

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

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

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

Exponential Smoothing

6.1 Mean Tracking

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 .

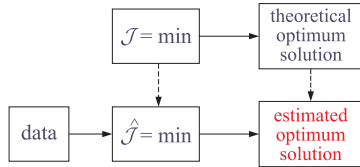
The output of either the FIR or the exponential smoothing filter tracks the signal a_n . To see how such filters arise in the context of estimating the mean level of a signal, consider first the stationary case. The mean $m = E[y_n]$ minimizes the following variance performance index (e.g., see Example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

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

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

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

In general, given a theoretical performance index \mathcal{J} , one must replace it in practice by an experimental one, say $\hat{\mathcal{J}}$, expressible in terms of the actual available data. The minimization of $\hat{\mathcal{J}}$ provides then estimates of the parameters or signals to be estimated.



Depending on the index $\hat{\mathcal{J}}$, the estimates may be calculated in a block processing manner using an entire block of data, or, on a sample-by-sample basis with the estimate being updated in real time in response to each new data sample. All adaptive filtering algorithms follow the latter approach.

We illustrate these ideas with the help of the simple performance index (6.1.2). We will apply this approach extensively in Chap. 16. Four possible practical definitions for $\hat{\mathcal{J}}$ that imitate (6.1.2) are:

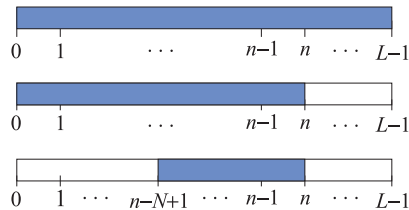
$$\hat{\mathcal{J}} = \sum_{n=0}^{L-1} (y_n - \hat{a})^2 = \min \tag{6.1.4a}$$

$$\hat{\mathcal{J}} = \sum_{k=0}^n (y_k - \hat{a})^2 = \min \tag{6.1.4b}$$

$$\hat{\mathcal{J}} = \sum_{k=n-N+1}^n (y_k - \hat{a})^2 = \min \tag{6.1.4c}$$

$$\hat{\mathcal{J}} = \sum_{k=0}^n \lambda^{n-k} (y_k - \hat{a})^2 = \min \tag{6.1.4d}$$

The first assumes a length- L block of data $[y_0, y_1, \dots, y_{L-1}]$. The last three are suitable for real-time implementations, where n denotes the current time. The second and fourth use the first $n+1$ data $[y_0, y_1, \dots, y_n]$, while the third uses a length- N sliding window $[y_{n-N+1}, \dots, y_{n-1}, y_n]$. The third choice leads to the FIR averager, also known as the *simple moving average* (SMA), and the fourth, to the exponential smoother, or, *exponential moving average* (EMA), where we require that the exponential “forgetting factor” λ be in the range $0 < \lambda < 1$. These time ranges are depicted below.



In order for the $\hat{\mathcal{J}}$ s to be unbiased estimates of \mathcal{J} , the above expressions should have been divided by the sum of their respective weights, namely, the quantities L ,

$(n+1)$, N , and $(1 + \lambda + \dots + \lambda^n)$, respectively. However, such factors do not affect the minimization solutions, which are easily found to be:

$$\hat{a} = \frac{y_0 + y_1 + \dots + y_{L-1}}{L} \tag{6.1.5a}$$

$$\hat{a}_n = \frac{y_0 + y_1 + \dots + y_n}{n+1} \tag{6.1.5b}$$

$$\hat{a}_n = \frac{y_n + y_{n-1} + \dots + y_{n-N+1}}{N} \tag{6.1.5c}$$

$$\hat{a}_n = \frac{y_n + \lambda y_{n-1} + \lambda^2 y_{n-2} + \dots + \lambda^n y_0}{1 + \lambda + \lambda^2 + \dots + \lambda^n} \tag{6.1.5d}$$

We have tacked on a subscript n to the last three to emphasize their dependence of their performance index on the current time instant n . Eqs. (6.1.4c) and (6.1.5c) tentatively assume that $n \geq N - 1$; for $0 \leq n < N - 1$, one should use the running average (6.1.4b) and (6.1.5b). Initialization issues are discussed further in Sections 6.6 and 6.19.

All four estimates are *unbiased* estimators of the true mean m . Their quality as estimators can be judged by their variances, which are (assuming that $y_n - m$ are mutually independent):

$$\sigma_{\hat{a}}^2 = E[(\hat{a} - m)^2] = \frac{\sigma_y^2}{L} \tag{6.1.6a}$$

$$\sigma_{\hat{a}_n}^2 = E[(\hat{a}_n - m)^2] = \frac{\sigma_y^2}{n+1} \tag{6.1.6b}$$

$$\sigma_{\hat{a}_n}^2 = E[(\hat{a}_n - m)^2] = \frac{\sigma_y^2}{N} \tag{6.1.6c}$$

$$\sigma_{\hat{a}_n}^2 = E[(\hat{a}_n - m)^2] = \sigma_y^2 \frac{1 - \lambda}{1 + \lambda} \cdot \frac{1 + \lambda^{n+1}}{1 - \lambda^{n+1}} \tag{6.1.6d}$$

The first two, corresponding to ordinary sample averaging, are asymptotically consistent estimators having variances that tend to zero as $L \rightarrow \infty$ or $n \rightarrow \infty$. The last two are not consistent. However, their variances can be made as small as desired by proper choice of the parameters N or λ .

The exponential smoothing filter may also be derived from a different point of view. The estimates (6.1.5) are the *exact* least-squares solutions of the indices (6.1.4). An alternative to using the exact solutions is to derive an LMS (least-mean-square) type of adaptive algorithm which minimizes the performance index iteratively using a steepest-descent algorithm that replaces the theoretical gradient (6.1.3) by an “instantaneous” one in which the expectation instruction is ignored:

$$\frac{\partial \mathcal{J}}{\partial a} = -2E[y_n - a] \quad \rightarrow \quad \frac{\partial \hat{\mathcal{J}}}{\partial a} = -2[y_n - \hat{a}_{n-1}] \tag{6.1.7}$$

The LMS algorithm then updates the previous estimate by adding a correction in the direction of the negative gradient using a small positive adaptation parameter μ :

$$\Delta a = -\mu \frac{\partial \hat{\mathcal{J}}}{\partial a}, \quad \hat{a}_n = \hat{a}_{n-1} + \Delta a \tag{6.1.8}$$

The resulting difference equation is identical to that of the steady-state exponential smoother (see Eq. (6.1.11) below),

$$\hat{a}_n = \hat{a}_{n-1} + 2\mu(y_n - \hat{a}_{n-1})$$

In adaptive filtering applications, the use of the exponentially discounted type of performance index (6.1.4d) leads to the so-called recursive least-squares (RLS) adaptive filters, which are in general different from the LMS adaptive filters. They happened to coincide in this particular example because of the simplicity of the problem.

The sample mean estimators (6.1.5a) and (6.1.5b) are geared to stationary data, whereas (6.1.5c) and (6.1.5d) can track nonstationary changes in the statistics of y_n . If y_n is nonstationary, then its mean $a_n = E[y_n]$ would be varying with n and a good estimate should be able to track it well and efficiently. To see the problems that arise in using the sample mean estimators in the nonstationary case, let us cast Eqs. (6.1.5b) and (6.1.5d) in recursive form. Both can be written as follows:

$$\hat{a}_n = (1 - \alpha_n)\hat{a}_{n-1} + \alpha_n y_n = \hat{a}_{n-1} + \alpha_n(y_n - \hat{a}_{n-1}) \quad (6.1.9)$$

where the gain parameter α_n is given by

$$\alpha_n = \frac{1}{n+1}, \quad \alpha_n = \frac{1}{1 + \lambda + \dots + \lambda^n} = \frac{1 - \lambda}{1 - \lambda^{n+1}} \quad (6.1.10)$$

for (6.1.5b) and (6.1.5d), respectively. The last side of Eq. (6.1.9) is written in a so-called “predictor/corrector” Kalman filter form, where the first term \hat{a}_{n-1} is a tentative prediction of \hat{a}_n and the second term is the correction obtained by multiplying the “prediction error” ($y_n - \hat{a}_{n-1}$) by a positive gain factor α_n . This term always corrects in the right direction, that is, if \hat{a}_{n-1} overestimates/underestimates y_n then the error tends to be negative/positive reducing/increasing the value of \hat{a}_{n-1} .

There is a dramatic difference between the two estimators. For the sample mean, the gain $\alpha_n = 1/(n+1)$ tends to zero rapidly with increasing n . For stationary data, the estimate \hat{a}_n will converge quickly to the true mean. Once n is fairly large, the correction term becomes essentially unimportant because the gain is so small. If after converging to the true mean the statistics of y_n were to suddenly change with a new value of the mean, the sample-mean estimator \hat{a}_n would have a very hard time responding to such a change and converging to the new value because the new changes are communicated only through the already very small correction term.

On the other hand, for the exponential smoother case (6.1.5d), the gain tends to a constant for large n , that is, $\alpha_n \rightarrow \alpha = 1 - \lambda$. Therefore, the correction term remains finite and can communicate the changes in the statistics. The price one pays for that is that the estimator is not consistent. Asymptotically, the estimator (6.1.5d) becomes the ordinary exponential smoothing filter described by the difference equation,

$$\hat{a}_n = \lambda\hat{a}_{n-1} + \alpha y_n = \hat{a}_{n-1} + \alpha(y_n - \hat{a}_{n-1}) \quad (6.1.11)$$

Its transfer function and asymptotic variance are:

$$H(z) = \frac{\alpha}{1 - \lambda z^{-1}}, \quad \sigma_{\hat{a}_n}^2 = E[(\hat{a}_n - m)^2] = \sigma_y^2 \frac{1 - \lambda}{1 + \lambda} \quad (6.1.12)$$

The quantity $\sigma_{\hat{a}_n}^2 / \sigma_y^2$ is the NRR of this filter. The differences in the behavior of the sample-mean and exponential smoother can be understood by inspecting the corresponding performance indices, which may be written in an expanded form:

$$\begin{aligned} \hat{\mathcal{J}} &= (y_n - \hat{a})^2 + (y_{n-1} - \hat{a})^2 + (y_{n-2} - \hat{a})^2 + \dots + (y_0 - \hat{a})^2 \\ \hat{\mathcal{J}} &= (y_n - \hat{a})^2 + \lambda(y_{n-1} - \hat{a})^2 + \lambda^2(y_{n-2} - \hat{a})^2 + \dots + \lambda^n(y_0 - \hat{a})^2 \end{aligned} \quad (6.1.13)$$

The first index weighs all terms equally, as it should for stationary data. The second index emphasizes the terms arising from the most recent observation y_n and exponentially forgets, or discounts, the earlier observations and thus can respond more quickly to new changes. Even though the second index appears to have an ever increasing number of terms, in reality, the effective number of terms that are significant is finite and can be estimated by the formula:

$$\bar{n} = \frac{\sum_{n=0}^{\infty} n\lambda^n}{\sum_{n=0}^{\infty} \lambda^n} = \frac{\lambda}{1 - \lambda} \quad (6.1.14)$$

This expression is only a guideline and other possibilities exist. For example, one can define \bar{n} to be the effective time constant of the filter:

$$\lambda^{\bar{n}} = \epsilon \quad \Rightarrow \quad \bar{n} = \frac{\ln \epsilon}{\ln \lambda} \approx \frac{\ln(\epsilon^{-1})}{1 - \lambda}, \quad \text{for } \lambda \lesssim 1 \quad (6.1.15)$$

where ϵ is a small user-specified parameter such as $\epsilon = 0.01$. The sliding window estimator (6.1.5c) is recognized as a length- N FIR averaging filter of the type we considered in Sec. 2.4. It also can track a nonstationary signal at the expense of not being a consistent estimator. Requiring that it achieve the same variance as the exponential smoother gives the conditions:

$$\boxed{\frac{1}{N}\sigma_y^2 = \frac{1 - \lambda}{1 + \lambda}\sigma_y^2} \quad \Rightarrow \quad \lambda = \frac{N - 1}{N + 1} \quad \Rightarrow \quad \alpha = 1 - \lambda = \frac{2}{N + 1} \quad (6.1.16)$$

Such conditions are routinely used to set the parameters of FIR and exponential smoothing filters in inventory control applications and in tracking stock market data. A similar weighted average as in Eq. (6.1.14) can be defined for any filter by:

$$\boxed{\bar{n} = \frac{\sum_n n h_n}{\sum_n h_n}} \quad (\text{effective filter lag}) \quad (6.1.17)$$

where h_n is the filter's impulse response. Eq. (6.1.17) may also be expressed in terms of the filter's transfer function $H(z) = \sum_n h_n z^{-n}$ and its derivative $H'(z) = dH(z)/dz$ evaluated at DC, that is, at $z = 1$:

$$\bar{n} = - \left. \frac{H'(z)}{H(z)} \right|_{z=1} \quad (\text{effective filter lag}) \quad (6.1.18)$$

Alternatively, \tilde{n} is recognized as the filter's *group delay* at DC, that is, given the frequency response $H(\omega) = \sum_n h_n e^{-j\omega n} = |H(\omega)| e^{j \arg H(\omega)}$, we have (Problem 6.1):

$$\tilde{n} = - \left. \frac{d}{d\omega} \arg H(\omega) \right|_{\omega=0} \quad (\text{group delay at DC}) \quad (6.1.19)$$

The exponential smoother is a special case of (6.1.17) with $h_n = \alpha \lambda^n u(n)$, where $u(n)$ is the unit-step function. We may apply this definition also to the FIR averager filter that has $h_n = 1/N$, for $n = 0, 1, \dots, N - 1$,

$$\tilde{n} = \frac{1}{N} \sum_{n=0}^{N-1} n = \frac{N-1}{2}$$

The FIR averager can be mapped into an “equivalent” exponential smoother by equating the \tilde{n} lags of the two filters, that is,

$$\boxed{\tilde{n} = \frac{N-1}{2} = \frac{\lambda}{1-\lambda}} \quad (6.1.20)$$

This condition is exactly equivalent to condition (6.1.16) arising from matching the NRRs of the two filters. The two equations,

$$E[(\hat{a}_n - m)^2] = \frac{1-\lambda}{1+\lambda} \sigma_y^2 = \frac{1}{N} \sigma_y^2, \quad \tilde{n} = \frac{\lambda}{1-\lambda} = \frac{N-1}{2} \quad (6.1.21)$$

capture the main tradeoff between variance and speed in using an exponential smoother or an equivalent FIR averager, that is, the closer λ is to unity or the larger the N , the smaller the variance and the better the estimate, but the longer the transients and the slower the speed of response.

We summarize the difference equations for the exact exponential smoother (6.1.5d) and the steady-state one (6.1.11),

$$\begin{aligned} \hat{a}_n &= \frac{\lambda - \lambda^{n+1}}{1 - \lambda^{n+1}} \hat{a}_{n-1} + \frac{\alpha}{1 - \lambda^{n+1}} y_n = \hat{a}_{n-1} + \frac{\alpha}{1 - \lambda^{n+1}} (y_n - \hat{a}_{n-1}) \\ \hat{a}_n &= \lambda \hat{a}_{n-1} + \alpha y_n = \hat{a}_{n-1} + \alpha (y_n - \hat{a}_{n-1}) \end{aligned} \quad (6.1.22)$$

Clearly, the second is obtained in the large- n limit of the first, but in practice the steady one is often used from the start at $n = 0$ because of its simplicity.

To start the recursions at $n = 0$, one needs to specify the initial value \hat{a}_{-1} . For the exact smoother, \hat{a}_{-1} can have an arbitrary value because its coefficient vanishes at $n = 0$. This gives for the first smoothed value $\hat{a}_0 = 0 \cdot \hat{a}_{-1} + 1 \cdot y_0 = y_0$. For the steady smoother it would make sense to also require that $\hat{a}_0 = y_0$, which would imply that $\hat{a}_{-1} = y_0$ because then

$$\hat{a}_0 = \lambda \hat{a}_{-1} + \alpha y_0 = \lambda y_0 + \alpha y_0 = y_0$$

There are other reasonable ways of choosing \hat{a}_{-1} , for example one could take it to be the average of a few initial values of y_n . The convolutional solution of the steady smoother with arbitrary nonzero initial conditions is obtained by convolving the filter's

impulse response $\alpha \lambda^n u(n)$ with the causal input y_n plus adding a transient term arising from the initial value:

$$\hat{a}_n = \alpha \sum_{k=0}^n \lambda^{n-k} y_k + \lambda^{n+1} \hat{a}_{-1} \quad (6.1.23)$$

The influence of the initial value disappears exponentially.

Example 6.1.1: Fig. 6.1.1 illustrates the ability of the sample mean and the exponential smoother to track a sudden level change.

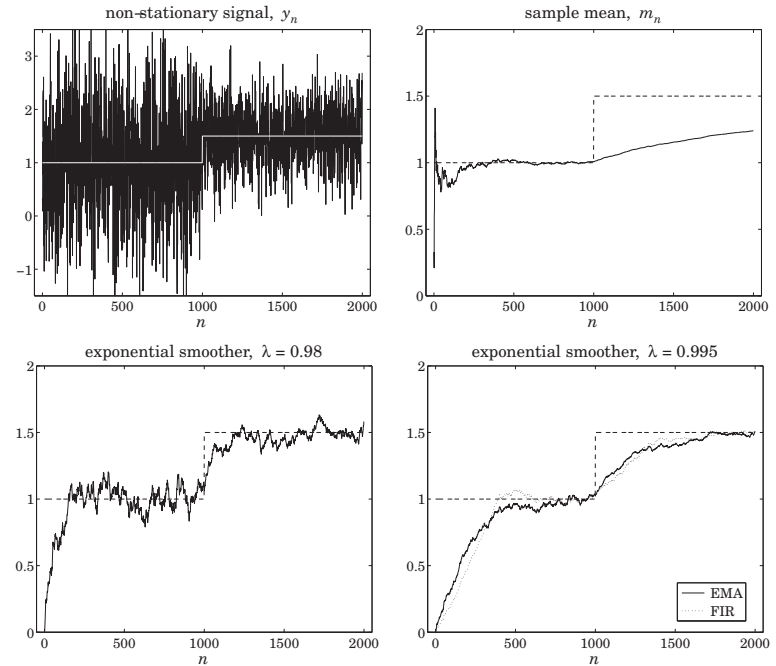


Fig. 6.1.1 Mean tracking with sample mean, exponential smoother, and FIR averager.

The first 1000 samples of the signal y_n depicted on the upper-left graph are independent gaussian samples of mean and variance $m_1 = 1, \sigma_1 = 1$. The last 1000 samples are gaussian samples with $m_2 = 1.5$ and $\sigma_2 = 0.5$.

The upper-right graph shows the sample mean computed recursively using (6.1.9) with $\alpha_n = 1/(n+1)$ and initialized at $\hat{a}_{-1} = 0$ (although the initial value does not matter since $\alpha_0 = 1$). We observe that the sample mean converges very fast to the first value of $m_1 = 1$, with its fluctuations becoming smaller and smaller because of its decreasing variance (6.1.6b). But it is extremely slow responding to the sudden change in the mean.

The bottom two graphs show the steady-state exponential smoother initialized at $\hat{a}_{-1} = 0$ with the two values of the forgetting factor $\lambda = 0.98$ and $\lambda = 0.995$. For the smaller λ the convergence is quick both at the beginning and after the change, but the fluctuations

quantified by (6.1.21) remain finite and do not improve even after convergence. For the larger λ , the fluctuations are smaller, but the learning time constant is longer. In the bottom-right graph, we have also added the equivalent FIR averager with N related to λ by (6.1.16), which gives $N = 399$. Its learning speed and fluctuations are comparable to those of the exponential smoother. \square

Example 6.1.2: Fig. 6.1.2 shows the daily Dow-Jones Industrial Average (DJIA) from Oct. 1, 2007 to Dec. 31, 2009. In the left graph an exponential smoothing filter is used with $\lambda = 0.9$. In the right graph, an FIR averager with an equivalent length of $N = (1 + \lambda)/(1 - \lambda) = 19$ is used. The data were obtained from <http://finance.yahoo.com>.

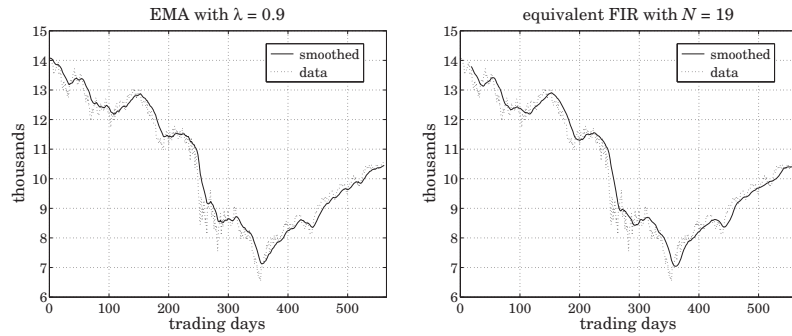


Fig. 6.1.2 Dow-Jones industrial average from 10-Oct-2007 to 31-Dec-2009.

