Savitzky Golay Filter

Josef Hoffmann

Contents

1 Savitzky Golay Filter

1.1 Introduction

The Savitzky-Golay filter is a mathematical smoothing filter that is often used in signal processing. It was first described in 1964 by Abraham Savitzky and Marcel Golay [1].

The filter uses a polynomial regression over a series of values to find a smoothed value. One advantage of the Savitzky-Golay filter is that, unlike other smoothing filters, high-frequency components are not simply cut off, but are included in the calculation. As a result, the filter shows excellent properties with regard to the relative maxima, minima and scatter.

The filter was originally intended for use in spectroscopy to suppress the noise component. The filter is now used not only for polynomial smoothing but also for determining smoothed derivatives of a sequence.

For this filter, a window range of the sampled values is approximated with a polynomial and the current filtered value is determined from this. After that, the window is shifted by one step and a new polynomial is estimated to determine another filtered value.

In Fig. 1a a range of samples from $-m_L$ to m_R is shown around the current value at $n = 0$. The x-coordinate with the normalized values $n = t/T_s$ is generally valid, regardless of the respective sampling period T_s of the application. In the picture the samples $u[n], -m_L \leq n \leq m_R$ are marked with `o` and the values $x[n]$ estimated via a polynomial of degrees k are marked with `x`. The dashed line representation of the function $x(t/T_s)$ better suggests the smoothing effect of the polynomial approximation.

The model of the samples is thus given by the following equation:

$$\begin{aligned} &u[n] = x[n] + \epsilon[n] \quad \text{with} \\ &x[n] = a_k n^k + a_{k-1} n^{k-1} + a_{k-2} n^{k-2} + \cdots + a_1 n + a_0 \\ &\text{and} \quad -m_L \leq n \leq m_R \end{aligned} \tag{1}$$

The difference between the values $x[n]$ approximated with the polynomial and the samples $u[n]$ was denoted by $\epsilon[n]$.

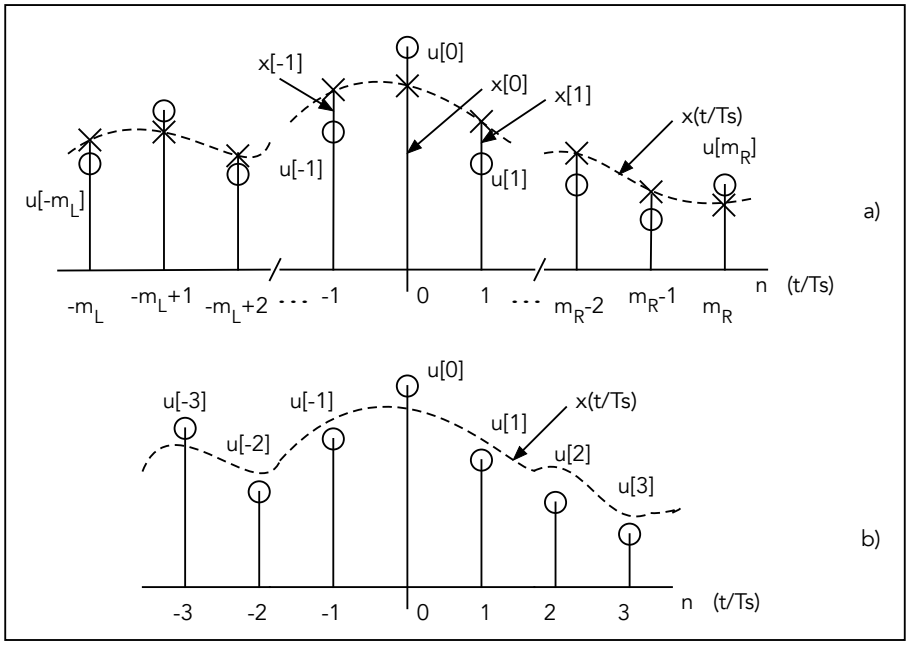

Fig. 1: a) Range of samples from $-m_L$ to m_R. b) Symmetrical range from -3 to 3

To determine the $k+1$ coefficients of the polynomial degree k, equation 1 is written for all samples in the window range with $-m_L \leq n \leq m_R$:

$$\begin{aligned} &u[m_R] = a_k (m_R)^k + a_{k-1} (m_R)^{k-1} + \cdots + a_1 (m_R) + a_0 + \epsilon[m_R] \\ &\cdots \\ &u[0] = a_0 + \epsilon[0] \\ &\cdots \\ &u[-m_L] = a_k (-m_L)^k + a_{k-1} (-m_L)^{k-1} + \cdots + a_1 (-m_L) + a_0 + \epsilon[-m_L] \end{aligned} \tag{2}$$

Written in a matrix form we get:

$$\begin{bmatrix} u[m_R] \\ \vdots \\ u[0] \\ \vdots \\ u[-m_L] \end{bmatrix} = \begin{bmatrix} m_R^k & m_R^{k-1} & \dots & m_R & 1 \\ \vdots & & & & \\ 0 & 0 & \dots & 0 & 1 \\ \vdots & & & & \\ -m_L^k & -m_L^{k-1} & \dots & -m_L & 1 \end{bmatrix} \begin{bmatrix} a_k \\ a_{k-1} \\ \vdots \\ a_1 \\ a_0 \end{bmatrix} + \begin{bmatrix} \epsilon[m_R] \\ \vdots \\ \epsilon[0] \\ \vdots \\ \epsilon[-m_L] \end{bmatrix} \quad (3)$$

