ELECTRONOTES
1016 Hanshaw Road
Ithaca, NY 14850

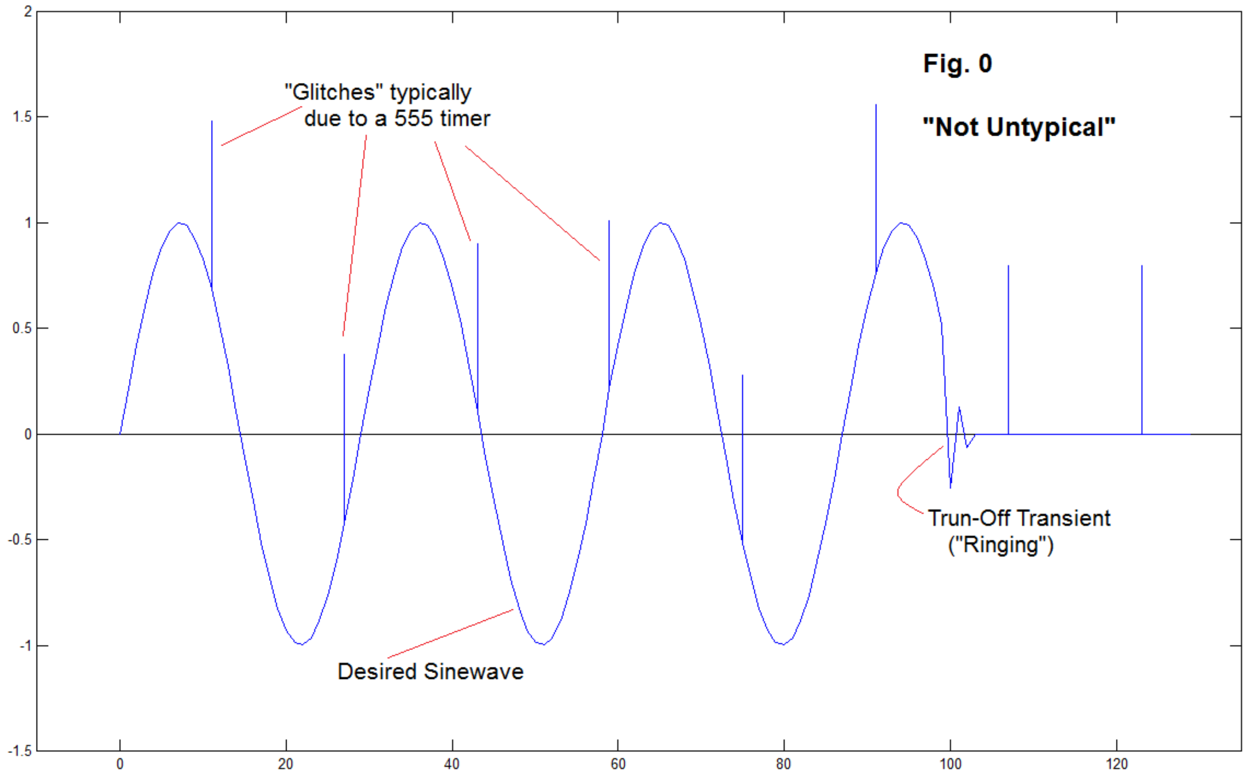
APPLICATION NOTE NO. 395

March 30, 2013

AVERAGING - AND ENDPOINT GARBAGE

PROLOG:

Longtime analog synthesizer designers and builders are accustomed to viewing waveforms on a scope and seeing, and ignoring, features of the trace which we know are not the main story. For example, if Fig. 0 were a scope trace, we might note as follows:



Well - it looks like a sinewave but I also see some sharp glitches, and I knew I should have added additional power supply bypass capacitors on that 555 timer – I will get to that. And, when the sinewave ended, something rang (perhaps just the scope) as the signal turned off.   But we likely don't looked at this as something we designed or wanted. It is a multiplicity of things, some of which just came along for free.

We look at signals in terms of what may have produced them, and here we not only know what we intended and what was unintended, but more importantly, sharp glitches and transient features are not produced by the same (basically low-pass) system as the one we are implementing. Physical systems we build, mechanical systems such as acoustic musical instruments, and the world itself tends to be low-frequency (smooth) until provoked by sharp input feature.

It is not as though we are surprised by much – the appearance of something unexpected (superimposed) is seldom a surprise in engineering. And most of the time, we no sooner see the unexpected than we know the cause, and often what if anything to do about it. Not that we don't love a good mystery as much as everyone else.

INTRODUCTION

Engineers tend to view any data presented in terms of how the numbers represented by the data might have been the result of something they are already familiar with. While it is true enough that someone could have just typed in whatever they pleased, lots of data series often do look familiar. The count of cookies in a cookie jar vs. time might look like the exponential decay of an RC circuit. When the jar is full, we take a handful. But when it is low, who wants to take a large fraction of those that remain? The two phenomenon, physically unrelated most likely, resemble each other.

Another example is when we see in a data sequence sharp features among smooth ones. That is, high frequency events, inside what otherwise seems to be low-pass processes (sharp spikes in smooth curves), are prima facie at least suspect. Further, engineers seeing strange happenings at the ends of data sequences always wonder if we are just seeing end transients.

There are many well-studied methods of extracting fundamental properties of a time series, such as DFT analysis, Prony's method, and principal components, to literally just name a few. Sometimes this is done as a matter of analysis: we want to identify parameters that describe a series (such as frequencies). At other times, we have as a first goal just getting a better "look" at the signal as it is represented by multiple, noisy observations.

Here we have in mind something like the identification of a time series representing historic global temperature (somewhat of a fictional entity in the first place) from proxy data. Possibly the first thing that comes to mind is simple averaging. Here we want to look at what problems can come up if we are not careful, and will use a stand-in "toy" for actual data. There is no real climate data used here.

EXPERIMENT 1 : HOLES IN DATA EFFECTING ORDINARY AVERAGE:

Let’s start with what we will take to be a length 13 time series that is exactly:

[0 1 2 3 4 3 2 1 0 1 2 3 4]

However we pretend that we don’t “know” this but only have some samples of these that are noisy. In particular, we have samples that are subject to random errors with a uniform distribution between -1 and +1. [In Matlab code, the added error is represented by 2*(rand-0.5).] We will also want to consider what happens when some samples are actually “missing”.

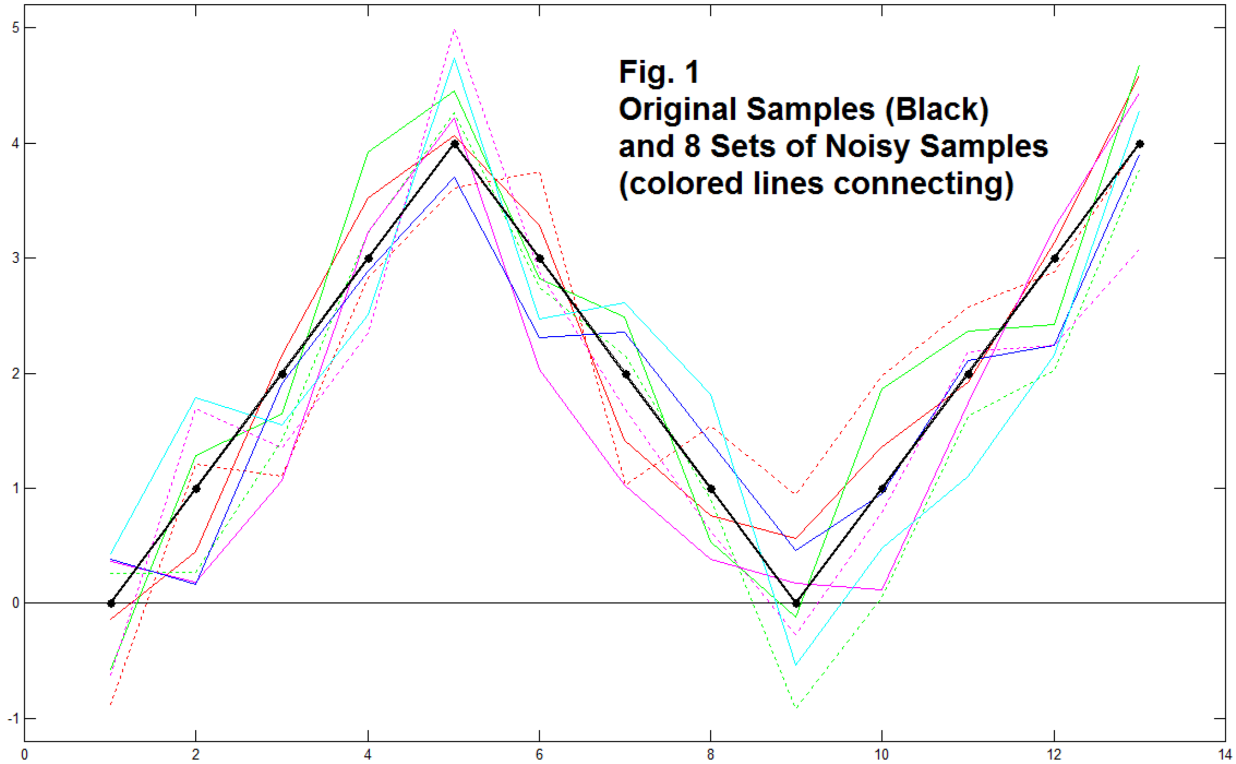


Fig. 1
Original Samples (Black)
and 8 Sets of Noisy Samples
(colored lines connecting)

Fig. 1 shows the 13 samples as black dots. The solid connecting line is just there for a better indication of the form here. The 8 colored curves are the noisy data sequences generated randomly. Here we are again just showing the connecting curves for better clarity. Fig. 1 contains only 13 original samples and (8 x 13 = 104) random samples. As we naturally would try with noisy data, we can average the eight noisy sequences, and this is shown by the blue dots in Fig. 2. Again we have connected the blue dots with a continuous line for clarity. We compare the continuous blue line with the continuous black line. We see that averaging has done a credible job of identifying the actual signal.

