ELECTRONOTES 209

Newsletter of the Musical Engineering Group

1016 Hanshaw Road, Ithaca, New York 14850

Volume 22, Number 209 February 2012

GROUP ANNOUNCEMENTS

Contents of EN#209

Basic Elements of Digital Signal Processing
Finite Wordlength Effects (Part 2)

Part 1 of this series was published in Electronotes 204

The Contents of Part 1 Were:

3d. Dither – Increasing Resolution
3e. Oversampling and Noise Shaping
  3e-1 Introduction
  3e-2 Two Applications to Consider
  3e-3 Oversampling Noise Shaping Recording
  3e-4 Oversampling Noise Shaping Playback

----------END OF PART 1 LISTING ----------

This series has included and continues to include portions that are newly presented here and portions that are perhaps 10 years old. In consequence, there is some unevenness. We hope the overall presentation does cover the general topics in a comprehensible way.

CONTENTS OF PART 2 – THIS ISSUE

3f Overflow Oscillations (And Some Rare LTI Equivalents)

3f-1 SECOND-ORDER NETWORK AND THE STABILITY TRIANGLE

The second-order digital network (Fig. 19) is a fundamental element of Infinite Impulse Response (IIR) digital filters and can also serve as an oscillator. The network has a transfer function:

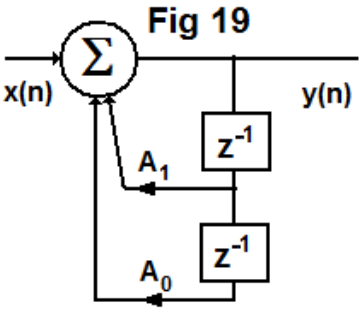
$$H(z) = 1/(1 - A_1 z^{-1} - A_0 z^{-2}) \quad (8)$$

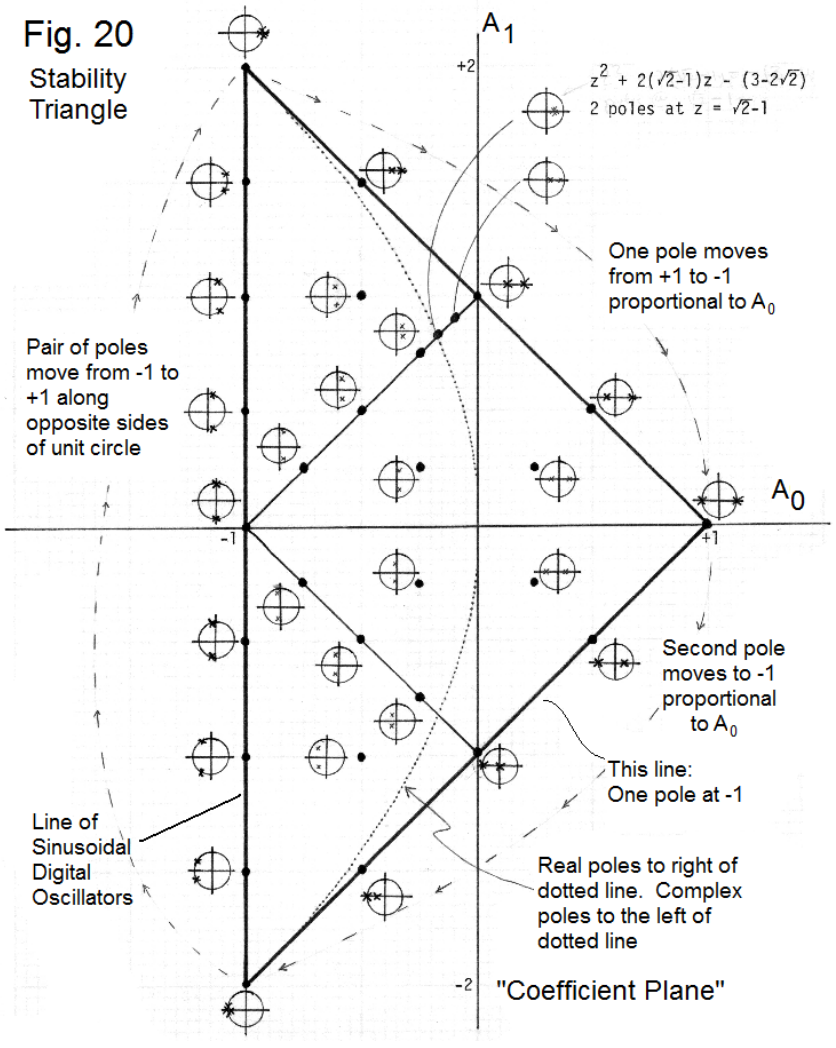
with two corresponding poles:

$$z_{p1} = A_1/2 + (1/2)(A_1^2 + 4A_0)^{1/2} \quad (9a)$$

$$z_{p2} = A_1/2 - (1/2)(A_1^2 + 4A_0)^{1/2} \quad (9b)$$

These poles could occur anywhere in the z-plane, but the useful cases are those where the poles are strictly inside the unit circle ($|z|<1$) which are useful for digital filters, and those where the poles are on the unit circle, $|z|=1$, which are useful as oscillators. It is always possible to choose values for A_1 and for A_0, and using equations (9a) and (9b), find the poles and check their magnitudes, relative to 1. More efficiently, we can solve the general problem by considering a "coefficient plane" of all possible values of A_1 and A_0, and see what region corresponds to the unit circle and its interior. The result, Fig. 20, is the so-called "stability triangle."





From equation (9b), we find that a pole occurs at z=-1 if (see Fig 20):

$$1 + A_1 = A_0 \qquad (10)$$

The case A_0=0 and A_1=-1 is an obvious solution to equation (10). The digital network corresponding to this case is shown in Fig. 21a. In this case, there is no second pole (or it is at z=0). The impulse response of this network is, by inspection:

$$h(n) = (-1)^n u(n) \qquad (11)$$

which is a “digital sinewave oscillation” at half the sampling frequency.

A second solution to equation (10) is A_0=1, A_1=0. The network for this case is seen in Fig. 21b. In this case, there is a second pole, and it is at z=+1 (again see Fig. 20). The pole at z=1 corresponds to dc (zero frequency). The impulse response of this network is again obtained by inspection and is:

$$h(n) = (1/2)[\, 1 + (-1)^n]u(n) \qquad (12)$$

which is the sequence ...1 0 1 0 1 0 1 0 ..... . In fact this situation represents an oscillator capable of supporting both a dc (constant) and an ac (half the sampling frequency) term, of which equation (12) is an example. In general, any sequence ...a b a b a b a b .... which is initialized on the network will be sustained as an oscillator.

Yet a third solution to equation (10) is A_0=-1 and A_1=-2. This solution (again see Fig. 20) corresponds to the case of a second pole at z=-1 . The digital network for this case is seen in Fig. 21c, and the impulse response is (by inspection):

$$h(n) = (-1)^n(n+1)u(n) \qquad (13)$$

This represents an unstable case. Because there are two poles in exactly the same place ("second-order" at z=-1), the network is not an oscillator. It can be seen as one of two limiting cases of complex conjugate poles on the unit circle (true sinusoidal digital oscillators - see Fig. 20 at A_0=-1 as A_1 approaches -2). Additional cases that are solutions of equation (10) such as A_0=2, A_1=1 (which has a second pole at z=2) are also unstable. In fact, Fig. 20 shows the stable cases of interest at the moment (a single pole at z=-1) as the bottom side of the stability triangle (excluding the point [-1 ,-2]).

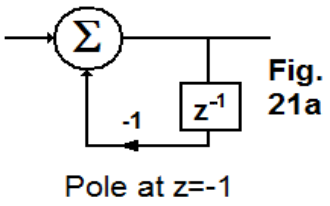


Fig. 21a

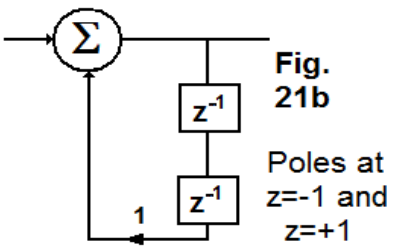


Fig. 21b

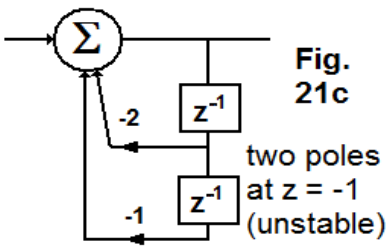


Fig. 21c

3f-2 DIGITAL SINEWAVE OSCILLATORS - SPECIAL CASES

As seen in Fig. 20, the case of A_0=-1 represents a class of networks with complex conjugate poles on the unit circle. These are digital oscillators producing a sinusoidal sequence of samples, and are extremely useful. These will be discussed a bit in Section 3g below. For the moment we can extend the discussion of Section 3f-1 above. Suppose that using equations (9a) and (9b) we place the poles on the unit circle at an angle of 60° (Fig. 22a) with respect to the real axis [at cos(60°) ± j sin(60°) = 0.5 ± j√3/2]. This gives us A_1=1 and A_0=-1. This network, shown in Fig. 22b, has an impulse response of (by inspection):

$$h(n) = [1\ 1\ 0\ -1\ -1\ 0\ 1\ 1\ 0\ -1\ -1\ 0\ \ldots..] \qquad n = 0,1,2,3,\ldots.$$

$$= (2/\sqrt{3})\cos(n\pi/3 - \pi/6)\,u(n) \tag{14}$$

which is indeed a sinusoidal sequence (Fig. 22c). Figures 23a, 23b, and 23c show a similar but slightly simpler case where A_0=-1 and A_1=0, where the poles are set at j and -j, which has impulse response (by inspection):

$$h(n)= [1\ \ 0\ -1\ 0\ 1\ \ 0\ -1\ \ 0....] \qquad n = 0,1,2,3\ldots\ldots$$

$$= \cos(n\pi/2)\,u(n) \tag{15}$$

(This network also supports the sequence …1 -1 -1 1 1 -1 -1….) At this point, all the true oscillators we have looked at (equations 11, 12, 13, 14), have involved samples with no more than two different magnitudes: one being zero and the other non-zero. This means that if we multiply these sequences by any gain g, we still have only two magnitudes, and a sequence may be preserved even under a non-linear processing.

For perspective, we may want to note that a large variety of much more general digital oscillators are possible [10], such as the example in Fig. 24. This situation has an arbitrary frequency and many arbitrary output amplitudes in the general case.

