Chapter

15

IIR Filters via the Bilinear Transformation

A popular technique for the design of IIR digital filters is the *bilinear transformation method*, which offers several advantages over the other techniques presented in the previous chapter.

15.1 Bilinear Transformation

The bilinear transformation converts the transfer function for an analog filter into the system function for a digital filter by making the substitution

$$s \rightarrow \frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}}$$

If the analog prototype filter is stable, the bilinear transformation will result in a stable digital filter.

Algorithm 15.1 Bilinear transformation

Step 1. Obtain the transfer function $H_a(s)$ for the desired analog prototype filter.

Step 2. In the transfer function obtained in step 1, make the substitution

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

where T is the sampling interval of the digital filter. Call the resulting digital system function $H(z)$.

Step 3. The analog prototype filter's transfer function $H_a(s)$ will, in general, be a ratio of polynomials in s. Therefore, the system function $H(z)$ obtained

in step 2 will, in general, contain various powers of the ratio $(1-z^{-1})/(1+z^{-1})$ in both the numerator and the denominator. Multiply both the numerator and denominator by the highest power of $1+z^{-1}$, and collect terms to obtain $H(z)$ as a ratio of polynomials in z^{-1} of the form

$$H(z)=\frac{\sum_{k=0}^{M} b_k z^{-k}}{1-\sum_{k=1}^{N} a_k z^{-k}} \tag{15.1}$$

Step 4. Use the a_k and b_k obtained in step 3 to realize the filter in any of the structures given in Sec. 14.1.

Example 15.1 Use the bilinear transform to obtain an IIR filter from a second-order Butterworth analog filter with a 3-dB cutoff frequency of 3 KHz. The sampling rate for the digital filter is 30,000 samples per second.

solution The analog prototype filter's transfer function is given by

$$H_a(s)=\frac{\omega_c^2}{s^2+\sqrt{2}\omega_c s+\omega_c^2}$$

where $\omega_c=6000\pi$. Making the substitution $s=2(1-z^{-1})/(T(1+z^{-1}))$ yields

$$H(z)=\frac{\omega_c^2}{\left(\frac{2}{T}\right)^2\left(\frac{1-z^{-1}}{1+z^{-1}}\right)^2+\sqrt{2}\omega_c\left(\frac{2}{T}\right)\left(\frac{1-z^{-1}}{1+z^{-1}}\right)+\omega_c^2}$$

where $T=1/30{,}000$. After the appropriate algebraic simplifications and making use of the fact that

$$\omega_c T=\frac{6000\pi}{30{,}000}=\frac{\pi}{5}$$

we obtain the desired form of $H(z)$ as

$$H(z)=\frac{0.063964+0.127929z^{-1}+0.063964z^{-2}}{1-1.168261z^{-1}+0.424118z^{-2}} \tag{15.2}$$

Comparison of (15.1) and (15.2) reveals that

$$a_1=-1.168261 \qquad a_2=0.424118$$

$$b_0=0.063964 \qquad b_1=0.127929 \qquad b_2=0.063964$$

15.2 Factored Form of the Bilinear Transformation

Often an analog prototype filter will be specified in terms of its poles and zeros—that is, the numerator and denominator of the filter's transfer function will be in factored form. The bilinear transformation can be applied directly to this factored form. An additional benefit of this approach is that the process of finding the *digital* filter's poles and zeros is greatly simplified. Each factor in the numerator of the analog filter's transfer function will be of the form $(s-q_n)$, and each factor of the denominator will be of the form

$(s - p_n)$, where q_n and p_n are, respectively, the nth zero and nth pole of the filter. When the bilinear transform is applied, the corresponding factors become

$$\left(\frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}} - q_n\right) \quad \text{and} \quad \left(\frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}} - p_n\right)$$

The zeros of the digital filter are obtained by finding the values of z for which

$$\frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}} - q_n = 0$$

The desired values of z are given by

$$z_z = \frac{2+q_nT}{2-q_nT} \tag{15.3}$$

In a similar fashion, the poles of the digital filter are obtained from the poles of the analog filter using

$$z_p = \frac{2+p_nT}{2-p_nT} \tag{15.4}$$

The use of (15.3) and (15.4) is straightforward for the analog filter's *finite* poles or zeros. Usually, only the finite poles and zeros of a filter are considered, but in the present context, *all* poles and zeros of the analog filter must be considered. The analog filter's infinite zeros will map into zeros of $z = -1$ for the digital filter.

