Filtration

Now that we have looked at the concept of interpolation we have seen practically that a "digital filter" (hold, or interpolate) can affect the frequency response of the overall system. We need to codify this and understand it.

The general form of the convolution sum for a "Finite Impulse Response" (FIR) filter is:

$$y[n] \quad = \quad \sum_{m=0}^{M} \; h[m] \; x[n-m]$$

There are two issues to be aware of at here:

1) What sort of frequency responses can we "cook up" using various forms of *h[t]*

2) How fast can we implement these in a system

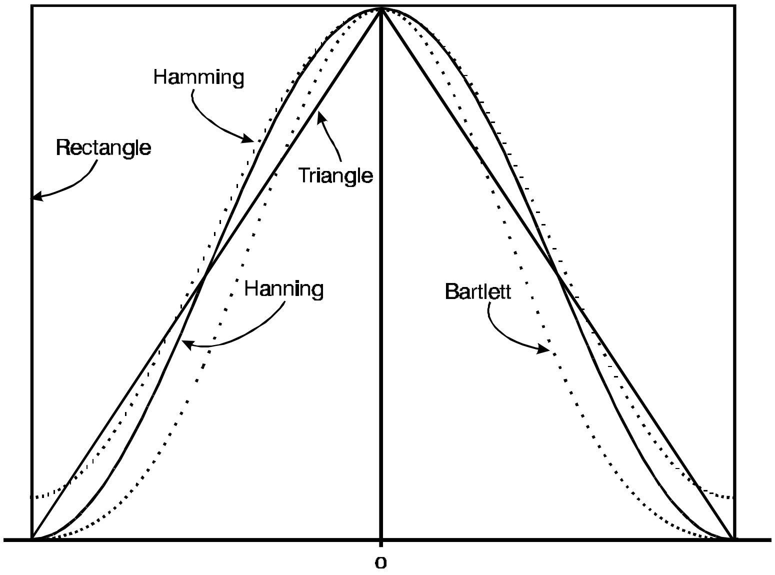
To deal with the second item first. There now exist families of specialised computers called "Digital Signal Processors" (DSPs) which are optimised for this sort of function. One of the simpler ones - the Motorola DSP56001 - is capable of evaluating such a sum at a rate of 16 x10^6 terms/second. This means that with a bit of overhead it can handle about 12-13 term convolutions at 1MSample/second - a Nyquist frequency of 500kHz. There are faster processors. Thus for the low frequency end of things the hardware is quite easily available.

Now to look at the first problem - what sort of frequency responses can we generate? The general answer to this is very complicated as there is no perfect procedure for getting from the frequency response which is required to the set of coefficients for *h[t]*. What I am going to do is suggest some ideas and show you at least one example. If you want to pursue this problem any further, then you will need to do some more extensive reading.

We first note that the set of coefficients should be symmetrical. For high-pass filters the total number of coefficients should also be an odd number. There are good reasons for this concerned with the phase response of the system which I won't go into here.

Now without any further ado, let's try to design a filter which is lo-pass and cuts off at $F = 1/4T$ - half the Nyquist frequency. We also know that we want to pass frequencies 20% below that, 0.2/*T*, and reject frequencies 20% above that, 0.3/*T*. This specifies the cut-off frequency and the "transition band", $\Delta f = (0.3\text{-}0.2)/T$. Since we can't design a "brick-wall" filter exactly we are going to have to do some approximation and we shall need to know how much attenuation we want in the stop band. This is assigned at -60dB or 10^{-3}. Now before we go on I should point out that this is a very challenging design in analog terms. The main problem with analog filters is matching the components to high precision - as the filter becomes more precise then the matching becomes more critical. Aging and general component drift as well as difficulty of manufacture take a severe toll in terms of cost and reliability. On the other hand a digital filter either works or it doesn't and if we produce 1000 of them, they are all identical.

Above we noted that the main problem with a "brick-wall" filter was that it had an infinite impulse response and so couldn't be implemented. However if we multiply the impulse response by a function which actually goes to zero at finite time then the response becomes finite. Multiplication in the time domain is convolution in the frequency domain and so this will tend to "spread" the rectangle in the frequency domain. This procedure, known as "windowing" allows a controlled approximation to the ideal. There are a fair number of window functions available. Some common ones are:



Common Windowing Functions

Square pulse - no further explanation needed

Triangular pulse (Bartlett) - similar to the interpolation filter we used above

Cosine function (Hanning) - $(1 + \cos x)$ out to $x = \pm\pi$

Modified Cosine (Hamming) - $(1.08 + 0.92\cos x)$ out to $x = \pm\pi$.

