ELECTRONOTES

1 Pheasant Lane
Ithaca, NY 14850
(607)-273-8030

APPLICATION NOTE NO. 317

January 1992

POLYNOMIAL FITTING FOR SAMPLE-RATE CHANGING AT RATIONAL AND IRRATIONAL FREQUENCY RATIOS

1. INTRODUCTION

Numerous methods for interpolating sample values are available. That is, we have a good variety of ways of finding samples that are, in some sense, the "correct" ones between samples actually available to us. For example, we might have available a set of samples, and we are able to increase this set by a factor of five by obtaining four additional samples for each one we were given. In such a case, we would usually find the additional samples spaced at 1/5, 2/5, 3/5, and 4/5 of the original spacing (in addition to keeping the original samples that are at integer values of the spacing). This is equivalent to increasing the sampling rate by a factor of five (without a corresponding increase in the bandwidth, of course). If we then separated out, from this larger set, every fourth sample, we would decrease the sampling rate by a factor of four. The final sampling rate would be 5/4 the original, and we would have achieved a 5/4 sampling rate change. Fig. 1 shows this general idea.

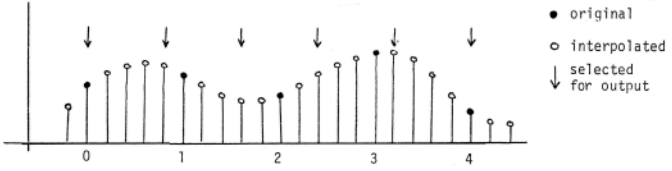

Fig. 1 A 5/4 sample rate change using interpolation by five followed by decimation by 4.

The interpolation process used is often (almost always?) a digital low-pass filtering. That is, the original sequence is first "zero-padded" by putting zeros into the positions to be filled. As such, the new rate that results from interpolation is usually an integer multiple of the original rate (five in Fig. 1). The low-pass filter (operating at the higher clock rate) fills in the interpolated values. Therefore, in the process of selecting the final samples from among these new samples (decimation), we are obliged to choose an integer factor (four in Fig. 1). Thus we have schemes for changing a sampling rate, as long as the rate change is an integer ratio.

We do have methods, however, for finding the samples at <u>any</u> points between existing samples - not just at equally spaced points. Indeed, the most well-known recovery process of all (ideal low-pass filtering - the reconstruction method of classical sampling theory) provides a continuous version of the signal that corresponds to the samples. In the time domain, we convolve the available samples with the impulse response of an ideal low-pass filter (a continuous sinc function) for the exact same result. This is the sinc interpolation method.

In such cases, where we can calculate the interpolated values at any point between existing samples, we could achieve any sampling rate change desired, and not be restricted to integer ratios. For example, if we wanted to change the sampling rate to a higher value by a factor of $\sqrt{2}$, we would calculate the first sample at the original position, the second at $1/\sqrt{2}$ of the original spacing, the third at $2/\sqrt{2}$ of the original spacing, and so on. (We would never again use an original sample by itself.) Fig. 2 shows this general idea, in contrast to Fig. 1

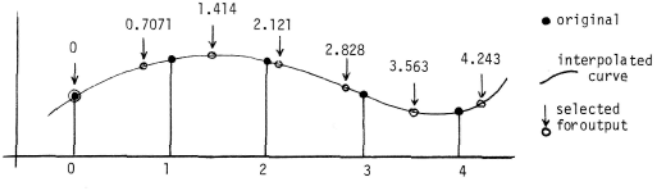

Fig. 2 A $\sqrt{2}/1$ sample rate change by calculating the output values using a continuous interpolation method.

Continuous interpolation can be done with sinc interpolation. There are two problems however. First, in theory the sinc interpolation (ideal low-pass filtering) needs to be calculated over all possible non-zero samples. In practice, we would probably have to settle for a finite set of samples spaced about the point where we are calculating the new output. (This need not be a very serious problem since the 1/t term inherent in the sinc function helps make samples outside the chosen range relatively insignificant.)

