

5

Finite Impulse Response Filters

Jesse D. Olson

A finite impulse response (FIR) filter has a unit impulse response that has a limited number of terms, as opposed to an infinite impulse response (IIR) filter which produces an infinite number of output terms when a unit impulse is applied to its input. FIR filters are generally realized nonrecursively, which means that there is no feedback involved in computation of the output data. The output of the filter depends only on the present and past inputs. This quality has several important implications for digital filter design and applications. This chapter discusses several FIR filters typically used for real-time ECG processing, and also gives an overview of some general FIR design techniques.

5.1 CHARACTERISTICS OF FIR FILTERS

5.1.1 Finite impulse response

Finite impulse response implies that the effect of transients or initial conditions on the filter output will eventually die away. Figure 5.1 shows a signal-flow graph (SFG) of a FIR filter realized nonrecursively. The filter is merely a set of “tap weights” of the delay stages. The unit impulse response is equal to the tap weights, so the filter has a difference equation given by Eq. (5.1), and a transfer function equation given by Eq. (5.2).

$$y(nT) = \sum_{k=0}^N b_k x(nT - kT) \quad (5.1)$$

$$H(z) = b_0 + b_1 z^{-1} + b_2 z^{-2} + \dots + b_N z^{-N} \quad (5.2)$$

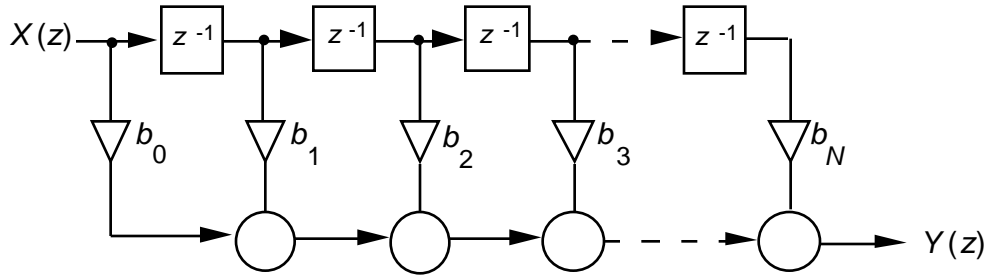


Figure 5.1 The output of a FIR filter of order N is the weighted sum of the values in the storage registers of the delay line.

5.1.2 Linear phase

In many biomedical signal processing applications, it is important to preserve certain characteristics of a signal throughout the filtering operation, such as the height and duration of the QRS pulse. A filter with linear phase has a pure time delay as its phase response, so phase distortion is minimized. A filter has linear phase if its frequency response $H(e^{j\theta})$ can be expressed as

$$H(e^{j\theta}) = H_1(\theta) e^{-j(\alpha\theta + \beta)} \quad (5.3)$$

where $H_1(\theta)$ is a real and even function, since the phase of $H(e^{j\theta})$ is

$$H(e^{j\theta}) = \begin{cases} -\alpha\theta - \beta & ; H_1(\theta) > 0 \\ -\alpha\theta - \beta - \pi & ; H_1(\theta) < 0 \end{cases} \quad (5.4)$$

FIR filters can easily be designed to have a linear phase characteristic. Linear phase can be obtained in four ways, as combinations of even or odd symmetry (defined as follows) with even or odd length.

$$\left. \begin{array}{l} h(N-1-k) = h(k), \text{ even symmetry} \\ h(N-1-k) = -h(k), \text{ odd symmetry} \end{array} \right\} \text{ for } 0 \leq k \leq N \quad (5.5)$$

5.1.3 Stability

Since a nonrecursive filter does not use feedback, it has no poles except those that are located at $z = 0$. Thus there is no possibility for a pole to exist outside the unit circle. This means that it is inherently stable. As long as the input to the filter is bounded, the output of the filter will also be bounded. This contributes to ease of design, and makes FIR filters especially useful for adaptive filtering where filter coefficients change as a function of the input data. Adaptive filters are discussed in Chapter 8.

5.1.4 Desirable finite-length register effects

When data are converted from analog form to digital form, some information is lost due to the finite number of storage bits. Likewise, when coefficient values for a filter are calculated, digital implementation can only approximate the desired values. The limitations introduced by digital storage are termed finite-length register effects. Although we will not treat this subject in detail in this book, finite-length register effects can have significant negative impact on a filter design. These effects include quantization error, roundoff noise, limit cycles, conditional stability, and coefficient sensitivity. In FIR filters, these effects are much less significant and easier to analyze than in IIR filters since the errors are not fed back into the filter. See Appendix F for more details about finite-length register effects.

5.1.5 Ease of design

All of the above properties contribute to the ease in designing FIR filters. There are many straightforward techniques for designing FIR filters to meet arbitrary frequency and phase response specifications, such as window design or frequency sampling. Many software packages exist that automate the FIR design process, often computing a filter realization that is in some sense optimal.

5.1.6 Realizations

There are three methods of realizing an FIR filter (Bogner and Constantinides, 1985). The most common method is direct convolution, in which the filter's unit impulse sequence is convolved with the present input and past inputs to compute each new output value. FIR filters attenuate the signal very gradually outside the passband (i.e., they have slow rolloff characteristics). Since they have significantly slower rolloff than IIR filters of the same length, for applications that require sharp rolloffs, the order of the FIR filter may be quite large. For higher-order filters the direct convolution method becomes computationally inefficient.

For FIR filters of length greater than about 30, the "fast convolution" realization offers a computational savings. This technique takes advantage of the fact that time-domain multiplication, the frequency-domain dual of convolution, is computationally less intensive. Fast convolution involves taking the FFT of a block of data, multiplying the result by the FFT of the unit impulse sequence, and finally taking the inverse FFT. The process is repeated for subsequent blocks of data. This method is discussed in detail in section 11.3.2.

The third method of realizing FIR filters is an advanced, recursive technique involving a comb filter and a bank of parallel digital resonators (Rabiner and Rader, 1972). This method is advantageous for frequency sampling designs if a large number of the coefficients in the desired frequency response are zero, and can be used for filters with integer-valued coefficients, as discussed in Chapter 7. For the remainder of this chapter, only the direct convolution method will be considered.

5.2 SMOOTHING FILTERS

One of the most common signal processing tasks is smoothing of the data to reduce high-frequency noise. Some sources of high-frequency noise include 60-Hz, movement artifacts, and quantization error. One simple method of reducing high-frequency noise is to simply average several data points together. Such a filter is referred to as a moving average filter.

5.2.1 Hanning filter

One of the simplest smoothing filters is the Hanning moving average filter. Figure 5.2 summarizes the details of this filter. As illustrated by its difference equation, the Hanning filter computes a weighted moving average, since the central data point has twice the weight of the other two:

$$y(nT) = \frac{1}{4} [x(nT) + 2x(nT - T) + x(nT - 2T)] \quad (5.6)$$

As we saw in section 4.5, once we have the difference equation representing the numerical algorithm for implementing a digital filter, we can quickly determine the transfer equation that totally characterizes the performance of the filter by using the analogy between discrete-time variables and z -domain variables.

Recognizing that $x(nT)$ and $y(nT)$ are points in the input and output sequences associated with the current sample time, they are analogous to the undelayed z -domain variables, $X(z)$ and $Y(z)$ respectively. Similarly $x(nT - T)$, the input value one sample point in the past, is analogous to the z -domain input variable delayed by one sample point, or $X(z)z^{-1}$. We can then write an equation for output $Y(z)$ as a function of input $X(z)$:

$$Y(z) = \frac{1}{4} [X(z) + 2X(z)z^{-1} + X(z)z^{-2}] \quad (5.7)$$

The block diagram of Figure 5.2(a) is drawn using functional blocks to directly implement the terms in this equation. Two delay blocks are required as designated by the -2 exponent of z . Two multipliers are necessary to multiply by the factors 2 and $1/4$, two summers are needed to combine the terms. The transfer function of this equation is

$$H(z) = \frac{1}{4} [1 + 2z^{-1} + z^{-2}] \quad (5.8)$$

This filter has two zeros, both located at $z = -1$, and two poles, both located at $z = 0$ (see section 4.6 to review how to find pole and zero locations). Figure 5.2(b) shows the pole-zero plot. Note the poles are implicit; they are not drawn since they influence all frequencies in the amplitude response equally.

