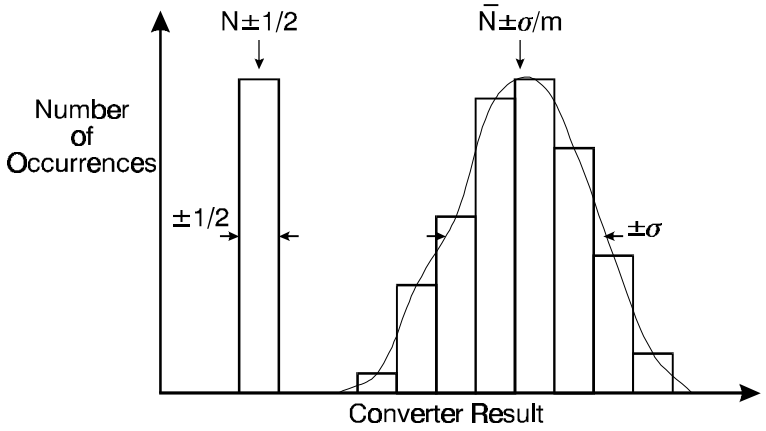
Averaging - When and When not to

Let us consider the problem of averaging the result of an a/d conversion. One does this quite frequently for two main reasons: Firstly to average the result in time and secondly to improve the resolution.

Consider the following argument: If we have an ideal converter and we average a steady voltage then we always get the same result which is within $\pm 1/2$ LSB of the exact answer (leaving aside questions of calibration). Thus averaging the result in this case produces no increase in resolution or accuracy because the result is always the same.



The Averaging Paradox

However suppose we are measuring a signal which consists of a constant voltage and a gaussian deviation from the constant which is much wider than the bit spacing, then we shall get a result which consists of a histogram of frequency of occurence of a given result as shown here and we can apply statistics to the histogram and various formulae assuming the gaussian shape. The result is that with a larger number of readings we can define the gaussian very precisely and therefore locate its peak to much better than the bit spacing, i.e. averaging has in this case increased our resolution.

The conclusion of the above paragraphs is that **with an ideal converter the average value of a noisy signal may be found to better accuracy than a quiet one!!!**

Like all theoretical results this one is theoretical and takes no account of the real errors that occur in the real world and (un)fortunately this affects the conclusions. In the ideal world we are creating a histogram of readings in which all the histogram bins are of a known width and equal to one another (ie an ideal converter has a DNL of 0LSB). The second condition is not important so long as the bin divisions are known, which means knowing the transition points of the ADC accurately. Since the number of readings gives the area of the histogram element, knowledge of the width is required to ascertain the height. One should note at this point that some converters have a distinctly variable distribution of

bin widths, typified by the successive approximation converter where the circuit can significantly change between successive states, e.g. between 7F(h) and 80(h), whereas others such as the dual-slope converter have inherently equal bin widths (equal interval property). In the following argument we concentrate on the successive approximation, or "random width" types since these are common.

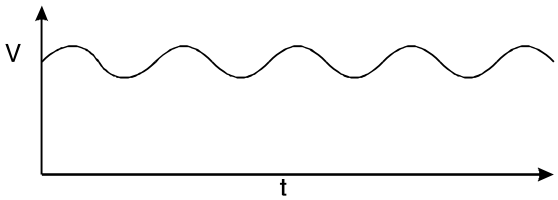
If we take an ADC whose DNL is specified as $\pm 1/2$LSB all we are saying is that the converter bin widths are between 0 and 2 LSB in size and we can say nothing about the histogram construction unless we premeasure the real positions. Thus unless you know the converter properties to the resolution that you are trying to achieve, you cannot use this averaging process on a successive-approximation converter. However you can make a nearly ideal converter by taking a longer converter and taking only the most-significant bits from it. For example if you take a 12-bit converter at ± 1LSB and only use 8 bits of the output, the accuracy becomes $\pm(1/2 + 1/16)$LSB or thereabouts and results may be averaged to get 12-bit resolution (or thereabouts).

In summary then, because some converters have "preferred states", they may not, in general, be averaged to improve the resolution beyond the initial resolution. However the integrating converters do not have any predisposition to individual states (DNL = 0LSB[26]), but only possess an INL so that they may be averaged to the DNL limit which may be very good. The essential justification for such a procedure being that the bins in the histogram are all of equal, if slightly unknown, width and therefore the statistics will work.

The above discussion does not imply that any particular converter may or may not be averaged but it does indicate the particular bias of the group. In general the integrating converters tend to be averagable whereas the successive-approximation kind tend to be non-averagable.

