ELECTRONOTES
1016 Hanshaw Road
Ithaca, NY 14850

APPLICATION NOTE NO. 417

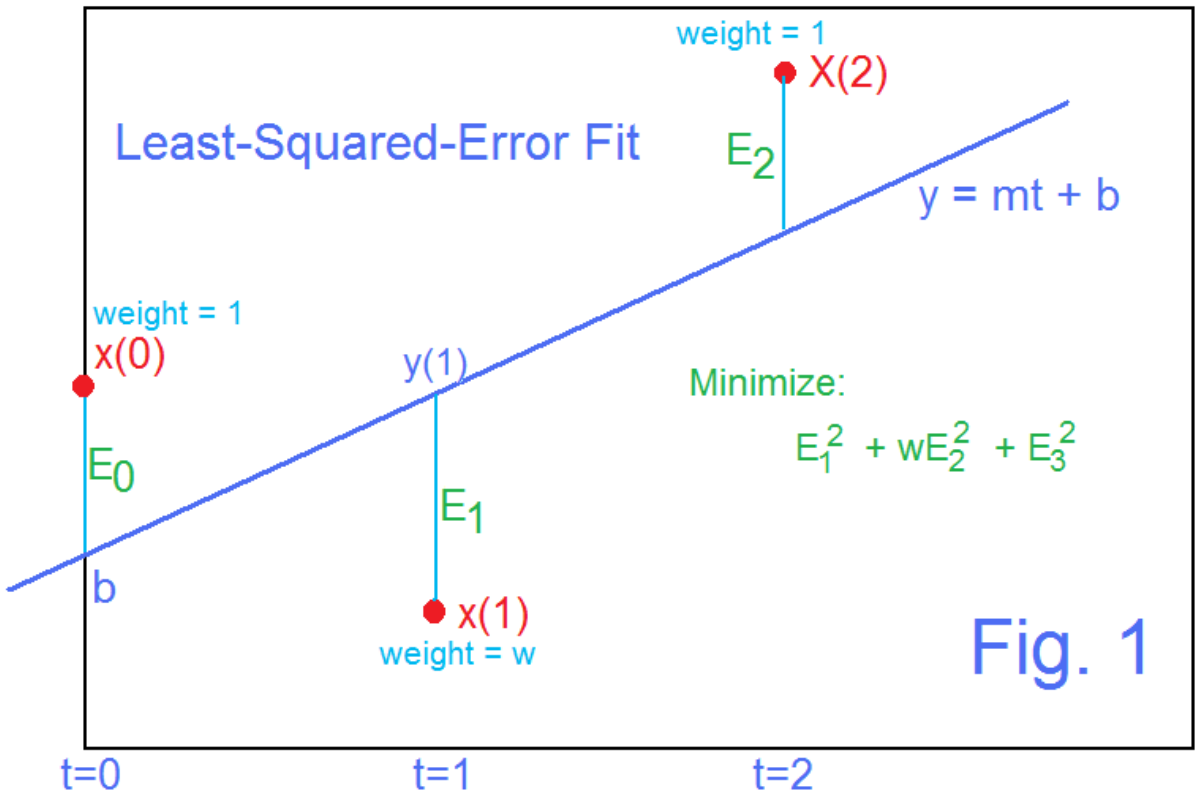
Nov 22, 2014

TIME DOMAIN WEIGHTED LEAST SQUARES

INTRODUCTION:

Recently in ***Electronotes,*** EN#223 [1], we had a review of time series smoothing which, among many other things, looked again at Savitzky-Golay (previously here as AN-404 [2]), and identified it with least-squares-error smoothing. At the very end, we briefly looked at a (point-by-point) individually weighted least-squares fit that extended an App Note from 1992, AN-318 [3], by using a simple example of a hand-derived case of weighted error. Specifically, while the equal weighted error in AN-318 (a fit of a straight line to three points) gave a simple length-three moving average h(n) = [1/3 1/3 1/3], weighting the error on the middle point by 2 rather than all by 1 gave h(n) = [1/4 1/2 1/4]. The details were not given there, but seemed quite correct.

Here we will first review both length-3 straight line hand-calculated fits as reported [1,3]. Then we can use a general weighted least squares using the matrix inversion techniques. Finally we will want to give an option for using a LOESS or LOWESS weighting [1]. This will be general in using higher-order polynomial fits.



Weighted Least-Squares Error Fit of Straight-Line to Three Points

Here we propose to work out the simplest possible case "by hand" so as to offer a test result for our programs that follow. Fig. 1 shows the setup that is the same as the presentation in AN-318 [3] EXCEPT we have added a weight w for the middle point instead of having all three points with equal weight of 1. We saw [3] that the three filter coefficients were a standard length-3 moving average when the errors were weighted the same on all three taps, and were h=[1/4 1/2 1/4] when the weight on the center tap was set to 2 rather than to 1 [1]. Here we give the details on those simple yet tedious calculations. We will see that the same result comes out in our general program below.

From Fig. 1 we see that there are three errors that result from trying to fit a straight line to three points (assuming they are not already on a straight line).

$$E_0 = x(0) - b \tag{1a}$$

$$E_1 = x(1) - m - b \tag{1b}$$

$$E_2 = x(2) - 2m - b \tag{1c}$$

The right sides of these three equations are squared to give the squared error on each of the three points. Then we want to calculate the total squared error, weighting the center error by w and the end terms by 1:

$$\begin{aligned} E^2 = {} & E_0^2 + w E_1^2 + E_2^2 \\ = {} & x(0)^2 + wx(1)^2 + x(2)^2 \\ & + b[-2x(0) - 2wx(1) - 2x(2)] \\ & + m[-2wx(1) - 4x(2)] \\ & + b^2(w+2) + m^2(w+4) + mb(2w+4) \end{aligned} \tag{2}$$

Now to minimize this, we take partial derivatives with respect to b and to m, set these to zero, and solve the two equations in two unknowns for b and m.

$$\frac{\partial E^2}{\partial b} = -2x(0) - 2wx(1) - 2x(2) + 2bw + 4b + 2wm + 4m = 0 \tag{3a}$$

$$\frac{\partial E^2}{\partial m} = -2wx(1) - 4x(2) + 2bw + 4b + 2wm + 8m = 0 \tag{3b}$$

Which go nicely to matrix form;

$$\begin{bmatrix} (w+2) & (w+2) \\ (w+2 & (w+4) \end{bmatrix} \begin{bmatrix} b \\ m \end{bmatrix} = \begin{bmatrix} x(0) + wx(1) + x(2) \\ wx(1) + 2x(2) \end{bmatrix} \tag{4}$$

which solve for :

$$m = \frac{x(2)-x(0)}{2} \tag{5a}$$

$$b = \frac{x(0)(4+w) + x(1)(2w) + x(2)(-w)}{2(w+2)} \tag{5b}$$

Here equation (5a) is notable as it indicates that the slope is independent of the weight of the error at the center. That this has to be so is easily seen be envisioning the line to pivot about its center.