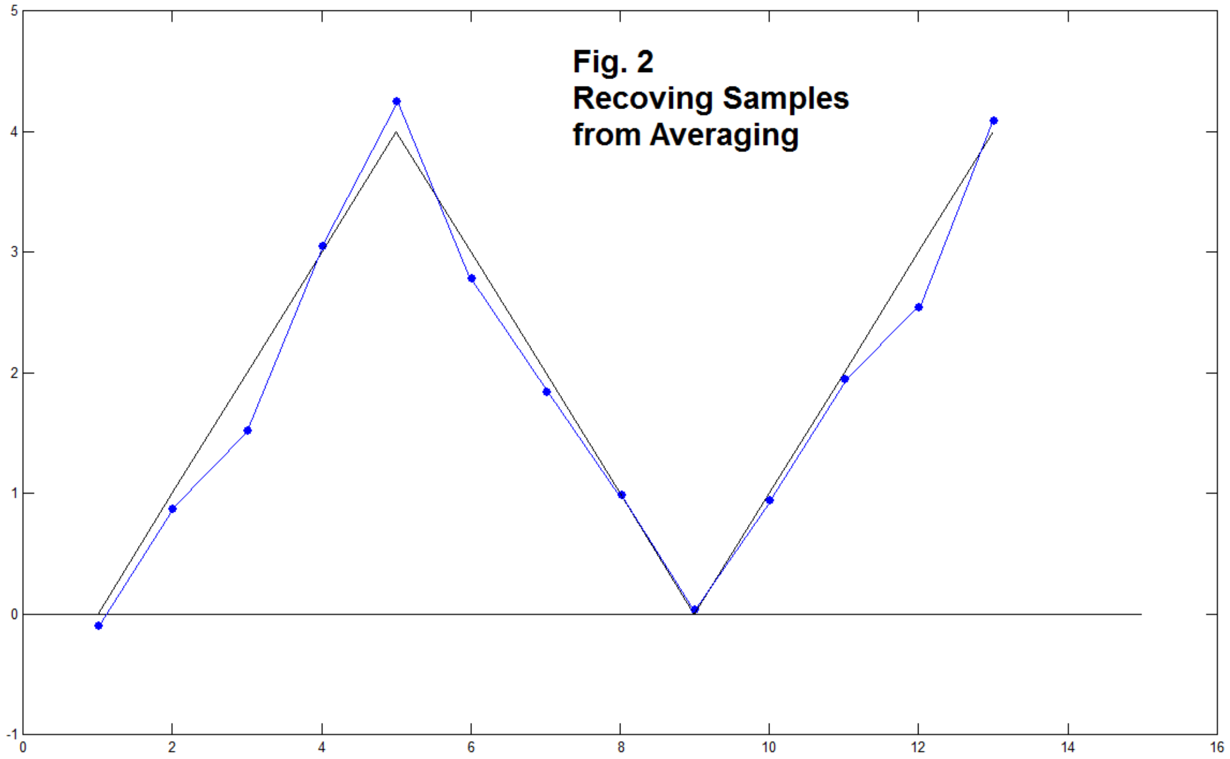


Fig. 2
Recoving Samples
from Averaging

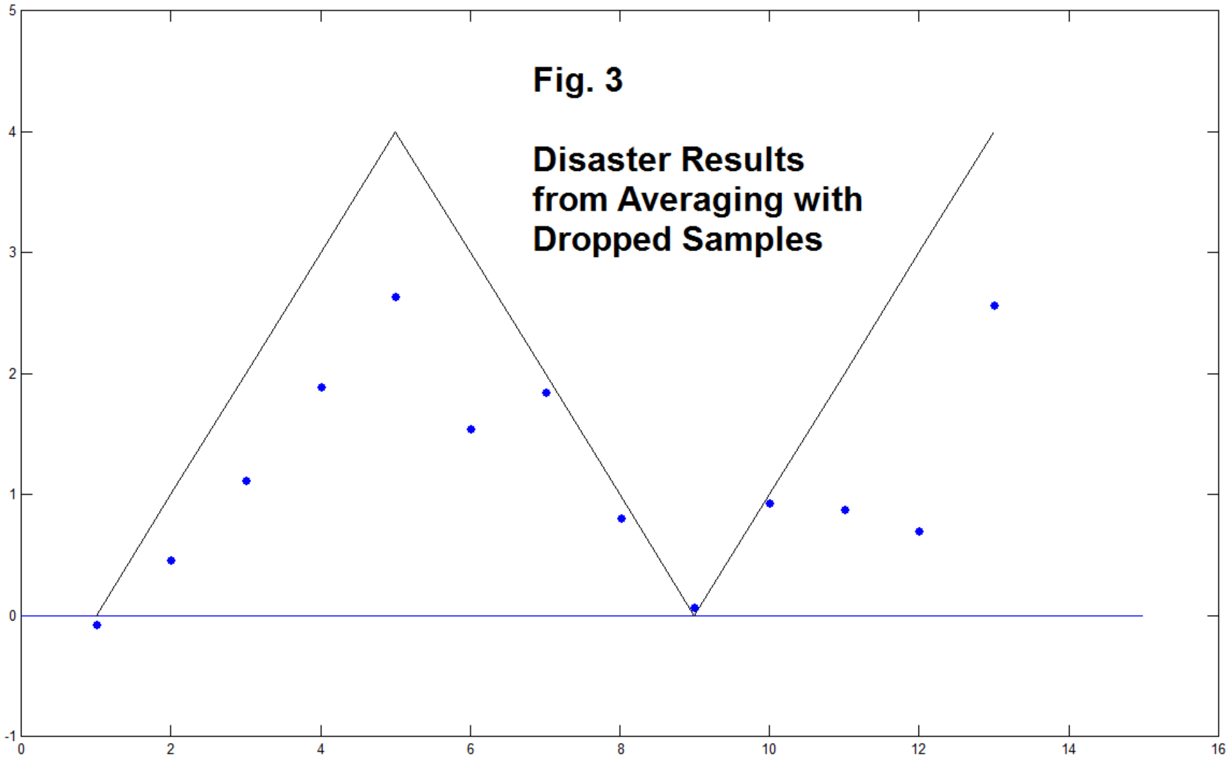
Next we consider the case where a fair number of the time series are incomplete. That is, for some elements of the series the data point is missing. It seems natural for us to represent missing samples with the number zero, but this is wrong as we remind the reader below. Our habit of putting in zeros perhaps comes from the correct notion of “zero-padding” a sequence prior to interpolation with an FIR filtering procedure. Here we will represent out failures to have a proper sample with the number zero in a “mask”. The number 1 represents a proper sampling instance. Our first mask is shown below:

```
m=[1 1 1 1 1 1 1 1 1 1 1 1 1 ;
   1 1 1 1 1 1 1 1 1 1 0 0 0 ;
   0 0 1 1 1 1 1 1 1 1 1 1 1 ;
   1 0 1 0 1 0 1 0 1 0 1 0 1 ;
   1 1 1 0 0 0 1 1 1 0 0 0 1 ;
   0 0 0 0 0 0 1 1 1 1 0 0 0 ;
   1 1 1 1 1 1 1 1 1 1 1 0 0 ;
   1 0 0 1 0 0 1 0 0 1 0 0 1 ]
```

Mask 1

**A Variety of
Sample Drops**

Mask 1 is a matrix (used for convenience) that is 8 rows by 13 columns, eight “signals” of length 13. Full sampling would be a 8 by 13 matrix of all ones. Here some samplings end to soon, start too late, or have a lower sampling rate or bunching of samples, as shown. In our experiments here, we will be multiplying the mask matrix by the data matrix (as plotted in Fig. 1). This will mean that samples that were not taken will be replaced by zeros. This is wrong in fact! However, here we can (for purposes of programming) use the fact that a result is zero to determine that the sample was skipped. This is because the possibility that an original sample is (exactly) zero is vanishingly small because it has a random number of many decimal places added to it. So what happens in this case?



We see in Fig. 3 that the 13 blue dots do not approximate the black curve well at all – generally being too small. Note in particular that points 6 and 12 are way too low. Points 7, 8, and 10 are not so bad. We do note that points 1 and 9 are not bad, but they are approximating zero anyway. So what is the difference. Consider the number of actual samples for points 6 and 12. The columns of Mask 1 sum to just 4 and 2, respectively, for these two points. The columns for points 7, 8, and 10 sum to 8, 6, and 6 respectively – higher numbers of actual samples. Note that the result for sample 7 is the same for both Fig. 2 and Fig. 3 (column 7 of Mask 1 is all ones). All of this is no surprise. For Fig. 2 and Fig. 3 we are finding the averages by summing and dividing by 8. If we have fewer than 8 actual samples at a particular position in the time series, we should divide not by 8, but by the actual number of actual samples at that position.

