Chapter

13

FIR Filter Design: Remez Exchange Method

In general, an FIR approximation to an ideal lowpass filter will have an amplitude response of the form shown in Fig. 13.1. This response differs from the ideal lowpass response in three quantifiable ways:

1. The pass band has ripples that deviate from unity by $\pm\delta_p$.
2. The stop band has ripples that deviate from zero by $\pm\delta_s$. (Note that Fig. 13.1 shows an *amplitude* response rather than the usual magnitude response, and therefore negative ordinates are possible.)
3. There is a transition band of finite nonzero width ΔF between the pass band and stop band.

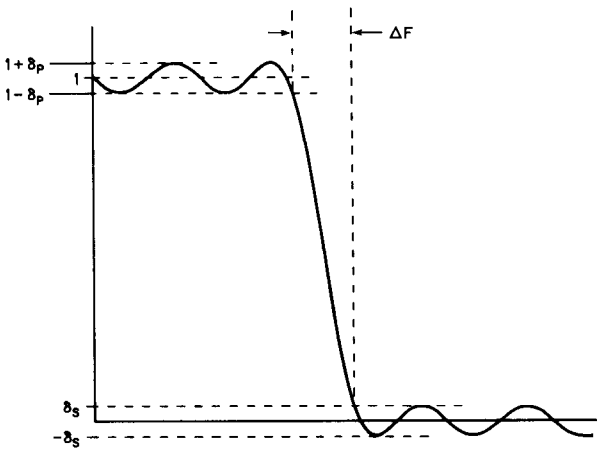

Figure 13.1 Typical amplitude response of an FIR approximation to an ideal lowpass filter.

The usual design goals are to, in some sense, minimize δ_p, δ_s, and ΔF. As it is generally not possible to simultaneously minimize for three different variables, some compromise is unavoidable. Chebyshev approximation is one approach to this design problem.

13.1 Chebyshev Approximation

In the Chebyshev approximation approach, the amplitude response of a type 1 (that is, odd-length, even-symmetric) linear phase lowpass N-tap FIR filter is formulated as a sum of r cosines:

$$A(f) = \sum_{k=0}^{r-1} c_k \cos(2\pi k f) \tag{13.1}$$

where $r = (N+1)/2$, and the coefficients c_k are chosen so as to yield an $A(f)$ which is optimal in a sense that will be defined shortly.

For a lowpass filter the pass band B_p and stop band B_s are defined as

$$B_p = \{F: 0 \le F \le F_p\} \tag{13.2}$$

$$B_s = \{F: F_s \le F \le 0.5\} \tag{13.3}$$

where F_p and F_s are, respectively, the edge frequencies for the pass band and stop band. [Equation (13.2) is read as "B_p is the set of all F such that F is greater than or equal to zero and less than or equal to F_p."] We can then define a set $\mathcal{F}$ as the union of B_p and B_s:

$$\mathcal{F} = B_p \cup B_s \tag{13.4}$$

In other words, $\mathcal{F}$ is the set of all frequencies between 0 and 0.5 not including the transition frequencies $F: F_p < F < F_s$. In mathematical terms, $\mathcal{F}$ is described as a *compact subset* of [0, 0.5]. The desired response $D(f)$ is the ideal lowpass response given by

$$D(f) = \begin{cases} 1 & F \in B_p \\ 0 & F \in B_s \end{cases} \tag{13.5}$$

Thus we could define the optimal approach as the one that minimizes the maximum error given by

$$\max_{F \in \mathcal{F}} |D(f) - A(f)| \tag{13.6}$$

However, the maximum error given by (13.6) treats pass-band error and stop-band error as equally important. A more general approach is to include a weighting function:

$$W(f) = \begin{cases} \dfrac{1}{K} & F \in B_p \\ 1 & F \in B_s \end{cases} \tag{13.7}$$

which allows stop-band errors to be given more importance than pass-band errors or vice versa. Thus we define the maximum approximation error as

$$\|E(f)\| = \max_{F \in \mathcal{F}} W(f) \cdot |D(f) - A(f)| \tag{13.8}$$

The crux of the Chebyshev approximation design approach is to identify the coefficients c_k for (13.1) that minimize $\|E(f)\|$.

Several examples of FIR design via Chebyshev approximation appear in the early literature (Martin 1962; Tufts, Rorabacher, and Moses 1970; Tufts and Francis 1970; Helms 1972; Herrman 1970; Hofstetter, Oppenheim, and Siegel 1971). However, the Chebyshev approximation method did not begin to enjoy widespread use until it was shown that the Remez exchange algorithm could be used to design linear phase FIR filters with the Chebyshev error criterion (Parks and McClellan 1972). Use of the Remez exchange algorithm depends upon an important mathematical result known as the *alternation theorem.*

Alternation theorem

The response $A(f)$ given by Eq. (13.1) will be the unique, best-weighted Chebyshev approximation to the desired response $D(f)$ if and only if the error function $E(f) = W(f)[D(f) - A(f)]$ exhibits at least $r + 1$ extrema at frequencies in $\mathcal{F}$. (Note: *Extrema* is a generic term that includes both maxima and minima.) The frequencies at which extrema occur are called *extremal frequencies.* Let f_n denote the nth extremal frequency such that

$$f_1 < f_2 < \cdots < f_{n-1} < f_n < f_{n+1} < \cdots < f_r < f_{r+1}$$

Then it can be proven (Cheyney 1966) that

$$E(f_n) = -E(f_{n+1}) \qquad n = 1, 2, \ldots, r \tag{13.9}$$

and

$$|E(f_n)| = \max_{f \in \mathcal{F}} E(f) \tag{13.10}$$

Together, (13.9) and (13.10) simply mean that the error is equal at all the extremal frequencies. Equation (13.9) further indicates that maxima and minima alternate (hence "alternation" theorem).

