3

Local Polynomial Filters

3.1 Introduction

We mentioned in Sec. 2.4 that there are limits to the applicability of the plain FIR averager filter—in order to achieve a high degree of noise reduction, its length N may be required to be so large that the filter's passband becomes smaller than the signal bandwidth, causing the removal of useful high frequencies from the desired signal.

In other words, in its attempt to smooth out the noise v_n, the filter begins to smooth out the desired signal x_n to an unacceptable degree. For example, if x_n contains some short-duration peaks, corresponding to the higher frequencies present in x_n, and the filter's length N is longer than the duration of the peaks, the filter will tend to smooth the peaks too much, broadening them and reducing their height.

Local polynomial smoothing filters [36–99] are generalizations of the FIR averager filter that can preserve better the higher frequency content of the desired signal, at the expense of not removing as much noise as the averager. They can be characterized in three equivalent ways:

1. They are the optimal lowpass filters that minimize the NRR, subject to additional constraints than the DC unity-gain condition (2.4.1)—the constraints being equivalent to the requirement that polynomial input signals go through the filter unchanged.
2. They are the optimal filters that minimize the NRR whose frequency response $H(\omega)$ satisfies certain *flatness* constraints at DC.
3. They are the filters that optimally fit, in a least-squares sense, a set of data points to polynomials of different degrees.

Local polynomial smoothing (LPSM) filters have a long history and have been rediscovered repeatedly in different contexts. They were originally derived in 1866 by the Italian astronomer Schiaparelli [36] who formulated the problem as the minimization of the NRR subject to polynomial-preserving constraints and derived the filters in complete generality, discussing also the case of even-length filters. They were rederived in 1871 by De Forest [65] who generalized them further to include the case of "minimum-roughness" or minimum-R_s filters. Subsequently, they were rediscovered many times

and used extensively in actuarial applications, for example, by Gram, Hardy, Sheppard, Henderson, and others. See Refs. [68–75] for the development and history of these filters. In the actuarial context, smoothing is referred to as the process of "graduation." They were revived again in the 1960s by Savitzky and Golay [42] and have been applied widely in chemistry and spectroscopy [42–53] known in that context as *Savitzky-Golay filters*. They, and their minimum-R_s versions [65–99] known typically as *Henderson filters*, are used routinely for trend extraction in financial, business, and census applications.

Some recent incarnations also include predictive FIR interpolation, differentiation, fractional-delay, and maximally-flat filters [152–187], and applications to the representation of speech and images in terms of orthogonal-polynomial moments [137–150].

The least-squares polynomial fitting approach also has a long history. Chebyshev [104] derived in 1864 the discrete Chebyshev orthogonal polynomials,[‡] also known as Gram polynomials, which provide convenient and computationally efficient bases for the solution of the least-squares problem and the design of local polynomial filters. Several applications and reviews of the discrete Chebyshev orthogonal polynomials may be found in [104–151]. The minimum-R_s Henderson filters also admit similar efficient representations in terms of the Hahn orthogonal polynomials, a special case of which are the discrete Chebyshev polynomials. We discuss Henderson filters in Sec. 4.2 and orthogonal polynomial bases in Sec. 4.3.

3.2 Local Polynomial Fitting

We begin with the least-squares polynomial fitting approach. We assume that the signal model for the observations is:

$$y_n = x_n + v_n$$

where v_n is white noise and x_n is a smooth signal to be estimated. Fig. 3.2.1 shows five noisy signal samples $[y_{-2}, y_{-1}, y_0, y_1, y_2]$ positioned symmetrically about the origin. Later on, we will shift them to an arbitrary position along the time axis. Polynomial smoothing of the five samples is equivalent to replacing them by the values that lie on smooth polynomial curves drawn between the noisy samples. In Fig. 3.2.1, we consider fitting the five data to a constant signal, a linear signal, and a quadratic signal.

The corresponding smoothed values are given by the 0th, 1st, and 2nd degree polynomials defined for $m = -2, -1, 0, 1, 2$:

$$\begin{aligned} \hat{y}_m &= c_0 && \text{(constant)} \\ \hat{y}_m &= c_0 + c_1 m && \text{(linear)} \\ \hat{y}_m &= c_0 + c_1 m + c_2 m^2 && \text{(quadratic)} \end{aligned} \tag{3.2.1}$$

For each choice of the polynomial order, the coefficients c_i must be determined optimally such that the corresponding polynomial curve best fits the given data. This can be accomplished by a *least-squares fit*, which chooses the c_i that minimize the total mean-square error. For example, in the quadratic case, we have the performance index:

$$\mathcal{J} = \sum_{m=-2}^{2} e_m^2 = \sum_{m=-2}^{2} \left(y_m - (c_0 + c_1 m + c_2 m^2)\right)^2 = \min \tag{3.2.2}$$

[‡] not to be confused with the ordinary Chebyshev polynomials.

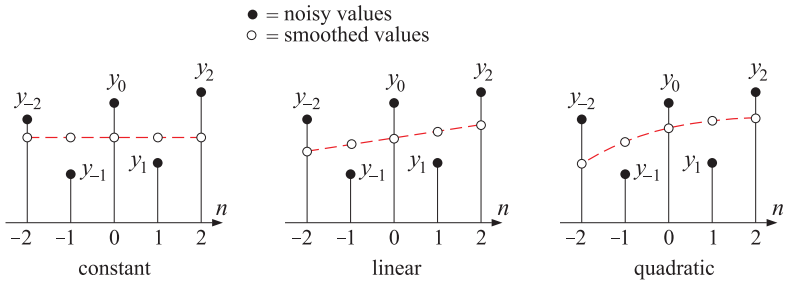


Fig. 3.2.1 Data smoothing with polynomials of degrees $d = 0, 1, 2$.

where the fitting errors are defined as

$$e_m = y_m - \hat{y}_m = y_m - (c_0 + c_1 m + c_2 m^2), \qquad m = -2, -1, 0, 1, 2$$

It proves convenient to express Eqs. (3.2.1) and (3.2.2) in a vectorial form, which generalizes to higher polynomial orders and to more than five data points. We define the five-dimensional vectors of data, estimates, and errors:

$$\mathbf{y} = \begin{bmatrix} y_{-2} \\ y_{-1} \\ y_0 \\ y_1 \\ y_2 \end{bmatrix}, \qquad \hat{\mathbf{y}} = \begin{bmatrix} \hat{y}_{-2} \\ \hat{y}_{-1} \\ \hat{y}_0 \\ \hat{y}_1 \\ \hat{y}_2 \end{bmatrix}, \qquad \mathbf{e} = \begin{bmatrix} e_{-2} \\ e_{-1} \\ e_0 \\ e_1 \\ e_2 \end{bmatrix} = \mathbf{y} - \hat{\mathbf{y}}$$

Similarly, we define the five-dimensional polynomial *basis vectors* $\mathbf{s}_0$, $\mathbf{s}_1$, $\mathbf{s}_2$, whose components are:

$$s_0(m) = 1, \qquad s_1(m) = m, \qquad s_2(m) = m^2, \qquad -2 \le m \le 2$$

Vectorially, we have:

$$\mathbf{s}_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \qquad \mathbf{s}_1 = \begin{bmatrix} -2 \\ -1 \\ 0 \\ 1 \\ 2 \end{bmatrix}, \qquad \mathbf{s}_2 = \begin{bmatrix} 4 \\ 1 \\ 0 \\ 1 \\ 4 \end{bmatrix} \tag{3.2.3}$$

In this notation, we may write the third of Eq. (3.2.1) vectorially:

$$\hat{\mathbf{y}} = c_0 \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} + c_1 \begin{bmatrix} -2 \\ -1 \\ 0 \\ 1 \\ 2 \end{bmatrix} + c_2 \begin{bmatrix} 4 \\ 1 \\ 0 \\ 1 \\ 4 \end{bmatrix} = c_0 \mathbf{s}_0 + c_1 \mathbf{s}_1 + c_2 \mathbf{s}_2$$

Therefore,

$$\hat{\mathbf{y}} = c_0 \mathbf{s}_0 + c_1 \mathbf{s}_1 + c_2 \mathbf{s}_2 = [\mathbf{s}_0, \mathbf{s}_1, \mathbf{s}_2] \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} \equiv S\mathbf{c} \tag{3.2.4}$$

The 5×3 basis matrix S has as columns the three basis vectors $\mathbf{s}_0$, $\mathbf{s}_1$, $\mathbf{s}_2$. It is given explicitly as follows:

$$S = [\mathbf{s}_0, \mathbf{s}_1, \mathbf{s}_2] = \begin{bmatrix} 1 & -2 & 4 \\ 1 & -1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \end{bmatrix} \tag{3.2.5}$$

Writing $\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}} = \mathbf{y} - S\mathbf{c}$, we can express the performance index (3.2.2) as the dot product:

$$\mathcal{J} = \mathbf{e}^T \mathbf{e} = (\mathbf{y} - S\mathbf{c})^T (\mathbf{y} - S\mathbf{c}) = \min \tag{3.2.6}$$

To minimize this expression with respect to $\mathbf{c}$, we set the gradient $\partial\mathcal{J}/\partial\mathbf{c}$ to zero:

$$\frac{\partial \mathcal{J}}{\partial \mathbf{c}} = -2S^T \mathbf{e} = -2S^T (\mathbf{y} - S\mathbf{c}) = -2(S^T \mathbf{y} - S^T S\mathbf{c}) = 0 \tag{3.2.7}$$

Therefore, the minimization condition gives the so-called *orthogonality equations* and the equivalent *normal equations*:

$$\frac{\partial \mathcal{J}}{\partial \mathbf{c}} = 0 \qquad \Leftrightarrow \qquad S^T \mathbf{e} = 0 \qquad \Leftrightarrow \qquad S^T S\mathbf{c} = S^T \mathbf{y} \tag{3.2.8}$$

with optimal solution:

$$\mathbf{c} = (S^T S)^{-1} S^T \mathbf{y} \equiv G^T \mathbf{y} \tag{3.2.9}$$

where we defined the 5×3 matrix G by

$$G = S(S^T S)^{-1} \tag{3.2.10}$$

We note that the solution (3.2.9) is none other than the unique least-squares solution of the full-rank overdetermined linear system $S\mathbf{c} = \mathbf{y}$, as given for example by Eq. (15.4.10), $\mathbf{c} = S^+ \mathbf{y}$, where $S^+ = (S^T S)^{-1} S^T$ is the corresponding pseudoinverse. Inserting the optimal coefficients $\mathbf{c}$ into Eq. (3.2.4), we find the smoothed values:†

$$\hat{\mathbf{y}} = S\mathbf{c} = SG^T \mathbf{y} = S(S^T S)^{-1} S^T \mathbf{y} \equiv B^T \mathbf{y} \tag{3.2.11}$$

where we defined the 5×5 matrix B by

$$B = B^T = SG^T = GS^T = S(S^T S)^{-1} S^T \tag{3.2.12}$$

The symmetric 3×3 matrix $F = S^T S$, which appears in the expressions for G and B, has matrix elements that are the *dot products* of the basis vectors, that is, the ijth matrix element is $F_{ij} = (S^T S)_{ij} = \mathbf{s}_i^T \mathbf{s}_j$. Indeed, using Eq. (3.2.5), we find:

$$F = S^T S = \begin{bmatrix} \mathbf{s}_0^T \\ \mathbf{s}_1^T \\ \mathbf{s}_2^T \end{bmatrix} [\mathbf{s}_0, \mathbf{s}_1, \mathbf{s}_2] = \begin{bmatrix} \mathbf{s}_0^T \mathbf{s}_0 & \mathbf{s}_0^T \mathbf{s}_1 & \mathbf{s}_0^T \mathbf{s}_2 \\ \mathbf{s}_1^T \mathbf{s}_0 & \mathbf{s}_1^T \mathbf{s}_1 & \mathbf{s}_1^T \mathbf{s}_2 \\ \mathbf{s}_2^T \mathbf{s}_0 & \mathbf{s}_2^T \mathbf{s}_1 & \mathbf{s}_2^T \mathbf{s}_2 \end{bmatrix} \tag{3.2.13}$$

† although B is symmetric, we prefer to write $\hat{\mathbf{y}} = B^T \mathbf{y}$, which generalizes to the non-symmetric case of minimum-roughness filters of Sec. 4.2.

Using Eq. (3.2.5), we calculate F and its inverse F^{-1}:

$$F = \begin{bmatrix} 5 & 0 & 10 \\ 0 & 10 & 0 \\ 10 & 0 & 34 \end{bmatrix}, \qquad F^{-1} = \frac{1}{35}\begin{bmatrix} 17 & 0 & -5 \\ 0 & 3.5 & 0 \\ -5 & 0 & 2.5 \end{bmatrix} \tag{3.2.14}$$

Then, we calculate the 5×3 matrix $G = S(S^TS)^{-1} = SF^{-1}$:

$$G = SF^{-1} = \frac{1}{35}\begin{bmatrix} 1 & -2 & 4 \\ 1 & -1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \end{bmatrix}\begin{bmatrix} 17 & 0 & -5 \\ 0 & 3.5 & 0 \\ -5 & 0 & 2.5 \end{bmatrix} \quad \text{or,}$$

$$G = \frac{1}{35}\begin{bmatrix} -3 & -7 & 5 \\ 12 & -3.5 & -2.5 \\ 17 & 0 & -5 \\ 12 & 3.5 & -2.5 \\ -3 & 7 & 5 \end{bmatrix} \equiv [\mathbf{g}_0, \mathbf{g}_1, \mathbf{g}_2] \tag{3.2.15}$$

As we see below, the three columns of G have useful interpretations as differentiation filters. Next, using Eq. (3.2.12), we calculate the 5×5 matrix B:

$$B = GS^T = \frac{1}{35}\begin{bmatrix} -3 & -7 & 5 \\ 12 & -3.5 & -2.5 \\ 17 & 0 & -5 \\ 12 & 3.5 & -2.5 \\ -3 & 7 & 5 \end{bmatrix}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ -2 & -1 & 0 & 1 & 2 \\ 4 & 1 & 0 & 1 & 4 \end{bmatrix} \quad \text{or,}$$

$$B = \frac{1}{35}\begin{bmatrix} 31 & 9 & -3 & -5 & 3 \\ 9 & 13 & 12 & 6 & -5 \\ -3 & 12 & 17 & 12 & -3 \\ -5 & 6 & 12 & 13 & 9 \\ 3 & -5 & -3 & 9 & 31 \end{bmatrix} \equiv [\mathbf{b}_{-2}, \mathbf{b}_{-1}, \mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2] \tag{3.2.16}$$

Because B is symmetric, its rows are the same as its columns. Thus, we can write it either in column-wise or row-wise form:

$$B = [\mathbf{b}_{-2}, \mathbf{b}_{-1}, \mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2] = \begin{bmatrix} \mathbf{b}_{-2}^T \\ \mathbf{b}_{-1}^T \\ \mathbf{b}_0^T \\ \mathbf{b}_1^T \\ \mathbf{b}_2^T \end{bmatrix} = B^T$$

The five columns or rows of B are the LPSM filters of length 5 and polynomial order 2. The corresponding smoothed values $\hat{\mathbf{y}}$ can be expressed component-wise in terms of these filters, as follows:

$$\begin{bmatrix} \hat{y}_{-2} \\ \hat{y}_{-1} \\ \hat{y}_0 \\ \hat{y}_1 \\ \hat{y}_2 \end{bmatrix} = \hat{\mathbf{y}} = B^T\mathbf{y} = \begin{bmatrix} \mathbf{b}_{-2}^T \\ \mathbf{b}_{-1}^T \\ \mathbf{b}_0^T \\ \mathbf{b}_1^T \\ \mathbf{b}_2^T \end{bmatrix}\mathbf{y} = \begin{bmatrix} \mathbf{b}_{-2}^T\mathbf{y} \\ \mathbf{b}_{-1}^T\mathbf{y} \\ \mathbf{b}_0^T\mathbf{y} \\ \mathbf{b}_1^T\mathbf{y} \\ \mathbf{b}_2^T\mathbf{y} \end{bmatrix}$$

or, for $m = -2, -1, 0, 1, 2$:

$$\hat{y}_m = \mathbf{b}_m^T\mathbf{y} \tag{3.2.17}$$

and more explicitly,

$$\begin{bmatrix} \hat{y}_{-2} \\ \hat{y}_{-1} \\ \hat{y}_0 \\ \hat{y}_1 \\ \hat{y}_2 \end{bmatrix} = \frac{1}{35}\begin{bmatrix} 31 & 9 & -3 & -5 & 3 \\ 9 & 13 & 12 & 6 & -5 \\ -3 & 12 & 17 & 12 & -3 \\ -5 & 6 & 12 & 13 & 9 \\ 3 & -5 & -3 & 9 & 31 \end{bmatrix}\begin{bmatrix} y_{-2} \\ y_{-1} \\ y_0 \\ y_1 \\ y_2 \end{bmatrix} \tag{3.2.18}$$

Thus, the mth filter $\mathbf{b}_m$ dotted into the data vector $\mathbf{y}$ generates the mth smoothed data sample. In a similar fashion, we can express the polynomial coefficients c_i as dot products. Using the solution Eq. (3.2.9), we have

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \mathbf{c} = G^T\mathbf{y} = \begin{bmatrix} \mathbf{g}_0^T \\ \mathbf{g}_1^T \\ \mathbf{g}_2^T \end{bmatrix}\mathbf{y} = \begin{bmatrix} \mathbf{g}_0^T\mathbf{y} \\ \mathbf{g}_1^T\mathbf{y} \\ \mathbf{g}_2^T\mathbf{y} \end{bmatrix}$$

or, explicitly,

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \frac{1}{35}\begin{bmatrix} -3 & 12 & 17 & 12 & -3 \\ -7 & -3.5 & 0 & 3.5 & 7 \\ 5 & -2.5 & -5 & -2.5 & 5 \end{bmatrix}\begin{bmatrix} y_{-2} \\ y_{-1} \\ y_0 \\ y_1 \\ y_2 \end{bmatrix} \tag{3.2.19}$$

Thus, the coefficients c_i can be expressed as the dot products of the columns of G with the data vector $\mathbf{y}$:

$$c_i = \mathbf{g}_i^T\mathbf{y}, \qquad i = 0, 1, 2 \tag{3.2.20}$$

Of the five columns of B, the middle one, $\mathbf{b}_0$, is the most important because it smooths the value y_0, which is symmetrically placed with respect to the other samples in $\mathbf{y}$, as shown in Fig. 3.2.1.