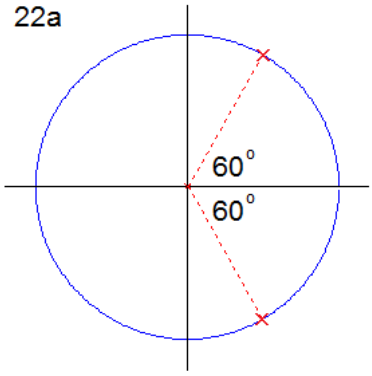
3f-3 ADDING A LOOP GAIN

A slight modification of Fig. 19, the addition of a gain g directly following the summer, results in the network of Fig. 25. This changes the transfer function and the poles as follows:

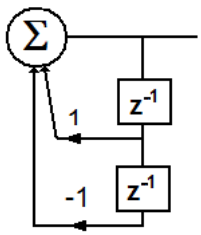
$$H(z) = g / (1 - gA_1z^{-1} - gA_0z^{-2}) \tag{16}$$

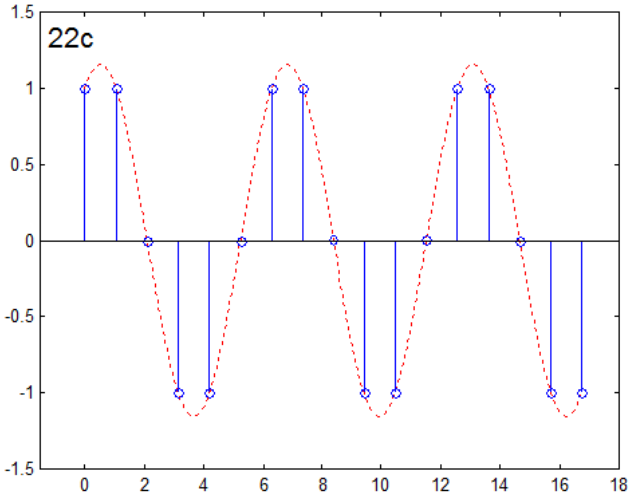
with two corresponding poles:

22a

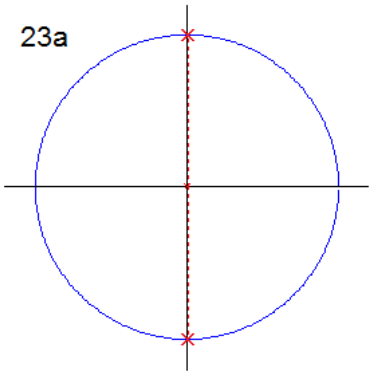


22b

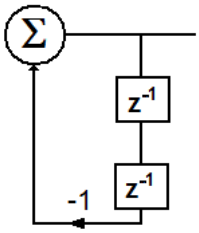


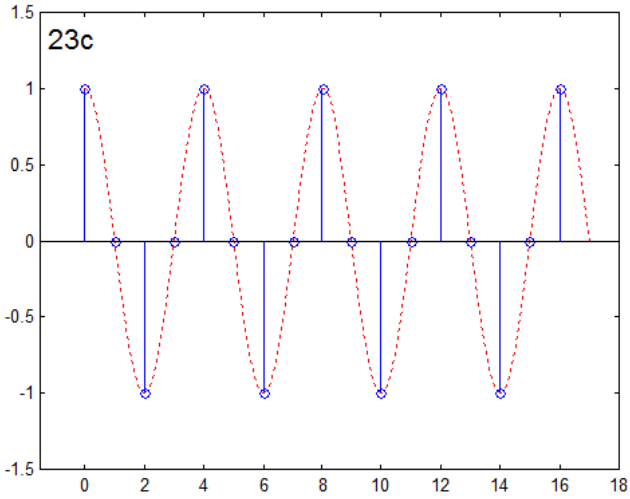


23a

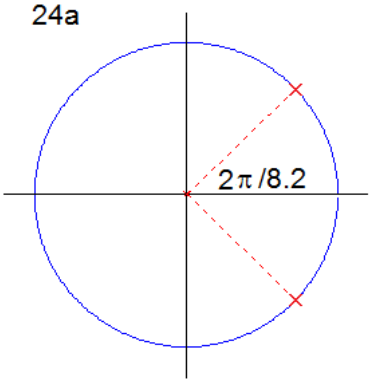


23b

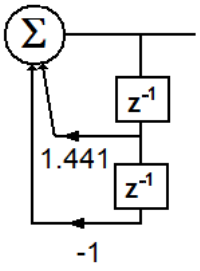




24a



24b



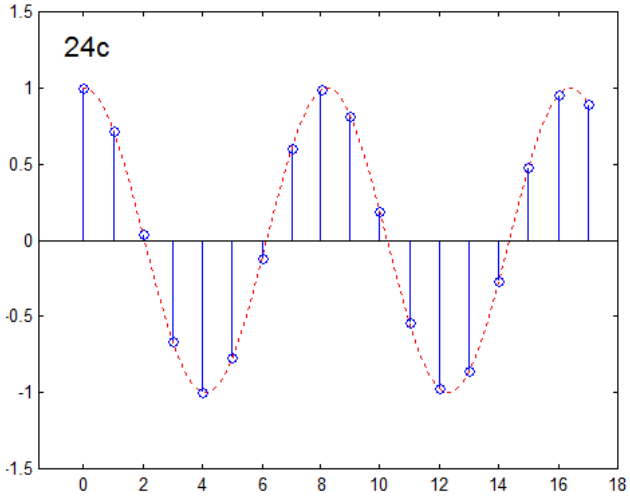
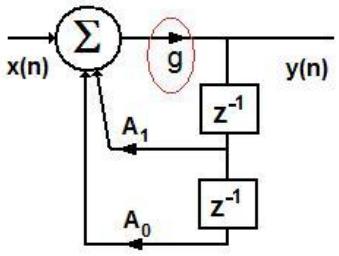
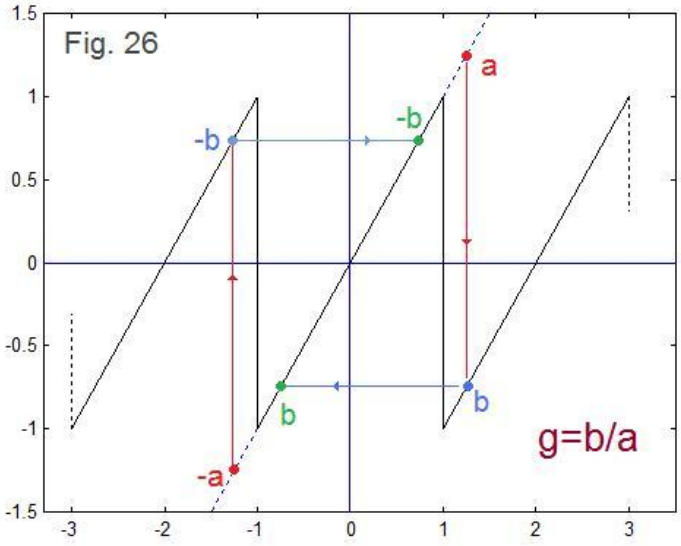


Fig. 25





$$z_{p1} = = gA_1/2 + (1/2)\,(g^2A_1^{\,2} + 4gA_0)^{1/2} \qquad (17a)$$

$$z_{p2} = = gA_1/2 - \ (1/2)\,(g^2A_1^{\,2} + 4gA_0)^{1/2} \qquad (17b)$$

In effect, all that has happened is that there is an overall gain of g added to the transfer function, and the feedback coefficients are multiplied by g. But all the rest of the analysis applies to this new case.

For example, if we want to place a pole at z=-1 with this modified network, we modify equation (10) as:

$$1 + gA_1 = gA_0 \qquad (18a)$$

or

$$g = 1/(A_0 - A_1) \qquad (18b)$$

This allows us to start with a stable network and calculate a value of g that will destabilize it in the sense of placing one pole at z=-1 with the resulting oscillation at half the sampling frequency. All this is rather pointless except as we want to look at a special way in which g may be realized, and here is where the finite wordlength aspects enter the picture. Specifically, we want to look at special cases where an overflow non-linearity is equivalent to a constant gain factor g, which is only possible when we guarantee that only certain numbers will appear in the network. The best way to describe what we are proposing is just to go forward with an example.

Fig. 26 shows a model for overflow arithmetic. Here we are assuming that all numbers are restricted to the range of -1 to +1, and that requests for numbers outside this range are

subject to the sawtooth overflow shown. In simpler terms, if we ask for a number outside this range, we either add 2 or subtract 2, as many times as necessary, to end up with a result that is within the range -1 to +1. Thus for example, as shown, if we ask for a number a which is outside the range we get a number b which is a-2, resulting in a gain:

$$g=b/a \tag{19a}$$

$$g = (a-2)/a \quad \text{or} \quad a = 2/(1-g) \tag{19b}$$

$$b = a-2 \tag{19c}$$

Likewise a number -a would become -b = -a+2 for a gain of (-a+2)/-a which is the same gain. With this arithmetic, numbers ±a are effectively multiplied by a constant value of g. Another special number, specifically the number 0 (which is not subject to overflow), can be multiplied by g, resulting of course in 0. This means that we can look at certain oscillation sequences as both overflow oscillations and as LTI equivalents.

3f-4 OSCILLATION AT z=-1

Equations (18b), (19a), (19b),and (19c) now offer us the opportunity to define an overflow oscillation that can be understood by its LTI equivalent. Suppose we choose a stable filter A_0=-0.5 and A_1=0.75 (see Fig. 20). From equation (18b) we obtain a destabilizing gain of g = -0.8. From this, equation (19b) leads to a = 10/9 (1.11111....) which is greater than 1. Following overflow, this a becomes b = a -2 = -8/9 (-0.888888...) which realizes the desired gain of -0.8. Fig. 27 shows the result, and it is easy to see that the network sustains the oscillation between +8/9 and -8/9. (The second pole in this case is at 0.4.)

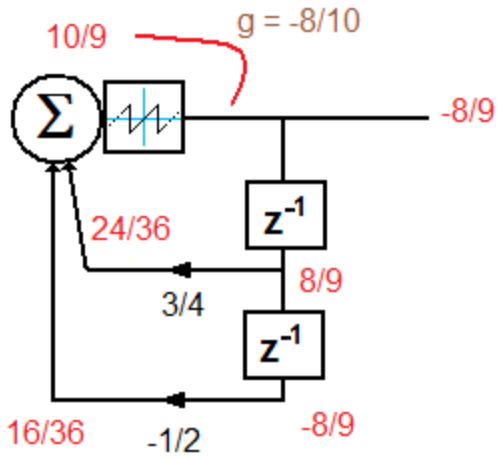


Fig. 27

= Fig. 26

Since we have determined that overflow results in an effective gain of -0.8 (a desired number of 10/9 changing to -8/9) we can continue to use this (for convenience - a similar case could serve as well) in the following examples. Fig. 28 shows two cases where poles are placed at ±j. This we achieve using A_0=10/8 (and A_1=0) so that when combined with the g=-0.8, the overall gain around the loop is (-1). This corresponds to Fig. 23b, and here works only because we initialize with magnitudes of 8/9 and zero. Interestingly, it also works when initialized with magnitude 8/9 at both delays. This represents an oscillation that is at the same frequency, but 45° shifted from the first case (and has a different amplitude).

