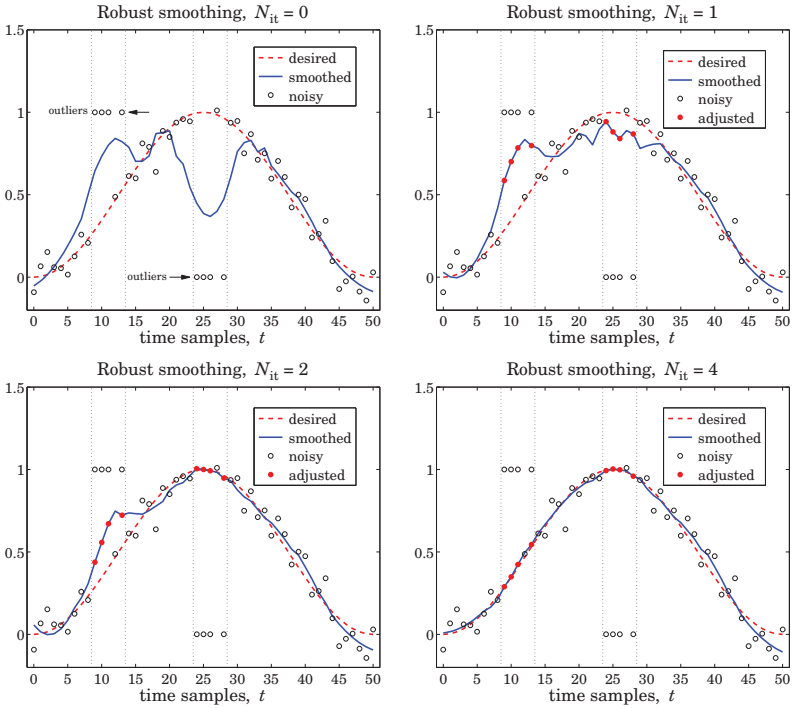


Fig. 4.5.3 Robust smoothing with outliers.

```
n1=10; n2=25; m = [-1 0 1 3];          % outlier indices relative to n1 and n2
y(n1+m+1)=1;  y(n2+m+1)=0;             % outlier values

Nit=4; K=4; x = rlpfilt(y,N,d,s,Nit,K);    % robust LP filtering

plot(t,x0,'--', t,y,'o', t,x,'-', n1+m,x(n1+m+1),'.', n2+m,x(n2+m+1),'.');
```

4.6 Problems

4.1 Using binomial identities, prove the equivalence of the three expressions in Eq. (4.4.14) for the maximally-flat filters. Then, show Eq. (4.4.15) and determine the proportionality constants indicated as (const.).

5

Local Polynomial Modeling

5.1 Weighted Local Polynomial Modeling

The methods of weighted least-squares local polynomial modeling and robust filtering can be generalized to unequally-spaced data in a straightforward fashion. Such methods provide enough flexibility to model a wide variety of data, including surfaces, and have been explored widely in recent years [188–231]. For equally-spaced data, the weighted performance index centered at time n was:

$$\mathcal{J}_n = \sum_{m=-M}^{M} \big(y_{n+m} - p(m)\big)^2 w(m) = \min, \quad p(m) = \sum_{r=0}^{d} c_i m^r \tag{5.1.1}$$

The value of the fitted polynomial $p(m)$ at $m = 0$ represents the smoothed estimate of y_n, that is, $\hat{x}_n = c_0 = p(0)$. Changing summation indices to $k = n + m$, Eq. (5.1.1) may be written in the form:

$$\mathcal{J}_n = \sum_{k=n-M}^{n+M} \big(y_k - p(k-n)\big)^2 w(k-n) = \min, \quad p(k-n) = \sum_{r=0}^{d} c_i (k-n)^r \tag{5.1.2}$$

For a set of N unequally-spaced observations $\{t_k, y(t_k)\}$, $k = 0, 1, \ldots, N-1$, we wish to interpolate smoothly at some time instant t, not necessarily coinciding with one of the observation times t_k, but lying in the interval $t_0 \le t \le t_{N-1}$. A generalization of the performance index (5.1.2) is to introduce a t-dependent window bandwidth h_t, and use only the observations that lie within that window, $|t_k - t| \le h_t$, to perform the polynomial fit:

$$\mathcal{J}_t = \sum_{|t_k - t| \le h_t} \big(y(t_k) - p(t_k - t)\big)^2 w(t_k - t) = \min, \quad p(t_k - t) = \sum_{r=0}^{d} c_r (t_k - t)^r \tag{5.1.3}$$

The estimated/interpolated value at t will be $\hat{x}_t = c_0 = p(0)$, and the estimated first derivative, $\hat{\dot{x}}_t = c_1 = \dot{p}(0)$, and so on for the higher derivatives, with $r!\, c_r$ representing the rth derivative. As illustrated in Fig. 5.1.1, the fitted polynomial,

$$p(x - t) = \sum_{r=0}^{d} c_r (x - t)^r, \quad t - h_t \le x \le t + h_t$$

is local in the sense that it fits the observations only within the local window $[t-h_t, t+h_t]$. The quantity $\hat{y}_k = p(t_k - t)$ represents the estimated value of the kth observation y_k within that window.

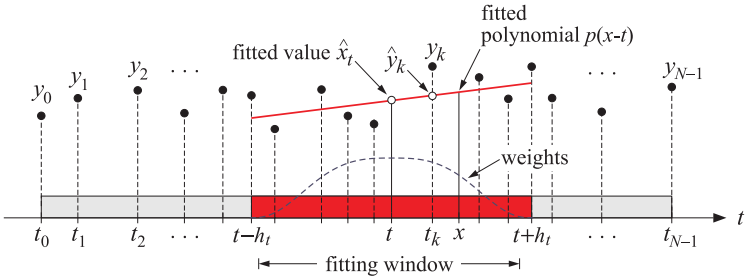


Fig. 5.1.1 Local polynomial modeling.

The weighting function $w(t_k - t)$ is chosen to have bandwidth $\pm h_t$. This can be accomplished by using a function $W(u)$ with finite support over the standardized range $-1 \le u \le 1$, and setting $u = (t_k - t)/h_t$:

$$w(t_k - t) = W\left(\frac{t_k - t}{h_t}\right) \tag{5.1.4}$$

Some typical choices for $W(u)$ are as follows [224]:

$$\begin{aligned}
&1.\ \text{Tricube}, && W(u) = (1 - |u|^3)^3\\
&2.\ \text{Bisquare}, && W(u) = (1 - u^2)^2\\
&3.\ \text{Triweight}, && W(u) = (1 - u^2)^3\\
&4.\ \text{Epanechnikov}, && W(u) = 1 - u^2\\
&5.\ \text{Gaussian}, && W(u) = e^{-\alpha^2 u^2/2}\\
&6.\ \text{Exponential}, && W(u) = e^{-\alpha|u|}\\
&7.\ \text{Rectangular}, && W(u) = 1
\end{aligned} \tag{5.1.5}$$

where all types have support $|u| \le 1$ and vanish for $|u| > 1$. A typical value for α in the gaussian and exponential cases is $\alpha = 2.5$. The curve shown in Fig. 5.1.1 is the tricube function; because it vanishes at $u = \pm 1$, the observations that fall exactly at the edges of the window do not contribute to the fit. The MATLAB function `locw` generates the above functions at any vector of values of u:

```
W = locw(u,type);      % local polynomial weighting functions W(u)
```

where `type` takes the values 1–7 as listed in Eq. (5.1.5). The bisquare, triweight, and Epanechnikov functions are special cases of the more general $W(u) = (1 - u^2)^s$, which may be thought of as the large-M limit of the Henderson weights; in the limit $s \to \infty$ they tend to a gaussian, as in the Krawtchouk case. The various window functions are depicted in Fig. 5.1.2.

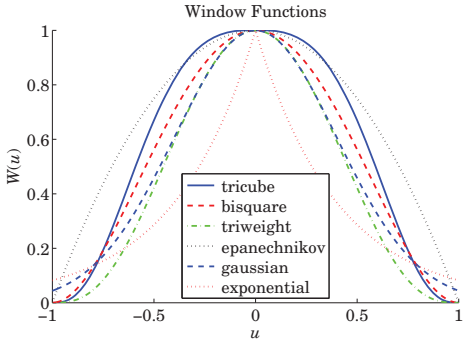


Fig. 5.1.2 Window functions.

Because of the assumed finite extent of the windows, the summation in Eq. (5.1.3) can be extended to run over all N observations, as is often done in the literature:

$$\mathcal{J}_t = \sum_{k=0}^{N-1} \big(y(t_k) - p(t_k - t)\big)^2 w(t_k - t) = \min, \quad p(t_k - t) = \sum_{r=0}^{d} c_r (t_k - t)^r \tag{5.1.6}$$

We prefer the form of Eq. (5.1.3) because it emphasizes the local nature of the fitting window. Let N_t be the number of observations that fall within the interval $[t-h_t, t+h_t]$. We may cast the performance index (5.1.3) in a compact matrix form by defining the $N_t \times 1$ vector of observations $\mathbf{y}_t$, the $N_t \times (d+1)$ basis matrix S_t, and the $N_t \times N_t$ diagonal matrix of weights by

$$\mathbf{y}_t = [\cdots, y(t_k), \cdots]^T, \qquad \text{for } t - h_t \le t_k \le t + h_t$$

$$S_t = \begin{bmatrix} \vdots & \vdots & & \vdots & & \vdots \\ 1 & (t_k - t) & \cdots & (t_k - t)^r & \cdots & (t_k - t)^d \\ \vdots & \vdots & & \vdots & & \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \mathbf{u}^T(t_k - t) \\ \vdots \end{bmatrix} \tag{5.1.7}$$

$$W_t = \text{diag}\big([\cdots, w(t_k - t), \cdots]\big)$$

where $\mathbf{u}^T(t_k - t)$ is the k-th row of S_t, defined in terms of the $(d+1)$-dimensional vector $\mathbf{u}^T(\tau) = [1, \tau, \tau^2, \ldots, \tau^d]$. For example, if $t - h_t < t_3 < t_4 < t_5 < t_6 < t + h_t$, then $N_t = 4$ and for a polynomial order $d = 2$, we have:

$$\mathbf{y}_t = \begin{bmatrix} y(t_3) \\ y(t_4) \\ y(t_5) \\ y(t_6) \end{bmatrix}, \quad S_t = \begin{bmatrix} 1 & (t_3 - t) & (t_3 - t)^2 \\ 1 & (t_4 - t) & (t_4 - t)^2 \\ 1 & (t_5 - t) & (t_5 - t)^2 \\ 1 & (t_6 - t) & (t_6 - t)^2 \end{bmatrix}$$

$$W_t = \begin{bmatrix} w(t_3 - t) & 0 & 0 & 0 \\ 0 & w(t_4 - t) & 0 & 0 \\ 0 & 0 & w(t_5 - t) & 0 \\ 0 & 0 & 0 & w(t_6 - t) \end{bmatrix}$$