The following code fragment generates the two graphs:

```

Y = loadfile('dow-oct07-dec09.dat');           % data file in OSP toolbox
y = Y(:,4)/1000;                               % extract closing prices
n = (0:length(y)-1);

la = 0.9; a1 = 1-la;
s0 = la*y(1);                                  % s0 is the initial state
m = filter(a1, [1,-la], y, s0);                % filter with initial state
% m = stema(y,0,la, y(1));                     % equivalent calculation

figure; plot(n,m,'-', n,y,':');

N = round((1+la)/(1-la));
h = ones(N,1)/N;                               % FIR averager
x = filter(h,1,y);

figure; plot(n(N:end),x(N:end),'-', n,y,':'); % discard first N-1 outputs

```

The initial value was set such that to get $\hat{a}_0 = y_0$ for the EMA. The built-in function `filter` allows one to specify the initial state. Because `filter` uses the transposed realization, in order to have $\hat{a}_0 = y_0$, the initial state must be chosen as $s_{in} = \lambda y_0$. This follows from the sample processing algorithm of the transposed realization for the EMA filter (6.1.12),

which reads as follows where s is the state:

$$\begin{array}{|l} \text{for each input sample } y \text{ do:} \\ \hat{a} = s + \alpha y \\ s = \lambda \hat{a} \end{array} \quad \text{or} \quad \begin{array}{|l} \hat{a}_n = s_n + \alpha y_n \\ s_{n+1} = \lambda \hat{a}_n \end{array}$$

Thus, in order for the first pass to give $\hat{a}_0 = y_0$, the initial state must be such that $s_0 = \hat{a}_0 - \alpha y_0 = \lambda y_0$. The FIR averager was run with zero initial conditions and therefore, the first $N - 1$ outputs were discarded as transients. After $n \geq N$, the EMA and the FIR outputs are comparable since they have the same \bar{n} . \square

Example 6.1.3: It is evident by inspecting the graphs of the previous example that both the EMA and the FIR filter outputs are lagging behind the data signal. To see this lag more clearly, consider a noiseless signal consisting of three straight-line segments defined by,

$$s_n = \begin{cases} 20 + 0.8n, & 0 \leq n < 75 \\ 80 - 0.3(n - 75), & 75 \leq n < 225 \\ 35 + 0.4(n - 225), & 225 \leq n \leq 300 \end{cases}$$

Fig. 6.1.3 shows the corresponding output from an EMA with $\lambda = 0.9$ and an equivalent FIR averager with $N = 19$ as in the previous example. The dashed line is the signal s_n and the solid lines, the corresponding filter outputs.

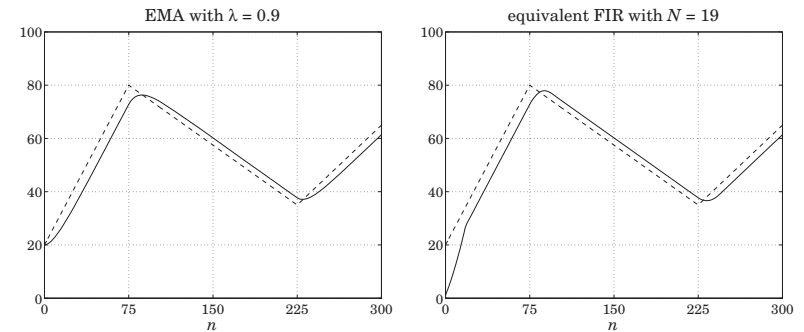


Fig. 6.1.3 Lag introduced by EMA and FIR averager filters.

The EMA was run with initial value $\hat{a}_{-1} = s_0 = 20$. The FIR filter was run with zero initial conditions, and therefore, its first $N - 1$ outputs are transients. The amount of delay introduced by the filters is exactly equal to the quantity \bar{n} of Eq. (6.1.20). \square

The delay \bar{n} is a consequence of the causality of the filters. Symmetric non-causal filters, such as the LPSM or LPRS filters, do not introduce a delay, that is, $\bar{n} = 0$.

To see how such a delay arises, consider an arbitrary causal filter h_n and a causal input that is a linear function of time, $x_n = a + bn$, for $n \geq 0$. The corresponding convolutional output will be:

$$y_n = \sum_{k=0}^n h_k x_{n-k} = \sum_{k=0}^n h_k [a + b(n-k)] = (a + bn) \sum_{k=0}^n h_k - b \sum_{k=0}^n kh_k$$

For large n , we may replace the upper limit of the summations by $k = \infty$,

$$y_n = (a + bn) \sum_{k=0}^{\infty} h_k - b \sum_{k=0}^{\infty} kh_k = (a + bn) \sum_{k=0}^{\infty} h_k - b\bar{n} \sum_{k=0}^{\infty} h_k = [a + b(n - \bar{n})] \sum_{k=0}^{\infty} h_k$$

where we used the definition (6.1.17) for \bar{n} . For filters that have unity gain at DC, the sum of the filter coefficients is unity, and we obtain,

$$y_n = a + b(n - \bar{n}) = x_{n-\bar{n}} \quad (6.1.24)$$

Such delays are of concern in a number of applications, such as the real-time monitoring of financial data. For FIR filters, the problem of designing noise reducing filters with a prescribed amount of delay \bar{n} has already been discussed in Sec. 3.8. However, we discuss it a bit further in Sec. 6.10 and 6.15 emphasizing its use in stock market trading. The delay \bar{n} can also be controlled by the use of higher-order exponential smoothing discussed in Sec. 6.5.

6.2 Forecasting and State-Space Models

We make a few remarks on the use of the first-order exponential smoother as a forecasting tool. As we already mentioned, the quantity \hat{a}_{n-1} can be viewed as a prediction of y_n based on the past observations $\{y_0, y_1, \dots, y_{n-1}\}$. To emphasize this interpretation, let us denote it by $\hat{y}_{n/n-1} = \hat{a}_{n-1}$, and the corresponding prediction or forecast error by $e_{n/n-1} = y_n - \hat{y}_{n/n-1}$. Then, the steady exponential smoother can be written as,

$$\hat{y}_{n+1/n} = \hat{y}_{n/n-1} + \alpha e_{n/n-1} = \hat{y}_{n/n-1} + \alpha(y_n - \hat{y}_{n/n-1}) \quad (6.2.1)$$

As discussed in Chapters 1 and 12, if the prediction is to be optimal, then the prediction error $e_{n/n-1}$ must be a white noise signal, called the innovations of the sequence y_n and denoted by $\varepsilon_n = e_{n/n-1}$. It represents that part of y_n that cannot be predicted from its past. This interpretation implies a certain innovations signal model for y_n . We may derive it by working with z-transforms. In the z-domain, Eq. (6.2.1) reads,

$$z\hat{Y}(z) = \hat{Y}(z) + \alpha E(z) = \hat{Y}(z) + \alpha(Y(z) - \hat{Y}(z)) = \lambda\hat{Y}(z) + \alpha Y(z) \quad (6.2.2)$$

Therefore, the transfer functions from $Y(z)$ to $\hat{Y}(z)$ and from $Y(z)$ to $E(z)$ are,

$$\hat{Y}(z) = \left(\frac{\alpha z^{-1}}{1 - \lambda z^{-1}} \right) Y(z), \quad E(z) = \left(\frac{1 - z^{-1}}{1 - \lambda z^{-1}} \right) Y(z) \quad (6.2.3)$$

In the time domain, using the notation $\nabla y_n = y_n - y_{n-1}$, we may write the latter as

$$\nabla y_n = \varepsilon_n - \lambda \varepsilon_{n-1} \quad (6.2.4)$$

Thus, y_n is an integrated ARMA process, ARIMA(0,1,1), or more simply an integrated MA process, IMA(1,1). In other words, if y_n is such a process, then the exponential smoother forecast $\hat{y}_{n/n-1}$ is optimal in the mean-square sense [242].

6.3 Higher-Order Polynomial Smoothing Filters

The innovations representation model can also be cast in an ordinary Wiener and Kalman filter form of the type discussed in Chap. 11. The state and measurement equations for such a model are:

$$\begin{aligned} x_{n+1} &= x_n + w_n \\ y_n &= x_n + v_n \end{aligned} \quad (6.2.5)$$

where w_n, v_n are zero-mean white-noise signals that are mutually uncorrelated. This model is referred to as a “constant level” state-space model, and represents a random-walk observed in noise. The optimal prediction estimate $\hat{x}_{n/n-1}$ of the state x_n is equivalent to \hat{a}_{n-1} . The equivalence between EMA and this model results in the following relationship between the parameters α and $q = \sigma_w^2 / \sigma_v^2$:

$$q = \frac{\alpha^2}{1 - \alpha} \Rightarrow \alpha = \frac{\sqrt{q^2 + 4q} - q}{2} \quad (6.2.6)$$

We defer discussion of such state-space models until chapters 11 and 13.

6.3 Higher-Order Polynomial Smoothing Filters

We recall that in fitting a local polynomial of order d to a local block of data $\{y_{n-M}, \dots, y_n, \dots, y_{n+M}\}$, the performance index was

$$\mathcal{J} = \sum_{k=-M}^M [y_{n+k} - p(k)]^2 = \sum_{k=-M}^M [y_{n+k} - \mathbf{u}_k^T \mathbf{c}]^2 = \min$$

where $p(k)$ is a d th degree polynomial, representing the estimate $\hat{y}_{n+k} = p(k)$,

$$p(k) = \mathbf{u}_k^T \mathbf{c} = [1, k, \dots, k^d] \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_d \end{bmatrix} = \sum_{i=0}^d c_i k^i$$

and we defined the monomial basis vector $\mathbf{u}_k = [1, k, k^2, \dots, k^d]^T$. The higher-order exponential smoother is obtained by restricting the data range to $\{y_0, y_1, \dots, y_n\}$ and using exponential weights, and similarly, the corresponding FIR version will be restricted to $\{y_{n-N+1}, \dots, y_{n-1}, y_n\}$. The resulting performance indices are then,

$$\begin{aligned} \mathcal{J}_n &= \sum_{k=-n}^0 \lambda^{-k} [y_{n+k} - \mathbf{u}_k^T \mathbf{c}]^2 = \min \quad \begin{array}{c} \text{---} \\ 0 \quad \dots \quad n \end{array} \\ \mathcal{J}_n &= \sum_{k=-N+1}^0 [y_{n+k} - \mathbf{u}_k^T \mathbf{c}]^2 = \min \quad \begin{array}{c} \text{---} \\ 0 \quad n-N+1 \quad \dots \quad n \end{array} \end{aligned}$$

or, replacing the summation index k by $-k$, the performance indices read,

$$\begin{aligned} \text{(EMA)} \quad \mathcal{J}_n &= \sum_{k=0}^n \lambda^k [y_{n-k} - \mathbf{u}_{-k}^T \mathbf{c}]^2 = \min \\ \text{(FIR)} \quad \mathcal{J}_n &= \sum_{k=0}^{N-1} [y_{n-k} - \mathbf{u}_{-k}^T \mathbf{c}]^2 = \min \end{aligned} \quad (6.3.1)$$

In both cases, we may interpret the quantities $p(\pm\tau) = \mathbf{u}_{\pm\tau}^T \mathbf{c}$ as the estimates $\hat{y}_{n\pm\tau}$. We will denote them by $\hat{y}_{n\pm\tau/n}$ to emphasize their causal dependence only on data up to the current time n . In particular, the quantity $c_0 = \mathbf{u}_0^T \mathbf{c} = p(0)$ represents the estimate \hat{y}_n , or $\hat{y}_{n/n}$, that is, an estimate of the local level of the signal. Similarly, $c_1 = \dot{p}(0) = \mathbf{u}_\tau^T \mathbf{c}|_{\tau=0}$ represents the local slope, and $2c_2 = \ddot{p}(0)$, the local acceleration. Eqs. (6.1.4d) and (6.1.4c) are special cases of (6.3.1) corresponding to $d = 0$.

Both indices in Eq. (6.3.1) can be written in the following compact vectorial form, whose solution we have already obtained in previous chapters:

$$\mathcal{J} = (\mathbf{y} - \mathbf{S}\mathbf{c})^T W (\mathbf{y} - \mathbf{S}\mathbf{c}) = \min \Rightarrow \mathbf{c} = (S^T W S)^{-1} S^T W \mathbf{y} \quad (6.3.2)$$

where the data vector \mathbf{y} is defined as follows in the EMA and FIR cases,

$$\mathbf{y}(n) = \begin{bmatrix} y_n \\ y_{n-1} \\ \vdots \\ y_0 \end{bmatrix}, \quad \mathbf{y}(n) = \begin{bmatrix} y_n \\ y_{n-1} \\ \vdots \\ y_{n-N+1} \end{bmatrix} \quad (6.3.3)$$

with the polynomial basis matrices S ,

$$S_n = [\mathbf{u}_0, \mathbf{u}_{-1}, \dots, \mathbf{u}_{-n}]^T, \quad S_N = [\mathbf{u}_0, \mathbf{u}_{-1}, \dots, \mathbf{u}_{-N+1}]^T = \begin{bmatrix} \mathbf{u}_0^T \\ \mathbf{u}_{-1}^T \\ \vdots \\ \mathbf{u}_{-k}^T \\ \vdots \\ \mathbf{u}_{-N+1}^T \end{bmatrix} \quad (6.3.4)$$

with, $\mathbf{u}_{-k}^T = [1, (-k), (-k)^2, \dots, (-k)^d]$, and weight matrices W in the two cases,

$$W_n = \text{diag}([1, \lambda, \dots, \lambda^n]), \quad \text{or}, \quad W = I_N \quad (6.3.5)$$

The predicted estimates can be written in the filtering form:

$$\hat{y}_{n+\tau/n} = \mathbf{u}_\tau^T \mathbf{c}(n) = \mathbf{h}_\tau^T(n) \mathbf{y}(n) \quad (6.3.6)$$

where in the exponential smoothing case,

$$\begin{cases} \mathbf{c}(n) = (S_n^T W_n S_n)^{-1} S_n^T W_n \mathbf{y}(n) \\ \mathbf{h}_\tau(n) = W_n S_n (S_n^T W_n S_n)^{-1} \mathbf{u}_\tau \end{cases} \quad (\text{EMA}) \quad (6.3.7)$$

We will see in Eq. (6.5.19) and more explicitly in (6.6.5) that $\mathbf{c}(n)$ can be expressed recursively in the time n . Similarly, for the FIR case, we find:

$$\begin{cases} \mathbf{c}(n) = (S_N^T S_N)^{-1} S_N^T \mathbf{y}(n) \\ \mathbf{h}_\tau = S_N (S_N^T S_N)^{-1} \mathbf{u}_\tau \end{cases} \quad (\text{FIR}) \quad (6.3.8)$$

We note also that the the solution for \mathbf{c} in Eq. (6.3.2) can be viewed as the least-squares solution of the over-determined linear system, $W^{1/2} \mathbf{S} \mathbf{c} = W^{1/2} \mathbf{y}$, which is particularly convenient for the numerical solution using MATLAB's backslash operation,

$$\mathbf{c} = (W^{1/2} \mathbf{S}) \setminus (W^{1/2} \mathbf{y}) \quad (6.3.9)$$

In fact, this corresponds to an alternative point of view to filtering and is followed in the so-called "linear regression" indicators in financial market trading, as we discuss in Sec. 6.18, where the issue of the initial transients, that is, the evaluation of $\mathbf{c}(n)$ for $0 \leq n \leq N-1$ in the FIR case, is also discussed.

In the EMA case, the basis matrices S_n are full rank for $n \geq d$. For $0 \leq n < d$, we may restrict the polynomial order d to to $d_n = n$ and thus obtain the first d_n coefficients of the vector $\mathbf{c}(n)$, and set the remaining coefficients to zero. For the commonly used case of $d = 1$, this procedure amounts to setting $\mathbf{c}(0) = [y_0, 0]^T$. Similarly, in the FIR case, we must have $N \geq d + 1$ to guarantee the full rank of S_N .

6.4 Linear Trend FIR Filters

The exact solutions of the FIR case have already been found in Sec. 3.8. The $d = 1$ and $d = 2$ closed-form solutions were given in Eqs. (3.8.10) and (3.8.11). The same expressions are valid for both even and odd N . For example, replacing $M = (N-1)/2$ in (3.8.10), we may express the solution for the $d = 1$ case as follows,

$$h_\tau(k) = \frac{2(N-1)(2N-1-3k) + 6(N-1-2k)\tau}{N(N^2-1)}, \quad k = 0, 1, \dots, N-1 \quad (6.4.1)$$

A direct derivation of (6.4.1) is as follows. From the definition (6.3.4), we find:

$$\begin{aligned} S_N^T S_N &= \sum_{k=0}^{N-1} \mathbf{u}_{-k} \mathbf{u}_{-k}^T = \sum_{k=0}^{N-1} \begin{bmatrix} 1 & -k \\ -k & k^2 \end{bmatrix} \\ &= \begin{bmatrix} N & -N(N-1)/2 \\ -N(N-1)/2 & N(N-1)(2N-1)/6 \end{bmatrix} \\ (S_N^T S_N)^{-1} &= \frac{2}{N(N^2-1)} \begin{bmatrix} (N-1)(2N-1) & 3(N-1) \\ 3(N-1) & 6 \end{bmatrix} \end{aligned} \quad (6.4.2)$$

then, from Eq. (6.3.8), because the k th row of S_N is \mathbf{u}_{-k}^T , we obtain the k th impulse response coefficient:

$$h_\tau(k) = \mathbf{u}_{-k}^T (S_N^T S_N)^{-1} \mathbf{u}_\tau = \frac{2}{N(N^2-1)} [1, -k] \begin{bmatrix} (N-1)(2N-1) & 3(N-1) \\ 3(N-1) & 6 \end{bmatrix} \begin{bmatrix} 1 \\ \tau \end{bmatrix}$$

which leads to Eq. (6.4.1). Thus, we obtain,

$$h_\tau(k) = h_a(k) + h_b(k) \tau, \quad k = 0, 1, \dots, N-1 \quad (6.4.3)$$

with

$$h_a(k) = \frac{2(2N-1-3k)}{N(N+1)}, \quad h_b(k) = \frac{6(N-1-2k)}{N(N^2-1)} \quad (6.4.4)$$

These are the FIR filters that generate estimates of the *local level* and *local slope* of the input signal. Indeed, setting $\mathbf{c}(n) = [a_n, b_n]^T$, where a_n, b_n represent the local level and local slope[†] at time n , we obtain from (6.3.8),

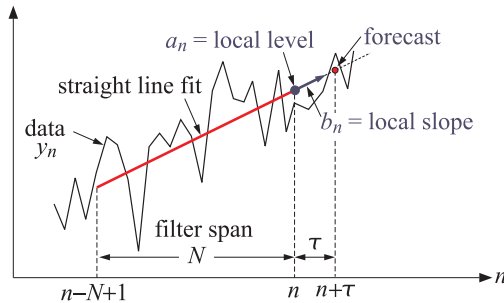
$$\begin{bmatrix} a_n \\ b_n \end{bmatrix} = (S_N^T S_N)^{-1} S_N^T \mathbf{y}(n) = (S_N^T S_N)^{-1} \sum_{k=0}^{N-1} \mathbf{u}_{-k} y_{n-k}$$

which is equivalent, component-wise, to the filtering equations:

$$\begin{aligned} a_n &= \sum_{k=0}^{N-1} h_a(k) y_{n-k} = \text{local level} \\ b_n &= \sum_{k=0}^{N-1} h_b(k) y_{n-k} = \text{local slope} \end{aligned} \quad (6.4.5)$$

Since, $\hat{y}_{n+\tau/n} = a_n + b_n \tau$, it is seen that the local level a_n is equal to $\hat{y}_{n/n}$. Similarly, the sum $a_n + b_n$ is the one-step-ahead forecast $\hat{y}_{n+1/n}$ obtained by extrapolating to time instant $n+1$ by extending the local level a_n along the straight line with slope b_n . This is depicted in the figure below. The sum, $a_n + b_n$, can be generated directly by the predictive FIR filter, $h_1(k) = h_a(k) + h_b(k)$, obtained by setting $\tau = 1$ in (6.4.1):

$$h_1(k) = \frac{2(2N-2-3k)}{N(N-1)}, \quad k = 0, 1, \dots, N-1 \quad (\text{predictive FIR filter}) \quad (6.4.6)$$



The filters $h_a(k)$, $h_b(k)$, and $h_1(k)$ find application in the technical analysis of financial markets [280]. Indeed, the filter $h_a(k)$ is equivalent to the so-called *linear regression indicator*, $h_b(k)$ corresponds to the *linear regression slope indicator*, and $h_1(k)$, to the *time series forecast indicator*. We discuss these in more detail, as well as other indicators, in Sections 6.14-6.24.