The second problem concerns implementation: sinc functions $[Sin(x)/x]$ are difficult to compute with the usual type of DSP chip, and we would like to use (or have to use) such high-speed DSP chips for practical applications. DSP chips are not good at calculating sine functions, or at doing division. Accordingly we might look for a different method of continuous interpolation that better suits DSP chips. Here we will look at polynomial fitting.

2. FITTING POLYNOMIALS

Finding polynomials that fit a certain set of data points is a classical problem, and one which is not too difficult. It is perhaps not at all clear that polynomials might serve as reasonable interpolating functions for practical signal bearing functions. [Polynomials "wiggle" a few times and then run off to infinity outside some range, while signaling functions are most often considered to be combinations of sinusoidals, which "wiggle" many times and remain well within certain amplitude limits.] What is even less obvious is the fact that a polynomial interpolating procedure can be reduced to a time invariant FIR filtering process.

2a. Fitting a Line to Two Points

Consider the problem of fitting a straight line to two given points. This is a very simple example, and at the same time, one that is quite special. In the first place, anyone can do it. Even if you don't remember the formulas from high-school math (and many people do remember these), you can always just plot the points and use a straight edge and a pencil. Conceptually, we have no difficulty with this idea. Secondly, this procedure - putting a straight line between two points - is linear interpolation. In addition, a straight line is a polynomial (a first-order one), so we already know that polynomial interpolation is a reality.

Regardless of whether or not we remember how to fit a straight line by formulas, we need to recognize here that what we are actually doing is solving simultaneous equations, where the coefficients of the polynomial are unknowns, and the given points are knowns. Based on this idea, we can rederive formulas if we don't remember them. We will be taking this one step further as well - to show that we are dealing with an FIR filtering problem.

We begin with the idea that we have two points $y_0(t_0)$ and $y_1(t_1)$ and that we want to find the straight line $y(t) = at + b$ that goes through them. Clearly we can cast this problem in the form of two simultaneous equations. For convenience, we will also take $t_0 = 0$ and $t_1 = 1$. Thus:

$$y_0(t_0) = y(0) = at + b = a \cdot 0 + b = b \tag{1a}$$

$$y_1(t_1) = y(1) = at + b = a \cdot 1 + b = a + b \tag{1b}$$

Solving these we have:

$$b = y(0) \tag{2a}$$

$$a = y(1) - y(0) \tag{2b}$$

and our straight line is thus:

$$y(t) = at + b = [y(1)-y(0)]t + y(0) \tag{3}$$

Equation (3) can be recast in the form:

$$y(t) = y(0)[1-t] + y(1)t \qquad (4)$$

which we can think of in two important ways. First, we can think of it as giving y(t) as a weighted sum of interpolating functions. The weights are y(0) and y(1) while the interpolating functions are (1-t) and t for this first-order case. Secondly, as we will see below, it is in the form of an FIR filter.

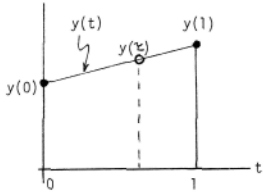

Fig. 3 Linear interpolation from equation (4)

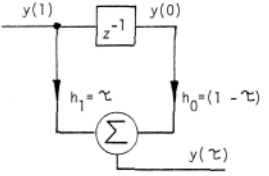

Fig. 4 FIR filter from equation (4)

At this point, instead of thinking of equation (4) as an equation for a function y(t), we can think of it as a function y(τ) where τ is a particular value of t. Thus $y(\tau) = y(0)[1-\tau] + y(1)\tau$ where we are now thinking of τ as a constant. In this view, Fig. 3 shows that y(τ) is a linear interpolation of a value of y(t) corresponding to t=τ when the points y(0) and y(1) are known. Note that τ can be any value of t, but is normally considered an interpolation only when it is between t=0 and t=1. For the purposes of this note (rate changing at non-integer ratios) it is important to realize that τ can be an irrational number. The interpolating functions (first-order polynomials or straight lines) are seen in Fig. 5.

