CHAPTER 6 INFINITE IMPULSE RESPONSE (IIR) FILTERS

This chapter introduces infinite impulse response (IIR) digital filters. Several types of IIR filters are designed using the Filter Design and Analysis Tool (FDAT). Comparisons are made with the FIR filters introduced in the previous chapter. Practical considerations including finite wordlength effects, arithmetic rounding, arithmetic overflow, and filter realizations (cascade and parallel) are also covered.

6.1 INFINITE IMPULSE RESPONSE (IIR) DIGITAL FILTERS

IIR (Infinite Impulse Response) digital filters can be described by the input/output difference equation:

\[
y(k) = \sum_{i=0}^{N} b_i x(k - i) - \sum_{i=1}^{N} a_i y(k - i)
\]  

(6.1)

- The constants \(b_0, b_1, \ldots, b_N\) and \(a_1, \ldots, a_N\) are the filter coefficients.
- \(N\) is the filter order. There are \(2N+1\) filter coefficients.
- Designing an IIR filter involves finding the filter order and a set of filter coefficients that meet the given filter specifications.
- The output of the filter is a multiply and accumulate (MAC) calculation: the current input and \(N\) past inputs and outputs are multiplied by the filter coefficients. The resulting products are added together.
- To implement the filter, the filter coefficients, the current input, a finite number of past inputs, and a finite number of past outputs must be stored in memory.
- An IIR filter is recursive; that is, the output of the filter depends on past filter outputs as well as the input signal. An FIR filter depends only on the input signal and is therefore non-recursive.

Impulse Response of IIR Filters

The impulse response was defined in Chapter 4 as the output response when the input signal is a discrete unit impulse; that is \(\delta(k) = 1\) for \(k = 0\) and \(\delta(k) = 0\) for \(k \neq 0\). The impulse response of an FIR filter is simply equal to the filter coefficients and therefore is finite in length. This is not true for an IIR filter. The impulse response of an IIR filter is infinite in length; however, as long as the filter is stable then \(h(k) \rightarrow 0\) as \(k \rightarrow \infty\). The impulse response can be derived using the recursion method on the difference equation (although an obvious pattern doesn’t always emerge), or by finding the inverse Z-Transform of the transfer function, or by using the `impulse` function in MATLAB.

Transfer Function of IIR Filters

The transfer function of an IIR filter can be derived by inspection from the input/output difference equation using the time shift property of Z-transforms. The transfer function of a general IIR filter is shown in Equation 6.2.
\[ H(z) = \frac{b_0 + b_1 z^{-1} + \ldots + b_N z^{-N}}{1 + a_1 z^{-1} + \ldots + a_N z^{-N}} = \frac{b_0 z^N + b_1 z^{N-1} + \ldots + b_N}{z^N + a_1 z^{N-1} + \ldots + a_N} \]

(6.2)

**Stability of IIR Filters**

Discrete-time systems are stable if the magnitude of each of the poles is strictly less than one. The poles of an IIR filter are the N roots of the denominator. Unlike an FIR filter which is always stable since all N poles are at the origin, stability can be an issue for IIR filters.

**Output of an IIR Filter**

The output of an IIR filter can be computed using several different methods including the following:

1. Recursively using the input/output difference equation
2. Through convolution using \( y(k) = x(k) \ast h(k) \)
3. Taking an inverse Z-Transform of \( Y(z) = H(z)X(z) \)
4. Through MATLAB using the command \( y = \text{filter}(\text{num}_\text{tf},\text{den}_\text{tf},x) \)
5. Using Fast Fourier Transforms (FFTs) to be discussed in Chapter 7

**Example 6.1: Transfer Function, Impulse Response, and Output Response of an IIR Filter**

An IIR filter is described by

\[ y(k) = 0.3y(k-1) + 0.1y(k-2) + x(k) - x(k-1) \]

(a) Find the filter order, the transfer function for the IIR filter, the filter poles, and determine whether or not the filter is stable.

(b) Find the impulse response of the filter.

(c) Find the output response for the IIR filter if the input sequence \( x(k) = \{1, 3, 6, -1, 5\} \).

**Solution**

(a) The filter order for the IIR filter is the largest delay on the input or output signal so \( N = 2 \).

The transfer function is derived by taking the \( Z \)-Transform of the difference equation and solving for \( Y(z) \). Be careful about the sign of the denominator coefficients. To avoid sign errors it is a good idea to move all \( y \) terms in the difference equation to the left-hand side before determining the transfer function.

\[ y(k) - 0.3y(k-1) - 0.1y(k-2) = x(k) - x(k-1) \]

\[ H(z) = \frac{1 - z^{-1}}{1 - 0.3z^{-1} - 0.1z^{-2}} = \frac{z^2 - z}{z^2 - 0.3z - 0.1} \]

The filter poles are found by finding the roots of the denominator:

\[ z^2 - 0.3z - 0.1 = (z - 0.5)(z + 0.2) = 0 \]
The filter poles are at 0.5 and −0.2. Since both poles have magnitude smaller than one, the IIR filter is stable. The MATLAB function, \texttt{roots}, can be used for a denominator that is not easily factored by inspection.

(b) The impulse response is the response of the system when the input is a discrete impulse. The first ten terms of the impulse response (h(0) through h(9)) can be found using the \texttt{dimpulse} function in MATLAB as follows:

\begin{verbatim}
num_tf = [1 -1 0];
den_tf = [1 -0.3 -0.1];
h = dimpulse(num_tf,den_tf,10)
\end{verbatim}

\begin{verbatim}
h =
     1.0000
    -0.7000
    -0.1100
    -0.1030
    -0.0419
    -0.0229
    -0.0111
    -0.0056
    -0.0028
    -0.0014
\end{verbatim}

Since the filter is stable, the impulse response will converge to zero as \(k \to \infty\). The function \texttt{dimpulse} will provide any specified number of values for \(h(k)\) but does not provide a mathematical expression for \(h(k)\). Solving for \(h(k)\) recursively using the difference equation will provide the same finite set of values although it takes a lot longer. If a mathematical expression is required for \(h(k)\), an inverse Z-Transform of the transfer function, \(H(z)\), will do the trick.

\[
\frac{H(z)}{z} = \frac{z - 1}{z^2 - 0.3z - 0.1}
\]

The partial fraction expansion can be easily computed using the MATLAB command:

\begin{verbatim}
[coeff poles] = residue([1 -1], [1 -0.3 -0.1]);
[coeff poles]
ans =
    -0.7143    0.5000
    1.7143    -0.2000
\end{verbatim}

\[
H(z) = \frac{-0.7143z}{(z-0.5)} + \frac{1.7143z}{(z+0.2)}
\]

\[
h(k) = -0.7143(0.5)^k + 1.7413(-0.2)^k
\]
(c) There are many methods for computing the output response. Using the `filter` function in MATLAB gives:

```matlab
num_tf = [1 -1 0];
den_tf = [1 -0.3 -0.1];
x = [1 3 6 -1 5];
y = filter(num_tf, den_tf, x)
y = 1.0000 2.3000 3.7900 -5.6330 4.6891
```

When using the filter function, the length of the output signal is always the same size as the length of the input signal. If more output values are desired, simply lengthen the input signal by padding with zeros as follows:

```matlab
x = [1 3 6 -1 5 zeros(1,5)];
y = filter(num_tf, den_tf, x)
y = 1.0000 2.3000 3.7900 -5.6330 4.6891 -4.1566
   -0.7781 -0.6491 -0.2725 -0.1467
```

**Comparison of IIR Filters and FIR Filters**

- **FIR filters** are always stable even when the filter coefficients are quantized. IIR filters may not be stable particularly when filter coefficients are quantized.
- IIR filters are generally of lower order than FIR filters.
- FIR filters have a linear phase response – IIR filters do not have a linear phase response.
- IIR filters are much more sensitive to coefficient word-length effects so choosing how to implement the filter (filter realization) is critical.
- The recursive property (feedback) of IIR filters creates an increased susceptibility to arithmetic overflow.

### 6.2 IIR FILTER DESIGN

There are two general approaches to designing digital IIR filters: indirect and direct. Both design methods are covered in this section.

**Indirect Design of IIR Filters**

The indirect design method involves first designing an analog filter to meet the filter specifications then transforming the analog filter to an “equivalent” digital filter as outlined in the following steps:

1. Convert the digital filter specifications to analog specifications.
2. Convert the analog specifications to Low-Pass (LP) prototype specifications.
3. Design the LP prototype analog filter (Butterworth, Chebychev I, Chebychev II, or Elliptic).
4. Transform the LP prototype filter to the desired analog filter.
5. Transform the analog filter to an “equivalent” digital filter.
Specific details on each step will not be provided because there are several excellent software tools including FDAT which complete these design steps automatically. However, a little background on this process, which appears somewhat convoluted, will be provided. In the early stages of digital filter design, there was a wealth of information about analog filter design: equations describing the frequency response of low-pass Butterworth, Chebychev, and Elliptic filters and equations for transforming among filter types (low-pass to high-pass or band-pass or notch). So, it made sense to design a good analog filter to meet the required specifications then transform that analog filter to an “equivalent” digital filter. The transform of choice is a Bilinear transform with pre-warping shown in Equation 6.3.

\[
H(z) = H(s)
\bigg|_{s} = \frac{2}{T_s \tan(w_n/T_s)} \left(\frac{z-1}{z+1}\right)
\]

\[T_s = 1/F_s = \text{sampling interval}\]
\[w_n = \text{desired frequency for exact match to analog filter} \]  

(6.3)

The analog filter transfer function, \(H(s)\), is converted to an equivalent digital filter transfer function, \(H(z)\), using the bilinear substitution for \(s\). Pre-warping ensures that the digital filter matches the analog filter at some specified desired matching frequency, \(w_n\). For low-pass or high-pass filters, the matching frequency would be the 3 dB cutoff frequency. For band-pass or notch filters, the matching frequency would be the center frequency.