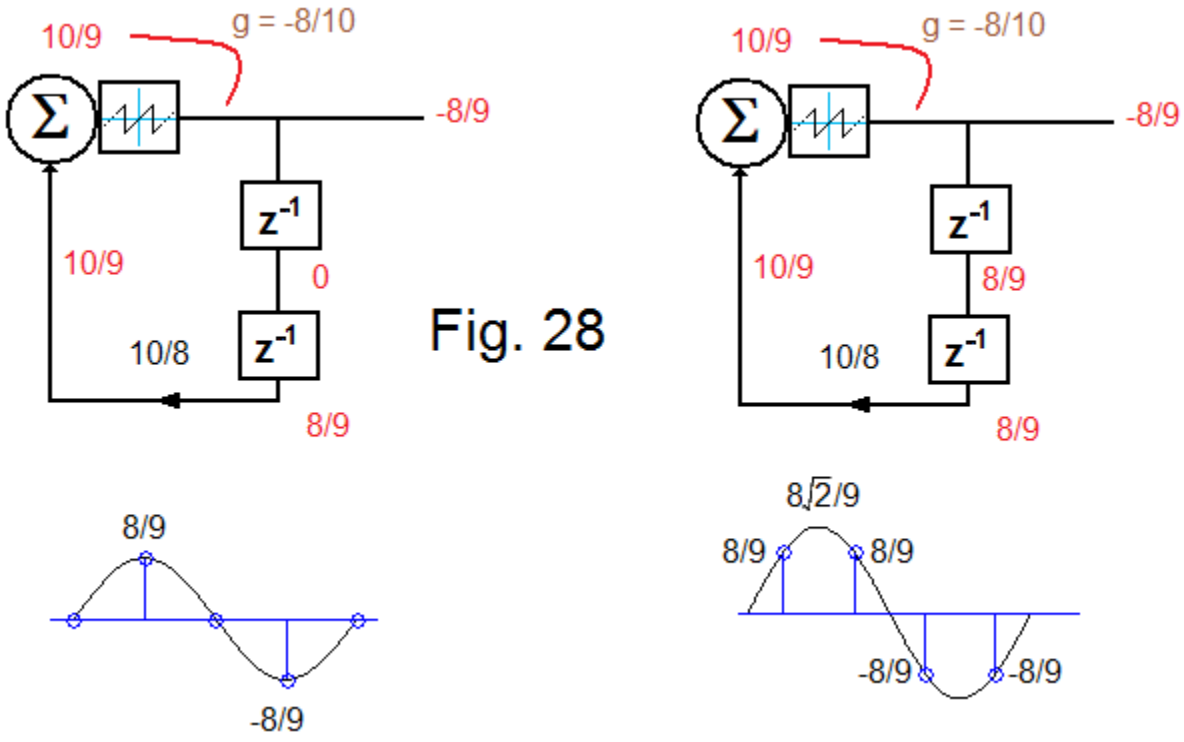


Fig. 28

Fig. 29 extends the above examples to a case where poles are at 60° as in Fig. 22a. Here the feedback coefficients are set to A_0 = 10/8 and A_1 = -10/8. Note that with the "gain" of 8/10 due to the overflow arithmetic, the coefficients are effectively A_0=-1 and A_1=+1 as in Fig. 22a. This of course assumes there are no numbers in the network other than those with magnitude 8/9 (i.e., 8/9 or -8/9) or zero. Fig. 29 shows the networks for all six steps of the six-step oscillation cycle.

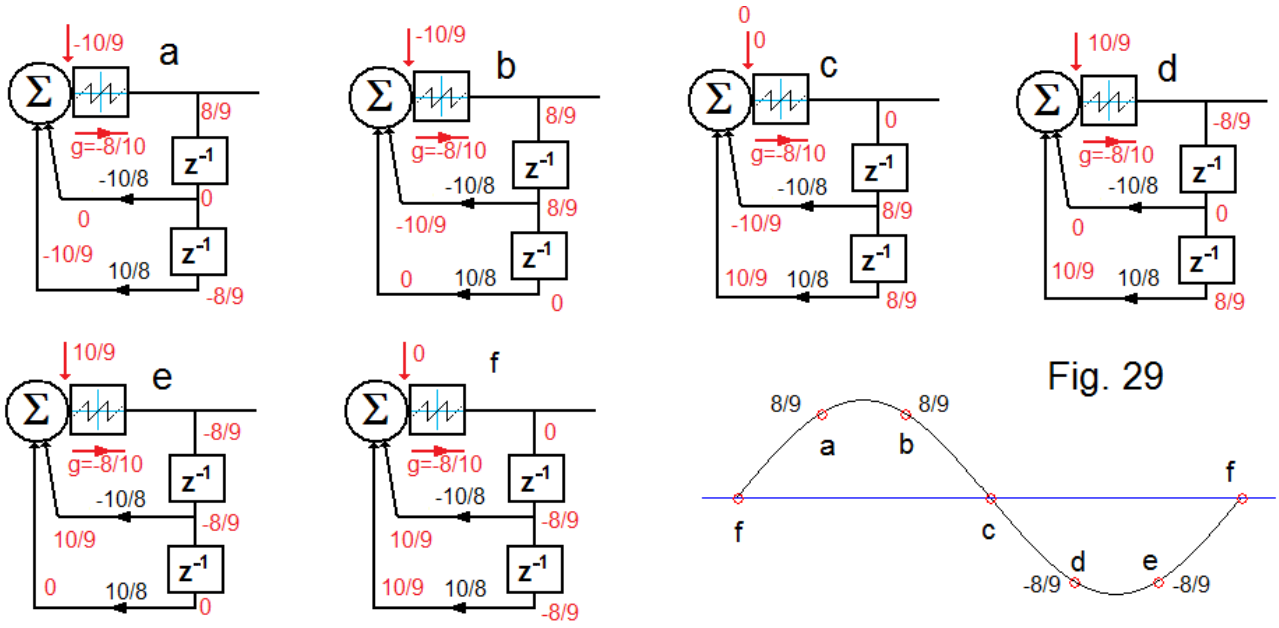
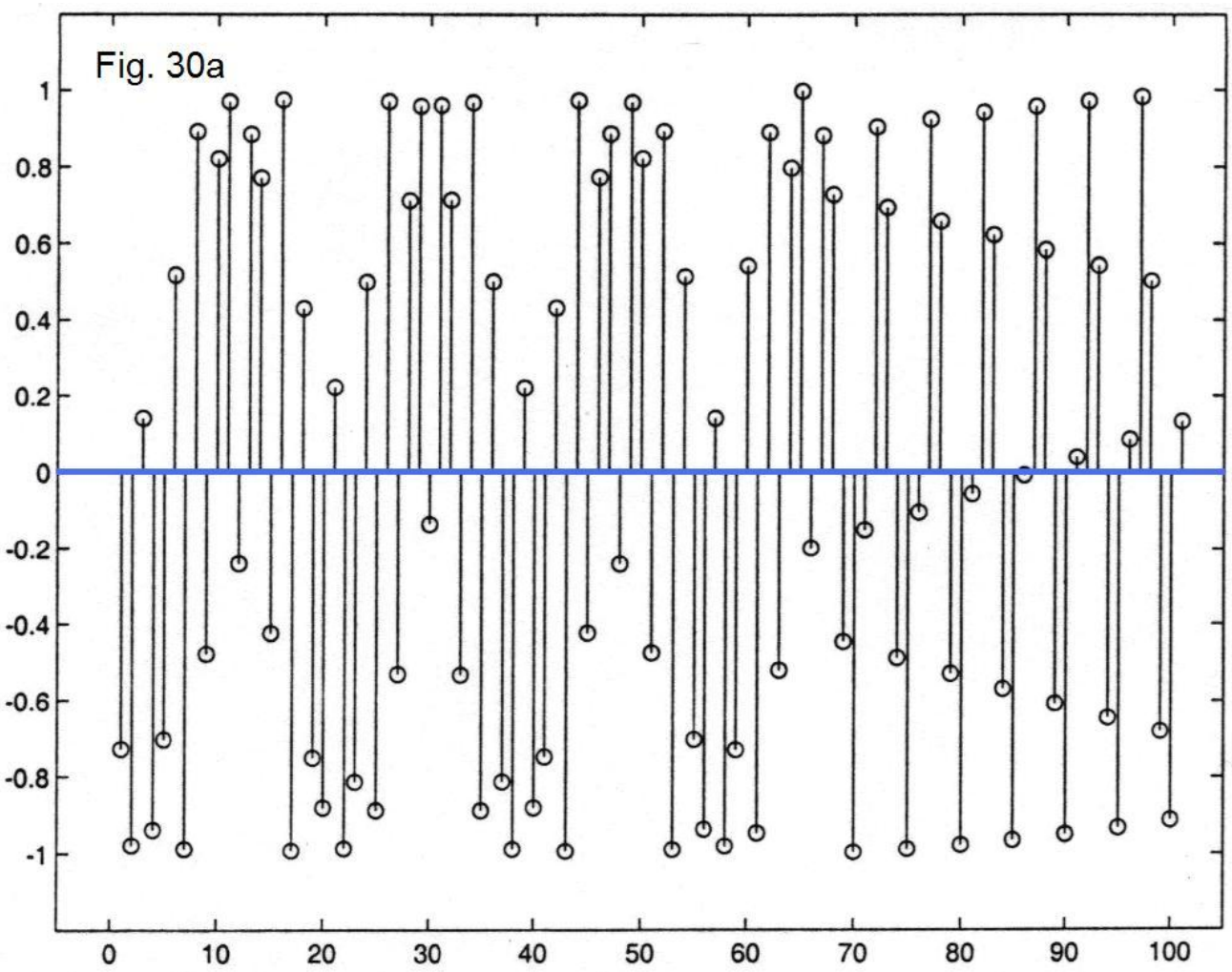


Fig. 29

3f-5 CHAOS AND FRACTALS

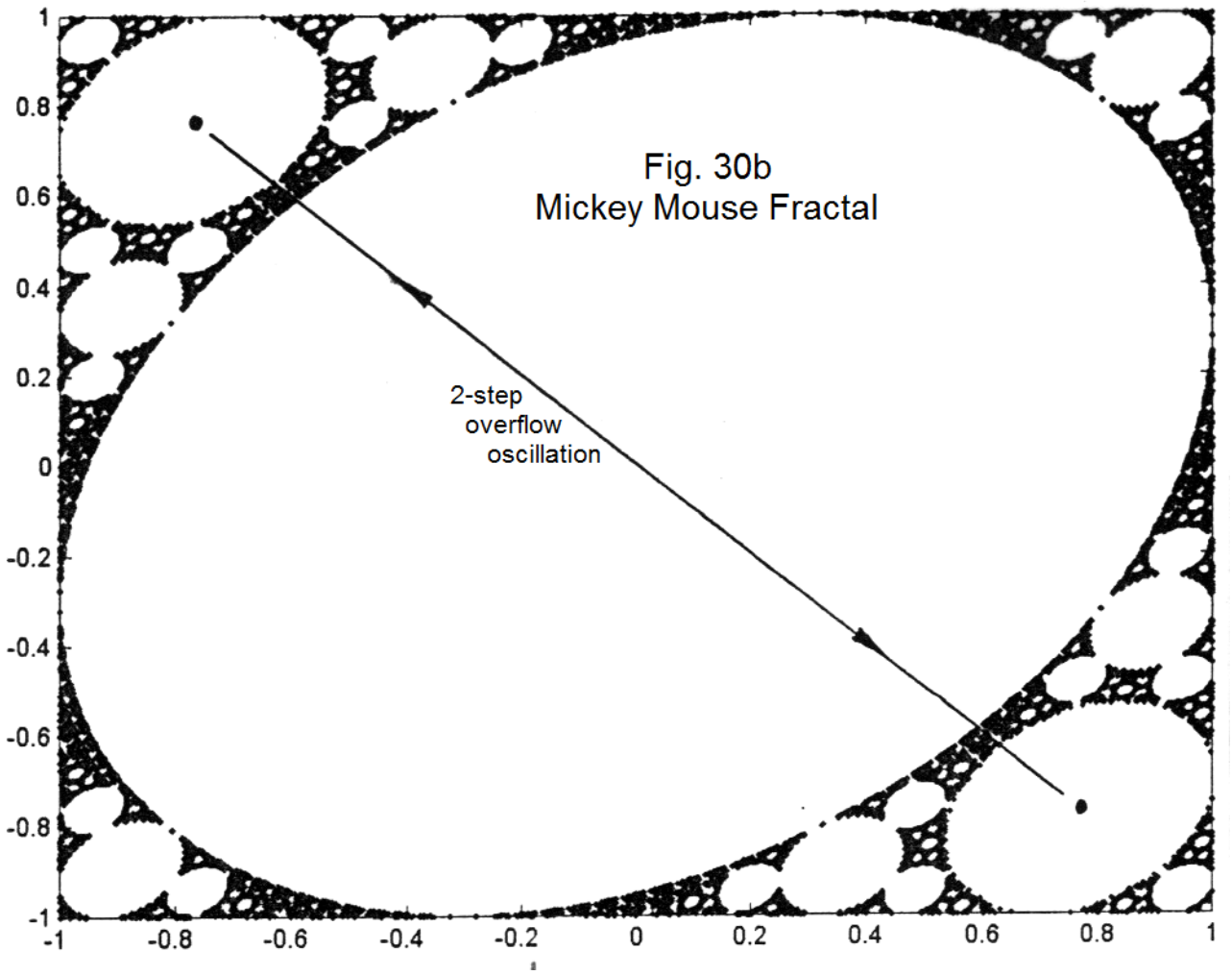
So far, we note that everything we have done can be understood in terms of rather ordinary computation. As shown in Fig. 29, we see what happens by computing successive states of the network. We did manage to use some LTI theory, and this helped us understand what we observed, but more importantly, it was fairly essential for finding the parameters of the special cases in the first place. In the general case, things get more bizarre.