13.2 Strategy of the Remez Exchange Method

The alternation theorem given in the previous section tells us how to recognize an optimal set of c_k for Eq. (13.1) when we have one, but it does not tell us how to go about obtaining such c_k. The Remez exchange algorithm provides an approach for finding the FIR filter corresponding to the optimal c_k as follows:

1. Make an initial guess of the $r + 1$ extremal frequencies.
2. Compute the error function corresponding to the candidate set of extremal frequencies (see Sec. 13.3).

3. Search to find the extrema (and therefore the extremal frequencies) of the error function (see Sec. 13.4).
4. Adopt the extremal frequencies found in step 3 as the new set of candidate extremal frequencies and return to step 2.
5. Repeat steps 2, 3, and 4 until the extremal frequencies have converged (see Sec. 13.4).
6. Use the final set of extremal frequencies to compute $P(f)$ and the corresponding impulse response coefficients for the filter (see Sec. 13.5).

The error function mentioned in step 2 is computed as

$$E(f) = W(f)[D(f) - A(f)] \tag{13.11}$$

where $D(f)$ is given by Eq. (13.5) and $W(f)$ is given by (13.7). Although Eq. (13.1) gives the form of $A(f)$, some other means must be used to evaluate $A(f)$ since the coefficients c_k are unknown. We can obtain $A(f)$ from the extremal frequencies F_k using

$$A(f) = \begin{cases} \gamma_k & \text{for } f = F_0, F_1, \ldots, F_{r-1} \\ \dfrac{\sum_{k=0}^{r-1} \dfrac{\beta_k}{x - x_k} \gamma_k}{\sum_{k=0}^{r-1} \dfrac{\beta_k}{x - x_k}} & \text{otherwise} \end{cases} \tag{13.12}$$

The parameters needed for evaluation of (13.12) are given by

$$\beta_k = \prod_{\substack{i=0 \\ i \neq k}}^{r-1} \frac{1}{x_k - x_i}$$

$$\gamma_k = D(F_k) - (-1)^k \frac{\delta}{W(F_k)}$$

$$\delta = \frac{\sum_{k=0}^{r} \alpha_k D(F_k)}{\sum_{k=0}^{r} \dfrac{(-1)^k \alpha_k}{W(F_k)}}$$

$$\alpha_k = \prod_{\substack{i=0 \\ i \neq k}}^{r} \frac{1}{x_k - x_i}$$

$$x = \cos(2\pi f)$$

$$x_k = \cos(2\pi F_k)$$

If estimates of the extremal frequencies rather than their "true" values are used in the evaluation of $A(f)$, the resulting error function $E(f)$ will exhibit extrema at frequencies that are different from the original estimates. If the frequencies of these newly observed extrema are then used in a subsequent evaluation of $A(f)$, th new $E(f)$ will exhibit extrema at frequencies that are closer to the true extremal frequencies. If this process is performed repeatedly, the observed extremal frequencies will eventually converge to the true extremal frequencies, which can then be used to obtain $A(f)$ and the filter's impulse response.

Although $A(f)$ is defined over continuous frequency, computer evaluation of $A(f)$ must necessarily be limited to a finite number of discrete frequencies—therefore, $A(f)$ is evaluated over a closely spaced set or *dense grid* of frequencies. The convergence of the observed extremal frequencies will be limited by the granularity of this dense grid, but it has been empirically determined that an average grid density of 16 to 20 frequencies per extremum will be adequate for most designs. Since the maximization of $E(f)$ is only conducted over $f \in \mathcal{F}$, it is not necessary to evaluate $A(f)$ at all within the transition band (except for possibly at the very end, just to see what sort of transition-band response the final filter design actually provides). The frequency interval between consecutive points should be approximately the same in both the pass band and stop band. Furthermore, the grid should be constructed in such a way that frequency points are provided at $f = 0$, $f = F_p$, $f = F_s$, and $f = 0.5$. An integrated procedure for defining the dense grid and making the initial (equispaced) guesses for the candidate extremal frequencies is provided in Algorithm 13.1.

Algorithm 13.1 Constructing the dense-frequency grid

Step 1. Compute the number of candidate extremal frequencies to be placed in the pass band as

$$m_p = \left\lfloor \frac{rF_p}{0.5 + F_p - F_s} - 0.5 \right\rfloor$$

Step 2. Determine the candidate extremal frequencies within the pass band as

$$F_k = \frac{kF_p}{m_p} \qquad k = 1, 2, \ldots, m_p$$

Step 3. Compute the number of candidate extremal frequencies to be placed in the stop band as

$$m_s = r + 1 - m_p$$

Step 4. Determine the candidate extremal frequencies within the stop band as

$$F_k = F_s + \frac{k(0.5 - F_s)}{m_s - 1} \qquad k = 0, 1, \ldots, m_s - 1$$

Step 6. For each frequency f_j in the dense grid, use $A(f_j)$ from step 5 to compute $E(f_j)$ as

$$E(f_j) = W(f_j)[D(f_j) - A(f_j)]$$

For computer evaluation, the error function is calculated by **remezError( )**, which makes use of **computeRemezA( )**. These two functions are provided in Listings 13.4 and 13.5, respectively. The function **computeRemezA( )** could have been made an integral part of **remezError( )** and designed to automatically generate $A( )$ and $E( )$ for all frequencies within the dense grid. However, a function in this form would not be useable for generating the uniformly spaced samples of the final $A( )$ that are needed to conveniently obtain the impulse response of the filter.