With these definitions, Eq. (5.1.3) can be written as

$$\boxed{\mathcal{J}_t = (\mathbf{y}_t - S_t\mathbf{c})^T W_t (\mathbf{y}_t - S_t\mathbf{c}) = \min} \tag{5.1.8}$$

with solution for the coefficient vector $\mathbf{c} = [c_0, c_1, \ldots, c_d]^T$:

$$\boxed{\mathbf{c} = (S_t^T W_t S_t)^{-1} S_t^T W_t \mathbf{y}_t} \tag{5.1.9}$$

The quantity $\hat{\mathbf{y}}_t = S_t\mathbf{c}$ represents the polynomial estimate of the local observation vector $\mathbf{y}_t$. It can be written as

$$\boxed{\hat{\mathbf{y}}_t = B_t^T \mathbf{y}_t\,, \quad B_t = W_t S_t (S_t^T W_t S_t)^{-1} S_t^T} \tag{5.1.10}$$

where the $N_t \times N_t$ matrix B_t generalizes (4.1.5), and satisfies a similar polynomial-preserving property as (4.1.6),

$$B_t^T S_t = S_t \tag{5.1.11}$$

Defining the usual $(d+1)$-dimensional unit vector $\mathbf{u}_0 = [1, 0, \ldots, 0]^T$, we obtain the estimated value at time t by $\hat{x}_t = c_0 = \mathbf{u}_0^T\mathbf{c}$, and the first derivative by $\hat{\dot{x}}_t = c_1 = \mathbf{u}_1^T\mathbf{c}$, where $\mathbf{u}_1 = [0, 1, 0, \ldots, 0]^T$,

$$\boxed{\begin{aligned} \hat{x}_t &= \mathbf{u}_0^T (S_t^T W_t S_t)^{-1} S_t^T W_t \mathbf{y}_t \\ \hat{\dot{x}}_t &= \mathbf{u}_1^T (S_t^T W_t S_t)^{-1} S_t^T W_t \mathbf{y}_t \end{aligned}} \tag{5.1.12}$$

Thus, the effective estimation weights are:

$$\mathbf{h}(t) = W_t S_t (S_t^T W_t S_t)^{-1} \mathbf{u}_0\,, \quad \hat{x}_t = \mathbf{h}^T(t)\mathbf{y}_t \tag{5.1.13}$$

Component-wise, we can write:

$$\hat{x}_t = \mathbf{h}^T(t)\mathbf{y}_t = \sum_{|t_k - t| \le h_t} h_k(t)\, y_k \tag{5.1.14}$$

where $y_k = y(t_k)$ and

$$\boxed{h_k(t) = w(t_k - t)\mathbf{u}^T(t_k - t)\,(S_t^T W_t S_t)^{-1}\mathbf{u}_0} \tag{5.1.15}$$

We note that $\mathbf{u}_0, \mathbf{u}_1$ are related to the vector $\mathbf{u}(\tau)$ and its derivative by $\mathbf{u}_0 = \mathbf{u}(0)$ and $\mathbf{u}_1 = \dot{\mathbf{u}}(0)$. We also have,

$$S_t^T W_t S_t = \sum_{|t_k - t| \le h_t} \mathbf{u}(t_k - t)\mathbf{u}^T(t_k - t)\, w(t_k - t) \tag{5.1.16}$$

or, component-wise,

$$(S_t^T W_t S_t)_{ij} = \sum_{|t_k - t| \le h_t} (t_k - t)^{i+j}\, w(t_k - t)\,, \quad i, j = 0, 1, \ldots, d \tag{5.1.17}$$

The solution is particularly easy in the special cases $d = 0$, corresponding to local constant fitting, and $d = 1$, corresponding to local linear fits. The case $d = 0$ leads to the

so-called *kernel smoothing* approach first proposed by Nadaraya and Watson [188,189]. In this case $\mathbf{u}(\tau) = [1]$ and we find:

$$S_t^T W_t S_t = \sum_{|t_k - t| \le h_t} w(t_k - t)\,, \quad h_k(t) = \frac{w(t_k - t)}{\sum_{|t_k - t| \le h_t} w(t_k - t)}$$

$$\hat{x}_t = \sum_{|t_k - t| \le h_t} h_k(t)\, y_k = \frac{\sum_{|t_k - t| \le h_t} w(t_k - t)\, y_k}{\sum_{|t_k - t| \le h_t} w(t_k - t)} \qquad \text{(kernel smoothing)} \tag{5.1.18}$$

For $d = 1$, we have $\mathbf{u}(\tau) = [1, \tau]^T$, and we obtain

$$S_t^T W_t S_t = \sum_{|t_k - t| \le h_t} \begin{bmatrix} 1 & (t_k - t) \\ (t_k - t) & (t_k - t)^2 \end{bmatrix} w(t_k - t) \equiv \begin{bmatrix} s_0(t) & s_1(t) \\ s_1(t) & s_2(t) \end{bmatrix}$$

$$(S_t^T W_t S_t)^{-1} = \frac{1}{s_0(t)s_2(t) - s_1^2(t)} \begin{bmatrix} s_2(t) & -s_1(t) \\ -s_1(t) & s_0(t) \end{bmatrix}$$

which gives for the filter weights $h_k(t)$:

$$h_k(t) = w(t_k - t)\,\frac{s_2(t) - (t_k - t)\,s_1(t)}{s_0(t)s_2(t) - s_1^2(t)} \qquad \text{(locally linear fits)} \tag{5.1.19}$$

In general, the invertibility of $S_t^T W_t S_t$ requires that $N_t \ge d + 1$. The QR factorization can be used to implement the above computations efficiently. If the weight function $W(u)$ vanishes at the end-points $u = \pm 1$, as in the tricube case, then the window interval must exclude those end-points. In other words, the diagonal entries of W_t are assumed to be strictly positive. Defining U to be the diagonal square root factor of W_t and carrying out the QR factorization of the matrix US_t, we obtain the efficient algorithm:

$$\begin{aligned} &U = \text{sqrt}(W_t)\,, \quad U \text{ is diagonal so that } U^T = U \text{ and } W_t = U^T U = U^2 \\ &US_t = QR\,, \quad Q^T Q = I_{d+1}\,, \quad R = (d+1)\times(d+1) \text{ upper-triangular} \\ &\mathbf{c} = R^{-1} Q^T U \mathbf{y}_t \end{aligned} \tag{5.1.20}$$

The above steps are equivalent to reducing the problem to an ordinary unweighted least-squares problem, that is, $\mathbf{c}$ is recognized to be the unique least-squares solution of the full-rank, overdetermined, $N_t \times (d+1)$-dimensional system $(US_t)\mathbf{c} = U\mathbf{y}_t$. Indeed, it follows from Eq. (15.4.10) of Chap. 15 that $\mathbf{c}$ is given by:

$$\mathbf{c} = \big[(US_t)^T (US_t)\big]^{-1} (US_t)^T (U\mathbf{y}_t) = (S_t^T W_t S_t)^{-1} S_t^T W_t \mathbf{y}_t \tag{5.1.21}$$

where $\big[(US_t)^T (US_t)\big]^{-1} (US_t)^T$ is the pseudoinverse of US_t. The corresponding performance indices are equivalent:

$$\mathcal{J}_t = (\mathbf{y}_t - S_t\mathbf{c})^T W_t (\mathbf{y}_t - S_t\mathbf{c}) = \|U\mathbf{y}_t - US_t\mathbf{c}\|^2 = \min$$

In MATLAB the least-squares solution (5.1.21) can be obtained by the backslash operation: `c` $= (US_t)\backslash(U\mathbf{y}_t)$. The construction of the quantities $\mathbf{y}_t, S_t, W_t$ is straightforward. Given the column vectors of observation times and observations,

$$t_{\text{obs}} = [t_0, t_1, \ldots, t_{N-1}]^T, \quad y_{\text{obs}} = [y(t_0), y(t_1), \ldots, y(t_{N-1})]^T \tag{5.1.22}$$

we may determine, with the help of `locw`, the column vector of indices k for which t_k lies in the local window, and then carry out the procedure (5.1.21):

```
w = locw((tobs - t)/h_t, type);     % weights of all observation times relative to a given t and h_t
k = find(w);                        % column vector of indices of nonzero weights within window
yt = yobs(k);                       % column vector of corresponding local observations y_t
Wt = diag(w(k));                    % diagonal matrix of nonzero local weights W_t
St = [];
for r=0:d,
   St = [St, (tobs(k) - t).^r];     % construct local polynomial basis S_t column-wise
end
U = sqrt(Wt);                       % diagonal square root of W_t
c = (U*St)\(U*yt);                  % least-squares solution
```

Most of the w's obtained from the first line of code are zero, except for those t_k that lie within the local window $t \pm h_t$. The second line, `k = find(w)`, finds the latter. These steps have been incorporated into the MATLAB function `locpol`:

```
[xhat,C] = locpol(tobs,yobs,t,h,d,type);     % local polynomial modeling
```

where `tobs,yobs` are as in (5.1.22), `t,h` are L-dimensional vectors of times and bandwidths at which to carry out the fit, and `d,type` are the polynomial order and window type, with default values $d = 1$, `type` $= 1$. The output `xhat` is the L-dimensional vector of estimates $\hat{x}_t$, and `C` is an $L\times(d+1)$ matrix, whose ith row is the vector $[c_0, c_1, \ldots, c_d]$ of polynomial coefficients corresponding to the ith fitting time and bandwidth $t(i), h(i)$. Thus, the first column of `C` is the same as `xhat`, while the second column contains the first derivatives. Separating the first column of `C` into `xhat` is done only for convenience in using the function.