The indirect method of design works very well for designing IIR digital filters. There are several excellent software tools including the Filter Design and Analysis Tool in MATLAB that utilize the indirect design method.

**Direct Design of IIR Filters**

The direct design method involves designing the digital filter directly (no analog intermediate design stage) to meet the filter specifications. There are two direct design algorithms available in MATLAB: `maxflat` and `yulewalk`. **Maxflat** directly designs a maximally flat digital Butterworth filter. The design algorithm can be run in the MATLAB workspace or in the Filter Design and Analysis Tool. The user must provide the order for the numerator, the order for the denominator, and the desired cut-off frequency. **Yulewalk** is a recursive filter design algorithm that uses the least squares method to match a digital filter of specified order \(N\) to a magnitude frequency response. The user provides a filter order, a vector of frequencies, and a vector of desired magnitudes at the specified frequencies. The `yulewalk` function then generates an IIR filter with the order specified that best matches the provided magnitude frequency response in a least squares sense. Both of these methods require the user to provide the filter order and will therefore require some experimentation on the part of the user to determine a reasonable filter order that successfully meets the filter specifications.
6.3 DESIGNING IIR FILTERS USING THE FDAT

The Filter Design and Analysis Tool was introduced in Chapter 5 for designing FIR filters. FDAT can also be used to design IIR filters: both indirect and direct designs. The filter types available are Butterworth, Chebychev I, Chebychev II, Elliptic, and Maximally flat. The filter type chosen will depend on the performance specifications for the filter.

Butterworth filters have a flat magnitude response in the passband and stopband but the transition region is wider than the transition region of Chebychev and Elliptic filters. Chebychev filters have a much sharper roll-off than Butterworth filters but exhibit ripple in either the passband (Chebychev I) or in the stopband (Chebychev II). Elliptic filters have the sharpest roll-off characteristics but exhibit equiripple in both the passband and stopband. Chebychev and Elliptic filters are generally of lower order than Butterworth filters and represent a better choice unless ripple in the passband and stopband are unacceptable. None of the IIR filters exhibit the ideal linear phase characteristics inherent in FIR filters, but Butterworth filters are the closest to a linear phase response.

Example 6.2: Design of Low-Pass IIR Filters using FDAT

(a) Design an IIR low-pass filter for each of the following filter types: Butterworth, Chebychev I, Chebychev II, and Elliptic to meet the following specifications:

i. Passband: 0 – 10 kHz
ii. Maximum Passband Ripple: 1 dB
iii. Stopband: \( \geq 12 \) kHz
iv. Minimum Stopband Attenuation: 60 dB
v. Sampling Frequency: 44.1 kHz

Make note of the filter order for each filter type. Plot the magnitude responses for all four filter types and compare them.

(b) Plot the phase response for the Butterworth and Elliptic filters. Which filter is closer to the ideal linear response?

(c) Find the difference equation, the transfer function, and the poles for the Elliptic filter.

Solution

(a) All four filter types are designed using FDAT in MATLAB. The screen shot for the Butterworth Design is shown in Figure 6.1. The remaining figures in this example are generated by clicking on the Full-View Analysis icon highlighted in Figure 6.1. This tool exports the data to a figure window in MATLAB that can be easily edited using plotting tools.

The minimum filter orders required to meet the given specifications are:

<table>
<thead>
<tr>
<th>Filter Type</th>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>Butterworth</td>
<td>27th</td>
</tr>
<tr>
<td>Chebychev I and II</td>
<td>11th</td>
</tr>
<tr>
<td>Elliptic</td>
<td>7th</td>
</tr>
</tbody>
</table>
The magnitude responses for each filter type are shown in Figure 6.2. The characteristics of each filter type are clearly illustrated. The Butterworth filter has a flat passband response with a gradual roll-off between passband and stopband. The Chebychev I filter exhibits some ripple in the passband but has a sharper roll-off than the Butterworth filter. The Chebychev II filter is flat in the passband but has ripple in the stopband. Roll-off is sharper than the Butterworth filter but not as steep as the Chebychev I filter. The Elliptic filter has the sharpest roll-off of the four filter types but exhibits ripple in both the passband and the stopband. All four filters meet the given filter specifications.
Chapter 6: Infinite Impulse Response (IIR) Filters

Figure 6.2a: Butterworth IIR Filter

Figure 6.2b: Chebyshev I IIR Filter
Figure 6.2c: Chebyshev II IIR Filter

Figure 6.2d: Elliptic IIR Filter

(b) The phase responses for the Butterworth and Elliptic Filters are shown in Figure 6.3. As expected, although neither filter has an ideal linear phase response, the Butterworth filter is closest to ideal.
(c) The filter coefficients for the Elliptic filter can be exported to the MATLAB workspace using the same Export feature used in Chapter 5 for FIR filters. Before exporting the coefficients, choose Edit → Convert to Single Section. The FDAT defaults to cascaded 2\textsuperscript{nd} order sections which will be discussed later in the chapter. For this example, a single 7\textsuperscript{th} order transfer function is required.

Once the design has been converted to a single section, select File → Export to export the filter coefficients to the MATLAB workspace. The default variables of Num and Den are fine for this example. Recall that the FIR filter only required a numerator because all of its poles are at the origin. This is not true for IIR filters so both a numerator and a denominator must be exported. The following MATLAB command can be used to generate a transfer function:

\[
\text{tf}(\text{Num, Den, } 1/44100)
\]
Transfer function:

\[
0.0194 z^7 + 0.04907 z^6 + 0.09724 z^5 + 0.1274 z^4 + 0.1274 z^3 + 0.09724 z^2 + 0.04907 z + 0.0194 \\
\frac{\text{z}^7 - 2.262 z^6 + 4.159 z^5 - 4.745 z^4 + 4.197 z^3 - 2.6 z^2 + 1.108 z - 0.2716}{\text{z}^7}
\]

Sampling time: 2.2676e-005

The difference equation can be determined from the transfer function or simply from the exported numerator and denominator:

\[
y(k) = 0.0194x(k) + 0.04907x(k - 1) + 0.09724x(k - 2) + 0.1274x(k - 3) + 0.1274x(k - 4) \\
+ 0.09724x(k - 5) + 0.04907x(k - 6) + 0.0194x(k - 7) + 2.262y(k - 1) - 4.159y(k - 2) \\
+ 4.745y(k - 3) - 4.197y(k - 4) + 2.6y(k - 5) - 1.108y(k - 6) + 0.2716y(k - 7)
\]

Notice the denominator coefficients are negated because the past output terms are being moved to the right side of the difference equation.

The poles are the roots of the denominator of the filter and are determined using `roots` in MATLAB:

```matlab
poles = roots(Den)
poles =
0.1433 + 0.9663i
0.1433 - 0.9663i
0.2339 + 0.8741i
0.2339 - 0.8741i
0.6132
0.4470 + 0.6058i
0.4470 - 0.6058i
```

```matlab
mag_poles = abs(poles)
mag_poles =
0.9769
0.9769
0.9049
0.9049
0.6132
0.7528
0.7528
```

Notice that while the filter is stable (all poles have magnitude less than one) some of the poles are quite close to the edge of the unit circle. Any rounding of the filter coefficients will cause these poles to drift – potentially outside the unit circle which would make the filter unstable.
Example 6.3: Design of a Band-Pass IIR Filter using FDAT

(a) Design an IIR elliptic band-pass filter to meet the following specifications. Determine the required filter order and plot the frequency magnitude response.

i. Passband: 8 – 12 kHz
ii. Maximum Passband Ripple: 0.1 dB
iii. Stopband: < 7 kHz and > 13 kHz
iv. Minimum Stopband Attenuation: 30 dB
v. Sampling Frequency: 48 kHz

(b) Compare this design to the optimal FIR elliptic filter designed in Chapter 5, Example 5.8.

Solution

(a) The band-pass IIR elliptic filter is designed using FDAT. The minimum filter order to meet the given specifications is 10th order. The magnitude and phase responses are shown in Figure 6.4.

![IIR Elliptic Filter - Magnitude Response](image)

![IIR Elliptic Filter - Phase Response](image)

Figure 6.4: 10th Order IIR Elliptic BPF: Magnitude and Phase Response
(b) An FIR Equiripple filter with the same specifications was designed in Example 5.8. The minimum filter order to meet the given specifications was 87th order. The magnitude and phase responses are shown in Figure 6.5.

The IIR filter is able to meet the given specifications with a much lower filter order and therefore less computational complexity than the FIR filter. Both filters exhibit a small amount of ripple in the passband (within specs) and ripple in the stopband. The phase response of the FIR Equiripple filter is linear (ideal); whereas, the phase response of the IIR Elliptic filter is very nonlinear. The IIR filter would therefore be an inappropriate choice in an application such as video where signal phase information is important.

Figure 6.5: 87th Order FIR Equiripple BPF: Magnitude and Phase Response
Challenge Question 6.1
How many multiplications per sampling instant must be performed with the FIR and IIR filters designed in Example 6.3? How many multiplications per second (MPS) must be performed for each of these filters?

6.4 FILTER REALIZATIONS

Three different filter realizations were described in Chapter 5 for FIR filters: Direct, Direct-Transposed, and Direct-Symmetric. The Direct and Direct-Transposed require \( N+1 \) multiply and accumulate taps for an \( N^{th} \) order filter and differ only in how the filter is implemented on a DSP processor or an FPGA. The Direct-Symmetric realization cuts the number of multiplies in half by taking advantage of the symmetry of the filter coefficients. Scaling can be added to eliminate or minimize arithmetic overflow. There are several options for implementing IIR filters also.