13.4 Selecting Candidate Extremal Frequencies

Once Eq. (13.11) has been evaluated, the values of $E(f_j)$ must be checked in order to determine what the values of F_k should be for the next iteration of the optimization algorithm. Based upon the particular frequencies being checked, the testing can be divided into the five different variations that are described in the paragraphs below. A C function, **remezSearch( )**, which performs this testing is provided in Listing 13.6.

Testing $E(f)$ for $f = 0$

If $E(0) > 0$ and $E(0) > E(f_1)$, then a ripple peak (local maximum) exists at $f = 0$. (Note that f_1 denotes the first frequency within the "dense grid" after $f = 0$, and due to the way we have defined the frequency spacing with the grid, we know that $f_1 = I_p$.) Even if a peak or valley exists at $f = 0$, it may be a *superfluous* extremum not needed for the next iteration. If a ripple peak does exist at $f = 0$, and $|E(0)| \geq |\rho|$, then the maximum is not superfluous and $f = f_0 = 0$ should be used as the first-candidate extremal frequency—in other words, set $F_0 = f_0 = 0$. Similarly, if $E(0) < 0$ and $E(0) < E(f_1)$, a ripple trough (ripple valley, local minimum) exists at $f = 0$. If $|E(0)| \geq |\rho|$, this minimum is not superfluous and we should set $F_0 = f_0 = 0$.

Testing $E(f)$ within the pass band and the stop band

The following discussion applies to testing of $E(f)$ for all values of f_j for which $f_0 < f_j < f_p$ or for which $f_s < f_j < 0.5$. A ripple peak exists at f_j if

$$E(f_j) > E(f_{j-1}) \quad \text{and} \quad E(f_j) > E(f_{j+1}) \quad \text{and} \quad E(f_j) > 0 \tag{13.13}$$

Equation (13.13) can be rewritten as (13.14) for frequencies in the pass band and as (13.15) for frequencies within the stop band:

$$E(f_j) > E(f_j - I_p) \quad \text{and} \quad E(f_j) > E(f_j + I_p) \quad \text{and} \quad E(f_j) > 0 \tag{13.14}$$

$$E(f_j) > E(f_j - I_s) \quad \text{and} \quad E(f_j) > E(f_j + I_s) \quad \text{and} \quad E(f_j) > 0 \tag{13.15}$$

A ripple trough exists at f_j if

$$E(f_j) < E(f_{j-1}) \quad \text{and} \quad E(f_j) < E(f_{j+1}) \quad \text{and} \quad E(f_j) < 0 \tag{13.16}$$

Equation (13.16) can be rewritten as (13.17) for frequencies in the pass band and as (13.18) for frequencies within the stop band:

$$E(f_j) < E(f_j - I_p) \quad \text{and} \quad E(f_j) < E(f_j + I_p) \quad \text{and} \quad E(f_j) < 0 \tag{13.17}$$

$$E(f_j) < E(f_j - I_s) \quad \text{and} \quad E(f_j) < E(f_j + I_s) \quad \text{and} \quad E(f_j) < 0 \tag{13.18}$$

If either (13.13) or (13.16) is satisfied, $f = f_j$ should be selected as a candidate extremal frequency—that is, set $F_k = f_j$ where k is the index of the next extremal frequency due to be specified.

Testing of $E(f)$ at the pass-band and stop-band edges

There is some disagreement within the literature regarding the testing of the pass-band and stop-band edge frequencies f_p and f_s. Some authors (such as Antoniou 1982) indicate the following testing strategy for f_p and f_s:

> If $E(f_p) > 0$ and $E(f_p) > E(f_p - I_p)$, then a ripple peak (local maximum) is deemed to exist at $f = f_p$ regardless of how $E(f)$ behaves in the transition band which lies immediately to the right of $f = f_p$. If a ripple peak does exist at $f = f_p$, and if $|E(f_p)| \geq |\rho|$, then the maximum is not superfluous and $f = f_p$ should be selected as a candidate extremal frequency—i.e., set $F_k = f_p$ where k is the index of the next extremal frequency due to be specified. Similarly, if $E(f_p) < 0$ and $E(f_p) < E(f_p - I_p)$, a ripple trough exists at $f = f_p$. If $|E(f_p)| \geq |\rho|$, this minimum is not superfluous and we should set $F_k = f_p$ where k is the index of the next extremal frequency due to be specified. If $E(f_s) > 0$ and $E(f_s) > E(f_s + I_s)$, then a ripple peak is deemed to exist at $f = f_s$ regardless of how $E(f)$ behaves in the transition band which lies immediately to the left of $f = f_s$. If a ripple peak does exist at $f = f_s$, and if $|E(f_s)| \geq |\rho|$, then the maximum is not superfluous and $f = f_s$ should be selected as a candidate extremal frequency—i.e., set $F_k = f_s$ where k is the index of the next extremal frequency due to be specified. Similarly, if $E(f_s) < 0$ and $E(f_s) < E(f_s + I_s)$, a ripple trough exists at $f = f_s$. If $|E(f_p)| \geq |\rho|$, this minimum is not superfluous and we should set $F_k = fs_p$ where k is the index of the next extremal frequency due to be specified.

Other authors (such as Parks and Burrus 1987) indicate that f_p and f_s are *always* extremal frequencies. In my experience the testing indicated by Antoniou is always satisfied, so f_p and f_s are always selected as extremal frequencies. I have opted to eliminate this testing both to reduce execution time and to avoid the danger of having small numerical inaccuracies cause one of these points to erroneously fail the test and thereby be rejected.