The choice of the bandwidth h_t is an important consideration that influences the quality of the estimate $\hat{x}_t$. Too large an h_t will oversmooth the signal but reduce the noise (i.e., increasing bias but lowering variance), and too small an h_t will undersmooth the signal and not reduce the noise as much (i.e., reducing bias and increasing variance).

Two simple bandwidth choices are the *fixed* and the *nearest-neighbor* bandwidths. In the fixed case, one chooses the same bandwidth at each fitting time, that is, $h_t = h$, for all t. In the nearest-neighbor case, one chooses a fixed number, say K, of observations to lie within each local window, where K is a fraction of the total number of observations N, that is, $K = \lfloor \alpha N \rfloor$, truncated to an integer, where $\alpha \leq 1$. Typical values are $\alpha = 0.2$–0.8. Given K, one calculates the distances of all the observation times from t, that is, $|t_k - t|$, $k = 0, 1, \ldots, N-1$, then sorts them in increasing order, and picks h_t to be the Kth shortest distance, and therefore, there will be K observations satisfying $|t_k - t| \leq h_t$. In summary, the fixed case selects $h_t = h$ but with varying N_t, and the nearest-neighbor case selects varying h_t but with fixed $N_t = K$.

The MATLAB function `locband` may be used to calculate the bandwidths h_t at each t, using either the fixed method, or the nearest-neighbor method:


```
h = locband(tobs,t,alpha,h0);     % bandwidth for local polynomial regression
```

where if $\alpha = 0$, the fixed bandwidth h_0 is selected, and if $0 < \alpha < 1$, the K-nearest bandwidths are selected, where t is a length-L vector of fitting times.

Example 5.1.1: As an example, consider the following 16 observation times t_{obs}, and 5 fitting times t, and choose $\alpha = 0.25$ so that $K = \alpha N = 0.25\times 16 = 4$:

$$t_{\text{obs}} = [0.5,\ 0.8,\ 1.1,\ 1.2,\ 1.8,\ 2.4,\ 2.5,\ 3.4,\ 3.5,\ 3.7,\ 4.0,\ 4.2,\ 4.9,\ 5.0,\ 5.1,\ 6.2]$$

$$t = [0.5,\qquad 1.5,\qquad 2.9,\qquad 3.6,\qquad 5.1]$$

then one finds the corresponding bandwidths for each of the five t's

$$\texttt{h = locband(tobs,t,0.25,0)} = [0.7,\ 0.7,\ 0.6,\ 0.6,\ 0.9]$$

and the corresponding local intervals, each containing $K = 4$ observation times:

h_t	$t - h_t$	t	$t + h_t$	included t_ks
0.7	−0.2	0.5	1.2	0.5, 0.8, 1.1, 1.2
0.7	0.8	1.5	2.2	0.8, 1.1, 1.2, 1.8
0.6	2.3	2.9	3.5	2.4, 2.5, 3.4, 3.5
0.6	3.5	4.1	4.7	3.5, 3.7, 4.0, 4.2
0.9	4.2	5.1	6.0	4.2, 4.9, 5.0, 5.1

By contrast, had we chosen a fixed bandwidth, say $h = 0.7$ (the average of the above five), then the corresponding intervals and included observation times would have been:

h_t	$t - h_t$	t	$t + h_t$	included t_ks
0.7	−0.2	0.5	1.2	0.5, 0.8, 1.1, 1.2
0.7	0.8	1.5	2.2	0.8, 1.1, 1.2, 1.8
0.7	2.2	2.9	3.6	2.4, 2.5, 3.4, 3.5
0.7	2.9	3.6	4.3	3.4, 3.5, 3.7, 4.0, 4.2
0.7	4.4	5.1	5.8	4.9, 5.0, 5.1

where now the number N_t of included observations depends on t. As can be seen from this example, both the nearest-neighbor and fixed bandwidth choices adapt well at the end-points of the available observations. □

Choosing t to be one of the observation times, $t = t_i$, Eq. (5.1.12) can be written in the simplified notation:

$$\hat{x}_i = \mathbf{u}_0^T (S_i^T W_i S_i)^{-1} S_i^T W_i \mathbf{y}_i \equiv \mathbf{h}_i^T \mathbf{y}_i\,, \quad \mathbf{h}_i^T = \mathbf{u}_0^T (S_i^T W_i S_i)^{-1} S_i^T W_i \tag{5.1.23}$$

where $\hat{x}_i, S_i, W_i, \mathbf{y}_i$ are the quantities $\hat{x}_t, S_t, W_t, \mathbf{y}_t$ evaluated at $t = t_i$. Component-wise,

$$\hat{x}_i = \sum_{|t_j - t_i| \leq h_i} \mathbf{u}_0^T (S_i^T W_i S_i)^{-1} \mathbf{u}(t_j - t_i)\, w(t_j - t_i)\, y_j = \sum_{|t_j - t_i| \leq h_i} H_{ij}\, y_j \tag{5.1.24}$$

where the matrix elements H_{ij} are,

$$H_{ij} = h_j(t_i) = \mathbf{u}_0^T (S_i^T W_i S_i)^{-1} \mathbf{u}(t_j - t_i)\, w(t_j - t_i) \tag{5.1.25}$$

Similarly, one may express $S_i^T W_i S_i$ and $S_i^T W_i \mathbf{y}_i$ as,

$$\begin{aligned} S_i^T W_i S_i &= \sum_{|t_j - t_i| \le h_i} \mathbf{u}(t_j - t_i)\mathbf{u}^T(t_j - t_i)\, w(t_j - t_i) \\ S_i^T W_i \mathbf{y}_i &= \sum_{|t_j - t_i| \le h_i} \mathbf{u}(t_j - t_i)\, w(t_j - t_i)\, y_j \end{aligned} \tag{5.1.26}$$

Because the factor $w(t_j - t_i)$ vanishes outside the local window $t_i \pm h_i$, the summations in (5.1.24) and (5.1.26) over t_j can be extended to run over all N observations. Defining the N-dimensional vectors $\hat{\mathbf{x}} = [\hat{x}_0, \hat{x}_1, \ldots, \hat{x}_{N-1}]^T$ and $\mathbf{y} = [y_0, y_1, \ldots, y_{N-1}]^T$, we may write (5.1.24) in the compact matrix form:

$$\hat{\mathbf{x}} = H\mathbf{y} \tag{5.1.27}$$

The filtering matrix H is also known as the "hat" matrix or the "smoothing" matrix. Its diagonal elements H_{ii} play a special role in bandwidth selection, where $w_0 = w(0)$,[†]

$$H_{ii} = h_i(t_i) = w_0\, \mathbf{u}_0^T (S_i^T W_i S_i)^{-1} \mathbf{u}_0 \tag{5.1.28}$$

5.2 Bandwidth Selection

There exist various automatic schemes for choosing the bandwidth. Such schemes may at best be used as guidelines. Ultimately, one must rely on one's judgment in making the final choice.

A popular bandwidth selection method is the so-called *cross-validation* criterion that selects the bandwidth h that minimizes the sum of squared prediction errors:

$$\mathrm{CV}(h) = \frac{1}{N} \sum_{i=0}^{N-1} (y_i - \hat{x}_i^-)^2 = \min \tag{5.2.1}$$

where $\hat{x}_i^-$ is the estimate or prediction of the sample $x_i = x(t_i)$ obtained by *deleting* the ith observation y_i and basing the estimation on the remaining observations, where we are assuming the usual additive-noise model $y(t_i) = x(t_i) + v(t_i)$ with $x(t_i)$ representing the desired signal to be extracted from $y(t_i)$. We show below that the predicted estimate $\hat{x}_i^-$ is related to the estimate $\hat{x}_i$ based on all observations by the relationship:

$$\hat{x}_i^- = \frac{\hat{x}_i - H_{ii} y_i}{1 - H_{ii}} \tag{5.2.2}$$

where H_{ii} is given by (5.1.28). It follows from (5.2.2) that the corresponding estimation errors will be related by:

$$y_i - \hat{x}_i^- = \frac{y_i - \hat{x}_i}{1 - H_{ii}} \tag{5.2.3}$$

[†] $w_0 = 1$ for all the windows in Eq. (5.1.5), but any other normalization can be used.

and therefore, the CV index can be expressed as:

$$\mathrm{CV}(h) = \frac{1}{N} \sum_{i=0}^{N-1} (y_i - \hat{x}_i^-)^2 = \frac{1}{N} \sum_{i=0}^{N-1} \left(\frac{y_i - \hat{x}_i}{1 - H_{ii}} \right)^2 = \min \tag{5.2.4}$$

A related selection criterion is based on the *generalized cross-validation* index, which replaces H_{ii} by its average over i, that is,

$$\mathrm{GCV}(h) = \frac{1}{N} \sum_{i=0}^{N-1} \left(\frac{y_i - \hat{x}_i}{1 - \bar{H}} \right)^2 = \min, \quad \bar{H} = \frac{1}{N} \sum_{i=0}^{N-1} H_{ii} = \frac{1}{N} \mathrm{tr}(H) \tag{5.2.5}$$

If the bandwidth is to be selected by the nearest-neighbor method, then, the CV and GCV indices may be regarded as functions of the fractional parameter α and minimized. Similarly, one could consider minimizing with respect to the polynomial order d, although in practice d is usually chosen to be 1 or 2.

