**APPLICATION NOTE NO. 405** 

**ELECTRONOTES** 1016 Hanshaw Road Ithaca, NY 14850

Feb 22, 2014

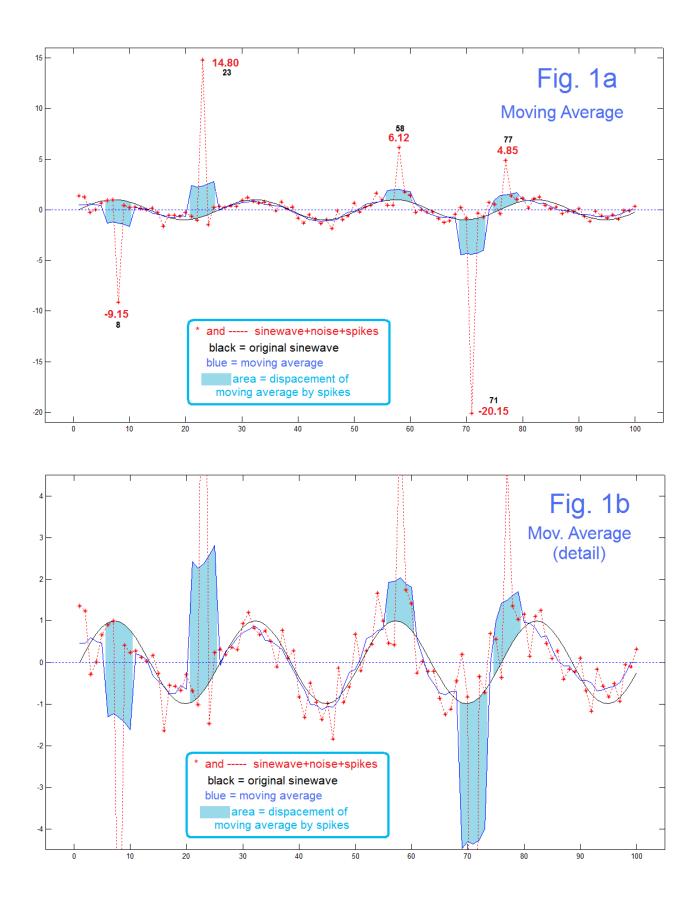
## **REMOVING "OUTLIERS" FROM AVERAGING**

Averaging [1] has been a traditional way of removing noisy data and "smoothing" a time series. Lately, we have looked at some issues with using a moving average [2-4] and considered alternatives [5]. Here we want to mention a direct attack on the problems of averaging when the data has "outliers". The old joke about Bill Gates walking into a bar and having a profound influence on the average salary (the mean salary) of the people there, is the general idea. (At the same time, the median salary is likely little changed). With a "moving average" the output can be sharply displaced (likely, just "wrong") until the outlier has passed through the averaging window.

Here we will be looking at two methods of approaching the outlier problem, the first being a well-established "median filter". The second methods proposed here is to look inside a length-N window, find the average there (as in a moving average) and compute the differences between each of the N samples in the window and the average. Then you throw out the one (or possibly several) samples where the difference is greatest, and average the remaining N-1 samples. That is, like a median filter, we throw out the outlier. (The median filter may actually throw out multiple outliers of course.) Then instead of keeping just the one value (the median) we compute an average of all the samples not discarded. Here we are only suggesting and illustrating, not actually evaluating, the method, which we will call here OLOA (OutLier Out Average).

We will begin with a test signal consisting of four cycles of a unit amplitude sine wave (See Fig. 1a) to which we have added a normal random noise (sigma = 0.5). In addition we have, with a probability of 5%, added at isolated points another random contribution with sigma = 10, producing "noise spikes" as outliers. Indeed, the illustration of Fig. 1 shows that we have exactly five such spikes in this example. So we have two goals. First we would like to just reduce the level of the noise, and this suggests the use of a moving average. Secondly, we are concerned for what happens with the noise spikes. Do they just get reduced like the other random noise, or is there a special concern? Here we have employed a moving average that is length 5 (the "impulse response" is five values of 1/5). The dark blue curve of Fig. 1, along with the lighter blue solid areas, show that the noise spikes have a drastic effect (Bill Gates walks in!). For a zoom, see Fig. 1b. Note that the widths of the blue offsets are indeed five samples.

At this point we encounter a well known conundrum. How do we handle data samples that are apparently (or obviously) outliers? Well one approach is to just throw them out, but this is not something we should resort to without due consideration, and full disclosure. At the very least, we need to have and need to describe an automatic procedure for removing the spikes in a prescribed manner.