Testing of $E(f)$ for $f = 0.5$

If $E(0.5) > 0$ and $E(0.5) > E(0.5 - I_s)$, then a ripple peak exists at $f = 0.5$. If a ripple peak does exist at $f = 0.5$, and if $|E(0)| \geq |\rho|$, then the maximum is not

superfluous and $f = f_0 = 0.5$ should be used as the final candidate extremal frequency. Similarly, if $E(0.5) < 0$ and $E(0.5) < E(0.5 - I_s)$, a ripple trough (ripple valley, local minimum) exists at $f = 0.5$. If $|E(0)| \geq |\rho|$, this minimum is not superfluous.

Rejecting superfluous candidate frequencies

The Remez algorithm requires that only $r + 1$ extremal frequencies be used in each iteration. However, when the search procedures just described are used, it is possible to wind up with more than $r + 1$ candidate frequencies. This situation can be very easily remedied by retaining only the $r + 1$ frequencies F_k for which $|E(F_k)|$ is the largest. The retained frequencies are renumbered from 0 to r before proceeding. An alternative approach is to reject the frequency corresponding to the smaller of $|E(F_0)|$ and $|E(F_r)|$, regardless of how these two values compare to the absolute errors at the other extrema. Since there is only one solution for a given set of filter specifications, both approaches should lead to the same result. However, one approach may lead to a faster solution or be less prone to numeric difficulties. This would be a good area for a small research effort.

Deciding when to stop

There are two schools of thought on deciding when to stop the exchange algorithm. The original criterion (Parks and McClellan 1972) examines the extremal frequencies and stops the algorithm when they do not change from one iteration to the next. This criterion is implemented in the C function **remezStop( )** provided in Listing 13.7. This approach has worked well for me, but it does have a potential flaw. Suppose that one of the true extremal frequencies for a particular filter lies at $f = F_T$, and due to the way the dense grid has been defined, F_T lies midway between two grid frequencies such that

$$F_T = \frac{f_n + f_{n+1}}{2}$$

It is conceivable that on successive iterations, the observed extremal frequency could alternate between f_j and f_{n+1} and therefore never allow the stopping criteria to be satisfied.

A different criterion, advocated by Antoniou (1982), uses values of the error function rather then the locations of the extremal frequencies. In theory, when the Remez algorithm is working correctly, each successive iteration will produce continually improving estimates of the correct extremal frequencies, and the values of $|E(F_k)|$ will become exactly equal for all values of k. However, due to the finite resolution of the frequency grid as well as finite precision arithmetic, the estimates may in fact never converge to exact equality. One remedy is to stop when the largest $|E(F_k)|$ and the

smallest $|E(F_k)|$ differ by some reasonably small amount. The difference as a fraction of the largest $|E(F_k)|$ is given by

$$Q = \frac{\max|E(F_k)| - \min|E(F_k)|}{\max|E(F_k)|}$$

Typically, the iterations are stopped when $Q \leq 0.01$. This second stopping criterion is implemented in the C function **remezStop2( )** provided in Listing 13.8.

13.5 Obtaining the Impulse Response

Back in Sec. 13.2, the final step in the Remez exchange design strategy consisted of using the final set of extremal frequencies to obtain the filter's impulse response. This can be accomplished by using Eq. (13.10) to obtain $P(f)$ from the set of extremal frequencies and then performing an inverse DFT on $P(f)$ to obtain the corresponding impulse response. An alternative approach involves deriving a dedicated inversion formula similar to the dedicated formulas presented in Sec. 12.3. For the case of the type 1 filter that has been considered thus far, the required inversion formula is

$$h[n] = h[-n] = \frac{1}{N}\left[A(0) + \sum_{k=1}^{r-1} 2A\left(\frac{2\pi k}{N}\right)\cos\left(\frac{2\pi kn}{N}\right)\right]$$

This formula is implemented via the **fsDesign( )** function (from Chap. 12), which is called by the C function **remezFinish( )** provided in Listing 13.9. Although the filter's final frequency response could be obtained using calls to **computRemezA( )**, I have found it more convenient to use **cgdFirResponse( )** from Chap. 10, since this function produces output in a form that is directly compatible with my plotting software.

13.6 Using the Remez Exchange Method

All of the constituent functions of the Remez method that have been presented in previous sections are called in the proper sequence by the function **remez( )**, which is presented in Listing 13.10. This function accepts the inputs listed in Table 13.1 and produces two outputs—**extFreq[ ]**, which is a vector containing the final estimates, and **h[ ]**, which is a vector containing the FIR filter coefficients.

Deciding on the filter length

To use the Remez exchange method, the designer must specify N, f_s, f_p, and the ratio δ_1/δ_2. The algorithm will provide the filter having the smallest values of $|\delta_1|$ and $|\delta_2|$ that can be achieved under these constraints. However, in many applications, the values specified are f_p, f_s, δ_1, and δ_2 with the

TABLE 13.1 Input Parameters for remez( ) Function

Mathematical symbol	C variable	Definition
N	**nn**	Filter length
r	**r**	Number of approximating functions
L	**gridDensity**	Average density of frequency grid (in grid points per extremal frequency) (must be an integer)
K	**kk**	Ripple ratio δ_1/δ_2
f_p	**freqP**	Pass-band edge frequency
f_s	**freqS**	Stop-band edge frequency

designer left free to set N as required. Faced with such a situation, the designer can use f_p, f_s, and $K = \delta_1/\delta_2$ as dictated by the application and design filters for increasing values of N until the δ_1 and δ_2 specifications are satisfied. An approximation of the required number of taps can be obtained by one of the formulas given below. For filters having pass bands of "moderate" width, the approximate number of taps required is given by

$$\tilde{N} = 1 + \frac{-20 \log \sqrt{\delta_1 \delta_1} - 13}{14.6(f_s - f_p)} \tag{13.19}$$