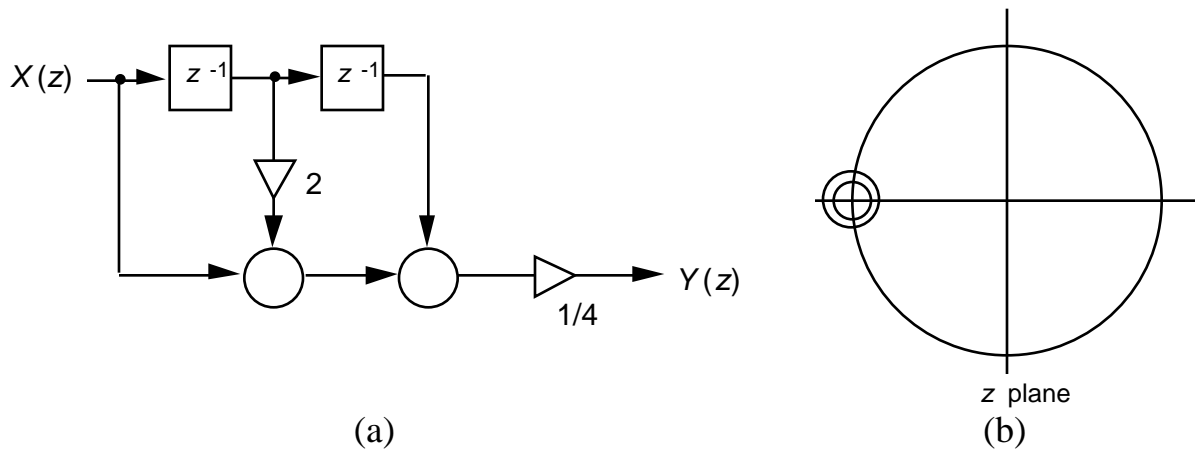


Figure 5.2 Hanning filter. (a) Signal-flow graph. (b) Pole-zero diagram.

The filter's amplitude and phase responses are found by substituting $e^{j\omega T}$ for z in Eq. (5.8):

$$H(\omega T) = \frac{1}{4} [1 + 2e^{-j\omega T} + e^{-j2\omega T}] \quad (5.9)$$

We could now directly substitute into this function the trigonometric relationship

$$e^{j\omega T} = \cos(\omega T) + j\sin(\omega T) \quad (5.10)$$

However, a common trick prior to this substitution that leads to quick simplification of expressions such as this one is to extract a power of e as a multiplier such that the final result has two similar exponential terms with equal exponents of opposite sign

$$H(\omega T) = \frac{1}{4} [e^{-j\omega T} (e^{j\omega T} + 2 + e^{-j\omega T})] \quad (5.11)$$

Now substituting Eq. (5.10) for the terms in parentheses yields

$$H(\omega T) = \frac{1}{4} e^{-j\omega T} [\cos(\omega T) + j\sin(\omega T) + 2 + \cos(\omega T) - j\sin(\omega T)] \quad (5.12)$$

The $\sin(\omega T)$ terms cancel leaving

$$H(\omega T) = \frac{1}{4} [(2 + 2 \cos(\omega T))e^{-j\omega T}] \quad (5.13)$$

This is of the form $Re^{j\theta}$ where R is the real part and θ is the phase angle. Thus, the magnitude response of the Hanning filter is $|R|$, or

$$|H(\omega T)| = \frac{1}{2} [1 + \cos(\omega T)] \quad (5.14)$$

Figure 5.3(a) shows this cosine-wave amplitude response plotted with a linear ordinate scale while Figure 5.3(b) shows the same response using the more familiar decibel plot, which we will use throughout this book. The relatively slow rolloff of the Hanning filter can be sharpened by passing its output into the input of another identical filter. This process of connecting multiple filters together is called cascading filters. The linear phase response shown in Figure 5.3(c) is equal to angle θ , or

$$H(\omega T) = -\omega T \quad (5.15)$$

Implementation of the Hanning filter is accomplished by writing a computer program. Figure 5.4 illustrates a C-language program for an off-line (i.e., not real-time) application where data has previously been sampled by an A/D converter and left in an array. This program directly computes the filter's difference equation [Eq. (5.6)]. Within the `for()` loop, a value for $x(nT)$ (called `xnt` in the program) is obtained from the array `idb[]`. The difference equation is computed to find the output value $y(nT)$ (or `ynt` in the program). This value is saved into the data array, replacing the value of $x(nT)$. Then the input data variables are shifted through the delay blocks. Prior to the next input, the data point that was one point in the past $x(nT - T)$ (called `xm1` in the program) moves two points in the past and becomes $x(nT - 2T)$ (or `xm2`). The most recent input $x(nT)$ (called `xnt`) moves one point back in time, replacing $x(nT - T)$ (or `xm1`). In the next iteration of the `for()` loop, a new value of $x(nT)$ is retrieved, and the process repeats until all 256 array values are processed. The filtered output waveform is left in array `idb[]`.

The Hanning filter is particularly efficient for use in real-time applications since all of its coefficients are integers, and binary shifts can be used instead of multiplications. Figure 5.5 is a real-time Hanning filter program. In this program, the computation of the output value $y(nT)$ must be accomplished during one sample interval T . That is, every new input data point acquired by the A/D converter must produce an output value before the next A/D input. Otherwise the filter would not keep up with the sampling rate, and it would not be operating in real time.