**Single Stage (Direct)**
IIR filters can be implemented directly as shown in Figure 6.6 for a 4\(^{th} \) order filter. A higher order filter would simply require additional feedforward and feedback loops with multipliers and adders. **Direct realizations of IIR filters are extremely sensitive to wordlength effects (coefficient quantization)!** Therefore, IIR filters above 3\(^{rd} \) order are not usually implemented in direct form (one single section), but are split into 2\(^{nd} \) order sections instead.

![Figure 6.6: Direct Realization of IIR Filter](image-url)
Chapter 6: Infinite Impulse Response (IIR) Filters

**Cascade with 2nd Order Sections**

A cascade realization breaks the IIR filter into the product of several 2nd order sections as follows:

\[ H(z) = \frac{\text{Num}(z)}{\text{Den}(z)} = G \left( \frac{N_1(z)}{D_1(z)} \frac{N_2(z)}{D_2(z)} \cdots \frac{N_M(z)}{D_M(z)} \right) \]

\[ H(z) = G \left( \frac{1 + b_{11} z^{-1} + b_{21} z^{-2}}{1 + a_{11} z^{-1} + a_{21} z^{-2}} \right) \left( \frac{1 + b_{12} z^{-1} + b_{22} z^{-2}}{1 + a_{12} z^{-1} + a_{22} z^{-2}} \right) \cdots \left( \frac{1 + b_{1M} z^{-1} + b_{2M} z^{-2}}{1 + a_{1M} z^{-1} + a_{2M} z^{-2}} \right) \]  \hspace{1cm} (6.4)

A block diagram realization of the cascade structure is shown in Figure 6.7 for a 4th order filter which of course requires only two 2nd order sections. Additional 2nd order sections are easily added for higher order filters.

There is no unique way to break a transfer function into the product of 2nd order sections. Nth order IIR filters have N poles and N zeros. 2nd order sections are created by picking two of the poles and two of the zeros and pairing them together. There are [(N/2)!]^2 different ways to break an Nth order transfer function into 2nd order sections. Although mathematically equivalent, the various options are not equivalent in terms of filter performance. Some general guidelines for choosing the 2nd order sections to maximize filter output signal to noise ratio are:

1. Pair the poles and zeros that are closest to one another together. This will help minimize the amplitude gain of the filter at frequencies close to the section poles.
2. Put the most stable 2nd order section first and end with the least stable 2nd order section (i.e., the section with poles closest to the edge of the unit circle).

Fortunately, there are several software algorithms (including the Filter Design and Analysis Tool) that create an optimal arrangement of 2nd order sections based on maximizing the filter signal-to-noise ratio (SNR). Examples are included later in the chapter.

Figure 6.7: Cascade Realization of IIR Filter
Parallel with 2nd Order Sections
A parallel realization breaks the IIR filter into the sum of 2nd order sections using partial fraction expansion as shown in Equation 6.5. A block diagram realization of the parallel structure is shown in Figure 6.8 for a 4th order filter. Additional 2nd order sections are easily added for higher order filters.

\[
H(z) = \frac{\text{Num}(z)}{\text{Den}(z)} = G + \left( \frac{N_1(z)}{D_1(z)} \right) + \left( \frac{N_2(z)}{D_2(z)} \right) + \cdots + \left( \frac{N_M(z)}{D_M(z)} \right)
\]

\[
H(z) = G + \left( \frac{b_{01} + b_{11}z^{-1}}{1 + a_{11}z^{-1} + a_{21}z^{-2}} \right) + \left( \frac{b_{02} + b_{12}z^{-1}}{1 + a_{12}z^{-1} + a_{22}z^{-2}} \right) + \cdots + \left( \frac{b_{0M} + b_{1M}z^{-1}}{1 + a_{1M}z^{-1} + a_{2M}z^{-2}} \right) \tag{6.5}
\]

Figure 6.8: Parallel Realization of IIR Filter
How does the parallel realization compare to a cascade realization?
- In cascade structures, the poles are paired with the closest zeros to minimize the section gain at frequencies close to the pole locations.
- Ordering of the 2nd order sections is not important for a parallel realization because each section feeds directly to the output. In cascade structures, where each section feeds into the next section, it is important to put the least stable section last.
- The signal-to-noise ratio (SNR) of a parallel structure is comparable to the best (optimized) cascade structure.
- The zero locations in a parallel structure are very sensitive to coefficient quantization errors unless the wordlength is 12 or more bits.
- In a cascade structure, 25-50% of the filter coefficients are simple integers when the filter is designed using a bilinear transformation of an analog filter.
- The Filter Design and Analysis Tool will create a cascade structure for IIR filters but does not provide a parallel realization.

### 6.5 PRACTICAL CONSIDERATIONS FOR IIR FILTERS

**Word-length Effects (Filter Coefficient Quantization Errors)**

In order to implement a filter on a DSP processor, the filter coefficients must be stored in memory, either in floating-point or fixed-point format depending on the processor. In Chapter 4, it was shown through several examples that quantization error often occurs when representing numbers in a specified format. The filter coefficients derived using the FDAT (or some other design tool) will very likely be quantized when stored in the memory of a DSP processor. In Chapter 5, the effects of coefficient quantization errors on filter performance were illustrated through several examples for FIR filters. IIR filters are considerably more sensitive to wordlength effects than FIR filters due to the recursive nature (feedback) of IIR filters. In digital filter design, it is absolutely necessary to check the effect of filter coefficient quantization on the performance of a filter! The next example illustrates how quantization of the filter coefficients can impact filter performance.

**Example 6.4: Coefficient Word-Length Effects on IIR Filter Performance**

Consider the 10th order IIR elliptic band-pass filter designed in Example 6.3 to meet the specifications:

i. Passband: 8 – 12 kHz
ii. Maximum Passband Ripple: 0.1 dB
iii. Stopband: ≤ 7 kHz and ≥ 13 kHz
iv. Minimum Stopband Attenuation: 30 dB
v. Sampling Frequency: 48 kHz

(a) Using a direct realization, determine the minimum number of bits required for the filter coefficients in order to keep the filter stable.
(b) Plot the filter magnitude response when the filter coefficients are quantized to the number of bits determined in part (a).
(c) Determine the minimum number of bits required to meet the given filter specifications and plot the corresponding magnitude response.
Solution
(a) Enter the specifications for the elliptic band-pass filter into the FDAT, select minimum order, and then click Design Filter to create a $10^{th}$ order elliptic band-pass filter. Since the FDAT defaults to a cascade structure, choose Edit → Convert to Single Section then File → Export to create a direct realization of the filter with the numerator (Num) and denominator (Den) exported to the MATLAB workspace. The filter coefficients are:

Num'
\[
0.03203191821694 \\
-0.05878107521081 \\
0.10186869443828 \\
-0.09766972701608 \\
0.07231401919473 \\
0.00000000000000 \\
-0.07231401919473 \\
0.09766972701608 \\
-0.10186869443828 \\
0.05878107521081 \\
-0.03203191821694
\]

Den'
\[
1.00000000000000 \\
-2.35045087132525 \\
6.00883384914710 \\
-8.37048075001733 \\
11.6691877878540 \\
-10.76965153277673 \\
9.81514391724758 \\
-5.90678320013480 \\
3.56194382465704 \\
-1.15506123192936 \\
0.41170437266444
\]

The filter poles are the roots of the denominator and the magnitudes of the poles are:

abs(roots(Den))
\[
0.97875292352670 \\
0.97875292352670 \\
0.90061863412144 \\
0.90061863412144 \\
0.98170470724316 \\
0.98170470724316 \\
0.91188762957079 \\
0.91188762957079 \\
0.81312355369549 \\
0.81312355369549
\]
The filter is stable since all poles have magnitude strictly less than one; however, many of the filter poles are very close to the edge of the unit circle.

In order to determine the minimum number bits required to maintain filter stability, the denominator coefficients must be quantized to a specified number of bits and the magnitude of the roots of the quantized denominator must be checked. Since this will require some experimentation with the number of bits, it is worthwhile to write a script m-file for this part with the following code:

```matlab
B=9;  
Fs=48000;  
Denq=round(Den*2^B)/(2^(B-1));  
abs(roots(Denq))  
Numq=round(Num*2^B)/(2^(B-1));  
[Hq,fq]=freqz(Numq,Denq,500,Fs);  
[H,f]=freqz(Num,Den,500,Fs);  
plot(f/1000,20*log10(abs(H)),fq/1000,20*log10(abs(Hq)));  
legend('Unquantized Response','Quantized Response');  
xlabel('Frequency (kHz)'); ylabel('Filter Magnitude (dB)');  
title(['IIR Filter Magnitude Response (' num2str(B) ' bits)']); grid
```

This m-file allows the user to specify a number of bits (B), quantizes the denominator to B bits, displays the magnitude of the poles of the quantized denominator, quantizes the numerator of the filter to B bits, and then plots the frequency response of the quantized filter and the original (unquantized) filter. Changing the number of bits, B, in Line 1 and then clicking on Run will re-run the code for any specified number of bits.

The magnitudes of the poles for B = 9 bits are:

0.99244484746493  
0.99244484746493  
0.98946521292787  
0.98946521292787  
0.88479603523624  
0.88479603523624  
0.90496389777822  
0.90496389777822  
0.81450421020215  
0.81450421020215

Since all poles are within the unit circle, the IIR filter is stable when quantized to 9 bits.

The magnitudes of the poles for B = 8 bits are:

1.01909858840302  
1.01909858840302
The IIR filter is unstable when quantized to 8 bits. So, the minimum number of bits for stability is 9 bits.