Algorithm 15.2 Bilinear transformation for transfer functions in factored form

Step 1. For the desired analog prototype filter, obtain the transfer function $H_a(s)$ in the factored form given by

$$H_a(s) = H_0\frac{\prod_{m=1}^{M}(s-q_m)}{\prod_{n=1}^{N}(s-p_n)}$$

Step 2. Obtain the poles z_{pn} of the analog filter from the poles p_n of the analog filter using

$$z_{pn} = \frac{2+p_nT}{2-p_nT} \qquad n = 1, 2, \ldots, N$$

Step 3. Obtain the zeros z_{zm} of the digital filter from the zeros q_m of the analog filter using

$$z_{zm} = \frac{2+q_mT}{2-q_mT} \qquad m = 1, 2, \ldots, M$$

Step 4. Using the values of z_{pn} obtained in step 2 and the values of z_{zm} obtained in step 3, form $H(z)$ as

$$H(z) = H_0 \frac{T^N}{\prod_{n=1}^{N}(2 - p_n T)} \cdot \frac{(z+1)^{N-M}\prod_{m=1}^{M}(z - z_{zm})}{\prod_{n=1}^{N}(z - z_{pn})} \tag{15.5}$$

The factor $(z+1)^{N-M}$ supplies the zeros at $z = -1$, which correspond to the zeros at $s = \infty$ for analog filter's having $M < N$. The first rational factor in Eq. (15.5) is a constant gain factor that is needed to obtain results which exactly match the results obtained via Algorithm 15.1. However, in practice, this factor is often omitted to yield

$$H(z) = H_0 \frac{(z+1)^{N-M}\prod_{m=1}^{M}(z - z_{zm})}{\prod_{n=1}^{N}(z - z_{pn})}$$

Example 15.2 The Butterworth filter of Example 15.1 has a transfer function given in factored form as

$$H_a(s) = \frac{\omega_c^2}{[s + \omega_c(\sqrt{2}/2) - j\omega_c(\sqrt{2}/2)][s + \omega_c(\sqrt{2}/2) + j\omega_c(\sqrt{2}/2)]}$$

Apply the bilinear transform to this factored form to obtain the IIR filter's system function $H(z)$.

solution The analog filter has poles at

$$s = \omega_c \frac{-\sqrt{2}}{2} \pm j\omega_c \frac{\sqrt{2}}{2}$$

Using (15.4), we then obtain the poles of the digital filter as

$$z_{P_1} = \frac{2 + \left(\frac{-\sqrt{2}}{2} + j\frac{\sqrt{2}}{2}\right)\omega_c T}{2 - \left(\frac{-\sqrt{2}}{2} + j\frac{\sqrt{2}}{2}\right)\omega_c T}$$
$$= 0.584131 + 0.28794j$$
$$z_{P_2} = \frac{2 + \left(\frac{-\sqrt{2}}{2} - j\frac{\sqrt{2}}{2}\right)\omega_c T}{2 - \left(\frac{-\sqrt{2}}{2} - j\frac{\sqrt{2}}{2}\right)\omega_c T}$$
$$= 0.584131 - 0.287941j$$