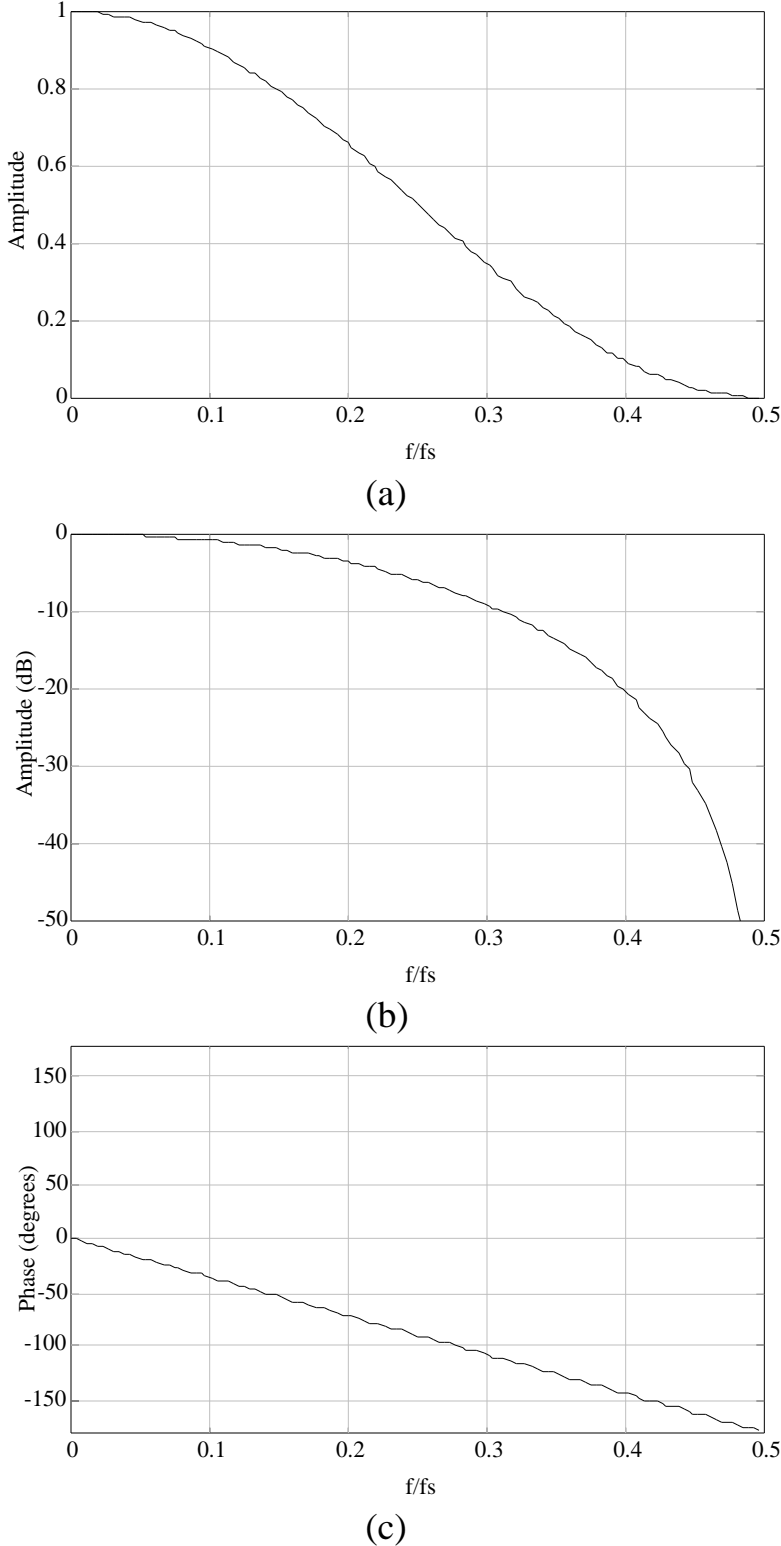


Figure 5.3 Hanning filter. (a) Frequency response (linear magnitude axis). (b) Frequency response (dB magnitude axis). (c)Phase response.

In this program, sampling from the A/D converter, computation of the results, and sending the filtered data to a D/A converter are all accomplished within a `for()` loop. The `wait()` function is designed to wait for an interrupt caused by an A/D clock tick. Once the interrupt occurs, a data point is sampled with the `adget()` function and set equal to `xnt`. The Hanning filter's difference equation is then computed using C-language shift operators to do the multiplications efficiently. Expression `<<1` is a binary shift to the left by one bit position corresponding to multiplication by a factor of two, and `>>2` is a binary shift right of two bit positions representing division by four.

```

/* Hanning filter
   Difference equation:
       y(nT) = (x(nT) + 2*x(nT - T) + x(nT - 2T))/4
   C language implementation equation:
       ynt = (xnt + 2*xm1 + xm2)/4;
*/
main()
{
    int i, xnt, xm1, xm2, ynt, idb[256];

    xm2 = 0;
    xm1 = 0;

    for(i = 0; i <= 255; i++)
    {
        xnt = idb[i];
        ynt = (xnt + 2*xm1 + xm2)/4;
        idb[i] = ynt;
        xm2 = xm1;
        xm1 = xnt;
    }
}

```

Figure 5.4 C-language code to implement the Hanning filter. Data is presampled by an ADC and stored in array `idb[]`. The filtered signal is left in `idb[]`.

The computed output value is sent to a D/A converter with function `daput()`. Then the input data variables are shifted through the delay blocks as in the previous program. For the next input, the data point that was one point in the past $x(nT - T)$ (called `xm1` in the program) moves two points in the past and becomes $x(nT - 2T)$ (or `xm2`). The most recent input $x(nT)$ (called `xnt`) moves one point back in time, replacing $x(nT - T)$ (or `xm1`). Then the `for()` loop repeats with the `wait()` function waiting until the next interrupt signals that a new sampled data point is available to be acquired by `adget()` as the new value for $x(nT)$.

```

/* Real-time Hanning filter
   Difference equation:
       y(nT) = (x(nT) + 2*x(nT - T) + x(nT - 2T))/4
   C language implementation equation:
       ynt = (xnt + xm1<<1 + xm2)>>2;
*/

#define AD 31;
#define DA 32;

main()
{
    int i, xnt, xm1, xm2, ynt;

    xm2 = 0;
    xm1 = 0;

    tmic 2000;          /* Start ADC clock ticking at 2000 μs */
                       /* intervals (2 ms period for 500 sps) */

    for( ; ; )
        {
            wait();          /* Wait for ADC clock to tick */
            xnt = adget(AD);
            ynt = (xnt + xm1<<1 + xm2)>>2;
            daput(ynt, DA);
            xm2 = xm1;
            xm1 = xnt;
        }
}

```

Figure 5.5 C-language code to implement the real-time Hanning filter.

5.2.2 Least-squares polynomial smoothing

This family of filters fits a parabola to an odd number ($2L + 1$) of input data points in a least-squares sense. Figure 5.6(a) shows that the output of the filter is the midpoint of the parabola. Writing the equation for a parabola at each input point, we obtain

$$p(nT + kT) = a(nT) + b(nT)k + c(nT)k^2 \quad (5.16)$$

where k ranges from $-L$ to L . The fit is found by selecting $a(nT)$, $b(nT)$ and $c(nT)$ to minimize the squared error between the parabola and the input data. Setting the partial derivatives of the error with respect to $a(nT)$, $b(nT)$, and $c(nT)$ equal to zero results in a set of simultaneous equations in $a(nT)$, $b(nT)$, $c(nT)$, k , and $p(nT - kT)$. Solving to obtain an expression for $a(nT)$, the value of the parabola at $k = 0$, yields an expression that is a function of the input values. The coefficients of this expression are the tap weights for the least-squares polynomial filter as shown in the signal-flow graph of Figure 5.6(b) for a five-point filter. The difference equation for the five-point parabolic filter is

$$y(nT) = \frac{1}{35} [(-3x(nT) + 12x(nT - T) + 17x(nT - 2T) + 12x(nT - 3T) - 3x(nT - 4T))] \quad (5.17)$$

Its transfer function is

$$H(z) = \frac{1}{35} [-3 + 12z^{-1} + 17z^{-2} + 12z^{-3} - 3z^{-4}] \quad (5.18)$$

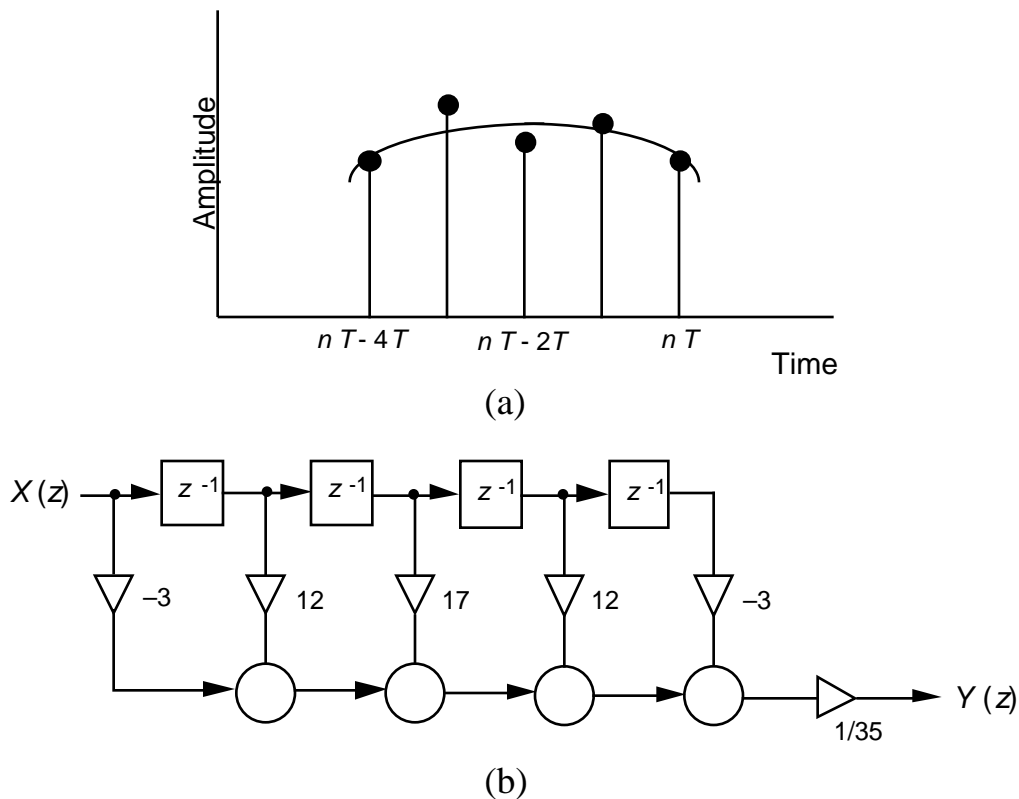


Figure 5.6 Polynomial smoothing filter with $L = 2$. (a) Parabolic fitting of groups of 5 sampled data points. (b) Signal-flow graph.