With $\mathbf{u}$ as the vector of samples, $\mathbf{a}$ as the vector of polynomial coefficients, and ϵ as the vector of interpolation errors, the above equation is written in the following form:

$$\mathbf{u} = \theta\mathbf{a} + \epsilon \quad (4)$$

Minimizing the mean square error of ϵ leads to an equation for the coefficients of the polynomial given by [2]:

$$\mathbf{u} = \theta\mathbf{a} \quad (5)$$

If $k+1 = m_L + m_R + 1 = N$ then the matrix θ is square and the solution for $\mathbf{a}$ is very simple:

$$\mathbf{a} = \theta^{-1}\mathbf{u} \quad (6)$$

Usually the number of coefficients $k+1$ is less than the number of samples in the window area $m_L + m_R + 1$ and then the solution of the equation 5 is given by the pseudo-inverse of the matrix θ [2]:

$$\mathbf{a} = (\theta^T\theta)^{-1}\theta^T\mathbf{u} = \mathbf{A}\mathbf{u} \quad (7)$$

It should be noted that the matrix θ depends only on the time instances within the window. The matrix is a Vandermonde matrix [2] or a part of such a matrix. It has $m_L + m_R + 1$ rows and $k+1$ columns.

The estimated interpolated values according to the polynomial degree k are given through

$$x[n] = a_k n^k + a_{k-1} n^{k-1} + a_{k-2} n^{k-2} + \cdots + a_1 n + a_0 \quad (8)$$

For a selected window area only the value $x[n]$ for $n = 0$ is of interest and as you can see this is equal to a_0:

$$x[0] = a_0 = \mathbf{A}(N, :)\mathbf{u} \quad (9)$$

In the same way, the first derivative of the interpolated value in the selected window $x'[n], n = 0$ is equal to a_1.

$$x'[0] = a_1 = \mathbf{A}(N-1, :)\mathbf{u} \quad (10)$$

Similarly, the other coefficients, such as a_2, a_3, are the interpolated values for the second and third derivatives, respectively.

1.2 Example with a symmetrical window

A simple example with a symmetrical window with $m_L = m_R = 3$ shown in Fig. 1b is examined. There are thus 7 interpolation points from -3 to 3. First, a third degree polynomial with k= 3 is chosen, which has 4 coefficients $a_k, k = 3, 2, 1, 0$.

First the matrix θ is calculated:

$$\theta = \begin{bmatrix} 27 & 9 & 3 & 1 \\ 8 & 4 & 2 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ -1 & 1 & -1 & 1 \\ -8 & 4 & -2 & 1 \\ -27 & 9 & -3 & 1 \end{bmatrix} \tag{11}$$

In MATLAB, this matrix can be calculated with the following statements:

```
v = [3:-1:-3];
theta_1 = vander(v);
```

The matrix `theta_1` is the Vandermonde matrix of size 7x7

```
theta_1 =
   729   243    81    27     9     3     1
    64    32    16     8     4     2     1
     1     1     1     1     1     1     1
     0     0     0     0     0     0     1
     1    -1     1    -1     1    -1     1
    64   -32    16    -8     4    -2     1
   729  -243    81   -27     9    -3     1
```

from which the desired matrix `theta` is extracted:

```
theta = theta_1(:,4:end);
```

The equation 5 for this example becomes:

$$\begin{bmatrix} u[3] \\ u[2] \\ u[1] \\ u[0] \\ u[-1] \\ u[-2] \\ u[-3] \end{bmatrix} = \begin{bmatrix} 27 & 9 & 3 & 1 \\ 8 & 4 & 2 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ -1 & 1 & -1 & 1 \\ -8 & 4 & -2 & 1 \\ -27 & 9 & -3 & 1 \end{bmatrix} \begin{bmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} = \theta \mathbf{a} \tag{12}$$

The pseudo-inverse of the matrix θ denoted by $\mathbf{A}$ is in MATLAB determined with:

```
A = pinv(theta);      % or
% A = inv(theta' * theta) * theta';
```

One receives:

```
A =
    0.0278   -0.0278   -0.0278         0    0.0278    0.0278   -0.0278
    0.0595         0   -0.0357   -0.0476   -0.0357         0    0.0595
   -0.0873    0.2659    0.2302         0   -0.2302   -0.2659    0.0873
   -0.0952    0.1429    0.2857    0.3333    0.2857    0.1429   -0.0952
```

According to equation 7, the coefficients of the interpolation polynomial are obtained by multiplying the matrix $\mathbf{A}$ with the sample values $\mathbf{u}$ from the window area:

$$\mathbf{a} = \mathbf{A}\mathbf{u}$$

The desired interpolated value $x[0]$ equal to the coefficients a_0 (according to eq. 9) is the last coefficient in the vector $\mathbf{a}$ and is calculated as the dot product between the last row of the Matrix $\mathbf{A}$ and the vector $\mathbf{u}$:

$$a_0 = x[0] = \mathbf{A}(\mathbf{4}, :)\mathbf{u} \tag{13}$$

This operation is similar to filtering with an FIR filter and the mirrored last row of the matrix $\mathbf{A}$ is the impulse response of this filter. For the present example, the impulse response is symmetrical and represents a low-pass filter.

After the interpolated value in a window has been calculated, the window is shifted with one sampling period and a new interpolated value is determined, as with filtering with an FIR filter.

If the number $k + 1$ of the coefficients of the interpolation polynomial is equal to the window size $k + 1 = m_L + m_R + 1$, then the matrix θ is square and the pseudoinverse becomes simply the inverse of this Matrix, which is the full Vandermonde Matrix.