In smoothing a long block of data, the filter $\mathbf{b}_0$ is used during the *steady-state* period, whereas the other columns of B are used only during the input-on and input-off *transients*. We will refer to $\mathbf{b}_0$ and the other columns of B as the *steady-state* and *transient* LPSM filters.

Setting $m = 0$ into Eq. (3.2.1), we note that the middle smoothed value $\hat{y}_0$ is equal to the polynomial coefficient c_0. Using Eqs. (3.2.17) and (3.2.20), we find: $\hat{y}_0 = c_0 = \mathbf{b}_0^T\mathbf{y} = \mathbf{g}_0^T\mathbf{y}$ (the middle column of B and the first column of G are always the same, $\mathbf{b}_0 = \mathbf{g}_0$.)

To express (3.2.18) as a true filtering operation acting on an input sequence y_n, we shift the group of five samples to be centered around the nth time instant, that is, we make the substitution:

$$[y_{-2}, y_{-1}, y_0, y_1, y_2] \longrightarrow [y_{n-2}, y_{n-1}, y_n, y_{n+1}, y_{n+2}]$$

The corresponding five smoothed values will be then:

$$\begin{bmatrix} \hat{y}_{n-2} \\ \hat{y}_{n-1} \\ \hat{y}_n \\ \hat{y}_{n+1} \\ \hat{y}_{n+2} \end{bmatrix} = \frac{1}{35} \begin{bmatrix} 31 & 9 & -3 & -5 & 3 \\ 9 & 13 & 12 & 6 & -5 \\ -3 & 12 & 17 & 12 & -3 \\ -5 & 6 & 12 & 13 & 9 \\ 3 & -5 & -3 & 9 & 31 \end{bmatrix} \begin{bmatrix} y_{n-2} \\ y_{n-1} \\ y_n \\ y_{n+1} \\ y_{n+2} \end{bmatrix} \tag{3.2.21}$$

In particular, the middle sample y_n is smoothed by the filter $\mathbf{b}_0$:

$$\boxed{\hat{x}_n = \frac{1}{35}(-3y_{n-2} + 12y_{n-1} + 17y_n + 12y_{n+1} - 3y_{n+2})} \tag{3.2.22}$$

where, in accordance with our assumed model of noisy observations $y_n = x_n + v_n$, we denoted $\hat{y}_n$ by $\hat{x}_n$, i.e., the estimated value of x_n.

The other estimated values $\{\hat{y}_{n+m},\ m = \pm 1, \pm 2\}$, are not used for smoothing, except, as we see later, at the beginning and end of the signal block y_n. They may be used, however, for prediction and interpolation.

The filter (3.2.22) corresponds to fitting every group of five samples $\{y_{n-2}, y_{n-1}, y_n, y_{n+1}, y_{n+2}\}$ to a quadratic polynomial and replacing the middle sample y_n by its smoothed value $\hat{x}_n$. It is a lowpass filter and is normalized to unity gain at DC, because its coefficients add up to one.

Its NRR is the sum of the squared filter coefficients. It can be proved in general that the NRR of any steady-state filter $\mathbf{b}_0$ is equal to the *middle* value of its impulse response, that is, the coefficient $b_0(0)$. Therefore,

$$\mathcal{R} = \mathbf{b}_0^T \mathbf{b}_0 = \sum_{m=-2}^{2} b_0(m)^2 = b_0(0) = \frac{17}{35} = \frac{17/7}{5} = \frac{2.43}{5} = 0.49$$

By comparison, the length-5 FIR averager operating on the same five samples is:

$$\boxed{\hat{x}_n = \frac{1}{5}(y_{n-2} + y_{n-1} + y_n + y_{n+1} + y_{n+2})} \tag{3.2.23}$$

with $\mathcal{R} = 1/N = 1/5$. Thus, the length-5 quadratic-polynomial filter performs 2.43 times worse in reducing noise than the FIR averager. However, the higher-order polynomial filters have other advantages to be discussed later.

We saw that the coefficient c_0 represents the smoothed value of y_0 at $m = 0$. Similarly, the coefficient c_1 represents the slope, the derivative, of y_0 at $m = 0$. Indeed, we have from Eq. (3.2.1) by differentiating and setting $m = 0$:

$$\dot{\hat{y}}_0 = \left.\frac{d\hat{y}_m}{dm}\right|_0 = c_1\,, \qquad \ddot{\hat{y}}_0 = \left.\frac{d^2\hat{y}_m}{dm^2}\right|_0 = 2c_2$$

Thus, c_1 and $2c_2$ represent the polynomial estimates of the first and second derivatives at $m = 0$. Using Eq. (3.2.20) we can express them in terms of the second and third columns of the matrix G:

$$\begin{aligned} \dot{\hat{y}}_0 &= c_1 = \mathbf{g}_1^T \mathbf{y} \\ \ddot{\hat{y}}_0 &= 2c_2 = 2\mathbf{g}_2^T \mathbf{y} \end{aligned} \tag{3.2.24}$$

Shifting these to the nth time sample, and denoting them by $\dot{\hat{x}}_n$ and $\ddot{\hat{x}}_n$, we find the length-5 local polynomial filters for estimating the *first and second derivatives* of x_n:

$$\boxed{\begin{aligned} \dot{\hat{x}}_n &= \frac{1}{35}(-7y_{n-2} - 3.5y_{n-1} + 3.5y_{n+1} + 7y_{n+2}) \\ \ddot{\hat{x}}_n &= \frac{2}{35}(5y_{n-2} - 2.5y_{n-1} - 5y_n - 2.5y_{n+1} + 5y_{n+2}) \end{aligned}} \tag{3.2.25}$$

The above designs can be generalized in a straightforward manner to an arbitrary degree d of the fitted polynomial and to an arbitrary length N of the data vector $\mathbf{y}$. We require only that $d \le N - 1$, a restriction to be clarified later. Assuming that N is odd, say, $N = 2M + 1$, the five-dimensional data vector $\mathbf{y} = [y_{-2}, y_{-1}, y_0, y_1, y_2]^T$ is replaced by an N-dimensional one, having M points on either side of y_0:

$$\mathbf{y} = [y_{-M}, \ldots, y_{-1}, y_0, y_1, \ldots, y_M]^T \tag{3.2.26}$$

The N data samples in $\mathbf{y}$ are then fitted by a polynomial of degree d:

$$\hat{y}_m = c_0 + c_1 m + \cdots + c_d m^d, \qquad -M \le m \le M \tag{3.2.27}$$

In this case, there are $d+1$ polynomial basis vectors $\mathbf{s}_i$, $i = 0, 1, \ldots, d$, defined to have components:

$$s_i(m) = m^i, \qquad -M \le m \le M \tag{3.2.28}$$

The corresponding $N \times (d+1)$ basis matrix S is defined to have the $\mathbf{s}_i$ as columns:

$$S = [\mathbf{s}_0, \mathbf{s}_1, \ldots, \mathbf{s}_d] \tag{3.2.29}$$

The smoothed values (3.2.27) can be written in the vector form:

$$\hat{\mathbf{y}} = \sum_{i=0}^{d} c_i \mathbf{s}_i = [\mathbf{s}_0, \mathbf{s}_1, \ldots, \mathbf{s}_d] \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_d \end{bmatrix} = S\mathbf{c} \tag{3.2.30}$$

The design steps for the LPSM filters can be summarized then as follows:

$$\begin{aligned} &F = S^T S \quad \Leftrightarrow \quad F_{ij} = \mathbf{s}_i^T \mathbf{s}_j, \quad i, j = 0, 1, \ldots, d \\ &G = SF^{-1} \equiv [\mathbf{g}_0, \mathbf{g}_1, \ldots, \mathbf{g}_d] \\ &B = GS^T = SF^{-1}S^T \equiv [\mathbf{b}_{-M}, \ldots, \mathbf{b}_0, \ldots, \mathbf{b}_M] \end{aligned} \tag{3.2.31}$$

The corresponding coefficient vector $\mathbf{c}$ and smoothed data vector $\hat{\mathbf{y}}$ will be:

$$\begin{aligned} \mathbf{c} = G^T\mathbf{y} \quad &\Leftrightarrow \quad c_i = \mathbf{g}_i^T\mathbf{y}, \quad i = 0,1,\ldots,d \\ \hat{\mathbf{y}} = B^T\mathbf{y} \quad &\Leftrightarrow \quad \hat{y}_m = \mathbf{b}_m^T\mathbf{y}, \quad -M \le m \le M \end{aligned} \tag{3.2.32}$$

The middle smoothed value $\hat{y}_0$ is given in terms of the middle LPSM filter $\mathbf{b}_0$:

$$\hat{y}_0 = \mathbf{b}_0^T\mathbf{y} = \sum_{k=-M}^{M} b_0(k)\,y_k$$

The N-dimensional vector $\mathbf{y} = [y_{-M},\ldots,y_{-1},y_0,y_1,\ldots,y_M]^T$ can be shifted to the nth time instant by the replacement:

$$[y_{-M},\ldots,y_{-1},y_0,y_1,\ldots,y_M] \longrightarrow [y_{n-M},\ldots,y_{n-1},y_n,y_{n+1},\ldots,y_{n+M}]$$

The resulting length-N, order-d, LPSM filter for smoothing a noisy sequence y_n will be, in its steady-state form (denoting again $\hat{x}_n = \hat{y}_n$):

$$\boxed{\hat{x}_n = \hat{y}_n = \sum_{k=-M}^{M} b_0(k)\,y_{n+k} = \sum_{k=-M}^{M} b_0(-k)\,y_{n-k}} \tag{3.2.33}$$

The second equation expresses the output in convolutional form.[†] Because the filter $\mathbf{b}_0$ is symmetric about its middle, we can replace $b_0(-k) = b_0(k)$. The non-central estimated values are obtained from the $\mathbf{b}_m$ filters:

$$\boxed{\hat{y}_{n+m} = \sum_{k=-M}^{M} b_m(k)\,y_{n+k} = \sum_{k=-M}^{M} b_m^R(k)\,y_{n-k}}, \quad -M \le m \le M \tag{3.2.34}$$

These filters satisfy the symmetry property $b_m^R(k) = b_m(-k) = b_{-m}(k)$ and can be used for prediction, as we discuss later.

The $d+1$ columns of the $N\times(d+1)$-dimensional matrix G give the LPSM *differentiation filters*, for derivatives of orders $i = 0,1,\ldots,d$. It follows by differentiating Eq. (3.2.27) i times and setting $m = 0$:

$$\hat{y}_0^{(i)} = \left.\frac{d^i\hat{y}_m}{dm^i}\right|_0 = i!\,c_i = i!\,\mathbf{g}_i^T\mathbf{y}$$

Shifting these to time n, gives the differentiation convolutional filtering equations:

$$\hat{x}_n^{(i)} = i!\sum_{m=-M}^{M} g_i^R(m)\,y_{n-m}, \qquad i = 0,1,\ldots,d \tag{3.2.35}$$

where, $g_i^R(m) = g_i(-m)$ and as in Eq. (3.2.33), we reversed the order of writing the terms, but here the filters $\mathbf{g}_i$ are not necessarily symmetric (actually, they are symmetric for even i, and antisymmetric for odd i.)

[†]We use the notation $\mathbf{b}^R$ to denote the reverse of a double-sided filter $\mathbf{b}$, that is, $b^R(k) = b(-k)$.

Example 3.2.1: We construct the length-5 LPSM filters for the cases $d = 0$ and $d = 1$. For $d = 0$, corresponding to the constant $\hat{y}_m = c_0$ in Eq. (3.2.1), there is only one basis vector $\mathbf{s}_0$ defined in Eq. (3.2.3). The basis matrix $S = [\mathbf{s}_0]$ will have just one column, and the matrix F will be the scalar

$$F = S^TS = \mathbf{s}_0^T\mathbf{s}_0 = [1,1,1,1,1]\begin{bmatrix}1\\1\\1\\1\\1\end{bmatrix} = 5$$

The matrix G will then be

$$G = SF^{-1} = \frac{1}{5}\mathbf{s}_0 = \frac{1}{5}[1,1,1,1,1]^T$$

resulting in the LPSM matrix B:

$$B = GS^T = \frac{1}{5}\mathbf{s}_0\mathbf{s}_0^T = \frac{1}{5}\begin{bmatrix}1\\1\\1\\1\\1\end{bmatrix}[1,1,1,1,1] = \frac{1}{5}\begin{bmatrix}1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix}$$

Thus, the steady-state LPSM filter is the length-5 averager:

$$\mathbf{b}_0 = \frac{1}{5}[1,1,1,1,1]^T$$

For the case $d = 1$, corresponding to the linear fit $\hat{y}_m = c_0 + c_1 m$, we have the two basis vectors $\mathbf{s}_0$ and $\mathbf{s}_1$, given in Eq. (3.2.3). We calculate the matrices S, F, and F^{-1}:

$$S = [\mathbf{s}_0,\mathbf{s}_1] = \begin{bmatrix}1&-2\\1&-1\\1&0\\1&1\\1&2\end{bmatrix},\quad F = S^TS = \begin{bmatrix}5&0\\0&10\end{bmatrix},\quad F^{-1} = \frac{1}{5}\begin{bmatrix}1&0\\0&0.5\end{bmatrix}$$

This gives for G and B:

$$G = SF^{-1} = \frac{1}{5}\begin{bmatrix}1&-1\\1&-0.5\\1&0\\1&0.5\\1&1\end{bmatrix},\quad B = GS^T = \frac{1}{5}\begin{bmatrix}3&2&1&0&-1\\2&1.5&1&0.5&0\\1&1&1&1&1\\0&0.5&1&1.5&2\\-1&0&1&2&3\end{bmatrix}$$

Thus, the steady-state LPSM filter $\mathbf{b}_0$ is still equal to the length-5 FIR averager. It is a general property of LPSM filters, that the filter $\mathbf{b}_0$ is the same for successive polynomial orders, that is, for $d = 0,1$, $d = 2,3$, $d = 4,5$, and so on. However, the transient LPSM filters are different. □

Example 3.2.2: Here, we construct the LPSM filters of length $N = 5$ and order $d = 3$. The smoothed estimates are given by the cubic polynomial:

$$\hat{y}_m = c_0 + c_1 m + c_2 m^2 + c_3 m^3$$

There is an additional basis vector $\mathbf{s}_3$ with components $s_3(m) = m^3$. Therefore, the basis matrix S is:

$$S = [\mathbf{s}_0, \mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3] = \begin{bmatrix} 1 & -2 & 4 & -8 \\ 1 & -1 & 1 & -1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \end{bmatrix} \Rightarrow F = S^T S = \begin{bmatrix} 5 & 0 & 10 & 0 \\ 0 & 10 & 0 & 34 \\ 10 & 0 & 34 & 0 \\ 0 & 34 & 0 & 130 \end{bmatrix}$$

Because of the checkerboard pattern of this matrix, its inverse can be obtained from the inverses of the two 2×2 interlaced submatrices:

$$\begin{bmatrix} 5 & 10 \\ 10 & 34 \end{bmatrix}^{-1} = \frac{1}{70}\begin{bmatrix} 34 & -10 \\ -10 & 5 \end{bmatrix}, \quad \begin{bmatrix} 10 & 34 \\ 34 & 130 \end{bmatrix}^{-1} = \frac{1}{144}\begin{bmatrix} 130 & -34 \\ -34 & 10 \end{bmatrix}$$

Interlacing these inverses, we obtain:

$$F^{-1} = \begin{bmatrix} 34/70 & 0 & -10/70 & 0 \\ 0 & 130/144 & 0 & -34/144 \\ -10/70 & 0 & 5/70 & 0 \\ 0 & -34/144 & 0 & 10/144 \end{bmatrix}$$

Then, we compute the derivative filter matrix G:

$$G = SF^{-1} = \frac{1}{35}\begin{bmatrix} -3 & 35/12 & 5 & -35/12 \\ 12 & -70/3 & -2.5 & 35/6 \\ 17 & 0 & -5 & 0 \\ 12 & 70/3 & -2.5 & -35/6 \\ -3 & -35/12 & 5 & 35/12 \end{bmatrix}$$

and the LPSM matrix B:

$$B = SG^T = \frac{1}{35}\begin{bmatrix} 34.5 & 2 & -3 & 2 & -0.5 \\ 2 & 27 & 12 & -8 & 2 \\ -3 & 12 & 17 & 12 & -3 \\ 2 & -8 & 12 & 27 & 2 \\ -0.5 & 2 & -3 & 2 & 34.5 \end{bmatrix}$$

As mentioned above, the steady-state LPSM filter $\mathbf{b}_0$ is the same as that of case $d = 2$. But, the transient and differentiation filters are different. □

3.3 Exact Design Equations

In practice, the most common values of d are $0, 1, 2, 3, 4$. For these ds and arbitrary filter lengths N, the LPSM matrix B can be constructed in closed form; see references [36–99],

as well as the extensive tables in [54]. Denoting the inverse of the $(d+1)\times(d+1)$ matrix $F = S^T S$ by $\Phi = F^{-1}$, we can write

$$B = SF^{-1}S^T = S\Phi S^T = \sum_{i=0}^{d}\sum_{j=0}^{d} \mathbf{s}_i \mathbf{s}_j^T \Phi_{ij} \tag{3.3.1}$$

which gives for the mkth matrix element

$$\boxed{B_{mk} = \sum_{i=0}^{d}\sum_{j=0}^{d} s_i(m)\, s_j(k)\, \Phi_{ij} = \sum_{i=0}^{d}\sum_{j=0}^{d} m^i k^j \Phi_{ij}}, \qquad -M \le m,k \le M \tag{3.3.2}$$

Because of symmetry, $B_{mk} = B_{km}$, these matrix elements represent the kth component of the LPSM filter $\mathbf{b}_m$ or the mth component of the filter $\mathbf{b}_k$, that is,

$$B_{mk} = B_{km} = b_m(k) = b_k(m) = \sum_{i=0}^{d}\sum_{j=0}^{d} m^i k^j \Phi_{ij} \tag{3.3.3}$$