AN-405 (2)

For this note, various Matlab programs were used mainly to produce the figures used here, and we shall first limit out inclusion of code to various "snippets" of code for clarity. First, the test signal was written as:

```
 \begin{array}{l} n=1:100; \\ x=sin(2*pi*n/25)+(0.5)*randn(1,100); \\ p=-1000*ones(1,100); \\ for k=1:100 \\ if rand>.95 \\ x(k)=x(k)+10*randn; \\ p(k)=1; \\ end \\ end \end{array}
```

and this was run while looking at the result for a typical and fully illustrative case (such as positive and negative spikes in a central region). Interesting (and perhaps complicating) cases need to be examined where spikes are close together (relative to the averaging length which was 5 here).

The moving average code was just the simple filter:

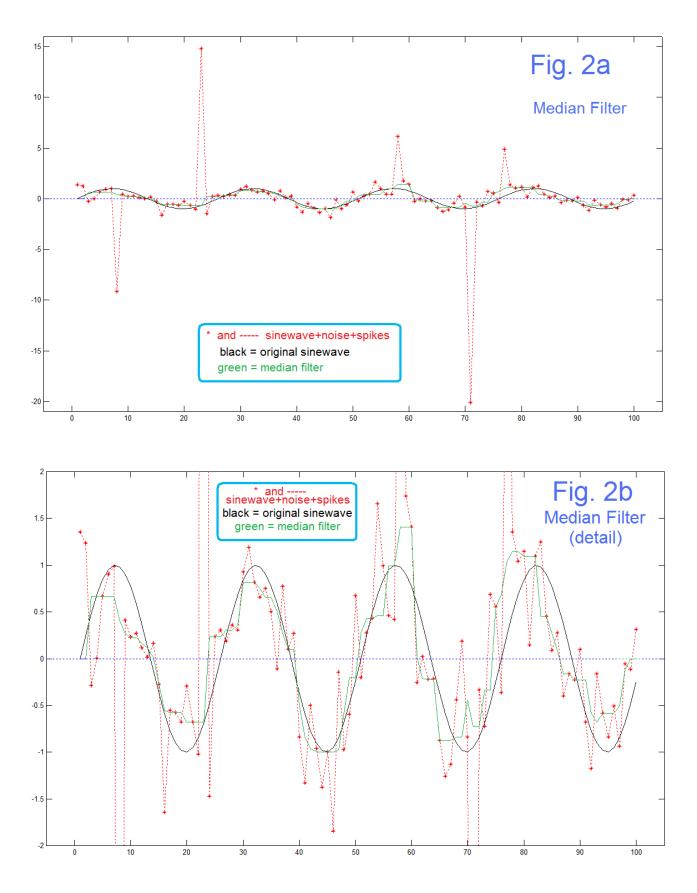
yma=filter([1 1 1 1 1]/5,1,x); yma=[yma(3:100),0 0];

while the median filter required Matlab's *median* function.

```
for k=3:98
ymed(k)=median( x((k-2):(k+2)) );
end
ymed=[0 0 ymed(3:98),0 0];
```

The result of using the median filter is seen in Fig. 2a (zoomed in Fig. 2b). Indeed we are successful in removing the large spikes, relative to the original (red), and relative to the problems with the moving average (Fig. 1). Median filters are famous for removing spiky stuff from a relatively flat background, particularly such things as images where we may have a dark spot against a blue sky which might be a distant bird, or a speck of dust. This could be important! At the same time, removing a speck from an image of a tree (very unflat) would likely go unnoticed. A tree looks like a tree, spot or no spot. So we might wonder if the median filter might cause undesirable distortions to signals that are relatively non-flat.

AN-405 (3)



AN-405 (4)

The moving average filter is an ordinary Linear Time-Invariant (LTI) system, and we can apply linear systems theory and examine such useful notions as frequency response [1, 5]. The median filter (should we even call it a filter?) is not LTI, nor is the 4-of-5 averaging scheme that follows LTI. A median filter may do nothing at all to some segments. A sequence inside the "window" of the median filter that is monotonically rising or falling <u>simply returns the center value</u>. If we have a change of slope within the window, this is not true. For example if we had samples 8, 10, 9, 7, 5 the median filter returns 8. The average would be 7.8.

The procedure here uses an example where we keep 4 of 5 samples and average them . It is an attempt to remove an outlier (as with the median filter) but to retain some of the good features of an averaging process. Here is to simple code used for the "OLOA" method, averaging of 4 of the 5.