In the example given, with $k = 6$, a sixth-degree polynomial with 7 coefficients for the 7 support points in the window, the matrix θ is the full Vandermonde matrix shown above. Their inverse is now equal to:

```
    0.0014   -0.0083    0.0208   -0.0278    0.0208   -0.0083    0.0014
    0.0042   -0.0167    0.0208    0.0000   -0.0208    0.0167   -0.0042
   -0.0069    0.0833   -0.2708    0.3889   -0.2708    0.0833   -0.0069
   -0.0208    0.1667   -0.2708   -0.0000    0.2708   -0.1667    0.0208
    0.0056   -0.0750    0.7500   -1.3611    0.7500   -0.0750    0.0056
    0.0167   -0.1500    0.7500    0.0000   -0.7500    0.1500   -0.0167
         0         0         0    1.0000         0         0         0
```

The last row as the impulse response of the FIR filter shows that this interpolation filter gives values equal to the sample values in the window. It is a limit filter. There is no smoothing effect. This leads to the conclusion that the number of coefficients of the interpolation polynomial must be smaller than the number of samples in the window area.

With some MATLAB functions one can get from this limit value filter the symmetric filter designed before:

```
hi = [0 0 0 1 0 0 0];          % The limit symmetrical filter
k = 3;                         % Polynomial Degree
```

```
ah = polyfit(-3:3, hi, k);      % Approach with polynomial
hFIR = polyval(ah, -3:3);
```

You get:

```
hFIR =
-0.0952    0.1429    0.2857    0.3333    0.2857    0.1429   -0.0952
```

This result is equal to the last row from the matrix `A` calculated earlier as the symmetric FIR filter.

With a symmetrical filter there are always errors at the beginning and at the end when filtering a sequence. With this filter with 7 coefficients, the first 3 and the last 3 values are in error. For these 3 values, you can select a non-symmetrical arrangement in the window and determine the corresponding interpolation. In this simple example, the 3 limit filters needed for interpolation at the beginning of a sequence would be:

```
h1i = [1 0 0 0 0 0 0];
h2i = [0 1 0 0 0 0 0];
h3i = [0 0 1 0 0 0 0];
```

Similarly, the 3 limit filters for the end of a sequence would be:

```
h1o = [0 0 0 0 0 0 1];
h2o = [0 0 0 0 0 1 0];
h3o = [0 0 0 0 1 0 0];
```

As an example, starting from `h2i` the corresponding filter is calculated with:

```
h2i = [0 1 0 0 0 0 0];           % The limit symmetrical filter
k = 3;                           % Polynomial Degree
ah = polyfit(-3:3, h2i, k);      % Approach with polynomial
h2iFIR = polyval(ah, -3:3);
```

You get:

```
h2iFIR =
0.1905    0.4524    0.3810    0.1429   -0.0952   -0.1667    0.0952
```

MATLAB provides the function `sgolay` with which you can determine this type of filters. For this example with the call

```
h = sgolay(3,7);
```

you get the following result:

```
h =
    0.9286    0.1905   -0.0952   -0.0952    0.0238    0.0952   -0.0476
    0.1905    0.4524    0.3810    0.1429   -0.0952   -0.1667    0.0952
   -0.0952    0.3810    0.4524    0.2857    0.0476   -0.0952    0.0238

   -0.0952    0.1429    0.2857    0.3333    0.2857    0.1429   -0.0952

    0.0238   -0.0952    0.0476    0.2857    0.4524    0.3810   -0.0952
    0.0952   -0.1667   -0.0952    0.1429    0.3810    0.4524    0.1905
   -0.0476    0.0952    0.0238   -0.0952   -0.0952    0.1905    0.9286
```

The first three rows contain the filters for correctly determining the first 3 values of the filtered sequence and the last three rows contain the corresponding filters for correctly determining the last 3 values of the filtered sequence. The line in the middle shows the symmetric filter `hFIR`. The second line represents the filter that was also determined above.

In the MATLAB script `sav_golay_2.m` the simple example is examined and used for smoothing a sequence with noise:

```
% Script sav_golay_2.m, in which the simple example is examined
clear;
% ------- Initializations
k = 3;              % Degree of the polynomial (1 <= k <= 6)
n = 3:-1:-3;        % Window area with length = 7
s = 97531;          % Seed for random sequences
rng(s);

nd = 3:-0.1:-3;     % Window area for continuous plots

x  = cos(2*pi*n/6+pi/3);  % Smoothed sequence in the window
xd = cos(2*pi*nd/6+pi/3); % Smoothed continuous function
noise = 0.3;              % Strength of the noise
u_n = x + noise*randn(1,length(n));  % Samples in the window

% ------- FIR Savitzky-Golay filter
v = 3:-1:-3;          % Vandermonde parameter
theta_1 = vander(v);
theta = theta_1(:,7-k:end);

%A = pinv(theta);
A = inv(theta'*theta)*theta',

hFIR = A(k+1,:);      % Impulse response of the FIR filter
x0 = hFIR*u_n'        % The value interpolated in the window

a = A*u_n';           % Interpolation polynomial coefficients

xe = polyval(a, nd);  %

figure(1);      clf;
subplot(211), stem(n, u_n,'k-','LineWidth',1);
hold on;
plot(nd, xd,'k--');
La = axis;   axis([-3.2, 3.2, La(3:4)]);
grid on;
title('The samples and the smoothed function in the window area');

subplot(212),  stem(n, u_n,'k-','LineWidth',1);
hold on;
plot(nd, xd,'k--');
La = axis;   axis([-3.2, 3.2, La(3:4)]);
```

```
grid on;
plot(nd, xe, 'k-');
stem(0, x0,'k','LineWidth',2.5)
title(['The samples, the smoothed function, the interpolated function'...
       ' and its value']);

% -------- Filtering a sequence
Ts = 0.5;
t = 0:Ts:200;
x = 2*cos(2*pi*t/10);              % Signal

u = x + 1.5*randn(1, length(x));  % Signal plus noise
uFIR = filter(hFIR,1,u);           % Filterung

figure(2);   clf;
subplot(211), stairs(t+3*0.5,u,'k-','LineWidth',1);
La = axis;   axis([La(1), 200, La(3:4)]);
grid on;
title('Cosine signal plus noise')

subplot(212), stairs(t,uFIR,'k-','LineWidth',1);
grid on;
title(['Filtered signal for degree interpolation polynomial k = '...
     ,num2str(k)])
```

