

The Audio Engineer's Approach to Understanding Digital Filters

For the idiot such as myself

By Nika Aldrich

The following is a paper written for the benefit of the audio engineer who yearns to understand his/her tools better for the purpose of creating better audio recordings. It is a very elementary primer on simply how and why digital filters are used and not meant to be a text for the person who already understands these concepts.

I Introduction to Digital Filters

A digital audio system is created by taking analog voltages and sampling them in regular moments in time. As the Nyquist theorem indicates, the sampling rate indicates the highest frequency able to be captured. The number of quantization steps (quanta) yields the accuracy with which the samples are taken. A greater amount of quantization steps between peak and negative peak yields less "quantization error" and thus less "quantization noise". Therefore, more quanta gives us a lower noise floor and a higher dynamic range. Since the digital information is stored in binary code, a relationship can be made between the number of "bits" used for quantization and the dynamic range (or signal to noise ratio) of the resultant digital audio capture.

We can notice a simple observation in this type of sampling system in that lower frequency waveforms have less movement from sample to sample, as the low frequency waveforms take a long time to go from any point to any other point (such as from zero to peak). The greatest variance between two subsequent sample points for a low frequency waveform will be very small. For higher frequencies being sampled, the difference in values between any two sampling points will be much higher. For example, a full scale (maximum amplitude) waveform of one quarter the sampling frequency could have a sample point at zero followed by a sampling point at the maximum value. This would not be possible using a lower frequency wave. It would take much longer for the values to get from zero to the maximum value.

Therefore, as a general rule, the higher the frequency the greater the difference in values between subsequent waveforms. While it is theoretically possible for two subsequent values to be at the same amplitude with a high frequency waveform, the average variation between all samples will be less than the same average variation for a low frequency waveform.

The overriding premise behind digital filters is that averaging the inbound samples with previous samples creates a predictable change to the data - and thus a filter. The effect of this is easy to see on a very basic example. Notice in plot 1.0 that we have a waveform that has both high frequency and low frequency content. If we simply take every sample

and average it with the sample before it one will notice that the high frequency information gets attenuated by means of the mathematical reality that drastic changes in amplitude are reduced. Large changes in amplitude cannot exist if the two samples that have such wide differences are averaged together. The averaging of two drastically variant sample amplitudes causes the output difference to be only half as drastic, as in plot 1.2.

Notice, however, that low frequencies pass through just fine. This is because averaging one sample with one that was nearly at the same amplitude causes the output sample to be nearly the same (if not precisely the same) amplitude as the sample amplitudes that went in. This is, in effect, a simple low-pass filter.

Plot 1.0

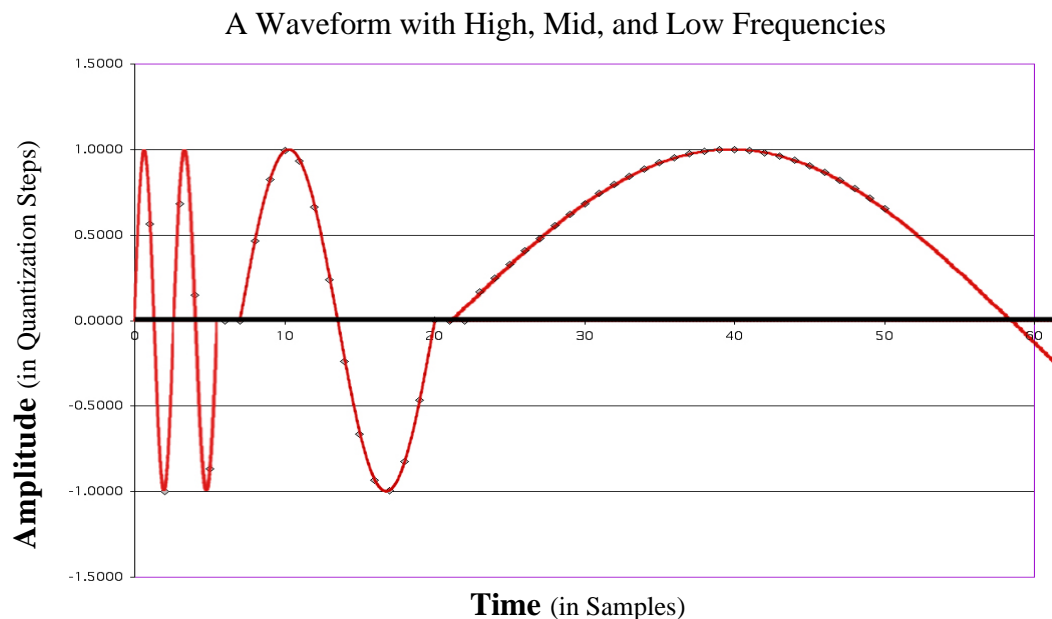


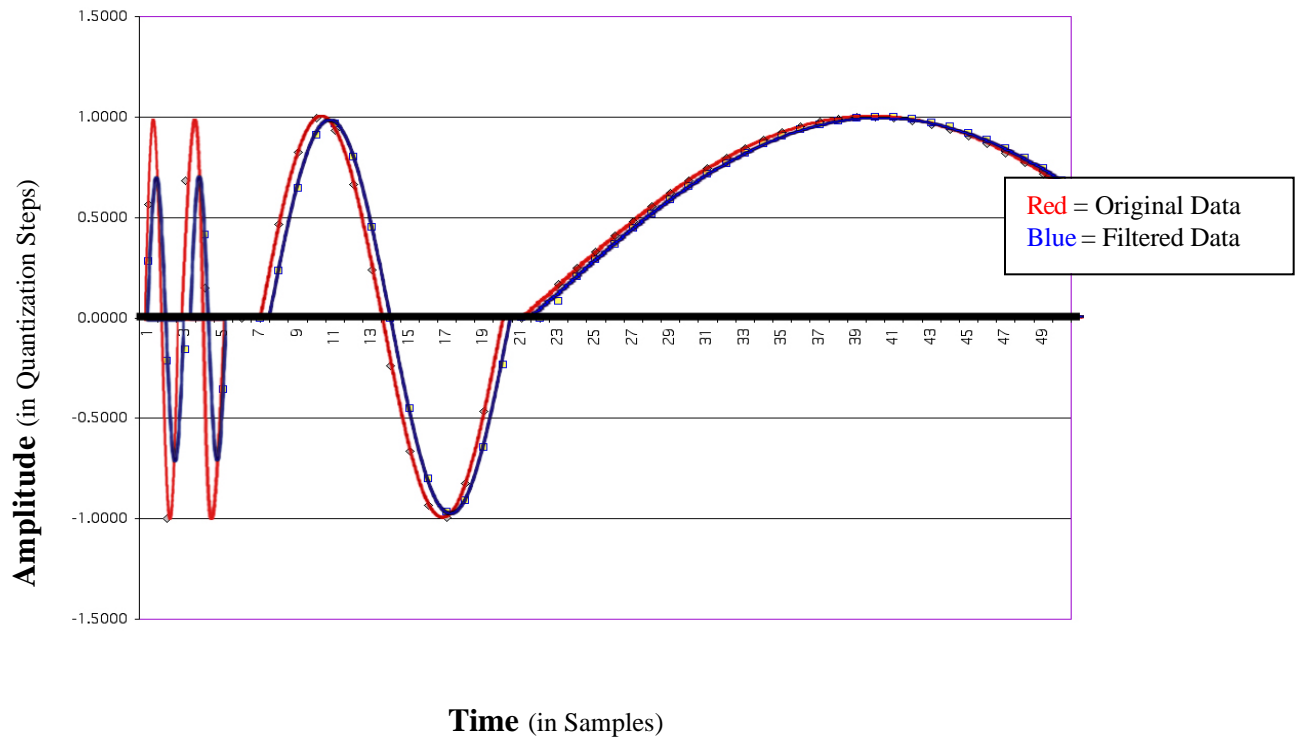
Table 1.1

Creation of Data on Plot 1.2 from Data on Plot 1.0

Sample #	Inbound Sample Value (From Plot 1.0)	Formula Average current sample and previous sample together.	Outbound Sample Value (See Plot 1.2)
1	.00	$(.00 + .00) / 2$.00
2	.56	$(.00 + .56) / 2$.28
3	-1.0	$(.56 + -1.0) / 2$	-.22
4	.68	$(-1.0 + .68) / 2$	-.16
5	.15	$(.68 + .15) / 2$.415
6	-.87	$(.15 + -.87) / 2$	-.36
7	.00	$(-.87 + .00) / 2$	-.435
8	.00	$(.00 + .00) / 2$.00
9	.46	$(.00 + .46) / 2$.23
10	.82	$(.46 + .82) / 2$.64
etc.			

Plot 1.2

The Waveform in Plot 1.0 Filtered with a Simple Low Pass Filter



We can see that this simple averaging of one sample with the sample before it yields a very simple low pass filter.

At this point it would probably be prudent to accept some mathematical terms and formulas in order to make the communication of these principles a little easier to explain. For the sake of this paper (actually, more so just common practice) x will always signify the inbound sample and y will always signify the outbound sample. If we have no processing involved at all then $y=x$. We can read this more easily as, “the outbound sample always equals the inbound sample”. Obviously when discussing filters this situation won’t occur. When we apply filters then most of the time y will not equal x .

To continue with our establishing of numerical conventions, a pair of parenthesis such as $()$ will indicate which specific sample value we’re speaking about. For example $x(1)$ would be, “the value of the first inbound sample.” $y(5)$ would be, “the value of the fifth outbound sample.”

Because we don’t often know which sample numbers we’re actually dealing with (and in order to keep our formulas a bit universal) we will most often refer to the current sample value as (n) . Therefore $x(n)$ can be rephrased in English, “the value of the current inbound sample”. When we want to talk about a sample other than the current one we’ll refer to it in reference to the current one such as $y(n-2)$. This would be “The value of the outbound sample 2 samples ago.” The very next value would have been $y(n-1)$, followed by $y(n)$, or the current sample. If we wanted to talk about future outbound samples we would refer to them as $y(n+3)$ or so. Most often we will refer to past samples.

Back to the example we had at the top of the page where we took every sample and averaged it with the sample before it we would have had the following formula:

$$y(n)=[x(n)+x(n-1)] / 2$$

Before this inflicts panic into those who get scared by math formulas, let’s break this down in plain English, “the current outbound sample is equivalent to the values of the current inbound sample plus the inbound sample before it, all divided by two.” One can see that this is the appropriate formula for averaging the most recent two samples together.

A revised way of writing this example and one that we will be using more consistently throughout this paper is:

$$y(n)=.5 x(n) + .5 x(n-1)$$

The English way of saying this would be “The current outbound sample is equivalent to half of the current sample plus half of the sample before it.” One will note that this yields the same result as the example above.

We can take a series of samples and plug them into that formula and examine the results that come out the other side:

Table 1.3

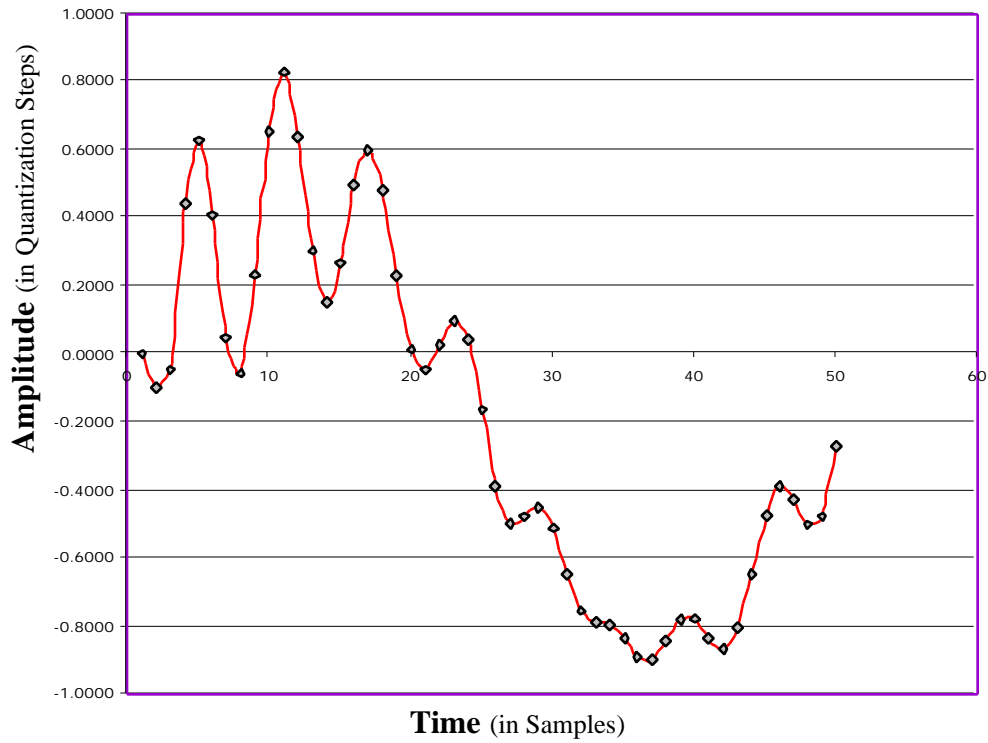
First Ten Values of Sample Data For Further Analysis

Sample #	Inbound Sample Value (From Plot 1.4)	Formula $y(n) = .5 x(n) + .5 x(n-1)$	Outbound Sample Value (See Plot 1.5)
1	.00	$(.5 \times .00) + (.5 \times .00)$.00
2	-.10	$(.5 \times -.10) + (.5 \times .00)$	-.05
3	-.05	$(.5 \times -.05) + (.5 \times -.10)$	-.075
4	.44	$(.5 \times .44) + (.5 \times -.05)$.195
5	.63	$(.5 \times .63) + (.5 \times .44)$.535
6	.41	$(.5 \times .41) + (.5 \times .63)$.52
7	.05	$(.5 \times .05) + (.5 \times .41)$.23
8	-.06	$(.5 \times -.06) + (.5 \times .05)$	-.005
9	.23	$(.5 \times .23) + (.5 \times -.06)$.085
10	.65	$(.5 \times .63) + (.5 \times .23)$.43
etc.			