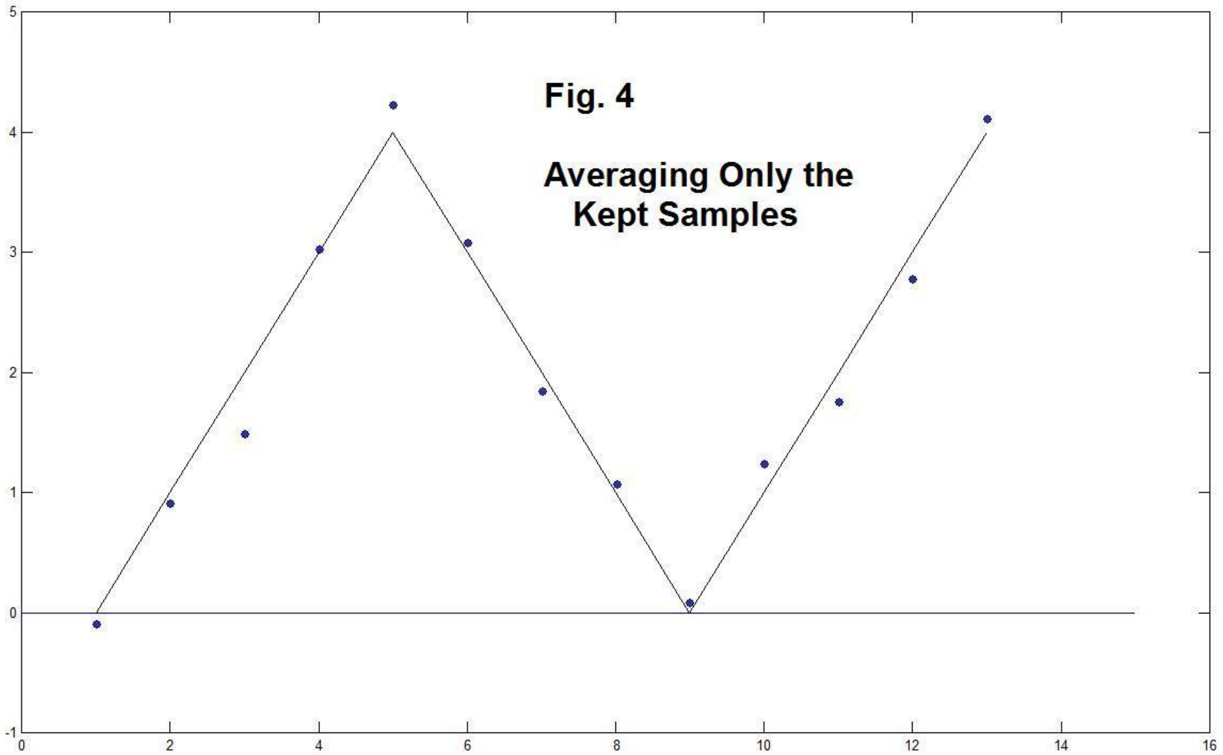
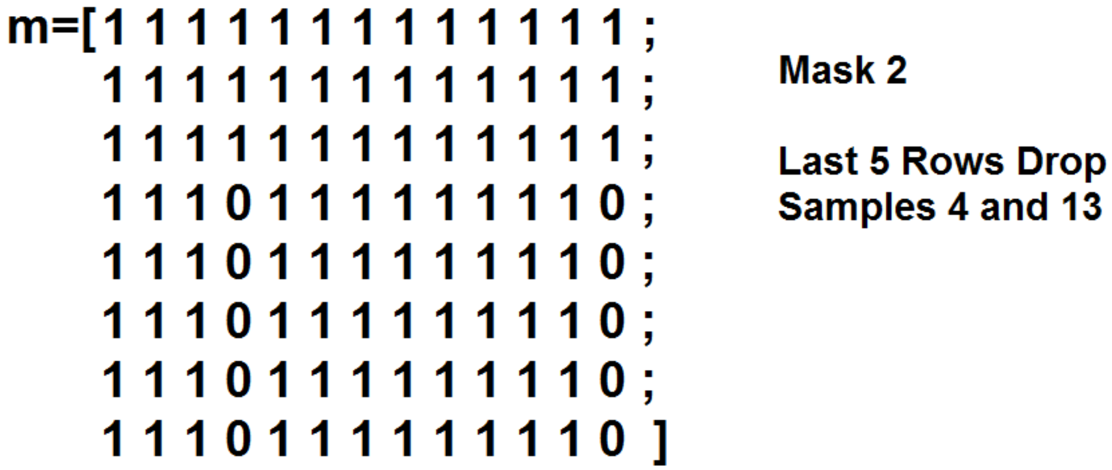


Fig. 4

Averaging Only the Kept Samples

When we do divide by the right number of samples, we get Fig. 4, which is clearly a great improvement over Fig. 3, and is not all that different (at least in this example), from Fig. 2, even though we have only 68 instead of 104 actual samples.


```
m=[1 1 1 1 1 1 1 1 1 1 1 1 1 ;
   1 1 1 1 1 1 1 1 1 1 1 1 1 ;
   1 1 1 1 1 1 1 1 1 1 1 1 1 ;
   1 1 1 0 1 1 1 1 1 1 1 1 0 ;
   1 1 1 0 1 1 1 1 1 1 1 1 0 ;
   1 1 1 0 1 1 1 1 1 1 1 1 0 ;
   1 1 1 0 1 1 1 1 1 1 1 1 0 ;
   1 1 1 0 1 1 1 1 1 1 1 1 0  ]
```

Mask 2

Last 5 Rows Drop Samples 4 and 13

Time series as irregular as those suggested by Mask 1 might seem unlikely. Possibly such situations occur when we try to reconciles a number of highly disparate measurements, such as temperature proxies. In such cases, we expect irregular endpoints times, drop-outs, interpolated (in-filled) points, and general disorder. Perhaps more typically we have time series that are simply of different lengths, perhaps of lengths differing by one or just a few samples. For example, smoothed sequences (produced by convolution typically) will lengthen sequences, and have end effects (transients). Here we will look at what happens with Mask 2. Our main objective was to shorten the last five sequences by one sample. At the same time, we could examine the effects of an earlier sample missing (such as an unconformity). Thus Mask 2 removes the 4th and 13th samples of the last five (of eight total) sequences. Fig. 5 shows the result where we get in the average, at time positions 4 and 13, values that are low because we have divided by 8 instead of by 3. Fig 6 shows the result of doing the same calculation while dividing by 3 in these two cases.

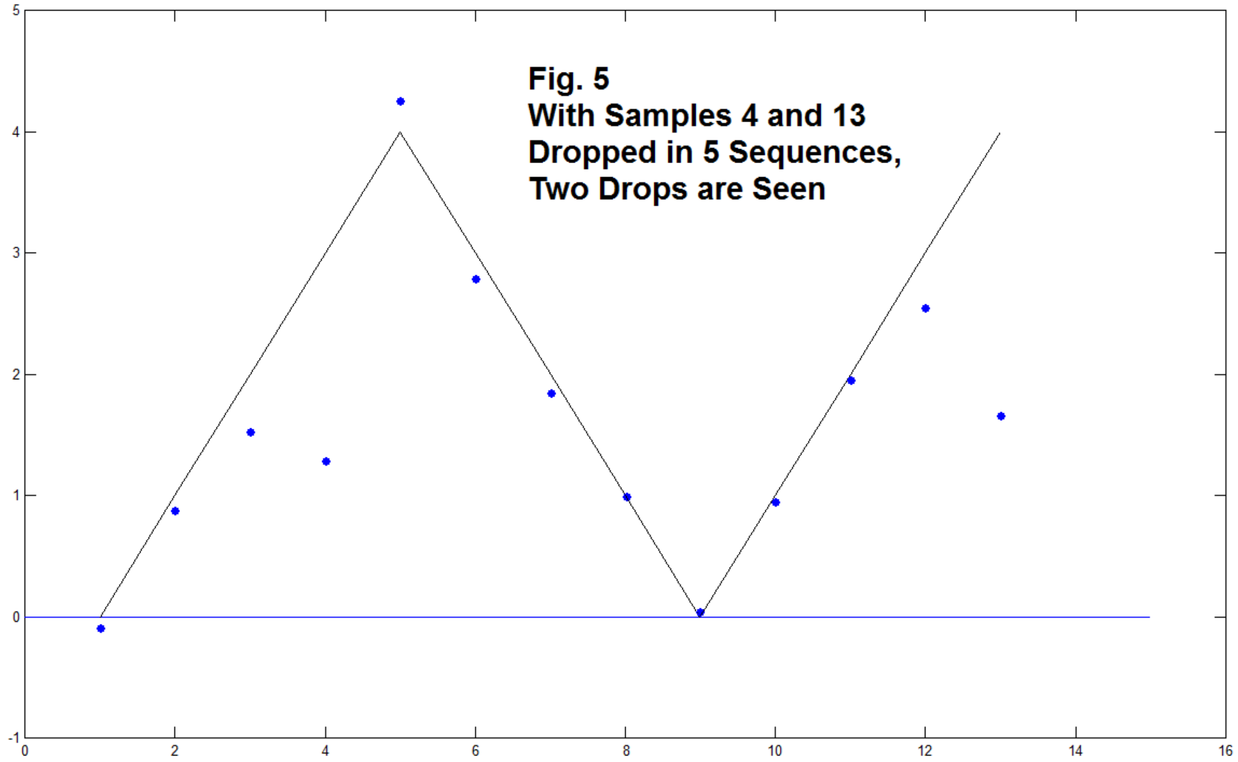


Fig. 5
With Samples 4 and 13
Dropped in 5 Sequences,
Two Drops are Seen

The program ***av.m*** at the end gives the code used for these six above figures.