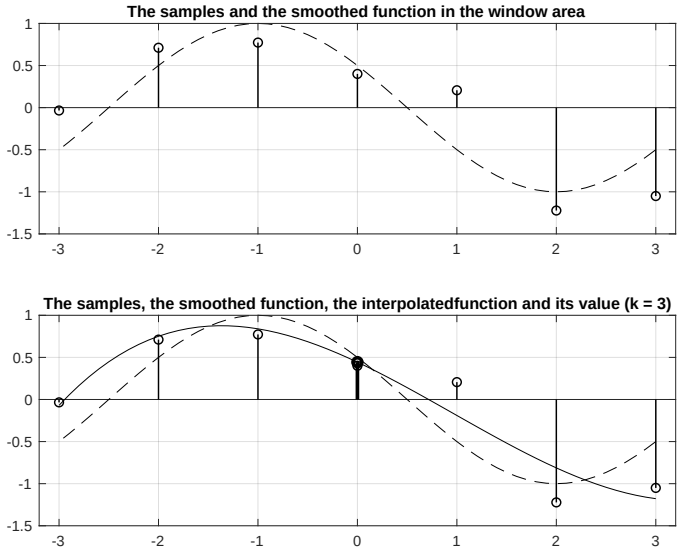

Fig. 2: a) The samples and the smoothed function in the window area b) The samples, the smoothed function, the interpolated function and its value

The script contains all of the previous MATLAB commands used to explain the Savitzky Golay filter issues. The signal in the window area consists of a smoothed part in the form of a cosine function and a part of random Gaussian distributed values. The 7 samples $u[n]$ and the cosine function, which gives the smoothed values $x[n]$, are shown in Fig. 2 above.

Below in Fig. 2, the sampled values $u[n]$ and the cosine function together with the interpolated function, shown as a continuous function, are displayed. The interpolated value for this window $x[n]$ for $n = 0$ is shown in bold. The representation in Fig. 2 corresponds to an interpolation polynomial of the third degree. For $k = 1$, which means a polynomial of degree 1, the interpolated function in the window will be a linear function, as shown in Fig. 3.

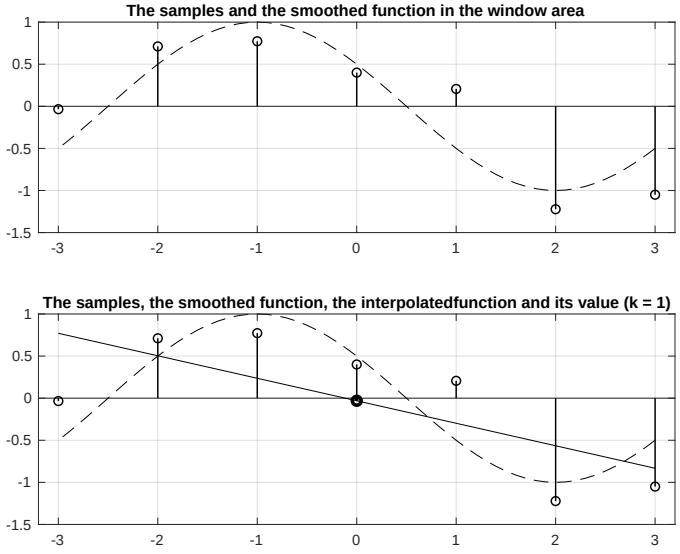

Fig. 3: a) The samples and the smoothed function in the window area b) The samples, the smoothed function, the interpolated linear function and its value

With $k = 6$, which corresponds to a sixth degree polynomial and means the highest value for the 7-sample window, one does not get smoothing because the values of the interpolation polynomial $x[n]$ are equal to the samples in the window $u[n]$. This case is shown in Fig. 4. The values of the interpolation function between the samples are irrelevant here. They are shown to see how this function involves the samples.

In the last part of the script, the filtering of a sequence consisting of a cosine function, as a smoothed portion, plus normally distributed noise is simulated. For $k = 6$, as expected, there is no smoothing effect as shown in Fig. 5. In Fig. 6 the same input/output sequences are shown for an interpolation polynomial of degree $k = 3$.

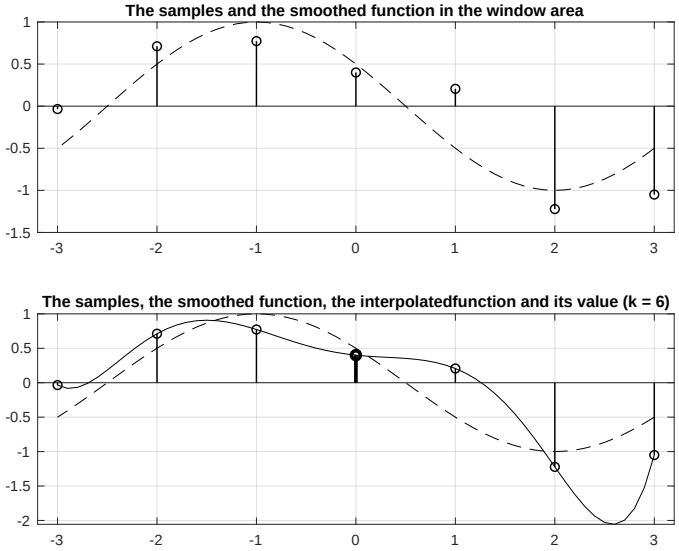

Fig. 4: a) The samples and the smoothed function in the window area b) The samples, the smoothed function, the interpolated function and its value