[†] a, b are the same as the components c_0, c_1 of the vector \mathbf{c} .

More generally, for order d polynomials, it follows from the solution (6.3.8), that the FIR filters \mathbf{h}_τ satisfy the moment constraints $S_N^T \mathbf{h}_\tau = \mathbf{u}_\tau$, or, component-wise:

$$\sum_{k=0}^{N-1} (-k)^r h_\tau(k) = \tau^r, \quad r = 0, 1, \dots, d \quad (6.4.7)$$

In fact, the solution $\mathbf{h}_\tau = S_N (S_N^T S_N)^{-1} \mathbf{u}_\tau$ is recognized (from Chap. 15) to be the minimum-norm, pseudoinverse, solution of the under-determined system $S_N^T \mathbf{h} = \mathbf{u}_\tau$, that is, it has minimum norm, or, minimum noise-reduction ratio, $\mathcal{R} = \mathbf{h}^T \mathbf{h} = \min$. A direct derivation is as follows. Introduce a $(d+1) \times 1$ vector of Lagrange multipliers, $\boldsymbol{\lambda} = [\lambda_0, \lambda_1, \dots, \lambda_d]^T$, and incorporate the constraint into the performance index,

$$\mathcal{J} = \mathbf{h}^T \mathbf{h} + 2\boldsymbol{\lambda}^T (\mathbf{u}_\tau - S_N^T \mathbf{h}) = \min$$

Then, its minimization leads to,

$$\frac{\partial \mathcal{J}}{\partial \mathbf{h}} = 2\mathbf{h} - 2S_N \boldsymbol{\lambda} = 0 \quad \Rightarrow \quad \mathbf{h} = S_N \boldsymbol{\lambda}$$

and, imposing the constraint $S_N^T \mathbf{h} = \mathbf{u}_\tau$ leads to the solutions for $\boldsymbol{\lambda}$ and for \mathbf{h} ,

$$\mathbf{u}_\tau = S_N^T \mathbf{h} = S_N^T S_N \boldsymbol{\lambda} \quad \Rightarrow \quad \boldsymbol{\lambda} = (S_N^T S_N)^{-1} \mathbf{u}_\tau \quad \Rightarrow \quad \mathbf{h} = S_N \boldsymbol{\lambda} = S_N (S_N^T S_N)^{-1} \mathbf{u}_\tau$$

Returning to Eq. (6.4.3) and setting $\tau = 0$, we note that the $d = 1$ local-level filter $h_a(k)$ satisfies the explicit constraints:

$$\sum_{k=0}^{N-1} h_a(k) = 1, \quad \sum_{k=0}^{N-1} k h_a(k) = 0 \quad (6.4.8)$$

The latter implies that its lag parameter \bar{n} is zero, and therefore, straight-line inputs will appear at the output undelayed (see Example 6.5.1). It has certain limitations as a lowpass filter that we discuss in Sec. 6.10, but its NRR is decreasing with N :

$$\mathcal{R} = \frac{2(2N-1)}{N(N+1)} \quad (6.4.9)$$

A direct consequence of Eq. (6.4.7) is that the filter $h_\tau(k)$ generates the exact predicted value of any polynomial of degree d , that is, for any polynomial $P(x)$ with degree up to d in the variable x , we have the exact convolutional result,

$$\sum_{k=0}^{N-1} P(n-k) h_\tau(k) = P(n+\tau), \quad \text{with } \deg(P) \leq d \quad (6.4.10)$$

6.5 Higher-Order Exponential Smoothing

For any value of d , the FIR filters \mathbf{h}_τ have length N and act on the N -dimensional data vector $\mathbf{y}(n) = [y_n, y_{n-1}, \dots, y_{n-N+1}]^T$. By contrast, the exponential smoother weights $\mathbf{h}_\tau(n)$ have an ever increasing length. Therefore, it proves convenient to recast them

recursively in time. The resulting recursive algorithm bears a very close similarity to the so-called *exact recursive-least-squares* (RLS) adaptive filters that we discuss in Chap. 16. Let us define the quantities,

$$\begin{aligned} R_n &= S_n^T W_n S_n = \sum_{k=0}^n \lambda^k \mathbf{u}_{-k} \mathbf{u}_{-k}^T = (d+1) \times (d+1) \text{ matrix} \\ \mathbf{r}_n &= S_n^T W_n \mathbf{y}(n) = \sum_{k=0}^n \lambda^k \mathbf{u}_{-k} y_{n-k} = (d+1) \times 1 \text{ vector} \end{aligned} \quad (6.5.1)$$

Then, the optimal polynomial coefficients (6.3.7) are:

$$\mathbf{c}(n) = R_n^{-1} \mathbf{r}_n \quad (6.5.2)$$

Clearly, the invertibility of R_n requires that $n \geq d$, which we will assume from now on. The sought recursions relate $\mathbf{c}(n)$ to the optimal coefficients $\mathbf{c}(n-1) = R_{n-1}^{-1} \mathbf{r}_{n-1}$ at the previous time instant $n-1$. Therefore, we must actually assume that $n > d$. To proceed, we note that the basis vector $\mathbf{u}_\tau = [1, \tau, \tau^2, \dots, \tau^d]^T$ satisfies the time-propagation property:

$$\mathbf{u}_{\tau+1} = F \mathbf{u}_\tau \quad (6.5.3)$$

where F is a $(d+1) \times (d+1)$ unit lower triangular matrix whose i th row consists of the binomial coefficients:

$$F_{ij} = \binom{i}{j}, \quad 0 \leq i \leq d, \quad 0 \leq j \leq i \quad (6.5.4)$$

This follows from the binomial expansion:

$$(\tau + 1)^i = \sum_{j=0}^i \binom{i}{j} \tau^j$$

Some examples of the F matrices are for $d = 0, 1, 2$:

$$F = [1], \quad F = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, \quad F = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad (6.5.5)$$

It follows from Eq. (6.5.3) that $\mathbf{u}_\tau = F \mathbf{u}_{\tau-1}$, and inverting $\mathbf{u}_{\tau-1} = F^{-1} \mathbf{u}_\tau$. The inverse matrix $G = F^{-1}$ will also be unit lower triangular with nonzero matrix elements obtained from the binomial expansion of $(\tau - 1)^i$:

$$G_{ij} = (-1)^{i-j} \binom{i}{j}, \quad 0 \leq i \leq d, \quad 0 \leq j \leq i \quad (6.5.6)$$

For example, we have for $d = 0, 1, 2$,

$$G = [1], \quad G = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}, \quad G = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 1 & -2 & 1 \end{bmatrix} \quad (6.5.7)$$

It follows from $\mathbf{u}_{\tau-1} = G \mathbf{u}_\tau$ that $\mathbf{u}_{-k-1} = G \mathbf{u}_{-k}$. This implies the following recursion for R_n :

$$\begin{aligned} R_n &= \sum_{k=0}^n \lambda^k \mathbf{u}_{-k} \mathbf{u}_{-k}^T = \mathbf{u}_0 \mathbf{u}_0^T + \sum_{k=1}^n \lambda^k \mathbf{u}_{-k} \mathbf{u}_{-k}^T \\ &= \mathbf{u}_0 \mathbf{u}_0^T + \lambda \sum_{k=1}^n \lambda^{k-1} \mathbf{u}_{-k} \mathbf{u}_{-k}^T \\ &= \mathbf{u}_0 \mathbf{u}_0^T + \lambda \sum_{k=0}^{n-1} \lambda^k \mathbf{u}_{-k-1} \mathbf{u}_{-k-1}^T \\ &= \mathbf{u}_0 \mathbf{u}_0^T + \lambda G \left(\sum_{k=0}^{n-1} \lambda^k \mathbf{u}_{-k} \mathbf{u}_{-k}^T \right) G^T = \mathbf{u}_0 \mathbf{u}_0^T + \lambda G R_{n-1} G^T \end{aligned}$$

where in the third line we changed summation variables from k to $k-1$, and in the fourth, we used $\mathbf{u}_{-k-1} = G \mathbf{u}_{-k}$. Similarly, we have for \mathbf{r}_n ,

$$\begin{aligned} \mathbf{r}_n &= \sum_{k=0}^n \lambda^k \mathbf{u}_{-k} y_{n-k} = \mathbf{u}_0 y_n + \sum_{k=1}^n \lambda^k \mathbf{u}_{-k} y_{n-k} \\ &= \mathbf{u}_0 y_n + \lambda \sum_{k=0}^{n-1} \lambda^k \mathbf{u}_{-k-1} y_{n-k-1} \\ &= \mathbf{u}_0 y_n + \lambda G \left(\sum_{k=0}^{n-1} \lambda^k \mathbf{u}_{-k} y_{(n-1)-k} \right) = \mathbf{u}_0 y_n + \lambda G \mathbf{r}_{n-1} \end{aligned}$$

Thus, R_n, \mathbf{r}_n satisfy the recursions:

$$\begin{aligned} R_n &= \mathbf{u}_0 \mathbf{u}_0^T + \lambda G R_{n-1} G^T \\ \mathbf{r}_n &= \mathbf{u}_0 y_n + \lambda G \mathbf{r}_{n-1} \end{aligned} \quad (6.5.8)$$

and they may be initialized to zero, $R_{-1} = 0$ and $\mathbf{r}_{-1} = 0$. Using $\hat{y}_{n+\tau/n} = \mathbf{u}_\tau^T \mathbf{c}(n)$, we may define the smoothed estimates, predictions, and the corresponding errors:

$$\begin{aligned} \hat{y}_{n/n} &= \mathbf{u}_0^T \mathbf{c}(n), & e_{n/n} &= y_n - \hat{y}_{n/n} \\ \hat{y}_{n+1/n} &= \mathbf{u}_1^T \mathbf{c}(n) = \mathbf{u}_0^T F^T \mathbf{c}(n), & e_{n+1/n} &= y_{n+1} - \hat{y}_{n+1/n} \\ \hat{y}_{n/n-1} &= \mathbf{u}_1^T \mathbf{c}(n-1) = \mathbf{u}_0^T F^T \mathbf{c}(n-1), & e_{n/n-1} &= y_n - \hat{y}_{n/n-1} \end{aligned} \quad (6.5.9)$$

where we used $\mathbf{u}_1 = F \mathbf{u}_0$. In the language of RLS adaptive filters, we may refer to $\hat{y}_{n/n-1}$ and $\hat{y}_{n/n}$ as the a priori and a posteriori estimates of y_n , respectively. Using the recursions (6.5.8), we may now obtain a recursion for $\mathbf{c}(n)$. Using $\mathbf{c}(n-1) = R_{n-1}^{-1} \mathbf{r}_{n-1}$ and the matrix relationship $GF = I$, we have,

$$\begin{aligned} R_n \mathbf{c}(n) &= \mathbf{r}_n = \mathbf{u}_0 y_n + \lambda G \mathbf{r}_{n-1} = \mathbf{u}_0 y_n + \lambda G R_{n-1} \mathbf{c}(n-1) \\ &= \mathbf{u}_0 y_n + \lambda G R_{n-1} G^T F^T \mathbf{c}(n-1) = \mathbf{u}_0 y_n + (R_n - \mathbf{u}_0 \mathbf{u}_0^T) F^T \mathbf{c}(n-1) \\ &= R_n F^T \mathbf{c}(n-1) + \mathbf{u}_0 (y_n - \mathbf{u}_0^T F^T \mathbf{c}(n-1)) = R_n F^T \mathbf{c}(n-1) + \mathbf{u}_0 (y_n - \hat{y}_{n/n-1}) \\ &= R_n F^T \mathbf{c}(n-1) + \mathbf{u}_0 e_{n/n-1} \end{aligned}$$

where in the second line we used $\lambda GR_{n-1}G^T = R_n - \mathbf{u}_0\mathbf{u}_0^T$. Multiplying both sides by R_n^{-1} , we obtain,

$$\mathbf{c}(n) = F^T \mathbf{c}(n-1) + R_n^{-1} \mathbf{u}_0 e_{n/n-1} \quad (6.5.10)$$

Again, in the language of RLS adaptive filters, we define the so-called a posteriori and a priori “Kalman gain” vectors \mathbf{k}_n and $\mathbf{k}_{n/n-1}$,

$$\mathbf{k}_n = R_n^{-1} \mathbf{u}_0, \quad \mathbf{k}_{n/n-1} = \lambda^{-1} F^T R_{n-1}^{-1} F \mathbf{u}_0 \quad (6.5.11)$$

and the “likelihood” variables,

$$\nu_n = \mathbf{u}_0^T \mathbf{k}_{n/n-1} = \lambda^{-1} \mathbf{u}_0^T F^T R_{n-1}^{-1} F \mathbf{u}_0 = \lambda^{-1} \mathbf{u}_1^T R_{n-1}^{-1} \mathbf{u}_1, \quad \mu_n = \frac{1}{1 + \nu_n} \quad (6.5.12)$$

Starting with the recursion $R_n = \mathbf{u}_0\mathbf{u}_0^T + \lambda GR_{n-1}G^T$ and multiplying both sides by R_n^{-1} from the left, then by F^T from the right, then by R_{n-1}^{-1} from the left, and then by F from the right, we may easily derive the equivalent relationship:

$$\lambda^{-1} F^T R_{n-1}^{-1} F = R_n^{-1} \mathbf{u}_0 \lambda^{-1} \mathbf{u}_0^T F^T R_{n-1}^{-1} F + R_n^{-1} \quad (6.5.13)$$

Multiplying on the right by \mathbf{u}_0 and using the definitions (6.5.11), we find

$$\begin{aligned} \mathbf{k}_{n/n-1} &= \mathbf{k}_n \nu_n + \mathbf{k}_n = (1 + \nu_n) \mathbf{k}_n, \quad \text{or,} \\ \mathbf{k}_n &= \mu_n \mathbf{k}_{n/n-1} \end{aligned} \quad (6.5.14)$$

Substituting this into (6.5.13), we obtain a recursion for the inverse matrix R_n^{-1} , which is effectively a variant of the matrix inversion lemma:

$$R_n^{-1} = \lambda^{-1} F^T R_{n-1}^{-1} F - \mu_n \mathbf{k}_{n/n-1} \mathbf{k}_{n/n-1}^T \quad (6.5.15)$$

This also implies that the parameter μ_n can be expressed as

$$\mu_n = 1 - \mathbf{u}_0^T R_n^{-1} \mathbf{u}_0 = 1 - \mathbf{u}_0^T \mathbf{k}_n \quad (6.5.16)$$

The a priori and a posteriori errors are also proportional to each other. Using (6.5.16), we find,

$$\hat{y}_{n/n} = \mathbf{u}_0^T \mathbf{c}(n) = \mathbf{u}_0^T (F^T \mathbf{c}(n-1) + \mathbf{k}_n e_{n/n-1}) = \hat{y}_{n/n-1} + (1 - \mu_n) e_{n/n-1} = y_n - \mu_n e_{n/n-1}$$

which implies that

$$e_{n/n} = \mu_n e_{n/n-1} \quad (6.5.17)$$

The coefficient updates (6.5.10) may now be expressed as:

$$\mathbf{c}(n) = F^T \mathbf{c}(n-1) + \mathbf{k}_n e_{n/n-1} \quad (6.5.18)$$

We summarize the complete set of computational steps for high-order exponential smoothing. We recall that the invertibility conditions require that we apply the recursions for $n > d$:

<ol style="list-style-type: none"> 1. $\mathbf{k}_{n/n-1} = \lambda^{-1} F^T R_{n-1}^{-1} F \mathbf{u}_0 = \lambda^{-1} F^T R_{n-1}^{-1} \mathbf{u}_1$ 2. $\nu_n = \mathbf{u}_0^T \mathbf{k}_{n/n-1}, \quad \mu_n = 1 / (1 + \nu_n)$ 3. $\mathbf{k}_n = \mu_n \mathbf{k}_{n/n-1}$ 4. $\hat{y}_{n/n-1} = \mathbf{u}_1^T \mathbf{c}(n-1), \quad e_{n/n-1} = y_n - \hat{y}_{n/n-1}$ 5. $e_{n/n} = \mu_n e_{n/n-1}, \quad \hat{y}_n = y_n - e_{n/n}$ 6. $\mathbf{c}(n) = F^T \mathbf{c}(n-1) + \mathbf{k}_n e_{n/n-1}$ 7. $R_n^{-1} = \lambda^{-1} F^T R_{n-1}^{-1} F - \mu_n \mathbf{k}_{n/n-1} \mathbf{k}_{n/n-1}^T$ 	(6.5.19)
--	----------

For $0 \leq n \leq d$, the fitting may be done with polynomials of varying degree $d_n = n$, and the coefficient estimate computed by explicit matrix inversion, $\mathbf{c}(n) = R_n^{-1} \mathbf{r}_n$. The above computational steps and initialization have been incorporated into the MATLAB function `ema` with usage:

```
C = ema(y, d, lambda);
```

% exponential moving average - exact version

The input y is an L -dimensional vector (row or column) of samples to be smoothed, with a total number $L > d$, and C is an $L \times (d+1)$ matrix whose n th row is the coefficient vector $\mathbf{c}(n)^T$. Thus, the first column, holds the smoothed estimate, the second column the estimated first derivative, and so on.

To understand the initialization process, consider an input sequence $\{y_0, y_1, y_2, \dots\}$ and the $d = 1$ smoother. At $n = 0$, we use a smoother of order $d_0 = 0$, constructing the quantities R_0, \mathbf{r}_0 using the definition (6.5.1):

$$R_0 = [1], \quad \mathbf{r}_0 = [y_0] \Rightarrow \mathbf{c}(0) = R_0^{-1} \mathbf{r}_0 = y_0$$

Next, at $n = 1$ we use a $d_1 = 1$ smoother, and definition (6.5.1) now implies,

$$\begin{aligned} R_1 &= \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} + \lambda \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} 1 + \lambda & -\lambda \\ -\lambda & \lambda \end{bmatrix} \\ \mathbf{r}_1 &= \begin{bmatrix} 1 \\ 0 \end{bmatrix} y_1 + \lambda \begin{bmatrix} 1 \\ -1 \end{bmatrix} y_0 = \begin{bmatrix} y_1 + \lambda y_0 \\ -\lambda y_0 \end{bmatrix} \end{aligned} \Rightarrow \mathbf{c}(1) = R_1^{-1} \mathbf{r}_1 = \begin{bmatrix} y_1 \\ y_1 - y_0 \end{bmatrix}$$

Starting with R_1, \mathbf{r}_1 , the recursion (6.5.8) may then be continued for $n \geq d + 1 = 2$. If we had instead $d = 2$, then there is one more initialization step, giving

$$R_2 = \begin{bmatrix} 1 + \lambda + \lambda^2 & -\lambda - 2\lambda^2 & \lambda + 4\lambda^2 \\ -\lambda - 2\lambda^2 & \lambda + 4\lambda^2 & -\lambda - 8\lambda^2 \\ \lambda + 4\lambda^2 & -\lambda - 8\lambda^2 & \lambda + 16\lambda^2 \end{bmatrix}, \quad \mathbf{r}_2 = \begin{bmatrix} y_2 + \lambda y_1 + \lambda^2 y_0 \\ -\lambda y_1 - 2\lambda^2 y_0 \\ \lambda y_1 + 4\lambda^2 y_0 \end{bmatrix}$$

resulting in

$$\mathbf{c}(2) = R_2^{-1} \mathbf{r}_2 = \begin{bmatrix} y_2 \\ 1.5y_2 - 2y_1 + 0.5y_0 \\ 0.5y_2 - y_1 + 0.5y_0 \end{bmatrix} \quad (6.5.20)$$

We note that the first $d + 1$ smoothed values get initialized to the first $d + 1$ values of the input sequence.

Example 6.5.1: Fig. 6.5.1 shows the output of the exact exponential smoother with $d = 1$ and $\lambda = 0.9$ applied on the same noiseless input s_n of Example 6.1.3. In addition, it shows the $d = 1$ FIR filter $h_a(k)$ designed to have zero lag according to Eq. (6.4.4).

Because $d = 1$, both filters can follow a linear signal. The input s_n (dashed curve) is barely visible under the filter outputs (solid curves). The length of the FIR filter was chosen according to the rule $N = (1 + \lambda) / (1 - \lambda)$.