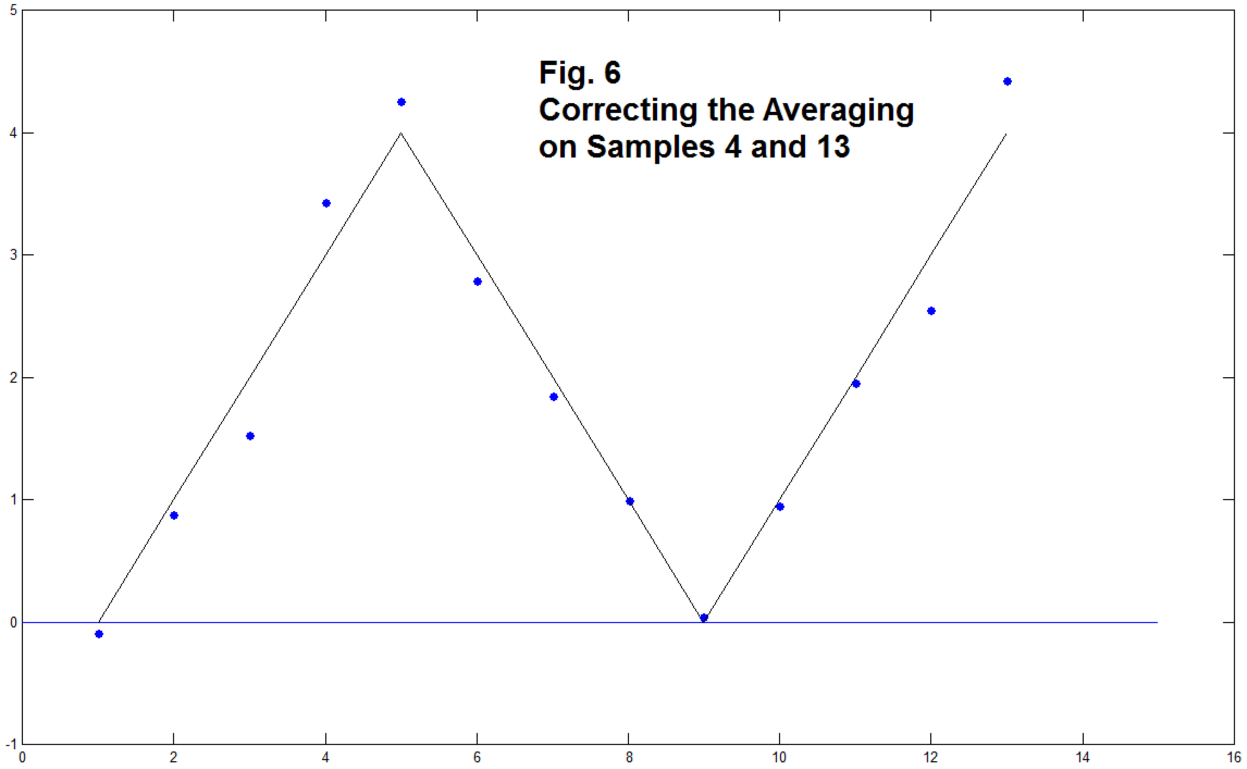


Fig. 6
Correcting the Averaging
on Samples 4 and 13

EXPERIMENT 2: AN EXTRA ENDPOINT

Consider the case where a low-frequency event is represented by many similar instances where the parameters of each instance may vary over a range. In particular, we envision a single broad event as represented by a single cycle of a cosine (a bump). We allow that the actual frequency, phase, amplitude, and “dc offset” may vary a bit. So we are not adding random noise to a target signal, as we did above, but jiggling four parameters randomly, while holding the parameters constant for any one signal. Fig. 7 shows such a collection of 20 such signals. The “prototype” cosine is shown as the light blue line in Fig. 9.

The results here are from the initial run of a program ***mar.m***. Other runs are similar but not identical of course. Each of the 20 signals has a central bump. The amplitudes vary about a value of 1 by ±25%. The frequencies vary about a value of 1/20 by ±20% (time ranges from -10 to +10). The initial phase shifts by ±10% of π. The constant offset varies by ±5. The 20 examples plotted are a modest representation. Note for example (by chance) the bottom two, one which is clearly a bit low in frequency (less than a full cycle) and the other a bit high in frequency (more than a full cycle.