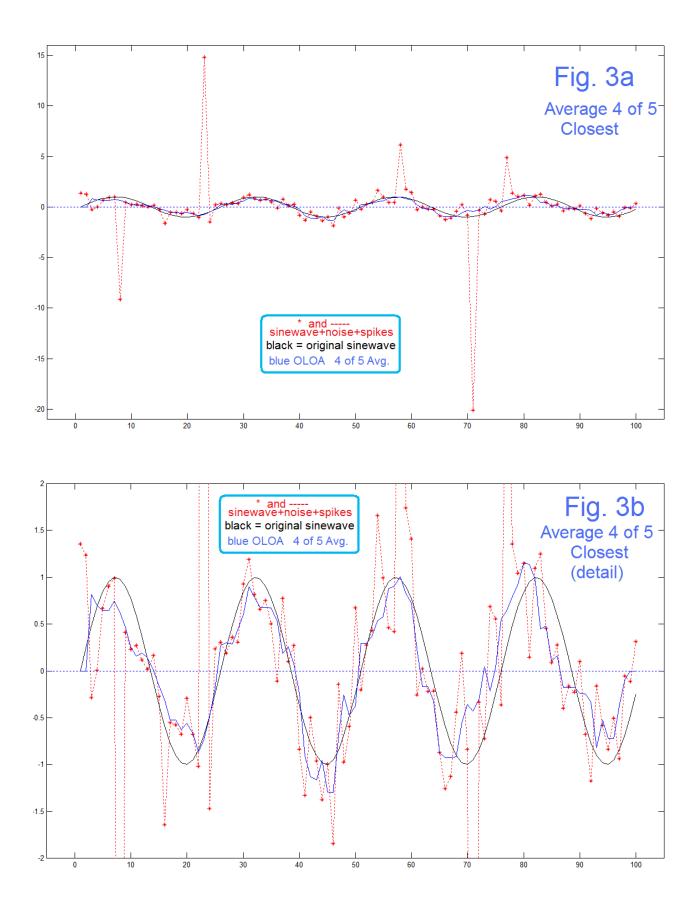
```
for k=3:98
for m=1:100
    xw=x((k-2):(k+2));
    av=sum(xw) / 5;
    d=[ abs(x(k-2)-av),abs(x(k-1)-av), abs(x(k)-av),abs(x(k+1)-av), abs(x(k+2)-av) ];
    [z,ix]=sort(d);
    xx(k)= ( xw(ix(1)) + xw(ix(2)) + xw(ix(3)) + xw(ix(4)) )/4;
    end
end
xx=[xx 0 0];
```

This defines what we did here. For each output, we window length 5 (xw) from -2 to +2, and average (av). Then we compute the distance (d) of the five samples in the window from the average, and sort and index (ix) these distances, and average the four that are closest to the average, as the output.

So we can now compare the different methods for two different types of sequence (of which there are many types) for length 5 examples. In the case of a monotonic sequence, say the sequence is 4, 6, 7, 8, and 9. The mean (average) is 6.8 and the median is 7. In the case of the OLOA averaging of 4 of 5, we would note that the distances of the samples from the average are 2.8, 0.8, 0.2, 1.2, and 2.2, so the largest difference is on the first sample (the 4). We average the remaining four (6 + 7 + 8 + 9)/4 = 7.5, and that is our output. Note that in a monotonic sequence, one end of the other is discarded by this new method and the result shifts away from the discard.

A sequence that reverses direction perhaps 8, 10, 9, 7, and 5 has an average of 7.8, and a median of 8. The distances are 0.2, 2.2, 1.2, 0.8, and 2.8, so in our average 4 of 5 we discard the last value, 5, and average (8 + 10 + 9 + 7)/4 = 8.5.

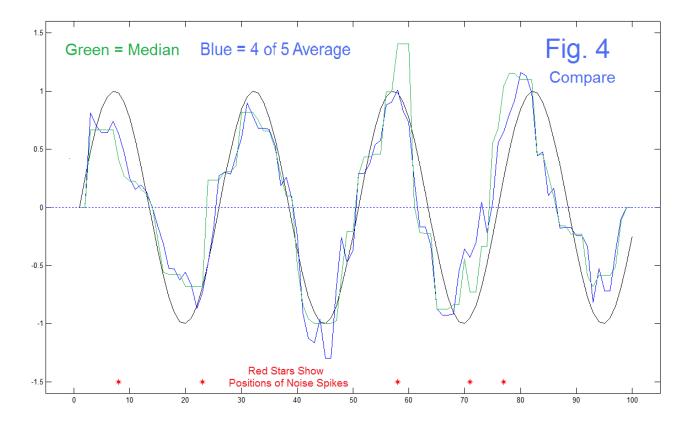
For extended monotonic regions it may well be the case that there is a consistent shift when averaging 4 of 5. In turn-around or relatively flat segments, less in the way of predicting what will happen is initially available.