The graphs of this would be:

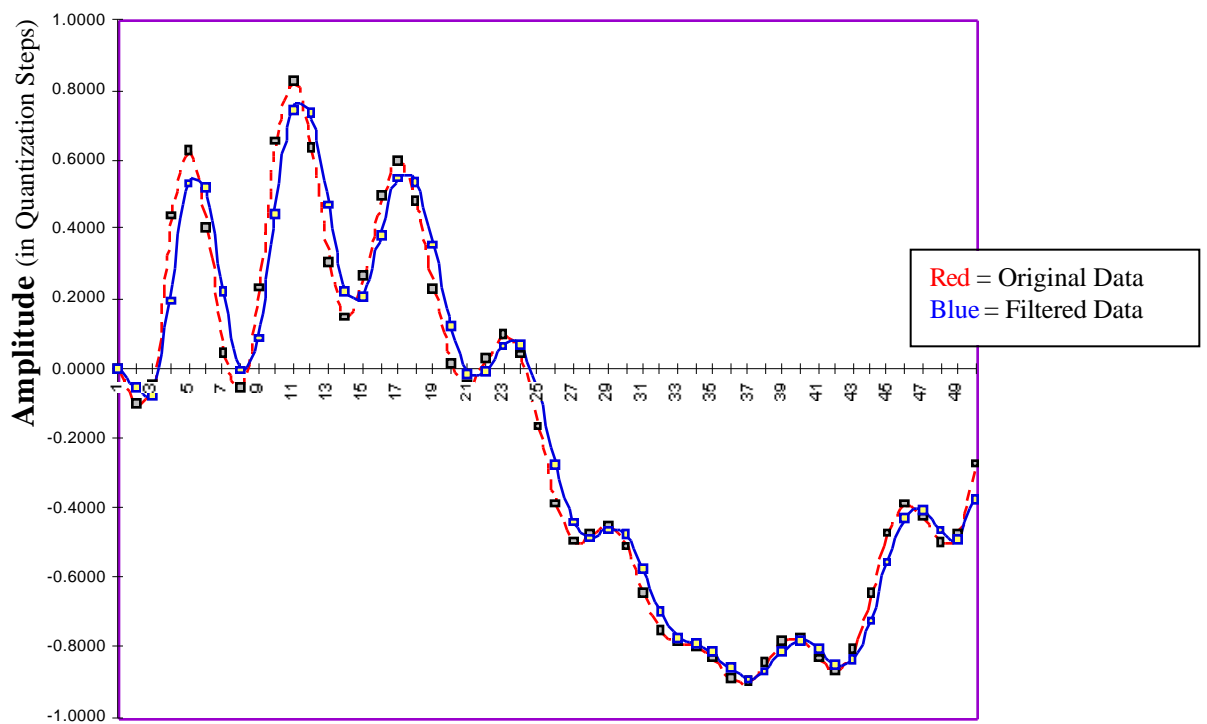
Plot 1.4

A Hypothetical Waveform



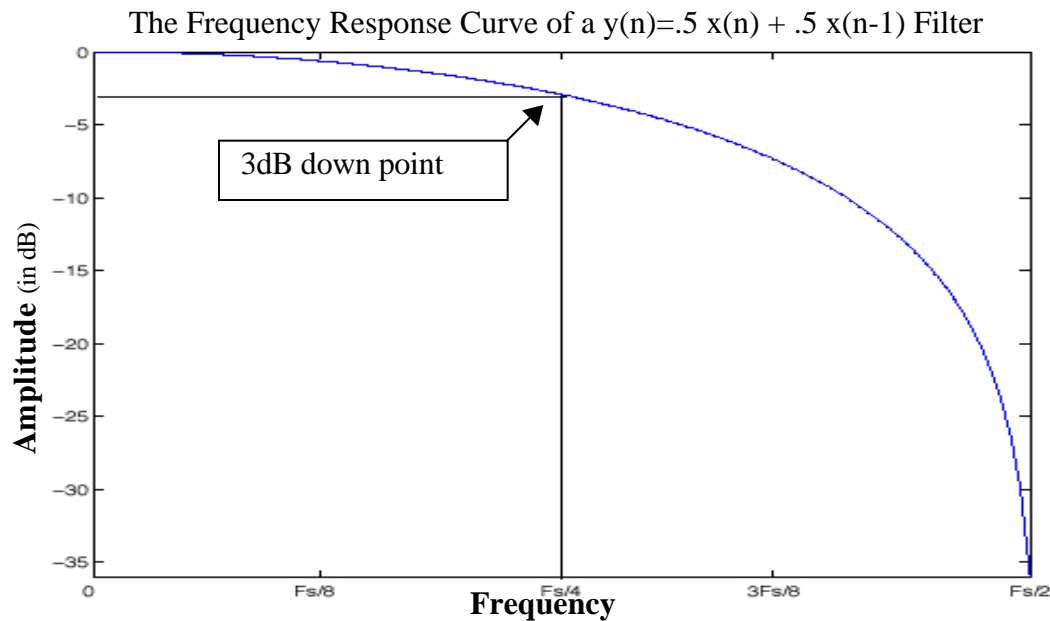
Plot 1.5

The Waveform in Plot 1.4 Filtered With a $y(n) = 0.5x(n) + 0.5x(n-1)$ Filter



It is visibly evident that the high frequencies are being attenuated, and the higher the frequency the greater the attenuation. A mathematical analysis would yield several properties about this digital filter. First, the frequency response curve of this filter is graphed below.

Plot 1.6



Secondly, this filter inherently yields a delay on the signal. In this particular filter the delay is $1/2$ sample as can actually be seen in plot 1.5. This means that waveforms that pass through this filter will come out the other side $1/2$ sample later. For a visual proof of this, observe the waveforms in Plot 1.5. Notice the location in time of the first peak of the filtered version with respect to the same peak in the original waveform. The peak occurs $1/2$ sample later.

As a minor sidebar, let's look at what would happen if we subtracted the previous sample from the current sample instead of added them together. The formula for this would be:

$$y(n) = x(n) - x(n-1)$$

Instead of, "the outbound sample is equivalent to the most recent two samples added together and divided by two," this formula would be, "the outbound sample is equivalent to the difference between the most recent two samples."

Looking at our same sets of sample data from table 1.3, the new sample values would be:

Table 1.7

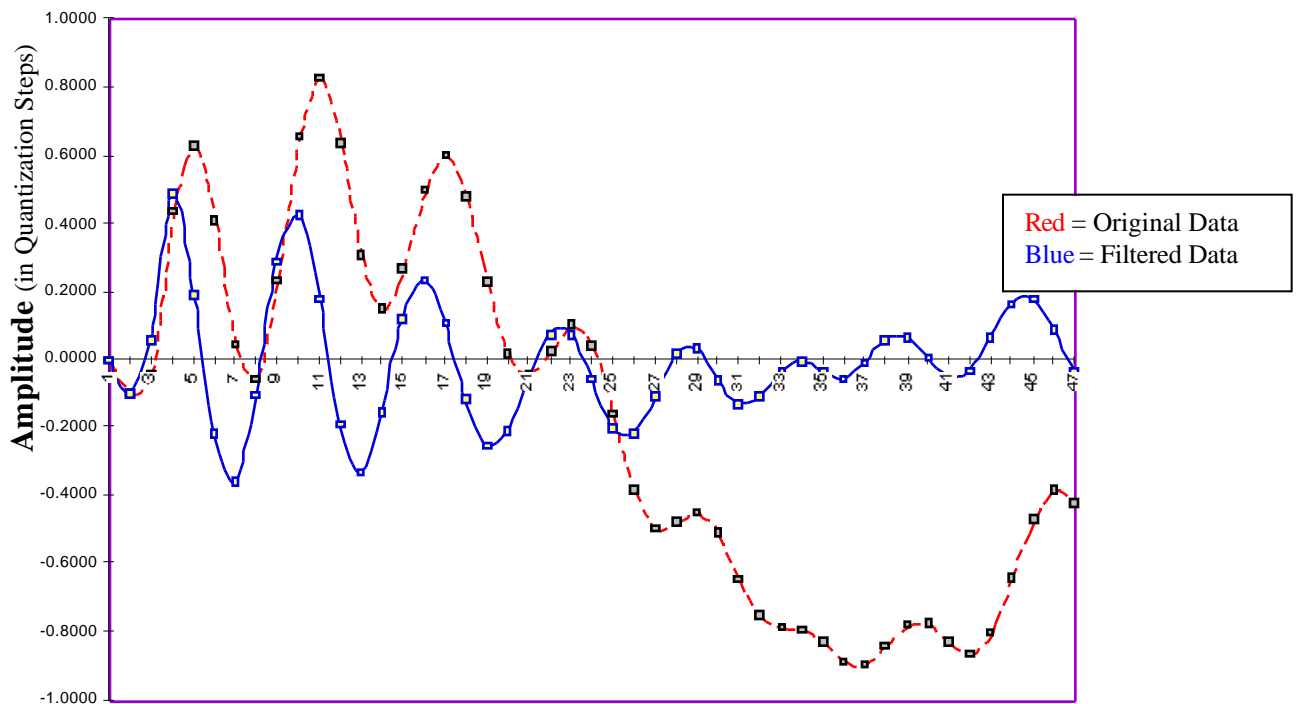
Data from Table 1.3 processed with the filter $y(n) = x(n) - x(n-1)$

Sample #	Inbound Sample Value (From Plot 1.4)	Formula $y(n) = x(n) - x(n-1)$	Outbound Sample Value (See Plot 1.8)
1	.00	(.00 - .00)	.00
2	-.10	(-.10 - .00)	-.10
3	-.05	(-.05 - -.10)	.05
4	.44	(.44 - -.05)	.49
5	.63	(.63 - .44)	.19
6	.41	(.41 - .63)	-.22
7	.05	(.05 - .41)	-.36
8	-.06	(-.06 - .05)	-.11
9	.23	(.23 - -.06)	.31
10	.65	(.63 - .23)	.40
etc.			

And the graph of this would be:

Plot 1.8

The Waveform in Plot 1.4 Filtered With a $y(n)=x(n) - x(n-1)$ Filter



Notice that this becomes a high pass filter. The difference is whether the previous inbound sample is added to or subtracted from the current inbound sample. For the sake of this paper we will not be discussing high pass filters for two reasons. First, the majority of areas where the understanding of digital filters will come in handy for idiots such as myself require low pass and not high pass filters. This includes A/D conversion, D/A conversion, and sample rate conversion. Secondly, the difference between the two types of filters is merely whether we're looking at averaging samples together or looking at the difference between them. Once one understands low pass filters, porting that knowledge over to high pass filters will be rather easy.

Back to low pass filters, we have so far looked at taking two samples and averaging them together. What would happen if we were to take three samples and average them together? This formula would look as follows:

$$y(n) = .33 x(n) + .33 x(n-1) + .33 x(n-2)$$

Or, "the current outbound sample is equal to one third of the current inbound sample plus a third of the value of the inbound sample before that plus a third of the value of the inbound sample before that one." Or, "the current outbound sample is equivalent to the average of the most recent three inbound sample values."

In order to understand the effect of this filter it is important to recognize that our previous example was really a notch filter, completely notching at the Nyquist frequency. Since there is no material in a digital system above the Nyquist frequency this acts as a simple low pass filter.

The effect of this three-sample filter is a filter with the same shape as the 2 sample average filter, but lower in frequency. It functions as a notch filter at one third of the sample frequency instead of half of the sample frequency. It is still, in effect, a low pass filter because it is not a symmetrical notch filter – it attenuates more above the center frequency than below. A plot of the frequency response of this filter can be found in plot 1.11.

Plugging in some values will yield the following sample data, starting from our same set of inbound sample data from table 1.3.

Table 1.9

Data from Table 1.3 processed with the filter

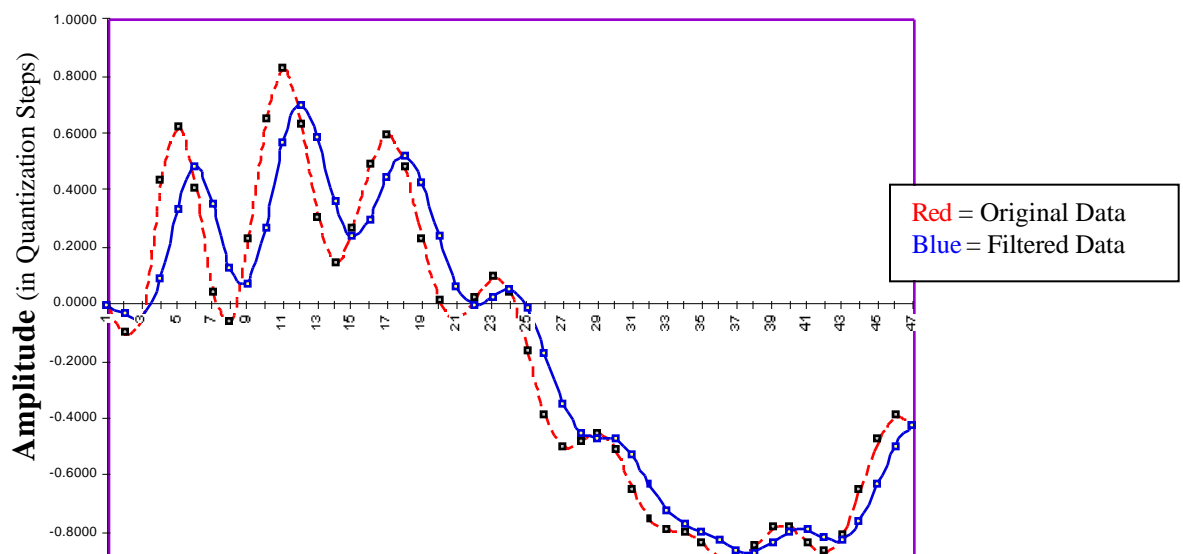
$$y(n) = .33 x(n) + .33 x(n-1) + .33 x(n-2)$$

Sample #	Inbound Sample Value (From Plot 1.4)	Formula $y(n) = .33 x(n) + .33 x(n-1) + .33 x(n-2)$	Outbound Sample Value (See Plot 1.5)
1	.00	$(.33 \times .00) + (.33 \times .00) + (.33 \times .00)$.0000
2	-.10	$(.33 \times -.10) + (.33 \times .00) + (.33 \times .00)$	-.0330
3	-.05	$(.33 \times -.05) + (.33 \times -.10) + (.33 \times .00)$	-.0495
4	.44	$(.33 \times .44) + (.33 \times -.05) + (.33 \times -.10)$.0957
5	.63	$(.33 \times .63) + (.33 \times .44) + (.33 \times -.05)$.3366
6	.41	$(.33 \times .41) + (.33 \times .63) + (.33 \times .44)$.4884
7	.05	$(.33 \times .05) + (.33 \times .41) + (.33 \times .63)$.3597
8	-.06	$(.33 \times -.06) + (.33 \times .05) + (.33 \times .41)$.1320
9	.23	$(.33 \times .23) + (.33 \times -.06) + (.33 \times .05)$.0726
10	.65	$(.33 \times .65) + (.33 \times .23) + (.33 \times -.06)$.2706
etc.			

