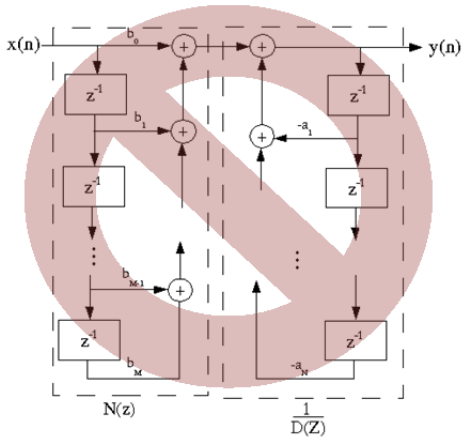
EE 419 - Project 6
Efficient IIR Filter Implementations



REMINDERS:
1) When you are finished, be sure to always copy your m-files and "C" source code files to a place that you can retrieve them again in the future.

Background:

One way that we can significantly reduce the cost of implementing a digital filter is to use integer values for all signal levels and all filter coefficients; so that the filter difference equation can be computed using simpler and less expensive integer multipliers and adders (instead of the more complex and more expensive floating-point adders and multipliers).

Fortunately, the digital signals arriving into our filter generally come from an A/D converter; and so they arrive already in an integer form (using either two's complement, offset binary, or unipolar binary coding forms). The more difficult adaptation is to convert the difference equation coefficients to integer values in a way that the filter's frequency response is not affected by the rounding/quantization errors that occur when we use less precise integer versions of the coefficients. The typical way to convert real-valued (floating point) filter coefficients to integers is to multiply them by a scale factor, and round the result to the nearest integer value:

$$\text{IntegerA}_k = \text{round}(\text{ExactAk} * \text{ScaleFactor})$$
$$\text{IntegerB}_k = \text{round}(\text{ExactBk} * \text{ScaleFactor})$$

Then the difference equation computations can be carried out using all integer mathematics. Note that the filter coefficient values are typically fractional ($|\text{ExactB}_k| < 1.0$), and so scaling is needed to achieve an integer result ($|\text{IntegerB}_k| >> 1$). Notice also that the larger the scale factor, the less rounding error that there is between the exact scaled value and the integer rounded scaled value:

Example: B1 = 0.141; ScaleFactor = 8: ScaledB1 = 8*.141 = 1.128 IntegerB1 = 1 Error = 0.128
Example: B1 = 0.141; ScaleFactor =64: ScaledB1 =64*141 = 9.024 IntegerB1 = 9 Error = 0.024

When we multiply all of the filter coefficients by a scale factor, though, we are adding an extra gain to the filter, and so the filter output signal will also be scaled by the same scale factor. Therefore, the scale factor gain must be removed from the computed filter output values before sending them out of the digital filter:

$$y]n] = y_{scaled}[n] \;/\; \text{ScaleFactor}$$

To make this unscaling of the output as simple as possible, the scale factor values that are used for integer filters are typically powers of 2. This way, the unscaling (integer division by a power of 2) can be carried out by simply bit shifting the scaled output values, rather than needing to carry out an actual division operation. (Multiplication and division of binary numbers by powers of 2 can be accomplished with bit shifts.)

1) Modeling the Effects of Coefficient Quantization on an IIR Double Notch Filter

In this exercise, you will be comparing the response of an IIR filter implemented with high precision, floating-point coefficients to identical filters implemented with equivalent, lower-precision coefficients. You will be modeling situations where the coefficients are scaled and then implemented with integers, with the scale factors chosen as a power of 2 to permit unscaling the results using simple binary shifts. ***Be aware at all times that in order to observe the desired effects and arrive at the most correct results, you must take care to preserve the precision of your computations at all times***. Therefore, avoid rounding pole/zero locations and intermediate computation results as much as possible; and allow Matlab to use the full precision results of previous computations to compute the next answers required (rather than entering displayed results that are generally rounded to 4 decimal places). For the best precision, specify pole/zero locations to Matlab that are not at exact fraction coordinates (like ½ + j ¼) using exponential polar form: Radius * exp(j* Angle). While there are many potential sets of results for this investigation that you might generate (and most will be considered acceptable), there IS a single, FULLY CORRECT set of answers that can be arrived at using correct precisions.