For filters with very narrow pass bands, (13.19) can be modified to be

$$\tilde{N} = \frac{0.22 - (20 \log \delta_2)/27}{(f_s - f_p)} \tag{13.20}$$

For filters with very wide pass bands, the required number of taps is approximated by

$$\tilde{N} = \frac{0.22 - (20 \log \delta_1)/27}{(f_s - f_p)} \tag{13.21}$$

Example 13.1 Suppose we wish to design a lowpass filter with a maximum pass-band ripple of $\delta_1 = 0.025$ and a minimum stop-band attenuation of 60 dB or $\delta_2 = 0.001$. The normalized cutoff frequencies for the pass band and stop band are, respectively, $f_p = 0.215$ and $f_s = 0.315$. Using (13.19) to approximate the required filter length N, we obtain

$$\tilde{N} = 1 + \frac{-20 \log \sqrt{(0.001)(0.025)} - 13}{14.6(0.315 - 0.215)}$$
$$= 23.6$$

The next larger odd length would be $N = 25$. If we run **remez( )** with the following inputs:

$$\mathbf{nn} = 25 \qquad \mathbf{r} = 13 \qquad \mathbf{gridDensity} = 16$$

$$\mathbf{kk} = 25.0 \qquad \mathbf{freqP} = 0.215 \qquad \mathbf{freqS} = 0.315$$

TABLE 13.2 Extremal Frequencies for Example 13.1

k	f_k
0	0.000000
1	0.042232
2	0.084464
3	0.126696
4	0.165089
5	0.199643
6	0.215000
7	0.315000
8	0.322708
9	0.343906
10	0.372813
11	0.407500
12	0.447969
13	0.500000

TABLE 13.3 Coefficients for 25-tap FIR Filter of Example 13.1

$h[0] = h[24]$	$= -0.004069$
$h[1] = h[23]$	$= -0.010367$
$h[2] = h[22]$	$= -0.001802$
$h[3] = h[21]$	$= 0.015235$
$h[4] = h[20]$	$= 0.003214$
$h[5] = h[19]$	$= -0.027572$
$h[6] = h[18]$	$= -0.005119$
$h[7] = h[17]$	$= 0.049465$
$h[8] = h[16]$	$= 0.007009$
$h[9] = h[15]$	$= -0.096992$
$h[10] = h[14]$	$= -0.008320$
$h[11] = h[13]$	$= 0.315158$
$h[12]$	$= 0.508810$

we obtain the extremal frequencies listed in Table 13.2 and the filter coefficients listed in Table 13.3. The frequency response of the filter is shown in Figs. 13.2 and 13.3. The actual pass-band and stop-band ripple values of 0.0195 and 0.000780 are significantly better than the specified values of 0.025 and 0.001.

Example 13.2 The ripple performance of the 25-tap filter designed in Example 13.1 exhibits a certain amount of overachievement, and the estimate of the minimum number

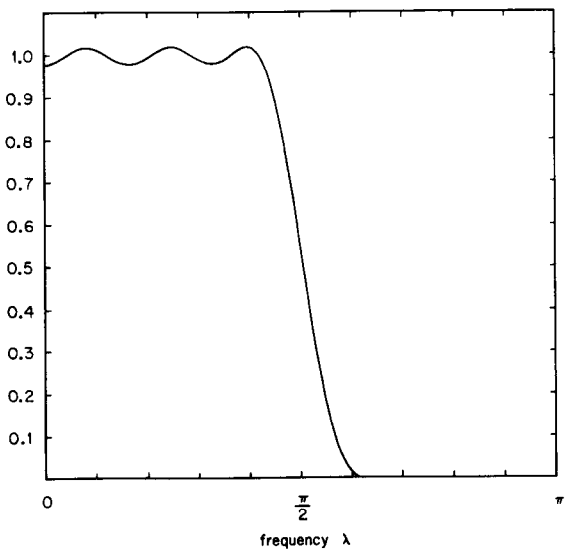

Figure 13.2 Magnitude response (as a fraction of peak) for the filter of Example 13.1.

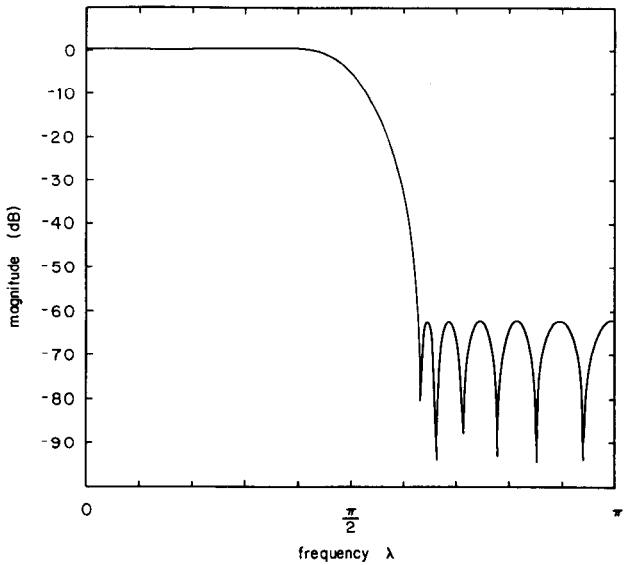

Figure 13.3 Magnitude response (in decibels) for the filter of Example 13.1.

of taps was closer to 23 than 25. Therefore, it would be natural for us to ask if we could in fact achieve the desired performance with a 23-tap filter. If we rerun **remez( )** with **nn** = 23, we obtain the extremal frequencies and filter coefficients listed in Tables 13.4 and 13.5. The frequency response of this filter is shown in Figs. 13.4 and 13.5. The pass-band ripple is approximately 0.034, and the stop-band ripple is approximately 0.00138—therefore, we conclude that a 23-tap filter does not satisfy the specified requirements.