The following MATLAB code generates the two graphs; it uses the function `upulse` which is a part of the OSP toolbox that generates a unit-pulse of prescribed duration

```
n = 0:300;
s = (20 + 0.8*n) .* upulse(n,75) + ...           % upulse is in the OSP toolbox
    (80 - 0.3*(n-75)) .* upulse(n-75,150) + ...
    (35 + 0.4*(n-225)) .* upulse(n-225,76);

la = 0.9; a1 = 1-la; d = 1;

C = ema(s,d,la);                               % exact exponential smoother output
x = C(:,1);

N = round((1+la)/(1-la));                       % equivalent FIR length, N=19

k=0:N-1;
ha = 2*(2*N-1-3*k)/N/(N+1);                   % zero-lag FIR filter

xh = filter(ha,1,s);                            % FIR filter output

figure; plot(n,s,'--', n,x,'-');               % left graph
figure; plot(n,s,'--', n,xh,'-');              % right graph
```

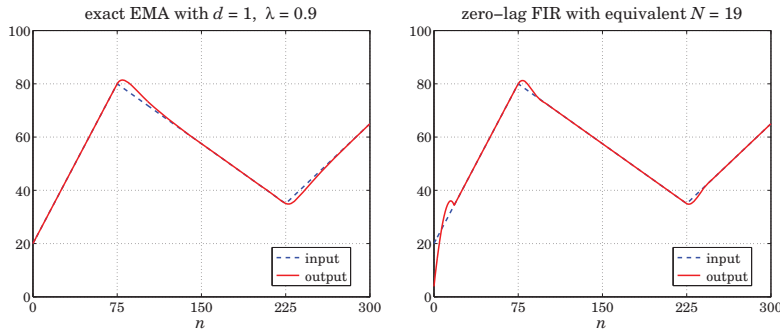


Fig. 6.5.1 Exact EMA with order $d = 1$, and zero-lag FIR filter with equivalent length.

Next, we add some noise $y_n = s_n + 4v_n$, where v_n is zero-mean, unit-variance, white noise. The top two graphs of Fig. 6.5.2 show the noisy signal y_n and the response of the exact EMA with $d = 0$ and $\lambda = 0.9$.

The bottom two graphs show the exact EMA with $d = 1$ as well as the response of the same zero-lag FIR filter to the noisy data. □

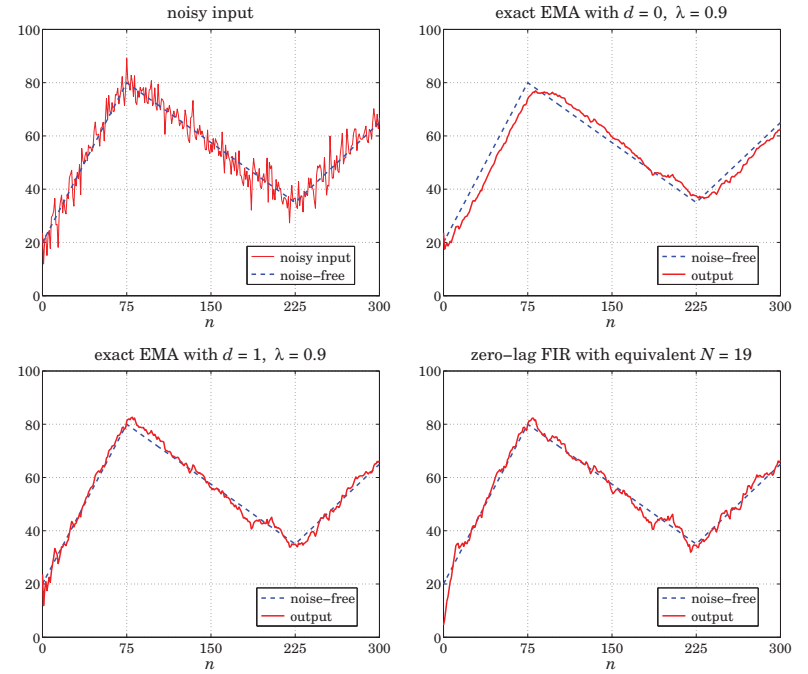


Fig. 6.5.2 EMA with order $d = 1$, and zero-lag FIR filter with equivalent length.

6.6 Steady-State Exponential Smoothing

Next, we look in more detail at the cases $d = 0, 1, 2$, which are the most commonly used in practice, with $d = 1$ providing the best performance and flexibility. We denote the polynomial coefficients by:

$$\mathbf{c}(n) = [a_n], \quad \mathbf{c}(n) = \begin{bmatrix} a_n \\ b_n \end{bmatrix}, \quad \mathbf{c}(n) = \begin{bmatrix} a_n \\ b_n \\ c_n \end{bmatrix} \quad (6.6.1)$$

Then, with $\mathbf{u}_\tau = [1]$, $\mathbf{u}_\tau = [1, \tau]^T$, and $\mathbf{u}_\tau = [1, \tau, \tau^2]^T$, the implied predicted estimates will be for arbitrary τ :

$$\begin{aligned} \hat{y}_{n+\tau/n} &= \mathbf{u}_\tau^T \mathbf{c}(n) = a_n \\ \hat{y}_{n+\tau/n} &= \mathbf{u}_\tau^T \mathbf{c}(n) = a_n + b_n \tau \\ \hat{y}_{n+\tau/n} &= \mathbf{u}_\tau^T \mathbf{c}(n) = a_n + b_n \tau + c_n \tau^2 \end{aligned} \quad (6.6.2)$$

Thus, a_n, b_n represent local estimates of the level and slope, respectively, and $2c_n$

represents the acceleration. The one-step-ahead predictions are,

$$\begin{aligned}\hat{y}_{n/n-1} &= \mathbf{u}_1^T \mathbf{c}(n-1) = a_{n-1} \\ \hat{y}_{n/n-1} &= \mathbf{u}_1^T \mathbf{c}(n-1) = a_{n-1} + b_{n-1} \\ \hat{y}_{n/n-1} &= \mathbf{u}_1^T \mathbf{c}(n-1) = a_{n-1} + b_{n-1} + c_{n-1}\end{aligned}\quad (6.6.3)$$

Denoting the a posteriori gains \mathbf{k}_n by,

$$\mathbf{k}_n = [\alpha(n)], \quad \mathbf{k}_n = \begin{bmatrix} \alpha_1(n) \\ \alpha_2(n) \end{bmatrix}, \quad \mathbf{k}_n = \begin{bmatrix} \alpha_1(n) \\ \alpha_2(n) \\ \alpha_3(n) \end{bmatrix}\quad (6.6.4)$$

then, the coefficient updates (6.5.18) take the forms, where $e_{n/n-1} = y_n - \hat{y}_{n/n-1}$,

$$\begin{aligned}a_n &= a_{n-1} + \alpha(n)e_{n/n-1} \\ \begin{bmatrix} a_n \\ b_n \end{bmatrix} &= \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a_{n-1} \\ b_{n-1} \end{bmatrix} + \begin{bmatrix} \alpha_1(n) \\ \alpha_2(n) \end{bmatrix} e_{n/n-1} \\ \begin{bmatrix} a_n \\ b_n \\ c_n \end{bmatrix} &= \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_{n-1} \\ b_{n-1} \\ c_{n-1} \end{bmatrix} + \begin{bmatrix} \alpha_1(n) \\ \alpha_2(n) \\ \alpha_3(n) \end{bmatrix} e_{n/n-1}\end{aligned}\quad (6.6.5)$$

Since $\mathbf{k}_n = R_n^{-1} \mathbf{u}_0$, the gains depend only on λ and n and converge to steady-state values for large n . For example, for $d = 0$, we have,

$$R_n = \sum_{k=0}^n \lambda^k = \frac{1 - \lambda^{n+1}}{1 - \lambda} \Rightarrow k_n = R_n^{-1} = \frac{1 - \lambda}{1 - \lambda^{n+1}} \rightarrow 1 - \lambda \equiv \alpha$$

Thus, the steady-state form of the $d = 0$ EMA smoother is as expected:

$$\begin{cases} e_{n/n-1} = y_n - \hat{y}_{n/n-1} = y_n - a_{n-1} \\ a_n = a_{n-1} + (1 - \lambda)e_{n/n-1} \end{cases} \quad (\text{single EMA, } d = 0) \quad (6.6.6)$$

initialized as usual at $a_{-1} = y_0$. The corresponding likelihood variable $\mu_n = 1 - \mathbf{u}_0^T \mathbf{k}_n$ tends to $\mu = 1 - (1 - \lambda) = \lambda$. Similarly, we find for $d = 1$,

$$R_n = \sum_{k=0}^n \lambda^k \begin{bmatrix} 1 \\ -k \end{bmatrix} [1, -k] = \sum_{k=0}^n \lambda^k \begin{bmatrix} 1 & -k \\ -k & k^2 \end{bmatrix} \equiv \begin{bmatrix} R_{00}(n) & R_{01}(n) \\ R_{10}(n) & R_{11}(n) \end{bmatrix}$$

where

$$\begin{aligned}R_{00}(n) &= \frac{1 - \lambda^{n+1}}{1 - \lambda}, \quad R_{01}(n) = R_{10}(n) = \frac{-\lambda + \lambda^{n+1}[1 + n(1 - \lambda)]}{(1 - \lambda)^2} \\ R_{11}(n) &= \frac{\lambda(1 + \lambda) - \lambda^{n+1}[1 + \lambda - 2n(1 - \lambda) + n^2(1 - \lambda)^2]}{(1 - \lambda)^3}\end{aligned}$$

which have the limit as $n \rightarrow \infty$,

$$\begin{aligned}R_n \rightarrow R &= \frac{1}{(1 - \lambda)^3} \begin{bmatrix} (1 - \lambda)^2 & -\lambda(1 - \lambda) \\ -\lambda(1 - \lambda) & \lambda(1 + \lambda) \end{bmatrix} \\ R^{-1} &= \begin{bmatrix} 1 - \lambda^2 & (1 - \lambda)^2 \\ (1 - \lambda)^2 & \lambda^{-1}(1 - \lambda)^3 \end{bmatrix}\end{aligned}\quad (6.6.7)$$

It follows that the asymptotic gain vector $\mathbf{k} = R^{-1} \mathbf{u}_0$ will be the first column of R^{-1} :

$$\mathbf{k}_n \rightarrow \mathbf{k} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} 1 - \lambda^2 \\ (1 - \lambda)^2 \end{bmatrix}\quad (6.6.8)$$

and the steady-state version of the $d = 1$ EMA smoother becomes:

$$\begin{cases} e_{n/n-1} = y_n - \hat{y}_{n/n-1} = y_n - (a_{n-1} + b_{n-1}) \\ \begin{bmatrix} a_n \\ b_n \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a_{n-1} \\ b_{n-1} \end{bmatrix} + \begin{bmatrix} 1 - \lambda^2 \\ (1 - \lambda)^2 \end{bmatrix} e_{n/n-1} \end{cases} \quad (\text{double EMA, } d = 1) \quad (6.6.9)$$

with estimated level $\hat{y}_{n/n} = a_n$ and one-step-ahead prediction $\hat{y}_{n+1/n} = a_n + b_n$. The corresponding limit of the likelihood parameter is $\mu = 1 - \mathbf{u}_0^T \mathbf{k} = 1 - (1 - \lambda^2) = \lambda^2$. The difference equation may be initialized at $a_{-1} = 2y_0 - y_1$ and $b_{-1} = y_1 - y_0$ to agree with the first outputs of the exact smoother. Indeed, iterating up to $n = 1$, we find the same answer for $\mathbf{c}(1)$ as the exact smoother:

$$\begin{aligned}\begin{bmatrix} a_0 \\ b_0 \end{bmatrix} &= \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a_{-1} \\ b_{-1} \end{bmatrix} + \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} e_{0/-1} = \begin{bmatrix} y_0 \\ y_1 - y_0 \end{bmatrix} \\ \begin{bmatrix} a_1 \\ b_1 \end{bmatrix} &= \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a_0 \\ b_0 \end{bmatrix} + \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} e_{1/0} = \begin{bmatrix} y_1 \\ y_1 - y_0 \end{bmatrix}\end{aligned}$$

Of course, other initializations are possible, a common one being to fit a straight line to the first few input samples and choose the intercept and slope as the initial values. This is the default method used by the function `stema` (see below). For the $d = 2$ case, the asymptotic matrix R is

$$R = \sum_{k=0}^{\infty} \lambda^k \mathbf{u}_{-k} \mathbf{u}_{-k}^T = \sum_{k=0}^{\infty} \lambda^k \begin{bmatrix} 1 & -k & k^2 \\ -k & k^2 & -k^3 \\ k^2 & -k^3 & k^4 \end{bmatrix}$$

which may be summed to

$$R = \begin{bmatrix} \frac{1}{1 - \lambda} & -\frac{\lambda}{(1 - \lambda)^2} & \frac{\lambda(1 + \lambda)}{(1 - \lambda)^3} \\ -\frac{\lambda}{(1 - \lambda)^2} & \frac{\lambda(1 + \lambda)}{(1 - \lambda)^3} & -\frac{\lambda(1 + 4\lambda + \lambda^2)}{(1 - \lambda)^4} \\ \frac{\lambda(1 + \lambda)}{(1 - \lambda)^3} & -\frac{\lambda(1 + 4\lambda + \lambda^2)}{(1 - \lambda)^4} & \frac{\lambda(1 + \lambda)(1 + 10\lambda + \lambda^2)}{(1 - \lambda)^5} \end{bmatrix}$$

with an inverse

$$R^{-1} = \begin{bmatrix} 1 - \lambda^3 & \frac{3}{2}(1 + \lambda)(1 - \lambda)^2 & \frac{1}{2}(1 - \lambda)^3 \\ \frac{3}{2}(1 + \lambda)(1 - \lambda)^2 & \frac{(1 + \lambda)(1 - \lambda)^3(1 + 9\lambda)}{4\lambda^2} & \frac{(1 - \lambda)^4(1 + 3\lambda)}{4\lambda^2} \\ \frac{1}{2}(1 - \lambda)^3 & \frac{(1 - \lambda)^4(1 + 3\lambda)}{4\lambda^2} & \frac{(1 - \lambda)^5}{4\lambda^2} \end{bmatrix}$$

The asymptotic gain vector $\mathbf{k} = R^{-1}\mathbf{u}_0$ and μ parameter are,

$$\mathbf{k} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} 1 - \lambda^3 \\ \frac{3}{2}(1 + \lambda)(1 - \lambda)^2 \\ \frac{1}{2}(1 - \lambda)^3 \end{bmatrix}, \quad \mu = 1 - \alpha_1 = \lambda^3 \quad (6.6.10)$$

and the steady-state $d = 2$ EMA smoother becomes:

$$\begin{array}{l} e_{n/n-1} = y_n - \hat{y}_{n/n-1} = y_n - (a_{n-1} + b_{n-1} + c_{n-1}) \\ \begin{bmatrix} a_n \\ b_n \\ c_n \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_{n-1} \\ b_{n-1} \\ c_{n-1} \end{bmatrix} + \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} e_{n/n-1} \end{array} \quad (\text{triple EMA, } d = 2) \quad (6.6.11)$$

They may be initialized to reach the same values at $n = 2$ as the exact smoother, that is, Eq. (6.5.20). This requirement gives:

$$\begin{bmatrix} a_{-1} \\ b_{-1} \\ c_{-1} \end{bmatrix} = \begin{bmatrix} y_2 - 3y_1 + 3y_0 \\ -1.5y_2 + 4y_1 - 2.5y_0 \\ 0.5y_2 - y_1 + 0.5y_0 \end{bmatrix} \Rightarrow \begin{bmatrix} a_2 \\ b_2 \\ c_2 \end{bmatrix} = \begin{bmatrix} y_2 \\ 1.5y_2 - 2y_1 + 0.5y_0 \\ 0.5y_2 - y_1 + 0.5y_0 \end{bmatrix}$$

Alternatively, they may be initialized by fitting a second degree polynomial to the first few input samples, as is done by default in the function `stema`, or they may be initialized to a zero vector, or to any other values, for example, $[a_{-1}, b_{-1}, c_{-1}] = [y_0, 0, 0]$.

For arbitrary polynomial order d , the matrix R_n converges to a $(d+1) \times (d+1)$ matrix R that must satisfy the Lyapunov-type equation:

$$R = \mathbf{u}_0 \mathbf{u}_0^T + \lambda G R G^T \quad (6.6.12)$$

where G is the backward boost matrix, $G = F^{-1}$. This follows by considering the limit of Eq. (6.5.1) as $n \rightarrow \infty$ and using the property $\mathbf{u}_{-k-1} = G\mathbf{u}_{-k}$. Multiplying from the left by F , and noting that $F\mathbf{u}_0 = \mathbf{u}_1$, we have

$$FR = \mathbf{u}_1 \mathbf{u}_0^T + \lambda R G^T \quad (6.6.13)$$

Taking advantage of the unit-lower-triangular nature of F and G , this equation can be written component-wise as follows:

$$\sum_{k=0}^i F_{ik} R_{kj} = \mathbf{u}_1(i) \mathbf{u}_0(j) + \lambda \sum_{k=0}^j R_{ik} G_{jk}, \quad 0 \leq i, j \leq d \quad (6.6.14)$$

Noting that $u_1(i) = 1$ and $u_0(j) = \delta(j)$, and setting first $i = j = 0$, we find

$$R_{00} = 1 + \lambda R_{00} \Rightarrow R_{00} = \frac{1}{1 - \lambda} \quad (6.6.15)$$

Then, setting $i = 0$ and $1 \leq j \leq d$,

$$R_{0j} = \lambda \sum_{k=0}^j R_{ik} G_{jk} = \lambda R_{0j} + \lambda \sum_{k=0}^{j-1} R_{0k} G_{jk}$$

which can be solved recursively for R_{0j} :

$$R_{0j} = R_{j0} = \frac{\lambda}{1 - \lambda} \sum_{k=0}^{j-1} R_{0k} G_{jk}, \quad j = 1, 2, \dots, d \quad (6.6.16)$$

Next, take $i \geq 1$ and $j \geq i$, and use the symmetry of R :

$$R_{ij} + \sum_{k=0}^{i-1} F_{ik} R_{kj} = \lambda R_{ij} + \lambda \sum_{k=0}^{j-1} R_{ik} G_{jk}$$

or, for $i = 1, 2, \dots, d$, $j = i, i+1, \dots, d$,

$$R_{ij} = R_{ji} = \frac{1}{1 - \lambda} \left[\lambda \sum_{k=0}^{j-1} R_{ik} G_{jk} - \sum_{k=0}^{i-1} F_{ik} R_{kj} \right] \quad (6.6.17)$$

To clarify the computations, we give the MATLAB code below:

```
R(1,1) = 1/(1-lambda);
for j=2:d+1,
    R(1,j) = lambda * R(1,1:j-1) * G(j,1:j-1)' / (1-lambda);
    R(j,1) = R(1,j);
end
for i=2:d+1,
    for j=i:d+1,
        R(i,j) = (lambda*R(i,1:j-1)*G(j,1:j-1)' - F(i,1:i-1)*R(1:i-1,j))/(1-lambda);
        R(j,i) = R(i,j);
    end
end
```

Once R is determined, one may calculate the gain vector $\mathbf{k} = R^{-1}\mathbf{u}_0$. Then, the overall filtering algorithm can be stated as follows, for $n \geq 0$,

$$\begin{array}{l} \hat{y}_{n/n-1} = \mathbf{u}_1^T \mathbf{c}(n-1) \\ e_{n/n-1} = y_n - \hat{y}_{n/n-1} \\ \mathbf{c}(n) = F^T \mathbf{c}(n-1) + \mathbf{k} e_{n/n-1} \end{array} \quad (\text{steady-state EMA}) \quad (6.6.18)$$

which requires specification of the initial vector $\mathbf{c}(-1)$. The transfer function from the input y_n to the signals $\mathbf{c}(n)$ can be determined by taking z-transforms of Eq. (6.6.18):

$$\mathbf{C}(z) = z^{-1} F^T \mathbf{C}(z) + \mathbf{k}(Y(z) - z^{-1} \mathbf{u}_1^T \mathbf{C}(z)), \quad \text{or,}$$

$$\mathbf{H}(z) = \frac{\mathbf{C}(z)}{Y(z)} = [I - (F^T - \mathbf{k}\mathbf{u}_1^T)z^{-1}]^{-1}\mathbf{k} \quad (6.6.19)$$

The computational steps (6.6.18) have been incorporated in the MATLAB function `stema`, with usage,

```
C = stema(y,d,lambda,cinit); % steady-state exponential moving average
```

where `C`, `y`, `d`, `lambda` have the same meaning as in the function `ema`. The parameter `cinit` is a $(d+1) \times 1$ column vector that represents the initial vector $\mathbf{c}(-1)$. If omitted, it defaults to fitting a polynomial of order d to the first L input samples, where L is the effective length corresponding to λ , that is, $L = (1 + \lambda)/(1 - \lambda)$. The fitting is carried out with the help of the function `lpbasis` from Chap. 3, and is given in MATLAB notation by:

```
cinit = lpbasis(L,d,-1)\y(1:L); % fit order-d polynomial to first L inputs
```

where the fit is carried out with respect to the time origin $n = -1$. The length L must be less than the length of the input vector \mathbf{y} . If not, another, shorter L can be used. Other initialization possibilities for `cinit` are summarized in the help file for `stema`.