The two zeros at $s = \infty$ map into two zeros at $z = -1$. Thus the system function is given by

$$H(z) = H_c \frac{(z+1)^2}{(z - 0.584131 + 0.287941j)(z - 0.584131 - 0.287941j)}$$

$$\text{where } H_c = \frac{H_0 T^2}{(2 - p_1 T)(2 - p_2 T)}$$

$$= \frac{(6000\pi)^2}{(30{,}000)^2\left(2 + \frac{\pi\sqrt{2}}{10} - j\frac{\pi\sqrt{2}}{10}\right)\left(2 + \frac{\pi\sqrt{2}}{10} + j\frac{\pi\sqrt{2}}{10}\right)}$$

$$= 0.063964$$

If the numerator and denominator factors are multiplied out and all terms are divided by z^2, we obtain

$$H(z) = \frac{0.063964(1 + 2z^{-1} + z^{-2})}{1 - 1.168261z^{-1} + 0.424118z^{-2}} \tag{15.6}$$

which matches the result of Example 15.1.

15.3 Properties of the Bilinear Transformation

Assume that the analog prototype filter has a pole at $s_P = \sigma + j\omega$. The corresponding IIR filter designed via the bilinear transformation will have a pole at

$$z_P = \frac{2 + sT}{2 - sT}$$

$$= \frac{2 + (\sigma + j\omega)T}{2 - (\sigma + j\omega)T}$$

$$= \frac{2 + \sigma T + j\omega T}{2 - \sigma T - j\omega T}$$

The magnitude and angle of this pole are given by

$$|z_P| = \sqrt{\frac{(2 + \sigma T)^2 + (\omega T)^2}{(2 - \sigma T)^2 + (\omega T)^2}} \tag{15.7}$$

$$\arg(z_P) = \tan^{-1}\left(\frac{\omega T}{2 + \sigma T}\right) - \tan^{-1}\left(\frac{-\omega T}{2 - \sigma T}\right)$$

The poles of a stable analog filter must lie in the left half of the s plane—that is, $\sigma < 0$. When $\sigma < 0$, the numerator of (15.7) will be smaller than the denominator, and thus $|z_P| < 1$. This means that analog poles in the left half of the s plane map into digital poles inside the unit circle of the z plane—stable analog poles map into stable digital poles. Poles that lie on the $j\omega$ axis of the s plane have $\sigma = 0$ and consequently map into z-plane poles which have unity magnitude and hence lie on the unit circle. Analog poles at $s = 0$ map into digital poles at $z = 1$, and analog poles at $s = \pm j\infty$ map into digital poles at $z = -1$.

Frequency warping

The mapping of the s plane's $j\omega$ axis into the z plane's unit circle is a highly nonlinear mapping. The analog frequency ω_0 can range from $-\infty$ to $+\infty$, but the digital frequency ω_d is limited to the range $\pm\pi$. The relationship between ω_a and ω_d is given by

$$\omega_d = 2\tan^{-1}\frac{\omega_a T}{2} \tag{15.8}$$

If an analog prototype filter with a cutoff frequency of ω_a is used to design a filter via the bilinear transformation, the resulting digital filter will have a cutoff frequency of ω_d, where ω_d is related to ω_a via (15.8).

Example 15.3 A lowpass filter with a 3-dB frequency of 3 kHz is used as the prototype for an IIR filter with a sampling rate of 30,000 samples per second. What will be the 3-dB frequency of the digital filter designed via the bilinear transformation?

solution Equation (15.8) yields

$$\omega_d = 2\tan^{-1}\frac{(6000\pi)(1/30{,}000)}{2}$$

$$= 0.6088$$

Since $\omega_d = \pi$ corresponds to a frequency of 30,000/2 = 15,000 Hz, the cutoff frequency of the filter is given by

$$\omega_c = \frac{0.6088}{\pi}(15{,}000) = 2906.8 \text{ Hz}$$

The frequency-warping effects become more severe as the frequency of interest increases relative to the digital filter's sampling rate.