Fig. 4 shows the FIR filter form of equation (4). Note again that τ is a constant, so the tap weights of this filter are constants. It is clear that the network performs the summation of equation (4). Here it may also seem that this filter only solves the problem for t=τ, and for no other points. However, when we recall the nature of an FIR filter, we note that samples are not simply loaded into the filter, to remain there forever, but actually clock through the filter. That is, during each clocking interval, samples clock from left to right. Therefore, eventually y(1) moves to the y(0) position while y(2) (a new sample) moves to the y(1) position, while y(0) is gone. At this time, the interpolation problem is the same as it was, except now we are interpolating

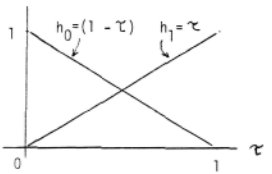

Fig. 5 Interpolating functions for first-order curve fitting

between y(1) and y(2). Accordingly, even though the tap weights remain τ and $(1-\tau)$, we are now solving the problem at $1+\tau$. After another clocking, we are solving it at $2+\tau$, and then at $3+\tau$, and so on. We can therefore regard τ not just as an offset from t=0, but an offset from all original samples.

For example, suppose we want $\tau=1/2$, then $y(n+1/2) = y(n)/2 + y(n+1)/2$, which is the average value, the correct solution for a linear interpolation. Note however that the filter of Fig. 4 neither gives back the original samples, nor does it give back any value in the time positions of the original samples. It only calculates the samples in between, offset by τ. For our example of $\tau=1/2$, we could obtain a 2:1 interpolator by interleaving these calculated samples between the original ones, and this general idea is probably straightforward, useful, and will be seen in Section 3 below. Below, eventually, in the case of interpolating for an irrational rate change, we will not be looking at special (fixed) values of τ. In this cases, none of the original samples would ever be used, and all new samples will have to be calculated with time-varying tap weights.

2b. Fitting a Curve to Three Points

Much of Section 2a was fairly trivial. Here we will move on to the problem of fitting three points, and things change drastically in that we can not trivially find the answer. We might recognize that in order to fit three points we need a second-order curve, and that a second-order curve is a parabola, but we don't have a "parabola ruler" or any equivalent mental notion to work with. Instead, in this case we must set up and solve the equations. This was the reason we took several laborious extra steps in the two point case.

The three point case was actually presented as an example within a fairly comprehensive report in Electronotes. See B. Hutchins, "Interpolation, Decimation, and Prediction of Digital Signals," Electronotes, Vol. 15, No. 164-167 (Special Issue F), July 1986 for the original presentation and related material.

Following the same procedure we did for the two-point case, we have three known points at time t=0, t=1, and t=2, and will denotes these as y(0), y(1) and y(2) respectively. The polynomial will be taken to be:

$$y(t) = at^2 + bt + c \tag{5}$$

and our three equations are:

$$y(0) = c \tag{6a}$$

$$y(1) = a + b + c \tag{6b}$$

$$y(2) = 4a + 2b + c \tag{6c}$$

which are solved to give:

$$a = y(2)/2 - y(1) + y(0)/2 \tag{7a}$$

$$b = -y(2)/2 + 2y(1) - 3y(0)/2 \tag{7b}$$

$$c = y(0) \tag{7c}$$

so the polynomial is:

$$y(t) = [y(2)/2 - y(1) + y(0)/2]\, t^2 + [-y(2)/2 + 2y(1) - 3y(0)/2]\, t + y(0) \tag{8}$$

which can be recast as:

$$y(t) = [t^2/2 - 3t/2 + 1]\, y(0) + [-t^2 + 2t]\, y(1) + [t^2/2 - t/2]\, y(2) \tag{9}$$

which can then be written as an FIR filter:

$$y(t) = h_0\, y(0) + h_1\, y(1) + h_2\, y(2) \tag{10}$$

where the tap weights $h_n(t)$ are seen in equation (9) to be:

$$h_0(t) = [t^2/2 - 3t/2 + 1] \tag{11a}$$

$$h_1(t) = [-t^2 + 2t] \tag{11b}$$

$$h_2(t) = [t^2/2 - t/2] \tag{11c}$$

At this point we have all we need to go ahead with an FIR realization, as we did in the two point case. However, things are a bit different in that there are now two possible choices where we may wish to interpolate points: between t=0 and t=1, or between t=1 and t=2. [In the two point case, it was fairly obvious that we were

looking at the region between t=0 and t=1 only (although the possibility of extrapolation or prediction outside the range of given points is sometimes interesting to look at.)]

In this three point case, either choice of interval (t=0 to t=1, or t=1 to t=2) is possible. Our only preference might be to chose a value that is near the center of the given points (that is, close to t=1 for this case). Intuitively we feel that the fit is best near the center where there are more given points on each side to "stabilize" the curve fit. [In particular, we know that the polynomials will run off to $+\infty$ or $-\infty$ rapidly when we move outside the range of given points].

In the two point case, when it got to an actual FIR filter realization, we saw an equivalence between points that were separated by integers. That is, each time the filter clocked, we re-solved the interpolation problem for a new set of samples, or equivalently, for a different time. Only the offset from the integers, the value we have called τ, made any difference in determining the tap weights.

In this three point case, we might want to obtain interpolated points offset from the integers by +3/4. For this value, taking τ=3/4 (*inside the first interval*) *makes sense, since 3/4 is close* to 1, the center of the three given points. On the other hand, if we wanted interpolated points offset from the integers by +1/4, taking τ=1/4 probably would not be a good choice, since 1/4 is closer to 0 than it is to 1. For the 1/4 value, the choice of the second interval would be better, and we would take τ=1.25.

Fig. 6 and Fig. 7 show these ideas by showing the polynomial fit and corresponding FIR filters, with example points y(0)=1, y(1)=0, and y(2)=2. For the τ=1.25 case, the sample at the output is clearly the one offset from y(1) by 0.25 and offset from y(0) by 1.25. The sample offset from y(0) by 0.25 was available at the previous clocking of Fig. 7b, where y(-1) was in the y(0) position, y(0) in the y(1) position, and y(1) in the y(2) position. [Thus we

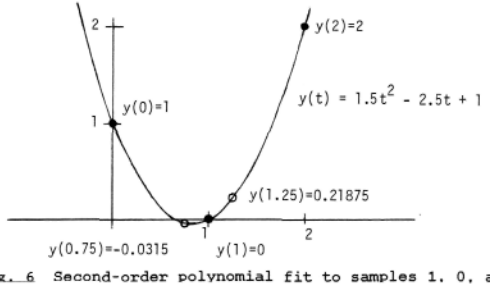

Fig. 6 Second-order polynomial fit to samples 1, 0, and 2

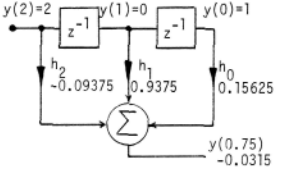

Fig. 7a Second-order interpolation at τ=0.75. Tap weights from equations (11a), (11b), and (11c) for $t = \tau = 0.75$

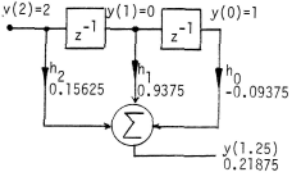

Fig. 7b Second-order interpolation at τ=1.25. Tap weights from equations (11a), (11b), and (11c) for $t = \tau = 1.25$

need to think about the clocking interval involved as well as the coefficients when determining exactly which point is interpolated.] Also, note that the τ=1.25 case can be thought of as the τ=3/4 case with samples in the reverse direction, and this fact is seen in Fig. 7 by the reversal of the tap weights for the two different cases.