(b) The minimum number of bits for stability was determined to be 9 bits. The m-file created for part (a) also includes the code to plot the frequency response (magnitude) of the quantized filter. Setting $B = 9$ and running the m-file creates the frequency response plot shown in Figure 6.9. Notice that even though the IIR filter is stable when the coefficients are quantized to 9 bits, the performance of the quantized filter does not meet the specifications and is therefore unacceptable!

![IIR Filter Magnitude Response (9 bits)](image)

**Figure 6.9: Frequency Response Quantized to 9 bits (Direct Realization)**

(c) The number of bits required to meet the specifications can be determined by experimenting with the number of bits, $B$, in the m-file from part (a) till the performance of the quantized filter matches that of the un-quantized filter. After some experimentation, 16 bits for the filter coefficients results in a filter which meets the given specification as shown in Figure 6.10.
In example 6.4, the poles of the IIR filter shift when the coefficients are quantized to $B$ bits and it is even possible to make the filter unstable. Will an FIR filter exhibit the same instability problem if the number of bits for the filter coefficients is continually reduced? Explain.

In the previous section on filter realizations, it was mentioned that breaking IIR filters into $2^{\text{nd}}$ order sections (either cascade or parallel) makes the filter less sensitive to word-length effects. The FDAT has the capability to create a cascade realization for IIR filters, in fact, it defaults to this structure. When File $\rightarrow$ Export is selected, the FDAT exports an SOS matrix and a G vector to the MATLAB workspace. The G (Gain) vector is the gain for each section. The SOS (Second Order Sections) matrix contains the numerator and denominator coefficients of each $2^{\text{nd}}$ order section and is structured as follows:

$$SOS = \begin{bmatrix} Num1 & Den1 \\ Num2 & Den2 \\ Num3 & Den3 \\ Num4 & Den4 \end{bmatrix} = \begin{bmatrix} 1 & b_{11} & b_{21} & 1 & a_{11} & a_{21} \\ 1 & b_{12} & b_{22} & 1 & a_{12} & a_{22} \\ 1 & b_{13} & b_{23} & 1 & a_{13} & a_{23} \\ 1 & b_{14} & b_{24} & 1 & a_{14} & a_{24} \end{bmatrix}$$

Row 1 of the SOS matrix contains the numerator and denominator coefficients for the first $2^{\text{nd}}$ order section. The next $2^{\text{nd}}$ order section coefficients are located in Row 2 and the pattern continues. This particular SOS matrix has only 4 rows and therefore describes an $8^{\text{th}}$ order filter (4 sections). Higher order filters would simply have additional rows in the SOS matrix. In MATLAB, specific sections can be referenced as follows:

$$\text{Den3} = \text{SOS}(3, 4:6); \quad \% \text{Row 3, Columns 4-6 of the SOS Matrix}$$
The next example illustrates the advantage of implementing IIR filters as 2nd order sections rather than using a direct (single stage) realization.

**Example 6.5: Using Cascade Realization to Reduce Effects of Coefficient Word-Length**

Repeat Example 6.4 using a cascade realization for the IIR elliptic band-pass filter and compare the results to those using a direct realization for the filter.

**Solution**

Enter the specifications for the elliptic band-pass filter into the FDAT, select minimum order, and then click Design Filter to create the 10th order elliptic band-pass filter. Convert the structure to Direct Form II Second Order Sections. The structure for the filter is now five 2nd order cascaded sections. Select File → Export to export the SOS matrix and G vector to the MATLAB workspace. The resulting SOS matrix and G vector can be viewed by typing `SOS` and `G` at the MATLAB prompt:

```
SOS =
    1.0000   -0.0000   -1.0000    1.0000   -0.4451    0.6612
    1.0000    0.3987    1.0000    1.0000   -0.0653    0.8111
    1.0000   -1.2720    1.0000    1.0000   -0.8654    0.8315
    1.0000    0.1385    1.0000    1.0000    0.0283    0.9580
    1.0000   -1.1004    1.0000    1.0000   -1.0030    0.9637

G =
    0.0320
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
```

To determine the minimum number of bits for stability, the denominator coefficients of each 2nd order section must be quantized and the poles must be checked to ensure the magnitude of each pole is less than one. This will require some experimentation on the number of bits so it is worthwhile to write an m-file for this example. Figure 6.11 shows the code for the script file.

This script first quantizes the SOS matrix and the G vector to B bits as specified by the user. The magnitudes of the poles of each quantized section are displayed in a matrix called `mag_poles`. In order to plot the frequency response, a single numerator and denominator are created by multiplying the numerators and denominators of each section together. For comparison purposes, this is done for both the quantized and un-quantized filter. The frequency response (magnitude) is then calculated and plotted.
B = 4;
Fs=48000;
% Quantize the SOS Matrix and the Gain, G to B bits
SOS_q=round(SOS*2^(B-1))/(2^(B-1));
G_q=round(G*2^(B-1))/(2^(B-1));
% Check stability of the Quantized Denominator of each section
rows=size(SOS,1);
for m=1:rows,
    mag_poles(m,1:2)=abs(roots(SOS_q(m,4:6)));
end
% Stability Check:
mag_poles

% Check filter performance for quantized filter coefficients
% Create a single numerator and denominator by convolving the
% numerators and denominators of each section together.
Num=prod(G); Den=1;
Numq=prod(G_q); Denq=1;
for m=1:rows,
    Num=conv(Num,SOS(m,1:3));
    Den=conv(Den,SOS(m,4:6));
    Numq=conv(Numq,SOS_q(m,1:3));
    Denq=conv(Denq,SOS_q(m,4:6));
end
[H f] = freqz(Num,Den,500,Fs);
[Hq fq]= freqz(Numq,Denq,500,Fs);
plot(f/1000, 20*log10(abs(H)),fq/1000, 20*log10(abs(Hq)));
legend('Unquantized Response','Quantized Response');
xlabel('Frequency (kHz)'); ylabel('Filter Magnitude (dB)');
title(['IIR Filter Magnitude Response (' num2str(B) ' bits)']);
grid

**Figure 6.11: m-file (IIR_Quant.m) for Example 6.5**

When the code in Figure 6.11 is executed for B = 4 bits, the magnitude of the sections poles are:

```
mag_poles =
    0.79056941504209   0.79056941504209
    0.86602540378444   0.86602540378444
    0.93541434669349   0.93541434669349
    1.00000000000000   1.00000000000000
    1.00000000000000   1.00000000000000
```

The last two sections of the filter are unstable since the poles have a magnitude of one. The IIR filter is therefore unstable when the coefficients are quantized to 4 bits.
Changing B to 5 bits and re-running the code yields section poles with the following magnitudes:

\[
\text{mag}_\text{poles} = \\
0.82915619758885 \quad 0.82915619758885 \\
0.90138781886600 \quad 0.90138781886600 \\
0.90138781886600 \quad 0.90138781886600 \\
0.96824583655185 \quad 0.96824583655185 \\
0.96824583655185 \quad 0.96824583655185 
\]

Since all of the poles have magnitude less than one, the IIR filter is stable when the coefficients are quantized to 5 bits. For a direct realization (Example 6.4), 9 bits were required to ensure stability of the IIR filter. Notice also that the most stable section is first (row 1) and the least stable section is last (row 5).

Setting B = 5 and running the code in Figure 6.11 creates the frequency response plot shown in Figure 6.12. Notice that even though the IIR filter is stable when the coefficients are quantized to 5 bits, the performance of the quantized filter does not meet the specifications and is therefore unacceptable.

![Figure 6.12: Frequency Response Quantized IIR Filters (Cascade Realization)](image)

The number of bits required to meet the specifications can be determined by experimenting with the number of bits, B, and re-running the code until the performance of the quantized filter matches that of the un-quantized filter. After some experimentation, 9 bits for the filter coefficients results in a filter which meets the given specification as shown in Figure 6.13. The direct realization required 16 bits to meet the specifications. Clearly, a cascade realization is significantly less sensitive to word-length effects than a direct realization!
**Roundoff Errors in Arithmetic Operations**

The output signal of an IIR filter is computed by:

\[
y(k) = \sum_{i=1}^{N} -a_i y(k-i) + \sum_{i=0}^{N} b_i x(k-i)
\]

or

\[
y(k) = -a_1 y(k-1) - a_2 y(k-2) - \ldots - a_N y(k-N) + b_0 x(k) + b_1 x(k-1) + \ldots + b_N x(k-n)
\]  \hspace{1cm} (6.6)

Suppose the filter coefficients are 16 bits and the processor utilizes a 16-bit ADC. Each multiplication will be 32 bits. Eventually, this product must be truncated or rounded to 16-bits. Rounding the result will result in a reduction of the Signal-Noise Ratio (SNR). To minimize the effect of rounding, it is much better to store each product in a double precision register and round after the final summation instead of rounding to 16 bits after each individual multiplication.

**Arithmetic Overflow in Fixed Point Processors**

As discussed in Chapter 4, fixed point digital signal processors typically use a Q15 format in which all numbers are scaled to be between −1 and +1. Any multiplication operations on numbers in Q15 format result in a number that is also between −1 and +1 (no overflow). However, addition operations on numbers in Q15 format can result in overflow. Arithmetic overflow will cause either a sign change in the result (undesirable) or clipping (saturating at the maximum or minimum value). Arithmetic overflow can be a messier problem for IIR filters because the feedback of IIR filters can cause an overflow limit cycle which can be difficult to
stop. Arithmetic overflow can be eliminated (or significantly reduced) by scaling the filter coefficients and the input data.

In Figure 6.14, scaling for one 2nd order section of an IIR filter is illustrated. The following observations on the effectiveness of scaling can be made:

- Dividing the input signal by the scale factor, s, reduces the input in order to either eliminate or minimize overflow through the feedback terms.
- Multiplying the numerator coefficients by the scale factor, s, ensures that the overall filter gain is not changed.
- Scaling will reduce the signal to noise ratio (SNR).