Example 15.4 Consider the case of an analog filter with a 3-dB frequency of 3 kHz used as the prototype for an IIR filter designed via the bilinear transformation. Determine the impact on the 3-dB frequency if the sampling rate is changed from 10,000 samples per second to 30,000 samples per second in steps of 1000 samples per second.

solution The various sampling rates and the corresponding warped 3-dB frequencies are listed in Table 15.1.

Fortunately, it is a simple matter to counteract the effects of frequency warping by prewarping the critical frequencies of the analog prototype filter in such a way that the warping caused by the bilinear transformation restores the critical frequencies to their original intended values. Equation (15.8) can be inverted to yield the equation needed for this prewarping:

$$\omega_a = \frac{2}{T}\tan\frac{\omega_d}{2} \tag{15.9}$$

Example 15.5 We wish to design an IIR filter with a 3-dB frequency of 3 kHz and a sampling rate of 30,000 samples per second. Determine the prewarped 3-dB frequency required for the analog prototype filter.

TABLE 15.1 Warped Cutoff Frequencies for Example 15.4

Sampling rate	Cutoff frequency, Hz	% error
10,000	2405.8	−19.81
11,000	2480.5	−17.32
12,000	2543.1	−15.23
13,000	2595.8	−13.47
14,000	2640.4	−11.99
15,000	2678.5	−10.72
16,000	2711.1	−9.63
17,000	2739.3	−8.69
18,000	2763.6	−7.88
19,000	2784.9	−7.17
20,000	2803.5	−6.55
21,000	2819.9	−6.00
22,000	2834.4	−5.52
23,000	2847.2	−5.09
24,000	2858.7	−4.71
25,000	2868.9	−4.37
26,000	2878.1	−4.06
27,000	2886.4	−3.79
28,000	2893.8	−3.54
29,000	2900.6	−3.31
30,000	2906.8	−3.11

solution Since $\omega_d = \pi$ corresponds to a frequency of 30,000/2 = 15,000 Hz, a frequency of 3 kHz corresponds to a ω_d of

$$\omega_d = \frac{3000\pi}{15{,}000} = \frac{\pi}{5}$$

The prototype analog frequency ω_a is obtained by using this value of ω_d in Eq. (15.9):

$$\omega_a = \frac{2}{(1/30{,}000)} \tan\frac{\pi}{10} = 19{,}495.18$$

The analog prototype filter must have a 3-dB frequency of $19{,}495.18/(2\pi) = 3102.75$ Hz in order for the IIR filter to have a 3-dB frequency of 3 kHz after warping.

15.4 Programming the Bilinear Transformation

Assume that the transfer function of the analog prototype filter is in the form given by

$$H_a(s) = H_0 \frac{\prod_{m=1}^{M} (s - q_m)}{\prod_{n=1}^{N} (s - p_n)}$$

where p_n and q_n denote, respectively, the filter's poles and zeros. To generate

a digital filter via the bilinear transformation, we make the substitution

$$s = \frac{2}{T}\left(\frac{1 - z^{-1}}{1 + z^{-1}}\right)$$

and obtain

$$H(z) = H_0 \frac{\prod_{m=1}^{M}\left[\frac{2}{T}\left(\frac{1 - z^{-1}}{1 + z^{-1}}\right) - q_m\right]}{\prod_{n=1}^{N}\left[\frac{2}{T}\left(\frac{1 - z^{-1}}{1 + z^{-1}}\right) - p_n\right]}$$

which, after some algebraic manipulation, can be put into the form

$$H(z) = H_0 \frac{(1 + z^{-1})^{N-M} \prod_{m=1}^{N}\left[\left(\frac{2}{T} - q_m\right) - \left(\frac{2}{T} + q_m\right)z^{-1}\right]}{\prod_{n=1}^{N}\left[\left(\frac{2}{T} - p_n\right) - \left(\frac{2}{T} + p_n\right)z^{-1}\right]}$$