Fig. 30a shows particular displays relating to overflow oscillations which illustrate the more general case. In Fig. 30a, we have a portion of a time sequence for an overflow oscillation ($A_0 = -1$, $A_1=0.6$, with the arithmetic of Fig. 26, initialized to have the outputs of both delays to start at +1). It is apparent from the time sequence that the oscillation is not simple in either being short in length, or in being simple in structure (such as a sinusoidal sequence). In fact, we see at one point a clear indication that the general aspects of the sequence change suddenly. The nature of the overflow oscillation, for this case, is that of deterministic chaos. In basic terms, "chaotic" suggests that there is no procedure for finding the values of the sequence except to run (or simulate) the "machinery" that is making the



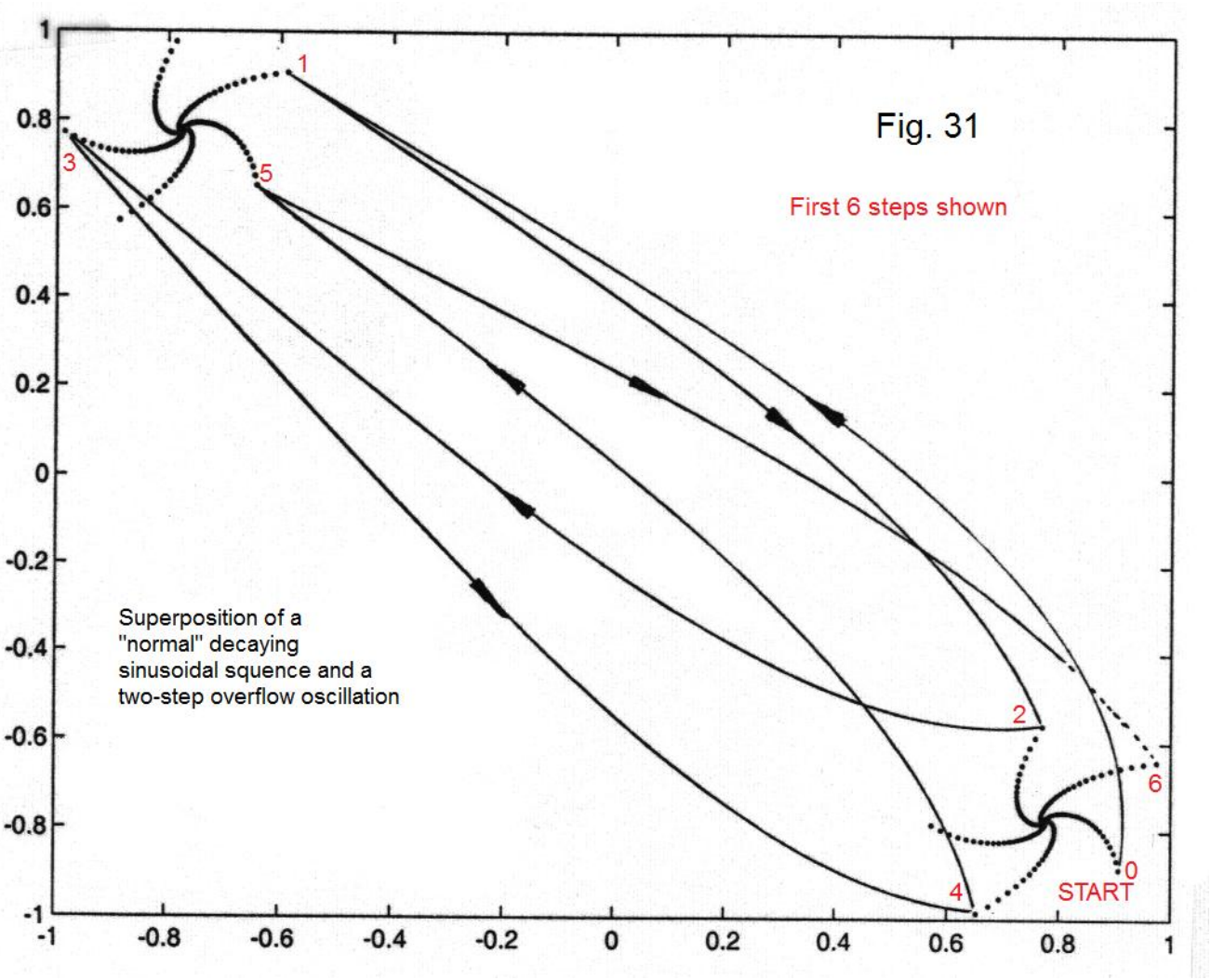
sequence. Contrast this with a sinusoidal digital oscillator (a very similar "machine"), for example, where a simple formula such as cos(2πfn) immediately gives us the current sample, or one that will occur a million steps later, with equal ease. In the chaotic case, it would be necessary to compute each step sequentially all the way to a million.

The picture of Fig. 30b, on the other hand, suggest that there is some hidden order in this same chaotic oscillation. What we have plotted here is not a time or frequency description, but rather, we plot all time samples against the previous sample. Thus each sample of Fig. 30a (and thousands more not plotted there), paired with the sample that precedes it, is a single point of Fig. 30b. This device of plotting consecutive states, essentially the same as a Lissajous figure seen on oscilloscopes, is called the "phase plane" and is an established tool for looking at dynamical systems. We note immediately, that when we consider a large

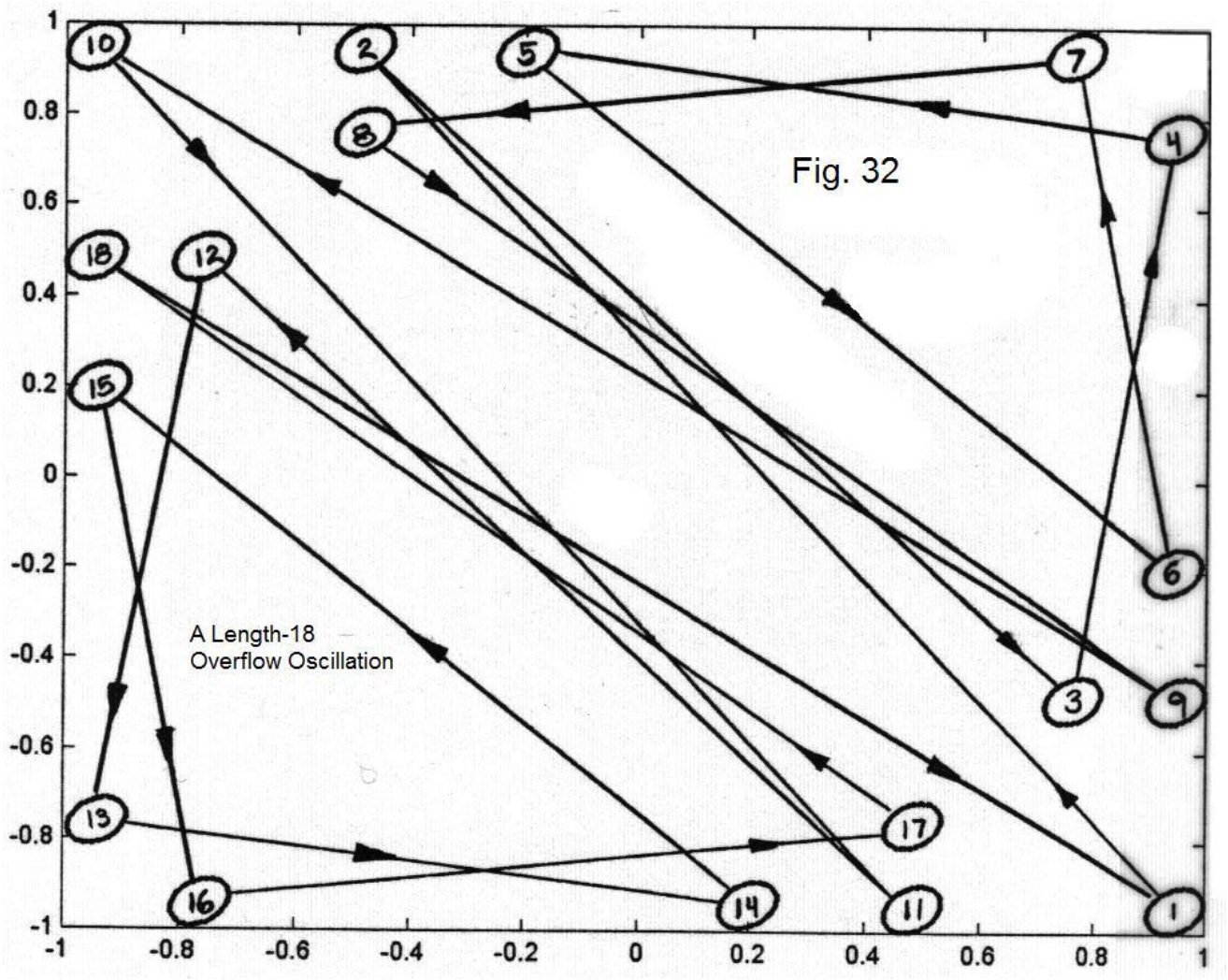


enough number of states, there is structure. The points only occur in certain places (or perhaps, they are forbidden in certain regions). Further, this particular phase plane looks and is fractal. At this point, while the “machine” here is very similar to our networks investigated above, we likely feel very far removed from our simple, contrived overflow oscillations.

Accordingly, we hasten to point out that the fractal contains at least two "old friends." First, many persons see the Disney cartoon character "Mickey Mouse" inside this particular fractal. The second old friend is the two-step overflow oscillation. In a manner the same as we used to calculated an initial state of -8/9 and 8/9 that led to the overflow oscillation of Fig. 27, the initial state -10/13=-0.7692 and +10/13=0.7692 leads to a two step overflow oscillation here. [That is, sith A_0=-1 and A_1=0.6, we calculate g = -5/8. This gives a = 16/13, and b=a-2 is -10/13.] Looking for this point on Fig. 30b shows that it is in the forbidden region. Indeed, this oscillation is not chaotic, and we would not expect it to be in the dark part of the fractal.



By no means does this brief description explain more than a tiny portion of the overflow phenomenon. Two additional examples will add a slight bit more. Fig. 31 shows $A_0 = -0.98$ and $A_1=0.6$, started at the point 0.91, -0.91. The A_0 coefficient of -0.98 rather than -1 makes the structure a “resonator” instead of an oscillator (the energy decays away). What we see is two decaying sinusoidal oscillations, spiraling in to two centers that correspond to the two-step overflow oscillation. This is pretty much what we have in mind when we think of actual overflow oscillations in actual digital filters. The region occupied by this decay "flower" is essentially the corresponding oval of the fractal of Fig. 30b. Accordingly, our humble two-step overflow oscillation does seem to determine this larger region, not just a single point. In fact, all the smaller ovals visible in the fractal do correspond to observable, periodic overflow oscillations. These we can study by setting an initial state inside the ovals to see what develops. What does not develop is a simple oscillation with an LTI equivalent. By setting $A_0=-1$, $A_1=0.6$, and initializing at 0.918 and -0.918 (just beyond the two-step limit), we get the pattern of Fig. 32. This corresponds to an 18-step overflow oscillation which drives the output violently from side to side (in the sequence indicated) with a superimposed much smaller sinusoidal oscillation driving result around the ovals. At this point, we must rest the topic. This has been discussed in detail in this newsletter before [7]. In the above, the general ideas have been shown.