Blackman - $(0.84\text{-}\cos(x) + 0.16\cos(2x))$ out to $x = \pm\pi$.

Kaiser - a window based on the zeroth-order modified Bessel function $I_0(\alpha)$[31]. $I_0(\beta(1-x^2)^{1/2})/I_0(\beta)$. Notice that this window has a second parameter β, which controls it's shape.

Now we need some "cooking" formulas to help us decide how long a series we will need and what the coefficients will be. For the Kaiser window, there are some good formulae which state that if A is the required attentuation in dB, F the cut-off frequnecy and Δf is the transition band both expressed in terms of the sampling rate T, then the term β is given by:

$$\beta = \begin{cases} 0.1102\ (A - 8.7) & A > 50 \\ 0.5842\ (A - 21)^{0.4} + 0.07886(A - 21) & 21 \leq A \leq 50 \\ 0.0 & A \leq 21 \end{cases}$$

and the <u>total</u> length of the series is given by:

$$M = \frac{A - 8}{14.357\, T\, \Delta f}$$

and finally the series itself is given by:

$$\frac{\sin 2\pi FT(n - \alpha)}{\pi\,(n - \alpha)} \quad \frac{I_0[\,\beta\,(1 - [\,(n - \alpha)/\alpha\,]^2)^{1/2}\,]}{I_0(\beta)} \qquad 0 < n < M$$

where $\alpha = M/2$.

For our problem:

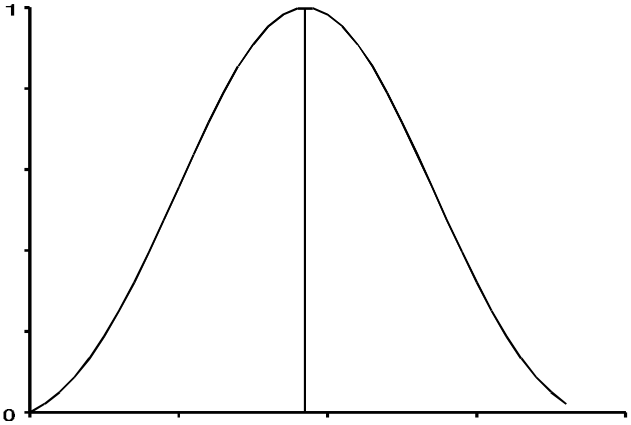
[31] Evidently we will need to know how to calculate $I_0(x)$. A formula is

$$I_0(x) = \sum_{n=1}^{\infty} \left\{ \prod_{m=1}^{n} \left(\frac{x}{2m} \right)^2 \right\}$$

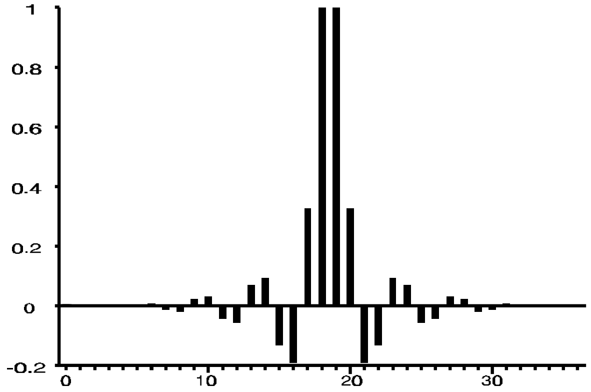
$$\begin{aligned} F &= 0.25/T \\ \Delta f &= 0.1/T \\ A &= 60 \end{aligned}$$

Thus from the first formula $\beta = 5.653$, from the second $M = 36.2$ or, since M must be integer $M = 37$ and then $\alpha = 18.5$.