Having now looked at this result in terms of an interpolation, and in terms of an FIR filtering process, it is instructive to look separately at the three functions in equations (11a), (11b), and (11c). This curve fitting interpolation method is known as "Lagrange Interpolation", but these three functions are usually called "sampling polynomials" [R.W. Hamming, Numerical Methods for Scientists and Engineers, Dover (1973)]. We will find it most useful here to continue using the notation $h_0(t)$, $h_1(t)$, and $h_2(t)$, thinking

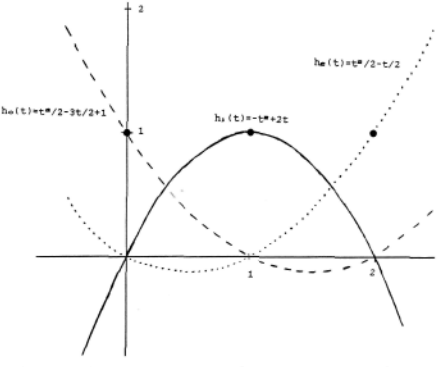

Fig. 8 Sampling Polynomials for 2nd-Order

of these as tap weights. By the time we finish, the idea of having these tap weights as functions of time will seem natural, although we will no longer think of them as an impulse response (which applies only to linear, time-invariant systems).

The "sampling polynomials" for the three-point cases are shown in Fig. 8. [Compare these with the corresponding two-point case, Fig. 5 where $h_0(t) = 1-t$ and $h_1(t) = t$.] We note several things. First, just as in Fig. 5, we see functions crossing over each other in an x-like manner. There are three such x-patterns: $h_0(t)$ and $h_1(t)$ cross in the interval from t=0 to t=1, $h_1(t)$ and $h_2(t)$ cross in the interval from t=1 to t=2, and $h_0(t)$ and $h_2(t)$ cross in the interval t=1/2 to t=3/2. These features suggest that curve fitting of orders greater than first-order still have component functions that perform a rough linear interpolation, which is then refined by the actual curvature of the functions and by additional contributing functions.

2c. Fitting a Curve to Four (or More) Points

In Section 3 of this note we want to show exactly how networks that calculate individual interpolated samples can be interleaved to form a full interpolator network - a low-pass filter, in fact. It will be convenient, and most illustrative, to use an interpolating polynomial that is of odd order, and greater than first-order. Accordingly, we will develop the third-order case below, despite the fact that the procedure is probably quite clear without doing this additional example.

The third-order polynomial is:

$$y(t) = at^3 + bt^2 + ct + d \tag{12}$$

so our equations are:

$$y(0) = d \tag{13a}$$

$$y(1) = a + b + c + d \tag{13b}$$

$$y(2) = 8a + 4b + 2c + d \tag{13c}$$

$$y(3) = 27a + 9b + 3c + d \tag{13d}$$

which are solved to get:

$$a = -(1/6)y(0) + (1/2)y(1) - (1/2)y(2) + (1/6)y(3) \tag{14a}$$

$$b = y(0) - (5/2)y(1) + 2y(2) - (1/2)y(3) \tag{14b}$$

$$c = -(11/6)y(0) + 3y(1) - (3/2)y(2) + (1/3)y(3) \tag{14c}$$

$$d = y(0) \tag{14d}$$

so the polynomial as given in (12) can be recast as:

$$y(t) = [\ -(1/6)t^3 + t^2 - (11/6)t + 1\ ]\ y(0)$$
$$+ [\ (1/2)t^3 - (5/2)t^2 + 3t\ ]\ y(1)$$
$$+ [\ -(1/2)t^3 + 2t^2 - (3/2)t\ ]\ y(2)$$
$$+ [\ (1/6)t^3 - (1/2)t^2 + (1/3)t\ ]\ y(3) \qquad (15)$$

so the tap weights are:

$$h_0(t) = [\ -(1/6)t^3 + t^2 - (11/6)t + 1\ ] \qquad (16a)$$

$$h_1(t) = [\ (1/2)t^3 - (5/2)t^2 + 3t\ ] \qquad (16b)$$

$$h_2(t) = [\ -(1/2)t^3 + 2t^2 - (3/2)t\ ] \qquad (16c)$$

$$h_3(t) = [\ (1/6)t^3 - (1/2)t^2 + (1/3)t\ ] \qquad (16d)$$

Once again, the tap weights are the sampling polynomials, and are sketched in Fig. 9. We again see the various x-patterns in the central region: $h_0(t)$ and $h_1(t)$ cross between t=0 and t=1, $h_1(t)$ and $h_2(t)$ cross between t=1 and t=2, and $h_2(t)$ and $h_3(t)$ cross between t=2 and t=3. Note that between t=1 and t=2 that $h_1(t)$ and $h_2(t)$ cross in a manner that looks very much like linear interpolation (Fig. 5) since $h_0(t)$ and $h_3(t)$ are very small in that region.

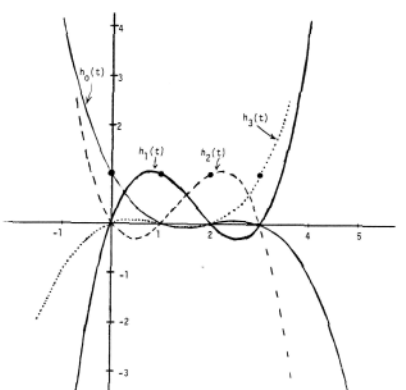

Fig. 9 Sampling Polynomials for Third-Order

We may desire to fit even higher-order polynomials to larger and larger number of points. In such cases, a general formula for the sampling polynomials (tap weights) $h_n(t)$ will prove more efficient than equation solving as we have been using above. The general formula is:

$$h_n(t) = \prod_{\substack{m=0 \\ m \neq n}}^{N} (t - t_m) / (t_n - t_m) \qquad (17)$$

where N is the order of the polynomial fit (N+1 points to fit). This formula comes from the idea that we want the n-th sampling polynomial to have zeros at all times t_m except at time t_n when it should be equal to one. [See Fig. 8 and Fig. 9 for examples.] For example, we could generate $h_3(t)$ for the 3rd-order case, where the four points to fit occur at times t_n, where $t_n = n$ for $n = 0, 1, 2,$ and 3.

$$h_3(t) = (t-0)(t-2)(t-3) / (1-0)(1-2)(1-3)$$

$$= t(t^2 - 5t + 6) / 2$$

which agrees with equation (16b) above.

3. INTERLEAVING INTERPOLATING SECTIONS FOR AN OVERALL FILTER

We have noted above that the FIR filters we have developed are used to calculate the interpolated value of a function at positions in time that are offset from the original points by a certain amount which we have denoted by τ. Thus while these networks calculate points to be interpolated, they do not interpolate points in the sense of inserting these calculated points between original points and/or among other interpolated points. Thus we must develop additional mechanisms to complete the job. What we do here is very similar to what was done in the previous application note.

Fig. 10 shows a intermediate scheme that helps us understand what needs to be done. [For a number of reasons, this scheme is not practical, but we have no intentions of using this except as an intermediate step.] We have here four FIR filters in parallel. The top one computes the sample at time offset τ=1. As can be seen, this is simply a matter of setting one tap to one, and the other three taps to zero. The remaining three FIR filters are not as trivial, and compute the samples at time offsets τ=1.25, τ=1.50, and τ=1.75. The switch shown then operates at four times the original sampling frequency, and sequences these outputs in the proper order.

Fig. 11 shows a more conventional way of realizing the full interpolation scheme. Here we achieve a standard FIR filter operating at four times the original sampling rate. The function of the switch in Fig. 10 is thus replaced by having the input sequence zero-padded with three zeros between each of the original samples.