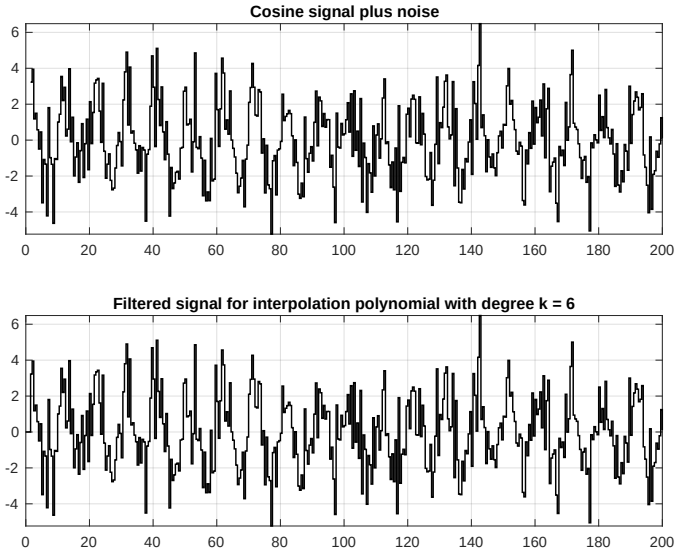

Fig. 5: a) The input sequences b) The output sequences for $k = 6$ (there is no smoothing)

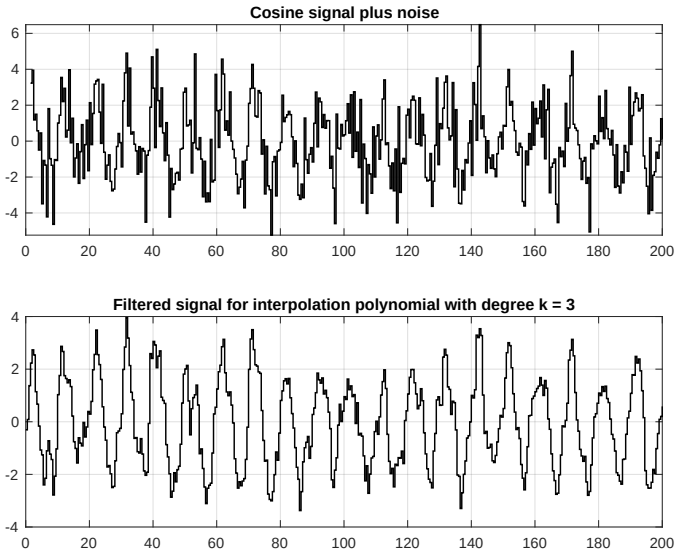

Fig. 6: a) The input sequences b) The output sequences for $k = 3$

1.3 Correction the beginning and the end of a filtered sequence

Errors occur at the beginning and at the end of a sequence filtered with a symmetrical Savitzky Golay FIR filter. Fig. 7b shows the position of the impulse response of the filter at the beginning and the end, which does not lead to errors. Here all coefficients of the filter are multiplied by values of the input signal and the output is correct and delayed with $(N-1)/2$ relative to the current input. The number of coefficients of the FIR filter (an odd number) was denoted by N. For the areas of this size at the beginning and at the end, you have to calculate the interpolation with asymmetrical filters.

In MATLAB with the function `sgolay` you get the filters for the interpolation at the beginning, the symmetrical filter for the interpolation in the central area and the filters for the interpolation at the end. Using a small script `sav_golay_3`, the application of the function `sgolay` for an input sequence of the form shown in Fig. 7a is explained as an example. This form is very well suited to evaluating the correction values.

The window size, which is also the length of the filters N, is chosen to be 13 in this example. This means that the correction at the beginning and at the end extends to 6 sampling intervals each. The degree of the interpolation polynomial k can be chosen between 1 and 12 and you can experiment with these parameter in the script, especially if you also add noise to the input signal using the variable `noise`.

Initially, the input signal is used without noise (`noise = 0`) in order to view and evaluate the correction of the beginning and the end during the filtering. Calling the function `b = sgolay(k,N)` with $N = 13$ and $k = 4$ results in a