What we have here is not yet a filter. We have only a curve fit, not extended to an arbitrary center output point. To find the filter, we can imagine an impulse passing through for x, and always solving for y(1) at the center. This is our familiar ploy that the impulse response of the filter is the response of the filter to an impulse. (A tautological but useful ploy at times as it really does suggest a methodology.) Here we can simply and directly solve for y(1), y at t=1:

$$y(t=1) = mt + b = m + b = \frac{x(0)+wx(1)+x(2)}{w+2} \tag{6}$$

or

$$h(0) = \frac{1}{w+2} \tag{7a}$$

$$h(1) = \frac{w}{w+2} \tag{7b}$$

$$h(2) = \frac{1}{w+2} \tag{7c}$$

Which agrees with earlier findings. Note also the limiting cases of w=0 means the center point is ignored and the output is the average of the endpoints. If w=∞ we have only the middle point. Makes sense.

We will plot some examples when we finish the general case (higher order polynomials).

The General Problem

The algebra above is of the sort I never enjoy – tedious to the point where one is tempted to take short-cuts that almost certainly guarantee error. It took me three tries to get it to look right. Certainly anything beyond a first-order fit to three points is daunting. Fine – that's why we have computers. And there's nothing like running "exercises" on a computer to see what the mathematics really means.

Keep in mind that here we are involved with curve fitting to data points. Classically we begin with fitting N equation coefficients to N data points (see example below) and solving N equations in N unknowns (making the usual assumptions). We carry this further to the case of extra points (just above, two coefficients and three data points)

which requires a least-square solution, manifesting itself in a least-squares or "pseudo inverse". Finally, we can consider the case where not only are there excess points, but some are more important than others and we look to weight the errors associated with these points more heavily. The simplest case of this weighting was given in the hand-example just above.

Finally we note that we are applying these ideas to two related but different-looking problems: that of basic curve fitting, and an unexpected bonus of a filter-design method. An excellent and brief presentation of the least-squares matrix calculations is that of Jackson [4] while we have also done similar weighted least squares calculations in the frequency domain for filter designs in a frequency-sampling context [5].

Just as a first-order polynomial is characterized by two coefficients (traditionally slope m and intercept b as x(t) = mt+b. we can have higher order polynomials such as for example a fifth-order:

$$x(t) = at^5 + bt^4 + ct^3 + dt^2 + et + f \tag{8}$$

For a given set of samples of x(t) we can write instances of the equation as:

$$x(t_1) = at_1^5 + bt_1^4 + ct_1^3 + dt_1^2 + et_1 + f \tag{9a}$$

$$x(t_2) = at_2^5 + bt_2^4 + ct_2^3 + dt_2^2 + et_2 + f \tag{9b}$$

$$x(t_3) = at_3^5 + bt_3^4 + ct_3^3 + dt_3^2 + et_3 + f \tag{9c}$$

. . . . . .

or in matrix form:

$$\boldsymbol{x} = \boldsymbol{T}\boldsymbol{a} \tag{10}$$

where $\boldsymbol{a}$ is the vector stand-in for the coefficient a,b,c,d,e,f.

If we have exactly six such equations, we solve exactly for a,b,c,d,e, and f (the matrix is square). If we have an excess of data points, equation (10) is solved to minimize the squared error, using conveniently, something called a "pseudo-inverse" which is available in many math packages. The pseudo-inverse is usually equivalent to $\mathbf{(T^tT)^{-1}T^t}$ or to just $\mathbf{T^{-1}}$ if **T** is square. In the case of weighting, the inversion is done with $\mathbf{(T^tWT)^{-1}\,T^tW}$ as was the case in the frequency domain [5]. Here **W** is a square matrix with the weights on the diagonal (0 else) and of size equal to the number of points in the data. In Matlab code, the inversion of equation (10) is:

```
a = inv(T'*W*T)*T'*W*x'
```
(11)

where **T'** is the transpose. This will appear in the two programs below.

Two Example Programs

Why two programs? Well there are two main applications. The first application is classical data smoothing, curve-fitting, and/or modeling. This could be a "scatter-plot" such as a plot of students' physics scores against calculus scores (where we might expect to uncover a strong correlation) or it could be a plot of some variable as a function of time (or other running independent variable), often at discrete intervals as a sequence of series. The second application jumps to what seems a complication: fitting a polynomial to a running set of neighboring data points of a series. While signal processing engineers expect this to ultimately result in an FIR (Finite Impulse Response) digital filter, it is largely unappreciated that this is, first of all true; and that the FIR scheme means that implementation is thus barely more complicated than a moving average.

The first program here is WLS (weighted least squares) that has as its inputs a proposed polynomial order, a signal sequence x, and a vector of weights that is the same length as x. (This weight vector could be the LOESS weights that are an option in Program 2). The idea is to fit a polynomial to the data points, not to design a filter.