The Kaiser window given by this formula and the set of coefficients which result are shown graphically below:

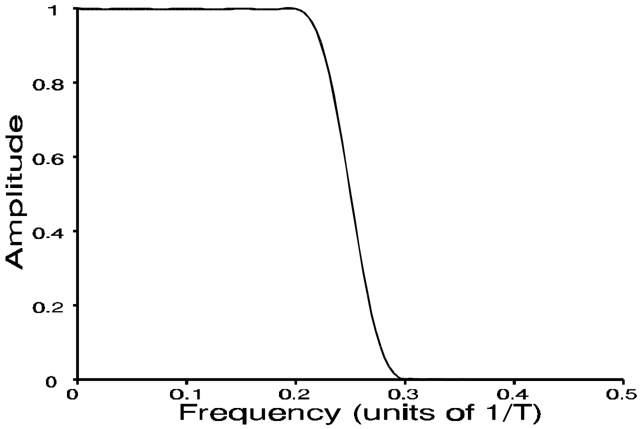


Kaiser Window for β=5.653

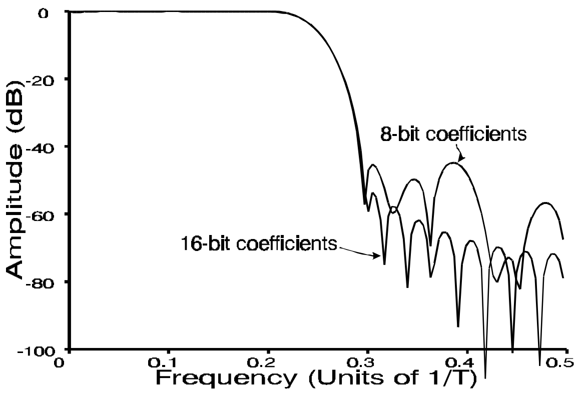


Coefficients of FIR Filter example

Then the response is given below.



Linear Response of Filter



Log Response of Filter

Since the linear response is so good, we need to use the log or dB plot to verify that the performance is being met. Now it appears that the specification is being violated in the first part of the passband. I can't tell whether this is due to my analysis routines or to an actual failure - in which case we would need a slightly longer series. Remember that the response of this filter is actually even about the origin and repeats aobut 1/*T*, the sampling frequency so that the plot immediately starts to mirror after the 0.5 mark.

A further point is illustrated in the dB plot (which is also known as a Bode plot) in that I played with the coefficients by changing the length of the represention of the number, ie changing the precision with which it is stored. I used a common storage method for these coefficients which is to use a "fixed-point fraction" which means that the MSB of the binary representation represents 1/2, the next 1/4 and so on to the LSB. The plots for 24-bit precision and 16-bit precision are identical to the resolution of the graph, but at 8-bit resolution the performance is degraded. This is equivalent to the problem of matching components in an analog filter - and has the same degrading effect on the performance.

The actual coefficients are shown below. Since the coefficients are symmetric only the first half of the sequence is tabulated to save space.

Term	**24-bit precision**	**16-bit precision**	**8-bit precision**
0	0.00000000	0.00000000	0.00000000
1	0.00061631	0.00061035	0.00000000
2	0.00153953	0.00152588	0.00000000
3	-0.00284916	-0.00285339	-0.00390625
4	-0.00463533	-0.00463867	-0.00781250
5	0.00700122	0.00698853	0.00390625
6	0.01006670	0.01005550	0.00781250
7	-0.01397450	-0.01397710	-0.01562500
8	-0.01890130	-0.01890560	-0.01953120
9	0.02507850	0.02507020	0.02343750
10	0.03282610	0.03282170	0.03125000
11	-0.04261850	-0.04263310	-0.04296880
12	-0.05520900	-0.05522160	-0.05859380
13	0.07189540	0.07188420	0.07031250
14	0.09513620	0.09512330	0.09375000
15	-0.13022700	-0.13023400	-0.13281200