![Figure 6.14: Scaling a 2nd Order Section of an IIR Filter](image)

How is the scale factor chosen? In Chapter 5, only $L_1$ scaling was explored for FIR filters. There are actually three common scaling schemes used for FIR and IIR filters: $L_1$, $L_2$, and $L_\infty$ scaling. Referring to Figure 6.14, the transfer function from the input $x(k)$ to the intermediate node signal $w(k)$ is:

$$H(z) = \frac{W(z)}{X(z)} = \frac{1}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

(6.7)

Letting $h(k)$ denote the impulse response (that is, $h(k) = Z^{-1}[H(z)]$), the three different scale factors are determined as follows:
\[ s = \sum_{k=0}^{\infty} |h(k)| = \|H(z)\|_1 \]

\[ s = \sqrt{\sum_{k=0}^{\infty} h^2(k)} = \|H(z)\|_2 \]

\[ s = \max |H(e^{j\omega})| = \|H(z)\|_\infty \]

How do these scale factors compare in terms of filter performance?

- `\|_{L_1}` scaling ensures that there will never be arithmetic overflow because this scale factor sets the overall filter gain from \(x(k)\) to \(w(k)\) equal to one. `\|_{L_1}` scaling results in the largest scale factor of the three schemes and therefore has the worst SNR performance. `\|_{L_1}` scaling is generally more drastic than what is required to keep arithmetic overflow in check.

- `\|_{L_2}` scaling places an energy constraint on the input signal and the transfer function. Arithmetic overflow is reduced but will occur sometimes. `\|_{L_2}` scaling results in the smallest scale factor of the three schemes and offers better SNR performance than `\|_{L_1}` scaling.

- `\|_{L_\infty}` scaling ensures that arithmetic overflow will not occur when the input signal is a sine wave. The scale factor using `\|_{L_\infty}` scaling lies between the `\|_{L_2}` scale factor and `\|_{L_1}` scale factor and is typically the best choice, offering a compromise between the other two schemes. Arithmetic overflow is not completely eliminated but is significantly reduced and this scaling scheme results in better SNR performance than `\|_{L_1}` scaling.

- The FDAT tool has the capability of creating filter realizations with either `\|_{L_\infty}` scaling or `\|_{L_2}` scaling. It does not offer an option for `\|_{L_1}` scaling. Choose Edit→Re-order and Scale Second Order Sections to create a scaled, cascade filter realization.

Figure 6.15 shows how scaling is implemented in a cascade realization for an IIR filter. The scale factor for the \(i\)th section, \(s_i\), is determined by using the `\|_{L_1}`, `\|_{L_2}`, or `\|_{L_\infty}` norm as follows:

\[ s_i = \|H_i(z)\|_p \quad p = 1, 2, \text{or } \infty \]

\[ H_i(z) = \frac{\prod_{m=1}^{i-1} T_m(z)}{1 + a_{1i}z^{-1} + a_{12}z^{-2}} \quad T_m(z) = \frac{b_{0m} + b_{1m}z^{-1} + b_{2m}z^{-2}}{1 + a_{1m}z^{-1} + a_{2m}z^{-2}} \]
In equation 6.9, $H_i(z)$ is the transfer function from the input, $x(k)$, to the intermediate node signal $w_i(k)$ for section $i$. So, the scaling factor for a particular section depends not only on the section of interest but also on all prior sections which certainly complicates matters.

**Figure 6.15: Scaling in a Cascade Structure**

In Figure 6.16, scaling is added to a parallel realization for an IIR filter. Here, the scale factor is determined independently for each 2nd order section using an $\ell_1$, $\ell_2$, or $\ell_\infty$ norm. Unlike the cascade structure, it is not necessary to take any of the other sections into account because each section feeds directly to the output signal from the input signal. Therefore, the scaling factors are much simpler to calculate for a parallel realization.
6.6 EXAMPLES OF IIR FILTERS

Fading Echo (Feedback Comb Filter)
An echo filter feeds back an attenuated output signal to produce a fading echo effect as follows:

\[ y(k) = x(k - N) + \alpha_N y(k - N) \quad 0 < \alpha_N < 1 \]  \hspace{1cm} (6.10)
The difference equation models a sound wave bouncing off a wall or surface and returning to the ear. The constant $\alpha$ is the round-trip attenuation factor and $N*Ts$ is the round-trip time delay. The transfer function of the echo filter is

$$H(z) = \frac{1}{1-\alpha_N z^{-N}} = \frac{z^N}{z^N - \alpha_N}$$

(6.11)

**Challenge Question 6.3**
What is the impulse response of the echo filter?

The frequency response of the echo filter is shown in Figure 6.17 for several values of gain, $\alpha$. It is interesting to note that the frequency response of this echo filter is the inverse of the feedforward comb filter discussed in Chapter 5. The feedforward comb filter rejects a specific frequency and its harmonics; whereas, the feedback comb filter amplifies a specific frequency and its harmonics.

![Figure 6.17: Frequency Response of Echo Filter](image)

**Challenge Question 6.4**
What will happen if the feedback factor, $\alpha_N$, is chosen to be greater than one for the echo filter? Why?

**All-Pass Filter**
The block diagram for an all-pass IIR filter is shown in Figure 6.18. All-pass filters are useful for creating digital audio effects such as reverb and phasing. All-pass filters can also be used to re-shape the phase of an IIR filter to reduce phase distortion.
The difference equation for the filter is

\[ y(k) = -\alpha x(k) + x(k - N) + \alpha y(k - N) \quad \text{where} \quad 0 < \alpha < 1 \quad (6.12) \]

The impulse response of the all-pass filter is sketched in Figure 6.19.

The transfer function of this filter is

\[ H(z) = \frac{-\alpha + z^{-N}}{1 - \alpha z^{-N}} = \frac{-\alpha z^N + 1}{z^N - \alpha} \quad (6.13) \]

The frequency response of a 5th order all-pass filter with \( \alpha = 0.3 \) can be computed then plotted in MATLAB using the following commands:

\[ \text{alpha} = 0.3; \]
\[ N = 5; \]
num = zeros(1,N+1); num(1) = -alpha; num(N+1) = 1;
den = zeros(1,N+1); den(1) = 1; den(N+1) = -alpha;
freqz(num,den,200,1000);
% 3rd argument in freqz is number of data points to
generate and the 4th argument in freqz is the sampling
frequency

The resulting frequency response is plotted in Figure 6.20. Not surprisingly, the gain of the filter
is 0 dB or 1.0 over the entire frequency range for the filter. The phase response is non-ideal and
introduces some distortion.

![Frequency Response of All-Pass Filter](image)

**Figure 6.20: Frequency Response of All-Pass Filter**

### 6.7 APPLICATION: AUDIO EFFECTS USING IIR FILTERS

**Phasing**

A phase shifter or phaser notches one or more frequencies in the original audio signal by filtering
the audio signal through a chain (cascade) of first order all-pass filters, then mixing the filtered
signal with the original audio signal as shown in Figure 6.21.

So where do the notches occur in the phaser? A notch will occur at all the frequencies at which
the all-pass filter chain produces a phase shift of 180° or an odd multiple of 180°. The 180°
phase shift inverts the signal frequency which is then re-combined with the original audio signal
through the mixer resulting in complete cancellation at the frequency if the depth (gain) is
chosen to be one, or partial cancellation for smaller gains. The all-pass filter chain can be
designed to control which frequencies are notched, the width of the notch, and how the notches vary or sweep over time.

A 1\textsuperscript{st} order all-pass filter is shown in Figure 6.22. Figure 6.23 shows the frequency response of a chain of four 1\textsuperscript{st} order all-pass filters, and the frequency response of the resulting 4-chain phaser with the depth set to one. In Figure 6.23(a), the gains of all four all-pass filters are set to 0.2; while in Figure 6.23(b), the gains of all four filters are set to 0.8.

As indicated in Figure 6.23, a chain of four 1\textsuperscript{st} order all-pass filters will have a phase of 180° at two frequencies, and will cause inversion of the audio signal at these two frequencies. The resulting notch in the output audio signal at these two frequencies is evident in the magnitude plot for the phaser in Figure 6.23. The phaser will boost signal level at frequencies that are not in the vicinity of the notch frequencies. Smaller values for alpha spread and increase the notch frequencies, while larger values of alpha compress and reduce the notch frequencies.

Phasing is a more sophisticated version of the flanging effect introduced in Chapter 5. With flanging, the notch frequencies are all evenly spaced, and are swept or varied using a low frequency oscillator to vary the time delay. With phasing, the notch frequencies are controlled by varying the gain, \( \alpha \), in each of the all-pass filters in the chain. The notch frequencies are not evenly spaced, and the frequencies are often swept exponentially resulting in small variations
initially, followed by faster changes in notch frequencies as time progresses. Another interesting effect can be achieved by stepping the notch frequencies up and down rather than varying them continually.

Commercial analog phaser pedals for guitars typically allow the musician to select either a four chain or an eight chain phaser, and have control knobs for speed of sweep and depth. Some also offer control knobs for resonance and a step option for the notch frequencies. Some commercial digital phasers offer four, eight, ten, or twelve chain options.

Figure 6.23: Frequency Response of All-Pass Filter Chain and Phase

(a) $\alpha = 0.2$

(b) $\alpha = 0.8$
Reverb
Reverb is a series of tightly spaced echoes that occur when sound travels in an acoustic environment (room, auditorium, or concert hall) to a listener. Sound waves travel not only directly to the listener but also as reflections off of the walls and ceiling. Reverb is affected by the dimensions of a room, by the materials the walls are made of, whether or not the room has carpet, and even by the number of people in the room. Figure 6.23 illustrates what the impulse response of a room might look like.