AN-405 (6)

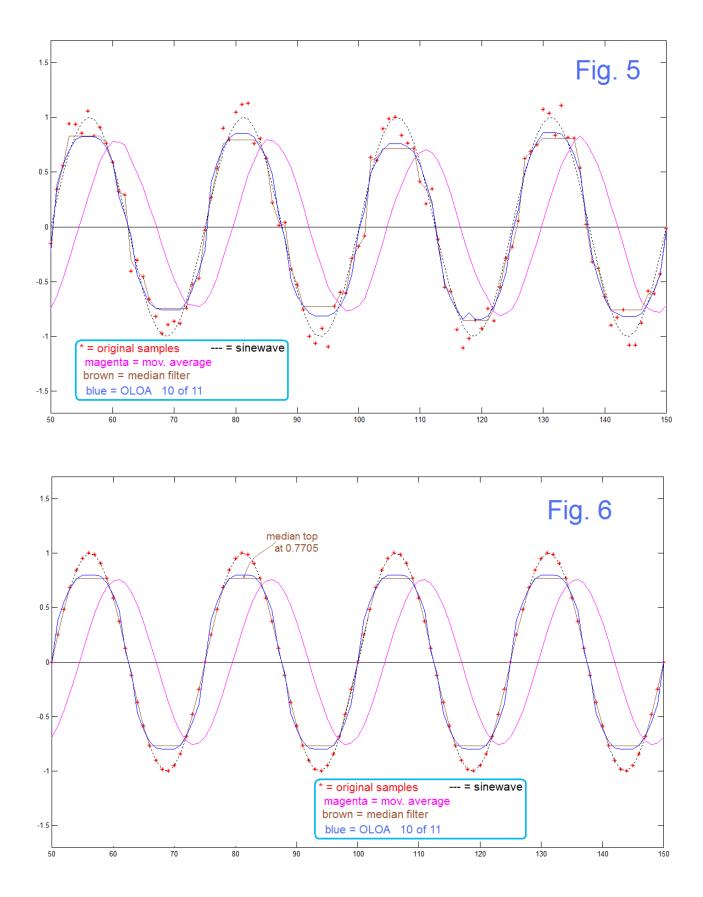
Fig. 3a and the zoom of Fig. 3b show the new OLOA averaging of four of the closest of 5, for exactly the same input test signal that was used for producing Fig. 1 and Fig. 2. We see that the new method is <u>quite similar to the median filter in the removal of spiky noise</u>, and relatively similar to the median filter in respect to the random noise.

It should be noted that in our very limited examples here we have not looked at one that has two spikes very close together (separated by less than 5). Here our probability of a spike was set to just 5%, and additionally there was a probability that the amplitude of the spike might end up small. So although both the median and the OLOA 4 of 5 averaging are effective against spiky noise, large problems similar to the large block displacements of the moving average may still be found, occasionally. Fig. 4 shows a zoomed comparison of the two spike-removing methods. There is no clear winner.



In order to get a better comparison (and we have only begun this study as of this note), we have stopped generating spikes, reduced the random noise level (to 0.15) and increased the length of the examination window (from 5 to 11) and the results shown in Fig. 5 reveal a lot more. First note the magenta curve that is offset to the right which is the moving-average for reference. The offset is the phase shift of this LTI system and here is useful for getting the moving-average curve clear of the other results. This is the nicest result were it not for the fact that we agree it is not well suited to outliers (Fig. 1).

The main curves of interest here are the brown median filter and the dark blue OLOA 10-of-11 average. In Fig. 4 we saw little to suggest a preference, but here there is a



AN-405 (8)

significant difference that manifests itself in one finding that the tops of the waveforms in the median filter case (brown) being exactly flat for a good number of samples, while the OLOA method tries to round significantly. Why the flat tops!