Thus, the denominator of $H(z)$ is given by

$$D(z) = \prod_{n=1}^{N} (\gamma_n + \delta_n z^{-1}) \tag{15.10}$$

where $\gamma_n = \dfrac{2}{T} - p_n$

$$\delta_n = \frac{-2}{T} - p_n$$

To see how (15.10) can be easily evaluated by computer, let's examine the sequence of partial products $\{D_k(z)\}$ encountered in the evaluation:

$$D_1(z) = (\gamma_1 + \delta_1 z^{-1})$$

$$D_2(z) = (\gamma_2 + \delta_2 z^{-1})D_1(z) = \gamma_2 D_1(z) + \delta_2 z^{-1} D_1(z)$$

$$D_3(z) = (\gamma_3 + \delta_3 z^{-1})D_2(z) = \gamma_3 D_2(z) + \delta_3 z^{-1} D_2(z)$$

$$D_4(z) = (\gamma_4 + \delta_4 z^{-1})D_3(z) = \gamma_4 D_3(z) + \delta_4 z^{-1} D_3(z)$$

$$\vdots$$

$$D(z) = D_N(z) = (\gamma_N + \delta_N z^{-1})D_{N-1}(z) = \gamma_N D_{N-1}(z) + \delta_N z^{-1} D_{N-1}(z)$$

Examination of this sequence reveals that the partial product $D_k(z)$ at iteration k can be expanded in terms of the partial product $D_{k-1}(z)$ as

$$D_k(z) = \gamma_k D_{k-1}(z) + \delta_k z^{-1} D_{k-1}(z)$$

The partial product $D_{k-1}(z)$ will be a $(k-1)$-degree polynomial in z^{-1}:

$$D_{k-1}(z) = \mu_0(z^{-1})^0 + \mu_1(z^{-1})^1 + \mu_2(z^{-1})^2 + \cdots + \mu_{k-1}(z^{-1})^{k-1}$$

The products $\gamma_k D_{k-1}(z)$ and $\delta_k z^{-1} D_{k-1}(z)$ are then given by

$$\gamma_k D_{k-1}(z) = \gamma_k \mu_0 (z^{-1})^0 + \gamma_k \mu_1 (z^{-1})^1 + \gamma_k \mu_2 (z^{-1})^2 + \cdots + \gamma_k \mu_{k-1} (z^{-1})^{k-1}$$

$$\delta_k z^{-1} D_{k-1}(z) = \delta_k \mu_0 (z^{-1})^1 + \delta_k \mu_1 (z^{-1})^2 + \delta_k \mu_2 (z^{-1})^3 + \cdots + \delta_k \mu_{k-1} (z^{-1})^k$$

and $D_k(z)$ is given by

$$D_k(z) = \gamma_k \mu_0 (z^{-1})^0 + (\gamma_k \mu_1 - \delta_k \mu_0)(z^{-1})^1 + (\gamma_k \mu_2 - \delta_k \mu_1)(z^{-1})^2 + \cdots + (\gamma_k \mu_{k-1} - \delta_k \mu_{k-2})(z^{-1})^{k-1} - \delta_k \mu_{k-1} (z^{-1})^k$$

Therefore, we can conclude that if μ_n is the coefficient for the $(z^{-1})^n$ term in $D_{k-1}(z)$, then the coefficient for the $(z^{-1})^n$ term in $D(z)$ is $(\gamma_k \mu_n + \delta_k \mu_{n-1})$ with the proviso that $\mu \triangleq 0$ in $D_{k-1}(z)$. The polynomial $D_{k-1}(z)$ is represented in the computer as an array of k coefficients, with the array index corresponding to the subscript on μ and the superscript (exponent) on (z^{-1}). Thus, array element **mu[0]** contains μ_0, array element **mu[1]** contains μ_1, and so forth. The coefficients for the partial product $D_k(z)$ can be obtained from the coefficients for $D_{k-1}(z)$, as indicated by the following fragment of pseudocode:

```
for( j=k; j >= 1; j--)
    {mu[j] = gamma * mu[j] + beta * mu[j-1];}
```