Figure 5.7 shows the tap weights for filters with L equal to 2, 3, 4, and 5, and Figure 5.8 illustrates their responses. The order of the filter can be chosen to meet the desired rolloff.

L	Tap weights
2	$\frac{1}{35}(-3, 12, 17, 12, -3)$
3	$\frac{1}{21}(-2, 3, 6, 7, 6, 3, -2)$
4	$\frac{1}{231}(-21, 14, 39, 54, 59, 54, 39, 14, -21)$
5	$\frac{1}{429}(-36, 9, 44, 69, 84, 89, 84, 69, 44, 9, -36)$

Figure 5.7 Tap weights of polynomial smoothing filters (Hamming, 1977).

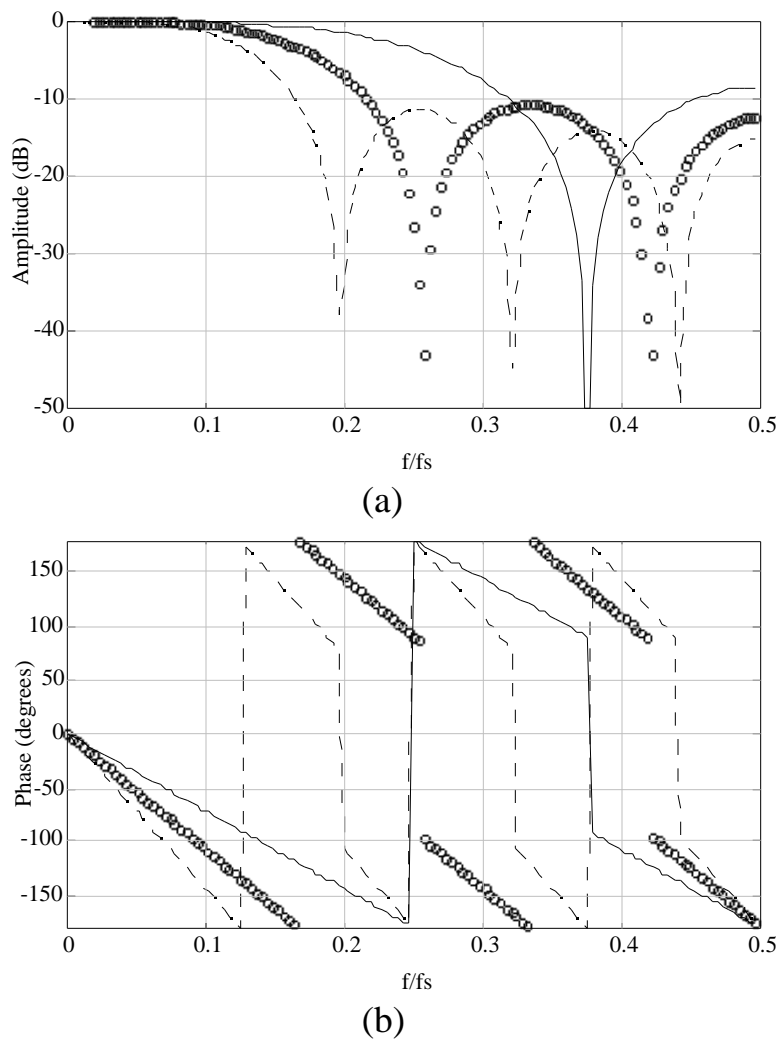


Figure 5.8 Polynomial smoothing filters (a) Amplitude responses. (b) Phase responses. Solid line: $L = 2$. Circles: $L = 3$. Dashed line: $L = 4$.

5.3 NOTCH FILTERS

A common biomedical signal processing problem involves the removal of noise of a particular frequency or frequency range (such as 60 Hz) from a signal while passing higher and/or lower frequencies without attenuation. A filter that performs this task is referred to as a notch, bandstop, or band-reject filter.

One simple method of completely removing noise of a specific frequency from the signal is to place a zero on the unit circle at the location corresponding to that frequency. For example, if a sampling rate of 180 samples per second is used, a zero at $2\pi/3$ removes 60-Hz line frequency noise from the signal. The difference equation is

$$y(nT) = \frac{1}{3} [x(nT) + x(nT - T) + x(nT - 2T)] \quad (5.19)$$

The filter has zeros at

$$z = -0.5 \pm j 0.866 \quad (5.20)$$

and its amplitude and phase response are given by

$$|H(\omega T)| = \frac{1}{3} [1 + 2\cos(\omega T)] \quad (5.21)$$

$$H(\omega T) = -\omega T \quad (5.22)$$

Figure 5.9 shows the details of the design and performance of this filter. The relatively slow rolloff of this filter causes significant attenuation of frequencies other than 60 Hz as well.

5.4 DERIVATIVES

The response of the true derivative function increases linearly with frequency. However, for digital differentiation such a response is not possible since the frequency response is periodic. The methods discussed in this section offer trade-offs between complexity of calculation, approximation to the true derivative, and elimination of high-frequency noise. Figure 5.10 shows the signal-flow graphs and pole-zero plots for three different differentiation algorithms: two-point difference, three-point central difference, and least-squares polynomial fit.