To clarify the fitting operation, we note that fitting the first L samples y_n , $n = 0, 1, \dots, L-1$, to a polynomial of degree d centered at $n = -1$ amounts to the minimization of the performance index:

$$\mathcal{J} = \sum_{n=0}^{L-1} (y_n - p_n)^2 = \min, \quad p_n = \sum_{i=0}^d (n+1)^i c_i = \mathbf{u}_{n+1}^T \mathbf{c}$$

which can be written compactly as

$$\mathcal{J} = \|\mathbf{y} - \mathbf{S}\mathbf{c}\|^2 = \min, \quad \mathbf{S} = [\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_{n+1}, \dots, \mathbf{u}_L]^T$$

with solution $\mathbf{c} = (\mathbf{S}^T \mathbf{S})^{-1} \mathbf{S}^T \mathbf{y} = \mathbf{S} \backslash \mathbf{y}$ in MATLAB notation.[†] The actual fitted values $\mathbf{p} = [p_0, p_1, \dots, p_{L-1}]^T$ are then computed by $\mathbf{p} = \mathbf{S}\mathbf{c}$.

Selecting $n = -1$ as the centering time, assumes that the filtering operation will start at $n = 0$ requiring therefore the value $\mathbf{c}(-1)$. The centering can be done at any other reference time $n = n_0$, for example, one would choose $n_0 = L-1$ if the filtering operation were to start at $n = L$. The performance index would be then,

$$\mathcal{J} = \sum_{n=0}^{L-1} (y_n - p_n)^2 = \min, \quad p_n = \sum_{i=0}^d (n - n_0)^i c_i = \mathbf{u}_{n-n_0}^T \bar{\mathbf{c}}$$

with another set of coefficients $\bar{\mathbf{c}}$. The MATLAB implementation is in this case,

```
cinit = lpbasis(L,d,n0)\y(1:L); % fit order-d polynomial to first L inputs
```

From $\mathbf{u}_{n+1} = F\mathbf{u}_n$, we obtain $\mathbf{u}_{n+1} = F^{n_0+1}\mathbf{u}_{n-n_0}$. By requiring that the fitted polynomials be the same, $p_n = \mathbf{u}_{n+1}^T \mathbf{c} = \mathbf{u}_{n-n_0}^T \bar{\mathbf{c}}$, it follows that,

$$\bar{\mathbf{c}} = (F^T)^{n_0+1} \mathbf{c} \quad (6.6.20)$$

[†] assuming that \mathbf{S} has full rank, which requires $L > d$.

In Sec. 6.8, we discuss the connection to conventional multiple exponential smoothing obtained by filtering in cascade through $d+1$ copies of a single exponential smoothing filter $H(z) = \alpha/(1 - \lambda z^{-1})$, that is, through $[H(z)]^{d+1}$. Example 6.11.1 illustrates the above initialization methods, as well as how to map the initial values of $\mathbf{c}(n)$ to the initial values of the cascaded filter outputs.

6.7 Smoothing Parameter Selection

The performance of the steady-state EMA may be judged by computing the covariance of the estimates $\mathbf{c}(n)$, much like the case of the $d = 0$ smoother. Starting with $\mathbf{c}(n) = R_n^{-1} \mathbf{r}_n$ and $\mathbf{r}_n = S_n^T W_n \mathbf{y}(n)$, we obtain for the correlation matrix,

$$E[\mathbf{c}(n)\mathbf{c}^T(n)] = R_n^{-1} S_n^T W_n E[\mathbf{y}(n)\mathbf{y}^T(n)] W_n S_n R_n^{-1}$$

and for the corresponding covariance matrix,

$$\Sigma_{cc} = R_n^{-1} S_n^T W_n \Sigma_{yy} W_n S_n R_n^{-1} \quad (6.7.1)$$

Under the typical assumption that y_n is white noise, we have $\Sigma_{yy} = \sigma_y^2 I_{n+1}$, where I_{n+1} is the $(n+1)$ -dimensional unit matrix. Then,

$$\Sigma_{cc} = \sigma_y^2 R_n^{-1} Q_n R_n^{-1}, \quad Q_n = S_n^T W_n^2 S_n \quad (6.7.2)$$

In the limit $n \rightarrow \infty$, the matrices R_n, Q_n tend to steady-state values, so that

$$\Sigma_{cc} = \sigma_y^2 R^{-1} Q R^{-1} \quad (6.7.3)$$

where the limit matrices R, Q are given by

$$R = \sum_{k=0}^{\infty} \lambda^k \mathbf{u}_{-k} \mathbf{u}_{-k}^T, \quad Q = \sum_{k=0}^{\infty} \lambda^{2k} \mathbf{u}_{-k} \mathbf{u}_{-k}^T \quad (6.7.4)$$

Since $\hat{y}_{n/n} = \mathbf{u}_0^T \mathbf{c}(n)$ and $\hat{y}_{n+1/n} = \mathbf{u}_1^T \mathbf{c}(n)$, the corresponding variances will be:

$$\sigma_{\hat{y}_{n/n}}^2 = \mathbf{u}_0^T \Sigma_{cc} \mathbf{u}_0, \quad \sigma_{\hat{y}_{n+1/n}}^2 = \mathbf{u}_1^T \Sigma_{cc} \mathbf{u}_1 \equiv \sigma_e^2, \quad (6.7.5)$$

Because y_n was assumed to be an uncorrelated sequence, the two terms in the prediction error $e_{n+1/n} = y_{n+1} - \hat{y}_{n+1/n}$ will be uncorrelated since $\hat{y}_{n+1/n}$ depends only on data up to n . Therefore, the variance of the prediction error $e_{n+1/n}$ will be:

$$\sigma_e^2 = \sigma_y^2 + \sigma_y^2 = \sigma_y^2 [1 + \mathbf{u}_1^T R^{-1} Q R^{-1} \mathbf{u}_1] \quad (6.7.6)$$

For the case $d = 0$, we have

$$R = \sum_{k=0}^{\infty} \lambda^k = \frac{1}{1-\lambda}, \quad Q = \sum_{k=0}^{\infty} \lambda^{2k} = \frac{1}{1-\lambda^2}$$

which gives the usual results:

$$\sigma_{\hat{y}}^2 = \Sigma_{cc} = \frac{1-\lambda}{1+\lambda} \sigma_y^2, \quad \sigma_e^2 = \sigma_y^2 + \sigma_y^2 = \frac{2}{1+\lambda} \sigma_y^2$$

For $d = 1$, we have as in Eq. (6.6.7),

$$R = \frac{1}{(1-\lambda)^3} \begin{bmatrix} (1-\lambda)^2 & -\lambda(1-\lambda) \\ -\lambda(1-\lambda) & \lambda(1+\lambda) \end{bmatrix},$$

with the Q matrix being obtained from R by replacing $\lambda \rightarrow \lambda^2$,

$$Q = \frac{1}{(1-\lambda^2)^3} \begin{bmatrix} (1-\lambda^2)^2 & -\lambda^2(1-\lambda^2) \\ -\lambda^2(1-\lambda^2) & \lambda^2(1+\lambda^2) \end{bmatrix}$$

It follows then that

$$\Sigma_{cc} = \sigma_y^2 R^{-1} Q R^{-1} = \frac{1-\lambda}{(1+\lambda)^3} \begin{bmatrix} 1+4\lambda+5\lambda^2 & (1-\lambda)(1+3\lambda) \\ (1-\lambda)(1+3\lambda) & 2(1-\lambda)^2 \end{bmatrix} \quad (6.7.7)$$

The diagonal entries are the variances of the level and slope signals a_n, b_n :

$$\sigma_a^2 = \frac{(1-\lambda)(1+4\lambda+5\lambda^2)}{(1+\lambda)^3} \sigma_y^2, \quad \sigma_b^2 = \frac{2(1-\lambda)^3}{(1+\lambda)^3} \sigma_y^2 \quad (6.7.8)$$

For the prediction variance, we find

$$\sigma_{\hat{y}}^2 = \sigma_y^2 \mathbf{u}_1^T (R^{-1} Q R^{-1}) \mathbf{u}_1 = \frac{(1-\lambda)(\lambda^2+4\lambda+5)}{(1+\lambda)^3} \sigma_y^2 \quad (6.7.9)$$

which gives for the prediction error:

$$\sigma_e^2 = \sigma_y^2 + \sigma_{\hat{y}}^2 = \left[1 + \frac{(1-\lambda)(\lambda^2+4\lambda+5)}{(1+\lambda)^3} \right] \sigma_y^2 = \frac{2(3+\lambda)}{(1+\lambda)^3} \sigma_y^2 \quad (6.7.10)$$

In order to achieve an equivalent smoothing performance with a $d = 0$ EMA, one must equate the corresponding prediction variances, or mean-square errors. If λ_0, λ_1 denote the equivalent $d = 0$ and $d = 1$ parameters, the condition reads:

$$\frac{2}{1+\lambda_0} = \frac{2(3+\lambda_1)}{(1+\lambda_1)^3} \Rightarrow \lambda_0 = \frac{(1+\lambda_1)^3}{3+\lambda_1} - 1 \quad (6.7.11)$$

Eq. (6.7.11) may also be solved for λ_1 in terms of λ_0 ,

$$\lambda_1 = \frac{1}{3} D_0 + \frac{1+\lambda_0}{D_0} - 1, \quad D_0 = \left[27(1+\lambda_0) + \sqrt{27(1+\lambda_0)^2(26-\lambda_0)} \right]^{1/3} \quad (6.7.12)$$

Setting $\lambda_0 = 0$ gives $D_0 = (27 + 3\sqrt{78})^{1/3}$ and $\lambda_1 = 0.5214$. For all $\lambda_1 \geq 0.5214$, the equivalent λ_0 is non-negative and the NRR $\sigma_{\hat{y}}^2/\sigma_y^2$ of the prediction filter remains less than unity.

The corresponding FIR averager would have length $N_0 = (1+\lambda_0)/(1-\lambda_0)$, whereas an equivalent zero-lag FIR filter should have length N_1 that matches the corresponding NRRs. We have from Eq. (6.4.9):

$$\frac{2(2N_1-1)}{N_1(N_1+1)} = \frac{1-\lambda_0}{1+\lambda_0}$$

which gives,

$$\lambda_0 = \frac{N_1^2 - 3N_1 + 2}{N_1^2 + 5N_1 - 2} \Leftrightarrow N_1 = \frac{3 + 5\lambda_0 + \sqrt{33\lambda_0^2 + 30\lambda_0 + 1}}{2(1-\lambda_0)} \quad (6.7.13)$$

The MATLAB function `emap` implements Eq. (6.7.12),

```
1a1 = emap(1a0); % mapping equivalent lambda's between d = 0 and d = 1 EMAs
```

The computed λ_1 is an increasing function of λ_0 and varies over $0.5214 \leq \lambda_1 \leq 1$ as λ_0 varies over $0 \leq \lambda_0 \leq 1$.

Example 6.7.1: The lower-right graph of Fig. 6.7.1 shows a zero-lag FIR filter defined by Eq. (6.4.4) with length $N_1 = 100$ and applied to the noisy signal shown on the upper-left graph. The noisy signal was $y_n = 20 + 0.2n + 4v_n$, for $0 \leq n \leq 300$, with zero-mean, unit-variance, white noise v_n .

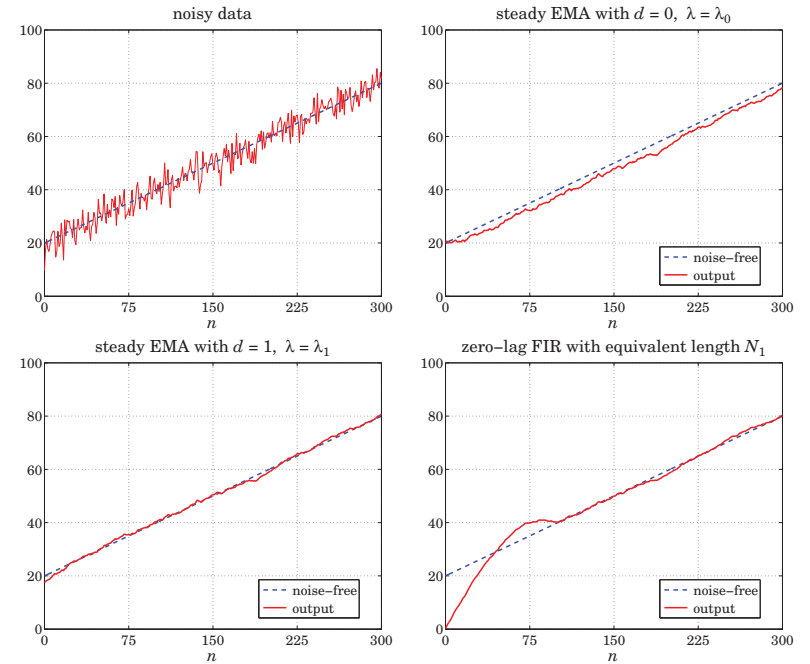


Fig. 6.7.1 Comparison of two equivalent steady-state EMAs with equivalent zero-lag FIR.

The equivalent EMA parameter for $d = 0$ was found from (6.7.13) to be $\lambda_0 = 0.9242$, which was then mapped to $\lambda_1 = 0.9693$ of an equivalent $d = 1$ EMA using Eq. (6.7.12). The upper-right graph shows the $d = 0$ EMA output, and the lower-left graph, the $d = 1$ EMA. The steady-state version was used for both EMAs with default initializations. The following MATLAB code illustrates the calculations:

```

t = 0:300; s = 20 + 0.2*t;
randn('state', 1000);
y = s + 4 * randn(size(t)); % noisy input

N1 = 100;
la0 = (N1^2-3*N1+2)/(N1^2+5*N1-2); % equivalent lambda_0
la1 = emap(la0); % equivalent lambda_1

C = stema(y,0,la0); x0 = C(:,1); % steady EMA with d = 0, lambda = lambda_0
C = stema(y,1,la1); x1 = C(:,1); % steady EMA with d = 1, lambda = lambda_1

k=0:N1-1; h = 2*(2*N1-1-3*k)/N1/(N1+1); % zero-lag FIR of length N1
% h = lpinterp(N1,1,-(N1-1)/2)'; % alternative calculation
xh = filter(h,1,y);

figure; plot(t,y,'-', t,s,'-'); figure; plot(t,s,'--', t,x0,'-');
figure; plot(t,s,'--', t,x1,'-'); figure; plot(t,s,'--', t,xh,'-');
    
```

We observe that all three outputs achieve comparable noise reduction. The $d = 0$ EMA suffers from the expected delay. Both the $d = 1$ EMA and the zero-lag FIR filter follow the straight-line input with no delay, after the initial transients have subsided. □

The choice of the parameter λ is more of an art than science. There do exist, however, criteria that determine an “optimum” value. Given the prediction $\hat{y}_{n/n-1} = \mathbf{u}_1^T \mathbf{c}(n-1)$ of y_n , and prediction error $e_{n/n-1} = y_n - \hat{y}_{n/n-1}$, the following criteria, to be minimized with respect to λ , are widely used:

$MSE = \text{mean}(e_{n/n-1}^2),$	(mean square error)	(6.7.14)
$MAE = \text{mean}(e_{n/n-1}),$	(mean absolute error)	
$MAPE = \text{mean}(100 e_{n/n-1}/y_n),$	(mean absolute percentage error)	

where the mean may be taken over the entire data set or over a portion of it. Usually, the criteria are evaluated over a range of λ 's and the minimum is selected. Typically, the criteria tend to underestimate the value of λ , that is, they produce too small a λ to be useful for smoothing purposes. Even so, the optimum λ has been used successfully for forecasting applications. The MATLAB function `emaerr` calculates these criteria for any vector of λ 's and determines the optimum λ in that range:

```
[err, lopt] = emaerr(y,d,lambda,type); % mean error criteria
```

where `type` takes one of the string values 'mse', 'mae', 'mape' and `err` is the criterion evaluated at the vector `lambda`, and `lopt` is the corresponding optimum λ .

Example 6.7.2: Fig. 6.7.2 shows the same Dow-Jones data of Example 6.1.2. The MSE criterion was searched over the range $0.1 \leq \lambda \leq 0.9$. The upper-left graph shows the MSE versus λ . The minimum occurs at $\lambda_{opt} = 0.61$.

The upper-right graph shows the $d = 1$ exact EMA run with $\lambda = \lambda_{opt}$. The EMA output is too rough to provide adequate smoothing. The other criteria are even worse. The MAE and MAPE optima both occur at $\lambda_{opt} = 0.56$. For comparison, the bottom two graphs show the $d = 1$ exact EMA run with the two higher values $\lambda = 0.90$ and $\lambda = 0.95$. The MATLAB code generating these graphs was as follows:

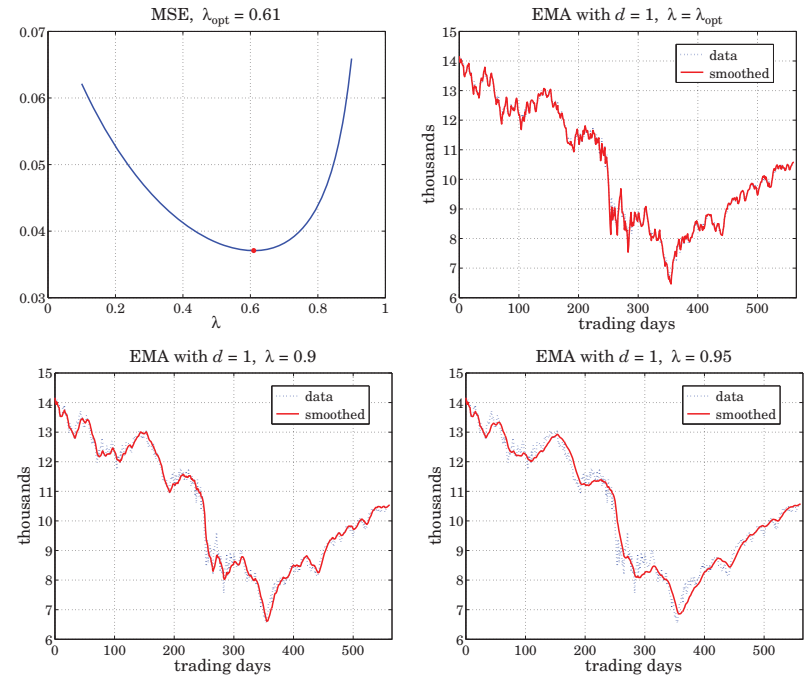


Fig. 6.7.2 MSE criterion for the DJIA data.

```

Y = loadfile('dow-oct07-dec09.dat'); % read data
y = Y(:,1)/1000; n = (0:length(y)-1)';

d = 1; u1 = ones(d+1,1); % polynomial order for EMA

la = linspace(0.1, 0.9, 81); % range of lambda's to search
[err, lopt] = emaerr(y,d,la,'mse'); % evaluate MSE at this range of lambda's

figure; plot(la,err, lopt,min(err),'-'); % upper-left graph

C = ema(y,d,lopt); yhat = C*u1; % upper-right graph
figure; plot(n,y,':', n,yhat,'-');

la=0.90; C = ema(y,d,la); yhat = C*u1; % bottom-left graph
figure; plot(n,y,':', n,yhat,'-'); % use la=0.95 for bottom-right
    
```

We note that the $d = 1$ smoother is more capable in following the signal than the $d = 0$ one. We plotted the forecasted value $\hat{y}_{n+1/n} = \mathbf{c}^T(n)\mathbf{u}_1$ versus n . Because the output matrix \mathbf{C} from the `ema` function has the $\mathbf{c}^T(n)$ as its rows, the entire vector of forecasted values can be calculated by acting by \mathbf{C} on the unit vector \mathbf{u}_1 , that is, $\mathbf{yhat} = \mathbf{C}^*\mathbf{u}_1$. □

6.8 Single, Double, and Triple Exponential Smoothing

Single exponential smoothing is the same as first-order, $d = 0$, steady-state exponential smoothing. We recall its filtering equation and corresponding transfer function:

$$a_n^{[1]} = \lambda a_{n-1}^{[1]} + \alpha y_n, \quad H^{[1]}(z) = H(z) = \frac{\alpha}{1 - \lambda z^{-1}} \quad (6.8.1)$$

where $\alpha = 1 - \lambda$. Double smoothing refers to filtering $a_n^{[1]}$ one more time through the same filter $H(z)$; and triple smoothing, two more times. The resulting filtering equations and transfer functions (from the overall input y_n to the final outputs) are:

$$a_n^{[2]} = \lambda a_{n-1}^{[2]} + \alpha a_n^{[1]}, \quad H^{[2]}(z) = \left(\frac{\alpha}{1 - \lambda z^{-1}} \right)^2 \quad (6.8.2)$$

$$a_n^{[3]} = \lambda a_{n-1}^{[3]} + \alpha a_n^{[2]}, \quad H^{[3]}(z) = \left(\frac{\alpha}{1 - \lambda z^{-1}} \right)^3$$

$$y_n \rightarrow \boxed{H} \rightarrow a_n^{[1]} \rightarrow \boxed{H} \rightarrow a_n^{[2]} \rightarrow \boxed{H} \rightarrow a_n^{[3]}$$