Averaging Changing Signals

There are however some more general problems associated with averaging which can be seen from the following argument: Suppose we have a d.c. level on which is superimposed an a.c. sinusoid and we wish to derive the average value of this waveform by taking a



DC plus AC Signal

[26] Of course there is some DNL - it's just not simple to determine.

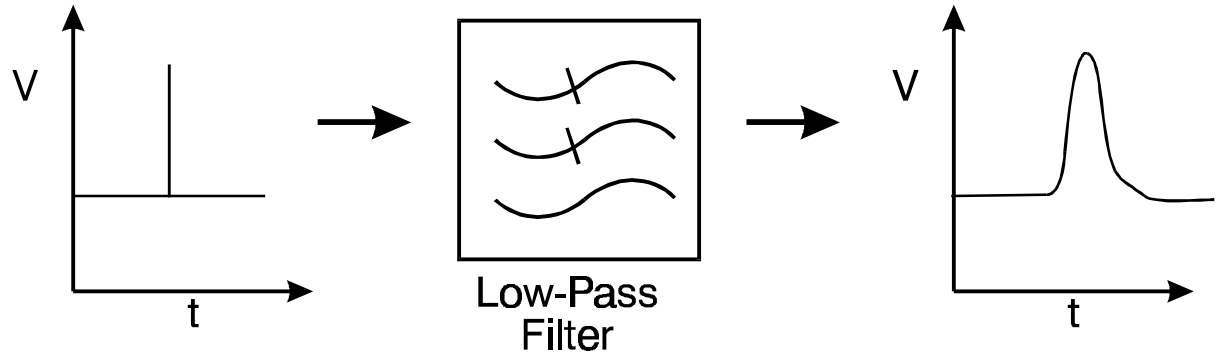
number of "spot" readings. We would assume that if we took 100 readings then we might get a very accurate answer, but the reality depends upon two things (assuming that we do not know the waveform shape beforehand):

- We must sample slower than the rate of fluctuation

- We must not sample synchronously with the waveform

The first argument is one of "independence". If all our samples are on the same "hump" of the waveform we will find a local average but not a general average. The second requirement is very tricky if you do not know the form of the waveform.

Now extend the argument to a waveform with random noise being filtered (to eliminate fast slew rates and therefore erroneous readings perhaps) before sampling. Our elementary noise theory says that (ignoring our other resolution arguments) the noise goes down as $1/(\#\text{ of readings})^{1/2}$ BUT this only applies if the readings are independent. We can see what trouble this causes if we imagine a sudden spike in the input.



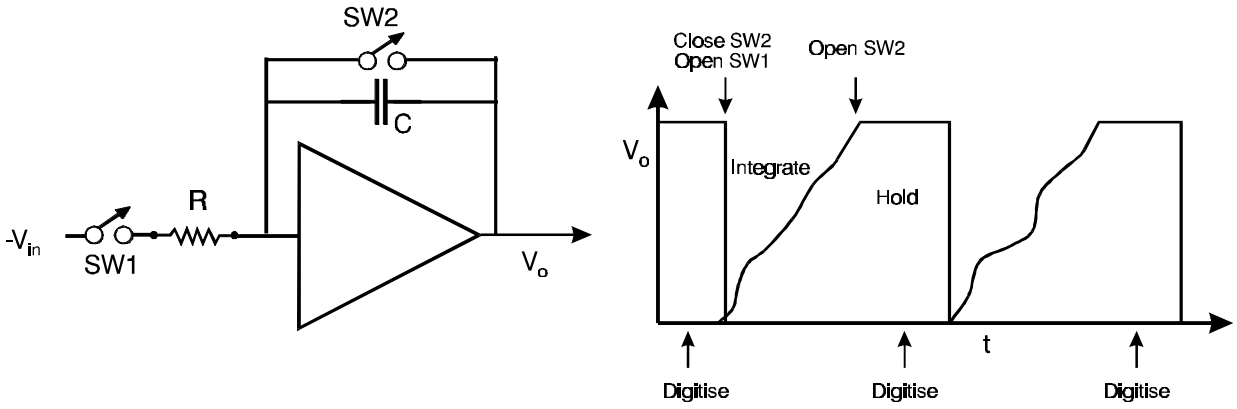
Independence and Spikes

The spike is "remembered" by the input for a time period and therefore the samples are not independent unless the sampling interval is much greater than the impulse decay time of the circuit. This may be roughly expressed as the sampling frequency being less than the bandwidth of the system.

This causes further problem with averaging as we are in danger of missing information. In order to ensure that our readings are independent we sacrifice information by sampling at a slow rate. We must therefore be careful to balance the bandwidth of the

electronics with the data rate we require. Too low a bandwidth reduces the independence of the samples - a "memory effect" - too high a bandwidth can introduce noise into the system.