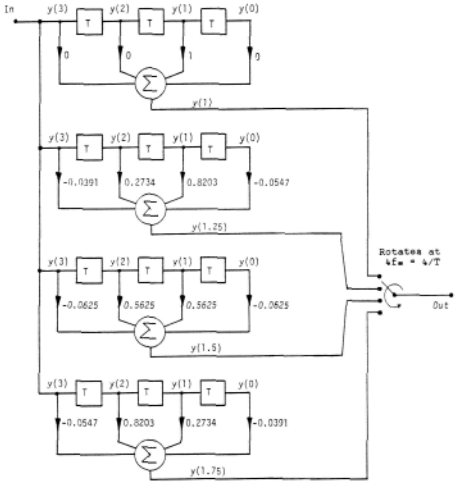

Fig. 10 Interleaving four FIR filters, each of which calculates an interpolated point.

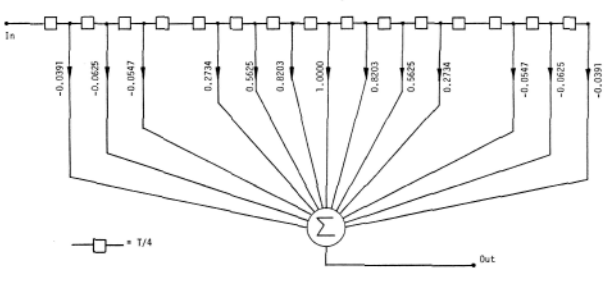

Fig. 11 An FIR filter that is equivalent to Fig. 10 which uses four times the original sampling rate

As in pervious case, the final result is a low-pass filter. Fig. 12 shows the frequency response of the component FIR filters of Fig. 10, while Fig. 13 shows the frequency response of the overall low-pass of Fig. 11.

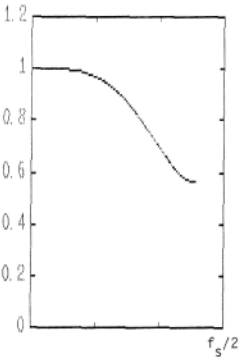

Fig. 12a Freq. response of interpolator for 1/4 or 3/4

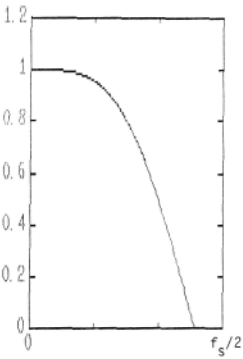

Fig. 12b Freq. response of interpolator for 1/2

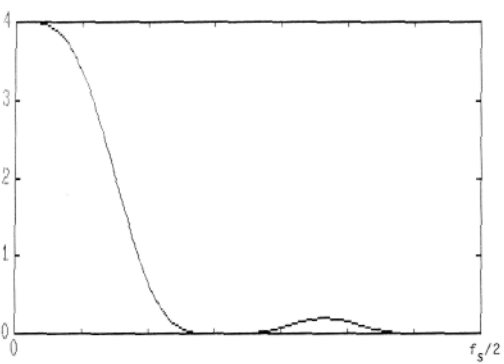

Fig. 13 Freq. response of overall interpolator of Fig. 11

4. IRRATIONAL RATE CONVERSION

One of the attractive features of polynomial fitting was the ability to compute the sample at any point between original samples, and not just at a finite set of equally spaced points between the original sample. We have also remarked about the ease of calculating the tap weights for the polynomial method, for the case where we want to do sample-rate conversion with a DSP chip. Below, we want to show more details of just how this might be done.

4a. Synchronization

One problem comes up immediately, and it is a consideration with either a sample rate conversion at a rational fraction, or at an irrational fraction. This is a question of synchronization. Even if we are converting at an integer ratio, just how precise are the original and target rates?

In some cases, we may be thinking in terms of converting one data file to another. For example, we might have a data file that has been recorded at one rate, and we want to double this rate before playback. This would be a matter of computing intermediate samples, putting them between originals, and ending up with a file that is twice as large. In such a case, the samples are exactly where they belong - precisely in the middle.