Ideal Filter Design and Analysis:

Design a notch filter using the method of P/Z placement. The system should have two notches, with center frequencies of 1000 Hz and 2000 Hz, with a sampling rate of S = 48 kHz. Use a pole radius of 0.95. Use exact values (high precision, floating point) for your filter coefficients in this design.

a) What are the pole and zero locations? (Using polar coordinates: Radius ∠ Angle)

Zeros: _____

Poles: _____

b) Analyze your filter using Matlab. Adjust your difference equation coefficients as needed to achieve a peak gain (between the analog input and output) of exactly 1.00000. Plot the Poles/Zeros, Frequency response (linear scale, digital frequencies), and 40-point unit sample response. (**Attach plots**).

Report your difference equation filter coefficients (floating point values): K = _____

Ak = _____

Bk = _____

Scaled Integer Filter Implementation Modeling – 4th-order Direct Form II Transposed Filter:

c)& d) Use Matlab to investigate what happens to the frequency response, pole/zero locations, and unit sample response of this notch filter when it is implemented as a **4th-order Direct Form II Transposed IIR filter** with **scaled integer** coefficients. [NOTE: The Matlab *filter( )* function implements a standard difference equation using the Direct Form II Transposed structure; so the correct form is taken care of for you when you use this command for finding the Unit Sample Response.] Since your Matlab analysis tools do not make provision for unscaling the filter output after the scaled difference equation computations, use the *floating-point equivalents* of your scaled integer coefficients for your analyses by scaling, rounding, and then unscaling the filter coefficients before applying them to your Matlab filter analysis programs.

$$\text{UnscaledIntegerA}_k = \text{round}(\text{ExactAk} * \text{ScaleFactor}) / \text{ScaleFactor}$$
$$\text{UnscaledIntegerB}_k = \text{round}(\text{ExactBk} * \text{ScaleFactor}) / \text{ScaleFactor}$$

These unscaled, integer coefficients will no longer be integers; but will still create the quantization error effects that occur from rounding the scaled filter coefficients to integer values. This form is more convenient for your analysis, as you will not see the scaling effects on the filter gain in your frequency response and unit sample response plots.

You should compare the results to those achieved with full coefficient precision in the previous section, for the following cases.:

c) **Using a scale factor of 512** for all filter coefficients before rounding to integers.

d) Using whatever 2^S scale factor for all integer filter coefficients (which you determine by repeated trials) that is large enough to provide sufficient coefficient precision to achieve a frequency response that "reasonably resembles" (achieves close to the shape of) the non-quantized (floating point) coefficient filter, when you analyze the quantized coefficient filter in Matlab. **You should achieve notch center frequencies that are within ±5% of the intended frequencies** when using your chosen scale factor and quantized coefficients.

For your two cases c) & d)**, provide the following:**
(You do not have to include the results for unsatisfactory scale factors, other than SF=2^9=512)

1) **Scale factor** used for the filter coefficients
2) **Filter coefficients values** - both the scaled integer values, and the unscaled effective floating-point Ak and Bk values due to rounding (on same scale as the original, high precision floating-point values).
3) **Number of bits required** to represent the integer versions of your Ak and Bk values.
4) Resulting locations of **poles, zeros** (numeric values]
5) **Pole/Zero Diagram** (Matlab)
6) **Frequency Response plot** (Matlab - linear scale, Digital freq)
7) **Unit sample response plot** (40 points) (Matlab)
8) Determine the achieved **analog notch center frequencies** (in Hz) for each case from your plots (if possible). [Use a very large # of freqz samples to improve the frequency resolution for your notch center determination.] Determine also the **% Error** in the achieved notch center frequencies, compared to the intended notch frequencies.
9) Determine the **% Error** in the effective Ak and Bk values (after scaling, rounding, and unscaling) compared to the ideal floating point values.

Scaled Integer Filter Implementation – Cascaded 2nd-order Filters:

e) Use Matlab to show the results that could be achieved by breaking up your filter into a **cascade** combination of **two 2nd-order IIR filters**. Be sure to apply all the appropriate guidelines for grouping poles/zeros in each of the two filter sections. Also, be sure to maintain unity passband gain in each of the two stages (even if there is a higher peak near the notch), and again **use integer coefficients with a scale factor of 512** in both sections (or the lowest power of 2 that achieves the desired overall frequency response if 512 does not work for your filter.)