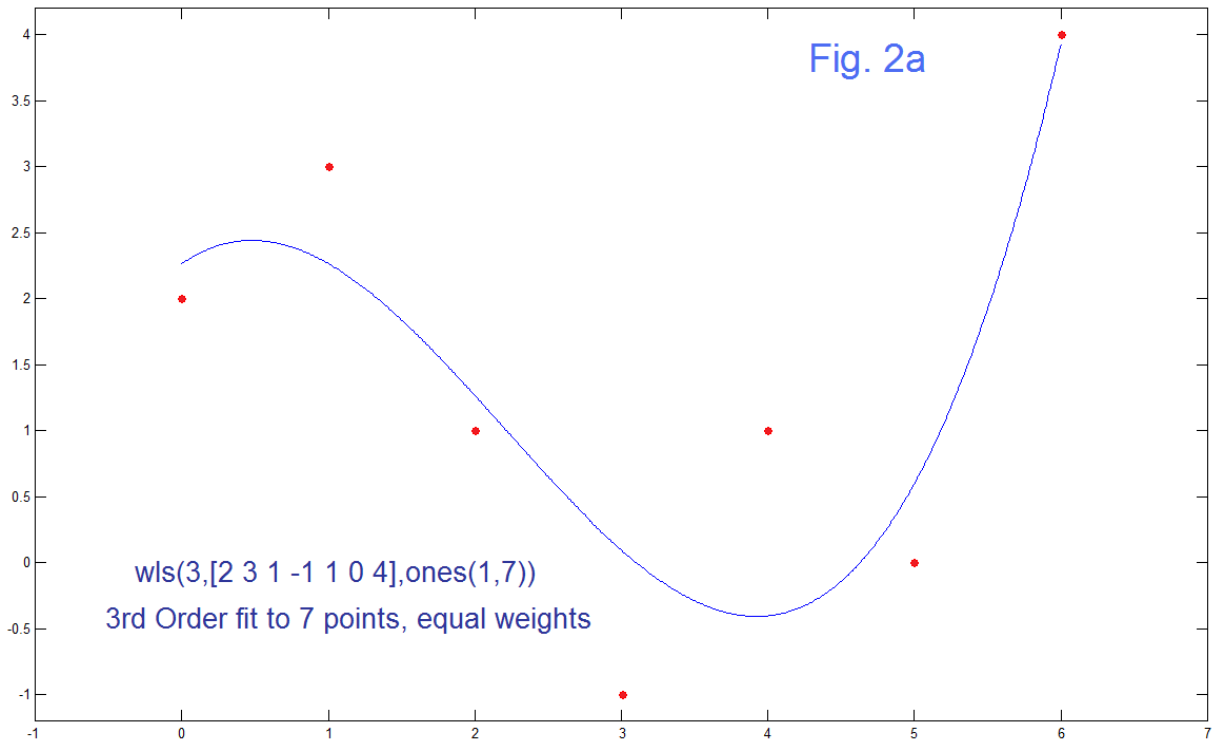
PROGRAM 1

```
function WLS(pord,x,weights)
% plots weighted fit
T=[]
for t=0:length(x)-1
      for k=pord:-1:0
          T(k+1,t+1)=t^k;
      end
end

T=flipud(T)
T=T'

W=diag(weights)
a=inv(T'*W*T)*T'*W*x'

t=0:length(x)-1;
tt=0:0.01:length(x)-1;;
xt=polyval(a,tt);

figure(1)
plot(tt,xt)
hold on
plot(t,x,'or')
mx=max([x xt]);
mn=min([x xt]);
if mn>0
   mn=o;
end
axis([-1 length(x)  mn-.2 mx+.2])
hold off
figure(1)
```

Fig. 2a shows the basic test where we are fitting a 3rd-order polynomial (two turn-arounds) to 7 point, [2 3 1 -1 1 -1 4] with equal weighting. We note that the curve does not go through any of the points exactly (in general, it might do very well at a point or two by accident). Fig. 2b shows a corresponding case where we have increased the weighting on the 6th point from 1 to 200, and the curve does seem to go right through the 6th point. Perhaps we have a desire to fit that point especially well. We can if we wish adjust the weights on multiple points – any two, three, four, or five of them. What we can't do is increase the weight on all six points with the expectation that we get anything except Fig. 2a back as a result. So that's the idea of weighting.

Fig. 2c and Fig. 2d are control cases. In Fig. 2c we have simply plotted the polynomial outside the original range. There are no points there to restrain the polynomial, and in consequence it is free to wonder about and thus reduce the error on the 7 points given. As expected of any polynomial, outside some central range it rather rapidly heads for +∞ or -∞. This is why polynomials which have vertical tendencies are not good models for signals (which have horizontal tendencies) except in a central region. The other control that had to be shown was that if you increase the order from 3 to 6 (Fig. 2d), the polynomial can go through all seven points.

We have not offered here as an option the idea of using a LOESS (or LOWESS) weighting, which looks like an envelope. This will be offered with the filter design. The reason we do not offer it here is that when we are fitting a curve to displayed points, we have no prior expectation that we have less concern for the points near the end relative to those near the middle. The program does however offer us a weight vector as an input, so we can of course try any weighting we want to look at – not just equal weighting. This LOESS is seen in Fig. 2e, and we see that the result pays less attention to the ends, but otherwise makes little difference – at least in this example.