The matrix Φ can be determined easily for the cases $0 \le d \le 4$. The matrix F is a *Hankel matrix*, that is, having the same entries along each antidiagonal line. Therefore, its matrix elements F_{ij} depend only on the sum $i+j$ of the indices. To see this, we write F_{ij} as the inner product:

$$F_{ij} = (S^T S)_{ij} = \mathbf{s}_i^T \mathbf{s}_j = \sum_{m=-M}^{M} s_i(m)\, s_j(m) = \sum_{m=-M}^{M} m^{i+j}, \qquad \text{or,}$$

$$\boxed{F_{ij} = \sum_{m=-M}^{M} m^{i+j} \equiv F_{i+j}}, \qquad 0 \le i,j \le d \tag{3.3.4}$$

Note that because of the symmetric limits of summation, F_{i+j} will be zero whenever $i+j$ is odd. This leads to the checkerboard pattern of alternating zeros in F that we saw in the above examples. Also, because $d \le 4$, the only values of $i+j$ that we need are: $i+j = 0, 2, 4, 6, 8$. For those, the summations over m can be done in closed form:

$$\begin{aligned} F_0 &= \sum_{m=-M}^{M} m^0 = N = 2M+1 \\ F_2 &= \sum_{m=-M}^{M} m^2 = \frac{1}{3}M(M+1)(2M+1) \\ F_4 &= \sum_{m=-M}^{M} m^4 = \frac{1}{5}(3M^2+3M-1)F_2 \\ F_6 &= \sum_{m=-M}^{M} m^6 = \frac{1}{7}(3M^4+6M^3-3M+1)F_2 \\ F_8 &= \sum_{m=-M}^{M} m^8 = \frac{1}{15}(5M^6+15M^5+5M^4-15M^3-M^2+9M-3)F_2 \end{aligned} \tag{3.3.5}$$

We can express F in terms of these definitions for various values of d. For example, for $d = 0, 1, 2, 3$, the F matrices are:

$$[F_0], \quad \begin{bmatrix} F_0 & 0 \\ 0 & F_2 \end{bmatrix}, \quad \begin{bmatrix} F_0 & 0 & F_2 \\ 0 & F_2 & 0 \\ F_2 & 0 & F_4 \end{bmatrix}, \quad \begin{bmatrix} F_0 & 0 & F_2 & 0 \\ 0 & F_2 & 0 & F_4 \\ F_2 & 0 & F_4 & 0 \\ 0 & F_4 & 0 & F_6 \end{bmatrix}$$

The corresponding inverse matrices $\Phi = F^{-1}$ are obtained by interlacing the inverses of the checkerboard submatrices, as in Example 3.2.2. For $d = 0, 1, 2$, we have for Φ:

$$[1/F_0], \quad \begin{bmatrix} 1/F_0 & 0 \\ 0 & 1/F_2 \end{bmatrix}, \quad \begin{bmatrix} F_4/D_4 & 0 & -F_2/D_4 \\ 0 & 1/F_2 & 0 \\ -F_2/D_4 & 0 & F_0/D_4 \end{bmatrix},$$

and for $d = 3$:

$$\Phi = F^{-1} = \begin{bmatrix} F_4/D_4 & 0 & -F_2/D_4 & 0 \\ 0 & F_6/D_8 & 0 & -F_4/D_8 \\ -F_2/D_4 & 0 & F_0/D_4 & 0 \\ 0 & -F_4/D_8 & 0 & F_2/D_8 \end{bmatrix}$$

where the D_4 and D_8 are determinants of the interlaced submatrices:

$$\begin{aligned} D_4 &= F_0F_4 - F_2^2 = \frac{1}{45}M(M+1)(2M+1)(2M+3)(4M^2-1) \\ D_8 &= F_2F_6 - F_4^2 = \frac{3}{35}M(M+2)(M^2-1)D_4 \end{aligned} \tag{3.3.6}$$

Inserting the above expressions for Φ into Eq. (3.3.3), we determine the corresponding LPSM filters. For $d = 0$, we find for $-M \le m, k \le M$:

$$b_m(k) = B_{mk} = \frac{1}{F_0} = \frac{1}{N} \tag{3.3.7}$$

For $d = 1$:

$$b_m(k) = B_{mk} = \frac{1}{F_0} + \frac{mk}{F_2} \tag{3.3.8}$$

For $d = 2$:

$$b_m(k) = B_{mk} = \frac{F_4}{D_4} + \frac{1}{F_2}mk - \frac{F_2}{D_4}(m^2+k^2) + \frac{F_0}{D_4}m^2k^2 \tag{3.3.9}$$

For $d = 3$:

$$\begin{aligned} b_m(k) = B_{mk} = {} & \frac{F_4}{D_4} + \frac{F_6}{D_8}mk - \frac{F_2}{D_4}(m^2+k^2) + \frac{F_0}{D_4}m^2k^2 \\ & - \frac{F_4}{D_8}(km^3+mk^3) + \frac{F_2}{D_8}m^3k^3 \end{aligned} \tag{3.3.10}$$

The required ratios are given explicitly as follows:

$$\begin{aligned} \frac{F_4}{D_4} &= \frac{3(3M^2+3M-1)}{(2M+3)(4M^2-1)} \\ \frac{F_2}{D_4} &= \frac{15}{(2M+3)(4M^2-1)} \\ \frac{F_0}{D_4} &= \frac{45}{M(M+1)(2M+3)(4M^2-1)} \\ \frac{F_6}{D_8} &= \frac{25(3M^4+6M^3-3M+1)}{M(M+2)(M^2-1)(2M+3)(4M^2-1)} \\ \frac{F_4}{D_8} &= \frac{35(3M^2+3M-1)}{M(M+2)(M^2-1)(2M+3)(4M^2-1)} \\ \frac{F_2}{D_8} &= \frac{175}{M(M+2)(M^2-1)(2M+3)(4M^2-1)} \end{aligned} \tag{3.3.11}$$

In a similar fashion, we also find for the case $d = 4$:

$$\begin{aligned} b_m(k) = B_{mk} = {} & \frac{D_{12}}{D} + \frac{F_6}{D_8}mk - \frac{D_{10}}{D}(m^2+k^2) + \frac{E_8}{D}m^2k^2 \\ & - \frac{F_4}{D_8}(km^3+mk^3) + \frac{F_2}{D_8}m^3k^3 + \frac{D_8}{D}(m^4+k^4) \\ & - \frac{D_6}{D}(m^2k^4+k^2m^4) + \frac{D_4}{D}m^4k^4 \end{aligned} \tag{3.3.12}$$

where

$$\begin{aligned} D_6 &= F_0F_6 - F_2F_4 & E_8 &= F_0F_8 - F_4^2 \\ D_{10} &= F_2F_8 - F_4F_6 & D_{12} &= F_4F_8 - F_6^2 \\ D &= F_0D_{12} - F_2D_{10} + F_4D_8 \end{aligned} \tag{3.3.13}$$

These are given explicitly as follows:

$$\begin{aligned} D_6 &= \frac{1}{7}(6M^2+6M-5)D_4 \\ D_{10} &= \frac{1}{21}M(M+2)(M^2-1)(2M^2+2M-3)D_4 \\ E_8 &= \frac{1}{5}(4M^4+8M^3-4M^2-8M+1)D_4 \\ D_{12} &= \frac{1}{735}M(M+2)(M^2-1)(15M^4+30M^3-35M^2-50M+12)D_4 \\ D &= \frac{4}{11025}M(M+2)(M^2-1)(2M+5)(4M^2-9)(4M^2-1)D_4 \end{aligned} \tag{3.3.14}$$

and the required ratios are:

$$\begin{aligned}
\frac{D_{12}}{D} &= \frac{15\,(15M^4+30M^3-35M^2-50M+12)}{4\,(2M+5)\,(4M^2-1)\,(4M^2-9)}\\
\frac{D_{10}}{D} &= \frac{525\,(2M^2+2M-3)}{4\,(2M+5)\,(4M^2-1)\,(4M^2-9)}\\
\frac{E_8}{D} &= \frac{2205\,(4M^4+8M^3-4M^2-8M+5)}{4M\,(M+2)\,(M^2-1)\,(2M+5)\,(4M^2-1)\,(4M^2-9)}\\
\frac{D_8}{D} &= \frac{945}{4\,(2M+5)\,(4M^2-1)\,(4M^2-9)}\\
\frac{D_6}{D} &= \frac{1575\,(6M^2+6M-5)}{4M\,(M+2)\,(M^2-1)\,(2M+5)\,(4M^2-1)\,(4M^2-9)}\\
\frac{D_4}{D} &= \frac{11025}{4M\,(M+2)\,(M^2-1)\,(2M+5)\,(4M^2-1)\,(4M^2-9)}
\end{aligned} \tag{3.3.15}$$

In this case, the matrix F and its two interlaced submatrices are:

$$F=\begin{bmatrix} F_0 & 0 & F_2 & 0 & F_4\\ 0 & F_2 & 0 & F_4 & 0\\ F_2 & 0 & F_4 & 0 & F_6\\ 0 & F_4 & 0 & F_6 & 0\\ F_4 & 0 & F_6 & 0 & F_8 \end{bmatrix},\quad \begin{bmatrix} F_0 & F_2 & F_4\\ F_2 & F_4 & F_6\\ F_4 & F_6 & F_8 \end{bmatrix},\quad \begin{bmatrix} F_2 & F_4\\ F_4 & F_6 \end{bmatrix}$$

Its inverse—obtained by interlacing the inverses of these two submatrices—can be expressed in terms of the determinant quantities of Eq. (3.3.13):

$$\Phi=F^{-1}=\begin{bmatrix} D_{12}/D & 0 & -D_{10}/D & 0 & D_8/D\\ 0 & F_6/D_8 & 0 & -F_4/D_8 & 0\\ -D_{10}/D & 0 & E_8/D & 0 & -D_6/D\\ 0 & -F_4/D_8 & 0 & F_2/D_8 & 0\\ D_8/D & 0 & -D_6/D & 0 & D_4/D \end{bmatrix}$$

Eqs. (3.3.5)–(3.3.15) provide closed-form expressions for the LPSM filters $b_m(k)$ of orders $d=0,1,2,3,4$. Setting $m=0$, we obtain the explicit forms of the steady-state filters $b_0(k)$, $-M\le k\le M$. For $d=0,1$:

$$b_0(k)=\frac{1}{2M+1} \tag{3.3.16}$$

for $d=2,3$:

$$b_0(k)=\frac{3\,(3M^2+3M-1-5k^2)}{(2M+3)\,(4M^2-1)} \tag{3.3.17}$$

and for $d=4,5$:

$$b_0(k)=\frac{15\,\big(15M^4+30M^3-35M^2-50M+12-35\,(2M^2+2M-3)k^2+63k^4\big)}{4\,(2M+5)\,(4M^2-1)\,(4M^2-9)} \tag{3.3.18}$$

Example 3.3.1: Determine the quadratic/cubic LPSM filters of lengths $N=5,7,9$. Using (3.3.17) with $M=2,3,4$, we find (for $-M\le k\le M$):

$$b_0(k)=\frac{17-5k^2}{35}=\frac{1}{35}[-3,12,17,12,-3]$$

$$b_0(k)=\frac{7-k^2}{21}=\frac{1}{21}[-2,3,6,7,6,3,-2]$$

$$b_0(k)=\frac{59-5k^2}{231}=\frac{1}{231}[-21,14,39,54,59,54,39,14,-21]$$

where the coefficients have been reduced to integers as much as possible. □

Example 3.3.2: Determine the quartic and quintic LPSM filters of length $N=7,9$. Using Eq. (3.3.18) with $M=3,4$, we find:

$$b_0(k)=\frac{131-61.25k^2+5.25k^4}{231}=\frac{1}{231}[5,-30,75,131,75,-30,5]$$

$$b_0(k)=\frac{179-46.25k^2+2.25k^4}{429}=\frac{1}{429}[15,-55,30,135,179,135,30,-55,15]$$

3.4 Geometric Interpretation

The LPSM filters admit a nice *geometric* interpretation, which is standard in least-squares problems. Let $\mathbb{Y}$ be the vector space of the N-dimensional real-valued vectors $\mathbf{y}$, that is, the space $\mathbb{R}^N$, and let $\mathbb{S}$ be the $(d+1)$-dimensional *subspace* spanned by all linear combinations of the basis vectors $\mathbf{s}_i$, $i=0,1,\ldots,d$.

Thus, the matrix $S=[\mathbf{s}_0,\mathbf{s}_1,\ldots,\mathbf{s}_d]$ is a (non-orthogonal) basis of the subspace $\mathbb{S}$. The smoothed vector $\hat{\mathbf{y}}$, being a linear combination of the $\mathbf{s}_i$, belongs to the subspace $\mathbb{S}$. Moreover, because of the orthogonality equations (3.2.8), $\hat{\mathbf{y}}$ is *orthogonal* to the error vector $\mathbf{e}$:

$$\hat{\mathbf{y}}^T\mathbf{e}=(S\mathbf{c})^T\mathbf{e}=\mathbf{c}^TS^T\mathbf{e}=0$$

Then, the equation $\mathbf{e}=\mathbf{y}-\hat{\mathbf{y}}$ can be rewritten as the *orthogonal decomposition*:

$$\mathbf{y}=\hat{\mathbf{y}}+\mathbf{e} \tag{3.4.1}$$

which expresses $\mathbf{y}$ as a sum of a part that belongs to the subspace $\mathbb{S}$ and a part that belongs to the *orthogonal complement* subspace $\mathbb{S}^\perp$. The decomposition is *unique* and represents the *direct sum* decomposition of the full vector space $\mathbb{Y}$:

$$\mathbb{Y}=\mathbb{S}\oplus\mathbb{S}^\perp$$

This geometric interpretation requires that the dimension of the subspace $\mathbb{S}$ not exceed the dimension of the full space $\mathbb{Y}$, that is, $d+1\le N$. The component $\hat{\mathbf{y}}$ that lies in $\mathbb{S}$ is the *projection* of $\mathbf{y}$ onto $\mathbb{S}$. The matrix B in Eq. (3.2.11) is the corresponding *projection matrix*. As such, it will be symmetric, $B^T=B$, and *idempotent*:

$$B^2=B \tag{3.4.2}$$

The proof is straightforward:

$$B^2 = \left(SF^{-1}S^T\right)\left(SF^{-1}S^T\right) = SF^{-1}(S^TS)F^{-1}S^T = SF^{-1}S^T = B$$

The matrix $(I-B)$, where I is the N-dimensional identity matrix, is also a projection matrix, projecting onto the orthogonal subspace $\mathbb{S}^\perp$. Thus, the error vector $\mathbf{e}$ belonging to $\mathbb{S}^\perp$ can be obtained from $\mathbf{y}$ by the projection:

$$\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}} = (I-B)\mathbf{y}$$

Because $(I-B)$ is also idempotent and symmetric, $(I-B)^2 = (I-B)$, we obtain for the *minimized* value of the performance index $\mathcal{J}$ of Eq. (3.2.6):

$$\mathcal{J}_{\min} = \mathbf{e}^T\mathbf{e} = \mathbf{y}^T(I-B)^2\mathbf{y} = \mathbf{y}^T(I-B)\mathbf{y} = \mathbf{y}^T\mathbf{y} - \mathbf{y}^TB\mathbf{y} \tag{3.4.3}$$

3.5 Orthogonal Polynomial Bases

Computationally, the non-orthogonal basis $S = [\mathbf{s}_0, \mathbf{s}_1, \ldots, \mathbf{s}_d]$ is not the most convenient one. The Gram-Schmidt orthogonalization process may be applied to the columns of S to obtain an orthogonal basis. This procedure amounts to performing the QR-factorization[†] on S, that is,

$$S = QR \tag{3.5.1}$$

where Q is an $N\times(d+1)$ matrix with orthonormal columns, that is, $Q^TQ = I$, and R is a $(d+1)\times(d+1)$ non-singular *upper-triangular* matrix.

