

Abstract

For a given filter specification, infinite impulse response (IIR) filters offer faster computation and less memory requirement than finite impulse response (FIR) filters. These advantages come with the complexity of nonlinear phase, coefficient round-off sensitivity, and possible instability. In this prelab exercise, students will use MATLAB to explore finite precision effects in IIR filters implemented as a cascade of second-order sections. Additionally, students will write pseudo-code for an assembly language implementation.

IIR Filter Structure

An IIR filter is a linear time-invariant system and, with sufficient order, is able to approximate any frequency response. The difference equation for a N th order IIR filter is

$$y[n] = \sum_{k=0}^N b_k x[n-k] - \sum_{k=1}^N a_k y[n-k] \quad (1)$$

The transfer function is given by

$$H(z) = \frac{b_0 + b_1 z^{-1} + \dots + b_N z^{-N}}{1 + a_1 z^{-1} + \dots + a_N z^{-N}} \quad (2)$$

Infinite impulse response filters are implemented as a cascade of second order (*i.e.*, bi-quadratic, or **bi-quad**) sections. This factorization provides robustness to both finite precision arithmetic and overflow, and allows simple implementation. In factorizing a filter transfer function, $H(z)$, conjugate pairs of poles and conjugate pairs of zeros are grouped together so that all coefficients remain real-valued. A second order section, or bi-quad, is shown in Figure 1 for the Direct-Form II implementation. Note that the state variables, $w[n]$, are delayed samples computed from both inputs and outputs.

The transfer function for the second order section shown in Figure 1 is

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \quad (3)$$

Exercise (a)

Derive the transfer function in Equation 3. Referring to Figure 1, begin by writing the difference equation for $w[n]$ in terms of the input, $x[n]$, and past values of $w[n]$. Next, write the difference equation for $y[n]$ in terms of $w[n]$ and past values of $w[n]$.

Finally, compute the corresponding Z-transforms and use the following relation to arrive at the transfer function

$$H(z) = \frac{Y(z)}{X(z)} = \frac{W(z)}{X(z)} \frac{W(z)}{Y(z)}$$

Exercise (b)

Use MATLAB to design a sixth-order IIR band-stop filter. The edges of the pass bands should be 9600 Hz and 12000 Hz, with stop-band attenuation of 50 dB and no more than 1 dB pass-band ripple. The sampling rate is 48 kHz. Inspect the frequency response of your filter. For example,

```

Fs = 48000;      % Sampling Frequency
N = 6;          % Filter Order
Fpass1 = 9600;  % First Pass-band Edge Frequency
Fpass2 = 12000; % Second Pass-band Edge Frequency
Ap = 1;         % Pass-band Ripple (dB)
As = 50;       % Stop-band Attenuation (dB)

% Calculate the transfer function using the ELLIP command.
[B,A] = ellip(N/2, Ap, As, [Fpass1 Fpass2]/(Fs/2), 'stop');
freqz(B,A);%display magnitude (dB) and phase

```

Finite Precision Numerical Issues

For implementation on a fixed-point processor, filter coefficients must be quantized. Although FIR filters are robust to coefficient **quantization**, the effects may be significant for IIR filters due to perturbation of pole locations. MATLAB uses

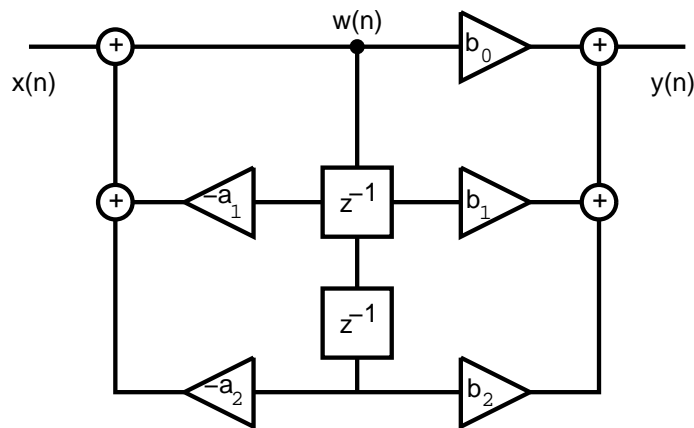


Figure 1: Direct-form II structure.