And the graph of this would be:

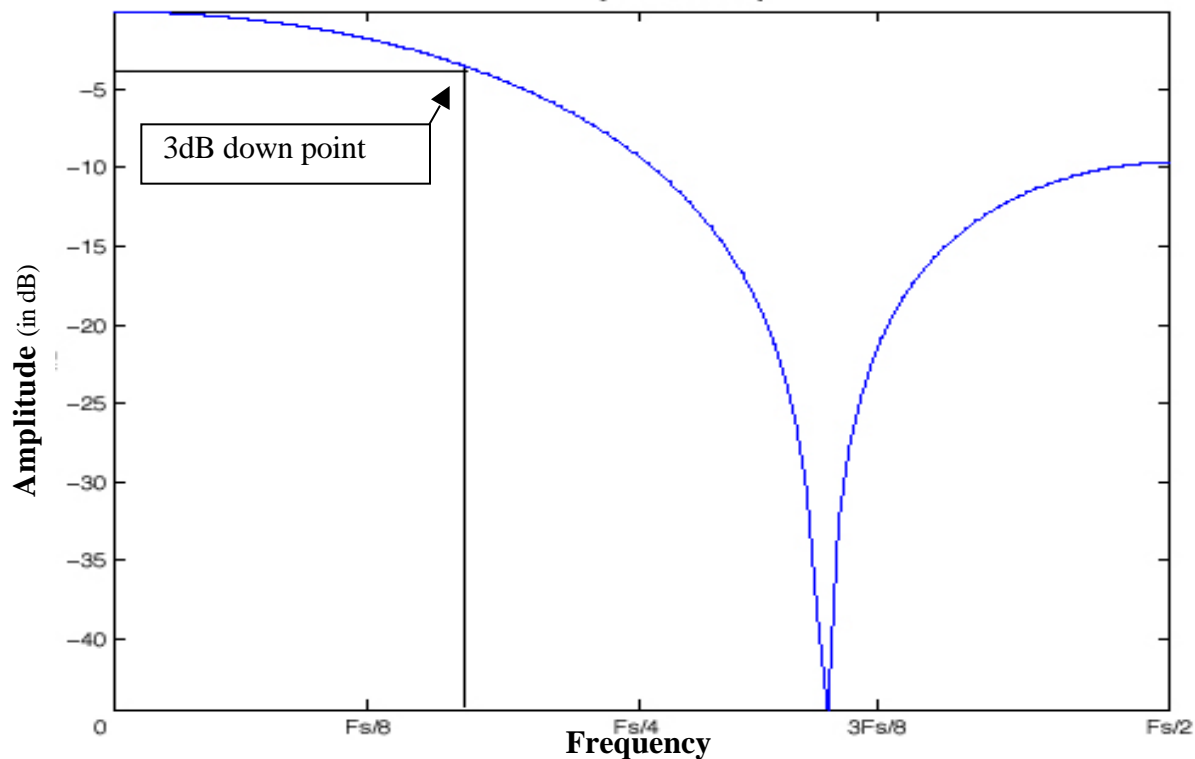
Plot 1.10

The Waveform in Plot 1.4 Filtered With a
 $y(n) = .33 x(n) + .33 x(n-1) + .33 x(n-2)$ Filter



Plot 1.11

The Frequency Response Curve of a
 $y(n) = .33 x(n) + .33 x(n-1) + .33 x(n-2)$ Filter



Let us go back for a moment to our examples where we used the following formula:

$$y(n) = .5 x(n) + .5 x(n-1)$$

Let us explore for a moment slightly modifying this formula to the following:

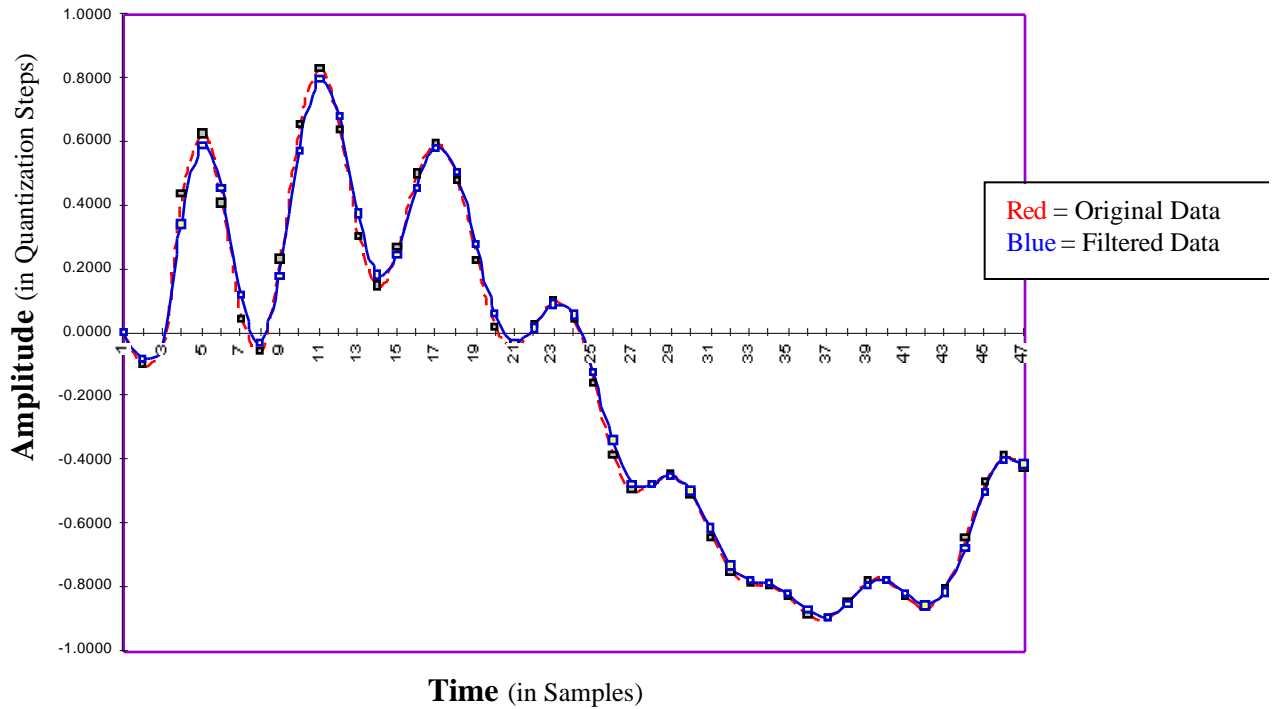
$$y(n) = .8 x(n) + .2 x(n-1)$$

Notice that this is the same as the previous formula in that we're averaging the current inbound sample with the previous sample, but in this formula we've heavily weighted the "effect" of the current sample. This means that the effect of the filter won't be nearly as steep because drastic differences in sample values can still occur, but will get slightly attenuated by averaging it with the sample before, though disproportionately. We could say that, "the outbound sample is equivalent to the current inbound sample averaged with the previous inbound sample, but with 4/5 weighting going to the current inbound sample.

The graph of this with the data in table 1.3 will yield the following:

Plot 1.12

The Waveform in Plot 1.4 Filtered With a
 $y(n) = .8x(n) + .2x(n-1)$ Filter



In this situation the numbers “.8” and “.2” are called “coefficients”. We multiply each of our inbound samples by a “coefficient”. The effect of these particular coefficients is that we are no longer really “averaging” samples together, per se. We are now “weighted averaging” samples together. “Weighted averaging” simply refers to the fact that each value does not have the same effect on the result, but rather that different samples have more effect than others on the results of the averaging.

II Getting to IIR Filters

We have now discussed two different changes to our basic formula that can have a drastic effect upon our basic formula. First we can change the number of samples that we're going to use in our "averaging" of samples to change the frequency of the filter with respect to the sample frequency. Secondly we can change the coefficients, thus giving certain samples some weighting as to the result and changing the slope of the filter. Combining these two variables we will notice that creating a more desirable filter will likely require the use of many samples, all of which will be weighted differently. Using more samples would provide us with the ability to have great control over the frequency of the filter, while tempering the effect of each sample by using coefficients in front of each sample. For example, if A, B, C, D, and E represent different coefficients, then

$$y(n) = A x(n) + B x(n-1) + C x(n-2) + D x(n-3) + E x(n-4)$$

would give us a very flexible filter design wherein the filter could be shallow or steep and at the frequency desired, all of which could ultimately be controlled by changing the values of the coefficients.

For that matter, the most flexible filter would really be:

$$y(n) = A x(n) + B x(n-1) + C x(n-2) \dots + ? x(n-n)$$

This filter could be rephrased, "the outbound sample is equivalent to the average of the current inbound sample, the one before it, and all of the inbound samples before that back to the very first sample."

The problem with using this formula is that the amount of math that will need to be done in order to calculate the outbound sample value will increase as the number of samples in our data stream continues to grow. At a sample rate of 44.1KS/s there will be 44,100 numbers to add up to figure out the sample value that happens at the end of the first second. Think of how many numbers will have to be added up for each sample that occurs five minutes into a recording!

The way that this is dealt with is by using the following formula instead:

$$y(n) = A x(n) + B y(n-1)$$

This could be restated, "the current outbound sample is equivalent to the weighted average of the current *inbound* sample and the previous *outbound* sample." This is

called a “recursive” filter in that the previous outbound sample “re-occurs” in the formula for the next sample.

For the sake of example, let’s use the simple formula:

$$y(n) = .5 x(n) + .5 y(n-1)$$

In this example we are truly averaging (not weighted averaging) the current inbound sample with the previous outbound sample. Let’s look at the effects of this filter on our sample data from table 1.3:

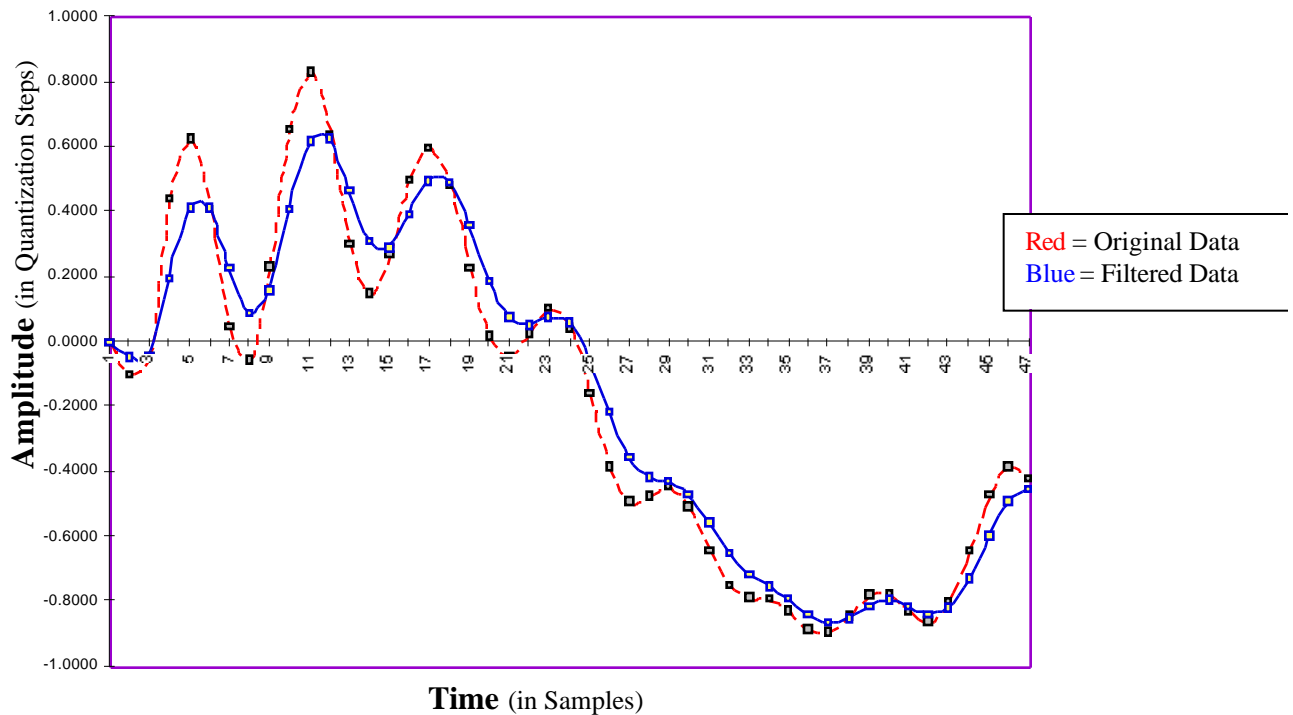
Table 2.1

Data from Table 1.3 processed with the filter $y(n) = .5 x(n) + .5 y(n-1)$

Sample #	Inbound Sample Value (From Plot 1.4)	Formula $y(n) = .5 x(n) + .5 y(n-1)$	Outbound Sample Value (See Plot 2.2)
1	.00	$(.5 \times .00) + (.5 \times .00)$.00
2	-.10	$(.5 \times -.10) + (.5 \times .00)$.05
3	-.05	$(.5 \times -.05) + (.5 \times -.05)$	-.05
4	.44	$(.5 \times .44) + (.5 \times -.05)$.195
5	.63	$(.5 \times .63) + (.5 \times .195)$.4125
6	.41	$(.5 \times .41) + (.5 \times .4125)$.4113
7	.05	$(.5 \times .05) + (.5 \times .4113)$.2306
8	-.06	$(.5 \times -.06) + (.5 \times .2306)$.0853
9	.23	$(.5 \times .23) + (.5 \times .0853)$.1577
10	.65	$(.5 \times .63) + (.5 \times .1577)$.4038
etc.			