3g Coefficient Roundoff – Digital Oscillator Frequency

All the material in Part 1, and in Part 2 above, was produced in full or in part prior to this 2012 addendum. Some of the material here in Part 2 was only at the level of text and hand drawings. The index for both parts has been around for many years (like 10 years!), and much of the material here, in various states of completion, was used for teaching. So why am I telling you this? Well, because a lot changes in 10 years.

With regard to this Section 3g, and Section 3j which follows, two things are true. First, they were more important 10 years ago, based on hardware considerations. Secondly, both

were quite familiar in their basic ideas from what was then fairly recent prior work in analog signal processing. If this were being presented for a "digital" audience, we might omit some of these topics. But the readers of this newsletter are still very much "analog" people, so these topics are much simpler for us to master.

Accordingly I can state that the "Coefficient Roundoff" issue here is the digital counterpart of the analog "sensitivity" problem. With an analog filter, we designed a filter and arrived at values for resistors and capacitors. We knew several things would happen if we decided to build the actual filter. First, we had to find parts, and there were not likely to be found in the values calculated, even to just two decimal places. Even if we did find a nominally "exact" value, you knew that any particular component had a "tolerance" associated with it, which means that the part might have any value within say 10%, 5%, or 1% - something like that Did it matter? Sometimes yes – sometimes – not much.

A standard topic in electrical engineering was "sensitivity analysis" [8]. This means that we could calculate, how sensitive a particular performance parameter (like cutoff frequency) was to a possible tolerance of a particular component. A full 5% tolerance error of a component might mean a 5% change of a performance parameter. Or it might be 2.5%. It could even be much larger than 5%. It could even have been 0%. [I recall a student 's lab report saying that a roll-off rate was 5.8 db/octave, not 6 db/octave, but that was within tolerance. In fact, the sensitivity was 0 – he had just not measured far enough down.] The point is, you had to do this analysis or take your chances. We did know that no two realizations would be exactly the same.

In the design of digital filters, we also arrive at numbers; not of components, but of coefficient multipliers. Instead of ordering parts, we typed in or programmed numbers – obviously to a finite wordlength. We did not get the number we wanted, particularly not with 8 bits that we often started with. This might not be good enough. But we did not generally even think of doing a sensitivity analysis as we would have with analog. There were no "statistics" involved. We knew we were getting the wrong numbers, but we knew exactly what wrong number we were getting, and every realization would be identically wrong. You just calculated the performance curves. It was good enough, or it was not. If not, you could at (rare) times manipulated a coefficient, but often the answer was a requirement for a larger wordlength (more bits). The solution was often, as was the case with analog, to break a overall filter into smaller pieces – like a 10^{th} order filter to 5 2^{nd}-order sections.

Fig. 33 shows the general idea. Here we have designed a digital 8^{th}-Order low-pass Butterworth filter using Bilinear-z Transform method [9]. The exact details are not important, but we assume we have first tried to realize this in "Direct form", and keeping full precision on the denominator coefficients, we get the black curve. Next we have rounded the denominator coefficients to about the nearest 0.1%, and we obtain the red curve. Rounding to about 1% gives the blue curve. Most likely, even the red curve is not good enough, and certainly the blue curve is highly unsatisfactory. In fact, this would immediately suggest we should try a cascade rather than a direct form, but the details don't matter. What is

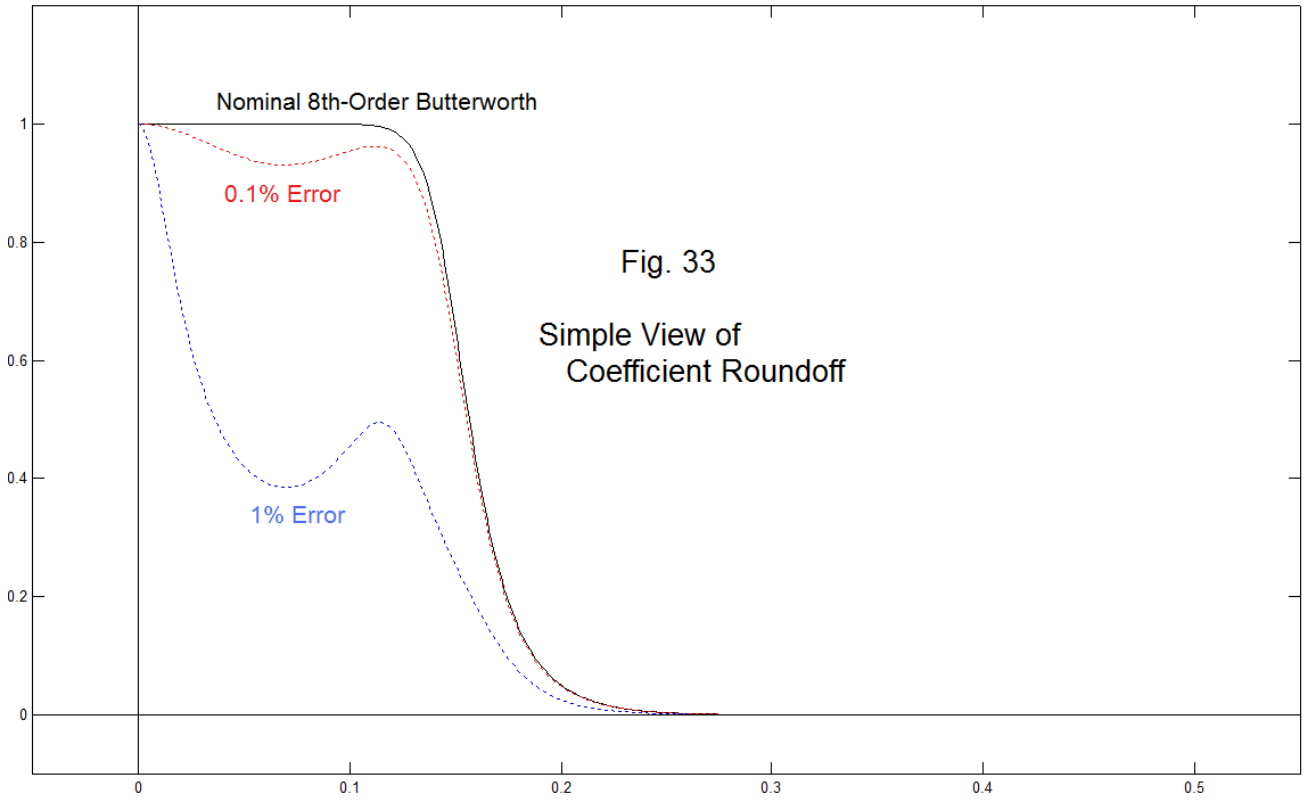


Fig. 33

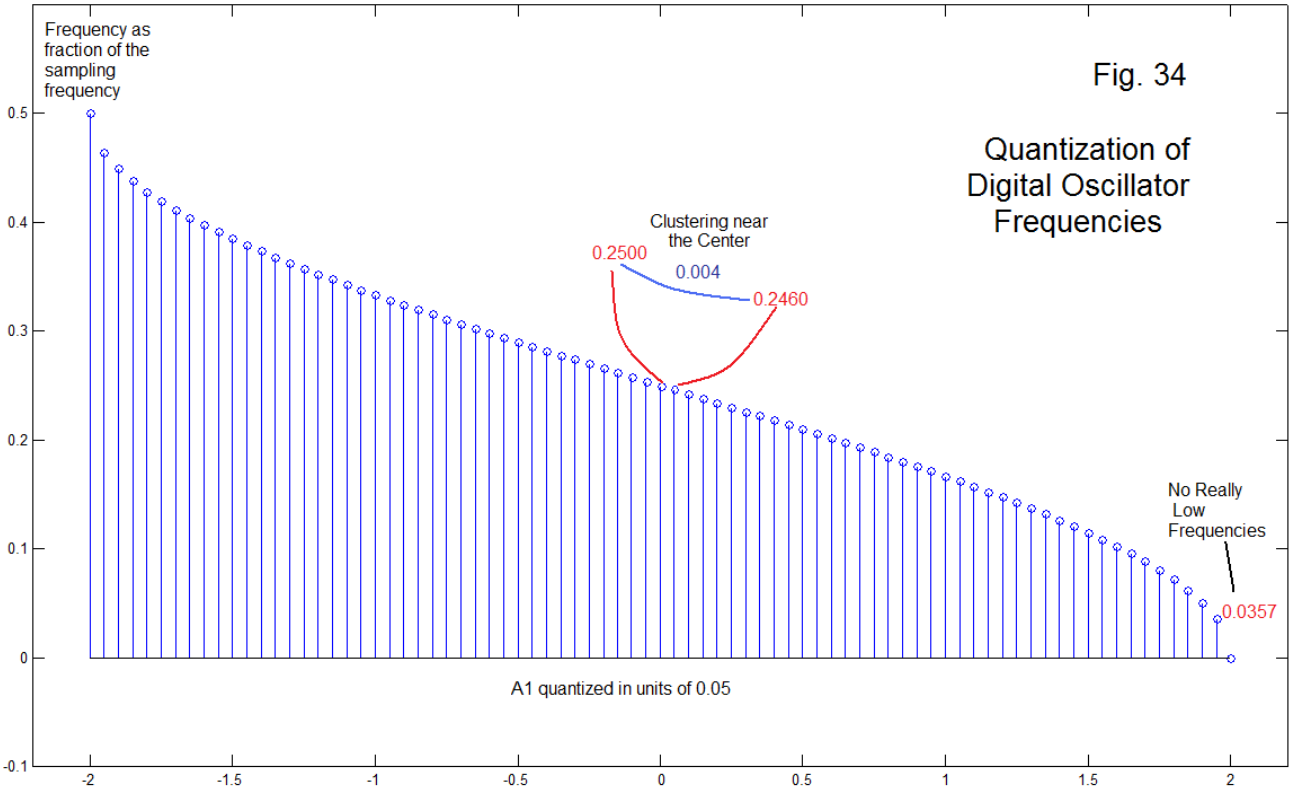
Simple View of
Coefficient Roundoff

important is that this sort of analysis gives us exactly what we would get, each and every time we implemented this filter with these coefficients. It is often less important to achieve every aspect of what we specified. If we said we want a cutoff at 1000 Hz, is it that exact cutoff, or is it a relatively flat passband, that is our real desire? This says "here is your filter" – take it or improve it. Less math is needed. It is the computing power that guides our designs.

A second example relates to our work with digital oscillators. As we saw, for example Fig. 24a, we can easily make digital oscillators. The frequency of the digital sinewave oscillator is exactly the angle of the poles on the unit circle, divided by 2π, and multiplied by the sampling rate. This is set by choosing the A_1 coefficient. The A_1 coefficient is 2cos(θ) where θ is the pole angle, and from Fig. 20 we see that A_1 should be between -2 and +2. Thus [10]:

$$f = f_s \cos^{-1}(A_1/2) \qquad (20)$$

Now consider the quantization of the A_1 coefficient, as this tells us what frequencies we have available. As an example where quantization is likely much more severe than is typical (used for illustration here) we show Fig. 34. Here we assume A_1 is quantized in units of 0.05, with equation (20) giving the possible frequencies. The result (inverse cosine) is of course not linear. In fact, the frequencies on both ends, as A_1 approaches +2 (low



frequencies) or -2 (half the sampling frequency) are widely spread. In fact, note that the lowest possible frequency (other than 0) is 3.57% of the sampling frequency.