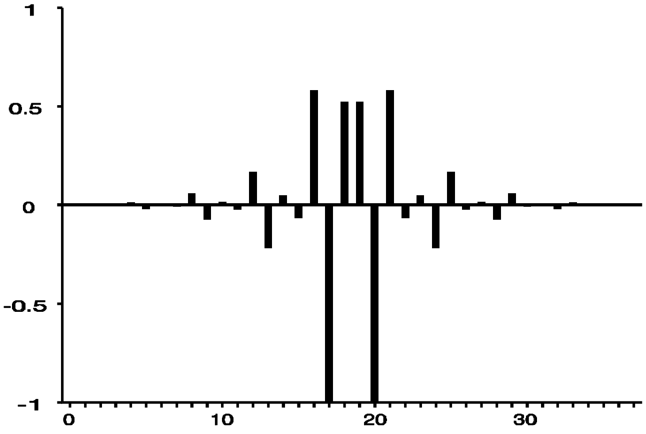
16	-0.19099300	-0.19099400	-0.19140600
17	0.32826700	0.32826200	0.32812500
18	1.00000000	1.00000000	1.00000000

So we have seen that using an real procedure we can actually design a filter which can achieve a reasonable performance using a very modest number of terms. I stress that this is not the only way of designing a filter - there are many computer programs which perform the same task - but this method is clearly laid out as a set of procedures.

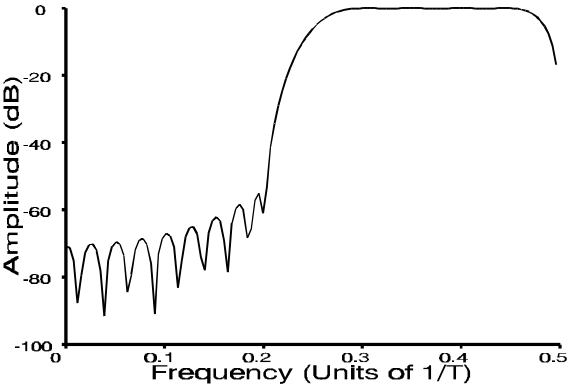
So much for lo-pass filters - how about hi-pass ones. Well here we can appeal to the properties of linear systems to say that a hi-pass filter is the result of subtracting a lo-pass filter from an all-pass filter (one which passes all frequencies). Further since the sampled functions are even in frequency and repeat at the sampling freqency $1/T$, an all-pass filter is the same as a low-pass filter with a cut-off at $1/2T$. Thus the design proceeds in exactly the same manner except that the coefficients are given by:

$$\left(\frac{\sin\pi(n-\alpha)}{\pi\,(n-\alpha)} - \frac{\sin 2\pi FT\,(n-\alpha)}{\pi\,(n-\alpha)}\right) \frac{I_0[\,\beta\,(1-[\,(n-\alpha)/\alpha\,]^2\,)^{1/2}\,]}{I_0(\beta)} \qquad 0 < n < M$$

To cut a swift calculation short here is the coefficient plot and the impulse response:



Coefficients of Hi-Pass Filter

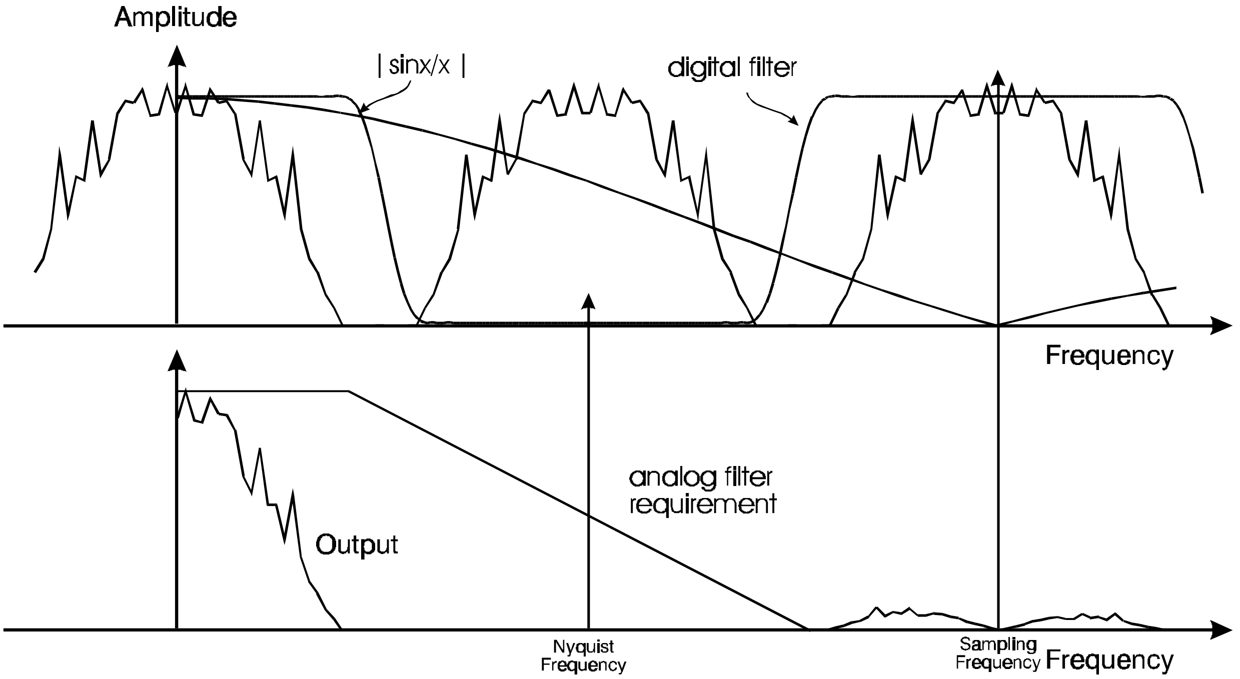


Log Response of Hi-Pass Filter

and the coefficients are:

Term	**24-bit Precision**
0	0.00000000
1	-0.00187761
2	0.00080466
3	-0.00148916
4	0.01412050
5	-0.02132800
6	0.00526148
7	-0.00730395
8	0.05757900
9	-0.07639660
10	0.01715700
11	-0.02227500
12	0.16818300
13	-0.21901500
14	0.04972400
15	-0.06806450
16	0.58182300
17	-1.00000000
18	0.52266200

One final note is the "roll-off" at the high frequency end of the hi-pass response. This is fundamentally due to the fact that there is an even number of coefficients in the convolution. If you look back in the notes, you will see that I said that hi-pass filters should have an odd number of coefficients.



DAC Using Interpolation and a Digital Filter

Now that we have seen an example of a filter design here is an application - the interpolation problem we were looking at earlier. You will remember that when we resampled the series with the zeros we moved the Nyquist frequency up but left the spectrum unchanged. In order to improve the "cleanliness" of the output and give any analog filters on the output of the DAC an easier time (ie easier design criteria) we need to get rid of the stuff around the Nyquist freqency. Our lo-pass digital filter designed above is just the job since it passes frequencies up to 1/2 the Nyquist and rejects betwween 1/2 and 3/2 the Nyquist - however it "turns on" again then and continues up to the sampling frequency. Now when the output is sent through the DAC there will be the required spectrum, nothing around Nyquist and something around the sampling freqency, but that will also be somewhat attenuated by the sinc*x* function of the DAC. A relatively modest analog filter will be able to pass the baseband and remove the higher frequencies.

We have therefore made real progress in moving the problem of filtering from the analog domain to the digital domain.

Other Filter Forms and Other Considerations

The major problem with the above filter functions is that they do not allow an impulse function which decays to zero smoothly - there must be some cut-off at a finite number of terms. A modification of the filter function which allows the impulse function to decay to zero smoothly is:

$$y[n] = \sum_{m=0}^{\infty} a[m]\ x[n-m] + \sum_{m=0}^{\infty} b[m]\ y[n-m]$$

It is apparent that the filters discussed above are simply those for which $b_m = 0$ - they are Finite-Impulse Response (FIR) filters. The more general case is the Infinite Impulse Response (IIR) filter for which both sets of coefficients are non-zero. The use of recursion in the filter function gives the property of slow decay to zero, lessens the number of terms required in implementation of some filter functions and renders the link to the transfer function more obscure!

An obvious problem is that the definition of an analog filter is (comparatively) simple - define h(t), but the translation into a set of coefficients, particularly for the case of the recursive filter is not simple. There are three general techniques for doing it which I shall mention only briefly:

- *Match the Impulse Response* as discussed above

- *Map the Poles and Zeros*. Essentially this involves expanding the impulse function in both cases (analogue and digital) into a ratio of polynomials and then matching the roots of the polynomials in the denominator (poles) and numerator (zeros). However this has to be done in complex space.

- *Bilinear Transformation*. Use a transformation between the analogue and digital descriptions which happens under reasonable conditions to map one function into the other fairly well.