Plot 2.2

The Waveform in Plot 1.4 Filtered With a
 $y(n) = .5 x(n) + .5 y(n-1)$ Filter



Notice that the first outbound sample is .00 because the previous outbound sample is assumed to be zero. The outbound sample is therefore half of the inbound sample. For the second outbound sample we end up taking half of the first *outbound* sample, which was half of the first *inbound* sample. Notice here that the first sample has an effect on both the first outbound sample AND the second outbound sample, but that it's "effect" is diminishing. By the time we get to the second outbound sample the effect of the first inbound sample is only one quarter (half of one half). At this point we really could simply rewrite this as:

$$y(2) = .5 x(2) + .25 x(1)$$

Or, "the second outbound sample is equal to half of the *second* inbound sample plus one quarter of the *first* inbound sample."

If we want to calculate the value of the third outbound sample we would take:

$$y(3) = .5 x(3) + .5 y(2)$$

And since we've spelled out above what $y(2)$ equals, we can plug that in as follows:

$$y(3) = .5 x(2) + .5 [.5 x(2) + .25 x(1)]$$

or

$$y(3) = .5 x(2) + .25 x(2) + .125 x(1)$$

We can see that the value of the first inbound sample still has an effect on the output of the third sample, but with a decreasing effect. One will notice that the effect of this is essentially the same as with plot 2.2 above, where we are averaging ALL of the samples together, but with more weighting on the most recent inbound samples and decreasing weighting on inbound samples from long ago. With a recursive filter such as this one each inbound sample will have an effect on an infinite number of outbound samples down the road, but with decreasing significance. This type of filter is called an “infinite impulse response” filter, or an IIR filter. The formula for a single pole lowpass IIR filter (6dB roll-off per octave) is indeed:

$$y(n) = A x(n) + B y(n-1)$$

We've already discussed that the more averaging that is done the steeper the filter can be made. While the above formula incorporates a bit of ALL previous samples in it to determine the value of a single outbound sample, it only uses two pieces of data (the inbound sample and the last outbound sample) to calculate it. This can still allow fairly drastic movements from sample point to sample point that, while flexibly controlled to be effective at a certain frequency, can still only be as steep as 6dB per octave. Creating a 12dB per octave filter utilizes 3 numbers averaged together, and the following formula is used:

$$y(n) = A x(n) + B y(n-1) + C y(n-2)$$

Or, “the outbound sample is equivalent to a weighted averaging of the current *inbound* sample, the previous *outbound* sample, and the *outbound* sample before that.” This formula is precisely the one that is used for the creation of all IIR lowpass filters in our industry. Any filter that needs a greater than 12dB per octave slope simply utilizes two of these filters run sequentially where the data goes through one filter and then another identical filter in series. Any steepness of filter at any frequency can be created simply

by utilizing multiple two pole filters, and any frequency of the filter below the Nyquist frequency can be the focal point of the filter.

At this point we should probably expound upon some additional terminology. With low pass and high pass filters there are three different “bands”. The “passband” is the area that we are NOT attenuating. With a 1KHz low pass filter the passband would be 1KHz and below. The “stopband” is the area that we ARE attenuating. On this filter this would obviously be the range above 1KHz. The “transition band” is the range in which the filter is transitioning between the passband and the stopband. On a 1KHz 1 pole filter (6dB per octave) in which we want 60dB of attenuation the transition band will be 10 octaves long (60dB of attenuation divided by 6dB per octave takes 10 octaves to be fully attenuated). In this situation the “stopband” is really not above 1KHz, but is actually the range in which the fully desired attenuation (60dB) is achieved. For this sample filter that would be 1024KHz and up (the 10th octave of 1KHz). The transition band would be 1KHz to 1024Khz

III Issues With IIR Filters

Back to our recursive IIR filter, one property of IIR filters that closely relates to filter design in the analog world is that they are not phase-linear. In other words they have phase “issues”. This is reflective of a property in filters and other processing called “group delay”. Group delay is the latency of the signal getting through the process, but is most often used in relation to different frequencies getting through that process. Some frequencies make it through some processes faster than other frequencies, and this is what causes filters to have their reputation of altering the phase of the signal as it passes through. This alteration happens increasingly as we get to the transition band. It is said that ALL filters have this affect, but this is definitely not the case. In order to eliminate this, however, the filter must be symmetrical. This means that the signal has to run through the formula once normally and then through the formula again backwards. For example, if we have ten samples going into a formula as follows:

$$y(n) = .5 x(n) + .25 y(n-1) - .25 y(n-2)$$

and our ten samples are labeled x(1) through x(10). First we would run our 10 samples in as follows:

$$y(1) = .5 x(1) + 0 - 0$$

$$y(2) = .5 x(2) + .25 y(1) \text{ which is also } .5 x(2) + .125 x(1)$$

$$y(3) = .5 x(3) + .25 y(2) - .25 y(1) \text{ which is also } .5 x(3) + .125$$

$$x(2) - .09375 x(1)$$

etc.

For the sake of example we’ll use the first 10 samples from table 1.3, and we’ll only compute 10 output samples (since this is an IIR filter it will theoretically go on forever, but we’ll stop the processing after the tenth outbound sample is produced).

Table 3.1

Data from Table 1.3 processed with the filter

$$y(n) = .5 x(n) + .25 y(n-1) - .25 y(n-2)$$

Sample #	Inbound Sample Value (From Plot 1.4)	Formula $y(n) = .5 x(n) + .25 y(n-1) - .25 y(n-2)$	Outbound Sample Value (See Plot 3.2)
1	.00	$(.5 \times .00) + (.25 \times .000) - (.25 \times .000)$.00
2	-.10	$(.5 \times -.10) + (.25 \times .000) - (.25 \times .000)$	-.05
3	-.05	$(.5 \times -.05) + (.25 \times -.050) - (.25 \times .000)$	-.037
4	.44	$(.5 \times .44) + (.25 \times -.037) - (.25 \times -.050)$.223
5	.63	$(.5 \times .63) + (.25 \times .223) - (.25 \times -.037)$.380
6	.41	$(.5 \times .41) + (.25 \times .380) - (.25 \times .223)$.244
7	.05	$(.5 \times .05) + (.25 \times .244) - (.25 \times .380)$	-.009
8	-.06	$(.5 \times -.06) + (.25 \times -.009) - (.25 \times .244)$	-.093
9	.23	$(.5 \times .23) + (.25 \times -.093) - (.25 \times -.009)$.094
10	.65	$(.5 \times .65) + (.25 \times .094) - (.25 \times -.093)$.372

Then the entire signal is run through the same filter in reverse order, so that:

$$y(1) = .5 x(10) + 0 - 0$$

$$y(2) = .5 x(9) + .25 y(10) \text{ which is also } .5 x(9) + .125 x(10)$$

$$y(3) = .5 x(8) + .25 y(9) - .25 y(10) \text{ which is also } .5 x(8) + .125$$

$$x(9) - .09375 x(10)$$

etc.

Table 3.2

Data from Table 3.1 processed in reverse order with the filter

$$y(n) = .5 x(n) + .25 y(n-1) - .25 y(n-2)$$

Sample #	Inbound Sample Value (From Table 3.1)	Formula $y(n) = .5 x(n) + .25 y(n-1) - .25 y(n-2)$	Outbound Sample Value (See Plot 3.2)
1	.372	$(.5 \times .372) + (.25 \times .000) - (.25 \times .000)$.186
2	.094	$(.5 \times .094) + (.25 \times .186) - (.25 \times .000)$.093
3	-.093	$(.5 \times -.093) + (.25 \times .093) - (.25 \times .186)$	-.070
4	-.009	$(.5 \times -.009) + (.25 \times -.070) - (.25 \times .093)$.045
5	.244	$(.5 \times .244) + (.25 \times .045) - (.25 \times -.070)$.128
6	.380	$(.5 \times .380) + (.25 \times .128) - (.25 \times .045)$.233
7	.223	$(.5 \times .223) + (.25 \times .233) - (.25 \times .128)$.138
8	-.037	$(.5 \times -.037) + (.25 \times .138) - (.25 \times .233)$	-.042
9	-.050	$(.5 \times -.050) + (.25 \times -.042) - (.25 \times .138)$	-.070
10	.00	$(.5 \times .000) + (.25 \times -.070) - (.25 \times -.042)$	-.007

It doesn't matter which direction the data is run through the filter - the filter will function the same and have the same effect, but by running the data through the opposite direction the phase shifting happens backwards, and the effects of the filter get essentially doubled. In other words, if a filter has a group delay such that at 1KHz the phase is off by + 90 degrees then running the data through backward provides the same filter characteristics, except that now the phase is off at 1KHz by -90 degrees. If the data can be run through the same filter twice, except once forward and once backward then the result will be no phase shifting at all.

There is a problem with this, however. In our example above we only had 10 samples to start with. It would be very easy to calculate new data from our 10 samples run through our filter and then do the same process in reverse followed by adding the results together. In the practical world, however, a "data stream" goes on for much longer than 10 samples. In a 3 minute piece of digital audio sampled at 44.1KHz there would be 7,938,000 samples to run through in each direction before ANY sample could be put out. This would require an "off-line" process, or one that was most definitely non-realtime. After all, in order to put out sample #1 we have to know what sample #7,938,000 is in order to

calculate the forwards and backwards results of this filter. Therefore, while it would be possible to create a linear phase IIR filter in non-realtime, doing so in real-time is nearly impossible (ignore the fact that I said “nearly” for now). This would also mean that the majority of the situations where filters are needed, such as oversampled A/D converters, would not be capable of having linear phase results. Further, as phase shift in the passband is directly related to the steepness of the filter, at 44.1KS/s sampling rates where we want to have our transition band only be from 20KHz to 22.05KHz and we want everything above 22.05KHz to be attenuated greater than 144dB (for 24 bit audio) we would need something like a 1000 pole filter which would have tremendous phase shift problems in the passband.

There is, however, a way to create linear phase digital filters.

IV Getting to FIR Filters

In order to understand how we create linear phase filters we need to look at how IIR filters work from a different angle:

First we must go back to the beginning and look at IIR filters as adhering to the following formula:

$$y(n) = A x(n) + B x(n-1) + C x(n-2) + D x(n-3) + E x(n-4) \dots ? x(n-n)$$

An IIR filter can be represented in this fashion, but only if we know how many samples we're dealing with and if we do it in non-realtime. One can recall that we don't often know how many samples we're dealing with in a datastream, so we make the actual formula reflect this lengthy formula by using previous *output* values instead of only using *inbound* sample values. As a refresher, let's look at a five sample data stream and a very simple filter. Our five sample data stream will be represented as $x(1)$ through $x(5)$. Our filter will be:

$$y(n) = .5 x(n) + .5 y(n-1).$$

Let's figure out the longhand formulas for the outbound samples:

$$y(1) = .5 x(1) + 0$$

$$y(2) = .5 x(2) + .25 x(1)$$

$$y(3) = .5 x(3) + .25 x(2) + .125 x(1)$$

$$y(4) = .5 x(4) + .25 x(3) + .125 x(2) + .0625 x(1)$$

$$\mathbf{y(5) = .5 x(5) + .25 x(4) + .125 x(3) + .0625 x(2) + .03125 x(1)}$$

$$y(6) = .25 x(5) + .125 x(4) + .0625 x(3) + .03125 x(2) + .015625 x(1)$$

$$y(7) = .5 \text{ of } y(6), \text{ etc. etc. etc.}$$

Notice that by the time we get to $y(5)$ we truly do have this formula:

$$\mathbf{y(n) = A x(n) + B x(n-1) + C x(n-2) + D x(n-3) + E x(n-4)}$$

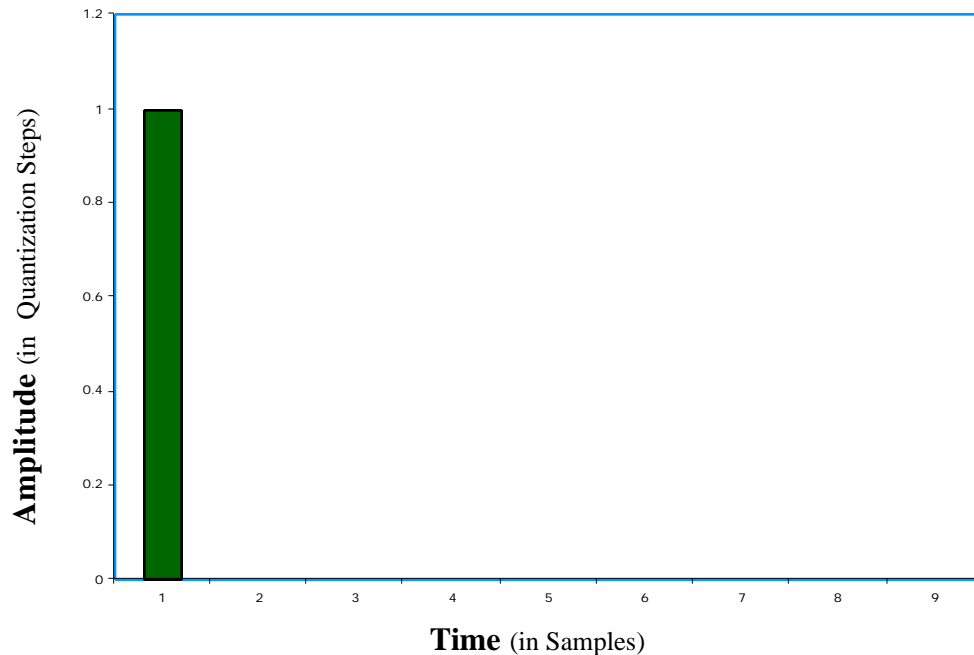
Whereas $A=.5$, $B=.25$, $C=.125$, $D=.0625$, $E=.03125$

Since we only have 5 inbound samples we really COULD replace our filter formula with the lengthy non-recursive one above so that we never used a previous outbound sample value for the next one. It is also easy to see, though, that using more sample values means much longer formulas. An infinite number of samples means an infinitely long formula

To look at it another way, let's take a hypothetical situation where we have a momentary impulse, and we're going to look at how that momentary impulse is treated by the filter and what effect that filter will have on that momentary impulse over time. To simulate this we make the first sample equal to 1 and all other samples equal to 0.