64-bit floating point numbers in all of its computation; these floating point numbers can represent approximately 19 digits of precision. In contrast, a 16-bit fixed-point DSP can express numbers in the range -1 to $1 - 2^{-15}$. This gives the DSP only about 5 digits of precision and only if the input is properly scaled to occupy the full range from -1 to 1 .

A second effect of finite precision arithmetic is **overflow**. The two's complement arithmetic can only represent numbers in the range from -1 to $1 - 2^{-15}$. If the magnitude response from input, $x[n]$, to state variables, $w[n]$, exceeds 1, overflow can occur.

Cascading Second Order Sections

A second-order factorization of the IIR transfer function provides robustness to both quantization and overflow. The factorization also allows simple implementation. In factorizing a filter transfer function, $H(z)$, conjugate pairs of poles and conjugate pairs of zeros are grouped together so that all coefficients remain real-valued. In a filter with $N/2$ number of bi-quad sections, note that there exist $((N/2)!)^2$ ways to pair poles with zeros and then order the bi-quad sections. The following is a simple rule of thumb for pairing poles with zeros and ordering the sections:

1. Find pole closest to unit circle; pair with zero closest to the pole. Repeat.
2. Order sections by closeness to unit circle: start closest to the origin.

Scaling is used to avoid or reduce the possibility of overflow at the filter state variables, $w_i[n]$. However, scaling reduces dynamic range and signal-to-noise ratio. Rather than simply scaling the input, improved performance can be achieved by distributing the gain among the bi-quad sections. Let $G_i(z)$ denote the transfer function from $x(n)$ to $w_i(n)$, as illustrated in Figure 2.

$$G_i(e^{j\omega}) = \frac{W_i(e^{j\omega})}{X(e^{j\omega})}$$

The overall transfer function for $nsec$ bi-quad sections is given by

$$H(z) = \prod_{i=1}^{nsec} \frac{b_{i0} + b_{i1}z^{-1} + b_{i2}z^{-2}}{1 + a_{i1}z^{-1} + a_{i2}z^{-2}} = \prod_{i=1}^{nsec} H_i(z)$$

Then, from Figure 2 the transfer functions $G_i(z)$ are

$$G_i(z) = \frac{\prod_{k=1}^{i-1} H_k(z)}{1 + a_{i1}z^{-1} + a_{i2}z^{-2}} \quad (4)$$

To insure that every frequency component avoids overflow at the i th state node, we require scale factors