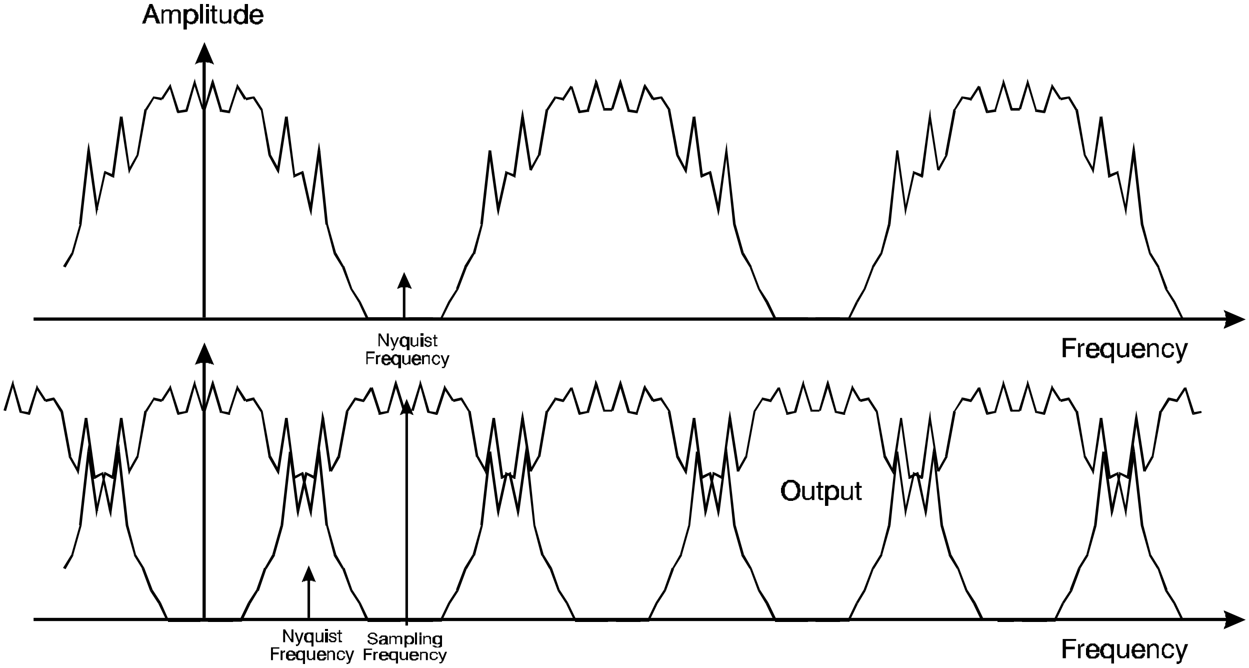
You will note that there is no "magic solution" for this problem.

Another technique for filtering a time series is to perform a fourier transform on the series to get a frequency series, multiply the frequency series by the filter function in frequency space and then transform back to the time series again. Although processing speeds for fourier transforms have speeded up considerably recently this is not a high-speed techniques and is more suited to "off-line" problems than "real-time" filtering. However one would have said that about the whole subject of digital filtering 10 years ago.

Decimation

If interpolation is a technique which is discussed primarily when considering DAC systems, then decimation - the removal of samples from a series to lower the effective sampling rate - is linked to the concepts of an ADC.

ADCs are of two main type for this discussion: *spot value* and *averaging*. We will first consider the spot value type.