Eq. (5.2.2) can be shown as follows. If the $t_j = t_i$ observation is deleted from Eq. (5.1.23), the corresponding optimum polynomial coefficients and optimum estimate will be given by

$$\mathbf{c}_- = (S_i^T W_i S_i)_-^{-1} (S_i^T W_i \mathbf{y}_i)_-, \quad \hat{x}_i^- = \mathbf{u}_0^T \mathbf{c}_-$$

where the minus subscripts indicate that the $t_j = t_i$ terms are to be omitted. It follows from Eq. (5.1.26) that

$$\begin{aligned} S_i^T W_i S_i &= (S_i^T W_i S_i)_- + w_0 \mathbf{u}_0 \mathbf{u}_0^T \\ S_i^T W_i \mathbf{y}_i &= (S_i^T W_i \mathbf{y}_i)_- + w_0 \mathbf{u}_0 y_i \end{aligned} \tag{5.2.6}$$

and then,

$$\mathbf{c}_- = [S_i^T W_i S_i - w_0 \mathbf{u}_0 \mathbf{u}_0^T]^{-1} [S_i^T W_i \mathbf{y}_i - w_0 \mathbf{u}_0 y_i] \tag{5.2.7}$$

Setting $F_i = S_i^T W_i S_i$ and noting that $\mathbf{c} = F_i^{-1} S_i^T W_i \mathbf{y}_i$ or $S_i^T W_i \mathbf{y}_i = F_i \mathbf{c}$, we may write,

$$\mathbf{c}_- = [F_i - w_0 \mathbf{u}_0 \mathbf{u}_0^T]^{-1} [F_i \mathbf{c} - w_0 \mathbf{u}_0 y_i]$$

Using the matrix inversion lemma, we have,

$$[F_i - w_0 \mathbf{u}_0 \mathbf{u}_0^T]^{-1} = F_i^{-1} + \frac{w_0 F_i^{-1} \mathbf{u}_0 \mathbf{u}_0^T F_i^{-1}}{1 - w_0 \mathbf{u}_0^T F_i^{-1} \mathbf{u}_0} \tag{5.2.8}$$

Noting that $H_{ii} = w_0 \mathbf{u}_0^T F_i^{-1} \mathbf{u}_0$, we obtain,

$$\begin{aligned} \mathbf{c}_- &= \left[F_i^{-1} + \frac{w_0 F_i^{-1} \mathbf{u}_0 \mathbf{u}_0^T F_i^{-1}}{1 - H_{ii}} \right] [F_i \mathbf{c} - w_0 \mathbf{u}_0 y_i] \\ &= \left[I + \frac{w_0 F_i^{-1} \mathbf{u}_0 \mathbf{u}_0^T}{1 - H_{ii}} \right] [\mathbf{c} - w_0 F_i^{-1} \mathbf{u}_0 y_i] \\ &= \mathbf{c} - w_0 F_i^{-1} \mathbf{u}_0 y_i + \frac{w_0 F_i^{-1} \mathbf{u}_0 [\mathbf{u}_0^T \mathbf{c} - w_0 \mathbf{u}_0^T F_i^{-1} \mathbf{u}_0 y_i]}{1 - H_{ii}} \end{aligned}$$

and since $\hat{x}_i = \mathbf{u}_0^T \mathbf{c}$, we find,

$$\mathbf{c}_- = \mathbf{c} - w_0 F_i^{-1} \mathbf{u}_0 y_i + \frac{w_0 F_i^{-1} \mathbf{u}_0 [\hat{x}_i - H_{ii} y_i]}{1 - H_{ii}} = \mathbf{c} + w_0 F_i^{-1} \mathbf{u}_0 \frac{\hat{x}_i - y_i}{1 - H_{ii}}$$

from which we find for $\hat{x}_i^- = \mathbf{u}_0^T \mathbf{c}_-$,

$$\hat{x}_i^- = \hat{x}_i + \frac{H_{ii}(\hat{x}_i - y_i)}{1 - H_{ii}} = \frac{\hat{x}_i - H_{ii} y_i}{1 - H_{ii}} \tag{5.2.9}$$

In practice, the CV and GCV indices are evaluated over a certain range of the smoothing parameter h or α to look for a minimum. The MATLAB function `locgcv` evaluates these indices at any vector of parameter values:

```
[GCV,CV] = locgcv(tobs,yobs,d,type,b,btype);    % CV and GCV evaluation
```

where `type` is the window type, `b` is either a vector of hs or a vector of αs at which to evaluate CV and GCV, and the string `btype` takes the values `'f'` or `'nn'` specifying whether the parameter vector `b` corresponds to a fixed or nearest-neighbor bandwidth.

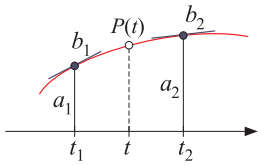
5.3 Local Polynomial Interpolation

The primary advantage of local polynomial modeling is its flexibility and ease of smoothing unequally-spaced data. Its main disadvantage is the potentially high computational cost, that is, the calculations (5.1.12) must be performed for each t, and generally a dense set of such t's might be required in order to get a visually smooth curve.

One way to cut down the cost is to evaluate the smoothed values $\hat{x}_t$ at a less dense grid of ts, and then interpolate smoothly between the computed points. This is akin to what plotting programs do by connecting the dots by straight-line segments (linearly interpolating)—the result being a visually continuous curve. But here, we can do better than just connecting the dots because we have available the slopes at each grid point. These slopes are contained in the second column of the fitting matrix C resulting from `locpol`, assuming of course that $d \geq 1$.

Consider two time instants t_1, t_2 at which the fitted signal values are a_1, a_2 with corresponding slopes b_1, b_2, as shown below. The lowest-degree polynomial $P(t)$ interpolating between the two points t_1, t_2 that matches the fitted values and their slopes at t_1 and t_2 is a cubic polynomial—the method being known as cubic Hermite interpolation. The four polynomial coefficients are fixed uniquely by the four conditions:

$$P(t_1) = a_1\,, \quad \dot{P}(t_1) = b_1$$
$$P(t_2) = a_2\,, \quad \dot{P}(t_2) = b_2$$



which result into the cubic polynomial, where $T = t_2 - t_1$,

$$\begin{aligned} P(t) = &\left(\frac{t-t_2}{T}\right)^2 \left[a_1 + (Tb_1 + 2a_1)\left(\frac{t-t_1}{T}\right)\right] \\ &+ \left(\frac{t-t_1}{T}\right)^2 \left[a_2 + (Tb_2 - 2a_2)\left(\frac{t-t_2}{T}\right)\right] \end{aligned} \tag{5.3.1}$$

For local-polynomial orders $d \geq 1$, we use Eq. (5.3.1) to interpolate at a denser grid of points between the less dense grid of fitted points. For the special case, $d = 0$, the slopes are not available and we can only use linear interpolation, that is,

$$P(t) = a_1 + (a_2 - a_1)\left(\frac{t - t_1}{T}\right) \tag{5.3.2}$$

The MATLAB function `locval` takes the output matrix `C` from `locpol` corresponding to a grid of fitting points t, and computes the interpolated points y_{grid} at the denser grid of points t_{grid}:

```
ygrid = locval(C,t,tgrid);    % interpolating local polynomial fits
```

The auxiliary function `locgrid` helps establish a uniform grid between the t points:

```
tgrid = locgrid(t, Ngrid);    % uniform grid with respect to t
```

which is simply a shorthand for,

```
tgrid = linspace(min(t), max(t), Ngrid);
```

Example 5.3.1: The motorcycle acceleration dataset [231] has served as a benchmark in many studies of local polynomial modeling and spline smoothing. The ordinate represents head acceleration (in units of g) during impact, and the abscissa is the time (in msec).

Fig. 5.3.1 shows a plot of the GCV index as a function of the nearest-neighbor fractional parameter α on the left, and as a function of the fixed bandwidth h on the right, for the two polynomial orders $d = 1, 2$.

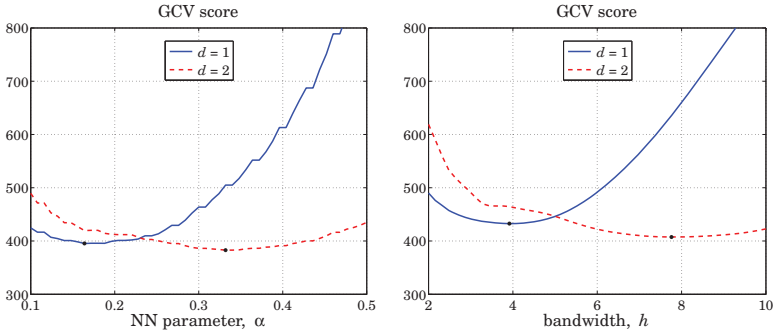


Fig. 5.3.1 GCV score for nearest-neighbor (left) and fixed bandwidths (right).

The "optimal" values of these parameters that minimize the GCV (and indicated by dots on the graphs) are as follows, where the subscripts indicate the value of d:

$$\alpha_1 = 0.16\,, \quad \alpha_2 = 0.33\,, \quad h_1 = 3.9\,, \quad h_2 = 7.8$$

The graphs (for $d = 1$) were produced by the MATLAB code:

```
Y = loadfile('mcyc.dat');                          % file included in the OSP toolbox
tobs = Y(:,1);  yobs = Y(:,2);                     % 133 data points

alpha = linspace(0.1, 0.5, 51);                    % vary over 0.1 ≤ α ≤ 0.5

d=1; type=1;
gcv = locgcv(tobs,yobs,d,type,alpha,'nn');         % GCV as function of α
[F,i] = min(gcv);  alpha1 = alpha(i);              % minimum at α = α1

figure; plot(alpha,gcv);                           % left graph

h = linspace(2, 10, 51);                           % vary over 2 ≤ h ≤ 10
gcv = locgcv(tobs,yobs,d,type,h,'f');              % GCV as function of h
[F,i] = min(gcv);  h1 = h(i);                      % minimum at h = h1
```

Fig. 5.3.2 shows the local polynomial fits corresponding to the above optimal parameter values. The left graph shows the nearest-neighbor cases for $d = 1, 2$, and the right graph, the fixed bandwidth cases. The tricube window was used (`type=1`).