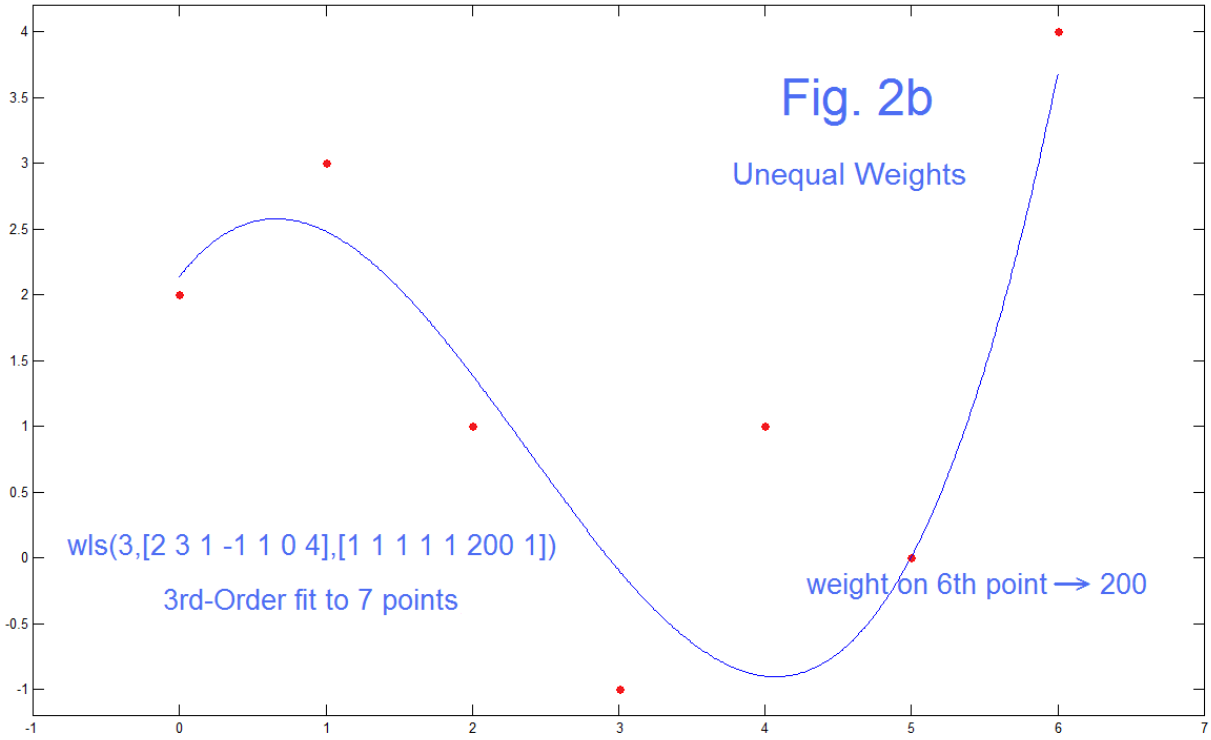


Fig. 2b
Unequal Weights
wls(3,[2 3 1 -1 1 0 4],[1 1 1 1 1 200 1])
3rd-Order fit to 7 points
weight on 6th point → 200

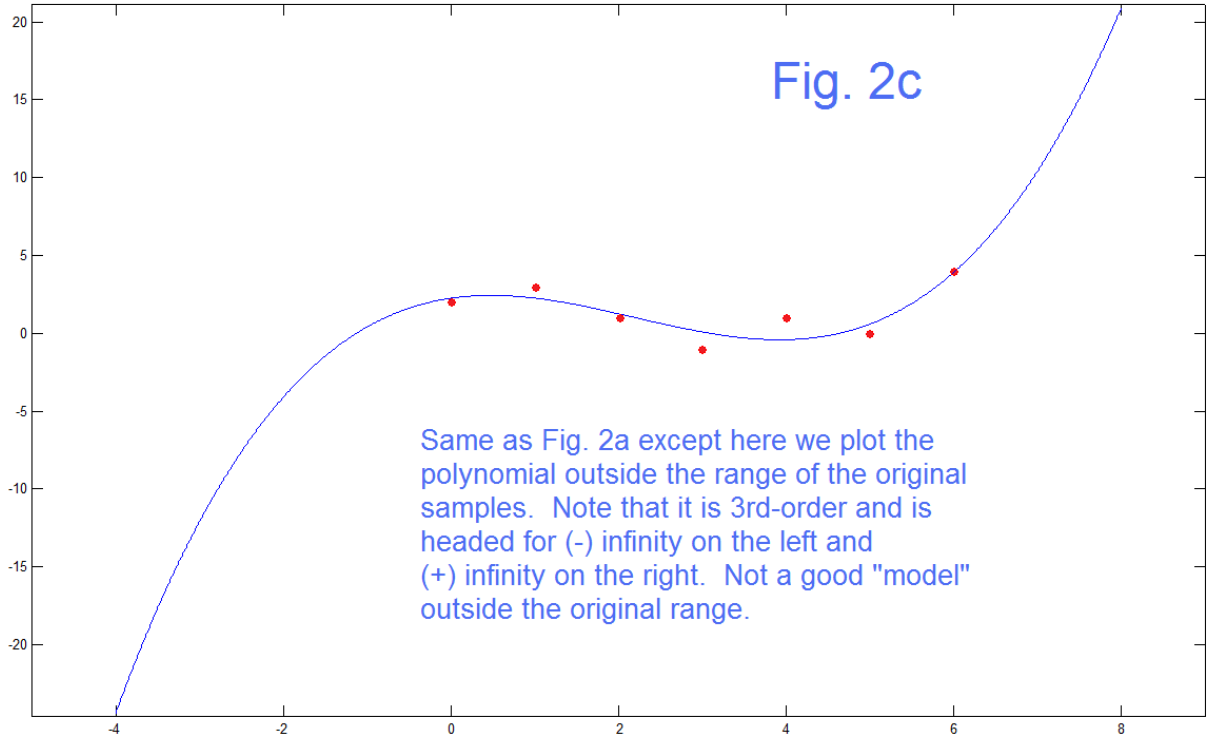


Fig. 2c
Same as Fig. 2a except here we plot the polynomial outside the range of the original samples. Note that it is 3rd-order and is headed for (-) infinity on the left and (+) infinity on the right. Not a good "model" outside the original range.