The columns of $Q = [\mathbf{q}_0, \mathbf{q}_1, \ldots, \mathbf{q}_d]$, correspond to the (orthonormalized) discrete Chebyshev or Gram polynomials $q_i(n)$, $i = 0, 1, \ldots, d$, constructed from the monomial basis $s_i(n) = n^i$ by the Gram-Schmidt process. Noting that $S^TS = R^T(Q^TQ)R = R^TR$, the design of the filter matrices B, G can be formulated more efficiently as follows:

$$\begin{aligned} F &= S^TS = R^TR \\ G &= SF^{-1} = QR^{-T} \\ B &= SF^{-1}S^T = QQ^T \end{aligned} \tag{3.5.2}$$

which lead to the explicit construction of the differentiation and LPSM filters in terms of the Chebyshev polynomials $q_i(n)$:

$$\begin{aligned} \mathbf{g}_i &= \sum_{j=0}^{i} \mathbf{q}_j\,(R^{-1})_{ij} \quad\Rightarrow\quad g_i(n) = \sum_{j=0}^{i} q_j(n)\,(R^{-1})_{ij} \\ B &= \sum_{i=0}^{d} \mathbf{q}_i\,\mathbf{q}_i^T \quad\Rightarrow\quad b_m(k) = B_{km} = \sum_{i=0}^{d} q_i(k)\,q_i(m) \end{aligned} \tag{3.5.3}$$

The expression for $b_m(k)$ can be simplified further using the Christoffel-Darboux identity for orthogonal polynomials. We discuss these matters further in Sec. 4.3. The MATLAB function `lpsm` implements (3.5.2). Its inputs are N, d and its outputs B, G:

[†] see Sec. 15.20.


```
[B,G] = lpsm(N,d);      % local polynomial smoothing and differentiation filter design
```

The function constructs the basis matrix S with the help of the function `lpbasis` and carries out its QR-factorization with the help of the built-in function `qr`. The following code fragment illustrates the computational steps:

```
S = lpbasis(N,d);       % construct polynomial basis
[Q,R] = qr(S, 0);       % economy form, R is (d+1)x(d+1) upper triangular
G = Q/R';               % differentiation filters
B = Q*Q';               % smoothing filters
```

3.6 Polynomial Predictive and Interpolation Filters

The case $d+1 = N$ or $d = N-1$ is of special interest, corresponding to ordinary polynomial *Lagrange interpolation*. Indeed, in this case, the basis matrix S becomes a square non-singular $N\times N$ matrix with an ordinary inverse S^{-1}, which implies that B becomes the identity matrix,

$$B = S(S^TS)^{-1}S^T = S(S^{-1}S^{-T})S^T = I$$

or, equivalently, the subspace $\mathbb{S}$ becomes the full space $\mathbb{Y}$. The optimal polynomial of degree $d = N-1$ fits through all the sample points of the N-dimensional vector $\mathbf{y}$, that is, $\mathbf{e} = 0$ or $\hat{\mathbf{y}} = \mathbf{y} = S\mathbf{c}$, with solution $\mathbf{c} = S^{-1}\mathbf{y}$, and interpolates between those samples. This polynomial is defined for any independent variable t by:

$$\hat{y}_t = \sum_{i=0}^{N-1} c_i t^i = \mathbf{c}^T\mathbf{u}_t = \mathbf{y}^TS^{-T}\mathbf{u}_t \equiv \mathbf{y}^T\mathbf{b}_t = \sum_{k=-M}^{M} b_t(k)\,y_k \tag{3.6.1}$$

where we set,

$$\mathbf{u}_t = \begin{bmatrix} 1 \\ t \\ \vdots \\ t^{N-1} \end{bmatrix}, \quad \mathbf{b}_t = S^{-T}\mathbf{u}_t \quad\Rightarrow\quad b_t(k) = \sum_{i=0}^{N-1} (S^{-1})_{ik}\,t^i \tag{3.6.2}$$

The polynomials $b_t(k)$ of degree $(N-1)$ in t are the ordinary Lagrange interpolation polynomials, interpolating through the points y_k. To see this, we note that at each discrete value of t, say $t = m$ with $-M \le m \le M$, we have:

$$b_m(k) = \sum_{i=0}^{N-1} (S^{-1})_{ik}\,m^i = \sum_{i=0}^{N-1} (S^{-1})_{ik}\,S_{mi} = (SS^{-1})_{mk} = I_{mk} = \delta(m-k) \tag{3.6.3}$$

so that the polynomial passes through the signal values at the sampling instants:

$$\hat{y}_t\big|_{t=m} = \sum_{k=-M}^{M} b_m(k)\,y_k = \sum_{k=-M}^{M} \delta(m-k)\,y_k = y_m$$

It is straightforward to show using the property (3.6.3) that $b_t(k)$ is given by the usual Lagrange interpolation formula:

$$b_t(k) = \prod_{\substack{m=-M \\ m\neq k}}^{M} \left(\frac{t-m}{k-m}\right), \qquad -M \le k \le M \tag{3.6.4}$$

Indeed, Eq. (3.6.4) states that the $(2M)$ roots of $b_t(k)$ are the points $t = m$, for $-M \le m \le M$ and $m \neq k$, which fixes the polynomial up to a constant. That constant is determined by the condition $b_k(k) = 1$.

Example 3.6.1: For $N = 5$ and $d = N - 1 = 4$, the fourth degree Lagrange polynomials, constructed from Eq. (3.6.4), can be expanded in powers of t:

$$\begin{bmatrix} b_t(-2) \\ b_t(-1) \\ b_t(0) \\ b_t(1) \\ b_t(2) \end{bmatrix} = \frac{1}{24} \begin{bmatrix} 0 & 2 & -1 & -2 & 1 \\ 0 & -16 & 16 & 4 & -4 \\ 24 & 0 & -30 & 0 & 6 \\ 0 & 16 & 16 & -4 & -4 \\ 0 & -2 & -1 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ t^1 \\ t^2 \\ t^3 \\ t^4 \end{bmatrix}$$

The coefficient matrix is recognized as the inverse transposed of the basis matrix S:

$$S = \begin{bmatrix} 1 & -2 & 4 & -8 & 16 \\ 1 & -1 & 1 & -1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 16 \end{bmatrix} \quad \Rightarrow \quad S^{-T} = \frac{1}{24} \begin{bmatrix} 0 & 2 & -1 & -2 & 1 \\ 0 & -16 & 16 & 4 & -4 \\ 24 & 0 & -30 & 0 & 6 \\ 0 & 16 & 16 & -4 & -4 \\ 0 & -2 & -1 & 2 & 1 \end{bmatrix}$$

which verifies Eq. (3.6.2). □

We note that $b_t(k)$ can be written in the following analytical form, which shows the relation of the Lagrange interpolation filter to the ideal sinc-interpolation filter:

$$b_t(k) = \frac{\Gamma(M+1+t)\Gamma(M+1-t)}{\Gamma(M+1+k)\Gamma(M+1-k)} \cdot \frac{\sin(\pi(t-k))}{\pi(t-k)} \tag{3.6.5}$$

Some alternative expressions are as follows:

$$b_t(k) = (-1)^{M+k} \sum_{m=M+k}^{2M} \binom{M+t}{m} \binom{m}{M+k} (-1)^m \tag{3.6.6}$$

$$b_t(k) = \frac{(-1)^{M+1-k}\Gamma(M+1-t)}{(t-k)\Gamma(-M-t)\Gamma(M+1+k)\Gamma(M+1-k)} \tag{3.6.7}$$

and since the $b_t(k)$ sum up to one, we also have [156]:

$$b_t(k) = \left[\sum_{n=-M}^{M} \frac{b_t(n)}{b_t(k)}\right]^{-1} = \left[\sum_{n=-M}^{M} (-1)^{k-n} \frac{(M+k)!\,(M-k)!}{(M+n)!\,(M-n)!} \frac{t-k}{t-n}\right]^{-1} \tag{3.6.8}$$

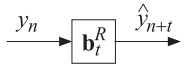
For polynomial orders $d < N-1$, one can still interpolate approximately and smoothly between the samples y_m. In this case, using $\mathbf{c} = G^T\mathbf{y} = (S^TS)^{-1}S^T\mathbf{y}$, we have:

$$\hat{y}_t = \sum_{i=0}^{d} c_i t^i = \mathbf{c}^T\mathbf{u}_t = \mathbf{y}^T G\mathbf{u}_t \equiv \mathbf{y}^T\mathbf{b}_t = \sum_{k=-M}^{M} b_t(k)\, y_k \tag{3.6.9}$$

where now

$$\mathbf{u}_t = \begin{bmatrix} 1 \\ t^1 \\ t^2 \\ \vdots \\ t^d \end{bmatrix}, \quad \mathbf{b}_t = G\mathbf{u}_t = S(S^TS)^{-1}\mathbf{u}_t \quad \Rightarrow \quad b_t(k) = \sum_{i=0}^{d} G_{ki}\, t^i \tag{3.6.10}$$

and shifting the origin $k = 0$ to the arbitrary time instant n, we obtain the interpolation formula for a shift t relative to the time instant n:



$$\boxed{\hat{y}_{n+t} = \sum_{k=-M}^{M} b_t(k)\, y_{n+k} = \sum_{k=-M}^{M} b_t^R(k)\, y_{n-k}} \tag{3.6.11}$$

where $b_t^R(k) = b_t(-k)$. Such formulas can also be used for prediction by choosing $t > M$ so that $n + t > n + M$, that is, it lies beyond the end of the filter range.

We can obtain closed-form expressions for the interpolation filters $b_t(k)$ for $d = 0, 1, 2, 3, 4$ and arbitrary M, by replacing in Eqs. (3.3.7)–(3.3.12) the variable m in $b_m(k)$ by the variable t. For example, for $d = 1, 2, 3, 4$, we have, respectively:

$$\begin{aligned}
b_t(k) &= \frac{1}{F_0} + \frac{tk}{F_2} \\
b_t(k) &= \frac{F_4}{D_4} + \frac{1}{F_2}tk - \frac{F_2}{D_4}(t^2+k^2) + \frac{F_0}{D_4}t^2k^2 \\
b_t(k) &= \frac{F_4}{D_4} + \frac{F_6}{D_8}tk - \frac{F_2}{D_4}(t^2+k^2) + \frac{F_0}{D_4}t^2k^2 - \frac{F_4}{D_8}(kt^3+tk^3) + \frac{F_2}{D_8}t^3k^3 \\
b_t(k) &= \frac{D_{12}}{D} + \frac{F_6}{D_8}tk - \frac{D_{10}}{D}(t^2+k^2) + \frac{E_8}{D}t^2k^2 - \frac{F_4}{D_8}(kt^3+tk^3) \\
&\quad + \frac{F_2}{D_8}t^3k^3 + \frac{D_8}{D}(t^4+k^4) - \frac{D_6}{D}(t^2k^4+k^2t^4) + \frac{D_4}{D}t^4k^4
\end{aligned} \tag{3.6.12}$$

where the required coefficient ratios are given by Eqs. (3.3.11) and (3.3.15). The interpolation filter (3.6.10) may be written in terms of the columns of $G = [\mathbf{g}_0, \mathbf{g}_1, \ldots, \mathbf{g}_d]$:

$$b_t(k) = \sum_{i=0}^{d} G_{ki}t^i = \sum_{i=0}^{d} g_i(k)\, t^i \quad \Rightarrow \quad \mathbf{b}_t = \sum_{i=0}^{d} \mathbf{g}_i t^i \tag{3.6.13}$$

This representation admits a convenient realization, known as a *Farrow structure*, which allows the changing of the parameter t on the fly without having to redesign the

filter. It is essentially a block-diagram realization of Eq. (3.6.13) written in nested form using Hörner's rule. For example, if $d = 3$, we have

$$\mathbf{b}_t = \mathbf{g}_0 + \mathbf{g}_1 t + \mathbf{g}_2 t^2 + \mathbf{g}_3 t^3 = \big((\mathbf{g}_3 t + \mathbf{g}_2)t + \mathbf{g}_1\big)t + \mathbf{g}_0 \tag{3.6.14}$$

Fig. 3.6.1 shows this realization where we replaced $\mathbf{g}_i$ by their reversed versions $\mathbf{g}_i^R$, which appear in the convolutional filtering equations. The parameter t appears only in the lower multipliers and can be independently controlled.

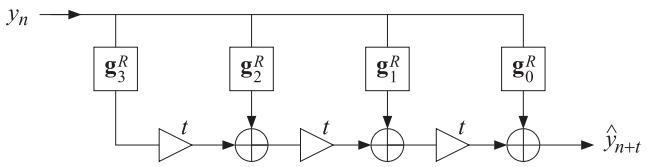


Fig. 3.6.1 Farrow structure for interpolating or predictive FIR filter.

The filtering equation (3.6.11) can also be written in a causal manner by setting $t = M + \tau$ and defining the causal filter, where $N = 2M + 1$:

$$\boxed{h_\tau(k) = b_{M+\tau}(M-k)}, \quad k = 0, 1, \ldots, N-1 \tag{3.6.15}$$

Replacing $n \to n-M$ and $k \to k-M$, Eq. (3.6.11) is transformed into a *causal* filtering operation that predicts the future sample $y_{n+\tau}$ from the present and past samples y_{n-k}, $k = 0, 1, \ldots, N-1$. The mapping of the time indices is explained in Fig. 3.6.2. The resulting filtering operation reads:

$$\boxed{\hat{y}_{n+\tau} = \sum_{k=0}^{N-1} h_\tau(k)\, y_{n-k}}, \quad \tau \geq 0 \tag{3.6.16}$$

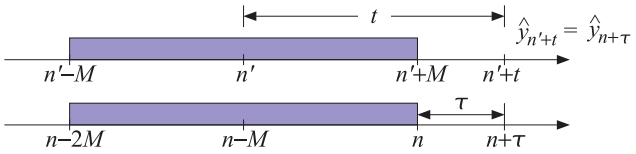


Fig. 3.6.2 Double-sided and causal predictive FIR filters, with $n' = n - M$ and $t = M + \tau$.

Since τ is any real number, the notation $n + \tau$ corresponds to the actual time instant $(n + \tau)T$ in seconds, where T is the sampling time interval. The filter $h_{-\tau}(k)$ may also be used for implementing a *fractional delay* as opposed to prediction, that is,

$$\boxed{\hat{y}_{n-\tau} = \sum_{k=0}^{N-1} h_{-\tau}(k)\, y_{n-k}} \quad \text{(fractional delay)} \tag{3.6.17}$$

The filters $b_t(k)$ and $h_\tau(k)$ satisfy the following polynomial-preserving moment constraints (being equivalent to $S^T\mathbf{b}_t = \mathbf{u}_t$), where $i = 0, 1, \ldots, d$:

$$\sum_{k=-M}^{M} k^i b_t(k) = t^i \quad \Rightarrow \quad \sum_{k=0}^{N-1} k^i h_\tau(k) = (-\tau)^i, \quad \sum_{k=0}^{N-1} k^i h_{-\tau}(k) = \tau^i \tag{3.6.18}$$

These constraints imply that Eqs. (3.6.16) and (3.6.17) are exact for polynomials of degree $r \leq d$. For any such polynomial $P(n)$, we have:

$$\sum_{k=0}^{N-1} h_{-\tau}(k) P(n-k) = P(n-\tau) \tag{3.6.19}$$

For example, we have for the monomial $P(n) = n^r$ with $r \leq d$:

$$\sum_{k=0}^{N-1} h_{-\tau}(k)\,(n-k)^r = \sum_{k=0}^{N-1} h_{-\tau}(k) \sum_{i=0}^{r} \binom{r}{i} n^{r-i}(-1)^i k^i$$

$$= \sum_{i=0}^{r} \binom{r}{i} n^{r-i}(-1)^i \sum_{k=0}^{N-1} k^i h_{-\tau}(k) = \sum_{i=0}^{r} \binom{r}{i} n^{r-i}(-1)^i \tau^i = (n-\tau)^r$$

It is in the sense of Eq. (3.6.19) that we may think of the transfer function of the filter $h_{-\tau}(k)$ as approximating the ideal fractional delay $z^{-\tau}$:

$$\sum_{k=0}^{N-1} h_{-\tau}(k) z^{-k} \simeq z^{-\tau} \tag{3.6.20}$$

Further insight into the nature of the approximation (3.6.20) can be gained by considering the Lagrange interpolation case, $d = N - 1$. From the definition of $h_{-\tau}(k) = b_{M-\tau}(M-k)$ and Eqs. (3.6.4) and (3.6.6), we obtain, for $k = 0, 1, \ldots, N-1$:

$$h_{-\tau}(k) = \prod_{\substack{i=0 \\ i\neq k}}^{N-1} \left(\frac{\tau - i}{k - i}\right) = \sum_{i=N-1-k}^{N-1} \binom{N-1-\tau}{i} \binom{i}{N-1-k} (-1)^{i-(N-1-k)} \tag{3.6.21}$$

The z-transform of $h_{-\tau}(k)$ is then,

$$\sum_{k=0}^{N-1} h_{-\tau}(k) z^{-k} = \sum_{k=0}^{N-1} \sum_{i=N-1-k}^{N-1} \binom{N-1-\tau}{i} \binom{i}{N-1-k} (-1)^{i-(N-1-k)} z^{-k}$$

$$= z^{-(N-1)} \sum_{i=0}^{N-1} \sum_{k=N-1-i}^{N-1} \binom{N-1-\tau}{i} \binom{i}{N-1-k} (-1)^{i-(N-1-k)} z^{N-1-k}$$

Changing summation variables and using the binomial expansion of $(z-1)^i$, we obtain,

$$\begin{aligned} \sum_{k=0}^{N-1} h_{-\tau}(k) z^{-k} &= z^{-(N-1)} \sum_{i=0}^{N-1} \sum_{j=0}^{m} \binom{N-1-\tau}{i} \binom{i}{j} (-1)^{i-j} z^j \\ &= z^{-(N-1)} \sum_{i=0}^{N-1} \binom{N-1-\tau}{i} (z-1)^i \end{aligned} \tag{3.6.22}$$

Applying the binomial identity,

$$(1+x)^\alpha = \sum_{m=0}^{\infty} \binom{\alpha}{i} x^i \tag{3.6.23}$$

with $x = z-1$ and $\alpha = N-1-\tau$, we have,

$$z^{N-1-\tau} = (1+z-1)^{N-1-\tau} = \sum_{i=0}^{\infty} \binom{N-1-\tau}{i} (z-1)^i \tag{3.6.24}$$

We recognize the sum in Eq. (3.6.22) to be the first N terms of (3.6.24). Thus, taking that sum to approximately represent $z^{N-1-\tau}$, we have,

$$\sum_{k=0}^{N-1} h_{-\tau}(k) z^{-k} \simeq z^{-(N-1)}\, z^{N-1-\tau} = z^{-\tau} \tag{3.6.25}$$

This approximation becomes exact whenever τ is an integer, say $\tau = m$, with $m = 0, 1, \ldots, N-1$. Indeed in this case, the summation range $0 \le i \le N-1$ in Eq. (3.6.22) can be restricted to $0 \le i \le N-1-m$ because the binomial coefficient vanishes whenever its (integer) arguments satisfy $N-1-m < i \le N-1$. We then have an ordinary binomial expansion for an integer power:

$$\sum_{k=0}^{N-1} h_{-m}(k) z^{-k} = z^{-(N-1)} \sum_{i=0}^{N-1-m} \binom{N-1-m}{i} (z-1)^i = z^{-(N-1)} (1+z-1)^{N-1-m} = z^{-m}$$

which implies the expected result $h_{-m}(k) = \delta(k-m)$. Eq. (3.6.22) is equivalent to *Newton's forward interpolation* formula. To see this, let us introduce the forward difference operator $\Delta = z-1$, or, $\Delta f_n = f_{n+1} - f_n$, and apply (3.6.22) in the time domain:

$$\hat{y}_{n-\tau} = \sum_{k=0}^{N-1} h_{-\tau}(k) y_{n-k} = \sum_{i=0}^{N-1} \binom{N-1-\tau}{i} \Delta^i y_{n-(N-1)} \tag{3.6.26}$$

This interpolates between the points $[y_{n-(N-1)}, \ldots, y_{n-1}, y_n]$ with τ measured backwards from the end-point y_n. We may measure the interpolation distance forward from the first point $y_{n-(N-1)}$ by defining $x = N-1-\tau$. Then, Eq. (3.6.26) reads,

$$\hat{y}_{n-(N-1)+x} = \sum_{i=0}^{N-1} \binom{x}{i} \Delta^i y_{n-(N-1)} \tag{3.6.27}$$

and setting $n = N-1$ so that the data range is $[y_0, y_1, \ldots, y_{N-1}]$, we obtain the usual way of writing Newton's polynomial interpolation formula:

$$\hat{y}_x = \sum_{i=0}^{N-1} \binom{x}{i} \Delta^i y_0 = \sum_{i=0}^{N-1} \frac{x(x-1)\cdots(x-i+1)}{i!}\, \Delta^i y_0 \tag{3.6.28}$$

We note also that Eq. (3.6.21) is valid for either even or odd values of N. For $N = 2, 3, 4$, we obtain for the corresponding filter coefficients:

$$\begin{bmatrix} h_{-\tau}(0) \\ h_{-\tau}(1) \end{bmatrix} = \begin{bmatrix} 1-\tau \\ \tau \end{bmatrix}, \quad \begin{bmatrix} h_{-\tau}(0) \\ h_{-\tau}(1) \\ h_{-\tau}(2) \end{bmatrix} = \frac{1}{2} \begin{bmatrix} (\tau-1)(\tau-2) \\ -2\tau(\tau-2) \\ \tau(\tau-1) \end{bmatrix}$$

$$\begin{bmatrix} h_{-\tau}(0) \\ h_{-\tau}(1) \\ h_{-\tau}(2) \\ h_{-\tau}(3) \end{bmatrix} = \frac{1}{6} \begin{bmatrix} -(\tau-1)(\tau-2)(\tau-3) \\ 3\tau(\tau-2)(\tau-3) \\ -3\tau(\tau-1)(\tau-3) \\ \tau(\tau-1)(\tau-2) \end{bmatrix} \tag{3.6.29}$$

and the corresponding interpolation formulas:

$$\begin{aligned} \hat{y}_{n-\tau} &= (1-\tau) y_n + \tau y_{n-1} \\ \hat{y}_{n-\tau} &= \frac{1}{2}(\tau-1)(\tau-2) y_n - \tau(\tau-2) y_{n-1} + \frac{1}{2}\tau(\tau-1) y_{n-2} \\ \hat{y}_{n-\tau} &= -\frac{1}{6}(\tau-1)(\tau-2)(\tau-3) y_n + \frac{1}{2}\tau(\tau-2)(\tau-3) y_{n-1} \\ &\quad - \frac{1}{2}\tau(\tau-1)(\tau-3) y_{n-2} + \frac{1}{6}\tau(\tau-1)(\tau-2) y_{n-3} \end{aligned} \tag{3.6.30}$$

Example 3.6.2: Fig. 3.6.3 shows in the top row an example of a Lagrange fractional-delay filter with $N = 3$ and polynomial order $d = N-1 = 2$ for the delay values $\tau = m/10$, $m = 1, 2, \ldots, 10$.