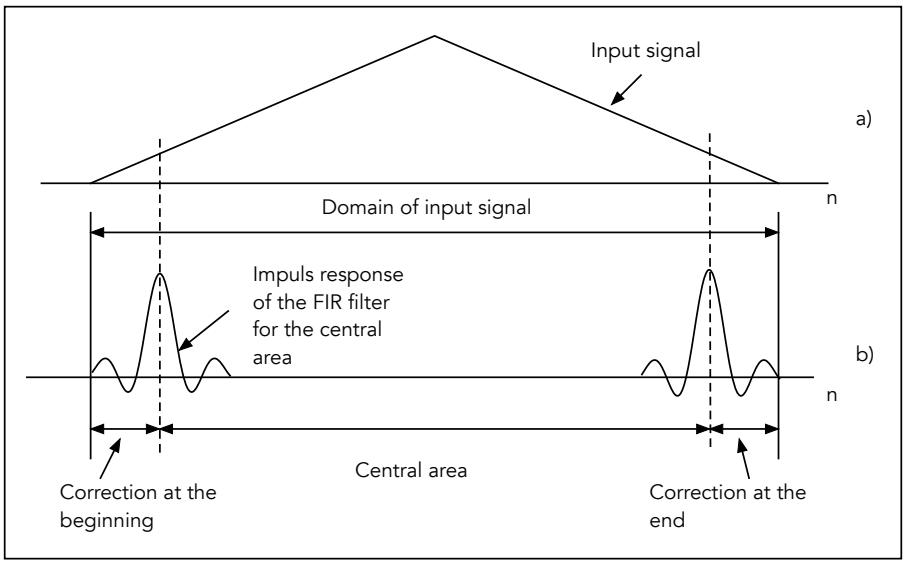

Fig. 7: a) The input sequences b) The correction of the beginning and of the end

matrix with 13 rows and 13 columns. The first 6 rows (`b(1:6,:)`) form the asymmetrical filters for the correction of the beginning. The middle row represents the symmetrical filter for filtering the central part of the input signal and the last 6 rows form the unbalanced filters for correction of the end. With the middle row separated and with row lengths truncated, the matrix `b` looks like this:

```
b =
    0.8720    0.2666   -0.0242   -0.1115   -0.0856   -0.0168    0.0452   ...
    0.2666    0.3387    0.2909    0.1842    0.0663   -0.0291   -0.0814   ...
   -0.0242    0.2909    0.3652    0.2979    0.1688    0.0376   -0.0555   ...
   -0.1115    0.1842    0.2979    0.2936    0.2240    0.1310    0.0452   ...
   -0.0856    0.0663    0.1688    0.2240    0.2363    0.2121    0.1604   ...
   -0.0168   -0.0291    0.0376    0.1310    0.2121    0.2550    0.2468   ...

    0.0452   -0.0814   -0.0555    0.0452    0.1604    0.2468    0.2785   ...

    0.0711   -0.0840   -0.0903   -0.0138    0.0923    0.1880    0.2468   ...
    0.0517   -0.0436   -0.0660   -0.0377    0.0210    0.0923    0.1604   ...
   -0.0016    0.0194   -0.0018   -0.0291   -0.0377   -0.0138    0.0452   ...
   -0.0572    0.0711    0.0635   -0.0018   -0.0660   -0.0903   -0.0555   ...
   -0.0630    0.0640    0.0711    0.0194   -0.0436   -0.0840   -0.0814   ...
    0.0533   -0.0630   -0.0572   -0.0016    0.0517    0.0711    0.0452   ...
```

The values in the middle row are symmetrical and represent an impulse response of a low-pass filter in the form of a sinc function, as suggested in Fig. 7b.

The multiplication of the partial matrix consisting of the first 6 rows of `b` with the first 13 values of the input signal `yi = b(1:6,:)*u_n(1:13)'`

results in the 6 corrected values at the beginning. Similarly, multiplying the last 6 rows of `b` by the last 13 values of the input signal leads to the 6 corrected values at the end `yo = b(8:end,:)*u_n(101-13:end)'`. With `ycenter = conv(u_n, b(7,:),'valid');` the central area of the input signal is filtered. The output signal is then with `y = [yi', ycenter, yo'];` obtained.

In Fig. 8 the corrected values at the beginning and at the end are shown at the top. Below is the filtered central area and at the very bottom the composition of these signals.

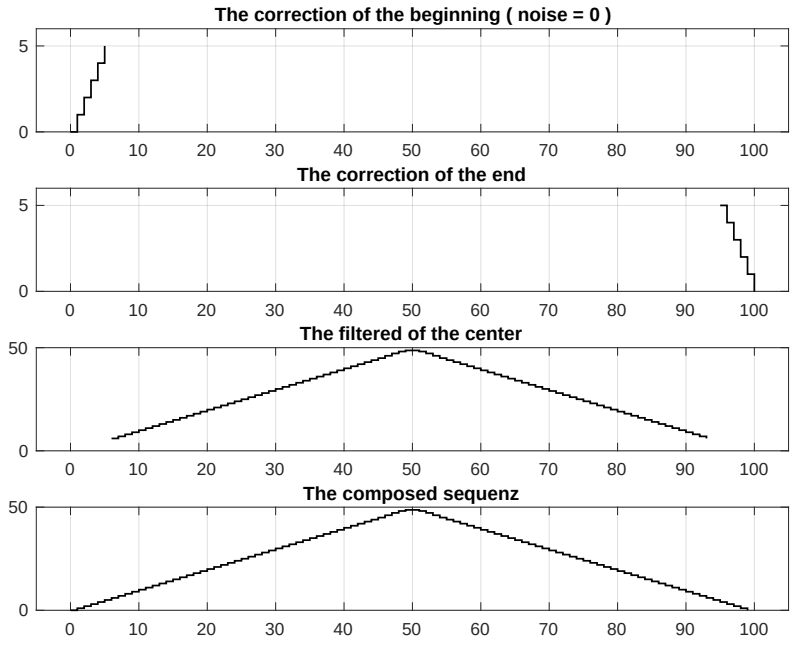

Fig. 8: a) The corrected values at the beginning. b) The corrected values at the end. c) The filtered central region and d) The composition of the signals

As you can see, the values for the beginning and the end have been determined correctly. The reader can continue to superimpose noise on the input signal (e.g. with `noise = 2`) and restart the simulation.

The script also determines and displays the impulse response and the amplitude response of the symmetrical filter, as shown in Fig. 9 for $k = 4$. For $k = 1$, which corresponds to a degree one interpolation polynomial with only two coefficients, the filter is the averaging filter over the whole window, with a constant impulse response, as shown in Fig. 10.