As illustrated in Figure 6.24, reverb is characterized by early reflections representing single reflections off of the walls followed by diffuse reverberation (very closely packed reflections coming to the listener from multiple directions). Because there are so many reflections very tightly packed, individual reflection or echo signals are not distinguishable by the ear, but are instead mixed together forming the overall reverb effect.

![Illustration of the Impulse Response of a Room](image)

**Figure 6.24: Impulse Response of a Room (Reverb)**

The key parameters for reverb are the reverb time, the early reflection pattern, the pre-delay time, and the high frequency damping. The reverb time, also known as overall decay time, is the amount of time it takes for the echo signals to drop 60 db below or 1/10,000 of the original sound level. Reverb time is dependent both on the size of the space and the surfaces within the space. Materials such as glass, brick or marble increase reverb time, while drapes, carpet, and people reduce reverb time. Larger rooms tend to have longer reverb times. For a large concert hall, reverb time would typically be between 1.5 and 2 seconds. The early reflection pattern is the first reflections of the original sound off the walls. The pre-delay time is the amount of time before these first reflections reach the listener. Longer pre-delay time and larger spacing between early reflections are indicative of a larger room or space. High frequency damping models the frequency-dependent effect of humidity and surfaces on signal absorption. Higher frequencies attenuate more rapidly than lower frequencies.
There are many audio effects products on the market that reproduce the natural reverb in an actual physical space by digitally adding reverb to music. A block diagram of a generic digital reverb filter, first developed and refined by Manfred R. Schroeder in the 1960s and 1970s, is shown in Figure 6.25. Schroeder’s work still forms the basis for reverb filter design. The parallel bank of comb filters produces the early reflection effect. The delays and gains are chosen to simulate the physical dimensions and materials of a specific room. Each filter has a different gain and the delays are chosen to be mutually prime. The cascaded all-pass filters form the diffuser.

Comb (Echo) filters and all-pass filters were introduced in the previous section as special types of IIR filters. The difference equation for a feedback comb filter (repeated from equation 6.10) is given by:

\[ y(k) = x(k - N) + \alpha_N y(k - N) \quad 0 < \alpha_N < 1 \]  

The difference equation models a sound wave bouncing off a wall or surface and returning to the ear. The constant \( \alpha_N \) is the round-trip attenuation factor and \( N^*Ts \) is the round-trip time delay. In order to maintain stability the attenuation constant, \( \alpha_N \), must be chosen less than one; otherwise, the output signal will continue to grow in magnitude eventually causing the sound system to saturate.

The impulse response for a comb filter is shown in Figure 6.26. The figure indicates that the comb filter produces a series of decaying echoes spaced apart by the time delay \( \tau = N^*Ts \). This time delay is also referred to as the comb filter loop time. Loop times in the vicinity of 50 ms are used to create the reverb effect in a large concert hall, while shorter loop times (around 10 ms) would simulate the reverb heard in a small tiled shower. It is important that the loop times for the comb filters are all mutually prime; otherwise, echoes from two or more of the comb filters will occur at the same time, get added at the summer, and produce an undesirable echo spike.
Chapter 6: Infinite Impulse Response (IIR) Filters

As illustrated in Figure 6.26, choosing the attenuation constant, \( \alpha_N \), close to one will produce a series of echoes that persist for a long period of time; whereas, choosing \( \alpha_N \) close to zero will produce a quickly diminishing series of echoes. Given the comb filter loop time, \( \tau \), and the total desired reverb time, \( T_R \), the attenuation constant is calculated as:

\[
\alpha_N = 0.001^{\tau/T_R}
\]

The difference equation for an all-pass filter (repeated from equation 6.12) is:

\[
y(k) = -\alpha x(k) + x(k - N) + \alpha y(k - N) \quad 0 < \alpha < 1
\]

As discussed previously, an all-pass filter has a magnitude of one at all frequencies (hence the name all-pass) but does introduce phase distortion. As mentioned earlier in this section, one application for all-pass filters is to create the phasing effect. In a reverb filter, the all-pass filters are the diffusers creating the closely spaced indistinguishable echoes that enrich the sound. Typically, there would be 1000-3000 reflections per second, and the spacing between these reflections would be random. Fewer reflections would make echoes distinguishable, while non-random spacing between reflections creates an undesirable metallic ringing effect. The loop times (or delay) for the all-pass filters is considerably shorter than the loop times for the comb filters.

**Challenge Question 6.5**

It is possible to create a reverb filter using a digital FIR filter instead of the IIR echo and all-pass filters. Explain how and discuss the obvious disadvantage of an FIR filter.

Modern reverb filters are based on Schroeder’s initial design ideas but are considerably more sophisticated. Impulse responses of spaces can be measured and matched. Designs and algorithms vary from manufacturer to manufacturer and are typically proprietary. Figure 6.27 illustrates a more sophisticated reverb design that uses an FIR filter to produce the early reflections, banks of comb and all-pass filters for diffusers, and low-pass or equalization filters to model frequency dependent absorption effects.
Example 6.6: Design of a Reverb Filter
This example utilizes the parameters of the reverb filter described in Dodge and Jerse; namely, the filter loop times and total desired reverb time suggested for each of the filters to simulate a medium-sized concert hall as shown in Table 6.1. Also shown in Table 6.1 are the calculated filter attenuation constants using equation 6.15, the calculated delay (in samples) based on a
sampling frequency of 44,100 Hz, and the adjusted delay (in samples) to ensure that all of the comb filter loop times are indeed mutually prime.

<table>
<thead>
<tr>
<th>Filter</th>
<th>Loop Time (Delay)</th>
<th>Total Desired Reverb Time</th>
<th>Attenuation Constant</th>
<th>Delay (Samples)</th>
<th>Delay (Factored)</th>
<th>Delay (Samples) for Mutually Prime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comb 1</td>
<td>29.7 ms</td>
<td>2 sec</td>
<td>0.9025</td>
<td>1310</td>
<td>2<em>5</em>131</td>
<td>1311</td>
</tr>
<tr>
<td>Comb 2</td>
<td>37.1 ms</td>
<td>2 sec</td>
<td>0.8797</td>
<td>1636</td>
<td>2<em>3</em>409</td>
<td>1636</td>
</tr>
<tr>
<td>Comb 3</td>
<td>41.1 ms</td>
<td>2 sec</td>
<td>0.8677</td>
<td>1813</td>
<td>7<em>7</em>37</td>
<td>1813</td>
</tr>
<tr>
<td>Comb 4</td>
<td>43.7 ms</td>
<td>2 sec</td>
<td>0.8599</td>
<td>1927</td>
<td>41*47</td>
<td>1927</td>
</tr>
<tr>
<td>Allpass1</td>
<td>5.0 ms</td>
<td>96.83 ms</td>
<td>0.7</td>
<td>221</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Allpass2</td>
<td>1.7 ms</td>
<td>32.92 ms</td>
<td>0.7</td>
<td>75</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Since 1310 and 1636 share a common factor of 2, the first delay was increased to 1311 sample periods which factors as 3*19*23 and is therefore mutually prime with the delays of the other three comb filters.

Download the Reverb.mdl shown in Figure 6.28 and the BACH PRELUDE.wav files from the text website. Run the simulation and use the Manual Switch to turn the reverb effect on and off. Notice that output signal from the reverb filter is combined with the original signal, and a gain of 0.1 is applied to the reverb signal to adjust for % of reverb. The interested reader can easily adjust any of the parameters in the simulation to explore the effects on the overall sound.

Figure 6.28: Simulink® Model of Reverb Filter

Figure 6.29 shows the impulse response of each comb filter and the overall impulse response of the parallel bank of comb filters. As expected, each comb filter produces a distinct set of decaying echoes at integer multiples of the time delay for that filter. The 1st comb filter produces echoes at 1311, 2*1311, 3*1311, … The impulse response of the parallel bank of comb filters is
simply the four individual impulse responses added together. Because the delays are mutually prime, none of the filters produces an echo at the same time as another filter within the time-span of the total reverb time, so undesirable spikes are not a problem.

Figure 6.29: Impulse Response of the Comb Filters

Figure 6.30 shows the impulse response for each of the all-pass filters, and for the cascaded combination of the two all-pass filters. Since these filters are cascaded, the overall impulse response is the convolution of the two individual impulse responses.
Finally, Figure 6.31 shows the impulse response of the entire reverb filter. The upper plot shows the impulse response for the first 10000 samples (about ¼ of a second), while the lower plot extends the impulse response to a time-span of 1 second.
A digital graphic equalizer consists of a parallel bank of FIR or IIR low-pass, band-pass and high-pass filters in the audio band. One of the most difficult aspects of the design is how to handle the overlap effect between adjacent filters where the two frequency responses are summed together. Figure 6.32 illustrates the unwanted effects (extra boost and/or ripple) that can occur between adjacent filters, as well as the desirable smooth response that can be achieved using digital filters with additional digital processing algorithms.
Example 6.7: Digital Graphic Equalizer
Download the GraphicEq model shown in Figure 6.33 and the BACH PRELUDE wav file from the text website. GraphicEq.mdl consists of one low-pass filter, four bandpass filters, and a high-pass filter connected in parallel. The filters were designed using the Filter Design and Analysis tool. Specific filter parameters are shown in Table 6.2. Play the simulation and experiment with the slider gains to boost or attenuate various frequencies.