Plot 4.1

A Momentary Impulse



Another way that we can find out the coefficients that could make a non-recursive filter (only inbound sample points) from a recursive filter is to run the momentary impulse through the recursive filter and observe the results. In Plot 4.2 below the results from this pass of the impulse through the filter $y(n) = .5 x(n) + .5 y(n-1)$ are as follows:

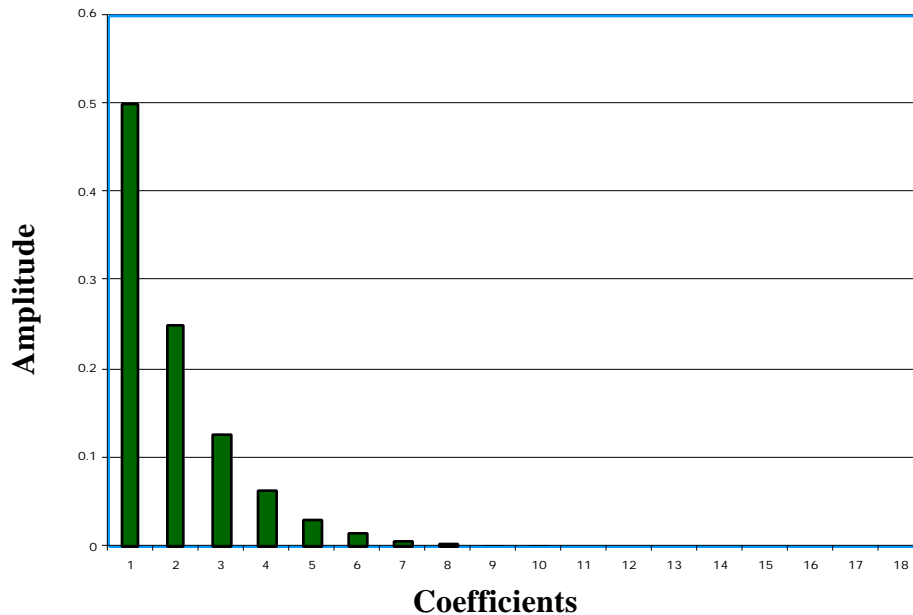
$y(1) = .5$
 $y(2) = .25$
 $y(3) = .125$
 $y(4) = .0625$
 $y(5) = .03125$
etc. etc.

This exercise shows us the effect of one sample over time, and the weight that that sample will have on future output values. This tells us that by the 5th sample this filter is only going to have a weighting of .03125. This tells us that a filter could be made using these as the first five coefficients and it would have the same results as the IIR filter - at

least for the first five samples of the IIR filter. Again, it also shows us that the more coefficients we use the closer it gets to the actual results of the IIR filter we're starting with.

Plot 4.2

Momentary Impulse Run Through the Filter $y(n) = .5 x(n) + .5 y(n-1)$



We can see that the outbound data stream will continue on forever as the output values get closer and closer to zero. This also means that the effect of the first sample is wearing off over time, but will also not ever reach zero. With the example formula above, by the time we get out several hundred samples the effect of the first sample value is going to be nearly moot, and depending upon rounding errors in our formulas it very well COULD be moot. If we accept this notion then we can stop making recursive filters and we can instead make NON-recursive filters, such that $y(n)$ will only ever equal various coefficients multiplied times $x(n-?)$. If we do this then the filter is only looking back a fixed number of samples to come up with it's result, but no more than that.

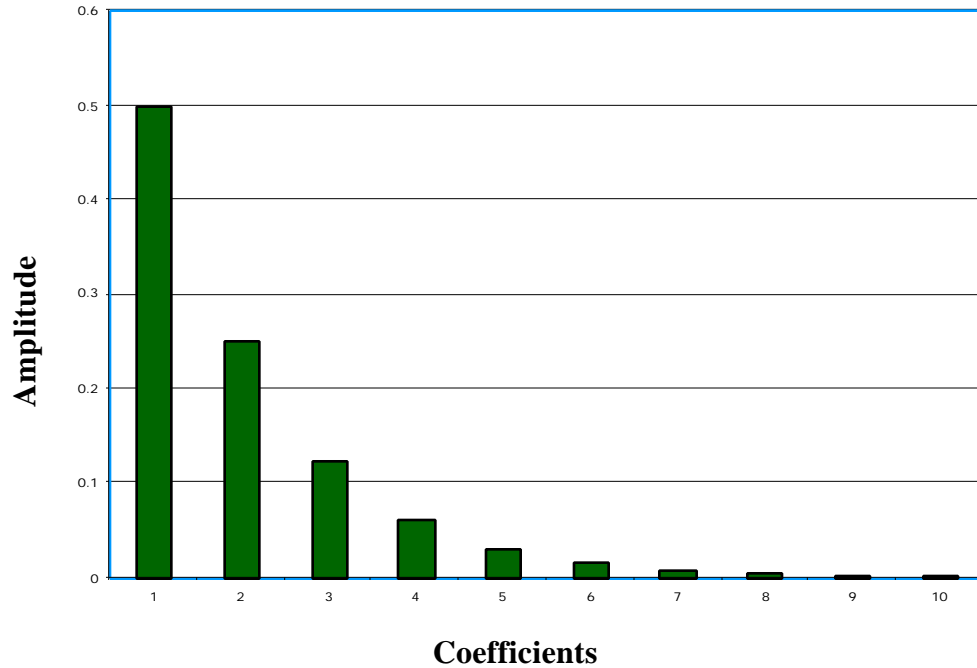
For instance, if we say that with the formula that we used above: $y(n) = .5 x(n) + .5 y(n-1)$ that by the time we get 11 samples out the first sample is effectively “irrelevant” to our answer then we can decide to rewrite this formula as the following:

$$y(n) = A x(n) + B x(n-1) + C x(n-2) + D x(n-3) + E x(n-4) \\ \dots J x(n-10)$$

Plot 4.3

The First 10 Coefficients Yielded From the Filter

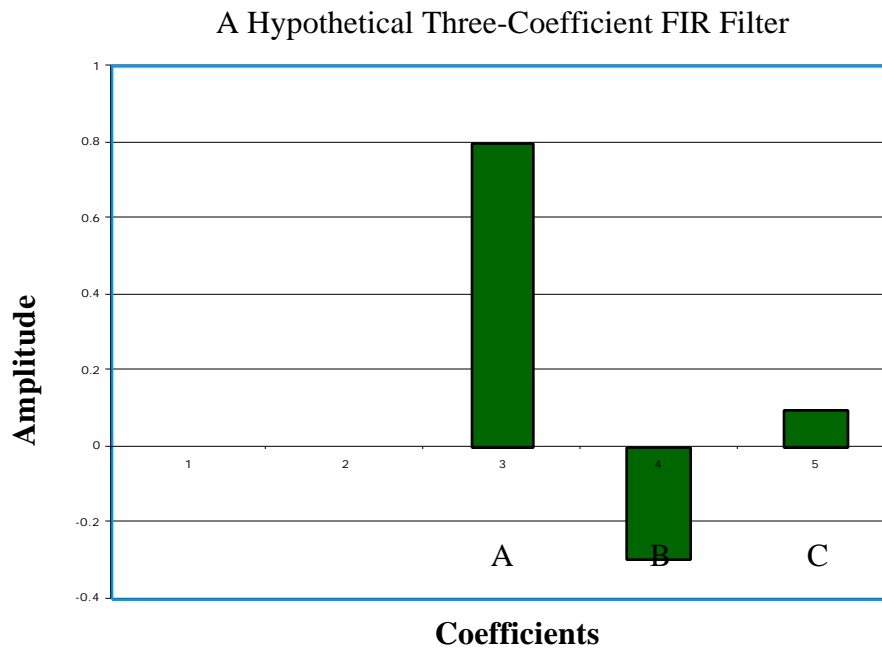
$$y(n) = .5 x(n) + .5 y(n-1)$$



And that is the end of the formula. This is no longer a formula that is looking infinitely far back to generate a sample value. It is now only looking ten samples back. Said another way, any inbound sample is no longer going to be affecting an infinite number of samples into the future with decreasing weighting. Now each sample is going to only be affecting the next ten samples out with decreasing weighting. This is no longer an “Infinite Impulse Response” filter as we are no longer affecting an infinite number of samples, but is now referred to as a “FINITE Impulse Response” filter (FIR). Since this filter is of finite duration we can now run the data through the filter both forwards and backwards so that this becomes a “linear phase filter”. The easiest way to do this is to make the filter go forwards and backwards and just let the data go through in one direction. Let’s make our example a little easier to manage. Let’s take an FIR filter that “weighted averages” the last three samples, as follows:

$$y(n) = A x(n) + B x(n-1) + C x(n-2)$$

Plot 4.4

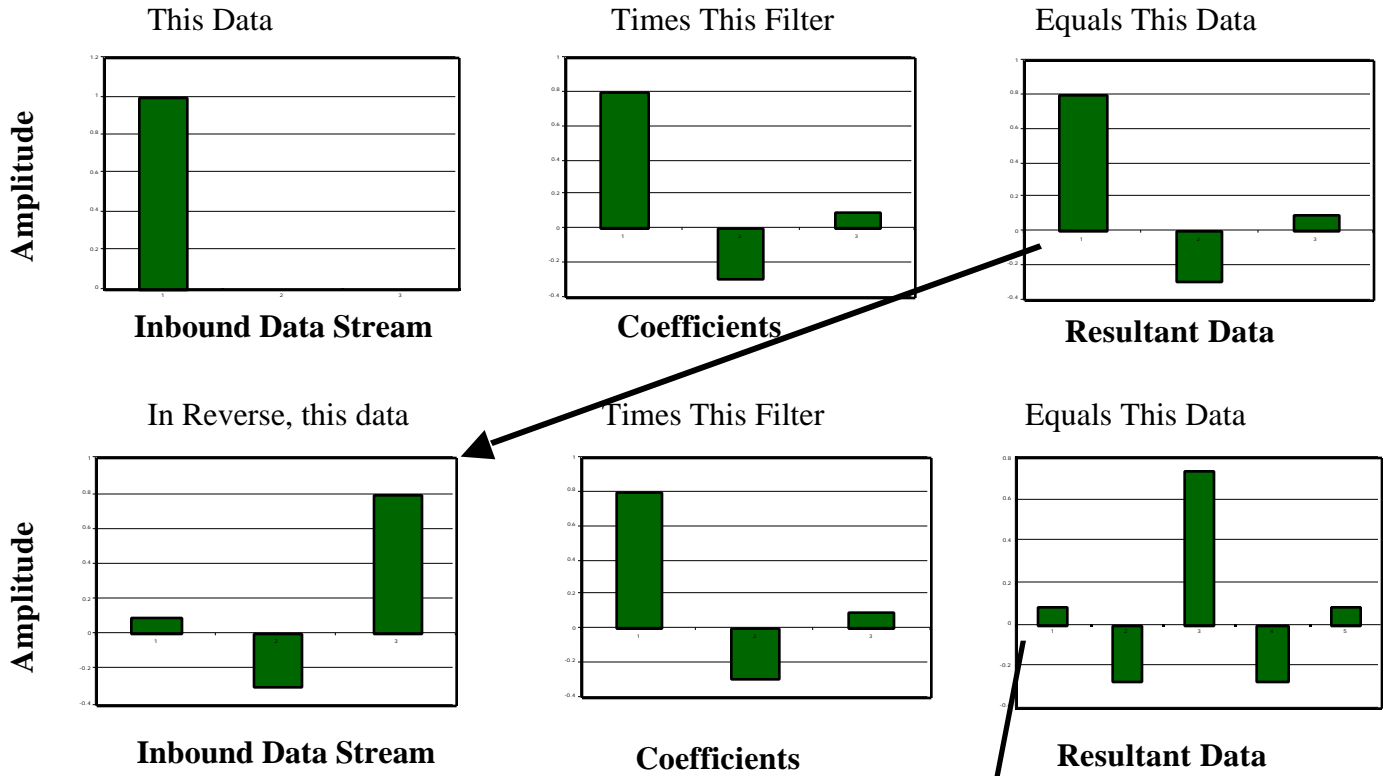


In order to make this a "linear phase" filter we have to figure out what would happen if we ran the information through this filter forwards and then backwards. In order to do that we could run our impulse through the filter, then take the results and run them through the filter again backwards. Looking at the three values above it is easy to see that if we ran our impulse through the filter above the result would be the same as the filter above. Then, if we take the result and ran it through the filter backwards we'd have a linear phase version of this filter. One would notice that this yields the method for creating linear phase filters: take the filter coefficients and run them through themselves backwards.

In the example above, the first outbound sample would equal $(C * A)$, or the last sample times the first sample. The second outbound sample would equal $(C * B) + (B * A)$. Plots 4.5 shows how a linear phase version of the filter in Plot 4.4 can be realized. Plot 4.6 yields the resulting coefficients

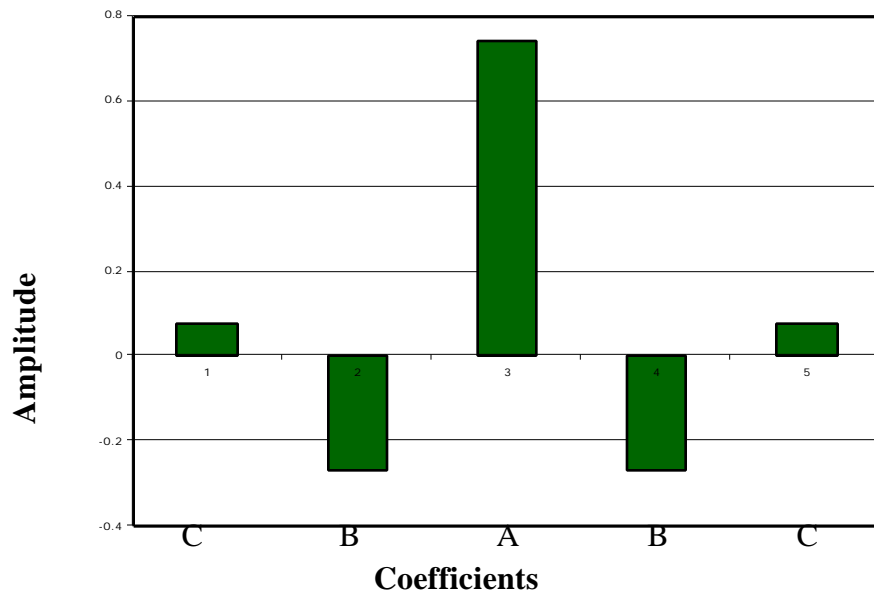
Plot 4.5

The Creation of a Linear Phase filter from the data in Plot 4.4



Plot 4.6

A Linear Phase Version of the Filter in Plot 4.4



Notice, though, that this means that we are no longer averaging the current sample with the past two samples. We are now averaging the current sample with the past two samples AND the NEXT two samples. This means that this becomes a sort of a “look ahead” and “look behind” filter in that it requires previous samples and future samples in order to properly calculate the outbound sample. This also means that this type of filter has inherent latency equivalent to nearly half of the length of the filter. In our example above we don't calculate the output until two samples later such that this FIR filter has a latency of actually 2 samples. The latency for a linear phase FIR filter will always be 1/2 of a sample less than half the number of coefficients. For example, a 13 coefficient linear phase FIR filter will have a latency of 6 samples. An 8 coefficient linear phase FIR filter will have a latency of 3 1/2 samples.