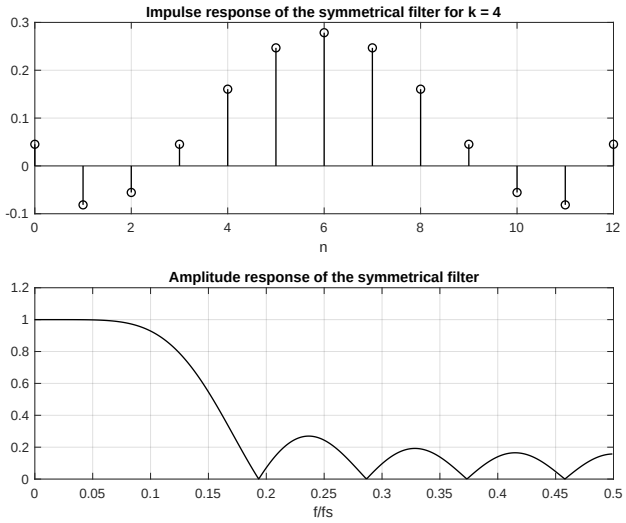

Fig. 9: a) Impulse response of the symmetrical filter for k = 4. b) The corresponding amplitude response

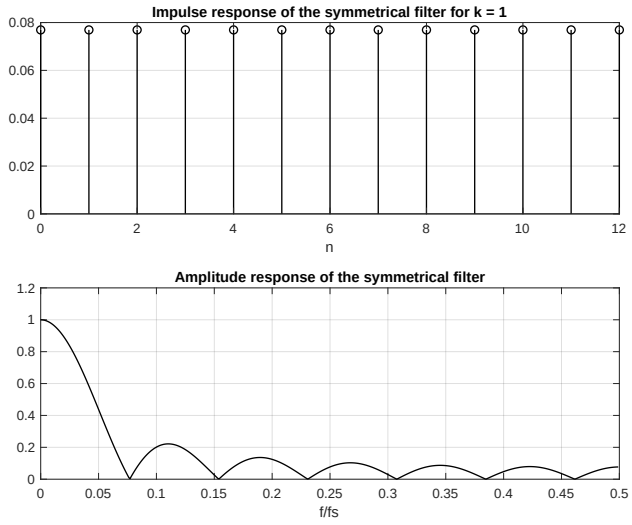

Fig. 10: a) Impulse response of the symmetrical filter for k = 1. b) The corresponding amplitude response

Calling the MATLAB function `sgolay` in the form `[h,g] = sgolay(k,N)` also supplies the smoothed derivatives of the input signal in the matrix `g`. The first column of this matrix gives the impulse response of the symmetrical filter viewed as a zero order derivative. The second column gives the impulse response of the filter for the first derivative and the other columns give the filters for the

other derivatives similarly. For the example from the script `sav_golay_3.m`, the call `[h,g]=sgolay(3,13)` results in the following matrix `g`:

```
g =
   -0.0769    0.0472    0.0110   -0.0032
    0.0000   -0.0275    0.0055    0.0000
    0.0629   -0.0657    0.0010    0.0017
    0.1119   -0.0748   -0.0025    0.0023
    0.1469   -0.0620   -0.0050    0.0020
    0.1678   -0.0346   -0.0065    0.0012
    0.1748    0.0000   -0.0070   -0.0000
    0.1678    0.0346   -0.0065   -0.0012
    0.1469    0.0620   -0.0050   -0.0020
    0.1119    0.0748   -0.0025   -0.0023
    0.0629    0.0657    0.0010   -0.0017
    0.0000    0.0275    0.0055    0.0000
   -0.0769   -0.0472    0.0110    0.0032
```

The third degree polynomial ($k = 3$) can give the three filters for the three derivatives in the form of the last three columns of the matrix g. The impulse response of the first derivative filter is given by the mirrored form of the second columns:

```
h1 = flipud(g(:,2));
```

The mirroring is necessary because the convolution in the filtering mirrors the impulse response and thus gives the correct multiplications.

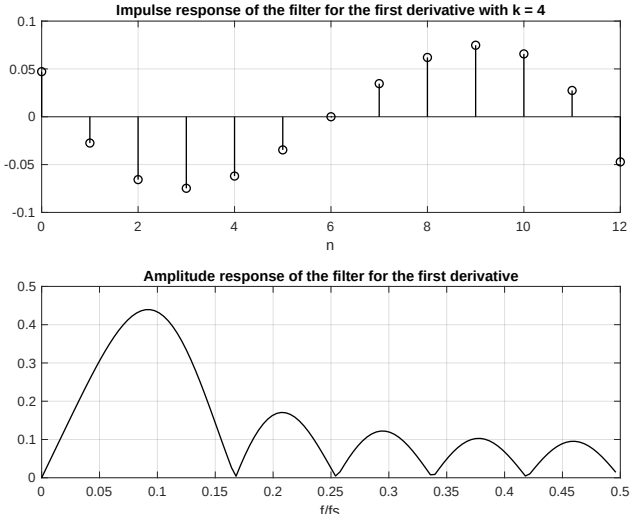

Fig. 11: a) Impulse response of the filter for the first derivative with $k = 4$ b) The corresponding amplitude response

With `y = conv(u_n,h1,'valid');` the first derivative of the central area of

the input signal is determined and then as in Fig. 12 for `noise = 0` and `k = 3` shown.

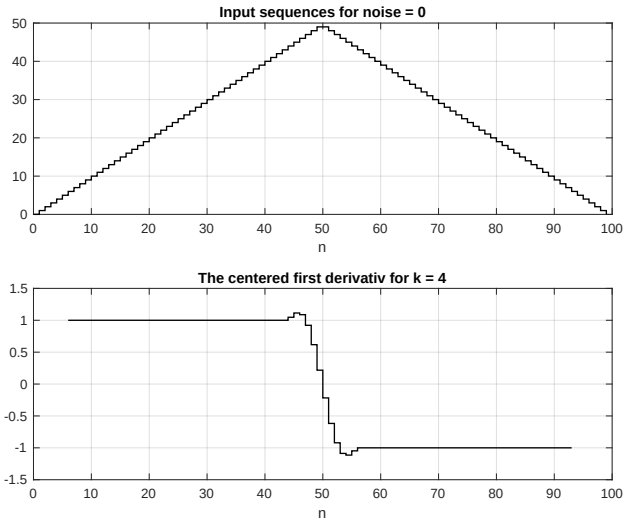

Fig. 12: a) Input sequence b) The first derivative of the central domain for k = 3 and noise = 0