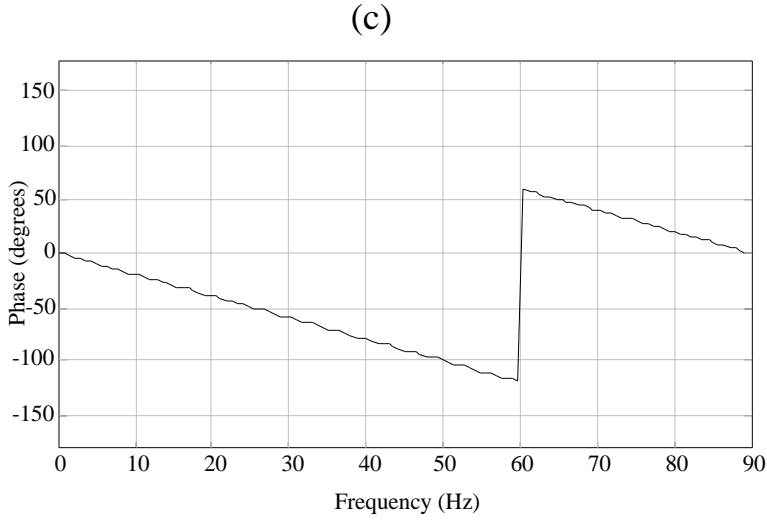
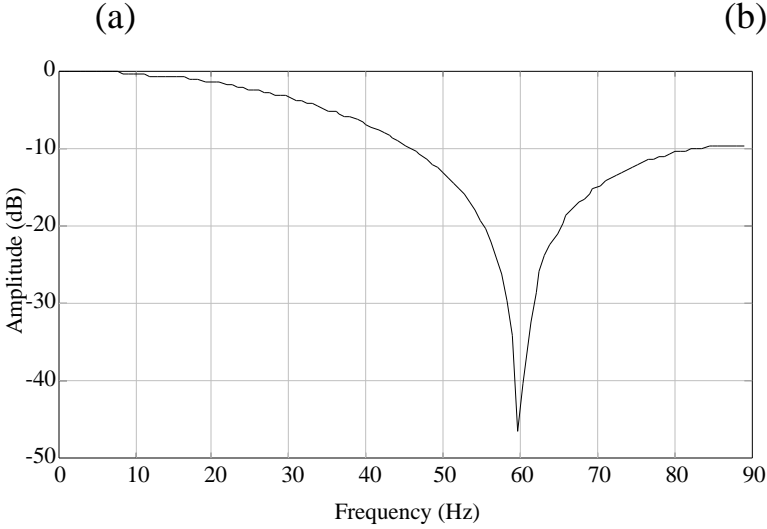
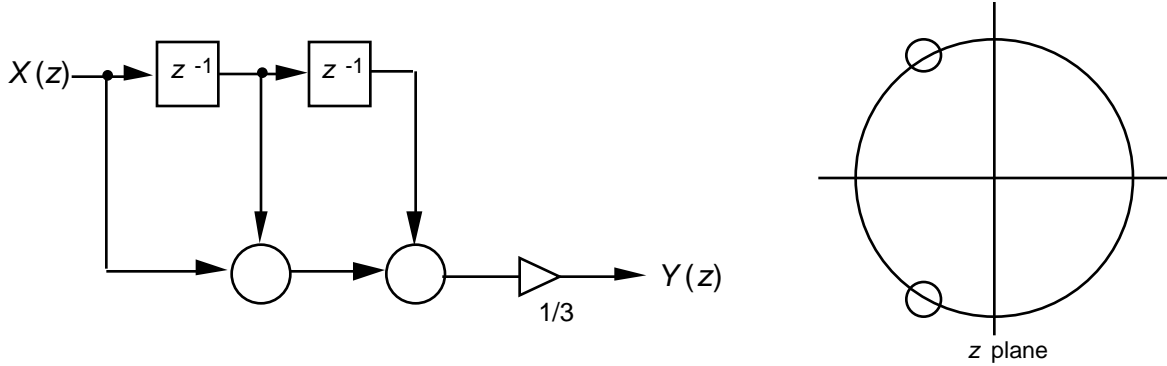


Figure 5.9 The 60-Hz notch filter. (a) Signal-flow graph. (b) Pole-zero plot. (c) Frequency response. (d) Phase response.

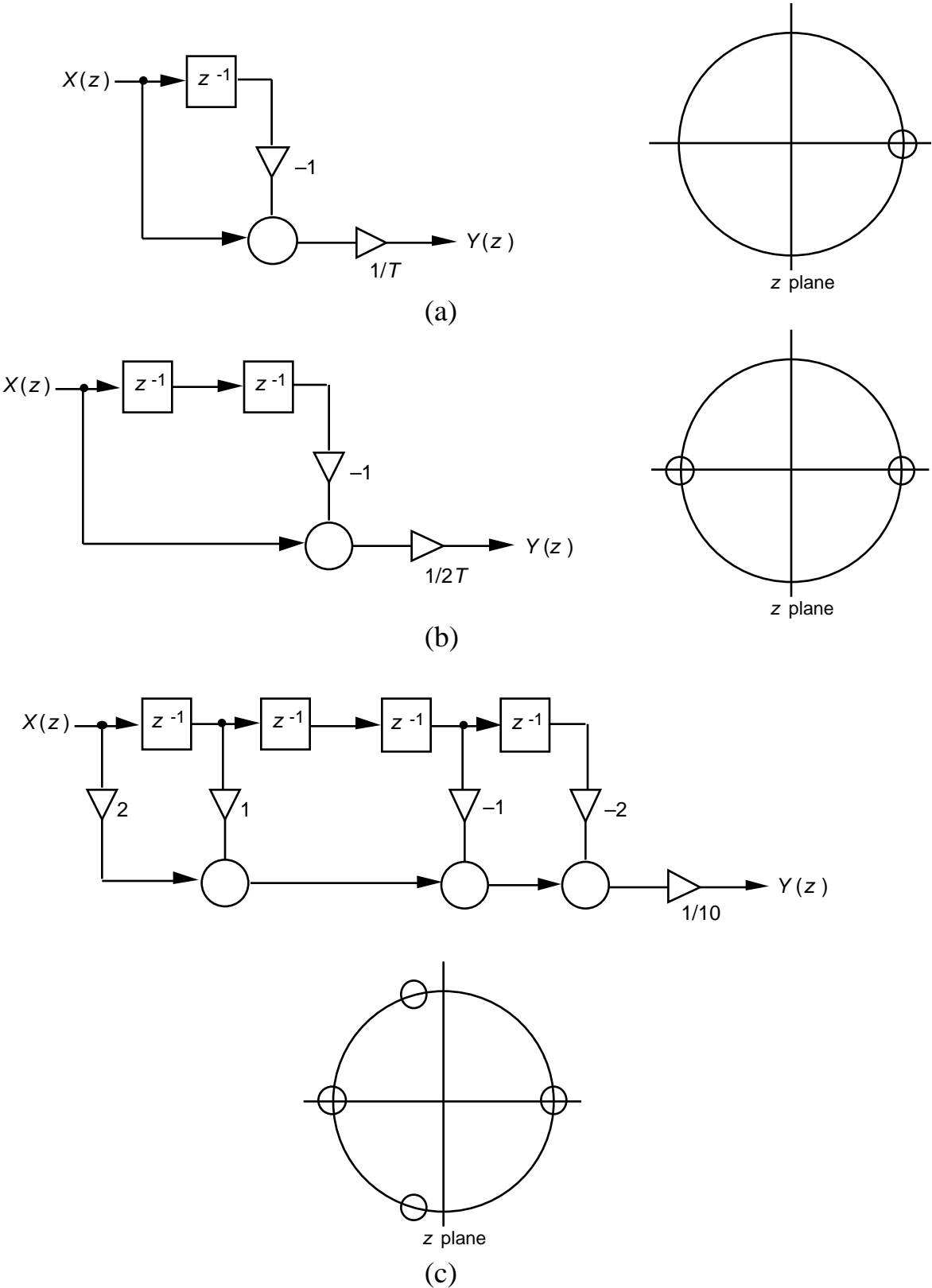


Figure 5.10 Signal flow graphs and pole-zero diagrams for derivative algorithms. (a) Two-point. (b) Three-point central difference. (c) 5-point least squares polynomial.

5.4.1 Two-point difference

The two-point difference algorithm, the simplest of these derivative algorithms, places a zero at $z = 1$ on the unit circle. Its amplitude response shown in Figure 5.11(a) closely approximates the ideal response, but since it does not go to zero at $f_s/2$, it greatly amplifies high-frequency noise. It is often followed by a low-pass filter. Its difference equation is

$$y(nT) = \frac{1}{T} [x(nT) - x(nT - T)] \quad (5.23)$$

Its transfer function is

$$H(z) = \frac{1}{T} (1 - z^{-1}) \quad (5.24)$$

5.4.2 Three-point central difference

The three-point central difference algorithm places zeros at $z = 1$ and $z = -1$, so the approximation to the derivative is poor above $f_s/10$ seen in Figure 5.11(a). However, since the response goes to zero at $f_s/2$, the filter has some built-in smoothing. Its difference equation is