The bottom row is the case $N = 5$ and $d = N-1 = 4$ with delays τ extending over the interval $0 \le \tau \le 2$. This filter interpolates between the samples $[y_{n-4}, y_{n-3}, y_{n-2}, y_{n-1}, y_n]$. The chosen range of τ's spans the gaps between $[y_{n-2}, y_{n-1}, y_n]$. For the subrange $0 \le \tau \le 1$ which spans $[y_{n-1}, y_n]$, the magnitude response is greater than one, while it is less than one for the more central range $1 \le \tau \le 2$ which spans $[y_{n-2}, y_{n-1}]$. The following MATLAB code segment illustrates the generation of the upper two graphs:

```
f = linspace(0,1,1001); w= pi*f;          % frequencies 0 ≤ ω ≤ π
N=3; d=N-1; M = floor(N/2);               % d = N−1 for Lagrange interpolation

Hmag = []; Hdel = [];
for m=1:10,
   tau = m/10;                            % desired delays
   h = flip(lpinterp(N,d,M-tau));         % lpinterp is discussed in Sec. 3.8
   H = freqz(h,1,w);
   Hmag = [Hmag; 10*log10(abs(H))];       % magnitude responses in dB
   Delay = -angle(H)./w; Delay(1) = tau;
   Hdel = [Hdel; Delay];                  % phase delays
end

figure; plot(f,Hmag); figure; plot(f,Hdel);
```

The filters were calculated with the function `lpinterp` (from Sec. 3.8) with arguments $d = N-1$, $t = M-\tau$, with reversed output to account for the definition $h_{-\tau}(k) = b_{M-\tau}(M-k)$.

In both cases, we observe that the useful bandwidth of operation, within which both the phase delays have the correct values and the magnitude response is near unity, is fairly narrow extending to about $\omega = 0.2\pi$, or $f = f_s/10$ in units of the sampling rate f_s. □

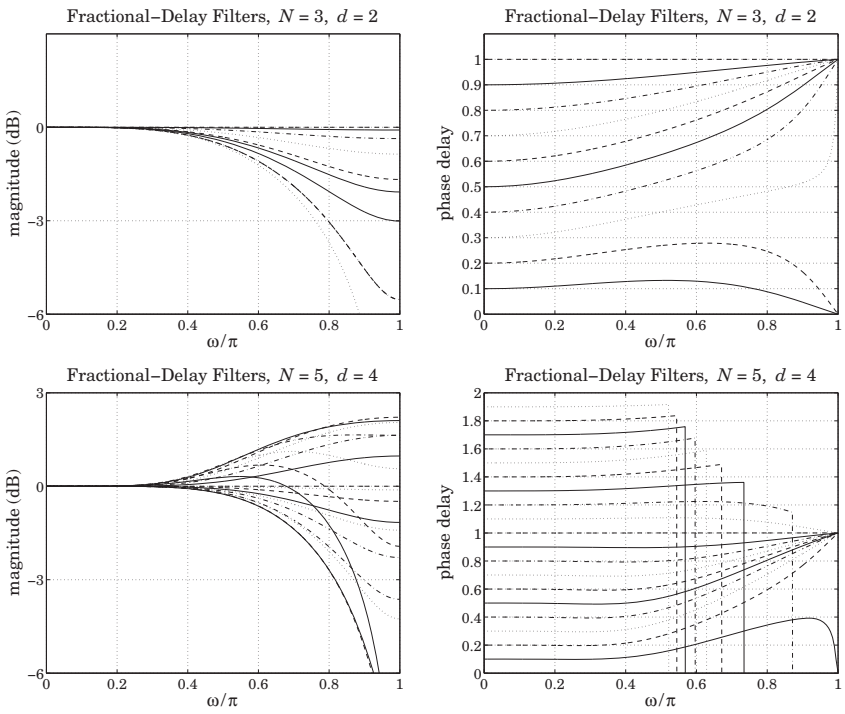


Fig. 3.6.3 Lagrange fractional-delay filters with $N = 3$.

References [152–173] contain further information on predictive FIR and fractional-delay filters. See also [174–187] for alternative implementations of fractional delay using maximally-flat and allpass filters. Ref. [162] provides a nice review of various approaches to the fractional-delay problem.

3.7 Minimum Variance Filters

Next we discuss the *equivalence* of the least-square polynomial fitting approach to the minimization of the NRR subject to linear moment constraints. In the actuarial context, such designs are referred to as "minimum $\mathcal{R}_0$" or "minimum variance" filters, as opposed to the "minimum $\mathcal{R}_s$" or "minimum roughness" filters— the nomenclature being explained in Sec. 4.2.

The projection properties of B may be used to calculate the NRR. For example, the property mentioned previously that the NRR of the filter $\mathbf{b}_0$ is the equal to the middle value $b_0(0)$ follows from Eq. (3.4.2). Using the symmetry of B, we have

$$B^T = B = B^2 = B^T B$$

Taking matrix elements, we have $B_{km} = (B^T)_{mk} = (B^T B)_{mk}$. But, B_{km} is the kth component of the mth column $\mathbf{b}_m$. Using a similar argument as in Eq. (3.2.13), we also have $(B^T B)_{mk} = \mathbf{b}_m^T \mathbf{b}_k$. Therefore,

$$\mathbf{b}_m^T \mathbf{b}_k = b_m(k)$$

For $k = m$, we have the diagonal elements of $B^T B = B$:

$$\mathcal{R} = \mathbf{b}_m^T \mathbf{b}_m = b_m(m) \tag{3.7.1}$$

These are recognized as the NRRs of the filters $\mathbf{b}_m$. In particular, for $m = 0$, we have $\mathcal{R} = \mathbf{b}_0^T \mathbf{b}_0 = b_0(0)$. Setting $k = 0$ in Eqs. (3.3.16)-(3.3.18), we find that the NRRs of the cases $d = 0, 1$, $d = 2, 3$, and $d = 4, 5$ are given by the coefficient ratios $1/F_0$, F_4/D_4, and D_{12}/D. Therefore:

$$\begin{aligned}
&(d = 0, 1) \qquad \mathcal{R} = \frac{1}{N} \\
&(d = 2, 3) \qquad \mathcal{R} = \frac{3(3M^2 + 3M - 1)}{(2M + 3)(4M^2 - 1)} \\
&(d = 4, 5) \qquad \mathcal{R} = \frac{15(15M^4 + 30M^3 - 35M^2 - 50M + 12)}{4(2M + 5)(4M^2 - 1)(4M^2 - 9)}
\end{aligned} \tag{3.7.2}$$

In the limit of large N or M, we have the approximate asymptotic expressions:

$$\begin{aligned}
&(d = 0, 1) \qquad \mathcal{R} = \frac{1}{N} \\
&(d = 2, 3) \qquad \mathcal{R} \simeq \frac{9/4}{N} = \frac{2.25}{N} \\
&(d = 4, 5) \qquad \mathcal{R} \simeq \frac{225/64}{N} = \frac{3.52}{N}
\end{aligned} \tag{3.7.3}$$

Thus, the noise reductions achieved by the quadratic/cubic and quartic/quintic cases are 2.25 and 3.52 times worse than that of the plain FIR averager of the same length N. Another consequence of the projection nature of B is:

$$B^T S = S, \qquad S^T B = S^T \tag{3.7.4}$$

Indeed, $B^T S = BS = S(S^T S)^{-1} S^T S = S$. Column-wise the first equation states that:

$$B^T [\mathbf{s}_0, \mathbf{s}_1, \ldots, \mathbf{s}_d] = [\mathbf{s}_0, \mathbf{s}_1, \ldots, \mathbf{s}_d] \qquad \Rightarrow \qquad B^T \mathbf{s}_i = \mathbf{s}_i, \quad i = 0, 1, \ldots, d$$

Thus, the basis vectors $\mathbf{s}_i$ remain *invariant* under projection, but that is to be expected because they already lie in $\mathbb{S}$. In fact, any other linear combination of them, such as Eq. (3.2.30), remains invariant under B, that is, $B^T \hat{\mathbf{y}} = \hat{\mathbf{y}}$.

This property answers the question: When are the smoothed values equal to the original ones, $\hat{\mathbf{y}} = \mathbf{y}$, or, equivalently, when is the error zero, $\mathbf{e} = 0$? Because $\mathbf{e} = \mathbf{y} - B^T \mathbf{y}$, the error will be zero if and only if $B^T \mathbf{y} = \mathbf{y}$, which means that $\mathbf{y}$ already *lies* in $\mathbb{S}$, that is, it is a linear combination of $\mathbf{s}_i$. This implies that the samples y_m are already dth order polynomial functions of m, as in Eq. (3.2.27).

The second equation in (3.7.4) implies certain *constraints* on the filters $\mathbf{b}_m$, which can be used to develop an alternative approach to the LPSM filter design problem in terms of minimizing the NRR subject to constraints. To see this, we write the $(d+1)\times N$ transposed matrix S^T column-wise:

$$S^T = [\mathbf{u}_{-M}, \ldots, \mathbf{u}_{-1}, \mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_M] \tag{3.7.5}$$

For example, in the $N = 5$, $d = 2$ case, we have:

$$S^T = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ -2 & -1 & 0 & 1 & 2 \\ 4 & 1 & 0 & 1 & 4 \end{bmatrix} \equiv [\mathbf{u}_{-2}, \mathbf{u}_{-1}, \mathbf{u}_0, \mathbf{u}_1, \mathbf{u}_2]$$

It is easily verified that the mth column $\mathbf{u}_m$ is simply

$$\mathbf{u}_m = \begin{bmatrix} 1 \\ m \\ m^2 \\ \vdots \\ m^d \end{bmatrix}, \qquad -M \le m \le M \tag{3.7.6}$$

which is the same as $\mathbf{u}_t$ at $t = m$, in terms of the definition (3.6.10). Using $B = GS^T$, we can express the LPSM filters $\mathbf{b}_m$ in terms of $\mathbf{u}_m$, as follows:

$$[\mathbf{b}_{-M}, \ldots, \mathbf{b}_{-1}, \mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_M] = B = GS^T = G[\mathbf{u}_{-M}, \ldots, \mathbf{u}_{-1}, \mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_M]$$

which implies:

$$\mathbf{b}_m = G\mathbf{u}_m = SF^{-1}\mathbf{u}_m \tag{3.7.7}$$

Multiplying by S^T, we find $S^T\mathbf{b}_m = S^TSF^{-1}\mathbf{u}_m = \mathbf{u}_m$, or,

$$S^T\mathbf{b}_m = \mathbf{u}_m \qquad \Rightarrow \qquad \begin{bmatrix} \mathbf{s}_0^T\mathbf{b}_m \\ \mathbf{s}_1^T\mathbf{b}_m \\ \vdots \\ \mathbf{s}_d^T\mathbf{b}_m \end{bmatrix} = \begin{bmatrix} 1 \\ m \\ \vdots \\ m^d \end{bmatrix} \tag{3.7.8}$$

These relationships are the column-wise equivalent of $S^TB = S^T$. Thus, each LPSM filter $\mathbf{b}_m$ satisfies $(d+1)$ linear constraints:

$$\mathbf{s}_i^T\mathbf{b}_m = m^i, \qquad i = 0, 1, \ldots, d \tag{3.7.9}$$

Writing the dot products explicitly, we have equivalently:

$$\boxed{\sum_{n=-M}^{M} n^i b_m(n) = m^i}, \qquad i = 0, 1, \ldots, d \tag{3.7.10}$$

In particular, for the steady-state LPSM filter $\mathbf{b}_0$, we have $\mathbf{u}_0 = [1, 0, 0, \ldots, 0]^T$, with ith component $\delta(i)$. Therefore, the constraint $S^T\mathbf{b}_0 = \mathbf{u}_0$ reads component-wise:

$$\boxed{\sum_{n=-M}^{M} n^i b_0(n) = \delta(i), \quad i = 0, 1, \ldots, d} \tag{3.7.11}$$

For $i = 0$, this is the usual DC constraint:

$$\sum_{n=-M}^{M} b_0(n) = 1 \tag{3.7.12}$$

and for $i = 1, 2, \ldots, d$:

$$\sum_{n=-M}^{M} n^i b_0(n) = 0 \tag{3.7.13}$$

The quantity in the left-hand side of Eq. (3.7.11) is called the ith *moment* of the impulse response $b_0(n)$. Because of the symmetric limits of summation over n and the symmetry of $b_0(n)$ about its middle, the moments (3.7.13) will be zero for odd i, and therefore are not extra constraints. However, for even i, they are nontrivial constraints.

These moments are related to the *derivatives* of the frequency response at $\omega = 0$. Indeed, defining,

$$B_0(\omega) = \sum_{n=-M}^{M} b_0(n)\, e^{-j\omega n}$$

and differentiating it i times, we have:

$$j^i B_0^{(i)}(\omega) = j^i \frac{d^i B_0(\omega)}{d\omega^i} = \sum_{n=-M}^{M} n^i b_0(n)\, e^{-j\omega n}$$

Setting $\omega = 0$, we obtain:

$$j^i B_0^{(i)}(0) = j^i \left.\frac{d^i B_0(\omega)}{d\omega^i}\right|_{\omega=0} = \sum_{n=-M}^{M} n^i b_0(n) \tag{3.7.14}$$

Thus, the moment constraints (3.7.12) and (3.7.13) are equivalent to the DC constraint and the *flatness* constraints on the frequency response at $\omega = 0$:

$$\boxed{B_0(0) = 1, \qquad B_0^{(i)}(0) = 0, \quad i = 1, 2, \ldots, d} \tag{3.7.15}$$

The larger the d, the more derivatives vanish at $\omega = 0$, and the flatter the response $B_0(\omega)$ becomes. This effectively increases the cutoff frequency of the lowpass filter—letting through more noise, but at the same time preserving more of the higher frequencies in the desired signal.

Figure 3.7.1 shows the magnitude response $|B_0(\omega)|$ for the cases $N = 7, 15$ and $d = 0, 2, 4$. The quadratic filters are flatter at DC than the plain FIR averager because of the extra constraint $B_0''(0) = 0$. Similarly, the quartic filters are even flatter because

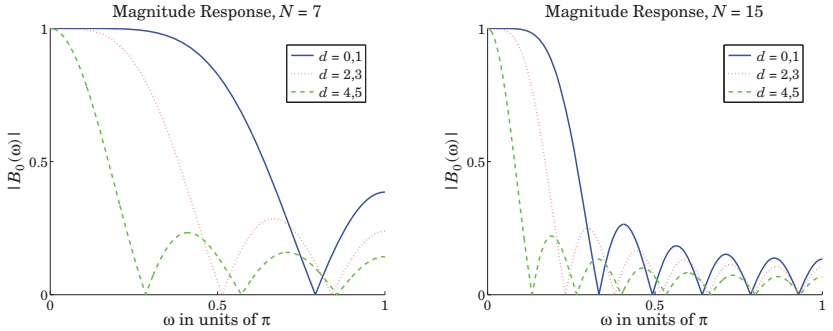


Fig. 3.7.1 LPSM filters of lengths $N = 7, 15$, and orders $d = 0, 2, 4$.

they satisfy two flatness conditions: $B_0''(0) = B_0^{(4)}(0) = 0$. The cutoff frequencies are *approximately* doubled and tripled in the cases $d = 2$ and $d = 4$, as compared to $d = 0$.