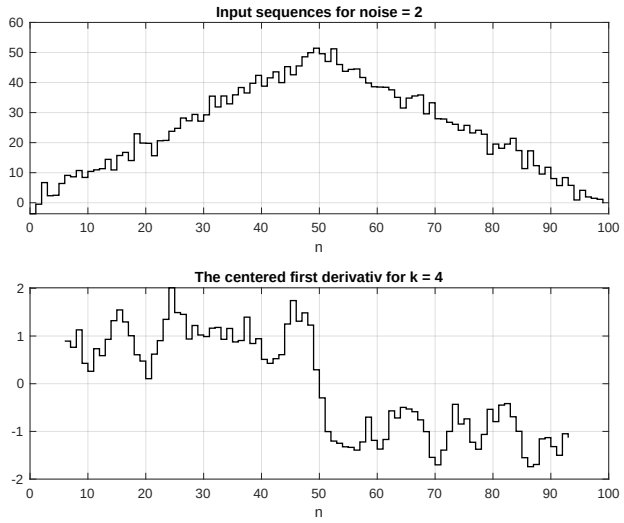

Fig. 13: a) Input sequence b) The first derivative of the central domain for k = 3 and noise = 2

The same derivation for `noise = 2` and `k = 2` is shown in Fig. 13. The reader

is advised to experiment here with the parameters `noise` and `k` and interpret and evaluate the results.

In the `sgolay` function in MATLAB there is no indication how to correct the start and end of derivatives.

```
% Script sav_golay_3.m, in which the composing of the beginning with the
% center and the end in the savitzky golay filter is examined
clear;
% ------- Initializations
s = 97531;              % Seed for random sequences
rng(s);
xd = [0:49, 49:-1:0];   % Smoothed continuous function (triangle)
n  = [0:length(xd)-1];  % Abscissa for the representations
noise = 0;              % Noise strength
u_n = xd + noise*randn(1,length(xd));  % Input signal
% ------- FIR Savitzky-Golay filter
% Window with N = mL + mR + 1 = 13 and polynomial degree k
N = 13;        k = 3 ;
b = sgolay(k,N),         % Savitzky Golay Filter
yi = b(1:6,:)*u_n(1:13)',     % Correction of the beginning
yo = b(8:end,:)*u_n(101-13:end)', % Correction of the end
ycenter = conv(u_n, b(7,:),'valid'); % Filtering the center
y = [yi', ycenter, yo'];      % The composing the filtered sequenz

figure(1),    clf;
subplot(411), stairs(0:5, yi, 'k-', 'LineWidth',1);
axis([-5,105,0,6]), grid on;
title(['The correction of the beginning ( noise = ',num2str(noise),' )']);
subplot(412), stairs([95:100], yo, 'k-', 'LineWidth',1);
axis([-5,105,0,6]), grid on;
title('The correction of the end');
subplot(413), stairs(6:100-7, ycenter, 'k-', 'LineWidth',1);
axis([-5,105,0,50]), grid on;
title('The filtered of the center');
subplot(414), stairs(n, y, 'k-', 'LineWidth',1);
axis([-5,105,0,50]), grid on;
title('The composed sequenz');

% ------ Impulse and frequency response of the symmetrical Filter
[H,f] = freqz(b(7,:),1);
f = f/(2*pi);       % or
% nfft = 256;
% H = fft(b(7,:),nfft);    f = (0:nfft/2-1)/nfft;
% H = H(1:nfft/2);

figure(2),    clf;
subplot(211), stem(0:12, b(7,:),'k-', 'LineWidth',1);
title(['Impulse response of the symmetrical filter for k = ',num2str(k)])
xlabel('n');
grid on;
```

```
subplot(212), plot(f, abs(H),'k-', 'LineWidth',1);
title('Amplitude response of the symmetrical filter')
xlabel('f/fs');
grid on;

% ------ Filter for the derivatives
[h,g] = sgolay(k,N);

% Impulse response and frequency response for the first derivative
nfft = 256;
h1 = flipud(g(:,2));
H1 = fft(h1,nfft);    f = (0:nfft/2-1)/nfft;
H1 = H1(1:nfft/2);

figure(3),    clf;
subplot(211), stem(0:12, h1,'k-', 'LineWidth',1);
title(['Impulse response of the filter for the first derivative'...
' with k = ',num2str(k)])
xlabel('n');
grid on;
subplot(212), plot(f, abs(H1),'k-', 'LineWidth',1);
title('Amplitude response of the filter for the first derivative')
xlabel('f/fs');
grid on;

% ------- First smoothed derivative of the input sequence
y = conv(u_n,h1,'valid');

figure(4);    clf;
subplot(211), stairs(n,u_n,'k-', 'LineWidth',1);
title(['Input sequences for noise = ',num2str(noise)]);
xlabel('n');   grid on;
subplot(212), stairs((0:length(y)-1)+6,y,'k-', 'LineWidth',1);
title(['The centered first derivativ for k = ',num2str(k)]);
xlabel('n');   grid on;
```

The MATLAB scripts are contained in the `sav_golay.zip` file, which can be accessed via the following link:
`https://dsprelated.com/blogimages/JosefHoffmann/sav_golay.zip`

References

[1] Sanjit K. Mitra and James F. Kaiser, editors. *Handbook for Digital Signal Processing.* John Wiley & Sons, 1993.

[2] Todd K. Moon, Wynn C. Stirling. *Mathematical Methods and Algorithms for Signal Processing.* Prentice Hall, 2000.