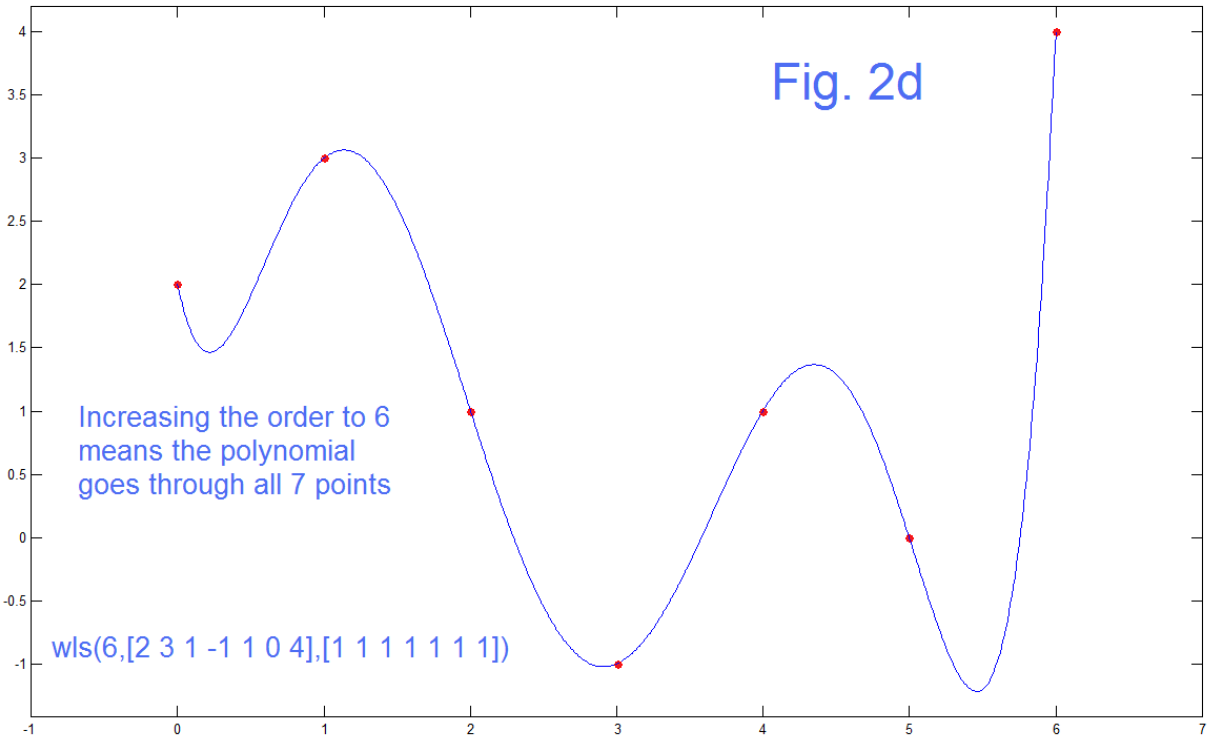


Fig. 2d
Increasing the order to 6
means the polynomial
goes through all 7 points
wls(6,[2 3 1 -1 1 0 4],[1 1 1 1 1 1 1])

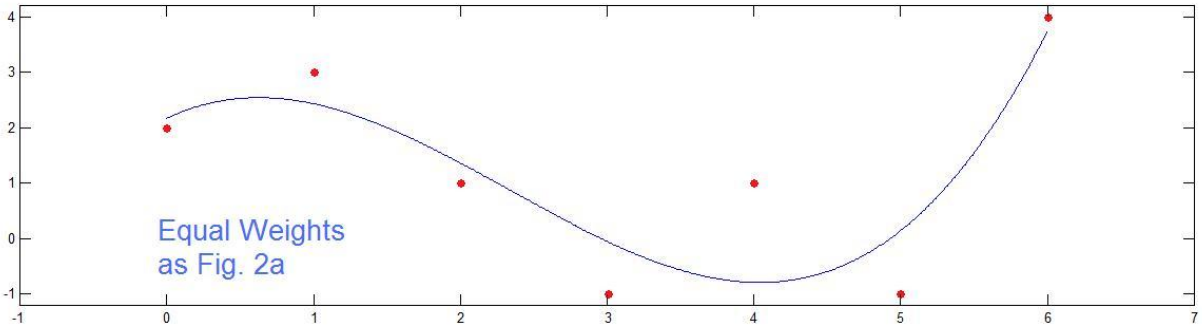
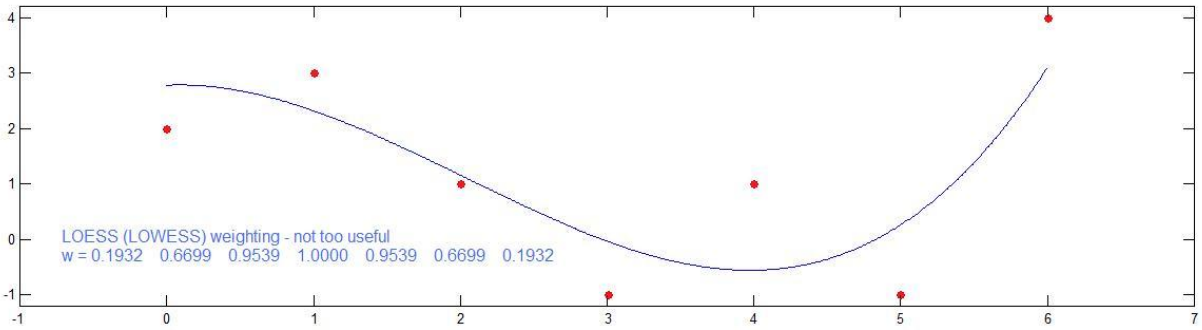

Equal Weights
as Fig. 2a

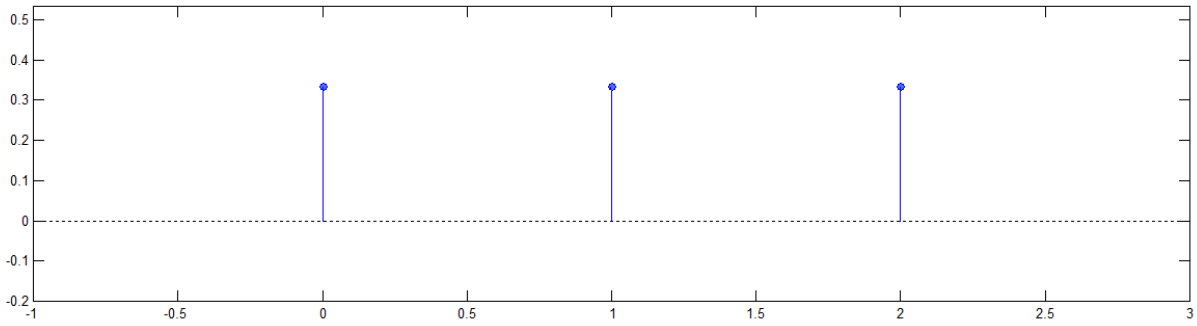
Fig. 2e


LOESS (LOWESS) weighting - not too useful
w = 0.1932 0.6699 0.9539 1.0000 0.9539 0.6699 0.1932