Such low frequencies may well be needed in some applications. For example, we may be shifting voice frequencies in PA system to prevent positive feedback squeal. In such a case, frequencies of only a few Hz are needed. We have suggested [11] that one approach in such a situation is the use of two close frequencies around ¼ the sampling frequency where the clustering is tighter, and obtain a difference frequency by multiplying. In such a case, the difference frequency can be very low, close to zero, while the sum frequency, also in the multiplier output, is close to half the sampling frequency. As a result, a modest low-pass filter can separate them.

It might seem that if we have floating point, there is really no problem in finding an A_1 coefficient sufficiently close to +2. This is not true. Two possible problems appear. First, floating point does not, as we have noted, correspond to high precision everywhere (like close to 2) but only the ability to approach 0. A second problem is that anything multiplied by a very small A_1 coefficient, and added, may well not be enough to change a much larger number to which it is summed. This is the essence of the scale-change intermodulation distortion in Section 3i.

The problems suggested here are typical of remaining coefficient roundoff issues.

3h. Noise Gain

The term "noise gain" is used in a number of different areas of electrical engineering. In digital filter theory, it refers to the factor by which a quantization (or roundoff) noise power is amplified, from the point of generation of the noise (at the input, or internally) to the output of the filter. It is usually recognized as the sum of the squares of the appropriate impulse response sequence. It is convenient to obtain this result through the use of an example.

Suppose we have a five-tap FIR filter with taps 1, 2, 3, 2, 1 (Fig. 35a). This filter's triangular tap pattern is well studied, and since it is the convolution of a three-tap FIR filter (taps 1, 1, 1) with itself, we easily obtain the pole-zero plot (Fig. 35b) and the magnitude of the frequency response (Fig. 36a). We suppose that the samples applied to the input are quantized, so there is some noise power σ_Q^2 applied as shown. We assume that this quantization noise has a flat (white) spectrum. What does the spectrum at the output look like?

Fig. 36b shows what might be an actual example of a white spectrum. Here a signal is generated using a pseudo-random sequence of length 200. Fig. 36b shows the first 100 points of the absolute value of the FFT of this sequence, scaled for an "amplitude" of approximately 1. If we then filter the sequence with our impulse response 1, 2, 3, 2, 1 and plot the absolute value of the first 100 points of the FFT of the output, we get Fig. 36c. Essentially this is the spectrum of Fig. 36b shaped by the frequency response (Fig. 36a) which is also plotted in Fig. 36d. We see reasonable agreement, allowing for the "fuzzy" variations of the particular random test signal used for the plot. (Note that if we repeat this run with a different random start, we would see a different plot, but with similar general features. In fact, if we were to average a sufficient number of output spectra for white inputs, we would expect a fairly good approximation to the frequency response.)

So we know what the filter does to the noise. It is shaped and in general, amplified. If we were to "eyeball" the average gain over frequencies, we might choose something like 4 or 5. We can get the power by squaring the spectrum and integrating over frequency. It is most sensible to integrate the frequency response of the filter, rather than use any one of (or average of) the actual output spectra. The filter's frequency response is:

$$H(e^{j\omega}) = \sum_{n=-\infty}^{\infty} h(n)\, e^{-j\omega n} = e^{2j\omega} + 2e^{j\omega} + 3 + 2e^{-j\omega} + e^{-2j\omega} = 3 + 4\cos(\omega) + 2\cos(2\omega) \qquad (21)$$

The power in the output spectrum, for a flat input of amplitude 1 is given as:

$$P = (1/2\pi) \int_{-\pi}^{\pi} |H(e^{j\omega})|^2 \, d\omega = 19 \qquad (22)$$

the integral being done analytically and/or numerically.

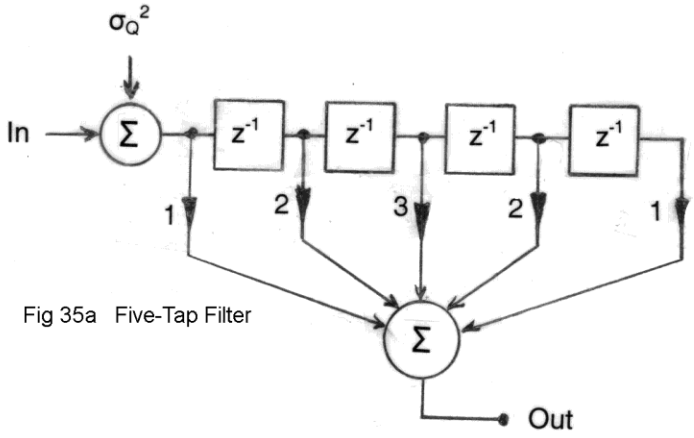
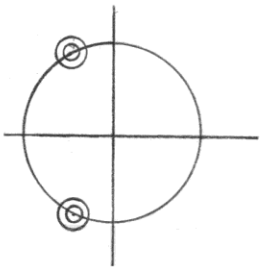
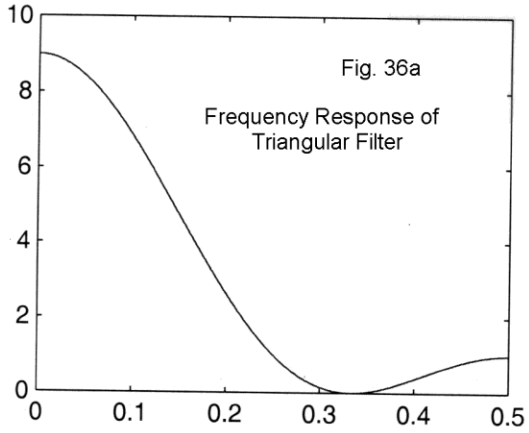
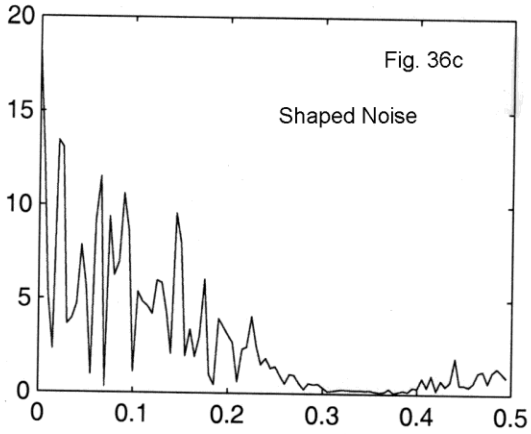


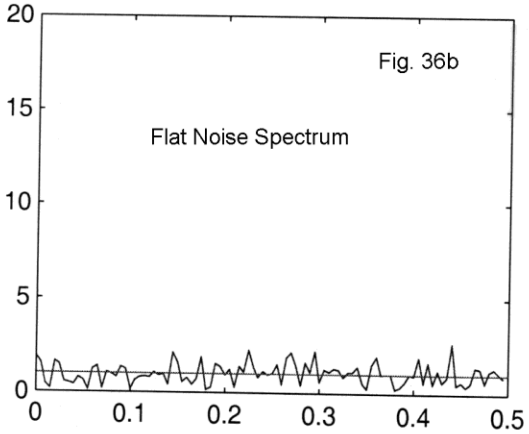
Fig 35a Five-Tap Filter

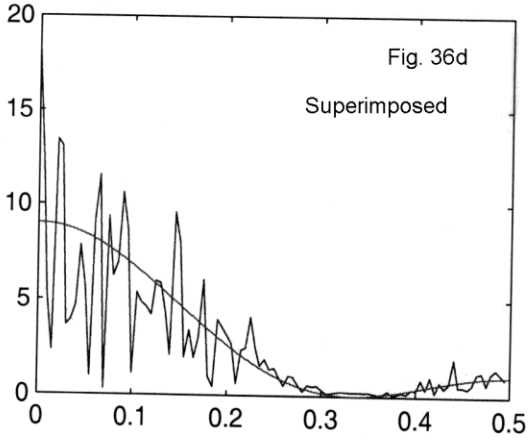
Fig. 35b
Zeros of
Triangular-Tap
Filter











But, by Parseval's theorem for the Discrete Time Fourier Transform, we also have P as:

$$P = \sum_{n=-\infty}^{\infty} h(n)^2 = 1 + 4 + 9 + 4 + 1 = 19 \qquad (23)$$

So the amplitude would be something like $\sqrt{19} = 4.36$.

Clearly in this case, the simpler way to get the correct answer was to sum the squares of the impulse response. Of course, it is usually trivial to simulate even an IIR filter using a computer, and as such, many values of the impulse response can be calculated, squared, and added up in a split second. In some other cases, it might be simpler to integrate the frequency response.

A good example of what can happen with an IIR filter is illustrated by the simple first-order feedback loop of Fig. 37. In this case, we know that the filter is stable if $|a|<1$, and the impulse response is $h(n)=a^n$ for n=0, 1, 2,... . Although this goes on forever, we can nonetheless sum a finite series to see if it will converge. For example, if a = 1/2, we could add 1 + 1/4 + 1/16 +1/64 + 1/256 + . . . which converges rapidly on 4/3. We can also simply sum the infinite series:

$$P = 1 + a^2 + a^4 + a^6 + ... \qquad (24a)$$

$$(1-a^2)P = 1 + a^2 + a^4 + a^6 + ... \\ -a^2 - a^4 - a^6 - .... \qquad (24b)$$

or:

$$P = 1/(1-a^2) \qquad (24c)$$

Note that as we might well expect, as |a| approaches 1, we get a very large noise gain. One way to understand this is that as |a| gets close to 1, numbers flowing around the loop do so with significant value for many cycles, and each time around offers the opportunity for the error to again influence the output. In fact, this leads us to the next point.

So far we have been calculating the sum of $h(n)^2$ for an entire digital filter in the usual sense of input/output. This we need when we are interested in how the noise in an input sequence, the result of quantization, is amplified at the output. We have also mentioned that additional quantization noise can be generated inside the filter, and that noise gain figures need to be calculated from the point of origin to the output. Fig. 38 shows a simple multi-stage filter. Here we have our usual quantized sequence as input, represented by the

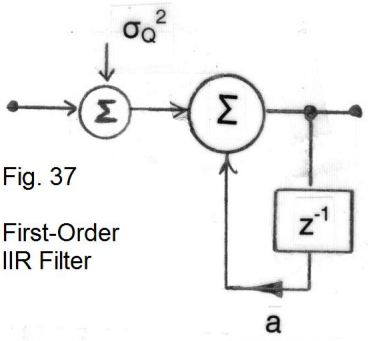


Fig. 37

First-Order IIR Filter

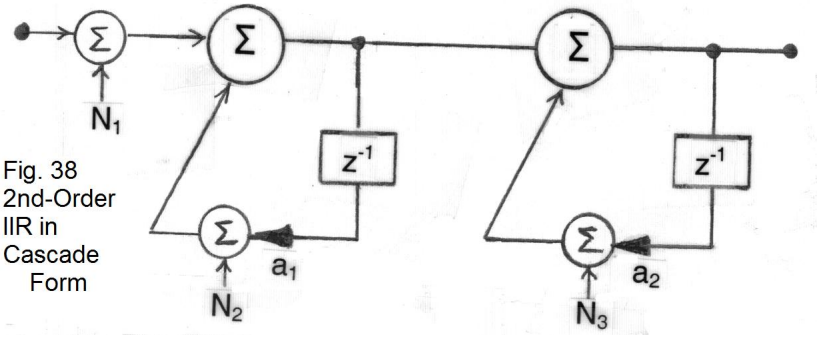


Fig. 38 2nd-Order IIR in Cascade Form

additive noise σ_Q^2. But we are also showing here two multipliers, a_1 and a_2. Each of these multipliers can be thought of as taking an N-bit number, multiplying it by another N-bit number, giving a 2N-bit product which is then truncated in some manner back to just N-bits. This is another form of quantization, and can be treated much as we did the original quantization.