$$s_i = \max_{\omega} |G_i(e^{j\omega})| \quad (5)$$

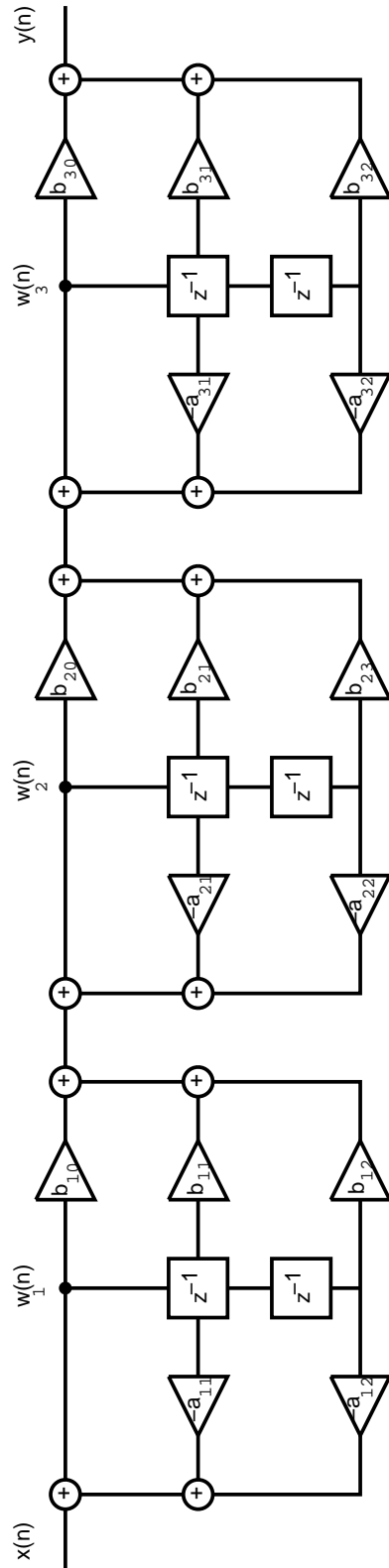


Figure 2: Cascade of three bi-quad sections

Using the factors $\{s_i\}_{i=1}^{nsec}$, the filter transfer function $H(z)$ can be equivalently written

$$H(z) = \frac{1}{\prod_{k=1}^{nsec} s_k} \prod_{i=1}^{nsec} \frac{s_i (b_{i0} + b_{i1}z^{-1} + b_{i2}z^{-2})}{1 + a_{i1}z^{-1} + a_{i2}z^{-2}} \quad (6)$$

In a cascade realization, the scaling factor s_i is a multiplicative term on the numerator coefficients for stage i . Additionally, the $1/s_k$ term in Eqn. 6 can be absorbed into the numerator of the previous stage; this is depicted in Table 1 for a three section IIR filter. For a sixth order filter ($nsec = 3$), implementation of this scaling scheme is summarized in Figure 3.

stage	numerator coefficient vector	scaled numerator coefficient vector
pre	–	$(1/s_1)$
1	B_1	$(s_1/s_2)B_1$
2	B_2	$(s_2/s_3)B_2$
3	B_3	$(s_3/2^k)B_3$
post	–	2^k

Table 1: In combating overflow in fixed point implementation, scale factors can be absorbed in the numerator of the previous IIR stage.

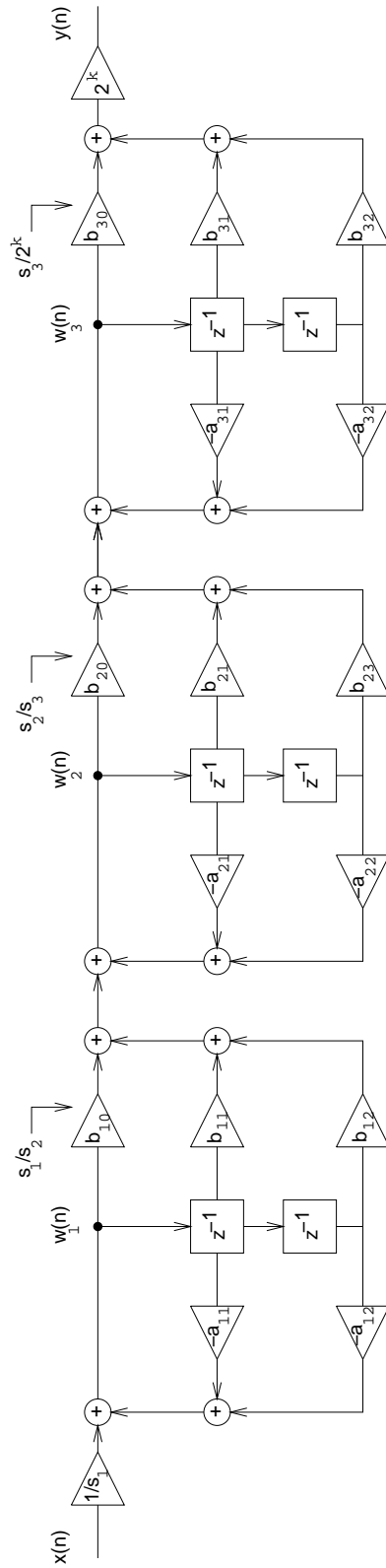


Figure 3: Cascade of three bi-quad sections with scaling.