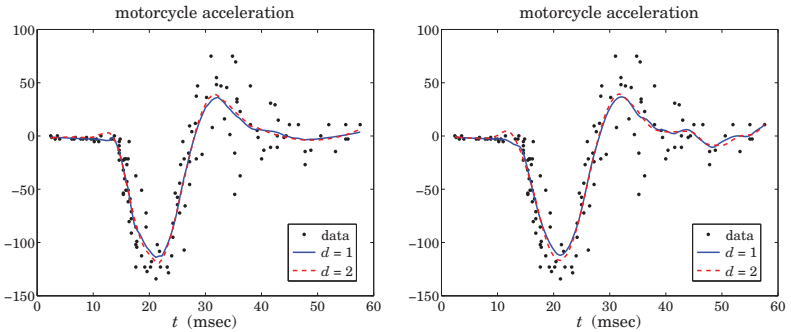


Fig. 5.3.2 Nearest-neighbor (left) and fixed bandwidths (right).

In all cases, the actual fitting was performed at 100 equally-spaced points t within the observation range t_{obs} and were connected by straight-line segments by the plotting program, instead of being interpolated by `locval`. Continuing with the above MATLAB code, the graphs were generated by

```
t = locgrid(tobs,101);                             % equally-spaced fitting times

h = locband(tobs, t, alpha1, 0);                   % NN bandwidths at each t
x1 = locpol(tobs,yobs,t,h,1,type);                 % fit at times t with d = 1

h = locband(tobs, t, alpha2, 0);
x2 = locpol(tobs,yobs,t,h,2,type);                 % fit at times t with d = 2

figure; plot(tobs,yobs,'.', t,x1,'-', t,x2,'--');  % left graph

h = locband(tobs, t, 0, h1);                       % fixed bandwidths at each t
x1 = locpol(tobs,yobs,t,h,1,type);                 % fit at times t with d = 1

h = locband(tobs, t, 0, h2);
```



```
x2 = locpol(tobs,yobs,t,h,2,type);                 % fit at times t with d = 2

figure; plot(tobs,yobs,'.', t,x1,'-', t,x2,'--');  % right graph
```

Fig. 5.3.3 demonstrates the Hermite interpolation procedure. The fitting times are 20 equally-spaced points spanning the observation interval t_{obs}. The 20 fitted points are then interpolated at 100 equally-spaced points over t_{obs}. The interpolated curves are essentially identical to those fitted earlier at 100 points.

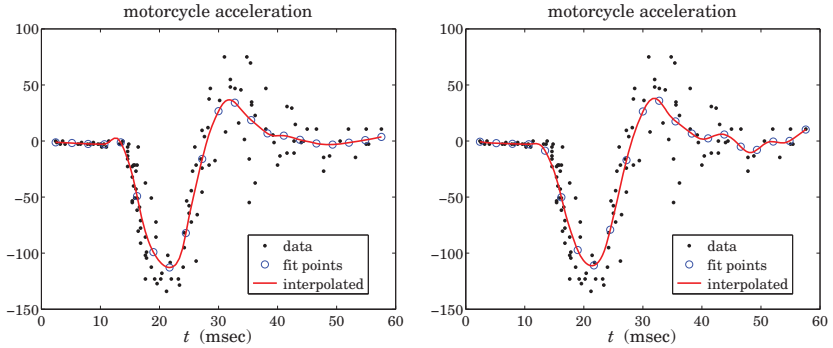


Fig. 5.3.3 Nearest-neighbor (left) and fixed bandwidths (right).

The polynomial order was $d = 1$ and the bandwidth parameters were $\alpha_1 = 0.21$ for the left graph and $h_1 = 4.4$ for the right one. The left graph was generated by the code segment:

```
tf = locgrid(tobs,21);                             % fitting times

h = locband(tobs, tf, alpha1, 0);                  % NN bandwidths at tf
[xf,C] = locpol(tobs,yobs,tf,h,1,type);            % fitted values and derivatives

tint = locgrid(tf,101);                            % interpolation times
xint = locval(C, tf, tint);                        % interpolated values

figure; plot(tobs,yobs,'.', tf,xf,'o', tint,xint,'-');
```

Example 5.3.2: The ethanol dataset [230] is also a benchmark example for smoothing techniques. The ordinate NO_x represents nitric oxide concentrations in the engine exhaust gases, and the abscissa E is the equivalence ratio, which is a measure of the richness of the ethanol/air mixture.

The GCV and CV bandwidth selection criteria tend sometimes to result in undersmoothed signals. This can be seen in Fig. 5.3.4 in which the GCV criterion for fixed bandwidth selects the values $h_1 = 0.039$ and $h_2 = 0.058$, for orders $d = 1, 2$.

As can be seen, the resulting fits are jagged, and can benefit form increasing the fitting bandwidth somewhat. The minima of the GCV plot are fairly broad and any neighboring values of the bandwidth would be just as good in terms of the GCV value. A similar effect happens in this example for the nearest-neighbor bandwidth method, in which the GCV criterion selects the value $\alpha = 0.19$ corresponding to jagged graph (not shown). Fig. 5.3.5

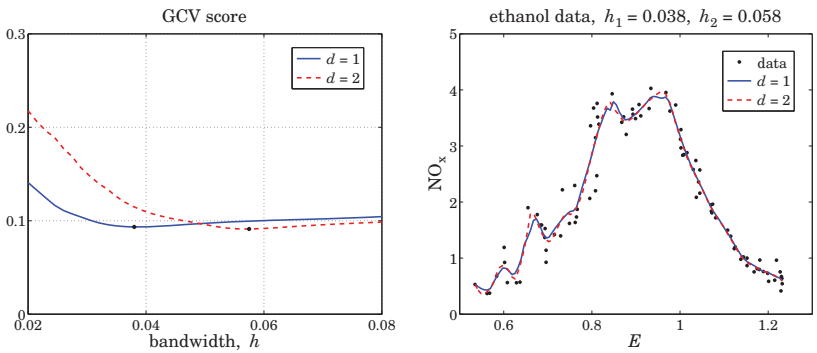


Fig. 5.3.4 GCV and local polynomial fits with $d = 1, 2$.

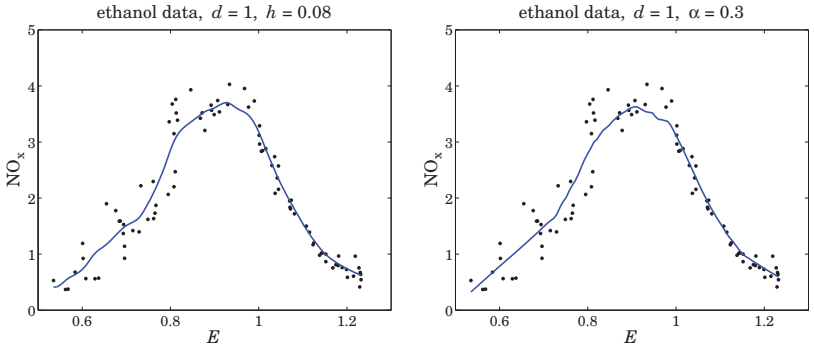


Fig. 5.3.5 Fits with fixed (left) and nearest-neighbor (right) bandwidths.

shows the fits when the fixed bandwidth is increased to $h = 0.08$ and the nearest-neighbor one to $\alpha = 0.3$. The resulting fits are much smoother.

The MATLAB code for generating the graphs of Fig. 5.3.4 is as follows:

```
Y = loadfile('ethanol.dat');                % file available in OSP toolbox
tobs = Y(:,1);  yobs = Y(:,2);              % data

t = locgrid(tobs,101);                      % uniform grid of 101 fitting points

h = linspace(0.02, 0.08, 41);               % vary h over 0.02 ≤ h ≤ 0.08
gcv1 = locgcv(tobs,yobs,1,1,h,'f');         % GCV as function of h
gcv2 = locgcv(tobs,yobs,2,1,h,'f');

figure; plot(h,gcv1,'-', h,gcv2,'--');      % left graph

h = locband(tobs, t, 0, h1);                % fixed bandwidths at t
x1 = locpol(tobs,yobs,t,h,1,1);             % fit with d = 1 and tricube window

h = locband(tobs, t, 0, h2);
x2 = locpol(tobs,yobs,t,h,2,1);             % fit with d = 2 and tricube window

figure; plot(tobs,yobs,'.', t,x1,'-', t,x2,'--');      % right graph
```


The MATLAB code for generating Fig. 5.3.5 is as follows:

```
h0 = 0.08; h = locband(tobs, t, 0, h0);     % fixed bandwidth case
x1 = locpol(tobs,yobs,t,h,1,1);             % order d = 1, tricube window
figure; plot(tobs,yobs,'.', t,x1,'-');      % left graph

alpha = 0.3; h = locband(tobs, t, alpha, 0);   % nearest-neighbor bandwidth case
x1 = locpol(tobs,yobs,t,h,1,1); x1 = C(:,1);   % order d = 1, tricube windowm
figure; plot(tobs,yobs,'.', t,x1,'-');         % right graph
```

Fig. 5.3.6 shows a fit at 10 fitting points and interpolated over 101 points. The fitting parameters are as in the right graph of Fig. 5.3.5. The following code generates Fig. 5.3.6:

```
tf = locgrid(tobs,10);                      % fitting points
alpha = 0.3; h = locband(tobs, tf, alpha, 0);   % nearest-neighbor bandwidths
[xf,C] = locpol(tobs,yobs,tf,h,1,1);        % order 1, tricube window

ti = locgrid(tf,101); yi = locval(C,tf,ti);     % interpolated points

figure; plot(tobs,yobs,'.', ti,yi,'-', tf,xf,'o');
```

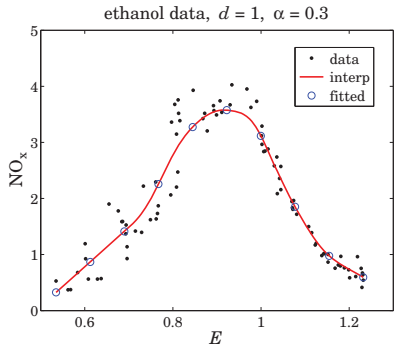


Fig. 5.3.6 Interpolated fits.