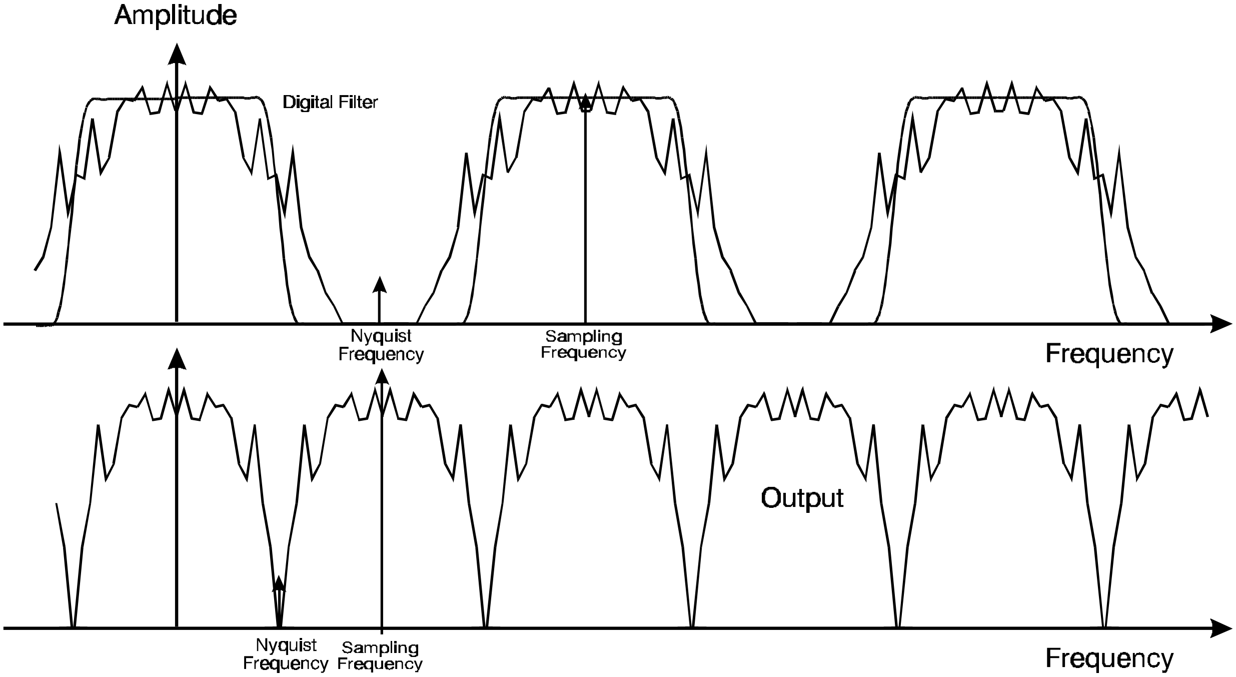


Result of Decimation - Aliasing is evident in this Case

We already understand quite clearly what is needed in order to make the system operate properly: In order to avoid aliasing we need to ensure that the input to the converter is band-limited to less than the Nyquist frequency before the digitiser. For a 12-bit digitiser this implies a response that is 1/4096 or about -72dB down at the Nyquist frequency. However we wish to look at all frequencies up to the Nyquist frequency. We therefore need a very sharp cut in the filter to ensure these two conditions are met. However sharp analog cut-off filters are very difficult to construct.

An obvious way to fix this is to increase the sampling rate that raises the Nyquist frequency and gives more room for the analog filter to cut-off the alised frequencies after the required passband - but now the data rate is rising. The solution is to use a high digitiser rate and then to "decimate" the output. Decimation removes every "nth" sample and therefore reduces the data rate.

Decimation by a factor of two is equivalent to multiplying the time series by an array of delta functions spaced $2T$ apart which in turn is equivalent to convolving the frequency spectrum with the fourier transform of the array of delta functions. We know that the transform of an array of delta functions spaced $2T$ apart is another array of delta functions spaced $1/2T$ apart. So applying the rule of multiplication in the time domain equals convolution in the frequency domain we find that the result of decimation by two is to halve the Nyquist and sampling frequency and to modify the frequency spectrum to repeat about the new sampling frequency.



Aliasing is Eliminated by the Digital Filter

So it is important, if we are to decimate the series successfully that the frequency spectrum be limited to half of the original (higher) Nyquist frequency so that it not extend beyond the Nyquist frequency at the lower sampling rate. Sounds like a job for the digital filter discussed above again![32]

So a successful system will analog filter the input to preserve the required frequency spectrum (below the final Nyquist frequency) and reject everything above the initial Nyquist frequency, then digitally filter the time series which will limit the bandwidth to one half of the initial Nyquist frequency, and finally decimate the series. By this means we can put the severe filtering requirement in the digital section and the sloppier requirement in the analog section.

[32] Actually, since the filter is not a strict "brick-wall" filter it won't do as it does pass frequencies above the Nyquist to some extent. However it does well to illustrate the idea.