However, another case would be one where we have two machines which operate at different nominal sampling frequencies, and these two machines need to exchange data. For example, one might be a playback machine and the other is a recorder, and we are essentially making a copy. Suppose for example that the playback machine is putting out data at a nominal 40 kHz rate and the recorder wants data at a nominal 50 kHz rate. This appears to be a simple 5:4 conversion (Fig. 1). However, neither machine has an infinite precision clock. The 40 kHz machine might actually have a 39.99931... kHz clock and the 50 kHz machine might actually have a 50.0024... kHz clock, and both might drift with time and temperature.

Accordingly, one of the two has to get to be the boss machine. It has to control the clock of the other, or tell the other machine when it want to receive or transmit a sample. If necessary, this also serves as a request to actually interpolate (calculate) a sample that the sending machine does not have available. If this synchronization did not happen, we would expect difficulties, at least to the point where some occasional samples would become confused or lost as timing edges pass over each other. Synchronization may thus already be a necessary part of some data transfer schemes, regardless of the fractional rate change.

4b. Linear Interpolation at an Irrational Rate

For our example, we will consider the problem of obtaining a sampling rate conversion at an irrational rate using linear

interpolation. As noted above, linear interpolation is the simplest case of polynomial interpolation. Extension of the method to a higher order polynomial fit is straightforward.

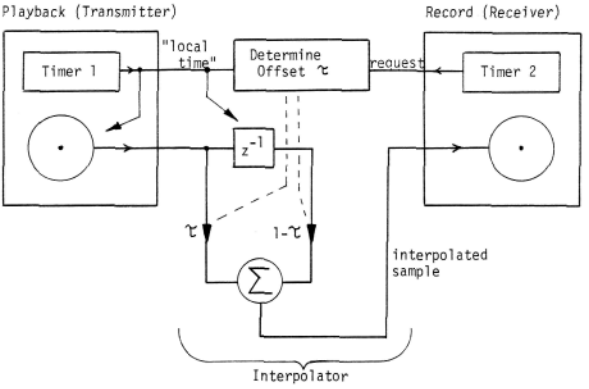

Fig. 14 Sample rate conversion at irrational fraction ratio using linear interpolation

Fig. 14 shows our example system. We assume that we have a transmitting (playback) machine and a receiving (recording) machine. We assume that the playback machine has stored data corresponding to the original rate, and that this rate is different from the desired rate of the recording machine. Each of the machine has its own timer which sets the sampling rate. Between the two machines, we show the interpolator devices.

We think of the system as having Timer 1 control the data output from the transmitting machine. This timer also clocks data on the delay line of the interpolator. Timer 2 controls the sampling rate of the receiving machine. When the receiver needs a sample, it sends a request to the interpolator. It then expects a sample back after a fixed time interval. Accordingly, Timer 2 requests a sample slightly in advance of the time it actually needs one.

When the interpolator receives the request, it looks up the "local time" of the transmitting machine. That is, it needs to determine the offset τ from the time of the last clock of Timer 1 to the time of the request from Timer 2. It may require some effort to find this τ, however.

Note that while this value of τ is easy to understand in principle, it is by no means always easy to obtain in practice. In fact, the whole method is sometimes trivial, except for the difficulty of getting τ. The problem is that we have explicitely invoked discrete time as a sampling rate of a system, and now we are asking to re-derive its continuous counterpart. In some cases, the sampling rate of a system is set by a clock associated with an A/D or D/A converter. Any DSP chip associated with this system is probably clocking much faster (doing its work, and then waiting for the sampling rate clock to initiate another processing cycle). In such cases, the finer "ticks" or the higher frequency clock can be used to give an excellent estimate of τ.

In any case, once we assume that we have τ, we calculate the tap weights of the interpolator, and then the interpolated sample itself. This sample is then transmitted to the receiving system. Note that the interpolator looks like an FIR filter in this case, but it is actually time-varying, and can not be treated as a time-invariant filter. It is just a device that calculates the correct answer.

The extension of this idea to higher order polynomial fits is straightforward - we just have more taps to deal with. In fact, we could also use τ to interpolate by other methods, such as sinc interpolation.