The loop is executed in reverse order so that the coefficients can be updated "in place" without prematurely overwriting the old values. Notice that I referred to the fragment shown above as "pseudocode." In actuality, **mu[ ]**, **gamma**, and **delta** are each complex valued; and the arithmetic operations shown in the fragment are incorrect. The following code fragment performs the complex arithmetic correctly, but all the complex functions tend to obscure the algorithm which is more clearly conveyed by the pseudocode above:

```
for( j=k; j >= 1; j--)
    {mu[j] = cSub(cMult(gamma, mu[j]), cMult(delta, mu[j-1]));}
```

If this fragment is placed within an outer loop with k ranging from 1 to **numPoles**, the final values in **mu[n]** will be the coefficients a_n for Eq. (15.1).

A similar loop can be developed for the numerator product $N(z)$ given by

$$N(z) = \prod_{m=1}^{M} (\alpha_m - \beta_m z^{-1}) \tag{15.11}$$

where $\alpha_m = \dfrac{-2}{T} + q_m$

$$\beta_m = \frac{-2}{T} - q_m$$

A C program for computation of the bilinear transformation is provided in Listing 15.1.

Listing 15.1 bilinear( )

```
/************************************/
/*                                  */
/*   Listing 15.1                   */
/*                                  */
/*   bilinear()                     */
/*                                  */
/************************************/


void bilinear(   struct complex pole[],
                 int numPoles,
                 struct complex zero[],
                 int numZeros,
                 real h2ero,
                 real bigT,
                 struct complex a[],
                 struct complex b[])
{
int j,k,m,n, maxCoef;
real hC;
struct complex mu[MAXPOLES];
struct complex alpha, beta, gamma, delta, eta;
struct complex work, cTwo;

for(j=0; j<MAXPOLES; j++) {
    mu[j] = cmplx(0.0,0.0);
    a[j] = cmplx(0.0,0.0);
    b[j] = cmplx(0.0,0.0);
    }
/*-------------------------------------*/
/*  compute constant gain factor       */
hC = 1.0;
work = cmplx(1.0,0.0);
cTwo = cmplx(2.0,0.0);
for(n=1; n<=numPoles; n++) {
    work = cMult(work, cSub(cTwo, sMult(bigT,pole[n])));
    hC = hC * bigT;
    }
hC = h2ero * hC / work.Re;
/*-------------------------------------*/
/*  compute numerator coefficients     */
mu[0] = cmplx(1.0, 0.0);
maxCoef = 0;
for( m=1; m<=(numPoles-numZeros); m++) {
    maxCoef++;
```

```
    for( j=maxCoef; j>=1; j--)
        { mu[j] = cAdd( mu[j], mu[j-1]);}
    }
for( m=1; m<=numZeros; m++) {
    maxCoef++;
    alpha = cAdd(cmplx( (-2.0/bigT), 0.0), zero[n]);
    beta = cSub(cmplx( (-2.0/bigT), 0.0), pole[n]);
    for(j=maxCoef; j>=1; j--)
        { mu[j] = cAdd( mu[j], cMult( beta, mu[j-1]));}
    }
for( j=0; j<=numPoles; j++) b[j] = sMult(hC, mu[j]);

/*-------------------------------------*/
/*  compute denominator coefficients   */
mu[0] = cmplx(1.0,0.0);
for(n=1; n<MAXPOLES; n++)
    {mu[n] = cmplx(0.0,0.0);}
for( n=1; n<=numPoles; n++) {
    gamma = cSub(cmplx( (2.0/bigT), 0.0), pole[n]);
    delta = cSub(cmplx( (-2.0/bigT), 0.0), pole[n]);
    eta = cDiv( delta, gamma);
    for( j=n; j>=1; j--)
        { mu[j] = cAdd( mu[j], cMult(eta, mu[j-1]));}
    }
for( j=1; j<=numPoles; j++)
    { a[j] = sMult(-1.0, mu[j] );}
return;
}
```