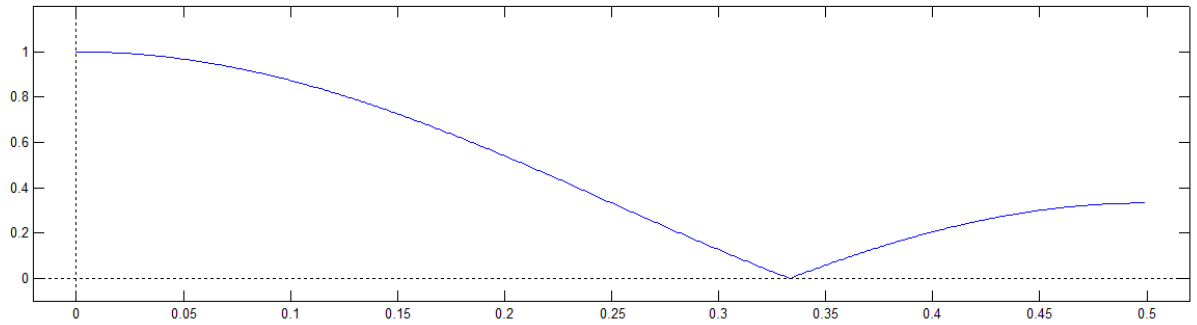
Program 2

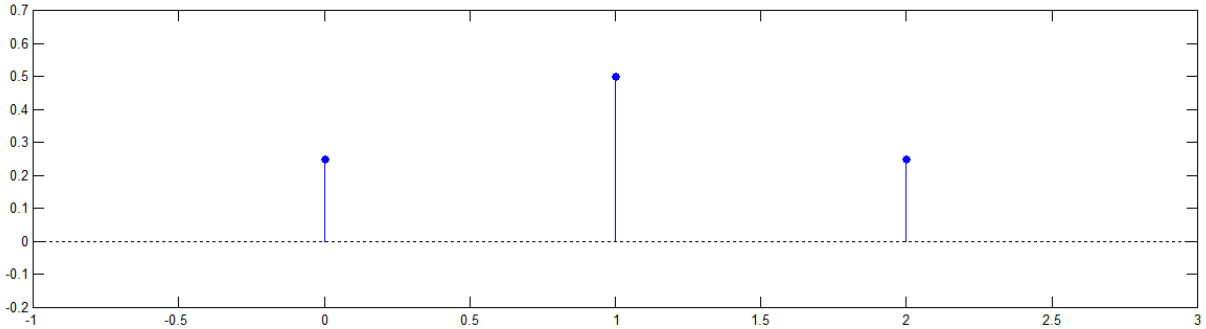
```
function h=wlsfilt(M,N,Weights)
% Calculates h(h)
% M = order
% N = number of points
% Weights = weights on error of N points
%
%  If no weight, use LOWESS weightm or remove comment for uniform weight
if nargin==2
   s=-1:2/(N+1):1;       % span
   ww=(1-abs(s).^3).^3; % tricubic
   Weights=ww(2:N+1)     % weights
   %  Weights=ones(1,N)
end


h=[];
for n=1:N
   x=zeros(1,N);
   x(n)=1 ;
      T=[];
   for t=0:length(x)-1
      for k=M:-1:0
          T(k+1,t+1)=t^k;
      end
   end
   T=flipud(T);
   T=T'

   W=diag(Weights);
   a=inv(T'*W*T)*T'*W*x';  % Polynomial coefficients
   h(n)=polyval(a,(N-1)/2);
end
MAT=inv(T'*W*T)*T'*W
%
figure(1)
subplot(211)
stem([0:N-1],h)
hold on
plot([-2 N+2],[0 0],'k:')
hold off
mih= min(h);
if min(h)>0; mih=0;end
mih=mih-.2;
mxh= max(h)+.2;
axis([-1 N mih mxh])
%
subplot(212)
plot([0:.001:.499],abs(freqz(h,1,500)))
hold on
plot([-.1 .6],[0 0],'k:')
plot([0 0],[-.2 1.5],'k:')
hold off
axis([-.02 .52 -.1 1.2])
figure(1)
```

The second program wlsfilt computes the weighted-least-squares filter and has options for LOESS (LOWESS) weighting. The main output is the impulse response and the frequency response of the FIR filter that does this fit on a running basis, and thus is an alternative to moving-average or related smoothers. Fig. 3a and Fig. 3b show the previously claimed results of h=[1/3 1/3 1/3] with equal weighting and h=[1/4 1/2 1/4] when the center is weighted by 2 instead of by 1.



h=wlsfilt(1,3,[1 1 1]) Equal Weighting ⟶ Length-3 Moving Average Fig. 3a





h = wlsfilt(1,3,[1 2 1]) Center Weighted Times 2 Fig. 3b

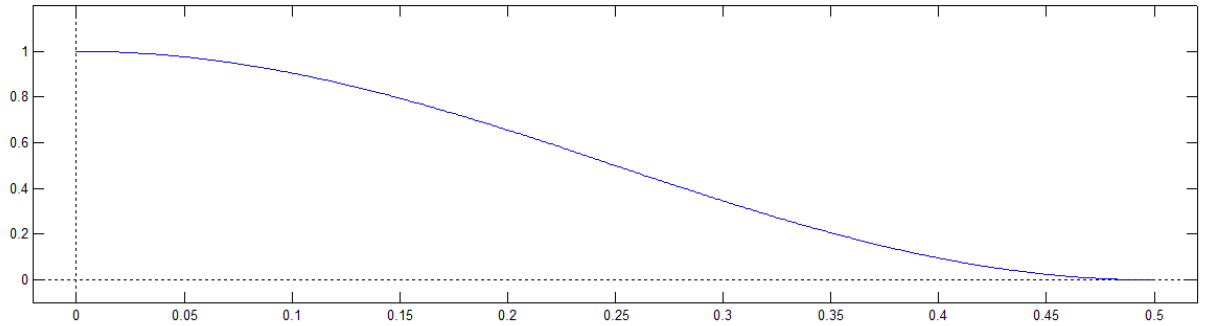
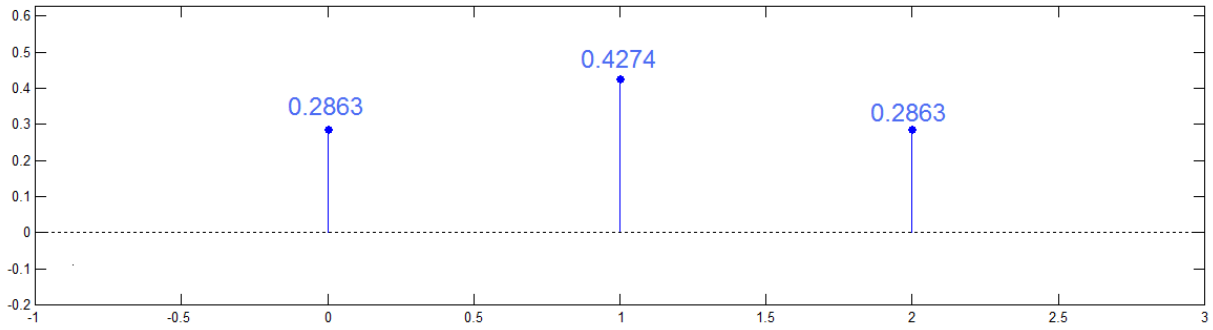
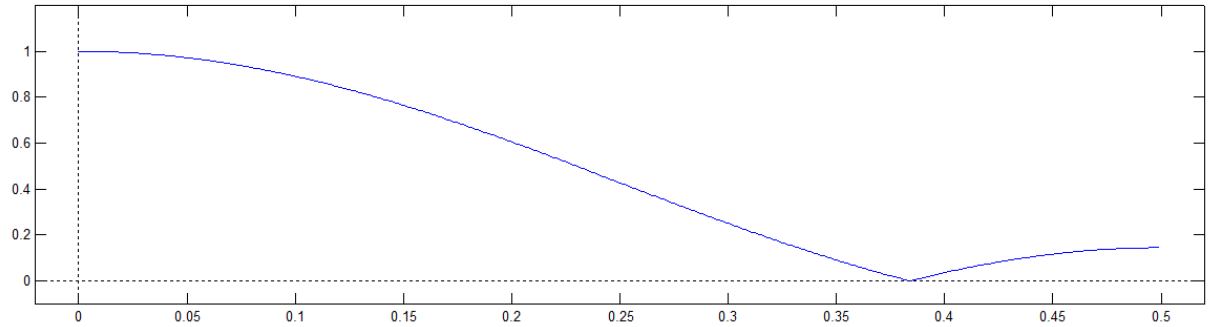