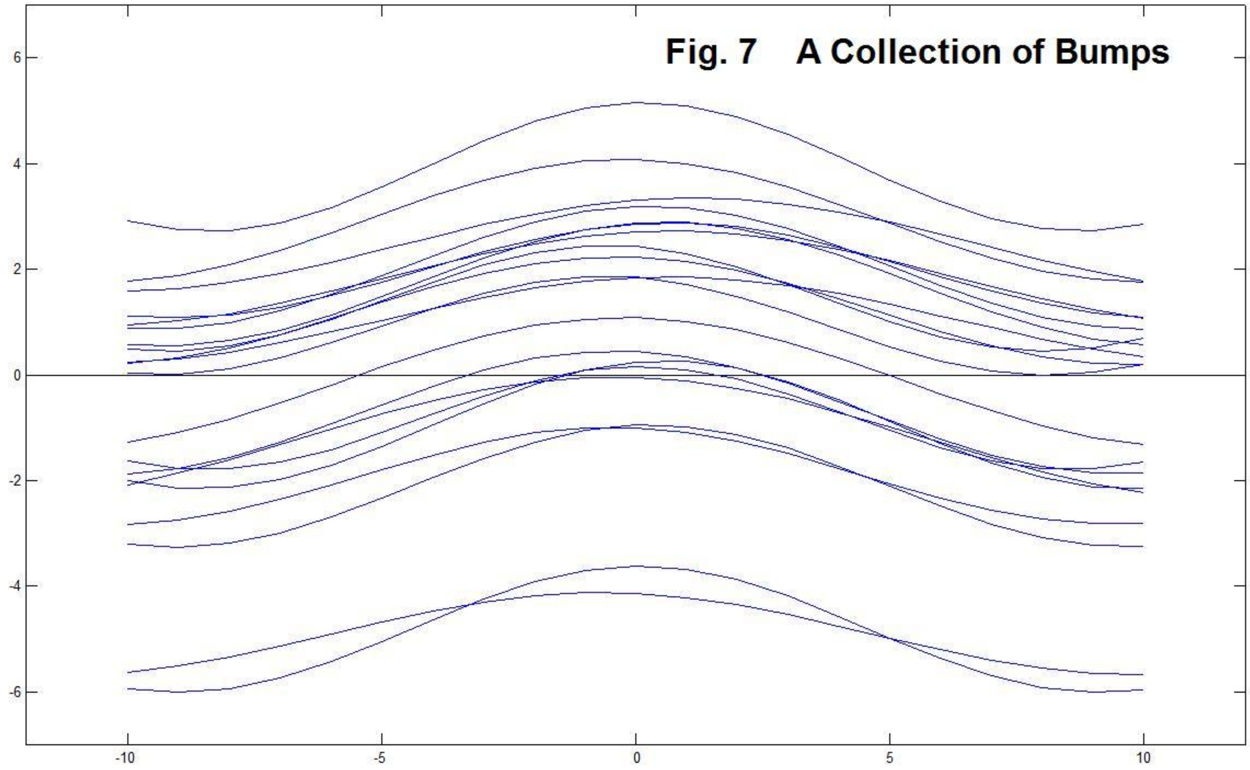


Fig. 7 A Collection of Bumps

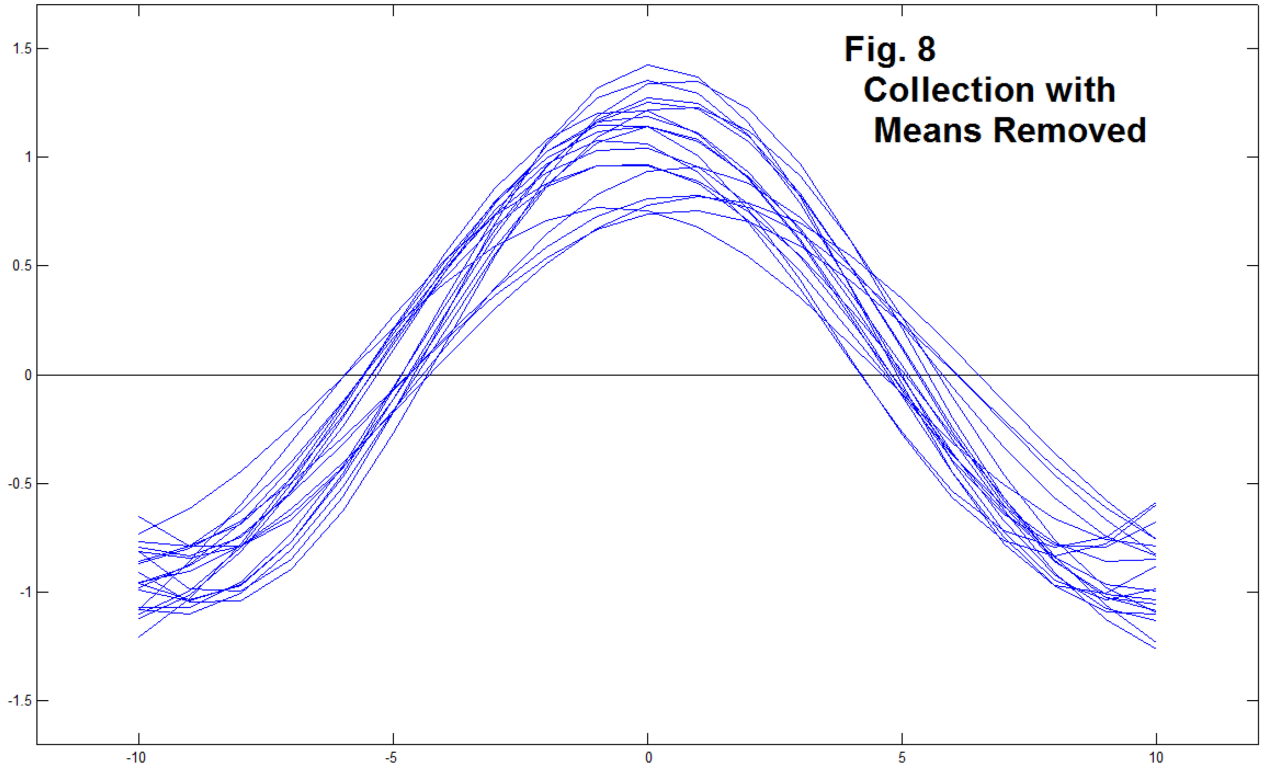


Fig. 8
Collection with
Means Removed

We are going to average these in an attempt to retrieve some essence, So while not essential, here we do show in Fig. 8 the same 20 signals with their means set to zero. Our purpose in the original offsets were to make it look more like global temperatures series taken at different latitudes. The "bump" here is basically thought of as a simple model for the Holocene (current interglacial going back 10,000 years). Accordingly, Fig. 8 is something like a plot of temperature anomalies over that time. Of course, the data would be derived from proxies, not thermometers. Again, we emphasize there is no actual climate data here. By definition of our coded formulas, the data is relatively symmetric about zero where there is a maximum. The slight asymmetry is due to our random starting phase. The ends are roughly an equal mix of curves going up and cures going down, as is evident in Fig. 2. So what happens when we average?

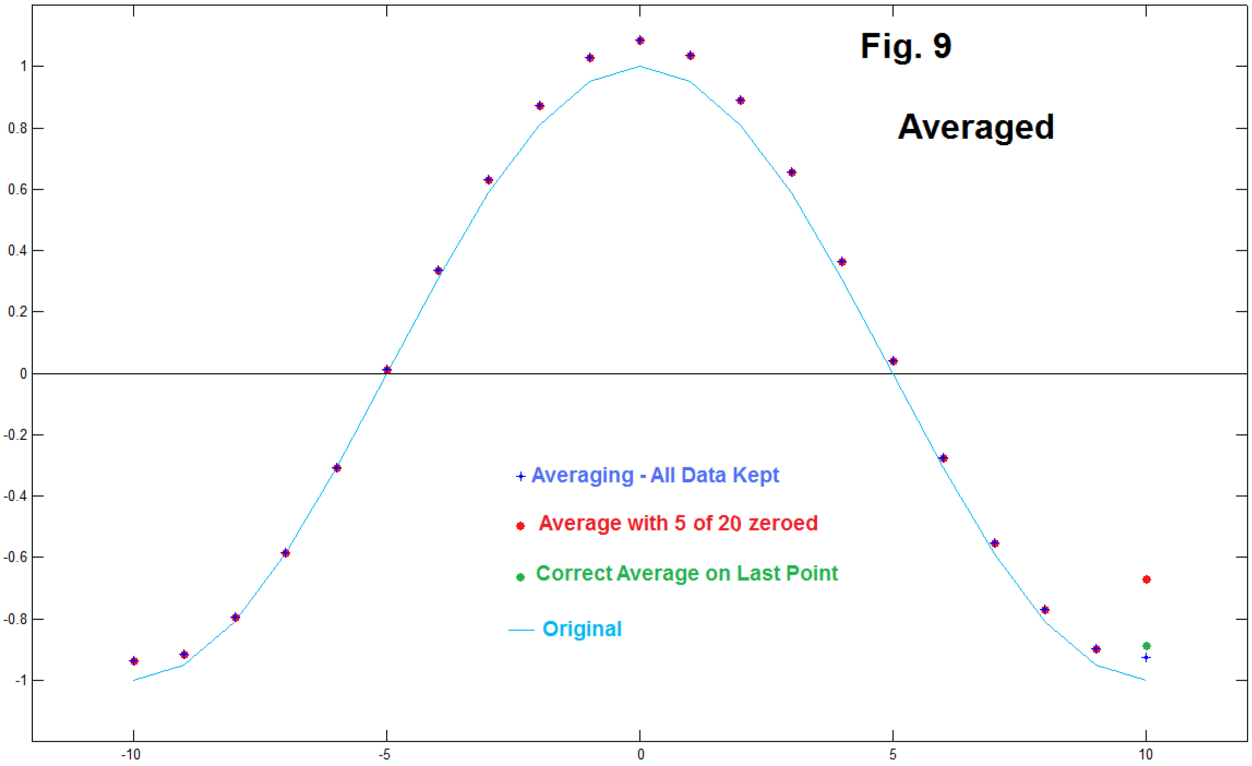


Fig. 9 shows the results. As mentioned, the light blue continuous curve can be thought of as the correct answer. All 20 of the time series are length 21 samples. We see, much as we did in the first experiment, that averaging works quite well. The actual average is represented by the dark blue stars.

There are no real surprises here. But the actual experiment we want to do here is a modification – what happens if we remove the end point (at +10) for some of the sequences. Here we are going to remove the end point from the last five sequences (any five would do). We then do the same averaging, and the results are the red dots, which are identical to the blue stars at every point except +10, where the average jumps upward toward the mean of 0. This is just what we expect. If we instead do things right, we obtain the average on the last point (at +10) by dividing by 15 instead of by 20, and this is shown by the green dot at time +10, instead of the red dot. The green dot is a much better estimate of the true situation, but not exactly the same as the blue star, because we did, after all, throw out five of the twenty samples at time +10. (Not untypically, on additional runs, the green dot is a near-perfect overlap of the blue star.) Additional runs are similar, and the red dot always jumps up at the end. After all, this is the result of an error which we characterize as a blunder.

MOVING AVERAGES – THEY MOVE (In Time) , AND CAN THE FILTER BECOME SHORTER?

Simple averages are not the same thing at all as "moving averages". An average is a single number, like the average of a student's test scores being 81 as a result of scoring 70, 90, 86, and 78 on four exams. But wait. Above we had averages that were not single numbers, but time series of numbers. None the less they were at each time point a single number, with averages for each time point computed in parallel.

A moving average is a traditional FIR digital filter. The weights of the impulse response are typically all the same (some sort of "window" may be used, but that it technically different of course). The average is not of samples from different signals at the same time point, but of a range of samples of the same signal at different time points. Thus it is inherently an average over time rather than an average over different measurements. After one such average is computed, typically we move along one time interval and find a new average, with each point constituting an element of an output time series.

Like all filters, we can and should worry about end transients as well as a steady state response. We showed above ways in which end effects could occur as the result of actual errors (blunders in fact) computing the ordinary averages at the end (or at other points as well). It may seem unlikely that one would makes such blunders. When dealing with moving averages however, it is possible to actually divide by the wrong number of points because we run off the end and the number of points added decreases from the full length of the filter down to one and eventually to zero.

What happens in this case. If we do continue to divide by the full length, the ends taper down. That is, our simple notion of having end points missing is more subtle – the ends are discounted gradually. On the other hand, if we correct for this by dividing by the actual number of samples, the moving average length still gets shorter, and the bandwidth increases accordingly (and dynamically, making the interpretation difficult). High frequency events which were previously being low-passed out (in the middle) are now being passed through. What to do? There is no answer. We are basically up against the uncertainty principle! In attempting to define a particular time (the end of the sequence) we are not allowed to also know the energy there (the value).