TABLE 13.4 Extremal Frequencies for Example 13.2

k	f_k
0	0.000000
1	0.051510
2	0.103021
3	0.152292
4	0.194844
5	0.215000
6	0.315000
7	0.324635
8	0.349688
9	0.382448
10	0.419062
11	0.459531
12	0.500000

TABLE 13.5 Coefficients for 23-tap FIR Filter of Example 13.2

$$h[0] = h[22] = -0.000992$$
$$h[1] = h[21] = 0.007452$$
$$h[2] = h[20] = 0.018648$$
$$h[3] = h[19] = 0.002873$$
$$h[4] = h[18] = -0.026493$$
$$h[5] = h[17] = -0.003625$$
$$h[6] = h[16] = 0.048469$$
$$h[7] = h[15] = 0.005314$$
$$h[8] = h[14] = -0.096281$$
$$h[9] = h[13] = -0.006601$$
$$h[10] = h[12] = -0.314911$$
$$h[11] = 0.507077$$

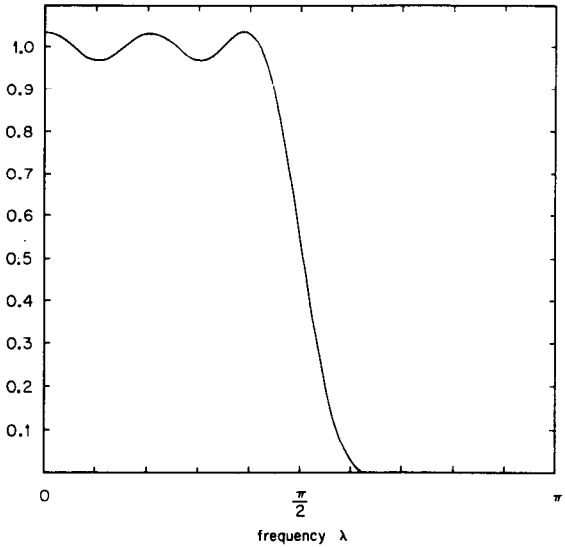

Figure 13.4 Magnitude response (as a fraction of peak) for the filter of Example 13.2.

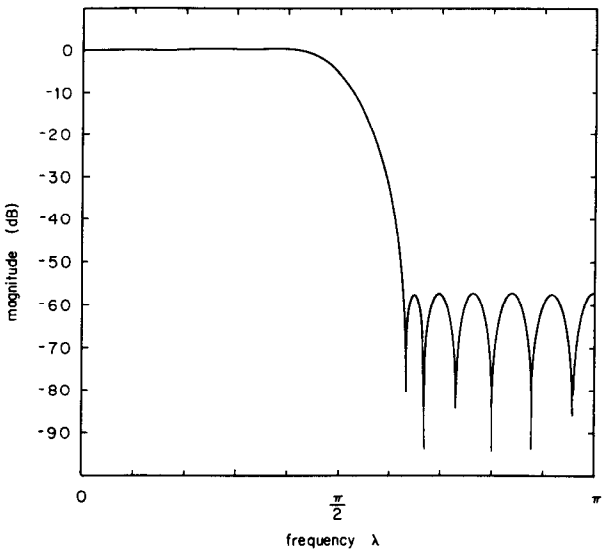

Figure 13.5 Magnitude response (in decibels) for the filter of Example 13.2.

13.7 Extension of the Basic Method

So far we have considered use of the Remez exchange method for odd-length, linear phase FIR filters having even-symmetric impulse responses (that is, type 1 filters). The Remez method was originally adapted specifically for the

design of type 1 filters (Parks and McClellan 1972). However, in a subsequent paper, Parks and McClellan (1973) noted that the amplitude response of any constant group-delay FIR filter can be expressed as

$$A(f) = Q(f)\,P(f)$$

where $P(f) = \sum_{k=0}^{r-1} c_k \cos(2\pi k f)$

$$Q(f) = \begin{cases} 1 & h[n] \text{ symmetric, } N \text{ odd} \\ \cos \pi f & h[n] \text{ symmetric, } N \text{ even} \\ \sin 2\pi f & h[n] \text{ antisymmetric, } N \text{ odd} \\ \sin \pi f & h[n] \text{ antisymmetric, } N \text{ even} \end{cases}$$

Recall that the error $E(f)$ was defined as

$$E(f) = W(f)[D(f) - A(f)] \tag{13.22}$$

If we substitute $Q(f)P(f)$ and factor out $Q(f)$, we obtain

$$E(f) = W(f)\,Q(f)\left[\frac{D(f)}{Q(f)} - P(f)\right]$$

We can then define a new weighting function $\hat{W}(f) = W(f)Q(f)$ and a new desired response $\hat{D}(f) = D(f)/Q(f)$, and thereby obtain

$$E(f) = \hat{W}(f)[\hat{D}(f) - P(f)] \tag{13.23}$$

Equation (13.23) is of the same form as (13.22) with $\hat{W}(f)$ substituted for $W(f)$, $\hat{D}(f)$ substituted for $D(f)$, and $P(f)$ substituted for $A(f)$. Therefore, the procedures developed in previous sections can be used to solve for $P(f)$ provided that $\hat{W}(f)$ is used in place of $W(f)$ and $\hat{D}(f)$ is used in place of $D(f)$. Once this $P(f)$ is obtained, we can multiply by the appropriate $Q(f)$ to obtain $A(f)$. The appropriate formula from Table 12.2 can then be used to obtain the impulse response coefficients $h[n]$.