The computation of second order sections and their ordering are conveniently computed in MATLAB by `tf2sos`. Additionally, the code below computes the pre-scaling factor, the scaled numerators, and the number of arithmetic left shifts, as depicted in Figure 3.

```

% convert transfer function to second-order sections.
[sos,g] = tf2sos(B,A,'UP','none');
sos(1,1:3) = g*sos(1,1:3);%incorp gain into first monic numerator
nsec=size(sos,1); %number of sections
Nf=1024;%number of frequency samples for inf norm (i.e., max abs)
H(1,:)=freqz(sos(1,1:3),sos(1,4:6),Nf);%first stage
G(1,:) = freqz(1,sos(1,4:6),Nf);%gain to first state variable
s(1) = max(abs(G(1,:)));
pre = 1/s(1);%prescale factor
for k=2:nsec
    H(k,:) = freqz(sos(k,1:3),sos(k,4:6),Nf);
    G(k,:) = freqz(1,sos(k,4:6),Nf).'* H(k-1,:);
    s(k) = max(abs(G(k,:)));
    scale(k-1) = s(k-1)/s(k);%scale factor for kth section
end
tmp = ceil(max(abs(s(nsec)*sos(nsec,1:3))));%at last numerator
if(tmp > 1), asl = nextpow2(tmp); else asl=0;end;% pull out left shifts
scale(nsec) = s(nsec)/( 2^asl );
%modify coefficient list
coeffs = sos(:,[1 2 3 5 6]);%a0=1, so omit
coeffs(:,4) = coeffs(:,4)/2;%a1 coefficients divide by 2
for k=1:nsec
    coeffs(k,1:3) = coeffs(k,1:3)*scale(k);%scale numerators
end
coeffs
pre
asl

```

A refined, and more complicated, pairing and scaling approach is discussed by Dehner (2003), doi:10.1016/S0165-1684(03)00075-6. In addition, avoidance of overflow in cascaded IIR implementations is discussed in Texas Instruments application note `spra509.pdf`.

Exercise (c)

Explore the effects of quantization by quantizing the filter coefficients for the notch filter. Use the `freqz` command to compare the response of the 64-bit Matlab filter with two quantized versions: first, quantize the entire fourth-order filter at once, and second, quantize the second-order (bi-quad) sections separately and recombine the resulting quantized sections using the `conv` function. Compare the response of the unquantized filter and the two quantized versions.

Use MATLAB to quantize the filter coefficients to the 16-bit precision used on the DSP. First take each vector of filter coefficients (that is, the A and B vectors) and divide by the smallest power of two (call it 2^d) such that the resulting absolute value of the largest filter coefficient is less than or equal to one. Second, quantize the resulting vectors to 16 bits of precision by multiplying them by $2^{15} = 32768$, rounding to the nearest integer (use `round`), and then dividing the resulting vectors by 32768. Third, multiply the resulting numbers, which will be in the range of -1 to 1 , by 2^d to restore the scaling factor from the first step.

Representing Coefficients

For a stable bi-quad section, the coefficient a_2 is less than one in magnitude, and $|a_1|$ is less than 2. Therefore, to represent the coefficients, a DSP implementation divides a_1 coefficients by 2 to save the value in memory; then, IIR implementation uses $a_1/2$.

The post-scaling gain shown in Figure 3 may be necessary. The implementation of the gain is simplified by using powers of two, implemented as an arithmetic shift left before saving accumulator contents to memory; with the TMS320C55x, see the `SFTS` command.

Exercise (d)

Write a draft assembly language program to implement a cascade of bi-quad sections. Simplify your work by reusing portions of your FIR filtering code tested previously.

Rather than write separate code for each second-order section, you are encouraged first to write one section, then write code that cycles through the second-order section code $nsec$ times using the repeat structure below. Because the IIR code will have to run inside the block I/O loop and this loop uses the block repeat counter (`BRC0`), you must use another looping structure to avoid corrupting the `BRC0`. Note that your code must use different coefficients and states for each bi-quad section within the repeat loop.

```

mov    #num_stages-1, AR1
start_stage
; IIR code goes here
BCC start_stage, *AR1- != #0

```

An implementation on the TMS320C54x is found in the Texas Instruments document [spra669.pdf](#), pages 8-9. The opcode mnemonics are different than for the C55x, but the code structure may provide useful guidance in constructing your solution.