Table 6.2: Filter Properties for Graphic Equalizer

<table>
<thead>
<tr>
<th>Filter Type</th>
<th>Design Method</th>
<th>Filter Order</th>
<th>Sampling Frequency (Hz)</th>
<th>Cutoff Frequencies (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowpass</td>
<td>FIR Hamming</td>
<td>256</td>
<td>44100</td>
<td>750</td>
</tr>
<tr>
<td>Bandpass</td>
<td>FIR Hamming</td>
<td>256</td>
<td>44100</td>
<td>750; 1500</td>
</tr>
<tr>
<td>Bandpass</td>
<td>FIR Hamming</td>
<td>256</td>
<td>44100</td>
<td>1500; 3000</td>
</tr>
<tr>
<td>Bandpass</td>
<td>FIR Hamming</td>
<td>256</td>
<td>44100</td>
<td>3000; 6000</td>
</tr>
<tr>
<td>Bandpass</td>
<td>FIR Hamming</td>
<td>256</td>
<td>44100</td>
<td>3000; 11500</td>
</tr>
<tr>
<td>Highpass</td>
<td>FIR Hamming</td>
<td>256</td>
<td>44100</td>
<td>11500</td>
</tr>
</tbody>
</table>
Figure 6.33: Graphic Equalizer Simulink Model

Figure 6.34 illustrates the frequency response for each individual filter as well as the overall frequency response with all six filters connected in parallel and all slider gains set to 1. The overall frequency response is not perfect; ideally, the response would be a flat line at 0 dB over the entire frequency range. This particular design does exhibit some ripple which would require some alteration to the design to flatten out.
Figure 6.34: Frequency Response of Graphic Equalizer
Answers to Challenge Questions

**Question 6.1** How many multiplications per sampling instant must be performed with the FIR and IIR filters designed in Example 6.3? How many multiplications per second (MPS) must be performed for each of these filters?

Assuming each of the filters is implemented in the time domain, we have:

**FIR Filter** 
\[ y(k) = \sum_{i=0}^{87} b_i x(k-i) \]

**IIR Filter** 
\[ y(k) = \sum_{i=0}^{10} b_i x(k-i) - \sum_{i=1}^{10} a_i y(k-i) \]

The FIR filter would require 88 (N+1) multiplications per sampling instant or 4,224,000 (88*48K) multiplications per second (MPS). The IIR filter would require 21 (2*N+1) multiplications per sampling instant or 1,008,000 multiplications per second.

Computational complexity can be reduced by taking advantage of the symmetry of the filters. Recall from Chapter 5 that for an FIR filter, \( b_0 = b_N, \ b_1 = b_{N-1}, \ldots \) so the number of multiplications could be cut in half to 44 multiplications per sampling instant or 2,112,000 MPS. The IIR filter also has a symmetric numerator (with a sign change):

\[
\begin{align*}
b_0 &= 0.03203191821694 \\
b_1 &= -0.05878107521081 \\
b_2 &= 0.10186869443828 \\
b_3 &= -0.09766972701608 \\
b_4 &= 0.07231401919473 \\
b_5 &= 0.00000000000000 \\
b_6 &= -0.07231401919473 \\
b_7 &= 0.09766972701608 \\
b_8 &= -0.10186869443828 \\
b_9 &= 0.05878107521081 \\
b_{10} &= -0.03203191821694
\end{align*}
\]

So the number of multiplies can be reduced to 15 per sampling instant or 720,000 MPS.

**Question 6.2** In example 6.4, the poles of the IIR filter shift when the coefficients are quantized to B bits and it is even possible to make the filter unstable. Will an FIR filter exhibit the same instability problem if the number of bits for the filter coefficients is continually reduced? Explain.

No, the FIR filter will not exhibit stability problems when the filter coefficients are quantized. The FIR filter has all of its poles at the origin (denominator = \( z^N \)) and these poles will not shift.
Another way to look at this is that the FIR filter is non-recursive (no feedback) and will not exhibit stability problems. However, as shown in Chapter 5, quantizing the FIR coefficients can result in poor filter performance (failure to meet design specifications).

**Question 6.3** What is the impulse response of the echo filter?

The impulse response of the echo filter can be found by setting the input \( x(k) = \delta(k) \) and solving the difference equation recursively until a pattern emerges. For simplicity, assume a value for \( N \), say 5.

\[
y(k) = x(k) + \alpha_N y(k - 5) \\
y(0) = \delta(0) + \alpha_N y(-5) = 1 \\
y(k) = \alpha_N y(k - 5) = 0 \quad \text{for } k = 1\ldots 4 \\
y(5) = \alpha_N y(0) = \alpha_N \\
y(k) = \alpha_N y(k - 5) = 0 \quad \text{for } k = 6\ldots 9 \\
y(10) = \alpha_N y(5) = \alpha_N^2 \\
y(k) = \alpha_N y(k - 5) = 0 \quad \text{for } k = 11\ldots 14 \\
y(15) = \alpha_N y(10) = \alpha_N^3
\]

The graph of the impulse response for an echo filter with a time delay of \( N \) is in the following graph.

![Impulse Response for Echo Filter](image)

Each of the impulses represents another reflection of the original sound wave off the two parallel walls.

**Question 6.4** What will happen if the feedback factor, \( \alpha_N \), is chosen to be greater than one for the echo filter? Why?

If the feedback gain, \( \alpha_N \), is chosen to be greater than one, the output signal will grow continually louder until the signal finally saturates the audio system. This can be seen from the difference equation (amplification of the feedback signal) or from the transfer function. The echo filter has \( N \) poles equally spaced around a circle of radius \( \alpha_N \) (i.e., \( z^{-N} = \alpha_N \)). If \( \alpha_N \) is greater than one, then the filter poles all lie outside the unit circle making the filter unstable.
Question 6.5
It is possible to create a reverb filter using a digital FIR filter instead of the IIR echo and all-pass filters. Explain how and discuss the obvious disadvantage of an FIR filter.

Figure 6.24 shows what the impulse response of a room might look like. If we measured the impulse response in some actual desired space, this impulse response would then be the coefficients of the corresponding digital FIR filter. The problem is that the filter order would be huge since there are thousands of reflections per second.
Chapter 6 Problems

Problem 6.1: The transfer function for an IIR filter is:

\[ 0.07083 z^5 + 0.1611 z^4 + 0.2523 z^3 + 0.2523 z^2 + 0.1611 z + 0.07083 \]
\[ \frac{\text{----------------------}}{z^5 - 1.086 z^4 + 1.6 z^3 - 0.9263 z^2 + 0.4804 z - 0.09993} \]

(a) Find the filter order and the filter poles. Is the filter stable?
(b) Write the difference equation for the filter.
(c) Using the MATLAB command `dimpulse`, plot the first twenty terms of the impulse response.
(d) Using the MATLAB command `freqz`, plot the frequency response for this filter (assume a sampling frequency of 16 kHz). What type of filter is this?

Problem 6.2:  
(a) Using the Filter Design and Analysis tool, design a Butterworth digital IIR LPF to meet the following specifications:
   i. Passband: 0 - 22.5kHz (with maximum 1 dB ripple)
   ii. Stopband: f ≥ 28 kHz with As ≥ 60 dB
   iii. Sampling Freq: 96 kHz
(b) Repeat part (a) using a Chebyshev II design.
(c) Repeat part (a) using an Elliptic design.
(d) Compare the frequency responses of the three filters. Also, compute the number of multiplications per second required for each filter.

Problem 6.3: (IIR Filter Design)
(a) Design a Chebyshev II digital IIR BPF to meet the following specifications:
   i. Passband: 5 - 10kHz (with maximum 0.1 dB ripple)
   ii. Stopband: f < 3 kHz and f ≥ 12 kHz with As ≥ 40 dB
   iii. Sampling Frequency: 30 kHz

Include the transfer function of your filter as well as magnitude and phase plots.
(b) Suppose the filter input is \( x = \sin(2\pi \times 2000 \times t) + \sin(2\pi \times 8500 \times t) \). Plot the filter input and the filter output. Also, compute the time delay through the filter for the two input frequencies.
(c) Repeat parts (a) and (b) for an Optimal Equiripple FIR filter.
(d) Compare your two filter designs in terms of phase distortion, number of multiplies required to implement the difference equation, etc.
Problem 6.4: (Effect of Wordlength on IIR Filter Stability and Performance)
(a) Design an IIR bandstop filter (with 2nd order sections) to meet the following specifications:

i. Notch Frequency: 2 kHz
ii. Attenuation at Notch Frequency \(\geq 60\) dB
iii. Passband edge frequencies: 1.8 kHz and 2.2 kHz
iv. Passband Ripple \(\leq 0.1\) dB
v. Sampling frequency \(fs = 8\) kHz

Provide the magnitude frequency response plot from the Filter Design and Analysis Tool. DO NOT turn in the filter coefficients.
(b) Determine the minimum number of bits required to maintain stability of the filter. Turn in the MATLAB code to document your answer.
(c) Quantize the filter coefficients to the number of bits determined in (b) and plot the frequency response of the quantized filter vs. the frequency response of the original filter. How do they compare? (Turn in your MATLAB code for this part)
(d) Determine the number of bits required for good filter performance. Again, plot the frequency response of the quantized filter vs. the frequency response of the original filter to verify good filter performance. (Turn in your MATLAB code for this part)

Problem 6.5:
(a) Write the difference equation for a 4th order all-pass filter assuming \(\alpha = 0.2\).
(b) Calculate and plot the first 10 terms of the impulse response.
(c) Find the poles of the all-pass filter.
(d) Plot the frequency response for the filter assuming a sampling frequency of 48 kHz.
(e) Repeat parts (a)-(d) assuming \(\alpha = 0.8\). How does \(\alpha\) affect the impulse response and the frequency response?

Problem 6.6:
(a) Write the difference equation for a 6th order echo filter assuming \(\alpha = 0.2\).
(b) Calculate and plot the first 20 terms of the impulse response for the echo filter.
(c) Find the poles of the echo filter.
(d) Plot the frequency response for the filter assuming a sampling frequency of 48 kHz. Identify the frequencies that are amplified by the echo filter.
(e) Repeat parts (a)-(d) assuming \(\alpha = 0.8\). How does \(\alpha\) affect the impulse response and the frequency response?
(f) Plot the frequency response for a 10th order echo filter assuming \(\alpha = 0.8\) and the sampling frequency is 48 kHz. How does the increase in order affect the frequency response?

