IIR Filtering: Filter-Design Exercise in MATLAB*

Douglas L. Jones
Swaroop Appadwedula
Matthew Berry
Mark Haun
Jake Janovetz
Michael Kramer
Dima Moussa
Daniel Sachs
Brian Wade

This work is produced by OpenStax-CNX and licensed under the
Creative Commons Attribution License 1.0†

Abstract

You will derive the transfer function of a second-order, Direct Form II, infinite impulse response (IIR) filter. Then you will create a fourth-order IIR filter, plot its frequency response, and decompose the fourth-order filter into two second-order sections, choosing an appropriate gain for each stage to prevent overflow.

1

The transfer function for the second-order section shown in IIR Filtering: Introduction[1] is

$$H(z) = G\frac{1 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{1}$$

1.1 Exercise

First, derive the above transfer function. Begin by writing the **difference equations** for $w[n]$ in terms of the input and past values ($w[n-1]$ and $w[n-2]$). Then write the difference equation for $y[n]$ also in

*Version 2.11: Feb 25, 2004 12:37 pm -0600
†http://creativecommons.org/licenses/by/1.0
[1]"IIR Filtering: Introduction", Figure 1 <http://cnx.org/content/m10025/latest/#fig1>

terms of the past samples of $w[n]$. After finding the two difference equations, compute the corresponding Z-transforms and use the relation $H(z) = \frac{Y(z)}{X(z)} = \frac{Y(z)W(z)}{W(z)X(z)}$ to verify the IIR transfer function in (1).

Next, design the coefficients for a fourth-order filter implemented as the cascade of two bi-quad sections. Write a MATLAB script to compute the coefficients. Begin by designing the fourth-order filter and checking the response using the MATLAB commands

```
[B,A] = ellip(4,.25,10,.25)
freqz(B,A)
```

NOTE: MATLAB's `freqz` command displays the frequency responses of IIR filters and FIR filters. For more information about this, type `help freqz`. Be sure to look at MATLAB's definition of the transfer function.

NOTE: If you use the `freqz` command as shown above, without passing its returned data to another function, both the magnitude (in decibels) and the phase of the response will be shown.

Next you must find the roots of the numerator, **zeros**, and roots of the denominator, **poles**, so that you can group them to create two second-order sections. The MATLAB commands `roots` and `poly` will be useful for this task. Save the scripts you use to decompose your filter into second-order sections; they will probably be useful later.

Once you have obtained the coefficients for each of your two second-order sections, you are ready to choose a **gain** factor, G, for each section. As part of your MATLAB script, use `freqz` to compute the response $\frac{W(z)}{X(z)}$ with $G = 1$ for each of the sets of second-order coefficients. Recall that on the DSP we cannot represent numbers greater than or equal to 1.0. If the maximum value of $|\frac{W(z)}{X(z)}|$ is or exceeds 1.0, an input with magnitude less than one could produce $w[n]$ terms with magnitude greater than or equal to one; this is **overflow**. You must therefore select a gain values for each second-order section such that the response from the input to the states, $\frac{W(z)}{X(z)}$, is always less than one in magnitude. In other words, set the value of G to ensure that $|\frac{W(z)}{X(z)}| < 1$.

1.2 Preparing for processor implementation

As the processor exercises become more complex, it will become increasingly important to observe good programming practices. Of these, perhaps the most important is careful planning of your program flow, memory and register use, and testing procedure. Write out pseudo-code for the processor implementation of a bi-quad. Make sure you consider the way you will store coefficients and states in memory. Then, to prepare for testing, compute the values of $w[n]$ and $y[n]$ for both second-order sections at $n = \{0, 1, 2\}$ using the filter coefficients you calculated in MATLAB. Assume $x[n] = \delta[n]$ and all states are initialized to zero. You may also want to create a frequency sweep test-vector like the one in DSP Development Environment: Introductory Exercise for TI TMS320C54x[2] and use the filter command to find the outputs for that input. Later, you can recreate these input signals on the DSP and compare the output values it calculates with those you find now. If your program is working, the values will be almost identical, differing only slightly because of quantization effects, which are considered in IIR Filtering: Filter-Coefficient Quantization Exercise in MATLAB[3].

[2] "DSP Development Environment: Introductory Exercise for TI TMS320C54x" <http://cnx.org/content/m10017/latest/>
[3] "IIR Filtering: Filter-Coefficient Quantization Exercise in MATLAB" <http://cnx.org/content/m10813/latest/>