$$y(nT) = \frac{1}{2T} [x(nT) - x(nT - 2T)] \quad (5.25)$$

Its transfer function is

$$H(z) = \frac{1}{2T} (1 - z^{-2}) \quad (5.26)$$

5.4.3 Least-squares polynomial derivative approximation

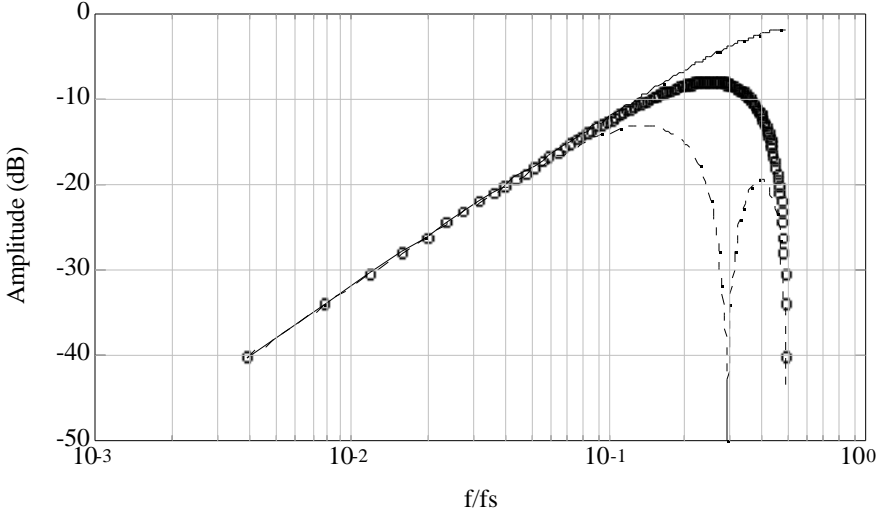
This filter is similar to the parabolic smoothing filter described earlier, except that the slope of the polynomial is taken at the center of the parabola as the value of the derivative. The coefficients of the transfer equations for filters with $L = 2, 3, 4$, and 5 are illustrated in Figure 5.12. Figure 5.10(c) shows the signal-flow graph and pole-zero diagram for this filter with $L = 2$. Note that the filter has zeros at $z = \pm 1$, as did the three-point central difference, with additional zeros at $z = -0.25 \pm j0.968$. The difference equation for the five-point parabolic filter is

$$y(nT) = \frac{1}{10T} [2x(nT) + x(nT - T) - x(nT - 3T) - 2x(nT - 4T)] \quad (5.27)$$

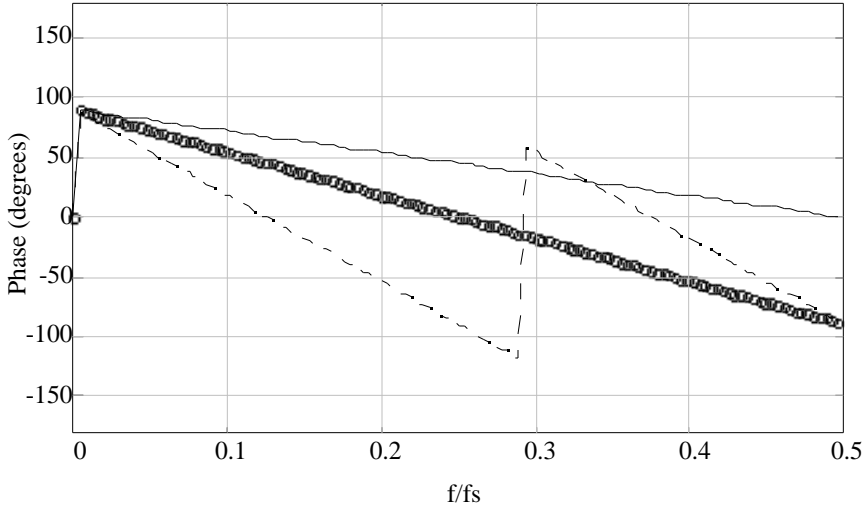
Its transfer function is

$$H(z) = \frac{1}{10T} [2 + z^{-1} - z^{-3} - 2z^{-4}] \quad (5.28)$$

As Figure 5.11(a) shows, the response only approximates the true derivative at low frequencies, since the smoothing nature of the parabolic fit attenuates high frequencies significantly.



(a)



(b)

Figure 5.11 Derivatives. (a) Amplitude response. (b) Phase response. Solid line: Two-point. Circles: Three-point central difference. Dashed line: Least-squares parabolic approximation for $L = 2$.

L	Tap weights
2	$\frac{1}{10T} (2, 1, 0, -1, -2)$
3	$\frac{1}{28T} (3, 2, 1, 0, -1, -2, -3)$
4	$\frac{1}{60T} (4, 3, 2, 1, 0, -1, -2, -3, -4)$
5	$\frac{1}{110T} (5, 4, 3, 2, 1, 0, -1, -2, -3, -4, -5)$

Figure 5.12 Least-squares derivative approximation coefficients for $L = 2, 3, 4,$ and 5 .

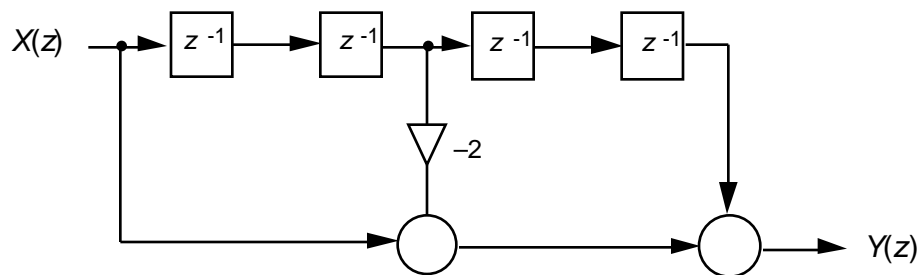
5.4.4 Second derivative

Figure 5.13 shows a simple filter for approximating the second derivative (Friesen et al., 1990), which has the difference equation

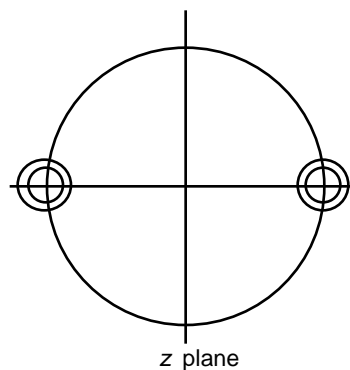
$$y(nT) = x(nT) - 2x(nT - 2T) + x(nT - 4T) \quad (5.29)$$

This filter was derived by cascading two stages of the three-point central difference derivative of Eq. (5.26) and setting the amplitude multiplier to unity to obtain the transfer function

$$H(z) = (1 - z^{-2}) \times (1 - z^{-2}) = (1 - 2z^{-2} - z^{-4}) \quad (5.30)$$



(a)



(b)

Figure 5.13 Second derivative. (a) Signal-flow graph. (b) Unit-circle diagram.

5.5 WINDOW DESIGN

A desired frequency response $H_d(\theta)$ (a continuous function) has as its inverse discrete-time Fourier transform (IDTFT), which is the desired unit pulse sequence, $h_d(k)$ (a discrete function). This sequence will have an infinite number of terms, so it is not physically realizable. The objective of window design is to choose an actual $h(k)$ with a finite number of terms such that the frequency response $H(e^{j\theta})$ will be in some sense close to $H_d(\theta)$. If the objective is to minimize the mean-squared error between the actual frequency response and the desired frequency response, then it can be shown by Parseval's theorem that the error is minimized by directly truncating $h_d(k)$. In other words, the pulse response $h(k)$ is chosen to be the first N terms of $h_d(k)$. Unfortunately, such a truncation results in large overshoots at sharp transitions in the frequency response, referred to as *Gibb's phenomenon*, illustrated in Figure 5.14.