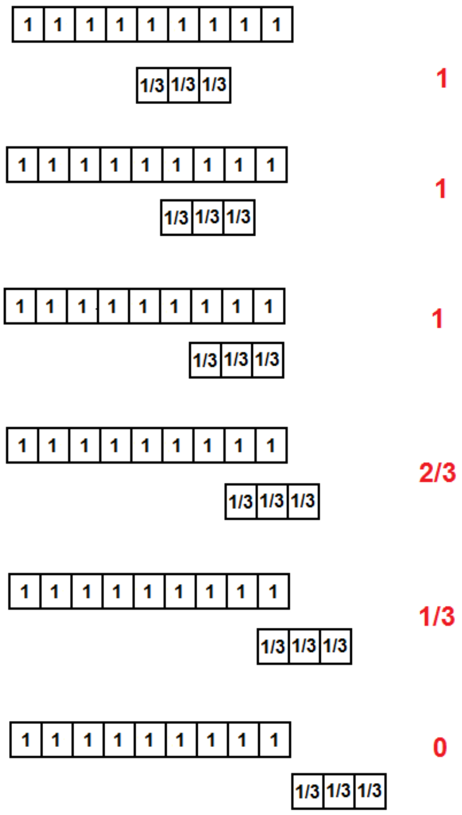


Fig. 10

Moving average as a FIR filter showing an end transient

The filter is linear-time-invariant

Fig. 10 shows the absolutely typical operation of a moving average. Here the signal being averaged is represented by a sequence of all ones, of which nine are shown. We assume this may be continued for a longer period of time on the left, but the right side is the end of the sequence <u>that is available</u>. It may well be the case that the signal also does continue as a sequence of ones on the right, perhaps viewed as future. This is just all we have, and we want to see the output of the moving average filter at the end. The impulse response of the length-three moving average FIR filter is [1/3 1/3 1/3]. When we apply this filter to the middle of the sequence, we get 1/3+1/3+1/3 = 1, the proper average.

It also seems clear that the last three values may not be what we intend. That is, perhaps the averages should remain at 1. It is easy for us to achieve these values by making the moving average time-variant, as in Fig. 11.

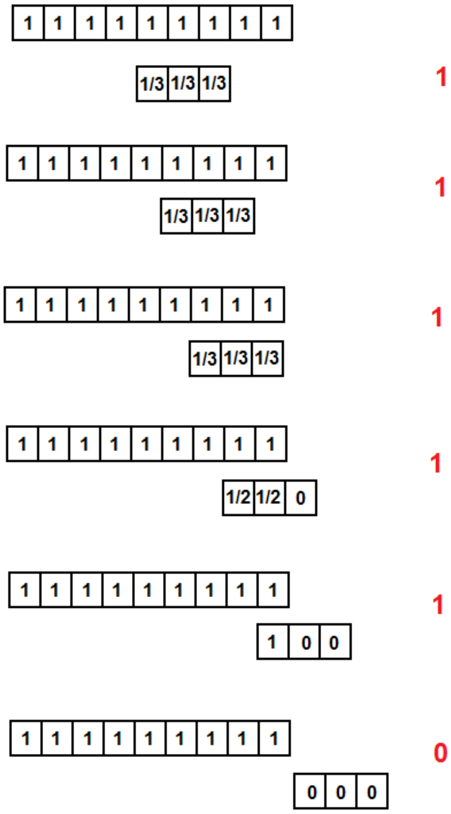


Fig. 11

Here we maintain what may be the "correct" value for the moving average until we are actually completely out of data. The filter is NOT time-invariant.

What we have done here is exactly what we suggested above of computing the average based on the actual number of valid data entries. We are assuming that we have good data to the very end. Thus by dividing by three in the middle cases, and then by 2 when we have only two points left, and eventually by 1 when here is only one point, we maintain the “correct” average to the end of the currently available data. This is not hard to program, but it is not a time-invariant filter (the tap weight change) and is not an ordinary convolution. On the positive side, it is probably exactly right, or the closest we can come to defeating the uncertainty principle.

As suggested, it is quite likely silly to try to average something that we postulated is a constant. So we might make an excuse that the data sequence might represent something like a slightly noisy temperature record. Or perhaps, the moving average is an attempt to “infill” (interpolate) missing data. In either case, since <u>recent</u> global temperature trends are of great interest, and since recent data records must terminate, <u>by definition</u>, with the present, we should be very conscious that end effects may be misleading.

Above we chose a length-three moving average as a matter of the smallest possible size that illustrated the points we wanted to show. A length 4 or length 20 might have been chosen. Often a specific length moving average is chosen specifically for its frequency domain consequences. A length N moving average has a zero in its frequency response that is at a frequency f_s/N, and integer multiples of this up to $f_s/2$. Thus the length N=3 moving average rejects $f_s/3$, and that's it (a familiar periodic sync frequency response magnitude). Anything with period 3 is rejected, while anything constant is passed.

Figures 12a, 12b, and 12c illustrate the end effect associated with this filter. We chose as input a constant plus a cosine of period three: specifically 1 + cos(2πn/3), which is a sequence of ….2, 1/2, 1/2, 2, 1/2, 1/2, ….. Note that any three consecutive samples sum to 3, or an average of 1, which is the chosen constant. So the average is always 1 <u>inside the signal, but there are some very different end transients</u>, as seen in the figures.

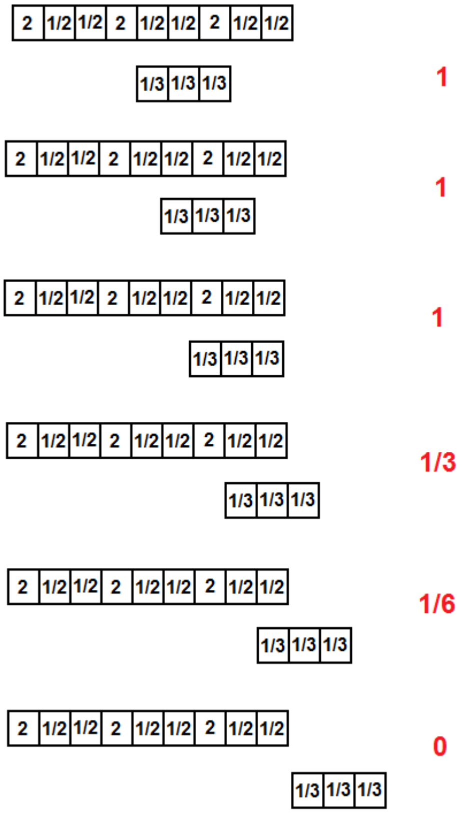


Fig. 12a

Here the periodic component is rejected and the constant passed - in the middle portion.

The end transient is a scaled version of the end transient with a constant signal (Fig. 10)

As the filter's impulse response [1/3 1/3 1/3] moves through the filter, as stated, we get an average of 1. As it walks off the end, assumed zeros beyond the end are included in the average, and as shown in Fig. 12a, since we have only two values, both of 1/2

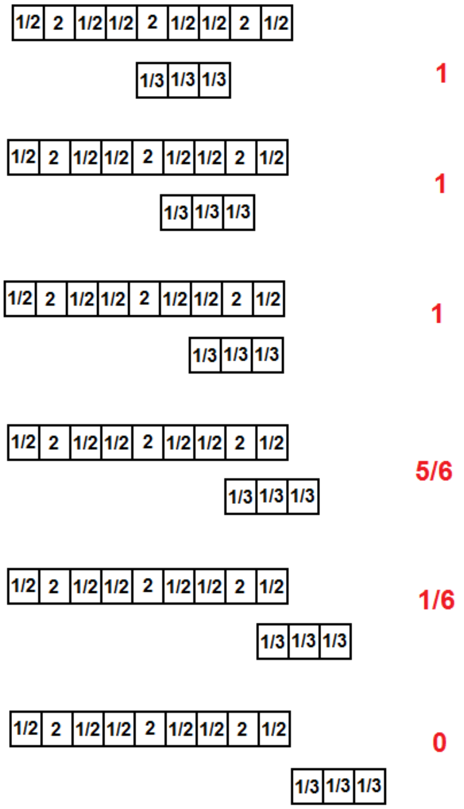


Fig. 12b

Here as in Fig. 12a the periodic component is rejected and the constant is passed in the middle.

Because the signal is not a constant, the end transient is changed however.

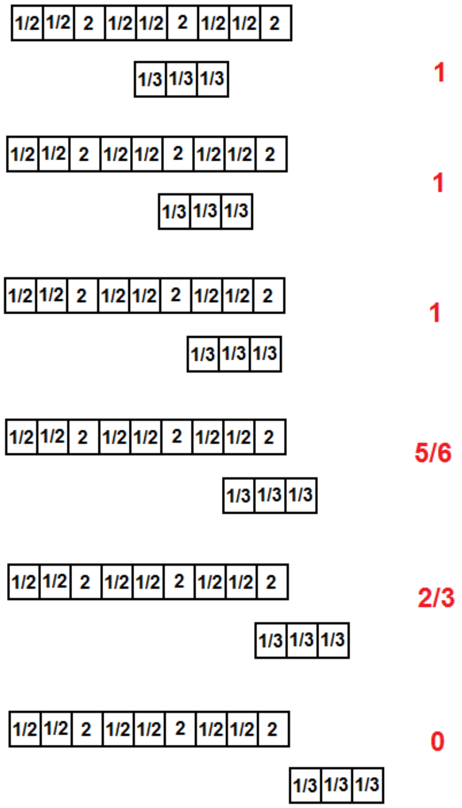


Fig. 12c

Similar to Fig. 12a and Fig. 12b, the periodic component is rejected and the constant passed in the middle.