In looking for the cause of the flat tops in the median filter case, we first note that it is NOT a case of an intruding periodicity, as the input (having a significant random component) is not periodic. This permits us completely remove the random noise for a cleaner result (Fig. 6). One then gets to suspecting some sort of programming error. Not so. Thus eventually one gets into the "nuts and bolts" and digs in to see what is really going on. Indeed the program does just what it is supposed to do.

The table in Fig. 7 displays the actual numbers in the calculations. These are for six sequential values where the median is flat at 0.7705 (columns 2 through 7) and three others (columns 1, 8, and 9) where it changes. The top set of columns simply show signal as it passes through a window. As we move left to right through the top set of columns,

| wind =   |
|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| -0.4818  | -0.2487  | 0.0000   | 0.2487   | 0.4818   | 0.6845   | 0.8443   | 0.9511   | 0.9980   |
| -0.2487  | 0.0000   | 0.2487   | 0.4818   | 0.6845   | 0.8443   | 0.9511   | 0.9980   | 0.9823   |
| 0.0000   | 0.2487   | 0.4818   | 0.6845   | 0.8443   | 0.9511   | 0.9980   | 0.9823   | 0.9048   |
| 0.2487   | 0.4818   | 0.6845   | 0.8443   | 0.9511   | 0.9980   | 0.9823   | 0.9048   | 0.7705   |
| 0.4818   | 0.6845   | 0.8443   | 0.9511   | 0.9980   | 0.9823   | 0.9048   | 0.7705   | 0.5878   |
| 0.6845   | 0.8443   | 0.9511   | 0.9980   | 0.9823   | 0.9048   | 0.7705   | 0.5878   | 0.3681   |
| 0.8443   | 0.9511   | 0.9980   | 0.9823   | 0.9048   | 0.7705   | 0.5878   | 0.3681   | 0.1253   |
| 0.9511   | 0.9980   | 0.9823   | 0.9048   | 0.7705   | 0.5878   | 0.3681   | 0.1253   | -0.1253  |
| 0.9980   | 0.9823   | 0.9048   | 0.7705   | 0.5878   | 0.3681   | 0.1253   | -0.1253  | -0.3681  |
| 0.9823   | 0.9048   | 0.7705   | 0.5878   | 0.3681   | 0.1253   | -0.1253  | -0.3681  | -0.5878  |
| 0.9048   | 0.7705   | 0.5878   | 0.3681   | 0.1253   | -0.1253  | -0.3681  | -0.5878  | -0.7705  |
|          |          |          |          |          |          |          |          |          |
| sorted = |
-0.4818	-0.2487	0.0000	0.2487	0.1253	-0.1253	-0.3681	-0.5878	-0.7705
-0.2487	0.0000	0.2487	0.3681	0.3681	0.1253	-0.1253	-0.3681	-0.5878
0.0000	0.2487	0.4818	0.4818	0.4818	0.3681	0.1253	-0.1253	-0.3681
0.2487	0.4818	0.5878	0.5878	0.5878	0.5878	0.3681	0.1253	-0.1253
0.4818	0.6845	0.6845	0.6845	0.6845	0.6845	0.5878	0.3681	0.1253
0.6845	0.7705	0.7705	0.7705	0.7705	0.7705	0.7705	0.5878	0.3681
0.8443	0.8443	0.8443	0.8443	0.8443	0.8443	0.8443	0.7705	0.5878
0.9048	0.9048	0.9048	0.9048	0.9048	0.9048	0.9048	0.9048	0.7705
0.9511	0.9511	0.9511	0.9511	0.9511	0.9511	0.9511	0.9511	0.9048
0.9823	0.9823	0.9823	0.9823	0.9823	0.9823	0.9823	0.9823	0.9823
0.9980	0.9980	0.9980	0.9980	0.9980	0.9980	0.9980	0.9980	0.9980
k =								
79	80	81	82	83	84	85	86	87
median =								
0.6845	0.7705	0.7705	0.7705	0.7705	0.7705	0.7705	0.5878	0.3681
		Fig 7						

Fig. 7

AN-405 (9)

the sequence moves up, one step at a time, the top number disappearing and a new number appearing at the bottom. Paying attention to the middle of these windows, note that it goes through a peak of the sinusoidal waveform: 0.6845 on the left, through 0.9980. to 0.3681 on the right. This is clear enough.

The bottom set of columns is just the windows above sorted according to value. The median is the middle of these columns. (We used the Matlab *median* function, but could have gotten the value by selecting from the sorted columns). Indeed we see that the median value of 0.7705 persists for columns 2 through 7. To our surprise (perhaps), in the case of the six columns with median 0.7705, the five most positive numbers (printed below the midpoints) are all identical for each column. So are these six columns identical? Not at all. In contrast, the five numbers more negative than the 0.7705 median are jumping around! Some of the numbers have to change of course since each column changes two numbers.