Fig. 3c is actually the default of our program. If you do not actually include a weighting, the program interprets this as a request for LOESS weighting. We are of course limited by what we can properly infer from length-3 examples, but it is clear that there is not much dramatically happening here.



h = wlsfilt(1,3) Default LOESS Weights [0.6699 1.0000 0.6699]

Fig. 3c



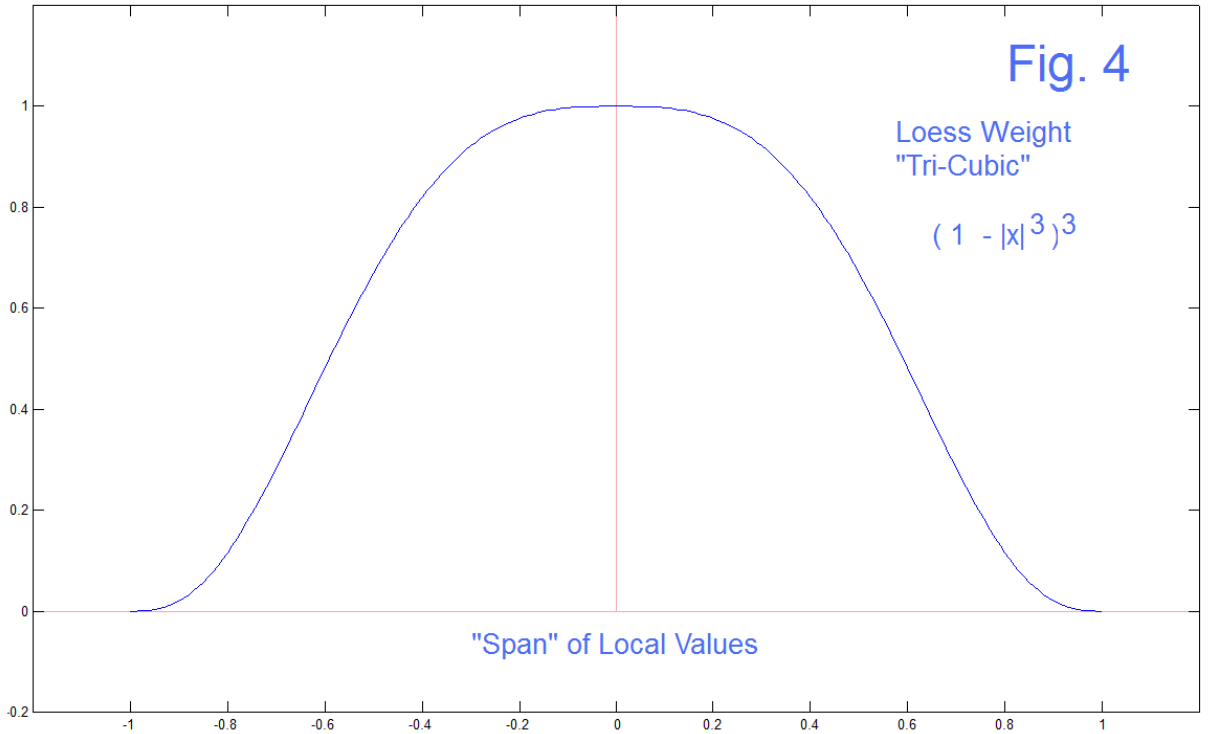
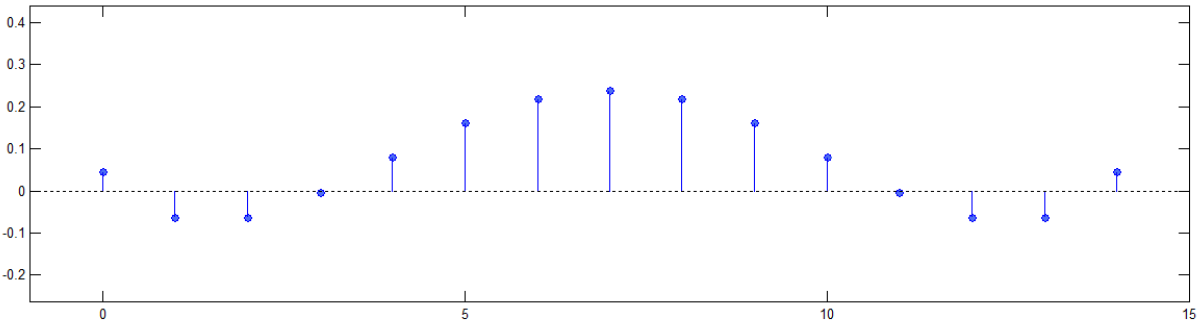
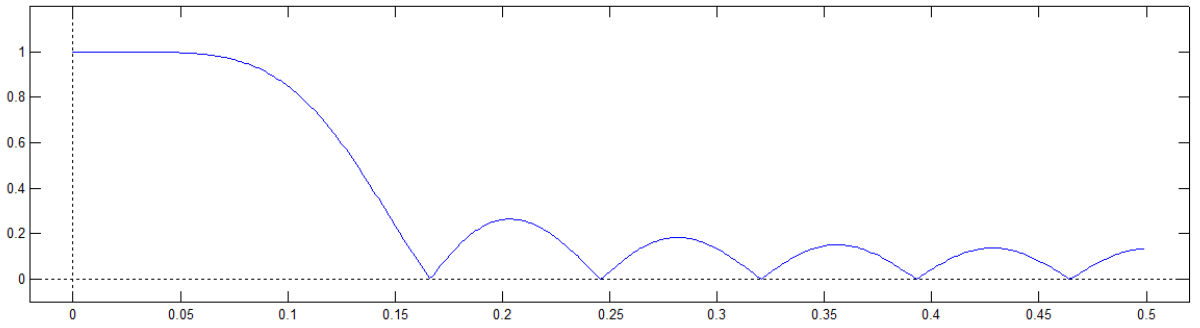