The end transient is changed yet again as the 2 moves to the very end.

remaiing on the line, the averages are 1/3 and 1/6, exactly half the values we had in Fig. 10. So this instance is not new. What will be different is when we consider the case where the signal sequence advances by one sample and the sample of value 2 comes into the end average. Fig. 12b shows the next case, and we see here 5/6 followed by 1/6. So the transient is a “decay” but a slower one. In the next example, Fig. 12c, the sample with value 2 is all the way to the end. Now the end transient is 5/6 followed by 2/3. Again we find a decay but one that gets off to a slower start.

Certainly these transients are not remarkable in themselves. They are notable in that they may well be artificially created by an artificial termination, and are not related to what happens after the end of the available sequence, for which we have no evidence that it should not be just a continuation of the constant-plus-period-three structure we have observed. That is, the average might well have remained at one. Our treatment here has encroached on the realm of prediction. Indeed, we predicted assuming zeros beyond the end. Averaging is particularly unsuited to prediction. Perhaps pretending to believe anything we do in such a situation is essentially just silly!

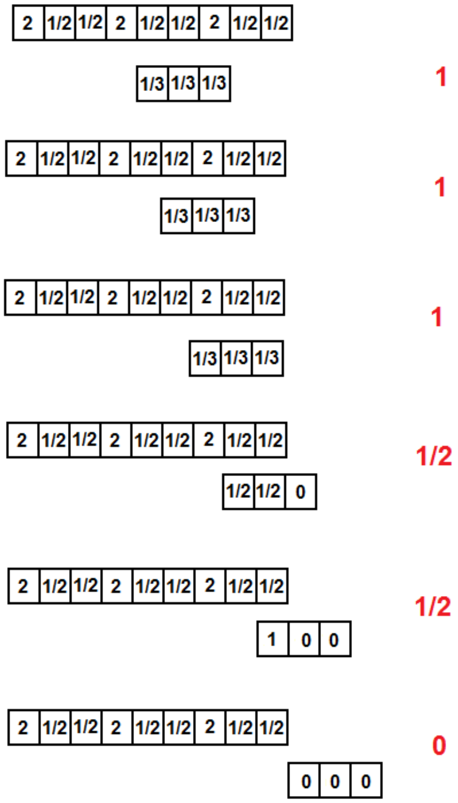


Fig. 13a

Here we use the time-varying averager to divide by the correct number.

The average works fine in the middle, rejecting period 3 and passing the constant

The end effect here changes as the sequence advances (Figs. 13b and 13c).

Here however, we might try to do something reasonable by dividing by the “right number” in the averages, as we eventually did in the first two experiments. This seemed clear enough although we recognized in turn that we had less data to average. Still, it seemed a better approach. Can we do this here?

In Fig. 13a, we have a case where we are agreeing to adjust the weights on the moving average. What we did in Fig. 12 was standard: convolution or FIR filtering. In the case of the two original experiments, we just thought of calculating averages correctly. Now that we are talking about filtering, the suggestion of changing the tap weights of the filter with time is rather radical. We are now talking about a time-varying processing, not a LTI (Linear Time-Invariant) system. For one thing, we don't have build in functions such as ***conv*** or ***filter*** to use. But we can just write out own code. So while we do step into the "wilderness" of time-varying systems, we do know exactly what we are doing, and this we see in Fig. 13a. In Fig. 13a, we see the proper average of 1 in the first three shifts. Then in the fourth shift, the length three filter is too long. We are assuming now that we have no data beyond the end. So we shorten the filter to length two and change the weights both to 1/2. In the bottom shift of Fig. 13a, we have just one remaining sample, so the filter becomes length one with weight 1 (the average is the sample). This does not look unreasonable (yet) because we get two transition samples of 1/2 between a correct average of 1, and a final average going to 0.

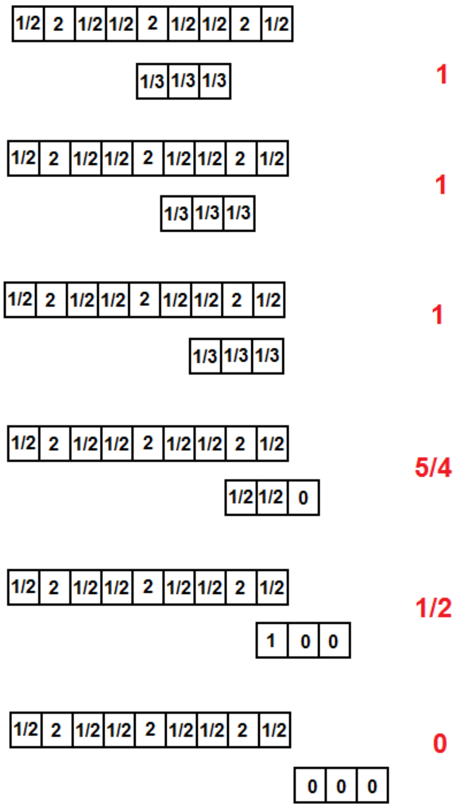


Fig. 13b

Once again the time varying averager.

Here the end transient even exceeds the average at one point, reaching 5/4.

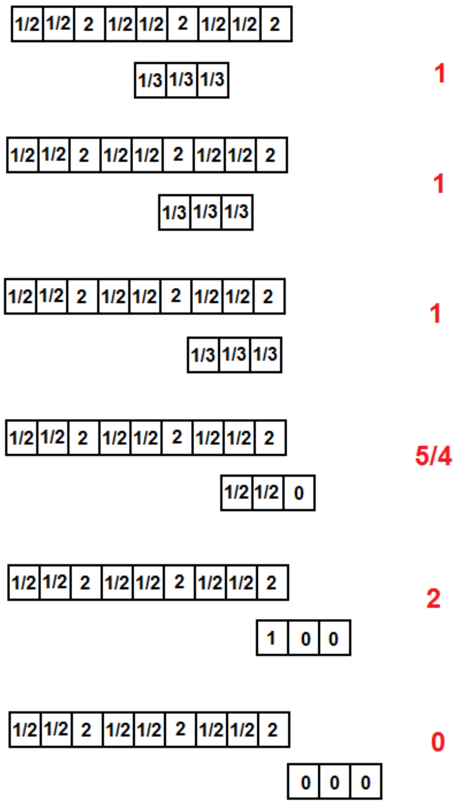


Fig. 13c

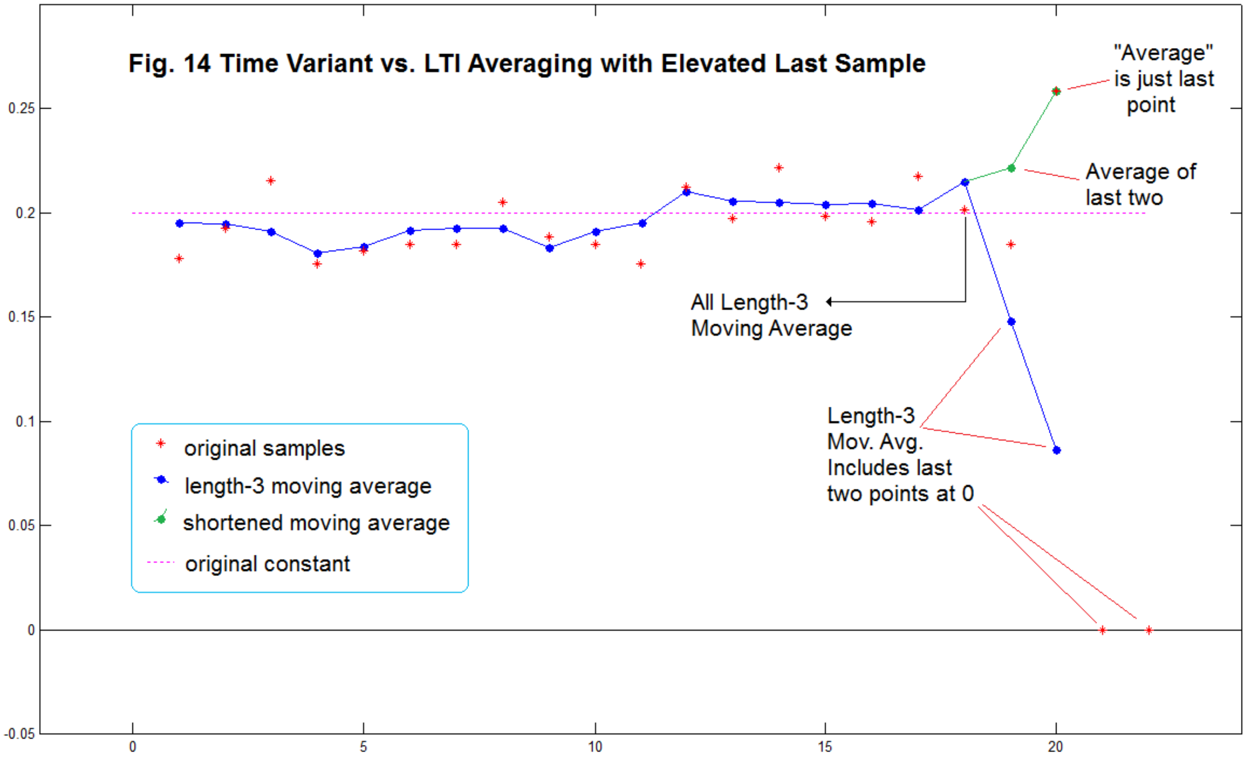
Again the time-varying averager works fine in the middle.