Problem 6.7:
(a) Referring to Figure 6.14, derive Equation 6.7.
(b) Also using Figure 6.14, derive the transfer function from the intermediate signal, \(w(k)\), to the output signal, \(y(k)\).
CHAPTER 6 LAB EXERCISE
IIR Filters

Objectives

1. Design two IIR filters using the Filter Design and Analysis Tool in MATLAB.
2. Investigate the effect of coefficient wordlength on the filter performance.
3. Explore the effect of phasing on music.

Procedure

A. IIR FILTER DESIGN

1. Choose two different filter types (low-pass, high-pass, band-pass, or notch). Enter your choices in the IIR Design Table.
2. Choose a sampling rate for each of the filters. Enter your choices in the IIR Design Table.
3. Specify passband and stopband for your filters. Remember, you are constrained to frequencies which are less than half the sampling frequency. Enter your choices in the IIR Design Table.

<p>| Table 1: IIR DESIGN SPECIFICATIONS |</p>
<table>
<thead>
<tr>
<th>Filter Type</th>
<th>Fs (kHz)</th>
<th>Passband</th>
<th>Stopband</th>
<th>Method</th>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4. Start MATLAB.
5. Launch the Filter Design and Analysis Tool (>> fdatool).
6. Design an IIR filter using your first set of specifications in the IIR Design table. Enter your filter method (Butterworth, Chebychev I, Chebychev II, or Elliptic) and the resulting filter order in the IIR Design Table.
7. Print the resulting magnitude response and phase response. Notice that an IIR filter does not have linear phase response!
8. Convert the IIR filter to a single section by selecting Edit→Convert to Single Section.
9. Export the filter coefficients by selecting File → Export. Choose some unique variable names for your filter numerator and denominator.
10. Repeat steps 6 - 9 for your second set of filter specifications in the IIR Design Table. Be sure to save the filter numerator and denominator under different variable names than those used for the first filter!
B. WORDLENGTH EFFECTS

The following m-file of MATLAB commands will quantize the filter denominator coefficients (Den) to a specified number of bits then calculate the magnitude of the poles of the quantized denominator. The m-file also quantizes the filter numerator (Num) and plots the frequency response of the quantized filter versus the original filter. You will need to modify this code for your filter; that is, enter your sampling frequency, Fs, and your variable names for Num and Den.

```
1  B=9;
2  Fs=48000;
3  Denq=round(Den*2^(B-1))/(2^(B-1));
4  abs(roots(Denq))
5  Numq=round(Num*2^(B-1))/(2^(B-1));
6  [Hq,fq]=freqz(Numq,Denq,500,Fs);
7  [H,f]=freqz(Num,Den,500,Fs);
8  plot(f,20*log10(abs(H)),fq,20*log10(abs(Hq)));
9  legend('Unquantized Response','Quantized Response');
10  xlabel('Frequency (Hz)'); ylabel('Filter Magnitude (dB)');
11  title(['IIR Filter Magnitude Response (' num2str(B) ' bits)']); grid
```

1. Vary the number of bits, B, in the m-file and find the minimum number of bits needed for stability for your first IIR filters. Record your answer in Table 2. Recall, a discrete system is stable if and only if the magnitude of the poles is strictly less than 1.

<table>
<thead>
<tr>
<th>IIR FILTER</th>
<th>Minimum Bits for Stability (Direct Realization)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. Using the minimum number of bits required for stability, plot the frequency response of the quantized system and the frequency response of the original IIR filter on the same plot. How does the quantized response compare to the original frequency response?
3. Determine the number of bits required to achieve a good match between the quantized filter and the original IIR filter. Fill in the answer in the Table 3.

<table>
<thead>
<tr>
<th>IIR FILTER</th>
<th>Minimum Bits for Good Performance (Direct Realization)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4. Repeat steps 1-3 for the 2\textsuperscript{nd} IIR filter making sure to modify the m-file appropriately.

C. CASCADE REALIZATION AND WORDLENGTH EFFECTS

1. Open the Filter Design and Analysis Tool in MATLAB.
2. Enter the design parameters for your first IIR filter designed in Part A.
3. Click on Edit→Convert Structure. Choose 2\textsuperscript{nd} Order Sections.
4. Export the SOS matrix and the G matrix (it is not necessary to change these names).
5. Return to the MATLAB workspace and execute the statement: \texttt{format long; SOS}

Row 1 is the transfer function for the 1\textsuperscript{st} biquad. The first three numbers (columns 1-3) are the numerator and the last three coefficients (columns 4-6) are the denominator. Row 2 contains the 2\textsuperscript{nd} biquad transfer function and the pattern continues.

The m-file, IIR-quant.m, (code provided in Figure 6.11) will quantize the denominator of each 2\textsuperscript{nd} order section and find the magnitude of the poles for each quantized section. It will also quantize the numerator of each section and plot the frequency response of the quantized cascaded IIR filter versus the un-quantized filter. \textit{You will need to modify this m-file by entering your sampling frequency, Fs. As long as you exported your cascaded filter under the default variables of SOS (Second Order Sections) and G (Gain), the rest of the code will execute fine.}

6. Vary the number of bits, B, in the m-file to determine the minimum number of bits required for stability in the cascade realization for your first filter. Enter your answer in Table 4.

<table>
<thead>
<tr>
<th>IIR Filter</th>
<th>Minimum Bits for Stability In Cascade Realization</th>
<th>Minimum Bits for Stability In Direct (Full) Realization</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

7. Determine the number of bits required for good filter performance (i.e. a good match to the original filter) for each of your IIR filters. Fill in the answer in Table 5.

<table>
<thead>
<tr>
<th>IIR Filter</th>
<th>Minimum Bits for Good Performance In Cascade Realization</th>
<th>Minimum Bits for Good Performance In Direct (Full) Realization</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

8. Repeat Steps 1-7 for your 2nd IIR filter using a cascade implementation.

How does the cascade implementation differ from the single stage implementation in terms of the number of bits needed to maintain stability and the number of bits required to meet the filter design specifications?

D. Phasing

A block diagram for a phaser is shown in Figure 1. As discussed in the text, the all-pass filter chain creates a set of notch frequencies in the phaser and the spectrum of incoming audio signal will be attenuated or even eliminated at these frequencies. The notch frequencies are swept or varied by changing the gains, α, in the all-pass filters. This portion of the lab will begin with an exploration of the all-pass filter chain to determine how the various parameters affect the frequency response of the phaser.

![Phaser Diagram](image)

**Figure 1: Phaser**

1. Copy the m-file `allpass.m` from the text website into the current MATLAB directory.
2. In the MATLAB workspace, type the command

\[
\text{allpass( [0.2 0.2 0.2 0.2] , 1, 44100)}
\]

You should see a plot exactly like Figure 6.23(a). The first argument in \texttt{allpass.m} specifies the gain for each 1st order all-pass filter. The gains do not need to be the same but they must be between 0 and 1. The size of the vector determines how many all-pass filters are in the chain. There would be four all-pass filters for the command just executed. The second argument in \texttt{allpass.m} is the depth, \(g\), and may range from 0 up to 1. The 3rd argument is the sampling frequency of the incoming audio signal.

3. Experiment with \texttt{allpass.m} in order to answer the following questions:

What is the relationship between the number of notch frequencies and the number of all-pass filters included in the chain?

How do the filter gains, \(\alpha\), affect the location of the notch frequencies?

What effect does the variable depth have on the frequency response of the phaser?

4. Copy the files \texttt{phaser\_env.m}, \texttt{phase.mdl} and \texttt{G\_Maj.wav} from the text website into the current MATLAB directory.

Note: The file \texttt{G\_maj.wav} was originally downloaded from the freesound website at \url{http://www.freesound.org}.

5. Open the Simulink file, \texttt{phaser.mdl}. The model should look like Figure 2.

The model consists of a chain of four 1st order all-pass filters. The first two filters use gains from \texttt{alpha1.signals.values} while the second two filters use gains from \texttt{alpha2.signals.values}. Variations in the gain values over time will cause the notch frequencies to sweep. These gains will be generated using the m-file \texttt{phaser\_env.m}. The model includes a slider gain to control depth. Setting depth to 0 will eliminate the output from the all-pass filter chain and simply play the original audio signal. Setting depth equal to 1 will cause 100% cancellation of the audio input spectrum at the notch frequencies.
6. In the MATLAB workspace, run the command

   \[
   \{\alpha_1, \alpha_2\} = \text{phaser\_env}([0.1 \ 0.9 \ 0.3 \ 0.9], 1, 3);
   \]

   The \texttt{phaser\_env.m} file will compute and plot the alpha1 and alpha2 signal values that are needed to run the \texttt{phaser.mdl}. The first argument (a vector of length 4) specifies min and max values for alpha1 and min and max values for alpha2 respectively. The second argument, T, specifies the period of time to complete one cycle over which the gains alpha1 and alpha 2 will sweep from max to min and back to max. Equivalently, T controls the sweep time of the notch frequencies. The third argument, K, controls the steepness of the gain transition from min to max.

7. Run the \texttt{phaser.mdl} and listen to the audio through headphones. The slider gain is initially set at 0 (original audio out). Vary this gain to hear the phasing effect.

8. Experiment with the input variables for phaser\_env, particularly cycle time, T, and steepness of transition, K. Explain the effect of the variables on the sound that you hear.

Comments:
- The \texttt{phaser.mdl} could be easily modified to an eight chain all-pass filter.
- The \texttt{phaser\_env.m} file could be modified for step changes in alpha which would cause step changes in the notch frequencies rather than continuous sweeping.