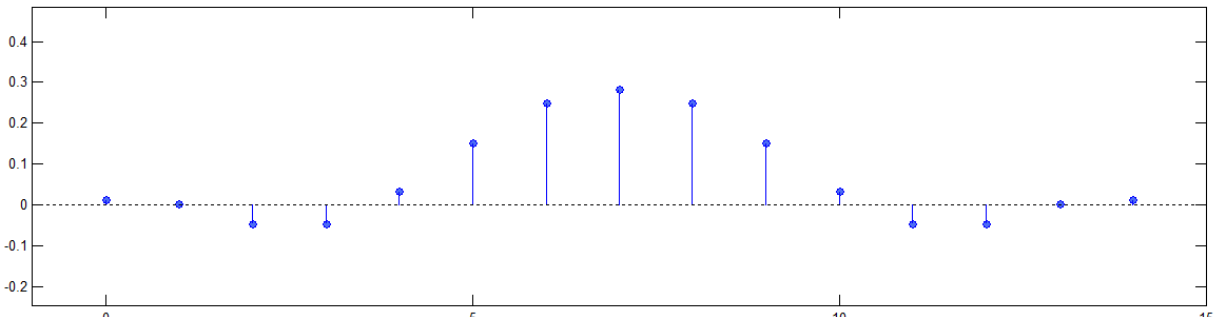
Fig. 4 shows how the LOESS weigh curve (called “tri-cubic”) is calculated. This curve is then “sampled” for the number of points in the fit (7 in Fig. 2e, 3 in Fig. 3c). Since the weight curve is 0 at -1 and at +1, we have chosen not to have samples there. If you wish, consider the number of samples as two longer, and these are cut by the weight curve. See the program code if this is not clear.



h = wlsfilt(5, 15, ones(1,15)) Same as Savitzky-Golay

Fig. 5a





h=wlsfilt(5,15)

Fig. 5b

LOESS Weights

0.0360 0.1932 0.4318 0.6699 0.8500 0.9539 0.9942 1 0.9942 0.9539 0.8500 0.6699 0.4318 0.1932 0.0360

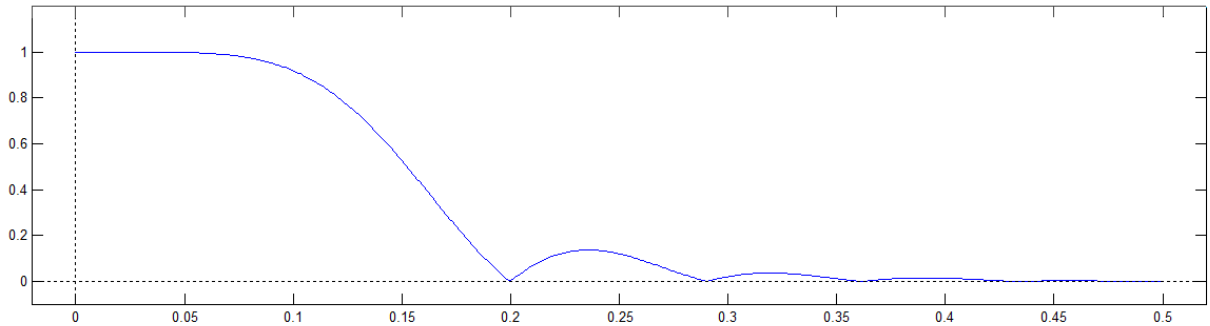


Fig. 5a shows an example for length-15, order-5 filter with equal weighting. This is simply the least squares we also saw with Savitzky-Golay. In fact, the exact same impulse response of obtained with the program h=sg(5,15) from AN-404 [2]. This leaves Fig. 5b, which uses the LOESS weights. The results are quite typical looking – similar to those we obtain with windowing such as Hamming [6].

Here we are unsure what advantages, if any, come from the LOESS (LOWESS) approach. It does appear that the method (as implemented in some software packages, but not very well documented) does deal with end effects by modifying the weight window at the ends in what seems to be a useful way. In such an eventuality, the implementation would likely not be FIR.

REFERENCES

[1] B. Hutchins "Time-Series Smoothing - A Review " ***Electronotes***, Vol. 23, No. 223, November 2014
http://electronotes.netfirms.com/EN223.pdf

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

[3] B. Hutchins, "Time domain Least Squared Low-Pass Filters," Electronotes App. Note No. 318, Feb. 1992
http://electronotes.netfirms.com/AN318.PDF

[4] Jackson, L.B., "Filter Design by Modeling," Chapter 10 of Digital Filters and Signal Processing (2nd Ed), Kluwer (1989). See pp 249-252 for concise and readable presentation.

[5] B. Hutchins, "Basic Elements of Digital Signal Processing: Filter Elements – Part 2" ***Electronotes***, Vol. 20, No. 198, June 2001, Section 3b
http://electronotes.netfirms.com/EN198.pdf

[6] B. Hutchins, "Basic Elements of Digital Signal Processing: Filter Elements – Part 1" ***Electronotes***, Vol. 20, No. 197, April 2001
http://electronotes.netfirms.com/EN197.pdf