Thus, the filter $H(z)$ acts once, twice, three times, or in general $d+1$ times, in cascade, producing the outputs,

$$y_n \rightarrow \boxed{H} \rightarrow a_n^{[1]} \rightarrow \boxed{H} \rightarrow a_n^{[2]} \rightarrow \boxed{H} \rightarrow a_n^{[3]} \rightarrow \dots \rightarrow a_n^{[d]} \rightarrow \boxed{H} \rightarrow a_n^{[d+1]} \quad (6.8.3)$$

The transfer function and the corresponding causal impulse response from y_n to the r -th output $a_n^{[r]}$ are, for $r = 1, 2, \dots, d+1$ with $u(n)$ denoting the unit-step function:

$$H^{[r]}(z) = [H(z)]^r = \left(\frac{\alpha}{1 - \lambda z^{-1}} \right)^r \Leftrightarrow h^{[r]}(n) = \alpha^r \lambda^n \frac{(n+r-1)!}{n!(r-1)!} u(n) \quad (6.8.4)$$

Double and triple exponential smoothing are in a sense equivalent to the $d = 1$ and $d = 2$ steady-state EMA filters of Eq. (6.6.9) and (6.6.11). From Eq. (6.6.19), which in this case reads $\mathbf{H}(z) = [H_a(z), H_b(z)]^T$, we may obtain the transfer functions from y_n to the outputs a_n and b_n :

$$H_a(z) = \frac{(1-\lambda)(1+\lambda-2\lambda z^{-1})}{(1-\lambda z^{-1})^2}, \quad H_b(z) = \frac{(1-\lambda)^2(1-z^{-1})}{(1-\lambda z^{-1})^2} \quad (6.8.5)$$

It is straightforward to verify that H_a and H_b are related to H and H^2 by

$$\boxed{\begin{aligned} H_a &= 2H - H^2 = 1 - (1-H)^2 && \text{(local level filter)} \\ H_b &= \frac{\alpha}{\lambda}(H - H^2) && \text{(local slope filter)} \end{aligned}} \quad (6.8.6)$$

In the time domain this implies the following relationships between the a_n, b_n signals and the cascaded outputs $a_n^{[1]}, a_n^{[2]}$:

$$\boxed{\begin{aligned} a_n &= 2a_n^{[1]} - a_n^{[2]} = \text{local level} \\ b_n &= \frac{\alpha}{\lambda}(a_n^{[1]} - a_n^{[2]}) = \text{local slope} \end{aligned}} \quad (6.8.7)$$

6.8. Single, Double, and Triple Exponential Smoothing

which can be written in a 2×2 matrix form:

$$\begin{bmatrix} a_n \\ b_n \end{bmatrix} = \begin{bmatrix} 2 & -1 \\ \alpha/\lambda & -\alpha/\lambda \end{bmatrix} \begin{bmatrix} a_n^{[1]} \\ a_n^{[2]} \end{bmatrix} \Rightarrow \begin{bmatrix} a_n^{[1]} \\ a_n^{[2]} \end{bmatrix} = \begin{bmatrix} 1 & -\lambda/\alpha \\ 1 & -2\lambda/\alpha \end{bmatrix} \begin{bmatrix} a_n \\ b_n \end{bmatrix} \quad (6.8.8)$$

Similarly, for the $d = 2$ case, the transfer functions from y_n to a_n, b_n, c_n are:

$$\begin{aligned} H_a(z) &= \frac{\alpha[1 + \lambda + \lambda^2 - 3\lambda(1 + \lambda)z^{-1} + 3\lambda^2 z^{-2}]}{(1 - \lambda z^{-1})^3} \\ H_b(z) &= \frac{1}{2} \frac{\alpha^2(1 - z^{-1})[3(1 + \lambda) - (5\lambda + 1)z^{-1}]}{(1 - \lambda z^{-1})^3} \\ H_c(z) &= \frac{1}{2} \frac{\alpha^3(1 - z^{-1})^2}{(1 - \lambda z^{-1})^3} \end{aligned} \quad (6.8.9)$$

which are related to H, H^2, H^3 by the matrix transformation:

$$\begin{bmatrix} H \\ H^2 \\ H^3 \end{bmatrix} = \begin{bmatrix} 1 & -\lambda/\alpha & \lambda(\lambda + 1)/\alpha^2 \\ 1 & -2\lambda/\alpha & 2\lambda(2\lambda + 1)/\alpha^2 \\ 1 & -3\lambda/\alpha & 3\lambda(3\lambda + 1)/\alpha^2 \end{bmatrix} \begin{bmatrix} H_a \\ H_b \\ H_c \end{bmatrix} \quad (6.8.10)$$

implying the transformation between the outputs:

$$\begin{bmatrix} a_n^{[1]} \\ a_n^{[2]} \\ a_n^{[3]} \end{bmatrix} = \begin{bmatrix} 1 & -\lambda/\alpha & \lambda(\lambda + 1)/\alpha^2 \\ 1 & -2\lambda/\alpha & 2\lambda(2\lambda + 1)/\alpha^2 \\ 1 & -3\lambda/\alpha & 3\lambda(3\lambda + 1)/\alpha^2 \end{bmatrix} \begin{bmatrix} a_n \\ b_n \\ c_n \end{bmatrix} \quad (6.8.11)$$

with corresponding inverse relationships,

$$\begin{bmatrix} H_a \\ H_b \\ H_c \end{bmatrix} = \frac{1}{2\lambda^2} \begin{bmatrix} 6\lambda^2 & -6\lambda^2 & 2\lambda^2 \\ \alpha(1 + 5\lambda) & -2\alpha(1 + 4\lambda) & \alpha(1 + 3\lambda) \\ \alpha^2 & -2\alpha^2 & \alpha^2 \end{bmatrix} \begin{bmatrix} H \\ H^2 \\ H^3 \end{bmatrix} \quad (6.8.12)$$

$$\begin{bmatrix} a_n \\ b_n \\ c_n \end{bmatrix} = \frac{1}{2\lambda^2} \begin{bmatrix} 6\lambda^2 & -6\lambda^2 & 2\lambda^2 \\ \alpha(1 + 5\lambda) & -2\alpha(1 + 4\lambda) & \alpha(1 + 3\lambda) \\ \alpha^2 & -2\alpha^2 & \alpha^2 \end{bmatrix} \begin{bmatrix} a_n^{[1]} \\ a_n^{[2]} \\ a_n^{[3]} \end{bmatrix} \quad (6.8.13)$$

In particular, we have:

$$H_a = 3H - 3H^2 + H^3 = 1 - (1 - H)^3 \quad (6.8.14)$$

and

$$\hat{y}_{n/n} = a_n = 3a_n^{[1]} - 3a_n^{[2]} + a_n^{[3]} \quad (6.8.15)$$

More generally, for an order- d polynomial EMA, we have [243],

$$H_a = 1 - (1 - H)^{d+1} \quad (6.8.16)$$

$$\hat{y}_{n/n} = a_n = - \sum_{r=1}^{d+1} (-1)^r \binom{d+1}{r} a_n^{[r]} \quad (6.8.17)$$

6.9 Exponential Smoothing and Tukey's Twicing Operation

There is an interesting interpretation [255] of these results in terms of Tukey's *twicing* operation [257] and its generalizations to *thricing*, and so on. To explain twicing, consider a smoothing operation, which for simplicity we may assume that it can be represented by the matrix operation $\hat{\mathbf{y}} = H\mathbf{y}$, or if so preferred, in the z -domain as the multiplication of z -transforms $\hat{Y}(z) = H(z)Y(z)$.

The resulting residual error is $\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}} = (I - H)\mathbf{y}$. In the twicing procedure, the residuals are filtered through the same smoothing operation, which will smooth them further, $\hat{\mathbf{e}} = H\mathbf{e} = H(I - H)\mathbf{y}$, and the result is added to the original estimate to get an improved estimate:

$$\hat{\mathbf{y}}_{\text{impr}} = \hat{\mathbf{y}} + \hat{\mathbf{e}} = [H + H(I - H)]\mathbf{y} = [2H - H^2]\mathbf{y} \quad (6.9.1)$$

which is recognized as the operation (6.8.6). The process can be continued by repeating it on the residuals of the residuals, and so on. For example, at the next step, one would compute the new residual $\mathbf{r} = \mathbf{e} - \hat{\mathbf{e}} = (I - H)\mathbf{e} = (I - H)^2\mathbf{y}$, then filter it through H , $\hat{\mathbf{r}} = H\mathbf{r} = H(I - H)^2\mathbf{y}$, and add it to get the "thriced" improved estimate:

$$\hat{\mathbf{y}}_{\text{impr}} = \hat{\mathbf{y}} + \hat{\mathbf{e}} + \hat{\mathbf{r}} = [H + H(I - H) + H(I - H)^2]\mathbf{y} = [3H - 3H^2 + H^3]\mathbf{y} \quad (6.9.2)$$

which is Eq. (6.8.14). The process can be continued d times, resulting in,

$$\hat{\mathbf{y}}_{\text{impr}} = H[I + (I - H) + (I - H)^2 + \cdots + (I - H)^d]\mathbf{y} = [I - (I - H)^{d+1}]\mathbf{y} \quad (6.9.3)$$

Twicing and its generalizations can be applied with benefit to any smoothing operation, for example, if we used an LPRS filter designed by $B = \text{lprs}(N, d, s)$, the computational steps for twicing would be:

$$\hat{\mathbf{y}} = \text{lpfilt}(B, \mathbf{y}) \Rightarrow \mathbf{e} = \mathbf{y} - \hat{\mathbf{y}} \Rightarrow \hat{\mathbf{e}} = \text{lpfilt}(B, \mathbf{e}) \Rightarrow \hat{\mathbf{y}}_{\text{impr}} = \hat{\mathbf{y}} + \hat{\mathbf{e}}$$

A limitation of twicing is that, while it drastically improves the passband of a lowpass smoothing filter, it worsens its stopband. To see this, we write for the improved transfer function, $H_{\text{impr}}(z) = 1 - (1 - H(z))^{d+1}$. In the passband, H is near unity, say $H \approx 1 - \epsilon$, with $|\epsilon| \ll 1$, then,

$$H_{\text{impr}} = 1 - (1 - (1 - \epsilon))^{d+1} = 1 - \epsilon^{d+1}$$

thus, making the passband ripple $(d+1)$ orders of magnitude smaller. On the other hand, in the stopband, H is near zero, say $H \approx \pm\epsilon$, resulting in a worsened stopband,

$$H_{\text{impr}} = 1 - (1 \mp \epsilon)^{d+1} \approx 1 - (1 \mp (d+1)\epsilon) = \pm(d+1)\epsilon$$

The twicing procedure has been generalized by Kaiser and Hamming [258] to the so-called "filter sharpening" that improves both the passband and the stopband. For example, the lowest-order filter combination that achieves this is,

$$H_{\text{impr}} = H^2(3 - 2H) = 3H^2 - 2H^3 \quad (6.9.4)$$

6.10. Twicing and Zero-Lag Filters

where now both the passband and stopband ripples are replaced by $\epsilon \rightarrow \epsilon^2$. More generally, it can be shown [258] that the filter that achieves p th order tangency at $H = 0$ and q th order tangency at $H = 1$ is given by

$$H_{\text{impr}} = H^{p+1} \sum_{k=0}^q \frac{(p+k)!}{p!k!} (1-H)^k \quad (6.9.5)$$

The multiple exponential moving average case corresponds to $p = 0$ and $q = d$, resulting in $H_{\text{impr}} = 1 - (1 - H)^{d+1}$, whereas Eq. (6.9.4) corresponds to $p = q = 1$.

6.10 Twicing and Zero-Lag Filters

Another advantage of twicing and, more generally, filter sharpening is that the resulting improved smoothing filter always has zero lag, that is, $\bar{n} = 0$.

Indeed, assuming unity DC gain for the original filter, $H(z)|_{z=1} = 1$, it is straightforward to show that the general formula (6.9.5) gives for the first derivative:

$$H'_{\text{impr}}(z)|_{z=1} = 0 \quad (6.10.1)$$

which implies that its lag is zero, $\bar{n} = 0$, by virtue of Eq. (6.1.18). The twicing procedure, or its generalizations, for getting zero-lag filters is not limited to the exponential moving average. It can be applied to any other lowpass filter. For example, if we apply it to an ordinary length- N FIR averager, we would obtain:

$$H(z) = \frac{1}{N} \sum_{n=0}^{N-1} z^{-n} = \frac{1}{N} \frac{1 - z^{-N}}{1 - z^{-1}} \Rightarrow H_a(z) = 2H(z) - H^2(z) \quad (6.10.2)$$

The impulse response of $H_a(z)$ can be shown to be, where $0 \leq n \leq 2(N-1)$,

$$h_a(n) = \left(\frac{2N-1-n}{N^2} \right) [u(n) - 2u(n-N) + u(n-2N+1)] \quad (6.10.3)$$

It is straightforward to show that $\bar{n}_a = 0$ and that its noise-reduction ratio is

$$\mathcal{R} = \frac{8N^2 - 6N + 1}{3N^3} \quad (6.10.4)$$

Because of their zero-lag property, double and triple EMA filters are used as *trend indicators* in the financial markets [297,298]. The application of twicing to the modified exponential smoother of Eq. (2.3.5) gives rise to a similar indicator called the *instantaneous trendline* [285], and further discussed in Problem 6.8. We discuss such market indicators in Sections 6.14-6.24.

The zero-lag property for a causal lowpass filter comes at a price, namely, that although its magnitude response is normalized to unity at $\omega = 0$ and has a flat derivative there, it typically bends upwards developing a bandpass peak near DC before it attenuates to smaller values at higher frequencies. See, for example, Fig. 6.10.1.

This behavior might be deemed to be objectionable since it tends to unevenly amplify the low-frequency passband of the filter.

To clarify these remarks, consider a lowpass filter $H(\omega)$ (with real-valued impulse response h_n) satisfying the gain and flatness conditions $H(0) = 1$ and $H'(0) = 0$ at $\omega = 0$. The flatness condition implies the zero-lag property $\bar{n} = 0$. Using these two conditions, it is straightforward to verify that the second derivative of the magnitude response at DC is given by:

$$\frac{d^2}{d\omega^2} |H(\omega)|^2_{\omega=0} = 2 \operatorname{Re}[H''(0)] + 2|H'(0)|^2 = 2 \operatorname{Re}[H''(0)] = -2 \sum_{n=0}^{\infty} n^2 h_n \tag{6.10.5}$$

Because $\bar{n} = \sum_n n h_n = 0$, it follows that some of the coefficients h_n must be negative, which can cause (6.10.5) to become positive, implying that $\omega = 0$ is a local minimum and hence the response will rise for ω s immediately beyond DC. This is demonstrated for example in Problem 6.7 by Eq. (6.25.1), so that,

$$\frac{d^2}{d\omega^2} |H(\omega)|^2_{\omega=0} = -2 \sum_{n=0}^{\infty} n^2 h_n = \frac{4\lambda^2}{(1-\lambda)^2}$$

A similar calculation yields the result,

$$\frac{d^2}{d\omega^2} |H(\omega)|^2_{\omega=0} = -2 \sum_{n=0}^{\infty} n^2 h_n = \frac{1}{3}(N-1)(N-2)$$

for the optimum zero-lag FIR filter of Eq. (6.4.4),

$$h_a(k) = \frac{2(2N-1-3k)}{N(N+1)}, \quad k = 0, 1, \dots, N-1 \tag{6.10.6}$$

We note that the first derivative of the magnitude response $|H(\omega)|^2$ is always zero at DC, regardless of whether the filter has zero lag or not. Indeed, it follows from $H(0) = 1$ and the reality of h_n that,

$$\frac{d}{d\omega} |H(\omega)|^2_{\omega=0} = 2 \operatorname{Re}[H'(0)] = 0 \tag{6.10.7}$$

Example 6.10.1: Zero-Lag Filters. In Problem 6.7, we saw that the double EMA filter has transfer function and magnitude response:

$$H_a(z) = \frac{(1-\lambda)(1+\lambda-2\lambda z^{-1})}{(1-\lambda z^{-1})^2}$$

$$|H_a(\omega)|^2 = \frac{(1-\lambda)^2 [1+2\lambda+5\lambda^2-4\lambda(1+\lambda)\cos\omega]}{[1-2\lambda\cos\omega+\lambda^2]^2}$$

and that a secondary peak develops at,

$$\cos \omega_{\max} = \frac{1+4\lambda-\lambda^2}{2(1+\lambda)}, \quad |H_a(\omega_{\max})|^2 = \frac{(1+\lambda)^2}{1+2\lambda}$$

The left graph of Fig. 6.10.1 shows the magnitude responses for the two cases of $\lambda = 0.8$ and $\lambda = 0.9$. The calculated peak frequencies are $\omega_{\max} = 0.1492$ and $\omega_{\max} = 0.0726$ rads/sample, corresponding to periods of $2\pi/\omega_{\max} = 42$ and 86 samples/cycle. The peak points are indicated by black dots on the graph.

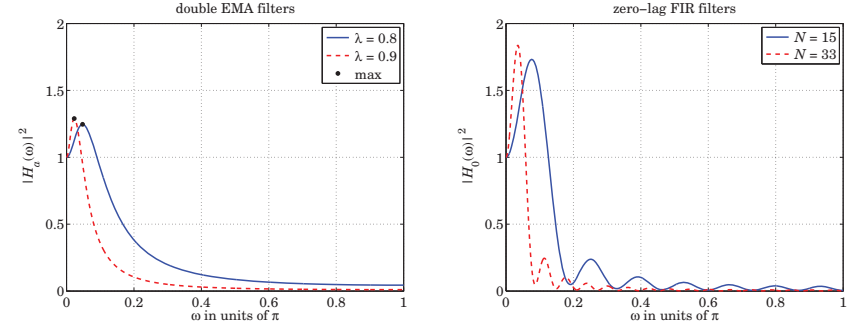


Fig. 6.10.1 Double EMA and zero-lag FIR filter responses.

The right graph shows the magnitude responses of the optimum zero-lag FIR filter $h_a(k)$ of Eq. (6.10.6) for the two lengths $N = 15$ and $N = 33$. The lengths N were calculated to achieve equivalent behavior in the vicinity of DC, i.e., equal second derivatives at $\omega = 0$,

$$\frac{4\lambda^2}{(1-\lambda)^2} = \frac{1}{3}(N-1)(N-2) \Rightarrow N = \frac{3}{2} + \sqrt{\frac{12\lambda^2}{(1-\lambda)^2} + \frac{1}{4}}$$

The magnitude responses were computed from the above formulas for the double EMA cases, and by the following MATLAB code in the FIR cases:

```
w = linspace(0,1,1001); % omega in units of pi
N = 15; k = (0:N-1);
h = 2*(2*N-1-3*k)/N/(N+1);
H2 = abs(freqz(h,1,pi*w)).^2; % magnitude response squared
```

We observe from these examples that the zero-lag filters have less than desirable passbands. However, they do act as lowpass filters, attenuating the high frequencies and thereby smoothing the input signal. □

Local Level, Local Slope, and Local Acceleration Filters

Since the twicing operation can be applied to any lowpass filter $H(z)$ resulting in the zero-lag local-level filter, $H_a(z) = 2H(z) - H^2(z)$, it raises the question as to what would be the corresponding local-slope filter $H_b(z)$, in other words, what is the generalization of Eqs. (6.8.6) for an arbitrary filter $H(z)$, and similarly, what is the generalization of Eq. (6.8.12) for the thricing operations.