Having done this, though, we have successfully created a filter such that the data essentially goes through once and becomes phase shifted, but then goes through again backwards and becomes phase shifted back the other way so that the data has no phase shift and is therefore “phase linear”.

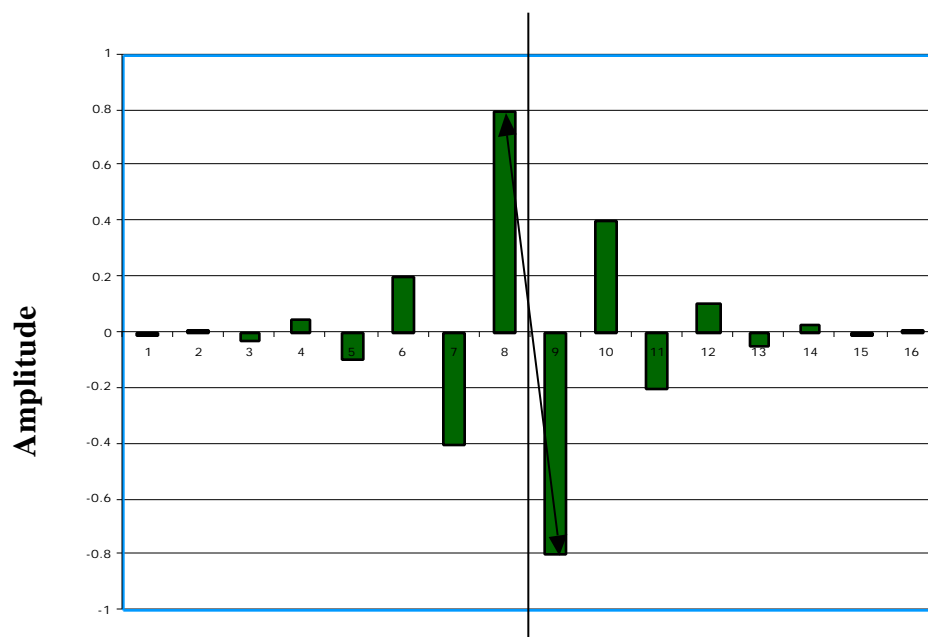
Conventionally the filter is written without future samples as follows:

$$y(n) = C x(n) + B x(n-1) + A x(n-2) + B x(n-3) + C x(n-4)$$

Not all FIR filters are linear phase filters. FIR filters can be constructed non-symmetrically, much like IIR filters inherently are. They can also be constructed anti-symmetrically such that half of it is the opposite “polarity” of the other half, such as the figure below represents.

Plot 4.7

An Example of an Asymmetrical FIR Filter



FIR filters are most useful in our industry, however, in that they can be easily made to pass data without any phase shift.

V Issues With FIR Filters

There are complications to FIR filters, however. Looking at IIR filters again for a moment we can see that the formula:

$$y(n) = A x(n) + B y(n-1) + C y(n-2)$$

only requires that the digital signal processor do three multiplication functions and then add three numbers together: coefficient “A” times the current sample, coefficient “B” times the previous outbound sample, and coefficient “C” times the previous outbound sample to that. Then the three results get added together. This very simple mathematical formula looks infinitely far back in time to calculate its results.

With an FIR filter that looks, say, 100 samples back in time to do its formula (and thus 100 samples forward in time as well for a linear phase version of this filter) that means 200 separate coefficients are used, meaning 200 separate multiplications followed by the addition of the 200 results together. This yields two problems:

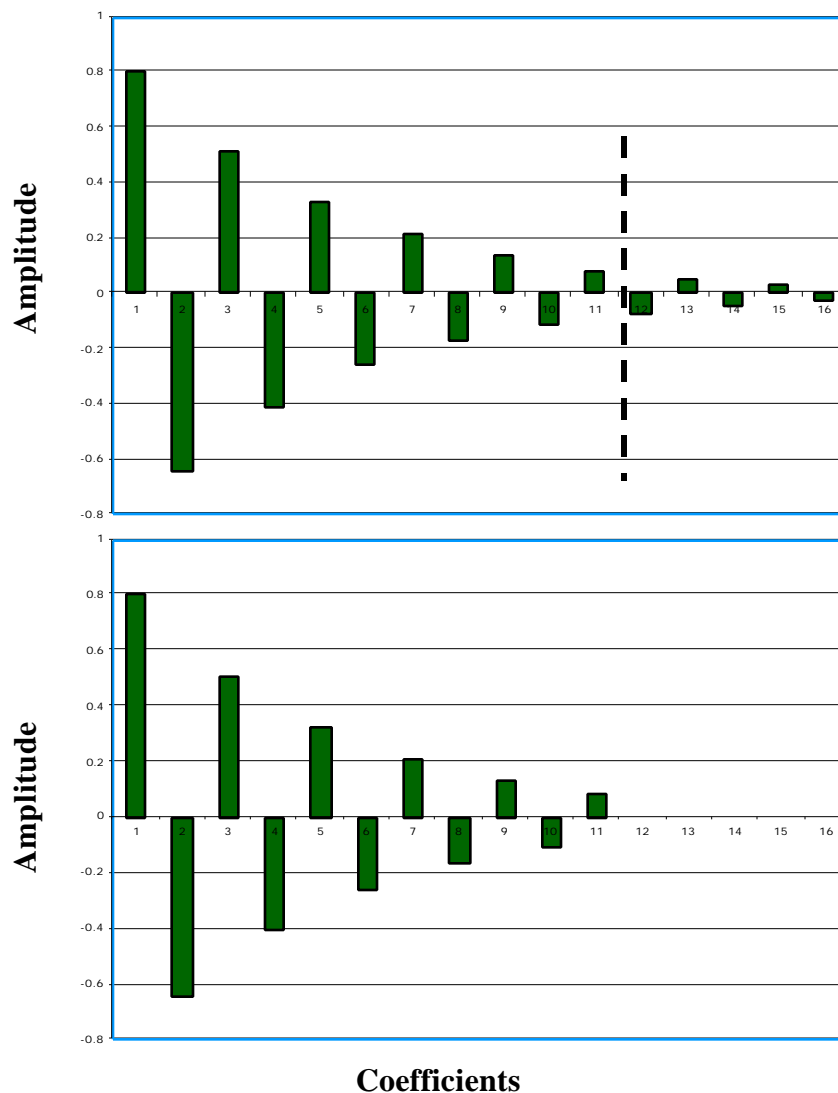
First, the accuracy of the coefficients is limited to the number of bits we allow for them. If we use 8 bit coefficients then the coefficients are limited to basically representing 256 possible steps. For very accurate numbers such as -.00784256 there would have to be significant rounding of the coefficient values if we only allowed 8 bit coefficients. For this reason we need very lengthy coefficients, but this requires higher powered processors to calculate the results. Regardless of how many bits are allowed for the coefficients, at some point the coefficient value is going to have to be rounded, and this will yield errors in our calculations. In IIR filters this is not much of an issue because the math is somewhat “automatically” done by using the previous outbound sample rather than calculating all of the coefficients back 100 samples individually. Further, the addition of 200 large numbers together (24 bit coefficients times 24 bit samples yields 48 bit numbers) requires further powerful processors. Having said this, the calculation of the appropriate coefficients is actually more crucial with an IIR filter because of the recursive nature of any errors that could be integrated into the coefficients.

Secondly, making a small change in the filter means recalculating ALL of the coefficient values. In an IIR filter a change in the frequency of the filter or the slope of the filter basically comes down to changing one of the three coefficients (or perhaps all three). In an FIR filter a small change that would be equivalent to changing one of the coefficients in an IIR filter by even a slight amount yields needing to recalculate all 100 coefficients. On the other hand, using FIR filters in A/D converters or sample rate converters is easily implementable because the coefficient values can be permanently built into ROM and never need to be changed.

Another complication with FIR filters is in establishing how many “taps” (previous samples) will be necessary. Again, with an IIR filter every sample infinitely back in time has an effect on the outbound sample, but with decreasing importance. With an FIR filter we have to essentially decide how many samples are TRULY important and only use that many taps. Depending on the frequency of the filter more taps will be required. The more taps the more accurate and flexible the results and the lower the frequency capable of being the focal point of the filter, but the longer the latency and the greater the opportunity for coefficient rounding errors. Wherever that point is we essentially have to “truncate” the filter’s effect before that (and after that for symmetrical filters). It really is desirable to select the coefficients appropriately and only use as many as are necessary to avoid the errors discussed above, but in doing so the truncation point of the filter may occur long before the coefficients are close to zero. See the example of this below:

Plot 5.1

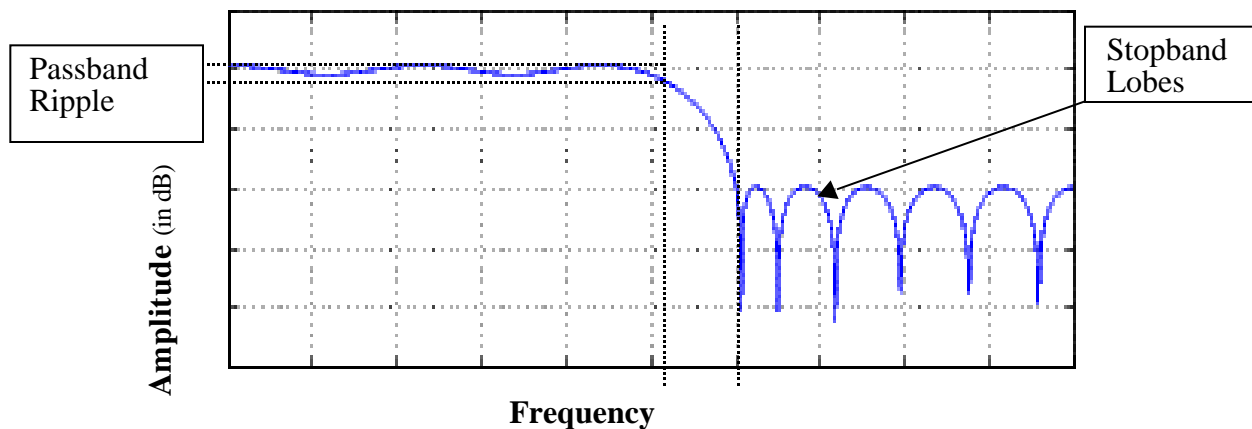
A Demonstration of Coefficient Truncation



At the point that this truncation happens there are several ways to handle the transition. One way would simply be to simply truncate the filter and allow the filter coefficients to never really get to zero. The effect of this is a very steep filter capability, but artifacts are created in the form of “lobes” in the stopband, such that the attenuation level in the stopband is not as was previously calculated. See the examples below:

Plot 5.2

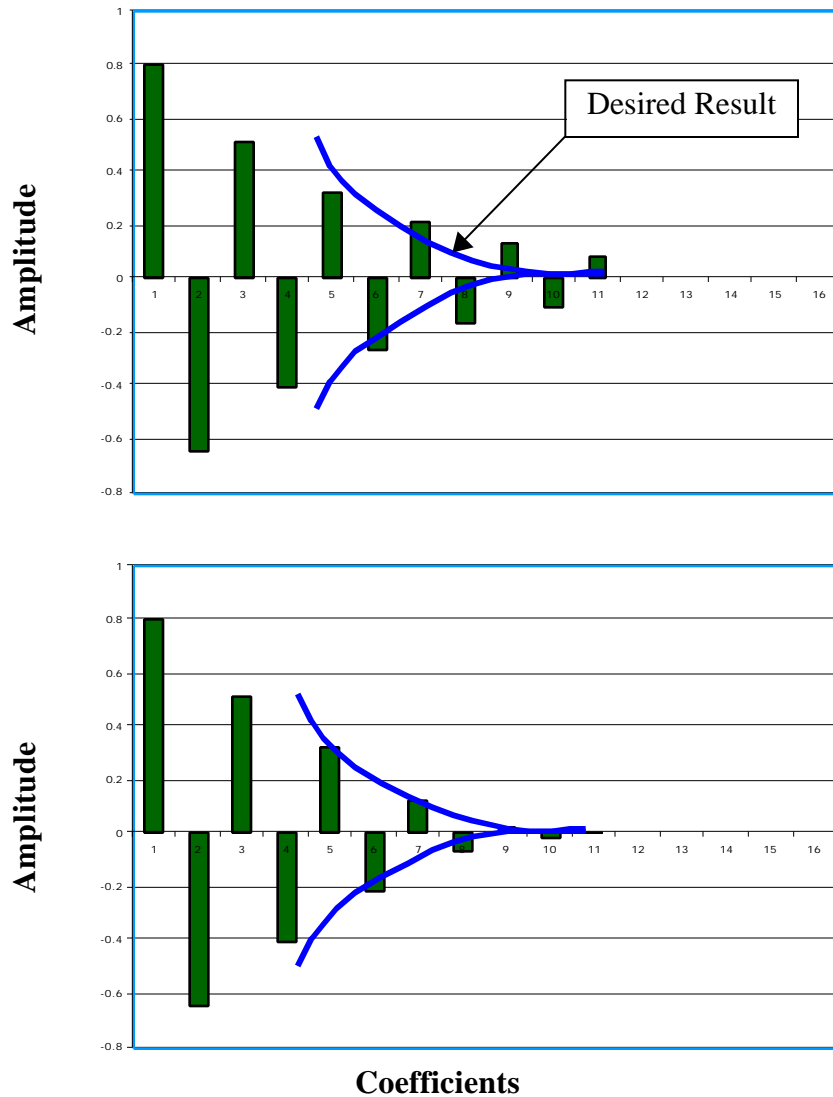
Demonstrations of Stopband Lobes and Passband Ripple



Because of this effect it is generally desirable to “force” the filter’s coefficients to actually get to zero through a process called “windowing”. There are several algorithms used for windowing, including Kaiser, Hann, Hamming, and others. These are essentially mathematical formulas to “treat” the last of the coefficients such that the FIR’s filter coefficients do smoothly get to zero, simulating the IIR filter’s coefficients getting to zero in an infinite amount of time. The different formulas all have a balance between the steepness of the filter and the size of the lobes in the stopband. A simple truncation of the filter without the use of any of the above techniques is called a “rectangular window”. A visible representation of the effect of windowing is shown below:

Plot 5.3

A Demonstration of Windowing



Another side-effect of overly shortened FIR filters that is often talked about is “passband ripple”, a sort of undesirable effect that is audible in the passband such that some frequencies are slightly attenuated in the passband causing a form of “ripple” in the frequency plot. See an exaggerated example of passband ripple in Plot 5.2. Passband ripple is mostly a byproduct of the method that is used to calculate the coefficients in most filters and can be minimized or decreased infinitely if appropriate steps in filter design are used.