A direct consequence of the moment constraints (3.7.11) is that the moments of the input signal $y(n)$ are *preserved* by the filtering operation (3.2.33), that is,

$$\boxed{\sum_n n^i \hat{x}(n) = \sum_n n^i y(n), \qquad i = 0, 1, \ldots, d} \tag{3.7.16}$$

This can be proved easily working in the frequency domain. Differentiating the filtering equation $\hat{X}(\omega) = B_0(\omega)Y(\omega)$ i times, and using the product rules of differentiation, we obtain:

$$\hat{X}^{(i)}(\omega) = \sum_{j=0}^{i} \binom{i}{j} B_0^{(j)}(\omega) Y^{(i-j)}(\omega)$$

Setting $\omega = 0$ and using the moment constraints satisfied by the filter, $B_0^{(j)}(0) = \delta(j)$, we observe that only the $j = 0$ term will contribute to the above sum, giving:

$$\hat{X}^{(i)}(0) = B_0(0) Y^{(i)}(0) = Y^{(i)}(0), \qquad i = 0, 1, \ldots, d$$

which implies Eq. (3.7.16), by virtue of Eq. (3.7.14) as applied to $x(n)$ and $y(n)$.

The preservation of moments is a useful property in applications, such as spectroscopic analysis or ECG processing, in which the desired signal has one or more sharp peaks, whose widths must be preserved by the smoothing operation. In particular, the second moment corresponding to $i = 2$ in Eq. (3.7.16) is a measure of the square of the width [42–52,56,58,178].

The above moment constraints can be used in a direct way to design the LPSM filters. We consider first the more general problem of designing an optimum length-N filter that *minimizes* the NRR subject to $d + 1$ arbitrary moment constraints. That is, minimize

$$\mathcal{R} = \mathbf{b}^T \mathbf{b} = \sum_{n=-M}^{M} b(n)^2 = \min \tag{3.7.17}$$

subject to the $d + 1$ constraints, with a given $\mathbf{u} = [u_0, u_1, \ldots, u_d]^T$:

$$\mathbf{s}_i^T \mathbf{b} = \sum_{n=-M}^{M} n^i b(n) = u_i, \quad i = 0, 1, \ldots, d \quad \Rightarrow \quad S^T \mathbf{b} = \mathbf{u} \tag{3.7.18}$$

The minimization of Eq. (3.7.17) subject to (3.7.18) can be carried out with the help of Lagrange multipliers, that is, adding the constraint terms to the performance index:

$$\mathcal{J} = \mathbf{b}^T \mathbf{b} + 2 \sum_{i=0}^{d} \lambda_i (u_i - \mathbf{s}_i^T \mathbf{b}) = \mathbf{b}^T \mathbf{b} + 2 \boldsymbol{\lambda}^T (\mathbf{u} - S^T \mathbf{b}) \tag{3.7.19}$$

The gradient of $\mathcal{J}$ with respect to the unknown filter $\mathbf{b}$ is:

$$\frac{\partial \mathcal{J}}{\partial \mathbf{b}} = 2\mathbf{b} - 2S\boldsymbol{\lambda}$$

Setting the gradient to zero, and solving for $\mathbf{b}$ gives:

$$\mathbf{b} = S\boldsymbol{\lambda} = [\mathbf{s}_0, \mathbf{s}_1, \ldots, \mathbf{s}_d] \begin{bmatrix} \lambda_0 \\ \lambda_1 \\ \vdots \\ \lambda_d \end{bmatrix} = \sum_{i=0}^{d} \lambda_i \mathbf{s}_i$$

Component-wise this means that $b(n)$ has the polynomial form:

$$b(n) = \sum_{i=0}^{d} \lambda_i s_i(n) = \sum_{i=0}^{d} \lambda_i n^i, \qquad -M \le n \le M$$

The Lagrange multiplier vector $\boldsymbol{\lambda}$ is determined by imposing the desired constraint:

$$\mathbf{u} = S^T \mathbf{b} = S^T S \boldsymbol{\lambda} = F \boldsymbol{\lambda} \qquad \Rightarrow \qquad \boldsymbol{\lambda} = F^{-1} \mathbf{u}$$

resulting in the optimum $\mathbf{b}$:

$$\mathbf{b} = S\boldsymbol{\lambda} = SF^{-1}\mathbf{u} = S(S^T S)^{-1}\mathbf{u} = G\mathbf{u} \tag{3.7.20}$$

Since the solution minimizes the norm $\mathbf{b}^T\mathbf{b}$, it is recognized to be the *minimum-norm* solution of the $(d+1)\times N$ full-rank under-determined linear system $S^T\mathbf{b} = \mathbf{u}$, which can be obtained by the pseudoinverse of S^T, that is, $\mathbf{b} = (S^T)^+\mathbf{u}$, where according to Eq. (15.4.10), $(S^T)^+ = S(S^T S)^{-1}$. In MATLAB, we can simply write $\mathbf{b} = \text{pinv}(S^T)\mathbf{u}$.

Comparing this solution with Eqs. (3.7.7) and (3.7.8), we conclude that the LPSM filters $\mathbf{b}_m$ can be thought of as the optimum filters that have minimum NRR with constraint vectors $\mathbf{u} = \mathbf{u}_m$, that is, the minimization problems,

$$\boxed{\mathcal{R} = \mathbf{b}_m^T \mathbf{b}_m = \min, \quad \text{subject to} \quad S^T \mathbf{b}_m = \mathbf{u}_m} \tag{3.7.21}$$

have solutions,

$$\mathbf{b}_m = SF^{-1}\mathbf{u}_m = G\mathbf{u}_m, \quad -M \le m \le M \tag{3.7.22}$$

and putting these together as the columns of B, we obtain Eq. (3.2.31):

$$B = [\dots, \mathbf{b}_m, \dots] = G[\dots, \mathbf{u}_m, \dots] = GS^T = SF^{-1}S^T \tag{3.7.23}$$

In particular, the steady-state LPSM filter $\mathbf{b}_0$ minimizes the NRR with the constraint vector $\mathbf{u} = \mathbf{u}_0 = [1, 0, \dots, 0]^T$. This was precisely the problem first formulated and solved using Lagrange multipliers by Schiaparelli [36].

Similarly, the interpolating filter $\mathbf{b}_t = G\mathbf{u}_t$ of Eq. (3.6.10) can be thought of as the solution of the constrained minimization problem:

$$\mathcal{R} = \mathbf{b}^T\mathbf{b} = \min, \quad \text{subject to } S^T\mathbf{b} = \mathbf{u}_t, \quad \text{where } \mathbf{u}_t = [1, t, t^2, \dots, t^d]^T$$

3.8 Predictive Differentiation Filters

Going back to the polynomial fit of Eq. (3.6.9), that is,

$$\hat{y}_t = \sum_{i=0}^{d} c_i t^i = \mathbf{c}^T\mathbf{u}_t = \mathbf{y}^T G\mathbf{u}_t = \mathbf{y}^T\mathbf{b}_t, \quad \text{where} \quad \mathbf{b}_t = G\mathbf{u}_t, \tag{3.8.1}$$

we recall that the differentiation filters (3.2.24) were derived by differentiating (3.8.1) at $t = 0$, and therefore, they correspond to the center of the data vector $\mathbf{y}$:

$$\hat{y}_t\big|_{t=0} = c_0 = \mathbf{b}_0^T\mathbf{y} = \mathbf{g}_0^T\mathbf{y}$$
$$\dot{\hat{y}}_t\big|_{t=0} = c_1 = \mathbf{g}_1^T\mathbf{y}$$
$$\ddot{\hat{y}}_t\big|_{t=0} = 2c_2 = \mathbf{g}_2^T\mathbf{y}, \quad \text{etc.},$$

The first derivative at an arbitrary value of t is given by:

$$\dot{\hat{y}}_t = \mathbf{y}^T\dot{\mathbf{b}}_t, \quad \dot{\mathbf{b}}_t = G\dot{\mathbf{u}}_t$$

where the differentiation operation can be expressed as matrix multiplication:

$$\mathbf{u}_t = \begin{bmatrix} 1 \\ t \\ t^2 \\ \vdots \\ t^d \end{bmatrix} \;\Rightarrow\; \dot{\mathbf{u}}_t = \begin{bmatrix} 0 \\ 1 \\ 2t \\ \vdots \\ dt^{d-1} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 0 \\ 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 2 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & \cdots & d & 0 \end{bmatrix} \begin{bmatrix} 1 \\ t \\ t^2 \\ \vdots \\ t^{d-1} \\ t^d \end{bmatrix} \equiv \mathcal{D}\mathbf{u}_t \tag{3.8.2}$$

where $\mathcal{D}$ is the $(d+1)\times(d+1)$ matrix with the sequence of numbers $\{1, 2, \dots, d\}$ along its first subdiagonal and zeros everywhere else. Such a matrix can be constructed trivially in MATLAB, for example, by:

```
D = diag(1:d, -1);
```


It follows that the first-order differentiation filter is $\dot{\mathbf{b}}_t = G\mathcal{D}\mathbf{u}_t$. In particular, the differentiation filter at the sample point $t = m$ is $\dot{\mathbf{b}}_m = G\mathcal{D}\mathbf{u}_m$ and the corresponding estimated derivative:

$$\dot{\hat{y}}_m = \dot{\mathbf{b}}_m^T\mathbf{y} = \mathbf{u}_m^T\mathcal{D}^T G^T\mathbf{y}, \quad -M \le m \le M \tag{3.8.3}$$

Stacking these together into a column vector, we obtain:

$$\dot{\hat{\mathbf{y}}} = S\mathcal{D}^T G^T\mathbf{y} = \dot{B}^T\mathbf{y}, \quad \text{where} \quad \dot{B} = G\mathcal{D}S^T = SF^{-1}\mathcal{D}S^T \tag{3.8.4}$$

so that $\dot{B}$ has the $\dot{\mathbf{b}}_m$ as columns. Higher-order derivatives correspond to higher powers of the matrix $\mathcal{D}$, for example, $\ddot{\mathbf{u}}_t = \mathcal{D}^2\mathbf{u}_t$, and so on, with the highest non-trivial power being $\mathcal{D}^d$, because $\mathcal{D}^{d+1} = 0$, or equivalently, because the elements of $\mathbf{u}_t$ are monomials up to t^d. Therefore, the order-i differentiation matrix will be:

$$\boxed{B^{(i)} = SF^{-1}\mathcal{D}^i S^T}, \quad i = 0, 1, \dots, d \tag{3.8.5}$$

Centering the data vector $\mathbf{y}$ at time n and denoting the m-th column of $B^{(i)}$ by $\mathbf{b}_m^{(i)}$, we obtain the filtering equation for the i-th estimated derivative:

$$\hat{y}_{n+m}^{(i)} = \sum_{k=-M}^{M} b_m^{(i)}(k)\, y_{n+k} = \sum_{k=-M}^{M} b_m^{(i)}(-k)\, y_{n-k} \tag{3.8.6}$$

We note that at the data-vector center $m = 0$, we have $\mathbf{b}_0^{(i)} = \mathbf{g}_i$. For arbitrary t, we have $\mathbf{b}_t^{(i)} = G\mathcal{D}^i\mathbf{u}_t$ and we obtain the estimated/interpolated derivative:

$$\boxed{\hat{y}_{n+t}^{(i)} = \sum_{k=-M}^{M} b_t^{(i)}(k)\, y_{n+k} = \sum_{k=-M}^{M} b_t^{(i)}(-k)\, y_{n-k}} \tag{3.8.7}$$

As in Eq. (3.6.15), the redefinition $h_\tau^{(i)}(k) = b_{M+\tau}^{(i)}(M-k)$ will result into a causal version of the predictive differentiator filter, with Eq. (3.8.7) transforming into:

$$\boxed{\hat{y}_{n+\tau}^{(i)} = \sum_{k=0}^{N-1} h_\tau^{(i)}(k)\, y_{n-k}} \qquad \text{(causal predictive differentiator)} \tag{3.8.8}$$

One can easily obtain closed-form expressions for the differentiation filters $b_t^{(i)}(k)$ for $d = 0, 1, 2, 3, 4$ and arbitrary M, by replacing the variable m in Eqs. (3.3.7)–(3.3.12) by the variable t and differentiating i-times with respect to t. For example, for $d = 1, 2, 3, 4$, we differentiate Eqs. (3.6.12) once to get the first derivative:

$$\begin{aligned}
\dot{b}_t(k) &= \frac{k}{F_2} \\
\dot{b}_t(k) &= \frac{1}{F_2}k - \frac{F_2}{D_4}(2t) + \frac{F_0}{D_4}(2tk^2) \\
\dot{b}_t(k) &= \frac{F_6}{D_8}k - \frac{F_2}{D_4}(2t) + \frac{F_0}{D_4}(2tk^2) - \frac{F_4}{D_8}(3t^2k + k^3) + \frac{F_2}{D_8}(3t^2k^3) \\
\dot{b}_t(k) &= \frac{F_6}{D_8}k - \frac{D_{10}}{D}(2t) + \frac{E_8}{D}(2tk^2) - \frac{F_4}{D_8}(k3t^2 + k^3) \\
&\quad + \frac{F_2}{D_8}(3t^2k^3) + \frac{D_8}{D}(4t^3) - \frac{D_6}{D}(2tk^4 + k^2 4t^3) + \frac{D_4}{D}(4t^3k^4)
\end{aligned} \tag{3.8.9}$$

For the causal versions, we have for $d = 1$:

$$\begin{aligned}
h_\tau(k) &= \frac{1}{F_0} + \frac{(M+\tau)(M-k)}{F_2} = \frac{M(M+1) + 3(M+\tau)(M-k)}{M(M+1)(2M+1)} \\
\dot{h}_\tau(k) &= \frac{M-k}{F_2} = \frac{3(M-k)}{M(M+1)(2M+1)}
\end{aligned} \tag{3.8.10}$$

where $k = 0, 1, \ldots, N-1$. We note that $\dot{h}_\tau$ can be obtained by differentiating h_τ with respect to τ. The derivative filter is independent of τ because it corresponds to fitting a first-order polynomial. For $d = 2$, we have similarly,

$$\begin{aligned}
h_\tau(k) &= \frac{F_4}{D_4} + \frac{1}{F_2}(M+\tau)(M-k) - \frac{F_2}{D_4}\big((M+\tau)^2 + (M-k)^2\big) + \frac{F_0}{D_4}(M+\tau)^2(M-k)^2 \\
\dot{h}_\tau(k) &= \frac{1}{F_2}(M-k) - \frac{F_2}{D_4}2(M+\tau) + \frac{F_0}{D_4}2(M+\tau)(M-k)^2
\end{aligned} \tag{3.8.11}$$

where, we recall from Eq. (3.3.11),

$$\frac{F_4}{D_4} = \frac{3(3M^2+3M-1)}{(2M+3)(4M^2-1)}, \quad \frac{F_2}{D_4} = \frac{15}{(2M+3)(4M^2-1)}$$

$$\frac{F_0}{D_4} = \frac{45}{M(M+1)(2M+3)(4M^2-1)}, \quad \frac{1}{F_2} = \frac{3}{M(M+1)(2M+1)}$$

Example 3.8.1: For the case $N = 5$, $d = 2$, we had found in Eqs. (3.2.5) and (3.2.16) that:

$$S = [\mathbf{s}_0, \mathbf{s}_1, \mathbf{s}_2] = \begin{bmatrix} 1 & -2 & 4 \\ 1 & -1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \end{bmatrix}, \quad G = \frac{1}{35}\begin{bmatrix} -3 & -7 & 5 \\ 12 & -3.5 & -2.5 \\ 17 & 0 & -5 \\ 12 & 3.5 & -2.5 \\ -3 & 7 & 5 \end{bmatrix}$$

The corresponding first- and second-order differentiation matrices will be:

$$\dot{B} = G\mathcal{D}^1 S^T = \frac{1}{35}\begin{bmatrix} -27 & -17 & -7 & 3 & 13 \\ 6.5 & 1.5 & -3.5 & -8.5 & -13.5 \\ 20 & 10 & 0 & -10 & -20 \\ 13.5 & 8.5 & 3.5 & -1.5 & -6.5 \\ -13 & -3 & 7 & 17 & 27 \end{bmatrix}, \quad \mathcal{D}^1 = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 2 & 0 \end{bmatrix}$$

$$\ddot{B} = G\mathcal{D}^2 S^T = \frac{1}{35}\begin{bmatrix} 10 & 10 & 10 & 10 & 10 \\ -5 & -5 & -5 & -5 & -5 \\ -10 & -10 & -10 & -10 & -10 \\ -5 & -5 & -5 & -5 & -5 \\ 10 & 10 & 10 & 10 & 10 \end{bmatrix}, \quad \mathcal{D}^2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 2 & 0 & 0 \end{bmatrix}$$

The central columns agree with Eq. (3.2.25). The interpolating smoothing and first-order differentiation filters are given by:

$$\mathbf{b}_t = G\mathbf{u}_t = \frac{1}{35}\begin{bmatrix} -3 & -7 & 5 \\ 12 & -3.5 & -2.5 \\ 17 & 0 & -5 \\ 12 & 3.5 & -2.5 \\ -3 & 7 & 5 \end{bmatrix}\begin{bmatrix} 1 \\ t \\ t^2 \end{bmatrix} = \frac{1}{35}\begin{bmatrix} -3-7t+5t^2 \\ 12-3.5t-2.5t^2 \\ 17-5t^2 \\ 12+3.5t-2.5t^2 \\ -3+7t+5t^2 \end{bmatrix}$$

$$\dot{\mathbf{b}}_t = G\mathcal{D}\mathbf{u}_t = \frac{1}{35}\begin{bmatrix} -3 & -7 & 5 \\ 12 & -3.5 & -2.5 \\ 17 & 0 & -5 \\ 12 & 3.5 & -2.5 \\ -3 & 7 & 5 \end{bmatrix}\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 2 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ t \\ t^2 \end{bmatrix} = \frac{1}{35}\begin{bmatrix} -7+10t \\ -3.5-5t \\ -10t \\ 3.5-5t \\ 7+10t \end{bmatrix}$$

where $\dot{\mathbf{b}}_t$ can be obtained either by the indicated matrix multiplication or by simply differentiating $\mathbf{b}_t$ with respect to t. □

The MATLAB function `lpdiff` implements the design of the differentiation matrices:

```
B = lpdiff(N,d,i);        % differentiation filters
```

Like `lpsm`, it carries out a Gram-Schmidt QR-transformation on the monomial basis S and constructs the $B^{(i)}$ by:

$$S = QR, \quad Q^TQ = I, \quad R = \text{upper triangular}$$

$$G = S(S^TS)^{-1} = QR^{-T}$$

$$B^{(i)} = GD^iS^T = Q(R^{-T}D^iR^T)Q^T$$

The predictive/interpolating differentiation filters $\mathbf{b}_t^{(i)}$ are the minimum-norm solution of the under-determined linear system $S^T\mathbf{b} = \mathcal{D}^i\mathbf{u}_t$, or, equivalently the solution of the constrained minimization problem:

$$\mathcal{R} = \mathbf{b}^T\mathbf{b} = \min, \quad \text{subject to } S^T\mathbf{b} = \mathcal{D}^i\mathbf{u}_t$$

The MATLAB function `lpinterp` implements the design of predictive and interpolating differentiation filters, essentially carrying out the operation $\mathbf{b} = \text{pinv}(S^T)\mathcal{D}^i\mathbf{u}_t$:

```
b = lpinterp(N,d,t,i);
```
% local polynomial interpolation and differentiation filters

The case $i = 0$ corresponds to the predictive interpolation filters of Sec. 3.6. For the integer values $t = m$, $-M \le m \le M$, the filter $\mathbf{b}$ agrees with the columns of $B^{(i)}$.