5.4 Variable Bandwidth

The issue of selecting the right bandwidth has been studied extensively, with approaches ranging from finding an optimum bandwidth that minimizes a selection criterion such as the GCV to using a locally-adaptive criterion that allows the bandwidth to automatically adapt to the local nature of the signal with different bandwidths being used in different parts of the signal [188–231].

There is no selection criterion that is universally successful or universally agreed upon and one must use one's judgment and visual inspection to decide how much smoothing is satisfactory. The basic idea is always to reduce the bandwidth in regions where the curvature of the signal is high in order not to oversmooth.

The function `locpol` can accept a different bandwidth h_t for each fitting time t. As we saw in the above examples, the function `locband` generates such bandwidths for input to `locpol`. However, `locband` generates either fixed or or nearest-neighbor bandwidths and is not adaptive to the local nature of the signal. One could manually, divide the range of the signal in non-overlapping regions and use a different fixed bandwidth in each region. In some cases, as in the Doppler example below, this is possible but in other cases a more automatic way of adapting is desirable.

A naive, but as we see in the examples below, quite effective way is to estimate the curvature, say κ_t, of the signal and define the bandwidth in terms of a suitable decreasing function $h_t = f(\kappa_t)$. We may define the curvature in terms of the estimate of the second derivative of the signal and normalize it to its maximum value:

$$\kappa_t = \frac{|\hat{\ddot{x}}_t|}{\max_t |\hat{\ddot{x}}_t|} \tag{5.4.1}$$

The second derivative $\hat{\ddot{x}}_t$ can be estimated by performing a local polynomial fit with polynomial order $d \geq 2$ using a fixed bandwidth h_0 or a nearest-neighbor bandwidth α. If one could determine a bandwidth range $[h_{\min}, h_{\max}]$ such that $h_{\max}$ would provide an appropriate amount of smoothing in certain parts of the signal and $h_{\min}$ would be appropriate in regions where the signal appears to have larger curvature, then one may choose $h_{\min} \leq h_0 \leq h_{\max}$, with $h_0 = h_{\max}$ as an initial trial value. An ad hoc but very simple choice for the bandwidth function $f(\kappa_t)$ then could be

$$h_t = h_{\max} \left(\frac{h_{\min}}{h_{\max}} \right)^{\kappa_t} \tag{5.4.2}$$

Other simple choices are possible, for example,

$$h_t = \frac{h_{\max} h_{\min}}{h_{\min} + (h_{\max} - h_{\min})\,\kappa_t^p}$$

for some power p. Since κ_t varies in $0 \leq \kappa_t \leq 1$, these choices interpolate between $h_{\max}$ at $\kappa_t = 0$ when the curvature is small, and $h_{\min}$ at $\kappa_t = 1$ when the curvature is large.

We illustrate the use of (5.4.2) with the three examples in Figs. 5.4.1–5.4.3, and we make a different bandwidth choice for Fig. 5.4.4. All four examples have been used as benchmarks in studying wavelet denoising methods [821] and we will be discussing them again in that context in Sec. 10.7.

In all cases, we use a second-order polynomial to determine the curvature, and then perform a locally linear fit ($d = 1$) using the variable bandwidth. Fig. 5.4.1 illustrates the test function "bumps" defined by

$$s(t) = \sum_{i=1}^{11} \frac{a_i}{[1 + |t - t_i|/w_i]^4}, \quad 0 \leq t \leq 1$$

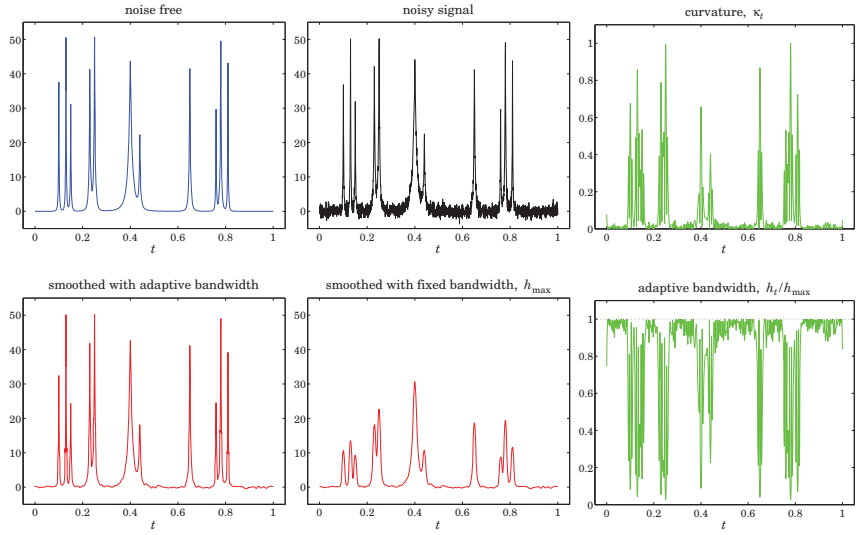


Fig. 5.4.1 Bumps function.

with the parameter values:

$$t_i = [10, 13, 15, 23, 25, 40, 44, 65, 76, 78, 81]/100$$

$$a_i = [40, 50, 30, 40, 50, 42, 21, 43, 31, 51, 42] \cdot 1.0523$$

$$w_i = [5, 5, 6, 10, 10, 30, 10, 10, 5, 8, 5]/1000$$

The function $s(t)$ is sampled at $N = 2048$ equally-spaced points t_n in the interval $[0, 1)$ and zero-mean white gaussian noise of variance $\sigma^2 = 1$ is added so that the noisy signal is $y_n = s_n + v_n$, where $s_n = s(t_n)$. The factor 1.0523 in the amplitudes a_i ensures that the signal-to-noise ratio has the standard benchmark value $\sigma_s/\sigma_v = 7$, where σ_s is the standard deviation of s_n, that is, $\sigma_s = \text{std}(s)$. The bandwidth range is defined by $h_{\max} = 0.01$ and $h_{\min} = 0.00025$. The value for $h_{\max}$ was chosen so that the flat portions of the signal between peaks are adequately smoothed.

The curvature κ_t, estimated using the bandwidth $h_0 = h_{\max}$, is shown in the upper right graph. The corresponding variable bandwidth h_t derived from Eq. (5.4.2) is shown in the bottom-right graph. The bottom-left graph shows the resulting local linear fit using the variable bandwidth h_t, while the bottom-middle graph shows the fit using the fixed bandwidth $h_{\max}$. Although $h_{\max}$ is adequate for smoothing the valleys of the signal, it is too large for the peaks and results in broadened peaks of reduced heights. On the other hand, the variable bandwidth preserves the peaks fairly well, while achieving comparable smoothing of the valleys. The MATLAB code for this example was as follows:

```
N=2048; t=linspace(0,1,N); s=zeros(size(t));
F = inline('1./(1 + abs(t)).^4');            % bumps function
```

```
ti = [10 13 15 23 25 40 44 65 76 78 81]/100;
ai = [40,50,30,40,50,42,21,43,31,51,42] * 1.0523;
wi = [5,5,6,10,10,30,10,10,5,8,5]/1000;

for i=1:length(ai),                          % construct signal
   s = s + ai(i)*F((t-ti(i))/wi(i));
end

hmax=10e-3; hmin=2.5e-4; h0=hmax;            % bandwidth limits

seed=2009; randn('state',seed);              % noisy signal
y = s + randn(size(t));

d=2; type=1;                                 % fit with d = 2 and tricube window
[x,C] = locpol(t,y,t,h0,d,type);             % using fixed bandwidth h0
kt = abs(C(:,3)); kt = kt/max(kt);           % curvature, kt
ht = hmax * (hmin/hmax).^kt;                 % bandwidth, ht

d=1; type=1;                                 % fit with d = 1
xv = locpol(t,y,t,ht,d,type);                % use variable bandwidth ht
xf = locpol(t,y,t,h0,d,type);                % use fixed bandwidth h0

figure; plot(t,s);  figure; plot(t,y);  figure; plot(t,kt);          % upper graphs
figure; plot(t,xv); figure; plot(t,xf); figure; plot(t,ht/hmax);     % lower graphs
```

Fig. 5.4.2 shows the "blocks" function defined by

$$s(t) = \sum_{i=1}^{11} a_i F(t - t_i)\,, \quad F(t) = \frac{1}{2}(1 + \operatorname{sign} t)\,, \quad 0 \le t \le 1$$

with the same delays t_i as above and amplitudes:

$$a_i = [40, -50, 30, -40, 50, -42, 21, 43, -31, 21, -42] \cdot 0.3655$$

The noisy signal is $y_n = s_n + v_n$ with zero-mean unit-variance white noise. The amplitude factor 0.3655 in a_i is adjusted to give the same SNR as above, $\operatorname{std}(s)/\sigma = 7$. The MATLAB code generating the six graphs is identical to the above, except for the part that defines the signal and the bandwidth limits $h_{\max} = 0.03$ and $h_{\min} = 0.0015$:

```
N=2048; t=linspace(0,1,N); s=zeros(size(t));

ti = [10 13 15 23 25 40 44 65 76 78 81]/100;
ai = [40,-50,30,-40,50,-42,21,43,-31,21,-42] * 0.3655;

for i=1:length(ai),
    s = s + ai(i) * (1 + sign(t - ti(i)))/2;          % blocks signal
end

hmax=0.03; hmin=0.0015; h0=hmax;                      % bandwidth limits
```