VI Practice and Applications

IIR and FIR filters are the crux of most digital signal processing and this paper has only skimmed the surface of how they are created and why they are used. There are a couple of applications that can be dealt with in slightly more detail that are still appropriate for this paper that may further help to clarify the differences and the practical uses of these filters.

Equalization

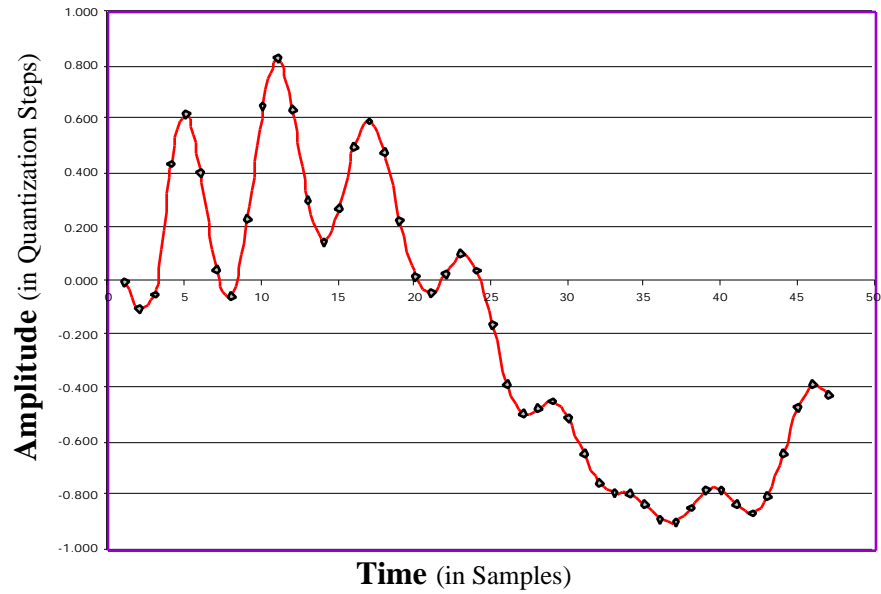
Digital EQ's such as plug-ins or hardware boxes use almost exclusively IIR filters. At the time of this writing there are only a handful of DSP EQ algorithms for the audio industry that have any form of FIR filters in them as processing tools, and in these situations they are pretty much fixed-setting low pass or high pass filters that have almost no variable parameters. In every other situation, however, when a user alters a setting on their EQ (or digital crossover or multiband compressor, etc.) they are essentially telling the CPU to calculate a few new coefficients to change the effect of the IIR filters. The difference in frequency, cut/boost, Q, or shape of the filter (shelf, bell, cut, etc.) is essentially controlled by the alterations of very few coefficients.

Analog to Digital Converters

A/D and D/A converters use almost exclusively FIR filters. A/D conversion is mostly done at sampling rates of 64x or higher. The audio would be sampled at perhaps 2.822MS/s for a resulting 44.1KS/s resultant data rate. The reason that these "oversampled" A/D converters are used is that designing very steep linear phase analog filters for anti-aliasing is impossible, whereas we now know that designing one in the digital domain is fairly simple (in concept, at least). The 2.822MS/s material is fed through FIR filters that manage to provide 144dB of attenuation at 22.05KHz while allowing everything below 20KHz to pass through unaffected, all with linear phase and minimum amounts of passband ripple. The method used is simply taking the 64x material and filtering out all material above the new Nyquist frequency and then removing all but every 64th sample. This process is called "decimating" and is shown below.

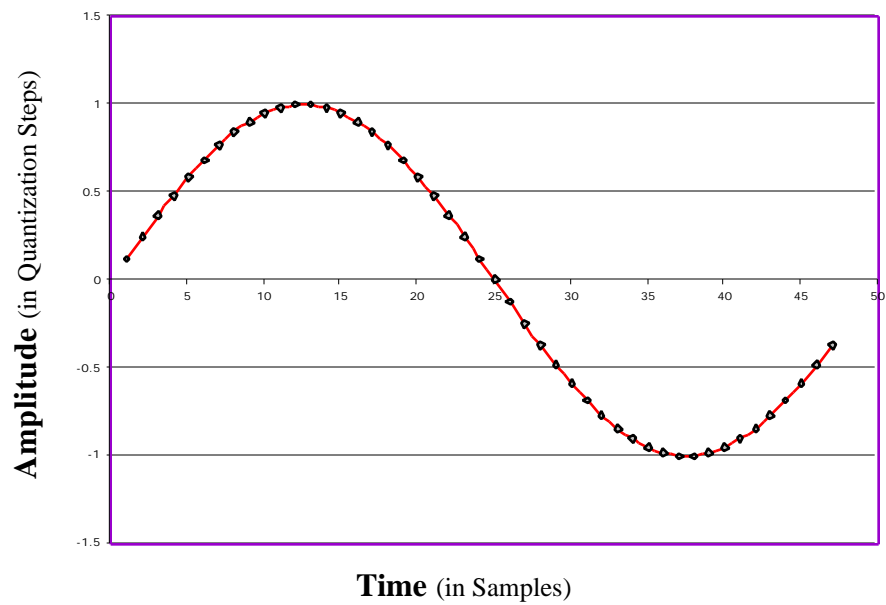
Plot 6.1a

Sample Data at 16x the Eventual Sample Rate ($16f_s$)



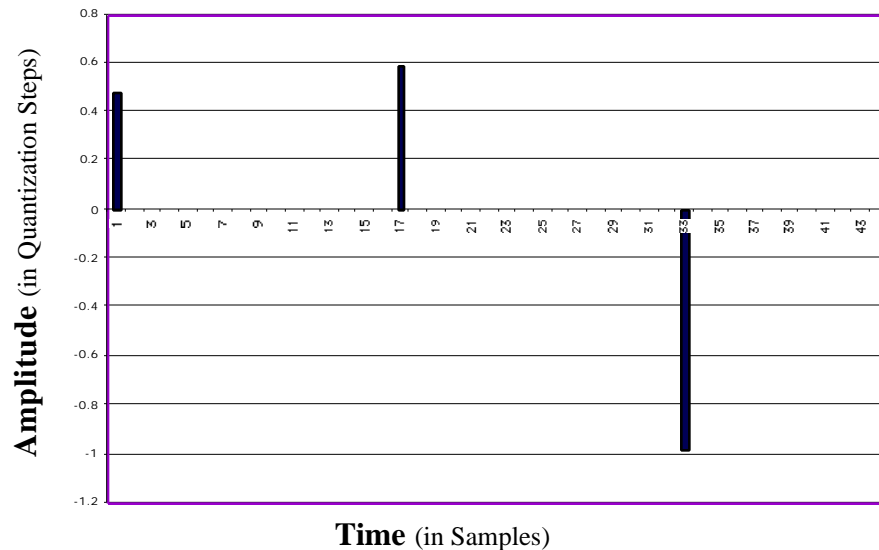
Plot 6.1b

Sample Data from Plot 6.1a Filtered to $(1/2)f_s$



Plot 6.1c

Sample Data from Plot 6.1a Fully Decimated



The reason that "filters" (plural) are used is that it has been theoretically demonstrated that the calculation requirements can be reduced to their minimum if multiple stages of filtering are used instead of one, long filter. For this reason the work is done by a series of small filters. The data may pass through a series of 10 tap filters serially that effectively filter the material from 2.822MS/s down to 96KS/s. The material would then pass through a larger filter - perhaps 100 taps - that would perform the critical stage of filtering the material down to it's desired final state.

Sometimes a type of FIR filter called a "Canonical Integer Coefficient filter", or CIC, is used. This is a type of filter that only utilizes whole number integer coefficients that are powers of 2, such as 2, 4, 8, and 16. An FIR filter using only these values as coefficients can significantly ease the processing load on the CPU because no multiplication has to happen. Instead the 24 bit sample values simply get shifted to the right or the left: a multiplication of a binary number times "2" simply adds a zero on the right (times "4" is two zeros, etc.). The beginning stages in the complex downsampling process in an A/D converter can be achieved using DSP friendly CIC's, though the results would be inadequate if not passed through a larger, more accurate, final FIR filter.

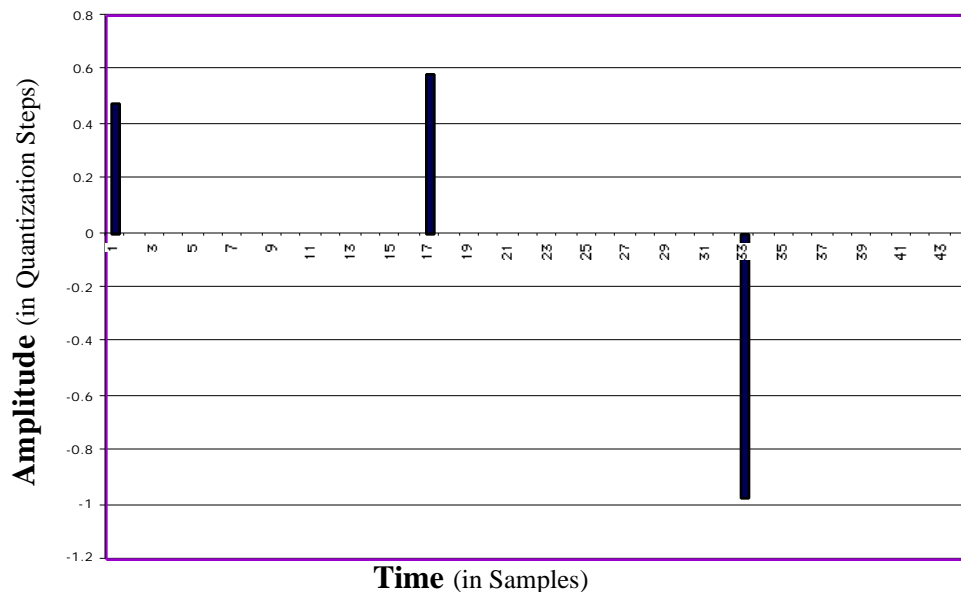
The digital "decimating" downsampling filters used by most A/D converter manufacturers are designed onto the chips provided by the converter IC manufacturer (such as AKM,

Burr-Brown, or Crystal). A few manufacturers are dissatisfied with the corners cut in the designs of these filters and the expenses spared to put these filters onto the inexpensive chips that come from the IC manufacturers. These manufacturers are known to take the data off of the IC chip while the data has not been filtered down all the way (if at all) and then supply their own IC chip that is used exclusively for the filtering process using a higher powered DSP IC and thus more accurate filters. This type of processing is often done on more expensive A/D and D/A converters because the math involved in creating superior FIR filters to what the converter IC manufacturers can supply is very involved and not necessarily conveniently spelled out in text books for the manufacturers. Further, the implementation of this and the cost of the additional DSP power can drive the price of these converters up. Many converter IC manufacturers don't provide the ability to tap off of the chip prior to the filters' being implemented, making choosing the appropriate converter chip for this type of modification crucial.

D/A converters use lowpass FIR filters as well in order to eliminate the effects of the "stair stepping" created by digital sampling. Oversampling is used in D/A converters about the same as it is in A/D converters. As the material is upsampled to many times the sample rate and FIR filters filter out high frequency information thus smoothing out the "stair stepped" sample data into analog waveforms again that are linear in phase to the original material. The process of oversampling is exactly the opposite of decimating - additional sample values are inserted into the material and the result is then filtered until those new values are where they accurately represent the original waveform. In a 16x oversampling D/A converter 15 additional samples are placed between every two sampling points, often at the zero as the example will show below.