Example 3.8.2: Fig. 3.8.1 illustrates the performance of the local polynomial differentiation filters on noiseless and noisy signals.

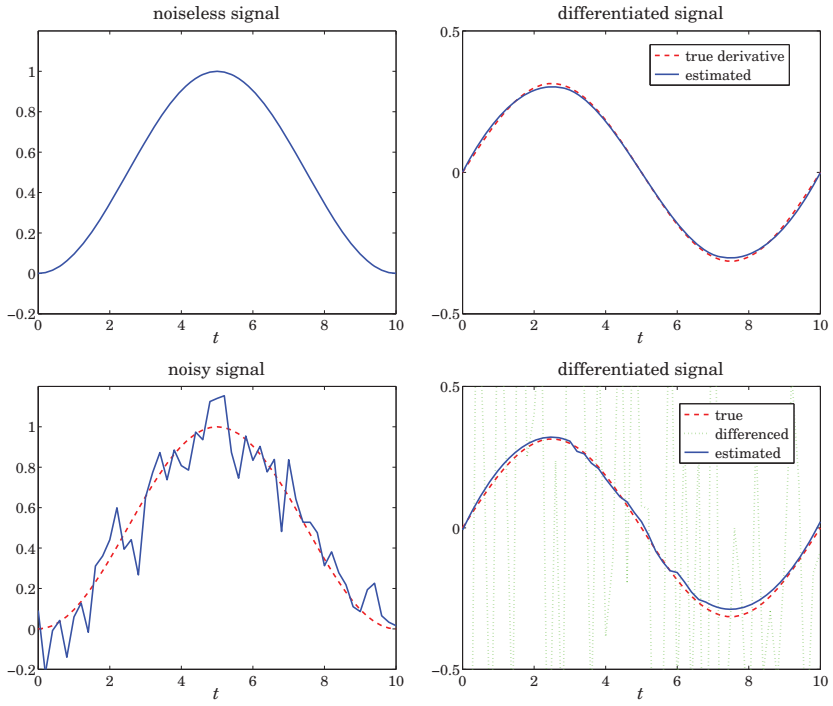


Fig. 3.8.1 Differentiating noisy signals.

The noiseless signal is a raised cosine $s(t) = 0.5 - 0.5\cos(\omega t)$, with $0 \le t \le T$ and $\omega = 2\pi/T$, so that it spans one cycle. Choosing a sampling time interval $\Delta t = T/L$, we can construct a noisy signal sampled at time instants $t_n = n\Delta t = nT/L$, $n = 0, 1, \ldots, L$, by adding zero-mean white gaussian noise v_n of variance, say σ^2, so that the noisy observations are:

$$y_n = s(t_n) + v_n\,, \quad n = 0, 1, \ldots, L$$

The first derivative of $s(t)$ is $\dot{s}(t) = 0.5\omega\sin(\omega t)$ and its samples, $\dot{s}(t_n) = 0.5\omega\sin(\omega t_n)$. The upper-left graph shows $s(t_n)$ versus t_n, with $T = 10$ and $L = 50$. The upper-right graph shows $\dot{s}(t_n)$ (dashed line) together with the estimated derivative (solid line) of the original signal $s(t_n)$ filtered through an LPSM differentiation filter designed with $N = 31$ and polynomial order $d = 3$. The output of the filter is divided by Δt in order to adjust its dimensions.

The bottom-left graph shows the noisy signal y_n. In the bottom-right graph, the output (solid line) of the same differentiation filter applied to the noisy signal y_n is compared with the true noiseless differentiated signal $\dot{s}_n$, as well as to the differenced signal $\mathrm{diff}(y)/\Delta t$. The following MATLAB code illustrates the generation of the bottom-right graph:

```
T = 10; L = 50; Dt =T/L; w = 2*pi/T; sigma = 0.1;
t = 0:Dt:T;
s = 0.5 - 0.5*cos(w*t);                  % noiseless signal

seed=100; randn('state',seed);
y = s + sigma * randn(1,length(s));      % noisy signal

N = 31; d = 3; B1 = lpdiff(N,d,1);       % first-order differentiation filter

sd = 0.5*w*sin(w*t);                     % derivative of s(t)
xd = lpfilt(B1,s)/Dt;                    % estimated derivative of s(t)
x1 = lpfilt(B1,y)/Dt;                    % estimated derivative from the noisy signal
yd = diff(y)/Dt;  td = t(2:end);         % differenced signal estimates the derivative

plot(t,sd,'--', td,yd,':', t,x1,'-');
```

The differencing operation amplifies the noise and renders the estimated derivative useless, whereas the local-polynomial derivative is fairly accurate. The filtering operation is carried out by the function `lpfilt`, which is explained in the next section. □

3.9 Filtering Implementations

In smoothing a length-L signal block y_n, $n = 0, 1, \ldots, L-1$, with a double-sided filter h_m, $-M \le m \le M$, the output signal $\hat{x}_n$ is given by the convolutional form:

$$\hat{x}_n = \sum_{m=\max(-M,n-L+1)}^{\min(n,M)} h_m y_{n-m}\,, \quad -M \le n \le L+M-1 \tag{3.9.1}$$

The length of $\hat{x}_n$ is $L+2M$, and the first $2M$ and last $2M$ output samples correspond to the input-on and input-off transients, while the central $L-2M$ points, $M \le n \le L-M-1$, correspond to the steady-state output computed from the steady-state version of Eq. (3.9.1):

$$\hat{x}_n = \sum_{m=-M}^{M} h_m y_{n-m}\,, \quad M \le n \le L-M-1 \tag{3.9.2}$$

The range of the output index n and the limits of summation in (3.9.1) are determined from the inequalities $-M \le m \le M$ and $0 \le n-m \le L-1$ that must be satisfied by the indices of h_m and y_{n-m}. However, only the subrange $\{\hat{x}_n\,,\ 0 \le n \le L-1\}$ is of interest since these output samples represent the smoothed values of the corresponding input samples $\{y_n\,,\ n = 0, 1, \ldots, L-1\}$. This is illustrated in Fig. 3.9.1.

The first and last M samples in the subrange $0 \le n \le L-1$ are still parts of the input-on and input-off transients. To clarify these remarks, we consider the case $L = 8$, $M = 2$. The full output (3.9.1) may be represented by the usual convolution matrix of the filter acting on the input signal block:

$$\begin{bmatrix} \hat{x}_{-2} \\ \hat{x}_{-1} \\ \cdots \\ \hat{x}_0 \\ \hat{x}_1 \\ \cdots \\ \hat{x}_2 \\ \hat{x}_3 \\ \hat{x}_4 \\ \hat{x}_5 \\ \cdots \\ \hat{x}_6 \\ \hat{x}_7 \\ \cdots \\ \hat{x}_8 \\ \hat{x}_9 \end{bmatrix} = \begin{bmatrix} h_{-2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ h_{-1} & h_{-2} & 0 & 0 & 0 & 0 & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ h_0 & h_{-1} & h_{-2} & 0 & 0 & 0 & 0 & 0 \\ h_1 & h_0 & h_{-1} & h_{-2} & 0 & 0 & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ h_2 & h_1 & h_0 & h_{-1} & h_{-2} & 0 & 0 & 0 \\ 0 & h_2 & h_1 & h_0 & h_{-1} & h_{-2} & 0 & 0 \\ 0 & 0 & h_2 & h_1 & h_0 & h_{-1} & h_{-2} & 0 \\ 0 & 0 & 0 & h_2 & h_1 & h_0 & h_{-1} & h_{-2} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & 0 & h_2 & h_1 & h_0 & h_{-1} \\ 0 & 0 & 0 & 0 & 0 & h_2 & h_1 & h_0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & h_2 & h_1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & h_2 \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ \cdots \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ \cdots \\ y_6 \\ y_7 \end{bmatrix}$$

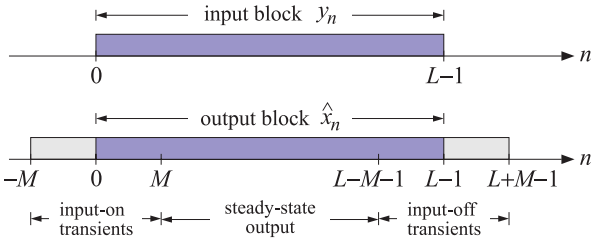


Fig. 3.9.1 Input and output signal blocks from a double-sided filter.

This matrix can be constructed in MATLAB with the built-in function `convmtx`, or with its sparse version `convmat`, or with the function `datamat`, the latter two being part of the OSP toolbox. Defining $\mathbf{h} = [h_{-M}, \ldots, h_0, \ldots, h_M]^T$, we have the syntax:

```
H = convmtx(h,L);        % built-in convolution matrix
H = convmat(h,L);        % sparse version of convmtx
H = datamat(h,L-1);      % used extensively in Chap. 15
```


Dropping the first and last two outputs, we obtain the outputs in the subrange $0 \le n \le 7$:

$$\hat{\mathbf{x}} = \begin{bmatrix} \hat{x}_0 \\ \hat{x}_1 \\ \cdots \\ \hat{x}_2 \\ \hat{x}_3 \\ \hat{x}_4 \\ \hat{x}_5 \\ \cdots \\ \hat{x}_6 \\ \hat{x}_7 \end{bmatrix} = \begin{bmatrix} h_0 & h_{-1} & h_{-2} & 0 & 0 & 0 & 0 & 0 \\ h_1 & h_0 & h_{-1} & h_{-2} & 0 & 0 & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ h_2 & h_1 & h_0 & h_{-1} & h_{-2} & 0 & 0 & 0 \\ 0 & h_2 & h_1 & h_0 & h_{-1} & h_{-2} & 0 & 0 \\ 0 & 0 & h_2 & h_1 & h_0 & h_{-1} & h_{-2} & 0 \\ 0 & 0 & 0 & h_2 & h_1 & h_0 & h_{-1} & h_{-2} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & 0 & h_2 & h_1 & h_0 & h_{-1} \\ 0 & 0 & 0 & 0 & 0 & h_2 & h_1 & h_0 \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ \cdots \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ \cdots \\ y_6 \\ y_7 \end{bmatrix} \equiv H\mathbf{y} \quad (3.9.3)$$

The first two and last two of these outputs are still transient and are being computed with only a subset of the filter coefficients, and therefore, may not adequately represent the corresponding smoothed values. This so-called "end-point problem" has been addressed repeatedly with a number of solutions.

One method that is widely used by the government to process census and business-cycle data (e.g., the X12-ARIMA method) is to backcast and forecast M estimated values at the beginning and end of the length-L input block, so that y_n is now defined over $-M \le n \le L-1+M$, and the desired output samples over the subrange $0 \le n \le L-1$ will be steady-state outputs being computed with the full filter.

Another method is to use different filters for the first M and last M outputs. For example, one can take the outputs $\hat{y}_{n+m}$ of the LPSM filters $b_m(k)$ to estimate the initial and final M transients, while using the central filter $b_0(k)$ for the steady-state outputs. Indeed, the first time index when one can use the steady-state filter $b_0(k)$ is $n = M$:

$$\hat{x}_M = \hat{y}_M = \sum_{k=-M}^{M} b_0(k)\, y_{M+k}$$

Instead of calculating the previous output $\hat{x}_{M-1}$ using the transient version of $b_0(k)$,

$$\hat{x}_{M-1} = \sum_{k=-(M-1)}^{M} b_0(k)\, y_{M-1+k}$$

one could estimate $\hat{x}_{M-1}$ using $\hat{y}_{M+m}$ with $m = -1$, that is, using $b_{-1}(k)$, and using $b_{-2}(k), b_{-3}(k), \ldots, b_{-M}(k)$ for the other initial M outputs:

$$\begin{aligned} \hat{x}_{M-1} &= \hat{y}_{M-1} = \sum_{k=-M}^{M} b_{-1}(k)\, y_{M+k} \\ \hat{x}_{M-2} &= \hat{y}_{M-2} = \sum_{k=-M}^{M} b_{-2}(k)\, y_{M+k} \\ &\vdots \\ \hat{x}_0 &= \hat{y}_{M-M} = \sum_{k=-M}^{M} b_{-M}(k)\, y_{M+k} \end{aligned} \quad (3.9.4)$$

Similarly, one can use the filters $b_m(k)$ for $m = 1, 2, \ldots, M$ to calculate the last M smoothed outputs, starting with the last steady-state output at $n = L-1-M$ and proceeding to the end $n = L-1$:

$$\begin{aligned}
\hat{x}_{L-M} &= \hat{y}_{L-1-M+1} = \sum_{k=-M}^{M} b_1(k)\, y_{L-1-M+k} \\
\hat{x}_{L-M+1} &= \hat{y}_{L-1-M+2} = \sum_{k=-M}^{M} b_2(k)\, y_{L-1-M+k} \\
&\vdots \\
\hat{x}_{L-1} &= \hat{y}_{L-1-M+M} = \sum_{k=-M}^{M} b_M(k)\, y_{L-1-M+k}
\end{aligned} \tag{3.9.5}$$

The following example illustrates the computational steps for the input-on, steady, and input-off output samples, where we denoted $b_{m,k} = b_m(k)$ for simplicity:

$$\hat{\mathbf{x}} = \begin{bmatrix} \hat{x}_0 \\ \hat{x}_1 \\ \cdots \\ \hat{x}_2 \\ \hat{x}_3 \\ \hat{x}_4 \\ \hat{x}_5 \\ \cdots \\ \hat{x}_6 \\ \hat{x}_7 \end{bmatrix} = \begin{bmatrix} b_{-2,-2} & b_{-2,-1} & b_{-2,0} & b_{-2,1} & b_{-2,2} & 0 & 0 & 0 \\ b_{-1,-2} & b_{-1,-1} & b_{-1,0} & b_{-1,1} & b_{-1,2} & 0 & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ b_{0,-2} & b_{0,-1} & b_{0,0} & b_{0,1} & b_{0,2} & 0 & 0 & 0 \\ 0 & b_{0,-2} & b_{0,-1} & b_{0,0} & b_{0,1} & b_{0,2} & 0 & 0 \\ 0 & 0 & b_{0,-2} & b_{0,-1} & b_{0,0} & b_{0,1} & b_{0,2} & 0 \\ 0 & 0 & 0 & b_{0,-2} & b_{0,-1} & b_{0,0} & b_{0,1} & b_{0,2} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & b_{1,-2} & b_{1,-1} & b_{1,0} & b_{1,1} & b_{1,2} \\ 0 & 0 & 0 & b_{2,-2} & b_{2,-1} & b_{2,0} & b_{2,1} & b_{2,2} \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ \cdots \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ \cdots \\ y_6 \\ y_7 \end{bmatrix} = H\mathbf{y}$$

In particular, for $N = 5$ and $d = 2$, the convolutional filtering matrix will be:

$$\hat{\mathbf{x}} = \begin{bmatrix} \hat{x}_0 \\ \hat{x}_1 \\ \cdots \\ \hat{x}_2 \\ \hat{x}_3 \\ \hat{x}_4 \\ \hat{x}_5 \\ \cdots \\ \hat{x}_6 \\ \hat{x}_7 \end{bmatrix} = \frac{1}{35} \begin{bmatrix} 31 & 9 & -3 & -5 & 3 & 0 & 0 & 0 \\ 9 & 13 & 12 & 6 & -5 & 0 & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ -3 & 12 & 17 & 12 & -3 & 0 & 0 & 0 \\ 0 & -3 & 12 & 17 & 12 & -3 & 0 & 0 \\ 0 & 0 & -3 & 12 & 17 & 12 & -3 & 0 \\ 0 & 0 & 0 & -3 & 12 & 17 & 12 & -3 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & -5 & 6 & 12 & 13 & 9 \\ 0 & 0 & 0 & 3 & -5 & -3 & 9 & 31 \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ \cdots \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ \cdots \\ y_6 \\ y_7 \end{bmatrix} = H\mathbf{y}$$

with entries obtained from the matrix B of Eq. (3.2.16):

$$B = \frac{1}{35} \begin{bmatrix} 31 & 9 & -3 & -5 & 3 \\ 9 & 13 & 12 & 6 & -5 \\ -3 & 12 & 17 & 12 & -3 \\ -5 & 6 & 12 & 13 & 9 \\ 3 & -5 & -3 & 9 & 31 \end{bmatrix} = [\mathbf{b}_{-2}, \mathbf{b}_{-1}, \mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2]$$

More generally, given any smoothing (or differentiation) matrix B whose central column contains the (reversed) steady-state filter, and its other columns, the (reversed) filters to be used for the initial and final transients, one can uniquely construct the corresponding $L \times L$ convolutional matrix H for filtering a length-L block of data $\mathbf{y}$.

The procedure is straightforward. First construct the ordinary full $(L+2M)\times L$ convolution matrix for the central filter, then delete its first M and last M rows, and finally, replace the first M and last M rows of the result by the transient filters.