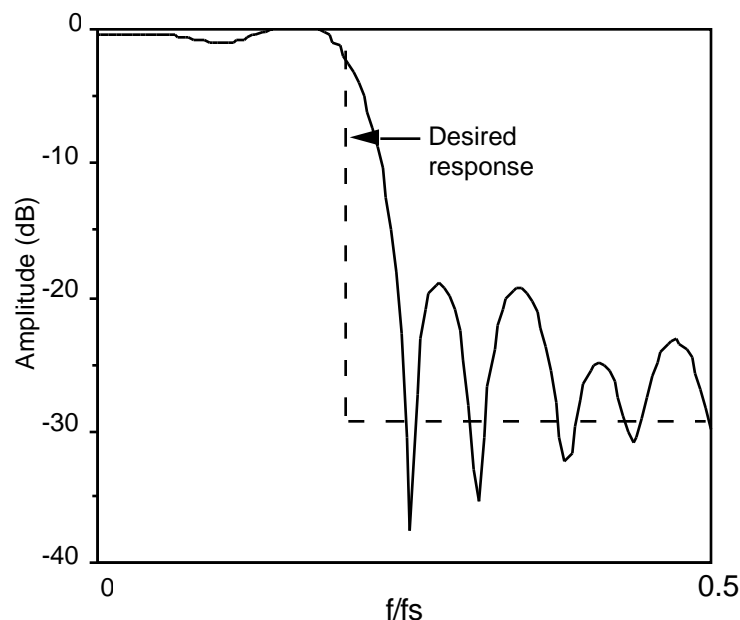


Figure 5.14 The overshoot that occurs at sharp transitions in the desired frequency response due to truncation of the pulse response is referred to as Gibbs's phenomenon.

To understand this effect, consider that direct truncation of $h_d(k)$ is a multiplication of the desired unit pulse sequence by a rectangular window $w_R(k)$. Since multiplication in the time domain corresponds to convolution in the frequency domain, the frequency response of the resulting $h(k)$ is the convolution of the desired frequency response with the frequency response of the window function.

For example, consider that the frequency response of a rectangular window of infinite length is simply a unit pulse. The convolution of the desired frequency response with the unit pulse simply returns the desired response. However, as the width of the window decreases, its frequency response becomes less like an impulse function and sidelobes become more evident. The convolution of these sidelobes with the desired frequency response results in the overshoots at the transitions.

For a rectangular window, the response function $W_R(e^{j\theta})$ is given by

$$W_R(e^{j\theta}) = \frac{\sin \frac{N\theta}{2}}{\theta \sin \frac{\theta}{2}} \quad (5.31)$$

where N is the length of the rectangular window. There are two important results of the convolution of the window with the desired frequency response. First the window function *smears* the desired response at transitions. This smearing of the transition bands increases the width of the transition from passband to stopband. This has the effect of increasing the width of the *main lobe* of the filter. Second, windowing causes undesirable ripple called *window leakage* or *sideband ripple* in the stopbands of the filter.

By using window functions other than a rectangular window, stopband ripple can be reduced at the expense of increasing main lobe width. Many types of windows have been studied in detail, including triangular (Bartlett), Hamming, Hanning, Blackman, Kaiser, Chebyshev, and Gaussian windows. Hanning and Hamming windows are of the form

$$w_H(k) = w_R(k) \left[\alpha + (1 - \alpha) \cos \frac{2}{N} k \right] \quad \text{for } 0 < \alpha < 1 \quad (5.32)$$

where $\alpha = 0.54$ for the Hamming window and $\alpha = 0.50$ for the Hanning window.

The Kaiser window is of the form

$$w_K(k) = w_R(k) I_0 \frac{\alpha \sqrt{1 - \frac{k^2}{M}}}{I_0(\alpha)} \quad (5.33)$$

where α allows the designer to choose main lobe widths from the extreme minimum of the rectangular window to the width of the Blackman window, trading off sidelobe amplitude (Antoniou, 1979). See Roberts and Mullis (1987) for a discussion of the Chebyshev and Gaussian windows. All of these window functions taper to zero at the edges of the window. Figure 5.15 compares the performance of several of these windows.

Window Type	Sideband Ripple (dB)	Main Lobe Width
Rectangular	-13	4 /N
Triangular	-25	8 /N
Hanning	-31	8 /N
Hamming	-41	8 /N
Blackman	-38	12 /N

Figure 5.15 Responses of various windows.

Window design is usually performed iteratively, since it is difficult to predict the extent to which the transition band of the window will smear the frequency response. To design an FIR filter to meet a specific frequency response, one computes the unit pulse sequence of the desired response by taking the IDTFT, applies the window, and then computes the actual response by taking the DFT. The band edges or filter order are adjusted as necessary, and the process is repeated. Many algorithms have been developed to perform this process by computer. The technique can easily be constrained to generate filters with linear phase responses. Window designs do not generally yield the lowest possible order filter to meet the specifications. Windowing is also discussed in Chapter 11.

5.6 FREQUENCY SAMPLING

The frequency sampling method of design is more straightforward than the window design method since it circumvents the transformations from the time domain to the frequency domain. The terms in the filter response $H(e^{j\theta})$ are directly specified to match $H_d(\theta)$ at N uniformly spaced frequencies around the unit circle. As in the window design method, large overshoots will occur at sharp transitions in the response. This overshoot can be minimized by allowing some unconstrained terms in the transition band. Figure 5.16 illustrates a frequency sampling design with the same desired response and number of terms as in Figure 5.14, but with one unconstrained value chosen to minimize stopband ripple. The unconstrained values can be chosen to minimize some measure of error between $H(e^{j\theta})$ and $H_d(\theta)$. The frequency sampling method generally yields more efficient filters than the window method.

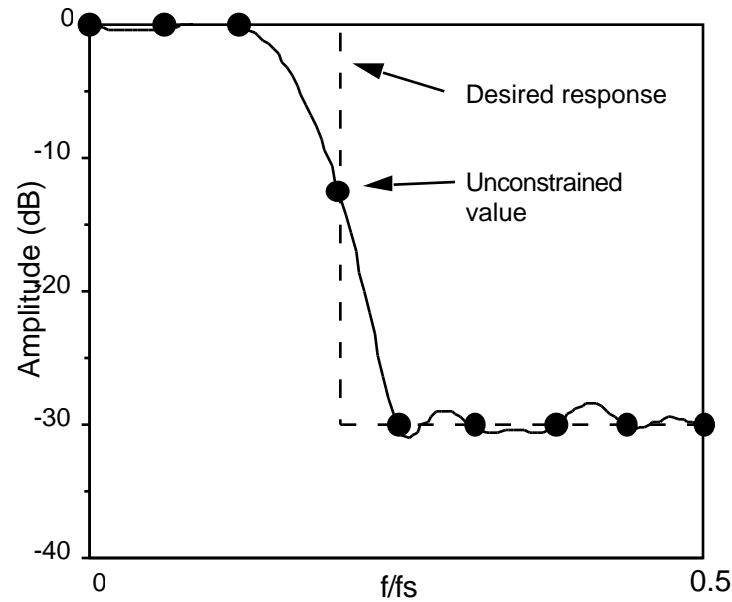


Figure 5.16 Frequency sampling design for $N = 16$ with one unconstrained term.

5.7 MINIMAX DESIGN

The window design method generates large errors at sharp transitions in the frequency response. Rather than minimizing the energy in the error

$$\epsilon^2 = \frac{1}{2} \int_0^{2\pi} |H_d(\theta) - H_1(e^{j\theta})|^2 d\theta \quad (5.34)$$

As in the window design method, it may be more desirable in certain applications to minimize the maximum error

$$\max_{\theta} |H_d(\theta) - H_1(e^{j\theta})| \quad (5.35)$$

The idea is to spread the error out evenly or in the more general case, in some weighted fashion across the frequency response. This is much more difficult than the window design problem, but algorithms have been developed to solve it by computer. The Remez algorithm, for example, allows the designer to select weighting factors throughout the passband and stopbands (Roberts and Mullis, 1987).

5.8 LAB: FIR FILTER DESIGN

The UW DigiScope software allows the user to experiment with several FIR design techniques, including direct specification of the pulse response, frequency sampling, window design, and placement of the zeros of the transfer equation. This lab studies some basic FIR filters such as the Hanning filter. It also uses specification of the pulse response and placement of the zeros of the transfer equation as design tools. Refer to Appendix D for information about using UW DigiScope.