Two points need to be made before we go on. First, in Fig. 37, we really should have considered the quantization following the multiplication by a, just as we are now doing here. Secondly, in some cases, these round-off following multiply are not too important. This is because the quantization may be far less severe following a multiply. For example, the original input sequence might have been the result of conversion by a 12-bit A/D. In a 16-bit processor, these would become the 12 most significant bits of the 16. Any roundoffs following 16-bit multiplies would be at the least significant bit level, which would be 16 times smaller, and $Q^2/12$ would be 256 times smaller.

So perhaps the main point to be made here is to see how we calculate the h(n) from points other than the input. Not surprisingly, this is a matter of thinking of the point at which quantization (multiplier roundoff or re-quantization) occurs as the input. For the example of Fig. 38, we see that the N_1 source (A/D quantization) and the N_2 source (roundoff following the a_1 multiply) are subject to h(n) of the filter itself. The N_3 source, the roundoff following the a_2 multiply, is subject only to the h(n) of the second stage. Thus we need to look at each source and find the appropriate $h_k(n)$. We still add up the powers. This would result in an output noise power, in general:

$$P = \sum_{\substack{\text{all} \\ \text{sources} \\ k}} \sigma_{Q(k)}^2 \sum_{n=-\infty}^{\infty} h_k(n)^2 \qquad (25)$$

It is easy to see that $h_3(n)$ is just a_2^n and we can sum the squares just as we did for the first-order example. The $h_1(n)=h_2(n)$ is more difficult. Remember that we want to get all the way

to the output, not just the output of the first stage. We can write down the $h_1(n)=h_2(n)$ impulse response without much trouble. [Note that it is the convolution of $a_1^n u(n)$ with $a_2^n u(n)$.] But then we would have to square this, and evaluate the infinite sum, so the ideas of solving the problem in the frequency domain, or of just calculating the sum numerically for particular values of a_1 and a_2 is all the more attractive.

It is always provocative to ask why we bother to do the things we suggest, and in this case, it might give us some ideas about noise levels. Perhaps more importantly, it allows us to compare different structures, all of which realize a particular filter, to see if any of them has a clear edge in output noise power. With all the assumptions going on, we might be hesitant to trust a calculation showing a 10% advantage, but if we saw our calculations giving a 2:1 or a 10:1 advantage (lower noise) for a particular structure, we would likely chalk up a mark in the favorable column for the better performer. One case where this happens is in IIR filters where we consider the choice between a direct form realization (all the delays in one line) as opposed to a cascade of second-order sections. We expect the cascade form to be far better. But this is a general fact, not something we need to calculate very many times. However, within cascade form, there are still choices to be made - in particular, the order of the different structures.

For example, does the order of a_1 and a_2 make a difference in Fig. 38. Not much if a_1 and a_2 are close together, or if the original quantization is much more severe than the multiplier roundoff. But suppose that a_1 and a_2 need to have the values 0.5 and 0.95 in some order, and that all quantizations are the same. Simulation shows that regardless of order, the contribution from $h_1(n)=h_2(n)$ gives the same power of 38.42. (It of course makes sense that the order does not matter for the impulse response of the entire filter.) But if a_2 is 0.5, then its power is only 4/3, while if it is 0.95, the power is 10.26. Adding up over the three sources we compare (38.42 + 38.42 + 1.33) = 78.17 with a_2=0.5 as compared to (38.42 + 38.42 + 10.26) = 87.10 with a_2=0.95. Perhaps not enough to worry about in this case, but all other things being equal, we would have a basis for a decision.

Extension of these ideas to higher order sections and to other structures are likely straightforward.

3i. Scale Change Intermodulation Distortion

This issue involves floating point arithmetic. We are interested in the case where there are two or more different components in a signal where the amplitudes of the different components vary greatly. In general, due to the very large dynamic range of the ear, we need to pay attention to the weaker as well as the stronger components. So while we don't want to suppose that a strong component completely overpowers a weak one at the ear, it might in the arithmetic, as we shall see.

Our first step will be to define a “toy” floating-point arithmetic as three decimal points (a mantissa - XYZ) and an exponents r. Both are fixed point numbers.

$$s = 0.XYZ \times 10^{r} \qquad (26)$$

Accordingly the number 1000 would be 0.100×10^{4} and the number 1 would be 0.100×10^{1}. If we add these, the result should be 0.1001×10^{4}, but we only have three decimal places so the sum is just 0.100×10^{4}. The smaller number is lost. If we have a signal component with amplitude about 1000, we can expect it may at least modulate the amplitude of the smaller signal with amplitude about 1, forcing it partially or completely out of the mantissa. Further, any signal component is likely itself varying. A sine wave with amplitude 1000 is at times large and at times small depending on the part of the waveform we are in.

For a demonstration, assume we have a signal composed of a small, higher frequency sine wave and a larger, lower frequency square wave, where the square wave varies from 0 to its maximum value. In such a case, we can envision the sine wave gated on when the square wave is low, and off when the square wave is high. Fig. 39a suggests such a sine wave, and Fig. 39b shows where the first 10 cycles are on, and the next 10 cycles are off. We understand the panel 39b to represent one “cycle” that repeats indefinitely, If the sine wave is on only half the time, we would first of all expect it to be audibly weaker relative to the case where it is on all the time. But more importantly, the off/on modulation has a very

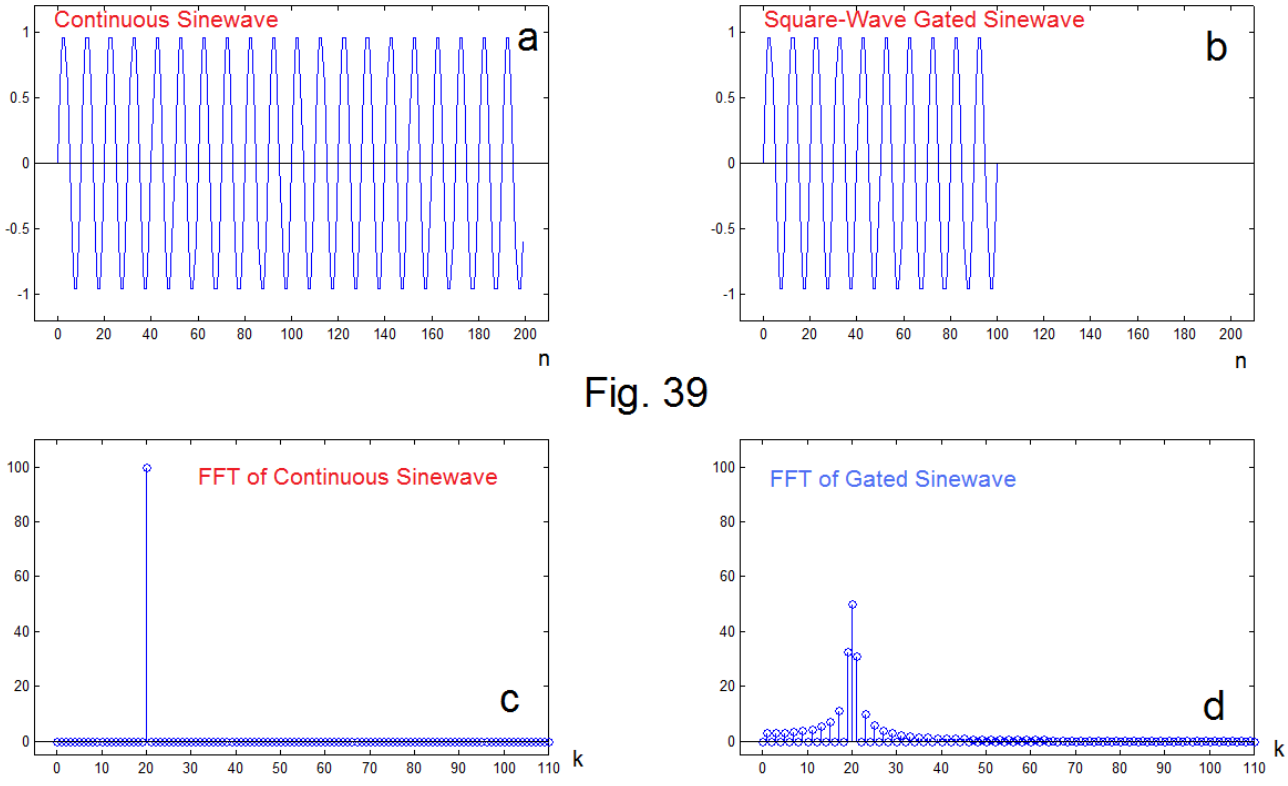


Fig. 39

different spectrum. Fig. 39c shows the spectrum obtained with the continuous sinewave (only the lower half of the FFT magnitude is needed here). We see the single frequency at k=20 as expected. Fig. 39d shows the corresponding spectrum of the sine wave gated on and off. Essentially the single component is smeared. What we see is in fact essentially a convolution of the square wave spectrum (think Fourier series) with the sinewave's spike spectrum. Note that the component at k=20 in Fig. 39d is dues to the dc term in our particular choice of square wave. The other spikes correspond to the squarewave components (frequencies 1, 3, 5….) falling off as 1/3, 1/5, and so on, the imbalance either side of k=20 being due to the contribution of the negative frequency (k=-20) not shown. All very interesting, but the essential thing is the added components of the modulation.

3j. Scaling

As promised, scaling is a topic that had, perhaps 10 years ago, not only a practical need but also a precedent in the analog world, which was very familiar. Basically this is a matter of keeping signal levels reasonable. In the analog case, we wanted to keep the signal small enough everywhere that it did not "clip" against the power supply levels anywhere. The flip side was that we wanted to keep the signal as large as possible to achieve the highest possible signal-to-noise ratio. Here the analog noise level was assumed independent of the signal level. A standard engineering trade-off that could be optimized by careful analysis.