Now to get the most information out of the circuit we need to take continuous averages which implies that we need either several dual-slope converters in tandem - except that we have to be careful with these devices that we ensure that they neither overlap nor have big gaps - or we can use V-F converters and count them in various time periods to get our various samples. In the latter case we can get true total time averages if we take care with our counting systems.



Independent Readings - Integrator + A/D

There is one other way to get true averaging which is useful because it produces well-defined, independent readings: We use an integrator ahead of a successive approximation converter. At the end of every time period we:

1) hold the integration (open SW1)
2) sample the output
3) reset the integrator(close SW2).

If we do this fast enough we will take negligible time out of our total average. Note again that the bandwidth of the circuitry ahead of the converter must be much greater than 1/integration time for the circuitry to contribute negligible interdependence to the readings. Notice also that the ADC in this example is only asked to digitise a constant signal.

An Introduction to Signal Processing

As we have seen the manner of collection of data often presupposes that we have a fair idea of what we are going to do with it and therefore the problems of data taking and data processing cannot, in general, be separated. Of course there are those times when data taken in one manner suddenly has to be processed in another but these are hopefully the exception rather than the rule.

Thus data that are going to be highly averaged should be taken by systems which are capable of such averaging and data required for strict time samples should not be averaged and so on, but these are just the start of some very complex data processing problems and we shall only have time to look at a very few ideas.

Let us first examine the problem of digital filtering of data since we have a fair idea of what that is all about - the "averaging" (note the word is now not used in a strict mathematical sense but in the sense of smoothing out the noise whilst retaining the signal).

Averaging Data

The simplest - although not the easiest - problem to start with is a constant signal with some random noise coming from an averagable converter. The formulae for the average and standard deviation are burned into every calculator with the same problem. If we use the usual expansions for the standard deviation:

$$\overline{X} = \frac{1}{N}\sum X_i$$

$$\sigma = \left[\frac{\sum X_i^2 - N(\overline{X})^2}{N-1}\right]^{1/2}$$

we can rapidly get into trouble with precision. Thus consider a set of numbers which should average to 1000 ± 1. When squared the individual terms of the sum in the standard deviation calculation is of order 1000000 and if we are using a large number of readings we might expect the numerator to be of order something like (10^6**N*) - (*N**10^6) and this is on (or beyond) the resolving power (or dynamic range) of most machines[27] to deal with. Your

[27] If you look back you will see that a single precision real number only has a resolution of 24-bits in the mantissa which means if the maximum number you can represent is 1, the

PASCAL is an exception here trading an appalling floating-point speed for lots of numerical resolution. The solution is to tackle the problem in a number of different ways.

a) Use an estimate of the mean, X_g to reduce the dynamic range. If you are taking a series of means of a slowly varying quantity then the last value you got might be a suitable guess. The formulae then become:

$$\overline{X} - X_g = \frac{1}{N}\sum \left(X_i - X_g\right)$$

$$\sigma = \left[\frac{\sum \left(X_i - X_g\right)^2 - N\left(\overline{X} - X_g\right)^2}{N-1}\right]^{1/2}$$

and the dynamic range is greatly reduced but the answer is still exact.

b) Store all the numbers, compute the mean first and then use the original formulae rather than the "quick" ones - this is a solution that you can't use in real-time as you can't start standard deviation calculations until all the dataset is in.

c) Use more dynamic range (extended precision) - very rarely necessary. Extended precision usage is either done after a lot of thought or none at all - I leave you to work out which is which.

The blind use of averages is in general a bad idea as one bad reading upsets the whole set and cannot be detected after processing. This leads us to the problem of "despiking" data which is generally good, but has a few bad points in it.

"**Despiking Data**"

The easiest way to despike data is to look at it - but nobody ever wants to do that these days so we seek a computerised method of despiking.

There isn't a perfect automatic method of despiking because despiking involves removing signals that you don't like and unless you can specify some "objective" criterion

---

smallest is $2^{-24} \approx 10^{-7}$. Thus two numbers which have a normalised difference of less than 10^{-7} will give an answer of zero (or nonsense) on subtraction

on which to base your rejection, the computer can't do it. However there are several useful ideas, the best of which is to store the data, calculate the mean and standard deviation and then go through the data throwing out points whose deviations are larger than some criterion - say 3x the standard deviation. The mean and standard deviation are then recalculated. The process could be repeated until no more points are removed, although in any civilised dataset one pass is generally enough. Some playing with the criterion may be necessary to get the required result but this technique is often good enough to remove the worst excesses of a "mostly good" dataset. Generally it is also a good idea to inform the user forcibly if a large amount of data has been removed by the process.