Starting with an arbitrary causal lowpass filter $H(z)$, with impulse response $h(n)$, and assuming that it has unity gain at DC, the local level, slope, and acceleration filters depend on the following two filter moments:

$$\mu_1 = \bar{n} = \sum_{n=0}^{\infty} n h(n), \quad \mu_2 = \sum_{n=0}^{\infty} n^2 h(n) \tag{6.10.8}$$

In terms of these parameters, the generalization of Eq. (6.8.6) is then,

$$\boxed{\begin{aligned} H_a(z) &= 2H(z) - H^2(z) && \text{(local level filter)} \\ H_b(z) &= \frac{1}{\mu_1} [H(z) - H^2(z)] && \text{(local slope filter)} \end{aligned}} \quad (6.10.9)$$

while the generalization of (6.8.12) is,

$$\begin{bmatrix} H_a(z) \\ H_b(z) \\ H_c(z) \end{bmatrix} = \frac{1}{2\mu_1^3} \begin{bmatrix} 6\mu_1^3 & -6\mu_1^3 & 2\mu_1^3 \\ \mu_2 + 4\mu_1^2 & -2(\mu_2 + 3\mu_1^2) & \mu_2 + 2\mu_1^2 \\ \mu_1 & -2\mu_1 & \mu_1 \end{bmatrix} \begin{bmatrix} H(z) \\ H^2(z) \\ H^3(z) \end{bmatrix} \quad (6.10.10)$$

and in particular,

$$H_a = 3H - 3H^2 + H^3 = 1 - (1 - H)^3 \quad (6.10.11)$$

For an EMA filter, $h(n) = (1 - \lambda)\lambda^n u(n)$, we have,

$$\mu_1 = \frac{\lambda}{1 - \lambda}, \quad \mu_2 = \frac{\lambda(1 + \lambda)}{(1 - \lambda)^2}$$

and Eqs. (6.10.9) and (6.10.10) reduce to (6.8.6) and (6.8.12), respectively. To justify (6.10.9), consider a noiseless straight-line input signal, $y_n = a + bn$. Since it is linearly rising and noiseless, the local-level will be itself, and the local-slope will be constant, that is, we may define,

$$a_n \equiv a + bn, \quad b_n \equiv b$$

Following the calculation leading to Eq. (6.1.24), we can easily verify that the two successive outputs, $a_n^{[1]}, a_n^{[2]}$, from the twice-cascaded filter $h(n)$, will be,

$$\begin{aligned} a_n^{[1]} &= a + b(n - \mu_1) = (a - \mu_1 b) + bn = a + bn - \mu_1 b = a_n - \mu_1 b_n \\ a_n^{[2]} &= (a - \mu_1 b - \mu_1 b) + bn = (a - 2\mu_1 b) + bn = a_n - 2\mu_1 b_n \end{aligned}$$

These may be solved for a_n, b_n in terms of $a_n^{[1]}, a_n^{[2]}$, leading to the following time-domain relationships, which are equivalent to Eqs. (6.10.9),

$$\begin{aligned} a_n &= 2a_n^{[1]} - a_n^{[2]} \\ b_n &= \frac{1}{\mu_1} (a_n^{[1]} - a_n^{[2]}) \end{aligned}$$

For the thricing case, consider a noiseless input that is quadratically varying with time, $y_n = a + bn + cn^2$, so that its local level, local slope, and local acceleration may be defined as,[†]

$$a_n \equiv a + bn + cn^2, \quad b_n \equiv b + 2cn, \quad c_n \equiv c$$

[†]true acceleration would be represented by $2c$.

Upon passing through the first stage of $h(n)$, the output will be,

$$\begin{aligned} a_n^{[1]} &= \sum_k [a + b(n - k) + c(n - k)^2] h(k) \\ &= \sum_k [a + b(n - k) + c(n^2 - 2nk + k^2)] h(k) \\ &= a + b(n - \mu_1) + c(n^2 - 2n\mu_1 + \mu_2) \\ &= (a - b\mu_1 + c\mu_2) + (b - 2c\mu_1)n + cn^2 \end{aligned}$$

and applying the same formula to the second and third stages of $h(n)$, we find the outputs,

$$\begin{aligned} a_n^{[1]} &= (a - b\mu_1 + c\mu_2) + (b - 2c\mu_1)n + cn^2 \\ a_n^{[2]} &= (a - 2b\mu_1 + 2c\mu_2 + 2c\mu_1^2) + (b - 4c\mu_1)n + cn^2 \\ a_n^{[3]} &= (a - 3b\mu_1 + 3c\mu_2 + 6c\mu_1^2) + (b - 6c\mu_1)n + cn^2 \end{aligned}$$

which can be re-expressed in terms of the a_n, b_n, c_n signals,

$$\begin{aligned} a_n^{[1]} &= a_n - \mu_1 b_n + \mu_2 c_n \\ a_n^{[2]} &= a_n - 2\mu_1 b_n + 2(\mu_2 + \mu_1^2) c_n \\ a_n^{[3]} &= a_n - 3\mu_1 b_n + 3(\mu_2 + 2\mu_1^2) c_n \end{aligned}$$

Solving these for a_n, b_n, c_n leads to the time-domain equivalent of Eq. (6.10.10),

$$\begin{bmatrix} a_n \\ b_n \\ c_n \end{bmatrix} = \frac{1}{2\mu_1^3} \begin{bmatrix} 6\mu_1^3 & -6\mu_1^3 & 2\mu_1^3 \\ \mu_2 + 4\mu_1^2 & -2(\mu_2 + 3\mu_1^2) & \mu_2 + 2\mu_1^2 \\ \mu_1 & -2\mu_1 & \mu_1 \end{bmatrix} \begin{bmatrix} a_n^{[1]} \\ a_n^{[2]} \\ a_n^{[3]} \end{bmatrix}$$

and in particular,

$$a_n = 3a_n^{[1]} - 3a_n^{[2]} + a_n^{[3]}$$

6.11 Basis Transformations and EMA Initialization

The transformation matrix between the $\mathbf{c}(n) = [c_0(n), c_1(n), \dots, c_d(n)]^T$ basis and the cascaded basis $\mathbf{a}(n) = [a_n^{[1]}, a_n^{[2]}, \dots, a_n^{[d+1]}]^T$ can be written in the general form:

$$\mathbf{a}(n) = M\mathbf{c}(n) \Rightarrow a_n^{[r]} = \sum_{i=0}^d M_{ri} c_i(n), \quad r = 1, 2, \dots, d+1 \quad (6.11.1)$$

The matrix elements M_{ri} can be found by looking at the special case when the input is a polynomial of degree d ,

$$x_{n+\tau} = \sum_{i=0}^d \tau^i c_i(n)$$

The convolutional output of the filter $H^{[r]}(z)$ is then,

$$a_n^{[r]} = \sum_{k=0}^{\infty} h^{[r]}(k)x_{n-k} = \sum_{k=0}^{\infty} h^{[r]}(k) \sum_{i=0}^d (-k)^i c_i(n)$$

It follows that,

$$M_{ri} = \sum_{k=0}^{\infty} h^{[r]}(k) (-k)^i = \sum_{k=0}^{\infty} \alpha^r \lambda^k \frac{(k+r-1)!}{k!(r-1)!} (-k)^i \quad (6.11.2)$$

with $1 \leq r \leq d+1$ and $0 \leq i \leq d$. The matrices for $d=1$ and $d=2$ in Eqs. (6.8.8) and (6.8.11) are special cases of (6.11.2). The function `emat` calculates M numerically,

```
M = emat(d, lambda); % polynomial to cascaded basis transformation matrix
```

One of the uses of this matrix is to determine the initial condition vector in the $a_n^{[r]}$ basis, $\mathbf{a}_{init} = M\mathbf{c}_{init}$, where \mathbf{c}_{init} is more easily determined, for example, using the default method of the function `stema`.

The function `mema` implements the multiple exponential moving average in cascade form, generating the individual outputs $a_n^{[r]}$:

```
[a,A] = mema(y,d,la,ainit); % multiple exponential moving average
```

where A is an $N \times (d+1)$ matrix whose n th row is $\mathbf{a}(n)^T = [a_n^{[1]}, a_n^{[2]}, \dots, a_n^{[d+1]}]$, and a is the a_n output of Eq. (6.8.17). The $(d+1) \times 1$ vector `ainit` represents the initial values of $\mathbf{a}(n)$, that is, $\mathbf{a}_{init} = [a_{init}^{[1]}, a_{init}^{[2]}, \dots, a_{init}^{[d+1]}]^T$. If the argument `ainit` is omitted, it defaults to the following least-squares fitting procedure:

```
L = round((1+la)/(1-la)); % effective length of single EMA
cinit = lpbasis(L,d,-1)\y(1:L); % fit order-d polynomial to first L inputs
M = emat(d, la); % transformation matrix to cascaded basis
ainit = M*cinit; % (d+1)x1 initial vector
```

The function `mema` calculates the filter outputs using the built-in function `filter` to implement filtering by the single EMA filter $H(z) = \alpha/(1-\lambda z^{-1})$ and passing each output to the next stage, for example, with $a_n^{[0]} = y_n$,

$$a^{[r]} = \text{filter}(\alpha, [1, -\lambda], a^{[r-1]}, \lambda a_{init}^{[r]}), \quad r = 1, 2, \dots, d+1 \quad (6.11.3)$$

The last argument of `filter` imposes the proper initial state on the filtering operation (the particular form is dictated from the fact that `filter` uses the transposed realization.) Eq. (6.11.3) is equivalent to the operations:

$$a_n^{[r]} = \lambda a_{n-1}^{[r]} + \alpha a_n^{[r-1]}, \quad r = 1, 2, \dots, d+1 \quad (6.11.4)$$

Example 6.11.1: EMA Initialization. To clarify these operations and the initializations, we consider a small example using $d=1$ (double EMA) and $\lambda = 0.9$, $\alpha = 1 - \lambda = 0.1$. The data vector y has length 41 and is given in the code segment below.

The top two graphs in Fig. 6.11.1 show the default initialization method in which a linear fit (because $d=1$) is performed to the first $L = (1 + \lambda)/(1 - \lambda) = 19$ data samples. In the

bottom two graphs, the initialization is based on performing the linear fit to just the first $L = 5$ samples.

In all cases, the linearly-fitted segments are shown on the graphs (short dashed lines). In the left graphs, the initialization parameters $\mathbf{c}_{init}, \mathbf{a}_{init}$ were determined at time $n = -1$ and processing began at $n = 0$. In the right graphs, the $\mathbf{c}_{init}, \mathbf{a}_{init}$ were recalculated to correspond to time $n = L-1$ (i.e., $n = 18$ and $n = 4$ for the top and bottom graphs), and processing was started at $n = L$. The table below displays the computations for the left and right bottom graphs.

For both the left and right tables, the same five data samples $\{y_n, 0 \leq n \leq 4\}$ were used to determine the initialization vectors \mathbf{c}_{init} , which were then mapped into $\mathbf{a}_{init} = M\mathbf{c}_{init}$. The transformation matrix M is in this example (cf. Eq. (6.8.8)):

$$M = \begin{bmatrix} 1 & -\lambda/\alpha \\ 1 & -2\lambda/\alpha \end{bmatrix} = \begin{bmatrix} 1 & -9 \\ 1 & -18 \end{bmatrix}$$

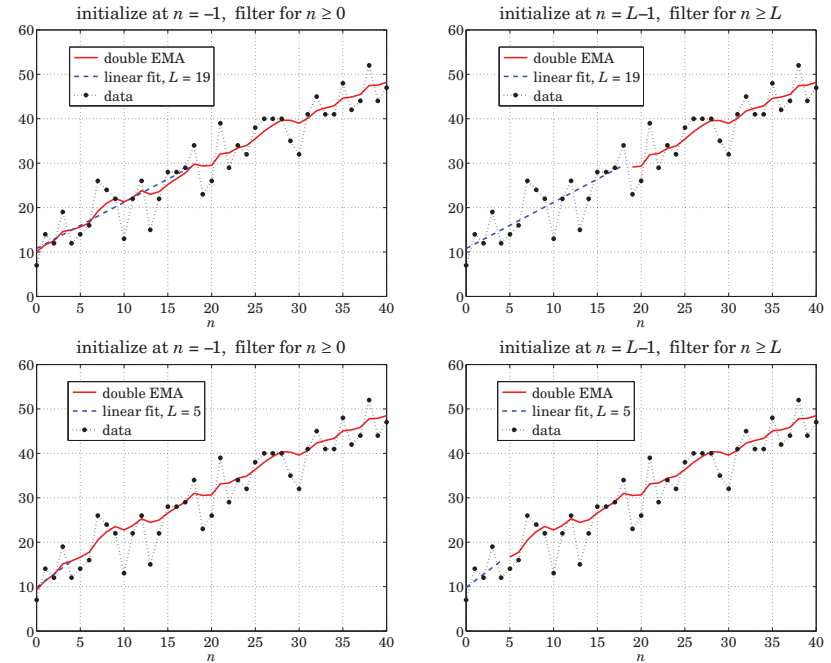


Fig. 6.11.1 Double-EMA initialization examples.

n	y_n	$a_n^{[1]}$	$a_n^{[2]}$	n	y_n	$a_n^{[1]}$	$a_n^{[2]}$
-1		-5.2000	-18.7000	-1			
0	7	-3.9800	-17.2280	0	7		
1	14	-2.1820	-15.7234	1	14		
2	12	-0.7638	-14.2274	2	12		
3	19	1.2126	-12.6834	3	19		
4	12	2.2913	-11.1860	4	12	2.3000	-11.2000
5	14	3.4622	-9.7211	5	14	3.4700	-9.7330
6	16	4.7160	-8.2774	6	16	4.7230	-8.2874
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
39	44	39.2235	30.5592	39	44	39.2237	30.5596
40	47	40.0011	31.5034	40	47	40.0013	31.5037

For the left table, the data fitting relative to $n = -1$ gives:

$$c_{init} = \begin{bmatrix} 8.3 \\ 1.5 \end{bmatrix} \Rightarrow a_{init} = M c_{init} = \begin{bmatrix} 1 & -9 \\ 1 & -18 \end{bmatrix} \begin{bmatrix} 8.3 \\ 1.5 \end{bmatrix} = \begin{bmatrix} -5.2 \\ -18.7 \end{bmatrix}$$

obtained from $c_{init} = S \backslash y(1:L)$, indeed, with $S = \text{lpbasis}(L, d, -1)$, we find

$$S = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \\ 1 & 5 \end{bmatrix} \Rightarrow c_{init} = (S^T S)^{-1} S^T y_{1:L} = \begin{bmatrix} 0.8 & 0.5 & 0.2 & -0.1 & -0.4 \\ -0.2 & -0.1 & 0.0 & 0.1 & 0.2 \end{bmatrix} \begin{bmatrix} 7 \\ 14 \\ 12 \\ 19 \\ 12 \end{bmatrix} = \begin{bmatrix} 8.3 \\ 1.5 \end{bmatrix}$$

These initial values are shown at the top of the left table. The rest of the table entries are computed by cranking the difference equations for $n \geq 0$,

$$a_n^{[1]} = \lambda a_{n-1}^{[1]} + \alpha y_n$$

$$a_n^{[2]} = \lambda a_{n-1}^{[2]} + \alpha a_n^{[1]}$$

for example,

$$a_0^{[1]} = \lambda a_{-1}^{[1]} + \alpha y_0 = (0.9)(-5.2) + (0.1)(7) = -3.980$$

$$a_0^{[2]} = \lambda a_{-1}^{[2]} + \alpha a_0^{[1]} = (0.9)(-18.7) + (0.1)(-3.98) = -17.228$$

For the right table, the initialization coefficients relative to $n = L - 1 = 4$ may be determined by boosting those for $n = -1$ by $L = 5$ time units:

$$\bar{c}_{init} = (F^T)^L c_{init} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}^5 \begin{bmatrix} 8.3 \\ 1.5 \end{bmatrix} = \begin{bmatrix} 1 & 5 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 8.3 \\ 1.5 \end{bmatrix} = \begin{bmatrix} 15.8 \\ 1.5 \end{bmatrix}$$

$$\bar{a}_{init} = M \bar{c}_{init} = \begin{bmatrix} 1 & -9 \\ 1 & -18 \end{bmatrix} \begin{bmatrix} 15.8 \\ 1.5 \end{bmatrix} = \begin{bmatrix} 2.3 \\ -11.2 \end{bmatrix}$$

Alternatively, \bar{c}_{init} can be computed from $c_{init} = \text{lpbasis}(L, d, L-1) \backslash y(1:L)$, i.e.,

$$S = \begin{bmatrix} 1 & -4 \\ 1 & -3 \\ 1 & -2 \\ 1 & -1 \\ 1 & 0 \end{bmatrix} \Rightarrow \bar{c}_{init} = (S^T S)^{-1} S^T y_{1:L} = \begin{bmatrix} -0.2 & 0.0 & 0.2 & 0.4 & 0.6 \\ -0.2 & -0.1 & 0.0 & 0.1 & 0.2 \end{bmatrix} \begin{bmatrix} 7 \\ 14 \\ 12 \\ 19 \\ 12 \end{bmatrix} = \begin{bmatrix} 15.8 \\ 1.5 \end{bmatrix}$$

The \bar{a}_{init} values are shown on the right table at the $n = 4$ line. The rest of the table is computed by cranking the above difference equations starting at $n = 5$. For example,

$$a_5^{[1]} = \lambda a_4^{[1]} + \alpha y_5 = (0.9)(2.3) + (0.1)(14) = 3.47$$

$$a_5^{[2]} = \lambda a_4^{[2]} + \alpha a_5^{[1]} = (0.9)(-11.2) + (0.1)(3.47) = -9.733$$

We note that the filtered values at $n = L - 1 = 4$ on the left table and the initial values on the right table are very close to each other, and therefore, the two initialization methods produce very comparable results for the output segments $n \geq L$. The following MATLAB code illustrates the generation of the bottom graphs in Fig. 6.11.1:

```

y = [ 7 14 12 19 12 14 16 26 24 22 13 22 26 15 22 28 28 29 34 23 26 ...
      39 29 34 32 38 40 40 40 35 32 41 45 41 41 48 42 44 52 44 47 ]';
n = (0:length(y)-1)';

d=1; F=binmat(d,1); L=5; % F = boost matrix - not needed
la = 0.9; al = 1-la; % use this L for the top two graphs
% L = round((1+la)/(1-la));

cinit = lpbasis(L,d,-1)\y(1:L); % fit relative to n = -1
M = emat(d,la); % transformation matrix
ainit = M*cinit; % initial values for cascade realization

C = stema(y,d,la,cinit); % needed for testing purposes only
[a,A] = mema(y,d,la,ainit); % filter for n ≥ 0

N1 = norm(A-C*M') + norm(a-C(:,1)); % compare stema and mema outputs

t = (0:L-1)'; p = lpbasis(L,d,-1)*cinit; % initial L fitted values

figure; plot(n,y, '.', n,a, '- ', t,p, '-- ', n,y, ': '); % bottom left graph

cinit = lpbasis(L,d,L-1)\y(1:L); % fit relative to n = L - 1
% or, multiply previous cinit by (F')^L
ainit = M*cinit; % initial values for cascade realization

nL = n(L+1:end); yL = y(L+1:end); % restrict input to n ≥ L

C = stema(yL,d,la,cinit); % needed for testing purposes only
[a,A] = mema(yL,d,la,ainit); % filter for n ≥ L

N2 = norm(A-C*M') + norm(a-C(:,1)); % compare stema and mema outputs

t = (0:L-1)'; p = lpbasis(L,d,L-1)*cinit; % initial L fitted values

figure; plot(n,y, '.', nL,a, '- ', t,p, '-- ', n,y, ': '); % bottom right graph

Ntot = N1 + N2 % overall test - should be zero

```

The first initialization scheme (finding $\mathbf{c}_{\text{init}}, \mathbf{a}_{\text{init}}$ at $n = -1$ and starting filtering at $n = 0$) is generally preferable because it produces an output of the same length as the input. \square

An alternative initialization scheme that is more common in financial market trading applications of EMA is discussed in Sec. 6.17.

6.12 Holt's Exponential Smoothing

We recall that the $d = 1$ steady-state EMA was given by

$$\begin{bmatrix} a_n \\ b_n \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a_{n-1} \\ b_{n-1} \end{bmatrix} + \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} (y_n - a_{n-1} - b_{n-1}) \quad (6.12.1)$$

with asymptotic gain factors $\alpha_1 = 1 - \lambda^2$ and $\alpha_2 = (1 - \lambda)^2$, which are both computable from a single λ parameter.

Holt [240] has generalized (6.12.1) to allow arbitrary values for α_1, α_2 . The additional flexibility has been found to be effective in applications. There is an alternative way of writing (6.12.1). From the first equation, we have

$$a_n = a_{n-1} + b_{n-1} + \alpha_1 (y_n - a_{n-1} - b_{n-1}) \Rightarrow y_n - a_{n-1} - b_{n-1} = \frac{1}{\alpha_1} (a_n - a_{n-1} - b_{n-1})$$

and substituting into the second,

$$b_n = b_{n-1} + \alpha_2 (y_n - a_{n-1} - b_{n-1}) = b_{n-1} + \frac{\alpha_2}{\alpha_1} (a_n - a_{n-1} - b_{n-1})$$

Defining $\tilde{\alpha}_2 = \alpha_2 / \alpha_1$, we may write the new system as follows:

$$\begin{aligned} a_n &= a_{n-1} + b_{n-1} + \alpha_1 (y_n - a_{n-1} - b_{n-1}) \\ b_n &= b_{n-1} + \tilde{\alpha}_2 (a_n - a_{n-1} - b_{n-1}) \end{aligned} \quad (6.12.2)$$

and defining the effective λ -parameters $\lambda_1 = 1 - \alpha_1$ and $\tilde{\lambda}_2 = 1 - \tilde{\alpha}_2$,

$$\begin{cases} a_n = \lambda_1 (a_{n-1} + b_{n-1}) + \alpha_1 y_n \\ b_n = \tilde{\lambda}_2 b_{n-1} + \tilde{\alpha}_2 (a_n - a_{n-1}) \end{cases} \quad \text{(Holt's exponential smoothing)} \quad (6.12.3)$$

Eq. (6.12.1) is referred to an exponential smoothing with "local trend". The first equation tracks the local level a_n , and the second, the local slope b_n , which is being determined by smoothing the first-order difference of the local level $a_n - a_{n-1}$.