Determine the frequency response values for **each** of the individual 2nd-order sections. Then combine the complex values from these two 2nd-order frequency responses together to plot the overall frequency response (magnitude and phase responses) of the total, cascade 4th-order filter.

Provide the following information:

a) Filter coefficients for each of the two filter sections (scaled integer and effective fixed-point Ak and Bk values due to rounding).
b) Resulting locations of poles, zeros (numeric values] **for each section**.
c) Pole/Zero Diagram for each section (Matlab)
d) Frequency Response plot for each section (Matlab - linear scale, Digital freq)
e) Determine the analog notch center frequency for each section (in Hz) from your plots (for the scaled integer implementations). [Use a very large # of freqz samples to improve the frequency resolution for your notch center determination.] Determine also the **% Error** in the achieved notch center frequencies, compared to the intended notch frequencies.
f) Total Frequency Response plot for the combined filter. (Magnitude and phase responses).
g) Compare these results to those from the 4th-0rder implementation with the same scale factor.

Scaled Integer Filter Implementation – Lattice-Ladder Filter Structure

f)) Use Matlab to show the results that would be achieved by implementing your filter in a **Lattice-Ladder IIR filter structure, using both exact (floating point) and scaled integer "C" and "M" coefficients.**

Use the Matlab `tf2latc( )` function to convert your exact, floating-point Ak and Bk filter coefficients into floating-point M and C (lattice and ladder) coefficients for an equivalent filter implementation. Confirm that this floating-point lattice-ladder filter does, in fact, match the desired double notch filter design when using the exact coefficients, by determining and plotting the **frequency response** of the IIR lattice-ladder filter. For this, you

can use the Matlab y=`latcfilt(M,C,x)` command (similar to the y=`filter(B,A,x)` command, except that it implements a lattice-ladder filter) to compute the response of such a filter to a suitable test input signal. If you need to adjust the overall gain of the lattice-ladder filter to achieve unity gain in the passband, you would scale the C coefficient values (not the M coefficients) the same way you would scale the Bk coefficients in a Direct Form implementation.

Then create a **quantized coefficient** (scaled integer coefficient) version of the lattice ladder filter, by scaling, rounding, and unscaling the C and M coefficients, again **using a scale factor of 512**. Determine and plot the frequency response of this less precise filter with quantized coefficients, using the same method you used on the floating-point filter.

Provide the following information:

a) Matlab code used for computing the lattice-ladder filter coefficients, and for plotting the frequency response plots.
b) **C and M Filter coefficients** for the two filters (exact floating-point values, scaled integer values, and unscaled effective fixed-point C and M values.)
c) **Frequency Response plot** for each filter (linear magnitude and phase vs. Digital frequency)
d) Determine the **analog notch center frequency** for each filter (in Hz) from your plots. Determine also the **% Error** in the achieved notch center frequencies, compared to the intended notch frequencies.
e) **Compare the frequency response results** of these two filters (compared the full precision and quantized filters to each other), and compare these results to those from the 4th-0rder Direct Form, and cascaded 2nd-order Direct Form implementations that used the same scale factors. Did the lattice-ladder implementation achieve similar or better/worse results? [Explain the filter performance metrics by which you reached your conclusions.]

2) Limit Cycles from Quantization of Multiplier Outputs

There are two different effects of finite precision mathematics in IIR digital filters that must be prevented if the filter is to operate properly in all instances. The first effect, arithmetic overflow at internal summing nodes of a digital filter, you will observed in a later lab. A related effect, known as *limit cycles*, is also sometimes observed in recursive filters. Limit cycles are very small oscillations or a "dead zone" around an output value of zero that prevent the filter output y[n] from ever fully settling down to zero after a transient stimulation, even if the x[n] input has all zero values from then on. Unlike arithmetic overflow, which results in full-scale errors in the output values and which originate from summation results having too few bits, limit cycles are generally caused by quantization (rounding) of multiplier output results. FIR filters that do not have feedback are not susceptible to this problem.