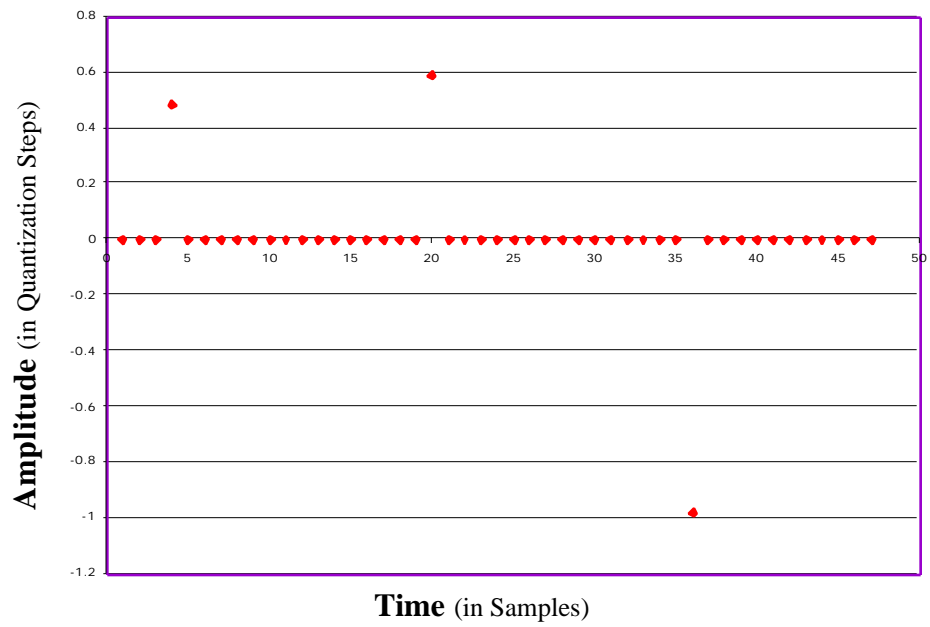
Plot 6.2a

Sample Data from 6.1c



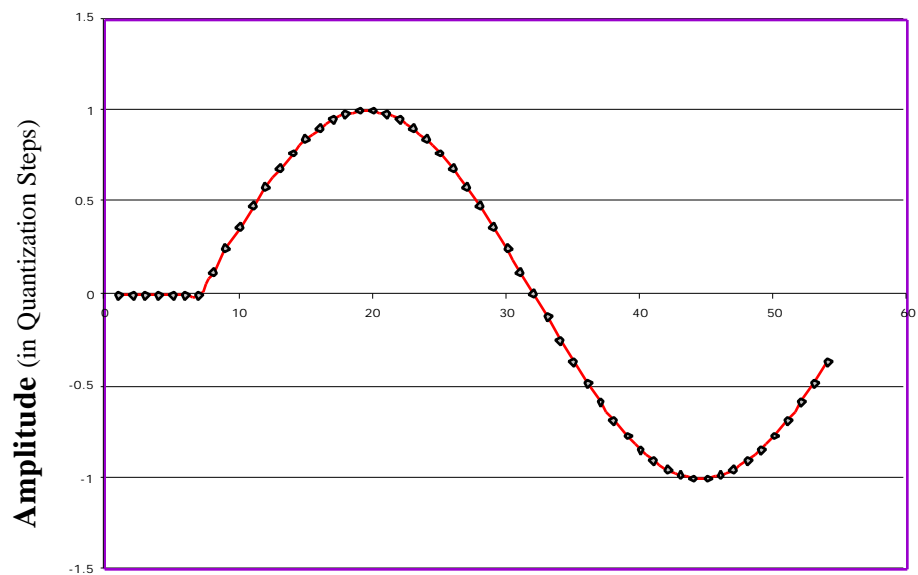
Plot 6.2b

Sample Data from 6.1c Upsampled to $16f_s$



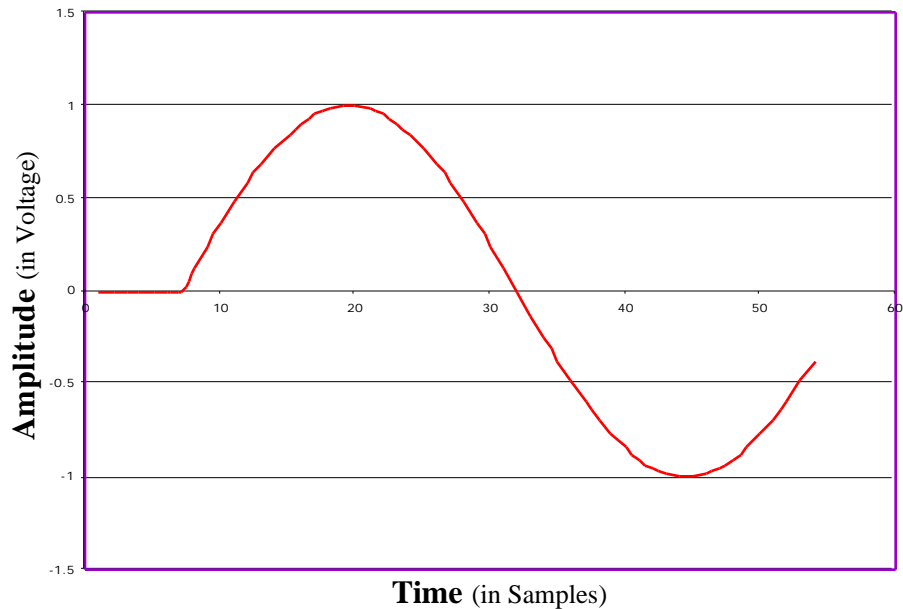
Plot 6.2c

Sample Data from 6.1c Upsampled and Filtered to $16f_s$



Plot 6.2d

Sample Data from 6.1c after Digital to Analog
Conversion and Further Analog Filtering



As can be seen above, when the material is oversampled enough an analog filter can finally finish the task at the output of the D/A. This analog filter can be well enough above the audible spectrum that it won't affect the audio.

Sample Rate Converters

Sample rate converters (SRCs) use FIR filters as well. There are two types of sample rate converters. Synchronous SRCs simply take the data coming in and convert it using a certain fixed ratio to the new sample rate for the output. Asynchronous SRCs constantly analyze the inbound clock rate and not only convert the material to a new sample rate, but “re-clock” the information as well. If the inbound data is at 48KS/s +/- 150 parts per million (ppm) then the asynchronous SRC will convert the slightly varying clock rate to exactly 44.1KS/s while the synchronous SRC will convert the data to only 44.1KS/s +/- 150 ppm.

Synchronous SRCs are rather simple to understand, though still certainly challenging to audibly transparently design. There is a mathematically determinable ratio between any two sample rates: between 44.1KHz and 48KHz the ratio is 147 to 160. All that need be

done is upsample the data the appropriate amount using FIR filters to remove the necessary amount of quantization distortion, then downsample or “decimate” the data back down using the appropriate sample rate, also using an FIR filter (or series of FIR filters). There will be a temporary new sampling rate of the lowest common multiple of the two original sampling rates (as well as a new temporary Nyquist frequency). For sample rate conversions from or to 44.1KS/s and 48KS/s the temporary new sample rate is 7.056MS/s. The temporary Nyquist frequency is 3.528MHz. For example, going from 48KS/s to 44.1KS/s one would add 146 sampling points between each pair of existing points. Then the material is filtered using linear phase FIR filters to eliminate all of the quantization noise between 24KHz and 3.528MHz. At this point there is a series of sample data that has a sample rate of 7.056MS/s but essentially a brickwall filter at 24KHz. The material is then decimated using FIR filters to get it to 44.1KS/s which involves removing 159 of every 160 samples. While this is done, however, a new series of FIR filters must be used because the new Nyquist frequency of 22.05KHz is lower than the original Nyquist frequency of 24KHz.

The basic formula here is to add new sampling data then filter it. Then filter it again and remove some sample data.

Asynchronous SRCs are very much more involved and will not be covered any more extensively in this paper.

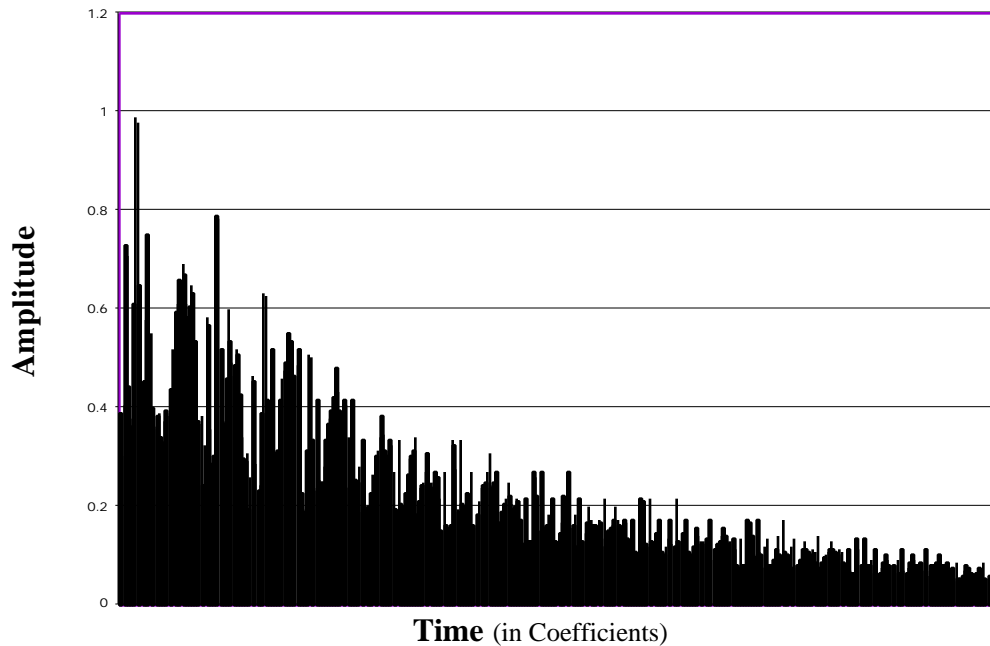
Since synchronous SRCs are as conceptually simple as the filtering done in A/D and D/A converters it is possible to design ones that are audibly transparent, just as A/D converters can be. Asynchronous SRCs, on the other hand, are more difficult to design well and are therefore often audibly inferior, though functionally a requirement in many situations. The quality of a synchronous SRC basically comes down to the quality of the implementation of the FIR filters used in their design.

Convolution Reverbs

While semantically debatable, the new reverb boxes hitting the market called “sampling reverbs” are also examples of where FIR filters are used. In traditional reverbs an algorithm is used much like the algorithms used in IIR filters. In fact, they are a series of IIR filters, gates, and delays. These algorithms simulate the natural means in which sound reverberates around a room as have been studied by acousticians for decades. Convolution reverbs such as the Sony DRES777 take samples of actual acoustical spaces and create FIR filter algorithms that mirror the reverberation in those acoustical spaces. If a single impulse of acoustical energy is played into one of those rooms a sample can be taken of that and then used as a set of filter coefficients for a lengthy FIR filter. “Lengthy” is an understatement, and an understanding of this will explain to some degree why this type of reverb box is expensive to manufacture.

Plot 6.3

A Set of Coefficients for a Small Reverb Sample



Previously we looked at filter algorithms that looked at 100 or 200 samples and explained the process demands necessary and the problems with adding the numbers for such lengthy FIR filters. For a reverb effect that lasts for 5 seconds the amount of sample points generated (and thus the number of taps used for the FIR filter) would be $44,100 \times 5$, or 220,500. This means that 220,500 multiplications of a coefficient times a sample value coming in, followed by the addition of those 220,500 results would be necessary in order to generate a single sample. Not only does this require an inordinate amount of DSP power, but is tricky to do at all with today's DSP chips. While the math involved in this type of reverb box is much simpler (conceptually) than the acoustical and mathematical models "created" for traditional reverb boxes, the implementation is still very difficult to accomplish because of the inordinate amount of brute-force simple math that needs to happen concurrently. Often times not all 220,500 values are used and many of them are simply simulated and the formula is reduced in size.

Convolution (or "sampling") reverbs do not use FIR filtering algorithms in order to become phase linear (as they aren't phase linear) but rather because there is not a predictable and mathematically identifiable relationship between all of the coefficients. If

there was then an IIR filter would work fine, but every sample of the sampled acoustical space would have to be somehow relateable to the previous samples in a fashion likened to a formula such as $x(n) = A x(n-1)$ or $x(n-1)^2$ or something consistent that could be held to be true for any sample at all in relation to the previous samples. Since this doesn't seem to be the case with real acoustical spaces we need to take each coefficient on it's own and thereby create a very long FIR filter instead, which does yield a nearly unweildy amount of data to process.

When properly used and when quality samples of existing rooms are used the results are simply stunning. This type of reverb is very likely to come down in price as DSP power becomes faster and less expensive. As there is not a fixed "formula" that is trademarkable by manufacturers such as Lexicon or TC Electronic this type of reverb box is sure to become less expensively available as other manufacturers enter the market with competition to the few that are out now. The sampling of the desirable rooms, however, may be logistically less possible and may still end up being more expensive by the time they hit the market. For foley work and less "refined" acoustical spaces such as bedrooms, bathrooms, racquetball courts, cars, etc. practically anyone could sample these spaces and use them on their convolution reverb box when they become more readily available. Rooms such as famous cathedrals and concert halls will probably be reserved for those who have access and the samples will continue to be sold for a fee.

These are but a few examples of where and why these filter designs are used in everyday DSP.

VII Conclusion

Clearly this paper has not been designed for the computer engineer, the digital programmer or the mathematician. It has rather been written at a very elementary level for the benefit of the audio engineer who has a basic curiosity about how these digital functions work. This paper has merely skimmed the surface of a science of designing filters that has been in development for nearly 50 years. There are books that are extremely comprehensive that can take the enthused reader further into the depths of the high levels of math required for actually designing these filters. This has merely been a primer for HOW they work and not how to make them. It is the author's desire that this bit of understanding helps to yield engineers who understand their tools more thoroughly and can use that knowledge to create better digital recordings.

VII Sources

The title of this paper is “The Audio Engineer’s Approach to Understanding Digital Filters For the idiot such as myself” because merely months ago I understood almost none of what I have just written. Finding decent texts written at the level of the audio engineer is a daunting task. I have done a lengthy amount of research, reading texts that were well above my head in order to glean a very basic understanding of how our digital tools actually operate. I wish to pass on my sincerest gratitude to a few people who have been extraordinarily giving of their energy, time, and knowledge to help me to understand enough to be able to write this paper. Special thanks are extended to:

Paul Frindle of Sony Corporation (Oxford, UK)

Glen Zelniker of Z Systems, Inc.

Jim Johnston formerly of AT&T Research

Russell Scott of Gentner Corp.

© Copyright 2002 Nika Aldrich of Cadenza Recording

All Rights Reserved. No part of this publication may be reproduced in any material form (including photocopying or storing in any medium by electronic means) without the written permission of the copyright holder.

Nika@CadenzaRecording.com

<http://www.cadenzarecording.com>