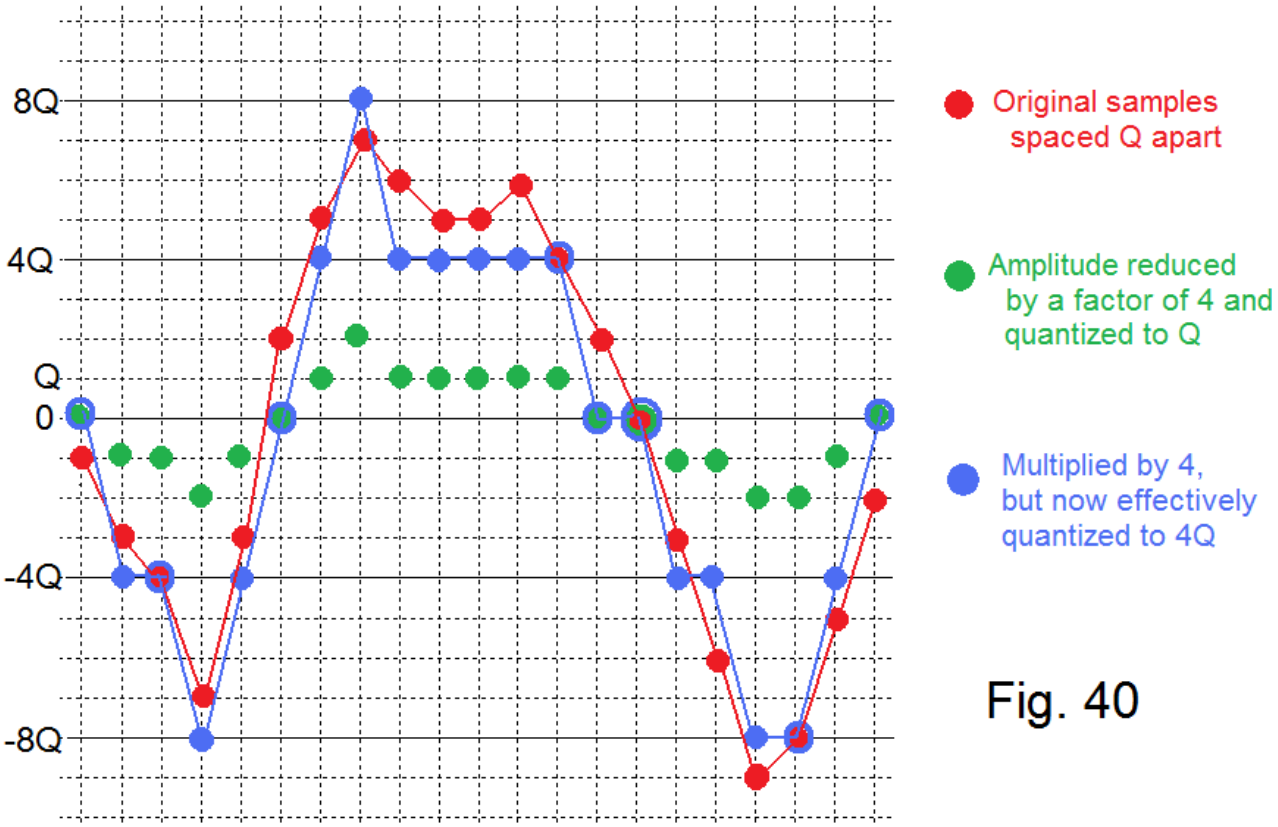


Fig. 40

In the digital case, we want signals to be large enough that they use all or at least most of the bits available, otherwise the signal-to-quantization-noise level may become unacceptable. And, the quantization noise will not go away if the digital signal is amplified (nor would analog noise). We don't get the bits back! Here we are thinking about the fixed point case.

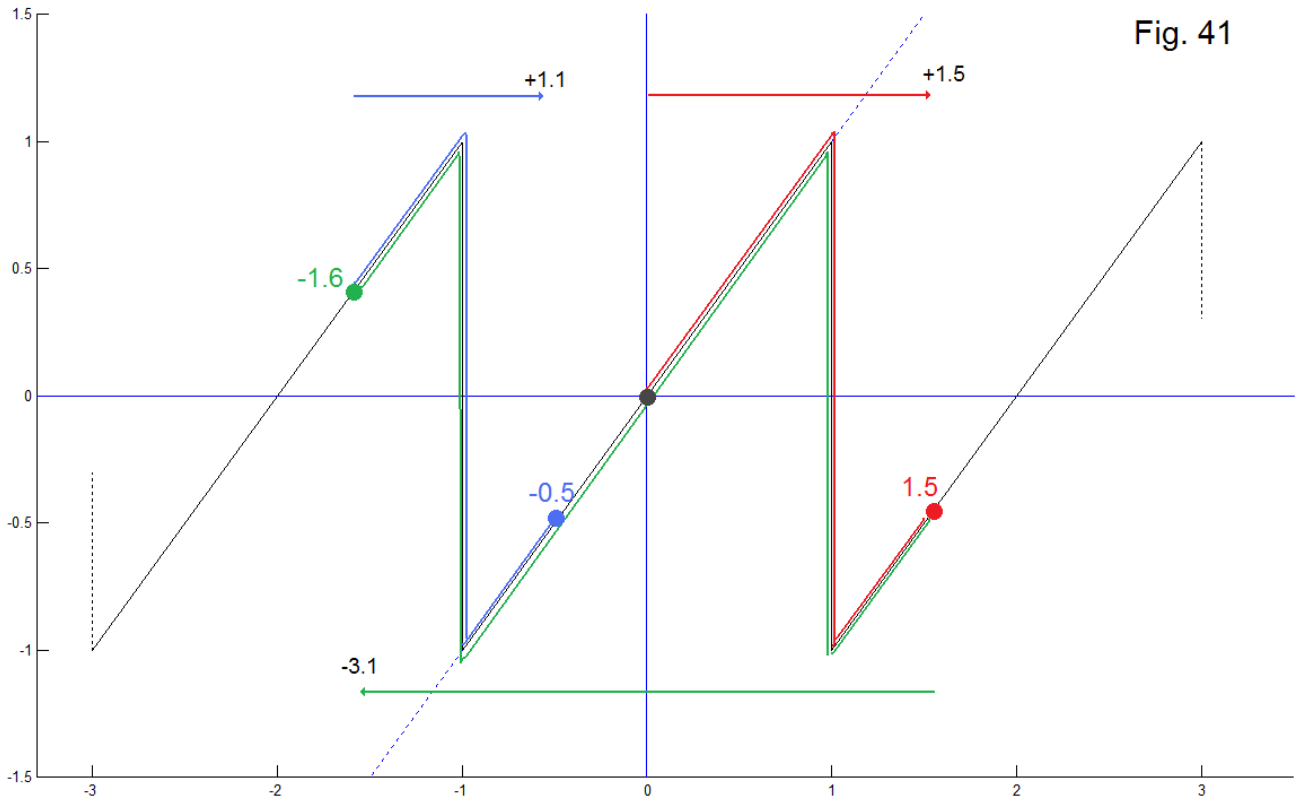
Fig. 40 shows that quantization level options are lost. The red samples are on the grid points separated by Q, and we assume that these are acceptable samples. Note that the signal ranges over 17 levels. Next suppose that the signal gets smaller by a factor of 4. We mean by this that the signal is made to get smaller, not that for example, a person whose voice is being recorded speaks more softly. Specifically, we may have reduced a gain for fear of overflow. Here, because we are changing the gain, each of the red samples must now change to a green sample, still on the grid sized at Q. The second red sample for example starts at 3Q, and now has to take on a value of ¾ Q, which quantizes to Q (green). We end up with only 5 quantization choices, size Q, used by the green dots. This is a smaller, noisier signal. Now we decide to multiply by 4 again. This means the green dots change to the blue dots. Note that there are still only 5 quantization choices (separated by 4Q instead of by Q) following the multiply, even though the signal is larger. This is quite similar to our analog case.

Thus when a digital signal is made smaller, it stops using the most significant bits. On the flip side of that, we didn't want the equivalent of analog "clipping" in the digital case. We should not require samples to have values beyond the number of bits available. We have seen above that overflow oscillations are possible and sometimes hard to remove once started. [However, see Section 3k below]. Once again, a careful design, as in an analog filter, can optimize the scaling issues so that bit levels are not too large or to small, to the extent this is possible to do, at every node in the filter network. [Sometimes, it is not possible because of an inherent peaking, generally in an IIR filter. We do the best we can.]

Here we do not propose to discuss this further. Readers wishing to see a detailed design case can find one worked out elsewhere [12].

3k. Overflow

Having to this point pretty much given low marks to the idea of overflow (like overflow oscillations) we need to remark that sometimes, overflow is good. Indeed, digital signal processing chips likely have a status bit that allows overflow, or forces saturation. In most all cases, a final result in the saturation region is not acceptable. However, there may be many cases where we are not interested in intermediate results – only the final answer is required to be right. A summer in a digital filter often adds up many partial products (perhaps hundreds of them) all of which are not used of kept, except as a partial sum. It is the final result that matters.



Consider the sum of three numbers, 1.5, -3.1, and 1.1 using the same overflow arithmetic in our overflow studies (e.g., Fig. 26). Assume we start our sum at zero. The correct answer should be -0.5. Numbers must be between +1 and -1 in this system, so the correct answer is within range. But as soon as we try to add +1.5, we are in trouble, as the first partial sum should be 1.5, outside the available range. Suppose we are in a saturation mode, so we end up with the biggest number available, +1 in this case. The second number would drive the sum down to 1-3.1 = -2.1, and this too is outside the range, and we would be saturated at -1. The third number added is +1.1, and the sum would end up at -1+1.1 = +0.1. We are inside the allowable range, but the answer should have been -0.5, not +0.1. The answer was wrong, unavoidably, at the first incidence of saturation.

Fig. 41 shows the happy situation that would happen if we allow overflow. The first number added, +1.5 (red) causes overflow and we find that +1.5 has become -0.5 in overflow. Wrong. We then add the -3.1 (green) and end up at -1.6, or 0.4 in overflow. Also wrong. Finally, we add the +1.1 (blue) and move to -0.5, inside the allowable range, and the right answer. Another way to put it is allowing overflow also makes it possible to "un-overflow". It can fix itself as long as we end up in the allowable range at the completion of the sum. Clearly this can be useful. The alternative would have been to cut the size of the signals, with more quantization noise resulting.

REFERENCES

[7] B. Hutchins & R. Bonneau, "Overflow Oscillations and the 'Mickey Mouse' Fractal", Electronotes, Vol. 16, No. 174, June 1989; B. Hutchins & R. Bonneau, "Spawning Locations and Mechanisms for Overflow Oscillation Modes in a Second-Order Digital Oscillator, Electronotes, Vol. 17, No. 177, Nov. 1990.

[8] B. Hutchins, "Analog Signal Processing, Chapter 7, Passive and Active Sensitivity", Electronotes, Vol. 20, No. 195, July 2000; "Tuning Equations Derived from Passive Sensitivities," Electronotes , Vol. 20, No. 196, Dec. 2000; Passive Sensitivities and Tuning Equations", Electronotes Application Note AN-361, May 15, 2005
http://electronotes.netfirms.com/EN195.pdf
http://electronotes.netfirms.com/EN196.pdf
http://electronotes.netfirms.com/AN361.pdf

[9] Our comprehensive digital filter design series is: B. Hutchins, "Basic Elements of Digital Signal Processing – Filter Element", Electronotes, Vol. 20, No. 197, April 2001 (Part 1), Vol. 20, No. 198, June 2001 (Part 2), Vol. 20, No. 199, Sept 2001 (Part 3).
http://electronotes.netfirms.com/EN197.pdf
http://electronotes.netfirms.com/EN198.pdf
http://electronotes.netfirms.com/EN199.pdf

[10] B. Hutchins, "Design Considerations for Digital Sinewave Oscillators," AN-299 Sept/Oct. 1987; B. Hutchins, A View of a Digital Sinewave Oscillator," AN-309, Sept 1990; B. Hutchins, "Frequency Errors and Distortion in Digital Sinewave Oscillators," Electronotes, Vol. 18, No. 186, Sept 1995 (NOTE: TABLE 3 HERE MAY BE SUSPECT.); B. Hutchins, "Some Comments on Fixed Point Digital Oscillator Design, " AN-367, June 2006.
http://electronotes.netfirms.com/AN367.pdf

[11] B. Hutchins, "Options for Low-Frequency, Quadrature, Digital Oscillators," Electronotes Application Note AN-313, Jan. 1991.

[12] B. A. Hutchins and T. W. Parks, Digital Signal Processing Laboratory Using the Tms320C25, Prentice Hall, 1990

* * * * * * *