Biased Samples

An additional point to recognise is that the data must not be restricted by the data taking equipment. By this we mean that data must not be constrained by the input range of the equipment. A concrete example to be avoided is a converter which cannot convert negative voltages trying to monitor a noisy signal of zero average value. Since all the values < 0 will be rendered as 0, the average will be wrong.

Averaging Signals - Running Means

A series of readings may be simply filtered by means of a "running mean" which is the average of the last *n* readings. This can easily be implemented "on-the-fly", particularly if $c = 2^m$. Thus in C:

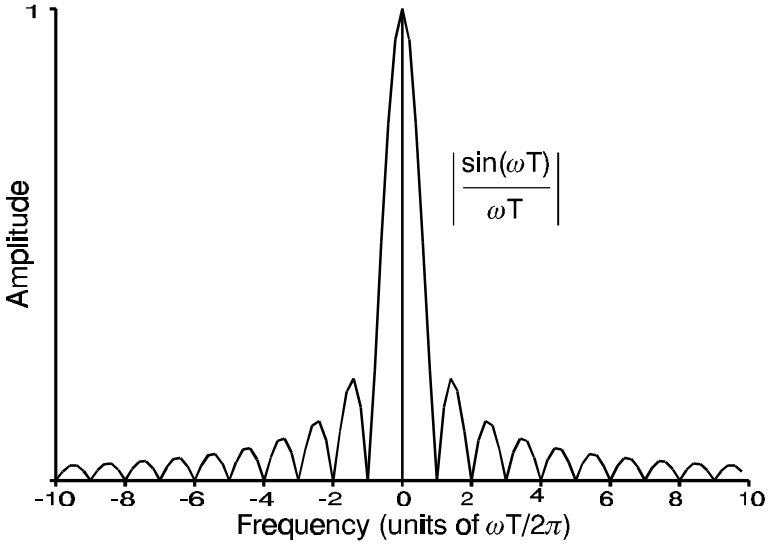
```
t = j = 0;                                  /* clear counters */
for ( i = 0; i < 16; i++ ) a[i] = 0;        /* clear array */
      FOREVER {
            i = *mem(IN);                   /* get a value */
            t = t + i - a[j];               /* add to total, subtract 16 ago */
            a[j] = i;                       /* update array */
            v = t>>4;                       /* calculate average */
            j = (j+1)&15;                   /* increment counter, wrap-round */
            }
```

takes readings and calculates a 16-point "running mean" (the average of the last 16 readings) of the results. There are obviously some problems at the start of the data and this is common to all time-series problems which have datasets of finite length.

This is "real-time" data processing in which the processing is required to keep up with the data taking. This places some restrictions on what can be done and also some restrictions on what should be done.

But what is a "running mean" really and what effect does it have on the various changing parts of the input? In order to answer that question we need to consider the problem in frequency space. The problem of the effect of a running mean on the waveform sin*(ωt)* can be written as:

$$y(t) = \frac{1}{T}\int_{-T}^{0} \sin\left(\omega(t+t')\right)\, dt'$$

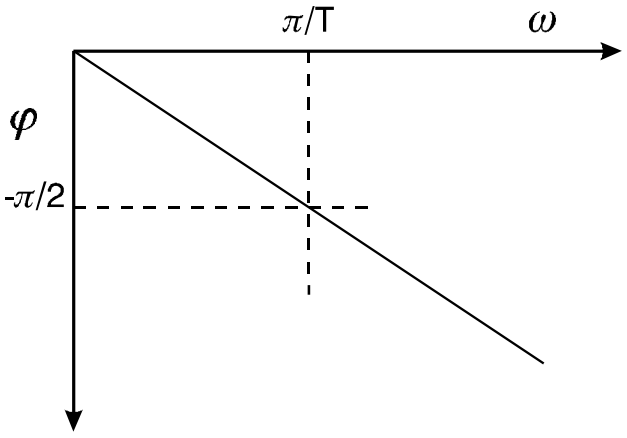


sin(x)/x - Amplitude

where we use an integral to simplify things for the moment. We can do the integral (I hope) and then collect up terms in the form of *A*sin*(ωt-ϕ)* as:

$$y(t) = \frac{\sin(\omega T/2)}{(\omega T/2)}\, \sin\left(\omega t - (\omega T/2)\right)$$

which says that the amplitude response has a number of maxima and minima arranged at intervals of $\omega T/2 = \pi$ or $f = 1/T$ and the phase response is characteristic of a time delay.



sin(x)/x) - Phase

Notice that this is really different from the usual forms for filter functions that you might get from analog filter design, but only in the fact that it is more complex. However it points out the fact that we must look very closely at the operation of any process on a series of readings whether done at the time of the data taking or later.