The following MATLAB code segment illustrates the procedure, where the matrix B is assumed to have size $N \times N$, with $N = 2M+1$, with the central column being the reversed steady-state filter and the other columns, the reversed transient filters:

```
H = convmat(flip(B(:,M+1)), L);          % ordinary (L+2M)×L convolution matrix
H = H(M+1:L+M,:);                        % extract the L×L convolution submatrix
H(1:M, 1:N) = B(:,1:M)';                 % redefine upper-left M×L corner
H(L-M+1:L, L-N+1:L) = B(:,M+2:N)';       % redefine lower-right M×L corner
```

The function `flip` reverses the central column of B because `convmat` expects as input the actual filter, not its reverse. The above steps have been incorporated into the function `lpmat` with syntax:

```
H = lpmat(B,L);          % local polynomial filter matrix of size L×L
```

Once the $L \times L$ matrix H is constructed, the actual filtering of a length-L input block $\mathbf{y}$ is straightforward, that is, $\hat{\mathbf{x}} = H\mathbf{y}$, and efficient because H is defined as sparse.

An alternative way to structure the filtering operation is to directly use Eqs. (3.9.4) and (3.9.5) for the transient parts and the following equation for the steady part:

$$\hat{x}_n = \sum_{k=-M}^{M} b_0(k)\, y_{n+k}\,, \quad M \le n \le L-1-M \tag{3.9.6}$$

The following MATLAB code illustrates this approach:

```
y = B(:,1:M)' * x(1:N);                  % first M transient outputs
for n = M+1:L-M,                         % middle L−2M steady-state outputs
   y = [y; B(:,M+1)' * x(n-M:n+M)];      % filtered by central column of B
end
y = [y; B(:,M+2:N)' * x(L-N+1:L)];       % last M transient outputs
```

These steps are implemented in the MATLAB function `lpfilt2`. A faster version is the function `lpfilt`, which uses MATLAB's built-in filtering functions. Thus, three possible ways of computing the filtered output $\hat{\mathbf{x}}$ given a smoothing matrix B are as follows (assuming that $\mathbf{y}$ is a length-L column vector):

```
x_hat = lpmat(B,L)*y;        % use L×L convolution matrix constructed from B
x_hat = lpfilt2(B,y);        % use directly the filtering equations (3.9.4)–(3.9.6)
x_hat = lpfilt(B,y);         % fast version using the function filtdbl
```

The function `lpfilt` internally calls the function `filtdbl`, which uses the built-in function `conv` to implement the FIR filtering by the steady-state double-sided central filter. The following code segment shows the essential part of `lpfilt`:

```
x_hat = filtdbl(flip(B(:,M+1)), y);                % filter with the central column of B
x_hat(1:M) = B(:,1:M)' * y(1:N);                   % correct the first M transient outputs
x_hat(end-M+1:end) = B(:,M+2:N)' * y(end-N+1:end); % correct the last M transient outputs
```

where the function `filtdbl` has usage:

```
y = filtdbl(h,x);      % filtering by double-sided FIR filter
```

The function `filtdbl` is essentially the ordinary convolution of the length-$(2M+1)$ filter $\mathbf{h}$ and the length-L signal $\mathbf{x}$, with the first M and last M output points discarded. The result is equivalent to that obtained using the convolution submatrix, as for example, in Eq. (3.9.3). We note, in particular, that the B matrix that gives rise to (3.9.3) is:

$$B = \begin{bmatrix} h_0 & h_1 & h_2 & 0 & 0 \\ h_{-1} & h_0 & h_1 & h_2 & 0 \\ h_{-2} & h_{-1} & h_0 & h_1 & h_2 \\ 0 & h_{-2} & h_{-1} & h_0 & h_1 \\ 0 & 0 & h_{-2} & h_{-1} & h_0 \end{bmatrix}$$

and contains the reversed filter $\mathbf{h}$ in the central column and the transient subfilters in the other columns.

There are other methods of handling the end-point problem, most notably Musgrave's *minimum-revision* method that uses end-point asymmetric filters constructed from a given central filter $\mathbf{h}$. We will discuss it in detail in Sec. 9.8. Here, we note that the output of this method is a B matrix, which can be passed directly into the filtering function `lpfilt`. The MATLAB function `minrev` implements Musgrave's method:

```
B = minrev(h,R);      % Musgrave's minimum revision asymmetric filters
```

where R is a scalar parameter to be explained in Sec. 9.8. The method is widely used in the X-11 method of seasonal adjustment and trend extraction.

Example 3.9.1: Schiaparelli was the first one to systematically pose and solve the minimum-NRR filtering problem. He gave the solution to many specific cases, such as filter lengths $N = 5$–13, and polynomial orders $d = 3, 4$.

Here, we reproduce the example from Schiaparelli's paper on smoothing lunar observations, the signal y_n being a measure of the moon's influence on atmospheric effects. Fig. 3.9.2 shows 30 noisy observations (one for each lunar day) and their smoothed versions produced with an LPSM filter of length $N = 13$ and polynomial order $d = 3$ on the left, and $d = 4$ on the right (Schiaparelli's case).

The central filters for the $d = 3$ and $d = 4$ cases are:

$$\mathbf{b}_0 = \frac{1}{143}[-11,\ 0,\ 9,\ 16,\ 21,\ 24,\ 25,\ 24,\ 21,\ 16,\ 9,\ 0,\ -11]$$

$$\mathbf{b}_0 = \frac{1}{2431}[110,\ -198,\ -135,\ 110,\ 390,\ 600,\ 677,\ 600,\ 390,\ 110,\ -135,\ -198,\ 110]$$

The following program segment illustrates the computations:

```
Y = loadfile('schiaparelli.dat');      % data file available in the OSP toolbox
n = Y(:,1); y = Y(:,2);                % extract n and y_n from the columns of Y

N=13; d=3; M=floor(N/2);               % filter length and polynomial order
B = lpsm(N,d);                         % construct LPSM matrix B
x = lpfilt(B,y);                       % filter noisy observations
b0 = B(:,M+1);                         % middle column of B
x0 = filtdbl(b0,y);                    % filter with b0 only
plot(n,y,'.', n,x,'-', n,x0,'--');
```

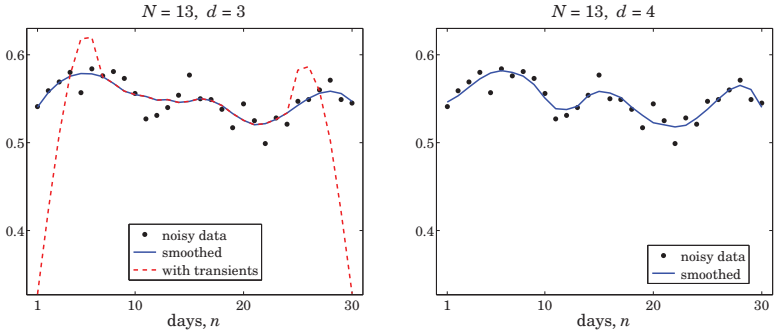



Fig. 3.9.2 Schiaparelli's smoothing example.

where the function `loadfile` extracts only the numerical data from the data file. In the left graph, we have also added the result of filtering with the steady-state filter $\mathbf{b}_0$, which illustrates the end-point problem. The two filtered curves differ only in their first 6 and last 6 points. □

Example 3.9.2: *Global Warming Trends.* Fig. 3.9.3 shows the annual average temperature anomalies (i.e., the differences with respect to the average of the period 1961–90) over the period 1856–2005 in the northern hemisphere. The data are available from the web site: `https://crudata.uea.ac.uk/cru/data/crutem2/`.

Five trend extraction methods are compared. In the upper left, a local polynomial smoothing filter was used of length $N = 65$ and polynomial order $d = 3$. The following MATLAB code illustrates the generation of that graph:

```
Y = loadfile('tavenh2v.dat');      % data file available in the OSP toolbox
n = Y(:,1); y = Y(:,14);           % extract n and y_n from Y

N = 65; d = 3; B = lpsm(N,d);      % design the LPSM matrix B
x = lpfilt(B,y);                   % smooth the data vector y

figure; plot(n,y,':', n,x,'-');
```

In the upper-right graph, a minimum-roughness, or minimum-R_s, Henderson filter was used with length $N = 65$, polynomial order $d = 3$, and smoothing order $s = 2$. Such filters are discussed in Sec. 4.2. The resulting trend is noticeably smoother than that of the LPSM filter on the upper-left.

The middle-left graph uses the SVD signal enhancement method, described in Chap. 15, with embedding order $M = 10$ and rank $r = 2$, with $K = 40$ iterations. The middle-right graph uses the Whittaker-Henderson smoothing method, discussed in Sec. 8.1, with smoothing order $s = 2$ and smoothing parameter $\lambda = 10^4$.

The lower left and right graphs use the Whittaker-Henderson method with the L_1 criterion with differentiation orders $s = 2$ and $s = 3$ and smoothing parameter $\lambda = 10$, implemented with the CVX package.[†] The $s = 2$ case represents the smoothed signal in piece-wise linear form, and the $s = 3$ case, in piece-wise parabolic form. This is further discussed in Sec. 8.7.

[†] `http://cvxr.com/cvx/`

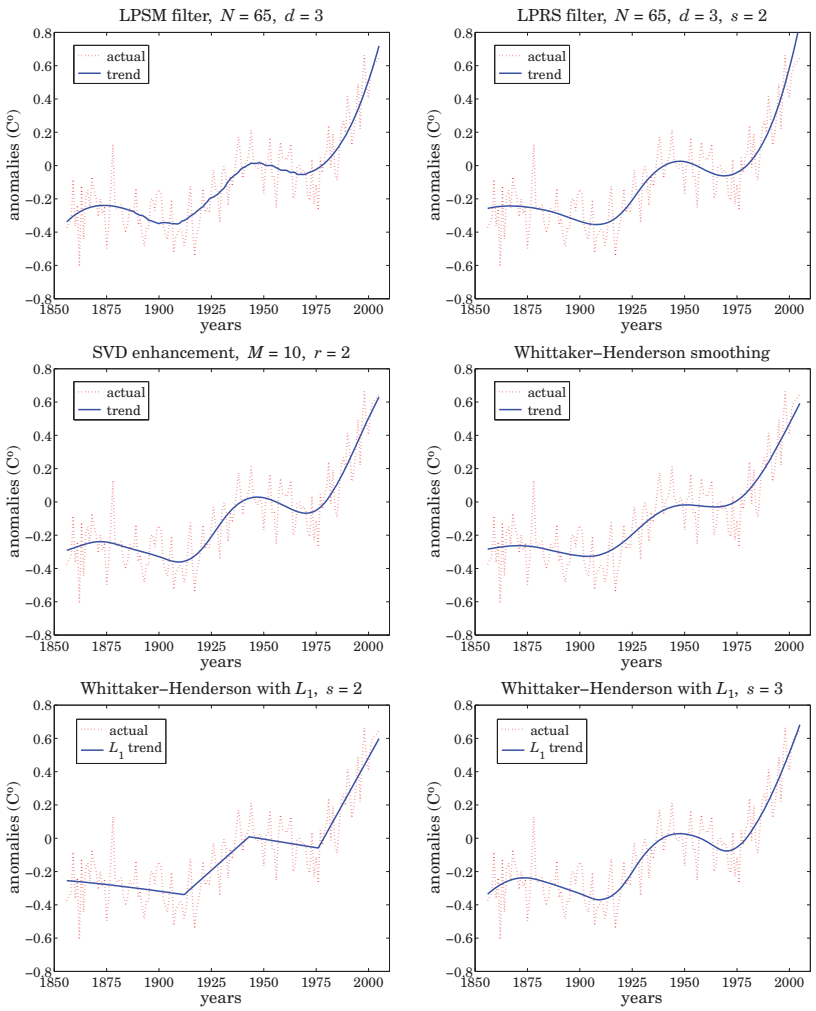


Fig. 3.9.3 Temperature trends determined by five methods.

The following MATLAB code segment illustrates the computation of the corresponding smoothed signals for these four methods:

```
N=65; d=3; s=2; x = lpfilt(lprs(N,d,s), y);   % minimum-R_s Henderson filter
M=10; r=2; K=40; x = svdenh(y,M,r,K);         % SVD enhancement method
la = 10000; s=2; x = whsm(y,la,s);            % Whittaker-Henderson smoothing

s = 2; la = 10; N = length(y);                % Whittaker-Henderson with L_1
D = diff(eye(N),s);                           % s-fold differentiation matrix
```



```
cvx_begin                                     % use CVX package
    variable x(N)
    minimize( sum_square(y-x) + la * norm(D*x,1) )
cvx_end
```

All methods adequately handle the end-point problem. Repeating the same filtering operation several times results in even smoother trend signals. For example, Fig. 3.9.4 shows the result of repeating the filtering operation two additional times. The following MATLAB code illustrates the generation of the left graph:

```
N = 65; d=3; B = lpsm(N,d); x = y;
for i=1:3, x = lpfilt(B,x); end
figure;  plot(n,y,':', n,x,'-');
```

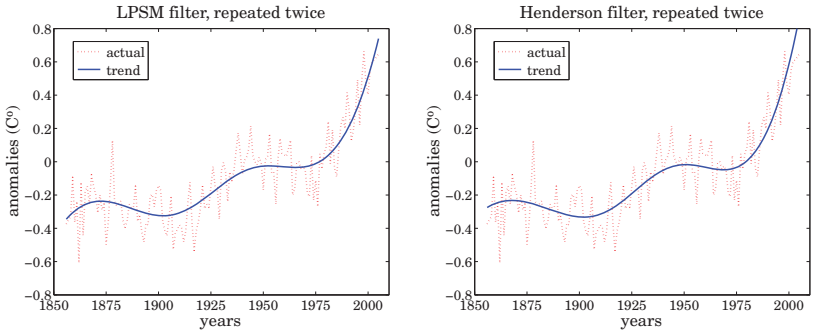


Fig. 3.9.4 Filtering repeated two additional times.

For the steady-state filters $B_0(\omega)$, filtering a total of K times is equivalent to an overall filter $[B_0(\omega)]^K$, an operation which makes a flat passband even flatter and a small stopband even smaller. The properties of iterated smoothing by local polynomial filters has been studied by De Forest, Schoenberg, and Greville [67,83,86].

Fig. 3.9.5 shows the estimated derivatives (solid line) of the temperature signal obtained by filtering it with the LPSM derivative filters, and compares them with the ordinary differencing operation, $\text{diff}(y)$, in MATLAB notation. Clearly, differencing is simply too noisy to give any usable results.

The upper two graphs compute the first derivative of the input by $\dot{x} = \text{lpfilt}(B_1, y)$ with the differentiator matrix obtained from $B_1 = \text{lpdiff}(N, d, i)$ with $N = 65$ and $i = 1$, and with $d = 1$ in the upper-left, and $d = 2$ in the upper-right graph. During the two periods of almost linear growth from 1910–1940 and 1970-2005, the derivative signal becomes an almost flat positive constant (i.e., the slope). During the other periods, the temperature signal has a very slow upward or downward trend and the derivative signal is almost zero.

We note the flat end-points in the the case $d = 1$, which are due to the fact that the asymmetric derivative filters are the same at the end-points ranges as shown in the first equation of (3.8.9). The case $d = 2$ estimates the end-point derivatives better and possibly indicates a faster than linear growth in recent years.

The lower-left graph uses a minimum-R_s derivative filter with $N = 65$, $d = 2$, and smoothness order $s = 3$, resulting in a noticeably smoother estimated derivative than the LPSM case (the W input in `lpdiff` is discussed in the next section.) Finally, the lower-right graph shows the second derivative computed with the filter $B_2 = \text{lpdiff}(N, d, i)$ with $i = 2$, and compares it with the second difference signal, $\text{diff}\big(\text{diff}(y)\big)$, which is even more noisy than the first difference. The following MATLAB code illustrates the computations:

```
d=1; i=1; B1 = lpdiff(N,d,i);                        % LPSM differentiation filters
plot(n, lpfilt(B1,y), n(2:end), diff(y),':');        % upper-left graph

d=2; i=1; B1 = lpdiff(N,d,i);
plot(n, lpfilt(B1,y), n(2:end), diff(y),':');        % upper-right graph

s=3; W = diag(hend(N,s));                            % Henderson weighting matrix
d=2; i=1; B1 = lpdiff(N,d,i,W);                      % LPRS differentiation filters
plot(n, lpfilt(B1,y), n(2:end), diff(y),':');        % lower-left graph

d=2; i=2; B2 = lpdiff(N,d,i);                        % second derivative filters
plot(n, lpfilt(B2,y), n(3:end), diff(y,2),':');      % lower-right graph
```

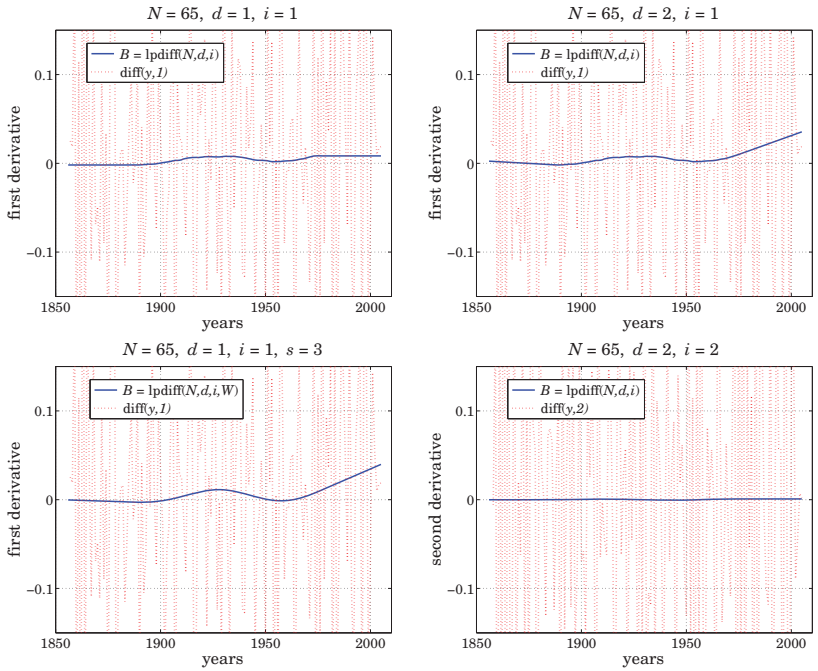


Fig. 3.9.5 Differentiated temperature signal.

The second derivative is essentially zero, being consistent with piecewise linear trends. Derivative signals can also be estimated for the SVD and Whittaker-Henderson methods. Since the outputs $\hat{x}_n$ of these methods are smooth signals, the corresponding derivatives

can be simply computed as the difference signals, $\text{diff}(\hat{x}_n)$, with comparable results as the local polynomial methods. □