In this exercise, you will implement a simple recursive filter using Matlab, and will induce a limit cycle in the output signal when observing its unit sample response (which will invoke a transient response, but then provide no further input stimulation). Since the filter is stable, the unit sample response output is supposed to settle down to zero eventually.

a) Compose a Matlab function that will compute and plot the unit sample responses for two filters with the following difference equations:

$$y[n] = x[n] - A_1 y[n-1] \quad \text{(unquantized)}$$

$$y_q[n] = x[n] - Q\{A_1 y[n-1]\} \quad \text{(quantized multiplier output)}$$

where Q{ w } is a quantized, finite bit precision version of w,

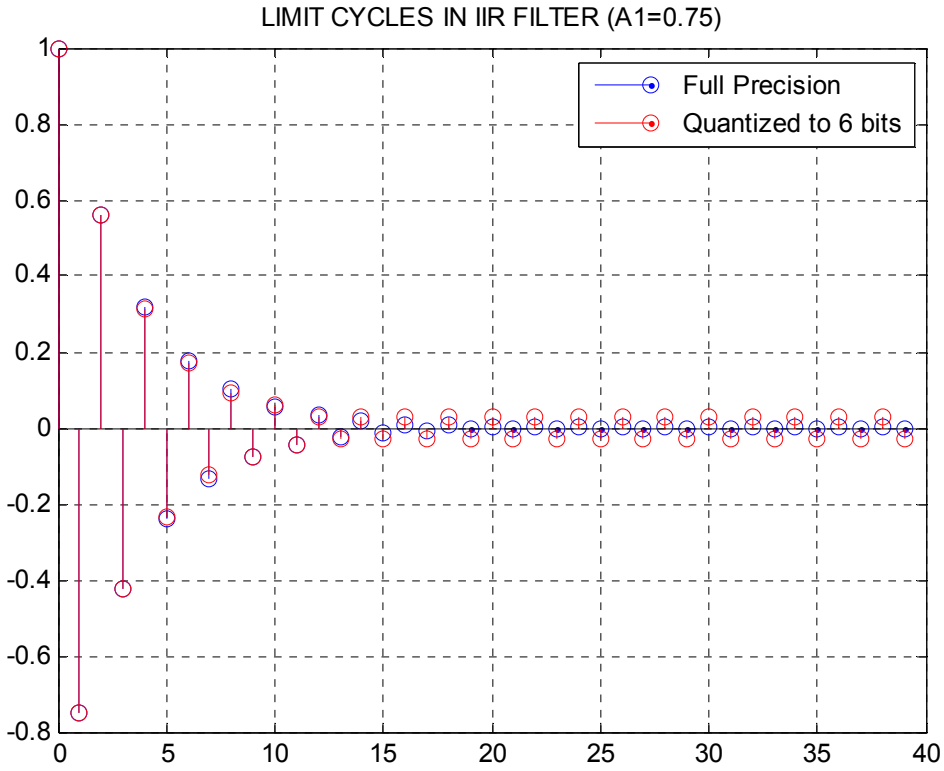
Q{ w } = `round[w *(2^# bits)] /(2^# bits)`

The function should accept the following input parameters:
1) Feedback loop filter coefficient: A_1
2) The number of bits for the multiplier output quantization: # bits
3) The length of the unit sample response sequence to compute & plot.

You will NOT be able to use the Unit Sample Response Matlab program you have already created, due to the quantization needed. However, that program should prove helpful to you for quickly designing this new function. You will also not be able to use the Matlab *filter( )* function for this. Instead, you will need to implement the difference equation computation in Matlab (probably using a "for" loop).

The two Unit Sample responses should be stem plotted on the same axes (two different colors, with a legend to identify each plot), so that you can compare the "expected" results to the results with quantization.

Include a copy of your Matlab m-file function code with your report.



b) Use your Matlab function to study the effects of changing the filter coefficient A_1 and the number of quantization bits used for the multiplier output. Observe the results for the following test cases:

1) $A_1 = 0.75$; # bits = 8
2) $A_1 = 0.75$; # bits = 7*
3) $A_1 = 0.75$; # bits = 6
4) $A_1 = 0.75$; # bits = 5*
5) $A_1 = 0.75$; # bits = 4
6) $A_1 = 0.75$; # bits = 3*
7) $A_1 = 0.75$; # bits = 2
8) $A_1 = 0.85$; # bits = 5*
9) $A_1 = 0.6$; # bits = 5*

Provide plot results in your report for the highlighted (*) test cases above.

c) **Describe your observations** about the effects of changing the number of quantization bits and the A_1 filter coefficient on the limit cycle oscillations.