5.8.1 Smoothing filters

Generate an ECG file with some random noise with the **(G)enwave** function, then run the Hanning filter by selecting **(F)ilters**, then **(L)oad filter**, then choosing **hanning.fil**. This function loads and immediately executes the filter function. By choosing **(R)un filter**, you can see the effect of cascading another Hanning filter with the first. The filter always operates on the data in the bottom channel.

The filters **poly2.fil**, **poly3.fil**, and **poly4.fil** are the least-squares smoothing filters with $L = 2, 3,$ and 4 respectively. To test each of these three filters on the ECG data, first move the original signal from the top channel to the bottom channel with the **(C)opy data** command. Then use **(L)oad filter** to load and run the filters on ECG data. Observe the frequency and phase responses of each filter. How does changing the sampling rate of the ECG data affect the performance of the filters? Load **poly4.fil** and measure the time difference between the peak of the QRS complex in the unfiltered data and in the filtered data using the **(M)easure** command. You will need to change channels using **(A)ctive channel** to make the measurements. How do you account for the delay?

5.8.2 Derivatives

Experiment with the various derivative filters **deriv2.fil**, **deriv3.fil**, and **deriv5.fil** (which are the two-point, three-point central limit, and least-squares with $L = 2$ derivative filters respectively) on ECG and square wave data. Which filter is least suited for use with noisy signals?

5.8.3 Pulse Response

Create a filter that calculates the second derivative by selecting **(F)ilters**, then **(D)esign**, then **(F)IR**. Choose **(P)ulse resp**, and specify a filter length equal to **5**. Enter the appropriate coefficients for the transfer equation and observe the frequency response. After saving the filter, return to the main menu, create a triangular wave with the **(G)enwave** function, and then use **(R)un filter** to observe the effects of the filter.

Create a linear phase filter with odd length (even order) and odd symmetry by satisfying Eq. (5.4). Observe its phase response.

5.8.4 Zero placement

The user may arbitrarily place zeros around the unit circle. The program automatically creates complex-conjugate pairs for values off the real axis. It then calculates the frequency, phase, and unit impulse responses for the filter.

Place zeros at $z = -0.5 \pm j0.866$ using the **(z)ero place** function for a second-order filter, and comment on the frequency response. Run this filter on ECG data with and without 60-Hz noise sampled at 180 samples/s, and summarize the results.

5.8.5 Unit pulse sequence

There is a data file in the **STDLIB** called **ups.dat**. It consists of a single pulse. Reading this file and running an FIR filter will generate the unit impulse output sequence of the filter. Executing **(P)wr Spect** on this resulting output signal will give the magnitude response of the filter. In this way, the frequency response of filter can be measured more accurately than by the graph generated in the filter design utility. Use this method to find the 3-dB frequency of the Hanning filter (i.e., file **hanning.fi1**).

5.8.6 Frequency sampling

Study the effect of increasing the width of the transition band by comparing the frequency response FIR filters of length 13 designed using frequency sampling by entering the following sets of values for the responses around the unit circle: $\{0, 0, -40, -40, -40, -40, -40\}$ and $\{0, 0, -6, -40, -40, -40, -40\}$. Optimize the transition value (the third term in the sequence) for minimum stopband ripple.

Design a 60-Hz notch filter with length 21 by using frequency sampling. Run the filter on ECG data with and without 60-Hz noise. Does the filter perform better than **notch60.fi1**? Optimize the filter for minimal gain at 60 Hz and minimal attenuation of other frequencies.

5.8.7 Window design

Compare the frequency response of low-pass filters designed with the various windows of the **(w)indow** function. Design the filters to have a cutoff frequency at 25 Hz with a sampling rate of 200 Hz. Make a table comparing main lobe width versus sideband ripple for the various windows. Measure the main lobe width at the first minimum in the frequency response.

5.8.8 Linear versus piecewise-linear phase response

Compare the phase response of the Hanning filter with that of the least-squares derivative filter with $L = 2$ (**deriv5.fi1**). Why is the phase response of the least-

squares filter not true linear? What causes the discontinuity? See Chapter 7 for additional information.

5.9 REFERENCES

- Antoniou, A. 1979. *Digital Filters: Analysis and Design*. New York: McGraw-Hill.
 Bogner, R. E. and Constantinides, A. G. 1985. *Introduction to Digital Filtering*, New York: John Wiley and Sons.
 Friesen, G. M. et al. 1990. A comparison of the noise sensitivity of nine QRS detection algorithms, *IEEE Trans. Biomed. Eng.*, **BME-37**(1): 85–98.
 Hamming, R. W. 1977. *Digital Filters*. Englewood Cliffs, NJ: Prentice Hall.
 Rabiner, L. R. and Rader, C. M. 1972. *Digital Signal Processing* New York: IEEE Press.
 Roberts, R. A. and Mullis, R. T. 1987. *Digital Signal Processing*. Reading, MA: Addison-Wesley.

5.10 STUDY QUESTIONS

- 5.1 What are the main differences between FIR and IIR filters?
- 5.2 What is the difference between direct convolution and “fast convolution”?
- 5.3 Why are finite-length register effects less significant in FIR filters than in IIR filters?
- 5.4 Compute and sketch the frequency response of a cascade of two Hanning filters. Does the cascade have linear phase?
- 5.5 Derive the phase response for an FIR filter with zeros located at $r \pm \theta$ and $r^{-1} \pm \theta$. Comment.
- 5.6 What are the trade-offs to consider when choosing the order of a least-squares polynomial smoothing filter?
- 5.7 Complete the derivation of coefficient values for the parabolic smoothing filter for $L = 2$.
- 5.8 Give some of the disadvantages of the simple 60-Hz notch filter described in section 5.3.1.
- 5.9 What are the main differences between the two-point difference and three-point central difference algorithms for approximating the derivative?
- 5.10 What are the three steps to designing a filter using the window method?
- 5.11 Explain the relationship between main-lobe width and sideband ripple for various windows.
- 5.12 How do the free parameters in the transition band of a frequency sampling design affect the performance of the filter?
- 5.13 What is the fundamental difference between minimax design and the window design method?
- 5.14 Design an FIR filter of length 15 with passband gain of 0 dB from 0 to 50 Hz and stopband attenuation of 40 dB from 100 to 200 Hz using the window design method. Compare the Hanning and rectangular windows. (Use a sampling rate of 400 sps.)
- 5.15 Repeat question 5.14 using frequency sampling.
- 5.16 The transfer function of the Hanning filter is

$$H_1(z) = \frac{1 + 2z^{-1} + z^{-2}}{4}$$

- (a) What is its gain at dc ?
- (b) Three successive stages of this filter are cascaded together to give a new transfer function [that is, $H(z) = H_1(z) \times H_1(z) \times H_1(z)$]. What is the overall gain of this filter at dc ?
- (c) A high-pass filter is designed by subtracting the output of the

Hanning filter from an all-pass filter with zero phase delay. How many zeros does the resulting filter have? Where are they located?

- 5.17 Two filters are cascaded. The first has the transfer function: $H_1(z) = 1 + 2z^{-1} - 3z^{-2}$. The second has the transfer function: $H_2(z) = 1 - 2z^{-1}$. A *unit impulse* is applied to the input of the cascaded filters. (a) What is the output sequence? (b) What is the magnitude of the amplitude response of the cascaded filter 1. at dc? 2. at 1/2 the foldover frequency? 3. at the foldover frequency?
- 5.18 Two filters are cascaded. The first has the transfer function: $H_1(z) = 1 + 2z^{-1} + z^{-2}$. The second has the transfer function: $H_2(z) = 1 - z^{-1}$. (a) A *unit impulse* is applied to the input of the cascaded filters. What is the output sequence? (b) What is the magnitude of the amplitude response of this cascaded filter at dc?