The end effect now has two values that exceed the average (one value is 2, twice the average).

A better notion that there is a problem here comes up when we look at Fig. 13b and Fig. 13c, where instead of the shorter filters dealing only with samples of 1/2, the sample 2 comes into play. In these two cases for the first time, we see transition averages that exceed what we suppose is the true average of 1. These transition averages include 5/4 and 2, as shown. We see these apparent errors are associated with a larger than expected sample at the end. More on this below.

We have noted that in making the tap weights time varying, we are changing th length of the filter, down from three to two to one in this case. Normally we would expect this to increase the bandwidth. However, since this is not a LTI system any more, the notion of a frequency response doesn't have the usual meaning. So while it is clear that some sharper events are apparently appearing, it is best to just think in terms of the calculations.

As a final example, we look at Fig. 14 where we have a set of 20 samples formed by adding random noise (±0.025) to a fixed value of 0.2 (red stars). The 20th sample has an additional 0.05 added to it. Above we expressed an interest as to what happens with extreme samples on the ends. The code ***upt*** at the end here gives the code used. Our purpose was to write code to use a length-3 moving average (adding two extra zeros at the end) and to also compute a length-2 moving average for the last two of the regular samples, and a length-1 moving averages (just the sample itself) as well. The blue dots connected by the blue line show the standard length-3 LTI moving average, and it rolls off at the end due to the appended zero samples at 21 and 22. The alternative time-varying procedure is identical to the length-3 up until the average is computed at time 19, at which time the average is of samples 19 and 20 (green dot). Note that this pulls the average up – in general, and certainly drastically as compared to the length-3 LTI. The ”average” at time 20 is just the sample there, a green dot with a red star center.



CONCLUSIONS:

The overall lesson here is perhaps that data manipulations involving endpoints may be tricky, even with well defined data provenance, and the best of intentions. Long before we might encounter the self-delusion of selection bias, or of having to “patch up” some data that is unavailable beyond our own control, we can fool ourselves when we forget that ends are usually special. If something looks funny, zealous caution is justified.

There is no way to correctly “pretty-up” the most recent data. It won’t confess to containing more information than the uncertainty principle allows, not about the recent data itself, and certainly not as a matter of prediction beyond the end.

```
% av.m  testing with averages

S=[0 1 2 3 4 3 2 1 0 1 2 3 4];

SS=[];
for k=1:8
   for n=1:13
      SS(k,n)=S(n)+2*(rand-.5);
   end
end
SS
figure(1)
plot([-2 15],[0 0],'k')
hold on
plot(SS(1,[1:13]),'r');
plot(SS(2,[1:13]),'r:');
plot(SS(3,[1:13]),'g');
plot(SS(4,[1:13]),'g:');
plot(SS(5,[1:13]),'m');
plot(SS(6,[1:13]),'b');
plot(SS(7,[1:13]),'c');
plot(SS(8,[1:13]),'m:');
plot(S,'k');
plot(S-.01,'k');
plot(S+.01,'k');
plot(S,'ko');

hold off
axis([0 14 -1.2 5.2])
figure(1)

% average
sav=zeros(1,13);
for n=1:13
   for k=1:8
      sav(n)=sav(n)+SS(k,n);
   end
end
sav=sav/8;
figure(2)
plot([-2 15],[0 0])
hold on
plot(S,'k')
plot(sav,'bo')
hold off
axis([0 16 -1 5])
figure(2)
```

```
%create mask
m=[1 1 1 1 1 1 1 1 1 1 1 1 1 ;
   1 1 1 1 1 1 1 1 1 1 0 0 0 ;
   0 0 1 1 1 1 1 1 1 1 1 1 1 ;
   1 0 1 0 1 0 1 0 1 0 1 0 1 ;
   1 1 1 0 0 0 1 1 1 0 0 0 1 ;
   0 0 0 0 0 0 1 1 1 1 0 0 0 ;
   1 1 1 1 1 1 1 1 1 1 1 0 0 ;
   1 0 0 1 0 0 1 0 0 1 0 0 1]

SSM=SS.*m
% average
sav=zeros(1,13);
for n=1:13
   for k=1:8
      sav(n)=sav(n)+SSM(k,n);
   end
end
sav=sav/8;
figure(3)
plot([-2 15],[0 0])
hold on
plot(S,'k')
plot(sav,'bo')
hold off
axis([0 16 -1 5])
figure(3)

% average only non-zeros
SSM=SS.*m
% average
sav=zeros(1,13);
nn=zeros(1,13);
for n=1:13
   for k=1:8
      if SSM(k,n)~=0;
         sav(n)=sav(n)+SSM(k,n);
         nn(n)=nn(n)+1;
      end
   end
end
nn
sav=sav./nn;
figure(4)
plot([-2 15],[0 0])
hold on
plot(S,'k')
plot(sav,'bo')
hold off
axis([0 16 -1 5])
figure(4)
```

```
%create new mask
m=[1 1 1 1 1 1 1 1 1 1 1 1 1 ;
   1 1 1 1 1 1 1 1 1 1 1 1 1 ;
   1 1 1 1 1 1 1 1 1 1 1 1 1 ;
   1 1 1 0 1 1 1 1 1 1 1 1 0 ;
   1 1 1 0 1 1 1 1 1 1 1 1 0 ;
   1 1 1 0 1 1 1 1 1 1 1 1 0 ;
   1 1 1 0 1 1 1 1 1 1 1 1 0 ;
   1 1 1 0 1 1 1 1 1 1 1 1 0] ;

SSM=SS.*m
% average
sav=zeros(1,13);
for n=1:13
   for k=1:8
      sav(n)=sav(n)+SSM(k,n);
   end
end
sav=sav/8;
figure(5)
plot([-2 15],[0 0])
hold on
plot(S,'k')
plot(sav,'bo')
hold off
axis([0 16 -1 5])
figure(5)

% average only non-zeros
SSM=SS.*m
% average
sav=zeros(1,13);
nn=zeros(1,13);
for n=1:13
   for k=1:8
      if SSM(k,n)~=0;
         sav(n)=sav(n)+SSM(k,n);
         nn(n)=nn(n)+1;
      end
   end
end
nn
sav=sav./nn;
figure(6)
plot([-2 15],[0 0])
hold on
plot(S,'k')
plot(sav,'bo')
hold off
axis([0 16 -1 5])
figure(6)
```

```
% mar

n=-10:10
% original target signal
s1=cos(2*pi*n/20)
figure(1)
plot([-12 12],[0 0],'k')
hold on
plot(n,s1)
hold off
axis([-12 12 -1.2 1.2])

%   20 test signals indexed by k
figure(2)
plot([-12 12],[0 0],'k')
hold on
for k=1:20
   A = 1+0.5*(rand-0.5);
   f=1/20 + 0.4*(rand-0.5)/20;
   ph=pi*(rand-0.5)/5;
   B=10*(rand-0.5);
   s0=A*cos(2*pi*n*f +ph)+ B;
   s(k,:)=s0;
   plot(n,s(k,:))
end
axis([-12 12 -7 7])
hold off

% Fig. 3  Remove Means
figure(3)
plot([-12 12],[0 0],'k')
hold on
for k=1:20
   s(k,:)=s(k,:)-mean(s(k,:));
   plot(n,s(k,:))
end
hold off
axis([-12 12 -1.7 1.7])
figure(3)

% now make signals with last sample 0 for r signals
r=5
ss=s;
for k=(20-r):20
   ss(k,21)=0;
end

% compute averages
figure(4)
plot([-12 12],[0 0],'k')
hold on
```

```
sav=zeros(1,21);
ssav=zeros(1,21);
for k=1:20;
   sav=sav+s(k,:);
   ssav=ssav+ss(k,:);
end
sav=sav/20;
ssav=ssav/20;
% - do correct compute of average
ssavr=0;
for k=1:(20-r)
   ssavr=ssavr+ s(k,20);
end
ssavr=ssavr/(20-r);
plot(n,sav,'*')
plot(n,ssav,'ro')
plot(10,ssavr,'go')
plot(n,s1,'c')
hold off
axis([-12 12 -1.2 1.2])
figure(4)

% upt.m
s=c*ones(1,20)+ 0.05*(rand(1,20)-.5 );
A=1;
for n=20:20
   s(n)=s(n)+A*0.05;
end
s=[s 0 0];
figure(1)
plot([-2 24],[0 0],'k')
hold on
plot([0 22],0.2*[1 1],'m:')
plot([1:22],s,'r*')
for k=1:20
   sav(k)=(s(k)+s(k+1)+s(k+2))/3;
end
savg19=(s(19)+s(20))/2;
savg20=s(20);
plot([1:20],sav,'ob')
%plot([1:18],sav(1:18),'og')
plot([1:20],sav(1:20),'b')
plot([18:20],[sav(18) savg19 savg20],'g')

plot(19,savg19,'go')
plot(20,savg20,'go')
hold off
axis([-2 24 -0.05 0.3])
figure(1)
```