So the question is: why is it the low-side numbers that jump around? <u>Well, it is</u> <u>because we are looking at samples surrounding a peak</u>. Near the peak, the median is large, and there are more large numbers. For a range of shifts within the window, enough larger values are inside the window that the sorting above the median contains the same actual values. This is why the result is flat for this type of example.

Here, for the case where there are large but rare "Noise Spikes" we have seen that a moving average can have serious drawbacks. In such cases, a median-filter is often employed, but here we suggest a new method (OLOA) as being possibly better. In cases where we are not concerned with spikes, the Savitzky-Golay [5] should be considered for its larger bandwidth. In most cases, alternatives to moving-average should probably be examined.

## REFERENCES

[1] We suggest a general familiarity with digital filter design, which often features a moving average filter as a first example, such as: <u>http://electronotes.netfirms.com/EN197.pdf</u>

[2] B. Hutchins, "Averaging - and Endpoint Garbage," Electronotes Application Note No. 395, March 30, 2013 http://electronotes.netfirms.com/AN395.pdf

[3] B. Hutchins, "Yearly Moving Averages as FIR Filters", Electronotes Application Note No. 401, Dec 22, 2013 http://electronotes.netfirms.com/AN401.pdf

[4] B. Hutchins, "Spurious Correlations Due to Filtering (of Noise)", Electronotes Application Note No. 403, Jan 27, 2014 http://electronotes.netfirms.com/AN403.pdf

[5] B. Hutchins, "Savitzky-Golay Smoothing", Electronotes Application Note No. 404, Feb 13, 2014 <u>http://electronotes.netfirms.com/AN404.pdf</u>

## **PROGRAM CODE**

The Matlab code below is included for complete documentation and will indicate how the material of Fig. 5, 6, and 7 was generated. The additional code snippets in the main text also define the methods here.

```
%smtest
          smooth test
function smtest(A,w)
n=0:199;
x=sin(2*pi*n/25);
xr=x+A*randn(1,200);
h=(1/w) * ones (1, w);
yma=filter(h,1,xr);
s=ceil(w/2)
ymed=zeros(1,200);
for k=25:175
   % Check Certain Values for Fig. 7 of text
   if k==79;
      test=xr( (k-s):(k+s) );
      wind=test'
      sorted=sort(test) '
      k
      tm=median(test)
   end
   if k==80;
      test=xr( (k-s):(k+s) );
      wind=test'
      sorted=sort(test) '
      k
      tm=median(test)
   end
   if k==81;
      test=xr( (k-s):(k+s) );
      wind=test'
      sorted=sort(test) '
      k
      tm=median(test)
   end
   if k==82;
      test=xr( (k-s):(k+s) );
      wind=test'
      sorted=sort(test) '
      k
      tm=median(test)
   end
   if k==83;
      test=xr( (k-s):(k+s) );
      wind=test'
      sorted=sort(test) '
      k
      tm=median(test)
   end
```

```
if k==84;
      test=xr( (k-s):(k+s) );
      wind=test'
      sorted=sort(test) '
      k
      tm=median(test)
   end
      if k==85;
      test=xr( (k-s):(k+s) );
      wind=test'
      sorted=sort(test) '
      k
      tm=median(test)
   end
    if k==86;
      test=xr( (k-s):(k+s) );
      wind=test'
      sorted=sort(test) '
      k
      tm=median(test)
   end
   if k==87;
      test=xr( (k-s):(k+s) );
      wind=test'
      sorted=sort(test) '
      k
      tm=median(test)
   end
   ymed(k) = median(xr((k-s):(k+s)));
end
% Proposed OLOA method
ynew=zeros(1,200);
for k=25:175
   xw = xr((k-s):(k+s));
   av=sum(xw)/w;
   d = abs(xw-av);
   [z,ix]=sort(d);
   ynew(k)=sum ( xw( ix (1:(w-1))) ) / (w-1) ;
end
figure(1)
plot(n,x,'k:')
hold on
plot([-10 210],[0 0],'k')
plot(n,xr,'r*')
plot(n,yma,'m')
plot(n,ymed,'c')
plot(n,ynew,'b')
hold off
axis([50 150 -1.7 1.7])
figure(1)
xr(75:90)
ymed(75:90)
```