.1 *Filter specifications for Problem P8.44*

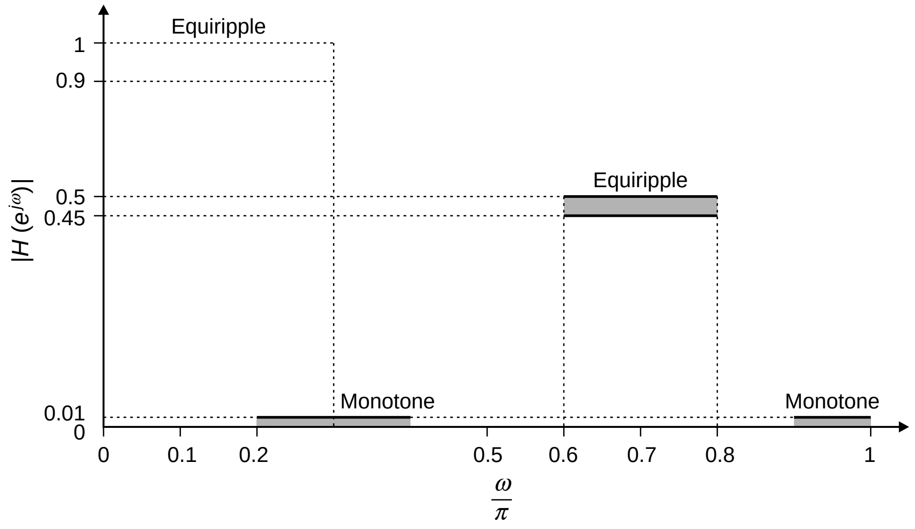


FIGURE P8.2 *Filter specifications for Problem P8.45*

- passband ripple of 1 dB
- stopband attenuation of 40 db
- equiripple passbands and stopband

Determine the system function $H(z)$ of the designed filter in the rational function form. Provide a plot of the log-magnitude response in dB. From your design and plot, answer the following questions:

1. What is the order of the filter?
2. From your plot what are the exact band-edge frequencies for the given passband and stopband attenuations?
3. Why is there a discrepancy between the specification frequencies and the exact frequencies?