We observe that the flat parts of the signal are smoothed equally well by the variable and fixed bandwidth choices, but in the fixed case, the edges are smoothed too much. The "HeaviSine" signal shown in Fig. 5.4.3 is defined by

$$s(t) = \left[4\sin(4\pi t) - \operatorname{sign}(t - 0.3) - \operatorname{sign}(0.72 - t)\right] \cdot 2.357\,, \quad 0 \le t \le 1$$

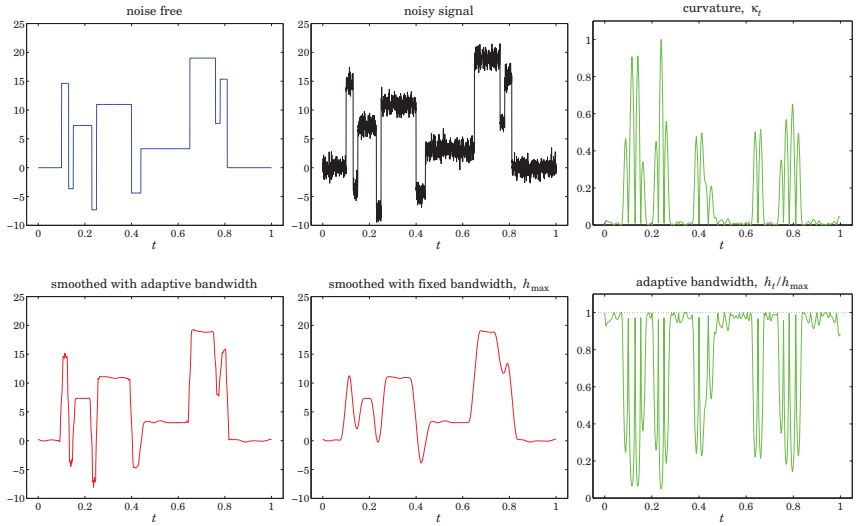


Fig. 5.4.2 Blocks function.

where the factor 2.357 is adjusted to give $\operatorname{std}(s) = 7$. The graphs shown in Fig. 5.4.3 are again generated by the identical MATLAB code, except for the parts defining the signal and bandwidths:

```
s = (4*sin(4*pi*t)-sign(t-0.3)-sign(0.72-t))*2.357;      % HeaviSine signal

hmax=0.035; hmin=0.0035; h0=hmax;                        % bandwidth limits
```

We note that the curvature κ_t is significantly large—and the bandwidth h_t is significantly small—only near the discontinuity points. The fixed bandwidth case smoothes the discontinuities too much, whereas the variable bandwidth tends to preserve them while reducing the noise equally well in the rest of the signal.

In the "doppler" example shown in Fig. 5.4.4, noticing that the curvature κ_t is significantly large only in the range $0 \le t \le 0.2$, we have followed a simpler strategy to define a variable bandwidth (although the choice (5.4.2) still works well). We took a fixed but small bandwidth over the range $0 \le t \le 0.2$ and transitioned gradually to a larger bandwidth for $0.2 \le t \le 1$. The signal is defined by

$$s(t) = 24\sqrt{t(1-t)}\,\sin\left(\frac{2.1\pi}{t + 0.05}\right)\,, \quad 0 \le t \le 1$$

The auxiliary unit-step function `ustep` was used to define the two-step bandwidth with a given rise time. The MATLAB code generating the six graphs was as follows:

```
N = 2048; t = linspace(0,1,N);

s = 24*sqrt(t.*(1-t)) .* sin(2.1*pi./(t+0.05));     % doppler signal
```

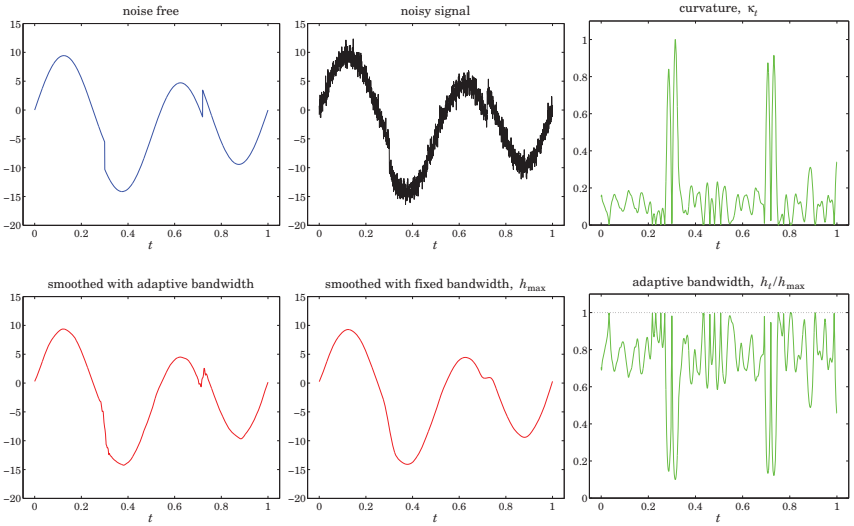


Fig. 5.4.3 HeaviSine function.

```
seed=2009; randn('state',seed);                    % noisy signal
y = s + randn(size(t));
```

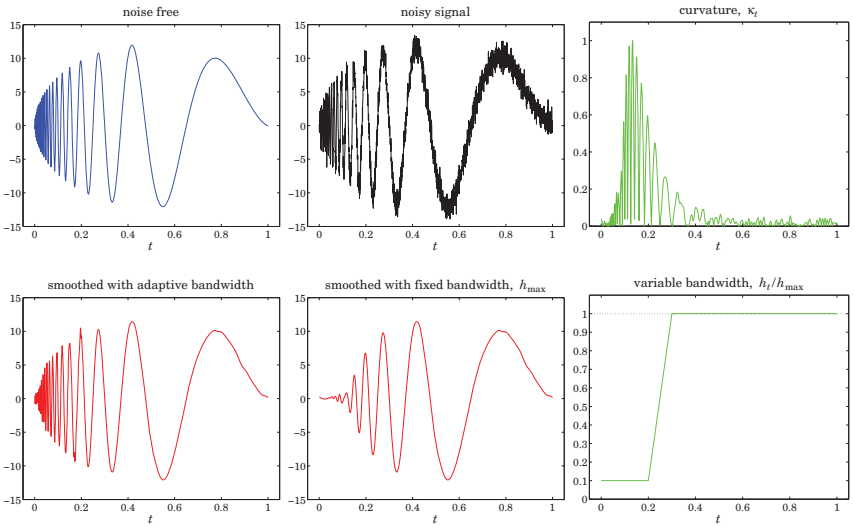


Fig. 5.4.4 Doppler function.


```
hmax=0.02; hmin=0.002; h0=hmax;                    % bandwidth limits

d=2; type=1;                                       % fit with d = 2 and tricube window
[x,C] = locpol(t,y,t,h0,d,type);                   % using fixed bandwidth h0
kt = abs(C(:,3)); kt = kt/max(kt);                 % curvature, kappa_t
ht = hmin + (hmax-hmin) * ustep(t-0.2, 0.1);       % two-step bandwidth, h_t
                                                   % ustep is in the OSP toolbox
d=1; type=1;
xv = locpol(t,y,t,ht,d,type);                      % fixed bandwidth h0
xf = locpol(t,y,t,h0,d,type);                      % fixed bandwidth h0

figure; plot(t,s);  figure; plot(t,y);  figure; plot(t,kt);          % upper graphs
figure; plot(t,xv); figure; plot(t,xf); figure; plot(t,ht/hmax);     % lower graphs
```

The local polynomial fitting results from these four benchmark examples are very comparable with the wavelet denoising approach discussed in Sec. 10.7.

5.5 Repeated Observations

Until now we had implicitly assumed that the observations were unique, that is, one observation $y(t_k)$ at each time t_k. However, in experimental data one often has repeated observations at a given t_k, all of which are listed as part of the data set. This is in fact true of both the motorcycle and the ethanol data sets. For example, in the motorcycle data, we have six repeated observations at $t = 14.6$,

k	t_k	y_k
⋮	⋮	⋮
22	14.6	−13.3
23	14.6	−5.4
24	14.6	−5.4
25	14.6	−9.3
26	14.6	−16.0
27	14.6	−22.8
⋮	⋮	⋮

and there other similar instances within the data set. In fact, among the 133 given observations, only 94 correspond to unique observation times.

To handle repeated observations one possibility is to simply keep one and ignore the rest—but which one? A better possibility is to allow all of them to be part of the least-squares performance index. It is easy to see that this is equivalent to replacing each group of repeated observations by their average and modifying the weighting function by the corresponding multiplicity of the group.