Listing 13.1 gridFreq( )

```
/***********************************/
/*                                 */
/*   Lisitng 13.1                  */
/*                                 */
/*   gridFreq()                    */
/*                                 */
/***********************************/

real gridFreq(   real gridParam[],
                 int gI)
{
real work;
static real incP, incS, freqP, freqS;
static int r, gridDensity, mP, mS, gP;

if(gridParam[0] == 1.0) {
    gridParam[0] = 0.0;
    freqP = gridParam[1];
    freqS = gridParam[2];
    r = gridParam[3];
    gridDensity = gridParam[4];
    work = (0.5 + freqP - freqS)/r;
    mP = floor(0.5 + freqP/work);
    gridParam[5] = mP;
    gP = mP * gridDensity;
    gridParam[7] = gP;
    mS = r +1 - mP;
    gridParam[6] = mS;
    incP = freqP / gP;
    incS = (0.5-freqS) / ((mS-1) * gridDensity);
    }
else {
    work = (gI<=gP) ? (gI*incP) : (freqS+(gI-(gP+1))*incS);
    }
return(work);
}
```

Listing 13.2 desLpfResp( )

```
/***********************************/
/*                                 */
/*   Listing 13.2                  */
/*                                 */
/*   desLpfResp()                  */
/*                                 */
/***********************************/

real desLpfResp( real freqP, real freq)
{
real result;
result = 0.0;
if(freq <= freqP) result = 1.0;
return(result);
}
```

Listing 13.3 weightLp( )

```
/***********************************/
/*                                 */
/*   Listing 13.3                  */
/*                                 */
/*   weightLp()                    */
/*                                 */
/***********************************/

real weightLp( real kk, real freqP, real freq)
{
real result;

result = 1.0;
if(freq <= freqP) result = 1.0/kk;
return(result);
}
```

Listing 13.4 remezError( )

```
/**********************************/
/*                                */
/*   Listing 13.4                 */
/*                                */
/*   remezError()                 */
/*                                */
/**********************************/

void remezError( real gridParam[],
                 int gridMax,
                 int r,
                 real kk,
                 real freqP,
                 int iFF[],
                 real ee[])
{
int j;
real freq,aa;

aa = computeRemezA(  gridParam, gridMax, r, kk,
                 freqP, iFF, 1, 0.0);

for( j=0; j<=gridMax; j++) {
    freq = gridFreq(gridParam,j);
    aa = computeRemezA(  gridParam,
                         gridMax, r, kk, freqP,
                         iFF, 0,freq);
    ee[j] = weightLp(kk,freqP,freq) *
                (desLpfResp(freqP,freq) - aa);
    }
return;
}
```

Listing 13.5 computeRemezA( )

```
/**********************************/
/*                                */
/*   Listing 13.5                 */
/*                                */
/*   computeRemezA()              */
/*                                */
/**********************************/

real computeRemezA(  real gridParam[],
                 int gridMax,
```

```
                    int r,
                    real kk,
                    real freqP,
                    int iFF[],
                    int initFlag,
                    real contFreq)
{
static int i, j, k, sign;
static real freq, denom, numer, alpha, delta;
static real absDelta, xCont, term;
static real x[50], beta[50], gamma[50];
real aa;

if(initFlag) {
    for(j=0; j<=r; j++) {
        freq = gridFreq(gridParam,iFF[j]);
        x[j] = cos(TWO_PI * freq);
        }

    /*  compute delta  */
    denom = 0.0;
    numer = 0.0;
    sign = -1;
    for( k=0; k<=r; k++) {
        sign = -sign;
        alpha = 1.0;
            for( i=0; i<=(r-1); i++) {
            if(i==k) continue;
            alpha = alpha / (x[k] - x[i]);
            }

    beta[k] = alpha;
    if( k != r ) alpha = alpha/(x[k] - x[r]);
    freq =  gridFreq(gridParam,iFF[k]);
    numer = numer + alpha * desLpfResp(freqP,freq);
    denom = denom + sign*(alpha/
                        weightLp(kk, freqP, freq));
    }
delta = numer/denom;
absDelta = fabs(delta);

sign = -1;
for( k=0; k<=r-1; k++) {
    sign = -sign;
    freq = gridFreq(gridParam,iFF[k]);
    gamma[k] = desLpfResp(freqP, freq) - sign * delta /
                    weightLp(kk,freqP,freq);
    }
```

```
    }
else {
    xCont = cos(TWO_PI * contFreq);
    numer = 0.0;
    denom = 0.0;
    for( k=0; k<r; k++) {
        term = xCont - x[k];
        if(fabs(term)<1.0e-7) {
            aa = gamma[k];
            goto done;
            }
        else {
            term = beta[k]/(xCont - x[k]);
            denom += term;
            numer += gamma[k]*term;
            }
        }
    aa = numer/denom;
    }
done:
return(aa);
}
```