Let n_k denote the multiplicity of the observations at time t_k, that is, let $y_i(t_k)$, $i = 1, 2, \ldots, n_k$ be the observation values that are given at the unique observation time t_k. Then, the performance index (5.1.3) must be modified to include all of the $y_i(t_k)$:

$$\mathcal{J}_t = \sum_{|t_k - t| \le h_t} \sum_{i=1}^{n_k} [y_i(t_k) - \mathbf{u}^T(t_k - t)\mathbf{c}]^2 w(t_k - t) = \min \tag{5.5.1}$$

Setting the gradient with respect to $\mathbf{c}$ to zero, gives the normal equations:

$$\sum_{|t_k-t|\le h_t}\sum_{i=1}^{n_k} w(t_k-t)\mathbf{u}(t_k-t)\mathbf{u}^T(t_k-t)\,\mathbf{c} = \sum_{|t_k-t|\le h_t} w(t_k-t)\mathbf{u}(t_k-t)\sum_{i=1}^{n_k} y_i(t_k)$$

Defining the average of the n_k observations,

$$\bar{y}(t_k) = \frac{1}{n_k}\sum_{i=1}^{n_k} y_i(t_k)$$

and noting that the left-hand side has no dependence on i, we obtain:

$$\sum_{|t_k-t|\le h_t} n_k w(t_k-t)\mathbf{u}(t_k-t)\mathbf{u}^T(t_k-t)\mathbf{c} = \sum_{|t_k-t|\le h_t} n_k w(t_k-t)\mathbf{u}(t_k-t)\bar{y}(t_k) \quad (5.5.2)$$

This is recognized to be the solution of an equivalent least-squares local polynomial fitting problem in which each unique t_k is weighted by $n_k w_k(t_k-t)$ with the kth observation replaced by the average $\bar{y}(t_k)$, that is,

$$\bar{J}_t = \sum_{|t_k-t|\le h_t} \big[\bar{y}(t_k) - \mathbf{u}^T(t_k-t)\mathbf{c}\big]^2 n_k w(t_k-t) = \min \quad (5.5.3)$$

Internally, the function `locpol` calls the function `avobs`, which takes in the raw data `tobs,yobs` and determines the unique observation times `ta`, averaged observations `ya`, and their multiplicities `na`:

```
[ta,ya,na] = avobs(tobs,yobs);
```
% average repeated observations

For example, if

```
tobs = [ 1   1   1   3   3   5   5   3   4   7   9   9   9   9];
yobs = [20  22  21  11  12  13  15  19  21  25  28  29  31  32];
```

the function first sorts the ts in increasing order,

```
tobs = [ 1   1   1   3   3   3   4   5   5   7   9   9   9   9];
yobs = [20  21  22  11  12  19  21  13  15  25  28  29  31  32];
```

and then returns the output,

```
ta = [ 1   3   4   5   7   9];
ya = [21  14  21  14  25  30];
na = [ 3   3   1   2   1   4];
```

5.6 Loess Smoothing

Loess, which is a shorthand for *lo*cal reg*ress*ion, is a method proposed by Cleveland [192] for handling data with outliers. A version of it was discussed in Sec. 4.5. The method carries out a local polynomial regression using a nearest-neighbor bandwidth and the tricube window function, and then uses the resulting error residuals to iteratively readjust the window weights giving less importance to the outliers.

The method is described as follows [192]. Given the N-dimensional vectors of observation times and observations $t_{\text{obs}}, y_{\text{obs}}$, the nearest-neighbor bandwidth parameter α, and the polynomial order d, the method begins by performing a preliminary fit to all the observation times. For example, in the notation of the `locband` and `locpol` functions:

$$\begin{aligned} h &= \texttt{locband}(t_{\text{obs}}, t_{\text{obs}}, \alpha, 0); && \text{(find local bandwidths at } t_{\text{obs}}\text{)} \\ \hat{x} &= \texttt{locpol}(t_{\text{obs}}, y_{\text{obs}}, t_{\text{obs}}, h, d, 1); && \text{(perform fit at all } t_{\text{obs}}\text{)} \end{aligned} \quad (5.6.1)$$

where the last argument of `locpol` designates the use of the tricube window. From the resulting N-dimensional signal $\hat{x}_k$, $k = 0, 1, \ldots, N-1$, we calculate the corresponding error residuals e_k and use their median to calculate "robustness" weights r_k:

$$\begin{aligned} e_k &= y_k - \hat{x}_k\,, \quad k = 0, 1, \ldots, N-1 \\ \mu &= \underset{0\le k\le N-1}{\text{median}}\,(|e_k|) \\ r_k &= W\left(\frac{e_k}{6\mu}\right) \end{aligned} \quad (5.6.2)$$

where $W(u)$ is the bisquare function defined in (5.1.5). The local polynomial fitting is now repeated at all observation points t_{obs}, but instead of using the weights $w(t_k - t_{\text{obs}})$ for the kth observation's contribution to the fit, one uses the modified weights $r_k w(t_k - t_{\text{obs}})$. The new residuals are then computed as in (5.6.2) and the process is repeated a few more times or until convergence (i.e., until the estimated signal $\hat{x}_k$ no longer changes).

After the final iteration resulting in the final values of the r_ks, one can carry out the fit at any other time point t, but again using weights $r_k w(t_k - t)$ for the contribution of the kth observation, that is, the weight matrix W_t in Eq. (5.1.7) is replaced by

$$W_t = \text{diag}\big([\cdots, r_k w(t_k - t), \cdots]\big)$$

The MATLAB function `loess` implements these steps:

```
[xhat,C] = loess(tobs,yobs,t,alpha,d,Nit);
```
% Loess smoothing

where t are the final fitting times and `xhat` and `C` have the same meaning as in `locpol`. This function is similar in spirit to the robust local polynomial filtering function `rlpfilt` that was discussed in Sec. 4.5.

Example 5.6.1: Fig. 5.6.1 shows the same example as that of Fig. 4.5.3, with nearest-neighbor bandwidth parameter $\alpha = 0.4$ and an order-2 polynomial. The graphs show the results of $N_{\text{it}} = 0, 2, 4, 6$ iterations—the first one corresponding to ordinary fitting with no robustness iterations. The MATLAB code for the top two graphs was:

```
t = (0:50); x0 = (1 - cos(2*pi*t/50))/2;      % desired signal
seed=2005; randn('state',seed);
y = x0 + 0.1 * randn(size(x0));               % noisy signal

m = [-1 0 1 3];                               % outlier indices
n0=25; y(n0+m+1) = 0;                         % outlier values
n1=10; y(n1+m+1) = 1;
```

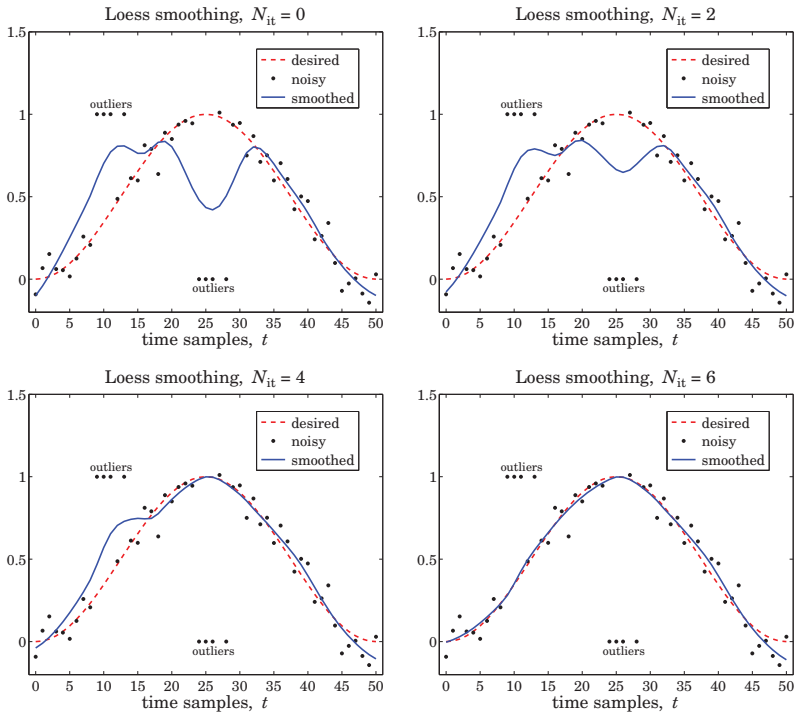


Fig. 5.6.1 Loess smoothing with $d = 2$, $\alpha = 0.4$, and different iterations.

```
alpha=0.4; d=2;                          % bandwidth parameter and polynomial order

Nit=0; x = loess(t,y,t,alpha,d,Nit);     % loess fit at t
figure; plot(t,x0,'--', t,y,'.', t,x,'-');   % left graph

Nit=2; x = loess(t,y,t,alpha,d,Nit);     % loess fit at t
figure; plot(t,x0,'--', t,y,'.', t,x,'-');   % right graph
```

The loess fit was performed at all t. We observe how successive iterations gradually diminish the distorting influence of the outliers. □

5.7 Problems

5.1 Prove the matrix inversion lemma identity (5.2.8). Using this identity, show that

$$H_{ii} = \frac{H_{ii}^-}{1 + H_{ii}^-}, \quad \text{where } H_{ii}^- = w_0 \mathbf{u}_0^T F_i^- \mathbf{u}_0, \quad F_i^- = (S_i^T W_i S_i)^-$$

then, argue that $0 \le H_{ii} \le 1$.

6 Exponential Smoothing

6.1 Mean Tracking

l

The exponential smoother, also known as an exponentially-weighted moving average (EWMA) or more simply an exponential moving average (EMA) filter is a simple, effective, recursive smoothing filter that can be applied to real-time data. By contrast, the local polynomial modeling approach is typically applied off-line to a block a signal samples that have already been collected.

The exponential smoother is used routinely to track stock market data and in forecasting applications such as inventory control where, despite its simplicity, it is highly competitive with other more sophisticated forecasting methods [232–279].

We have already encountered it in Sec. 2.3 and compared it to the plain FIR averager. Here, we view it as a special case of a weighted local polynomial smoothing problem using a causal window and exponential weights, and discuss some of its generalizations. Both the exponential smoother and the FIR averager are applied to data that are assumed to have the typical form:

$$y_n = a_n + v_n \tag{6.1.1}$$

where a_n is a low-frequency trend component, representing an average or estimate of the *local level* of the signal, and v_n a random, zero-mean, broadband component, such as white noise. If a_n is a deterministic signal, then by taking expectations of both sides we see that a_n represents the mean value of y_n, that is, $a_n = E[y_n]$. If y_n is stationary, then a_n is a constant, independent of n.

The output of either the FIR or the exponential smoothing filter tracks the signal a_n. To see how such filters arise in the context of estimating the mean level of a signal, consider first the stationary case. The mean $m = E[y_n]$ minimizes the following variance performance index (e.g., see Example 1.3.5):

$$\mathcal{J} = E[(y_n - a)^2] = \min \quad \Rightarrow \quad a_{\text{opt}} = m = E[y_n] \tag{6.1.2}$$

with minimized value $\mathcal{J}_{\min} = \sigma_y^2$. This result is obtained by setting the gradient with respect to a to zero:

$$\frac{\partial \mathcal{J}}{\partial a} = -2E[y_n - a] = 0 \tag{6.1.3}$$