Listing 13.6 remezSearch( )

```
/***********************************/
/*                                 */
/*   Listing 13.6                  */
/*                                 */
/*   remezSearch()                 */
/*                                 */
/***********************************/

void remezSearch(real ee[],
                 real absDelta,
                 int gP,
                 int iFF[],
                 int gridMax,
                 int r,
                 real gridParam[])
{
```

```
int i,j,k,extras,indexOfSmallest;
real smallestVal;

k=0;

/* test for extremum at f=0  */
if( ( (ee[0]>0.0) && (ee[0]>ee[1]) && (fabs(ee[0])>=absDelta) ) ||
    ( (ee[0]<0.0) && (ee[0]<ee[1]) && (fabs(ee[0])>=absDelta) ) ) {
    iFF[k]=0;
    k++;
    }

/*  search for extrema in passband  */
for(j=1; j<gP; j++) {
    if( ( (ee[j]>=ee[j-1]) && (ee[j]>ee[j+1]) && (ee[j]>0.0) ) ||
        ( (ee[j]<=ee[j-1]) && (ee[j]<ee[j+1]) && (ee[j]<0.0) )) {
        iFF[k] = j;
        k++;
        }
    }

/* pick up an extremal frequency at passband edge  */
    iFF[k]=gP;
    k++;

/* pick up an extremal frequency at stopband edge  */
j=gP+1;
    iFF[k]=j;
    k++;

/*  search for extrema in stopband  */

for(j=gP+2; j<gridMax; j++) {
    if( ( (ee[j]>=ee[j-1]) && (ee[j]>ee[j+1]) && (ee[j]>0.0) ) ||
        ( (ee[j]<=ee[j-1]) && (ee[j]<ee[j+1]) && (ee[j]<0.0) )) {
        iFF[k] = j;
        k++;
        }
    }
/* test for extremum at f=0.5  */
j = gridMax;
if( ( (ee[j]>0.0) && (ee[j]>ee[j-1]) && (fabs(ee[j])>=absDelta) ) ||
    ( (ee[j]<0.0) && (ee[j]<ee[j-1]) && (fabs(ee[j])>=absDelta) ) ) {
    iFF[k]=gridMax;
    k++;
    }
/*-------------------------------------------------*/
/*  find and remove superfluous extremal frequencies  */
```

```
if( k>r+1) {
    extras = k - (r+1);
    for(i=1; i<=extras; i++) {
        smallestVal = fabs(ee[iFF[0]]);
        indexOfSmallest = 0;
        for(j=1; j< k; j++) {
            if(fabs(ee[iFF[j]]) >= smallestVal) continue;
            smallestVal = fabs(ee[iFF[j]]);
            indexOfSmallest = j;
            }
        k--;
        for(j=indexOfSmallest; j<k; j++) iFF[j] = iFF[j+1];
        }
    }
return;
}
```

Listing 13.7 remezStop( )

```
/**********************************/
/*                                */
/*   Listing 13.7                 */
/*                                */
/*   remezStop()                  */
/*                                */
/**********************************/

int remezStop(   int iFF[],
                 int r)
{
static int oldIFF[50];
int j,result;

result = 1;
for(j=0; j<=r; j++) {
    if(iFF[j] != oldIFF[j]) result = 0;
    oldIFF[j] = iFF[j];
    }
return(result);
}
```

Listing 13.8 remezStop2( )

```
/**********************************/
/*                                */
/*   Listing 13.8                 */
/*                                */
/*   remezStop2()                 */
/*                                */
/**********************************/

int remezStop2(  real ee[],
                 int iFF[],
                 int r)
{
real biggestVal, smallestVal,qq;
int j,result;

result = 0;
biggestVal = fabs(ee[iFF[0]]);
smallestVal = fabs(ee[iFF[0]]);
for(j=1; j<=r; j++) {
    if(fabs(ee[iFF[j]]) < smallestVal) smallestVal = fabs(ee[iFF[j]]);
    if(fabs(ee[iFF[j]]) > biggestVal) biggestVal = fabs(ee[iFF[j]]);
    }
qq = (biggestVal - smallestVal)/biggestVal;
if(qq<0.01) result=1;
return(result);
}
```

Listing 13.9 remezFinish( )

```
/**********************************/
/*                                */
/*   Listing 13.9                 */
/*                                */
/*   remezFinish()                */
/*                                */
/**********************************/

void remezFinish(real extFreq[],
                 int nn,
                 int r,
                 real freqP,
                 real kk,
                 real aa[],
                 real h[])
{
```

```
int k,n, gridMax, iFF[1];
real freq,sum;
static real gridParam[1];

for(k=0; k<r; k++) {
    freq = (real) k/ (real) nn;
    aa[k] = computeRemezA(    gridParam, gridMax, r, kk,
                              freqP, iFF, 0,freq);
    }
fsDesign( nn, 1, aa, h);
return;
}
```

Listing 13.10 remez( )

```
/***********************************/
/*                                 */
/*   Listing 13.10                 */
/*                                 */
/*   remez()                       */
/*                                 */
/***********************************/

void remez(  int nn,
             int r,
             int gridDensity,
             real kk,
             real freqP,
             real freqS,
             real extFreq[],
             real h[])
{
int m, gridMax, j, mP, gP, mS;
real absDelta,freq;
static real gridParam[10];
static int iFF[50];
static real ee[1024];

/*--------------------------------*/
/*  set up frequency grid         */
gridParam[0] = 1.0;
gridParam[1] = freqP;
gridParam[2] = freqS;
gridParam[3] = r;
gridParam[4] = gridDensity;
freq = gridFreq(gridParam,0);
```

```
mP = gridParam[5];
mS = gridParam[6];
gP = gridParam[7];
freqP = freqP + (freqP/(2.0*gP));
gridMax = 1 + gridDensity*(mP+mS-1);
/*-------------------------------------------------*/
/*  make initial guess of extremal frequencies  */

for(j=0; j<mP; j++) iFF[j] = (j+1)* gridDensity;

for(j=0; j<mS; j++) iFF[j+mP] = gP + 1 + j * gridDensity;

/*-------------------------------------------------*/
/*  find optimal locations for extremal frequencies  */

for(m=1;m<=20;m++) {

    remezError(  gridParam, gridMax, r, kk, freqP, iFF, ee);

    remezSearch( ee, absDelta, gP, iFF, gridMax, r, gridParam);

    remezStop2(ee,iFF,r);
    if(remezStop(iFF,r)) break;
    }

for(j=0; j<=r; j++) {
    extFreq[j] = gridFreq(gridParam,iFF[j]);
    }
remezFinish( extFreq, nn, r, freqP,kk, ee, h);
return;
}
```