The predicted value is as usual $\hat{y}_{n/n-1} = a_{n-1} + b_{n-1}$, and for the next time instant, $\hat{y}_{n+1/n} = a_n + b_n$, and τ steps ahead, $\hat{y}_{n+\tau/n} = a_n + b_n \tau$. The MATLAB function `holt` implements Eq. (6.12.1):

```
C = holt(y,a1,a2,cinit); % Holt's exponential smoothing
```

where \mathbf{C} has the same meaning as `stema`, its n th row $\mathbf{c}^T(n) = [a_n, b_n]$ holding the local level and slope at time n . The initialization vector `cinit` can be chosen as in `stema` by a linear fit to the first L samples of \mathbf{y} , where $L = (1 + \lambda) / (1 - \lambda)$, with λ determined from α_1 from the relationship $\alpha_1 = 1 - \lambda^2$ or $\lambda = \sqrt{1 - \alpha_1}$. Another possibility is to choose $\mathbf{c}_{\text{init}} = [y_0, 0]^T$, or, $[y_0, y_1 - y_0]^T$.

Like `emaerr`, the MATLAB function `holterr` evaluates the MSE/MAE/MAPE errors over any set of parameter pairs (α_1, α_2) and produces the corresponding optimum pair $(\alpha_{1,\text{opt}}, \alpha_{2,\text{opt}})$:

```
[err,a1opt,a2opt] = holterr(y,a1,a2,type,cinit); % mean error measures
```

By taking z -transforms of Eq. (6.12.1), we obtain the transfer functions from y_n to the two outputs a_n, b_n :

$$\begin{aligned} H_a(z) &= \frac{\alpha_1 + (\alpha_2 - \alpha_1)z^{-1}}{1 + (\alpha_1 + \alpha_2 - 2)z^{-1} + (1 - \alpha_1)z^{-2}} \\ H_b(z) &= \frac{\alpha_2(1 - z^{-1})}{1 + (\alpha_1 + \alpha_2 - 2)z^{-1} + (1 - \alpha_1)z^{-2}} \end{aligned} \quad (6.12.4)$$

The transfer function from y_n to the predicted output $\hat{y}_{n+1/n}$ is $H(z) = H_a(z) + H_b(z)$. Making the usual assumption that y_n is a white noise sequence, the variance of $\hat{y}_{n+1/n}$ will be $\sigma_{\hat{y}}^2 = \mathcal{R}\sigma_y^2$, where \mathcal{R} is the NRR of $H(z)$:

$$\mathcal{R} = \sum_{n=0}^{\infty} h^2(n) = \frac{2\alpha_1^2 + \alpha_1\alpha_2 + 2\alpha_2}{\alpha_1(4 - 2\alpha_1 - \alpha_2)} \quad (6.12.5)$$

This reduces to Eq. (6.7.9) in the EMA case of $\alpha_1 = 1 - \lambda^2$ and $\alpha_2 = (1 - \lambda)^2$, while Eq. (6.12.4) reduces to (6.8.5). It can be shown that \mathcal{R} remains less than unity for $0 \leq \alpha_1 \leq 1$ and $0 \leq \alpha_2 \leq 2\alpha_1(1 - \alpha_1) / (1 + \alpha_1)$, with \mathcal{R} reaching unity at $\alpha_1 = \sqrt{2} - 1 = 0.4142$ and $\alpha_2 = 2\alpha_1(1 - \alpha_1) / (1 + \alpha_1) = 2(3 - 2\sqrt{2}) = 0.3431$.

6.13 State-Space Models for Holt's Method

Just like the $d = 0$ local level case could be represented by an innovations model, so can the linear trend model. We may define the model by assuming that the prediction is perfect and thus, the prediction error $e_{n/n-1} = y_n - \hat{y}_{n/n-1} \equiv \varepsilon_n$ is a white noise signal. The state vector may be defined as $\mathbf{x}_n = [a_n, b_n]^T$, leading to the innovations state-space model,

$$\begin{aligned} y_n &= [1, 1]\mathbf{x}_{n-1} + \varepsilon_n \\ \mathbf{x}_n &= \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \mathbf{x}_{n-1} + \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} \varepsilon_n \end{aligned} \quad (6.13.1)$$

Eliminating the state vector, we may obtain the transfer function from the innovations sequence ε_n to the observation y_n ,

$$\frac{Y(z)}{\mathcal{E}(z)} = \frac{1 + (\alpha_1 + \alpha_2 - 2)z^{-1} + (1 - \alpha_1)z^{-2}}{(1 - z^{-1})^2}$$

which leads to an ARIMA(0,2,2) or IMA(2,2) equivalent signal model:

$$\nabla^2 y_n = y_n - 2y_{n-1} + y_{n-2} = \varepsilon_n + (\alpha_1 + \alpha_2 - 2)\varepsilon_{n-1} + (1 - \alpha_1)\varepsilon_{n-2} \quad (6.13.2)$$

For $\alpha_1 = 1 - \lambda^2$ and $\alpha_2 = (1 - \lambda)^2$, we obtain the special case [253],

$$\frac{Y(z)}{\mathcal{E}(z)} = \frac{(1 - \lambda z^{-1})^2}{(1 - z^{-1})^2}, \quad \nabla^2 y_n = \varepsilon_n - 2\lambda\varepsilon_{n-1} + \lambda^2\varepsilon_{n-2} \quad (6.13.3)$$

The following more conventional formulation of the linear trend state-space model has steady-state Kalman filtering equations that are equivalent to Holt's method:

$$\begin{bmatrix} a_{n+1} \\ b_{n+1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a_n \\ b_n \end{bmatrix} + \begin{bmatrix} w_n \\ u_n \end{bmatrix}, \quad y_n = [1, 0] \begin{bmatrix} a_n \\ b_n \end{bmatrix} + v_n \quad (6.13.4)$$

where a_n, b_n represent the local level and local slope, respectively, and w_n, u_n, v_n are zero-mean, mutually uncorrelated, white-noise signals of variances $\sigma_w^2, \sigma_u^2, \sigma_v^2$. Denoting the state vector and its filtered and predicted estimates by,

$$\mathbf{x}_n = \begin{bmatrix} a_n \\ b_n \end{bmatrix}, \quad \hat{\mathbf{x}}_{n/n} = \begin{bmatrix} \hat{a}_n \\ \hat{b}_n \end{bmatrix}, \quad \hat{\mathbf{x}}_{n+1/n} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{a}_n \\ \hat{b}_n \end{bmatrix}$$

it turns out that the steady-state Kalman filtering equations take exactly the innovations form of Eq. (6.13.1):

$$\varepsilon_n = y_n - (\hat{a}_{n-1} + \hat{b}_{n-1}), \quad \begin{bmatrix} \hat{a}_n \\ \hat{b}_n \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{a}_{n-1} \\ \hat{b}_{n-1} \end{bmatrix} + \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} \varepsilon_n \quad (6.13.5)$$

where α_1, α_2 are related to the noise variances by:

$$\frac{\sigma_w^2}{\sigma_v^2} = \frac{\alpha_1^2 + \alpha_1\alpha_2 - 2\alpha_2}{1 - \alpha_1}, \quad \frac{\sigma_u^2}{\sigma_v^2} = \frac{\alpha_2^2}{1 - \alpha_1} \quad (6.13.6)$$

State-space models provide a modern way of thinking about exponential smoothing and will be explored further in Chap. 13.

There is an extensive literature on exponential smoothing, a small subset of which is [232-279]. There are many other variants (no less than 15), such as multiplicative, seasonal, adaptive versions. A recent review of all cases that emphasizes the state-space point of view is found in [239].

We finish by mentioning the Holt-Winters generalization [241] of Holt's method to seasonal data. In addition to tracking the level and slope signals a_n, b_n the method also tracks the local seasonal component, say s_n . For the additive version, we have:

$$\begin{cases} a_n = \lambda_1(a_{n-1} + b_{n-1}) + \alpha_1(y_n - s_{n-D}) \\ b_n = \lambda_2 b_{n-1} + \alpha_2(a_n - a_{n-1}) \\ s_n = \lambda_3 s_{n-D} + \alpha_3(y_n - a_{n-1} - b_{n-1}) \end{cases} \quad \text{(Holt-Winters)} \quad (6.13.7)$$

where D is the assumed periodicity of the seasonal data, and α_3 and $\lambda_3 = 1 - \alpha_3$ are the smoothing parameters associated with the seasonal component. The predicted estimate is obtained by $\hat{y}_{n+1/n} = a_n + b_n + s_{n-D}$.

6.14 Filtering Methods in Financial Market Trading

Technical analysis of financial markets refers to a family of signal processing methods and indicators used by stock market traders to make sense of the constantly fluctuating market data and arrive at successful "buy" or "sell" decisions.

Both linear and nonlinear filtering methods are used. A comprehensive reference on such methods is the Achelis book [280]. Some additional references are [281-347].

Here, we look briefly at some widely used indicators arising from FIR or EMA filters, and summarize their properties and their MATLAB implementation. In order to keep the discussion self-contained, some material from the previous sections is repeated.

6.15 Moving Average Filters - SMA, WMA, TMA, EMA

Among the linear filtering methods are smoothing filters that are used to smooth out the daily fluctuations and bring out the trends in the data. The most common filters are the *simple moving average* (SMA) and the *exponentially weighted moving average* (EMA), and variations, such as the *weighted or linear moving average* (WMA) and the *triangular moving average* (TMA). The impulse responses of these filters are:

$$\begin{aligned} \text{(SMA)} \quad h(n) &= \frac{1}{N}, \quad 0 \leq n \leq N-1 \\ \text{(WMA)} \quad h(n) &= \frac{2(N-n)}{N(N+1)}, \quad 0 \leq n \leq N-1 \\ \text{(TMA)} \quad h(n) &= \frac{N - |n - N + 1|}{N^2}, \quad 0 \leq n \leq 2N-2 \\ \text{(EMA)} \quad h(n) &= (1 - \lambda)\lambda^n, \quad 0 \leq n < \infty \end{aligned} \quad (6.15.1)$$

with transfer functions,

$$\begin{aligned} \text{(SMA)} \quad H(z) &= \frac{1 + z^{-1} + z^{-2} + \dots + z^{-N+1}}{N} = \frac{1}{N} \frac{1 - z^{-N}}{1 - z^{-1}} \\ \text{(WMA)} \quad H(z) &= \frac{2}{N(N+1)} \frac{N - (N+1)z^{-1} + z^{-N-1}}{(1 - z^{-1})^2} \\ \text{(TMA)} \quad H(z) &= \left[\frac{1}{N} \frac{1 - z^{-N}}{1 - z^{-1}} \right]^2 \\ \text{(EMA)} \quad H(z) &= \frac{\alpha}{1 - \lambda z^{-1}}, \quad \alpha = 1 - \lambda \end{aligned} \quad (6.15.2)$$

where N denotes the filter span for the SMA and WMA cases, while for the EMA case, λ is a forgetting factor such that $0 < \lambda < 1$, which is usually specified in terms an equivalent FIR length N given by the following condition, which implies that the SMA and the EMA filters have the same lag and the same noise reduction ratio, as discussed in Sec. 6.1,

$$N = \frac{1 + \lambda}{1 - \lambda} \Rightarrow \lambda = \frac{N-1}{N+1} \Rightarrow \alpha = 1 - \lambda = \frac{2}{N+1} \quad (6.15.3)$$

The TMA filter has length $2N - 1$ and is evidently the convolution of two length- N SMAs. Denoting by y_n the raw data, where n represents the n th trading day (or, weekly, monthly, or quarterly sample periods), we will denote the output of the moving average filters by a_n representing the smoothed *local level* of y_n . The corresponding I/O filtering equations are then,

$$\begin{aligned}
 \text{(SMA)} \quad a_n &= \frac{y_n + y_{n-1} + y_{n-2} + \cdots + y_{n-N+1}}{N} \\
 \text{(WMA)} \quad a_n &= \frac{2}{N(N+1)} \sum_{k=0}^{N-1} (N-k) y_{n-k} \\
 \text{(TMA)} \quad a_n &= \frac{1}{N^2} \sum_{k=0}^{2N-2} (N - |k - N + 1|) y_{n-k} \\
 \text{(EMA)} \quad a_n &= \lambda a_{n-1} + (1 - \lambda) y_n
 \end{aligned}
 \tag{6.15.4}$$

The typical *trading rule* used by traders is to “buy” when a_n is rising and y_n lies above a_n , and to “sell” when a_n is falling and y_n lies below a_n .

Unfortunately, these widely used filters have an inherent lag, which can often result in false buy/sell signals. The basic tradeoff is that longer lengths N result in longer lags, but at the same time, the filters become more effective in smoothing and reducing noise in the data. The noise-reduction capability of any filter is quantified by its “noise-reduction ratio” defined by,

$$\mathcal{R} = \sum_{n=0}^{\infty} h^2(n)
 \tag{6.15.5}$$

with smaller \mathcal{R} corresponding to more effective noise reduction. By construction, the above filters are lowpass filters with unity gain at DC, therefore, satisfying the constraint,

$$\sum_{n=0}^{\infty} h(n) = 1
 \tag{6.15.6}$$

The “lag” is defined as the group delay at DC which, after using Eq. (6.15.6), is given by,

$$\tilde{n} = \sum_{n=0}^{\infty} n h(n)
 \tag{6.15.7}$$

One can easily verify that the noise-reduction ratios and lags of the above filters are:

$$\begin{aligned}
 \text{(SMA)} \quad \mathcal{R} &= \frac{1}{N}, & \tilde{n} &= \frac{N-1}{2} \\
 \text{(WMA)} \quad \mathcal{R} &= \frac{4N+2}{3N(N+1)}, & \tilde{n} &= \frac{N-1}{3} \\
 \text{(TMA)} \quad \mathcal{R} &= \frac{2N^2+1}{3N^3}, & \tilde{n} &= N-1 \\
 \text{(EMA)} \quad \mathcal{R} &= \frac{1-\lambda}{1+\lambda} = \frac{1}{N}, & \tilde{n} &= \frac{\lambda}{1-\lambda} = \frac{N-1}{2}, \text{ for equivalent } N
 \end{aligned}
 \tag{6.15.8}$$

The tradeoff is evident, with \mathcal{R} decreasing and \tilde{n} increasing with N .

We include one more lowpass smoothing filter, the *integrated linear regression slope* (ILRS) filter [307] which is developed in Sec. 6.16. It has unity DC gain and its impulse response, transfer function, lag, and NRR are given by,

$$\begin{aligned}
 \text{(ILRS)} \quad h(n) &= \frac{6(n+1)(N-1-n)}{N(N^2-1)}, \quad n = 0, 1, \dots, N-1 \\
 H(z) &= \frac{6}{N(N^2-1)} \frac{N(1-z^{-1})(1+z^{-N}) - (1-z^{-N})(1+z^{-1})}{(1-z^{-1})^3} \\
 \tilde{n} &= \frac{N-2}{2}, \quad \mathcal{R} = \frac{6(N^2+1)}{5N(N^2-1)}
 \end{aligned}
 \tag{6.15.9}$$

Fig. 6.15.1 compares the frequency responses of the above filters. We note that the ILRS has similar bandwidth as the WMA, but it also has a smaller NRR and more suppressed high-frequency range, thus, resulting in smoother output.

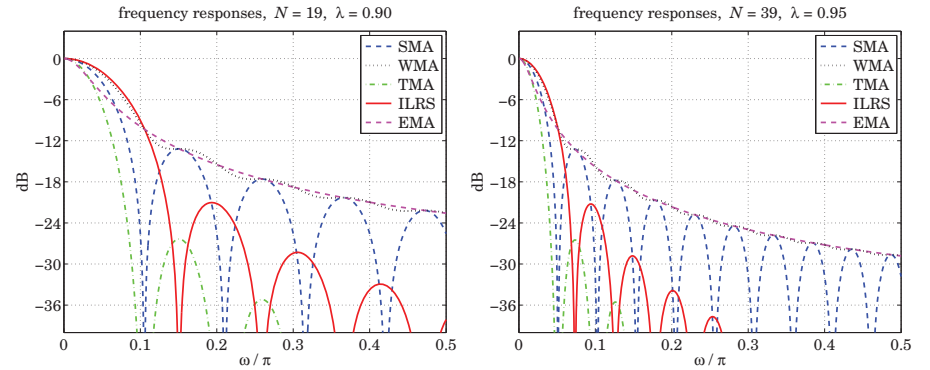


Fig. 6.15.1 Frequency responses of SMA, WMA, TMA, ILRS, and EMA filters.

As a small example, we also give for comparison the impulse responses, $\mathbf{h} = [h_0, h_1, \dots]$, of the SMA, WMA, TMA, and ILRS filters for the case $N = 5$,

$$\begin{aligned}
 \text{(SMA)} \quad \mathbf{h} &= \frac{1}{5} [1, 1, 1, 1, 1] \\
 \text{(WMA)} \quad \mathbf{h} &= \frac{1}{15} [5, 4, 3, 2, 1] \\
 \text{(TMA)} \quad \mathbf{h} &= \frac{1}{25} [1, 2, 3, 4, 5, 4, 3, 2, 1] \\
 \text{(ILRS)} \quad \mathbf{h} &= \frac{1}{10} [2, 3, 3, 2, 0]
 \end{aligned}$$

with the SMA having constant weights, the WMA having linearly decreasing weights, the TMA has triangular weights, and the last coefficient h_{N-1} of the ILRS always being zero.

The following MATLAB functions implement the SMA, WMA, TMA, and ILRS moving averages. The input array y represents the financial data to be filtered.

```
a = sma(y,N,yin); % simple moving average
a = wma(y,N,yin); % weighted moving average
a = tma(y,N,yin); % triangular moving average
a = ilrs(y,N,yin); % integrated linear regression slope
```

The string variable yin specifies the way the filters are initialized and can take on the following values as explained further in Sec. 6.19,

```
yin = 'f', % progressive filtering (default method)
yin = 'n', % initial transients are NaNs (facilitates plotting)
yin = 'c', % standard convolutional transients
```

Some comparisons of these and other moving average indicators are presented in Figures 6.18.2 and 6.21.1.

6.16 Predictive Moving Average Filters

The *predictive FIR* and *double EMA* filters discussed in Sects. 6.4 and 6.8 find application in stock market trading. Their main property is the elimination or shortening of the lag, which is accomplished by tracking both the local level and the local slope of the data. More discussion of these filters and their application in the trading context may be found in Refs. [297-308].

The local-level and local-slope FIR filters $h_a(k)$ and $h_b(k)$ were given in Eq. (6.4.4), and their filtering equations by (6.4.5). They define the following market indicators:

$$\begin{aligned} a_n &= \sum_{k=0}^{N-1} h_a(k) y_{n-k} = \text{linear regression indicator} \\ b_n &= \sum_{k=0}^{N-1} h_b(k) y_{n-k} = \text{linear regression slope indicator} \\ a_n + b_n &= \sum_{k=0}^{N-1} h_1(k) y_{n-k} = \text{time-series forecast indicator} \end{aligned} \quad (6.16.1)$$

where $h_1(k) = h_a(k) + h_b(k)$. The quantity $a_n + b_n$, denoted by $\hat{y}_{n+1/n}$, represents the one-step ahead forecast or prediction to time $n + 1$ based on the data up to time n . More generally, the prediction τ steps ahead from time n is given by the following indicator, which we will refer to as the *predictive moving average* (PMA),

$$\hat{y}_{n+\tau/n} = a_n + \tau b_n = \sum_{k=0}^{N-1} h_\tau(k) y_{n-k} \quad (\text{PMA}) \quad (6.16.2)$$

where, as follows from Eq. (6.4.4), we have for $n = 0, 1, \dots, N - 1$,

$$h_\tau(n) = h_a(n) + \tau h_b(n) = \frac{2(2N - 1 - 3n)}{N(N + 1)} + \tau \frac{6(N - 1 - 2n)}{N(N^2 - 1)} \quad (6.16.3)$$

The time “advance” τ can be non-integer, positive, or negative. Positive τ s correspond to forecasting, negative τ s to delay or lag. In fact, the SMA and WMA are special cases of Eq. (6.16.3) for the particular choices of $\tau = -(N - 1)/2$ and $\tau = -(N - 1)/3$, respectively.

The phrase “linear regression indicators” is justified in Sec. 6.18. The filters $h_\tau(n)$ are very flexible and useful in the trading context, and are actually the *optimal filters* that have *minimum noise-reduction ratio* subject to the two constraints of having unity DC gain and lag equal to $-\tau$, that is, for fixed N , $h_\tau(n)$ is the solution of the optimization problem (for $N = 1$, we ignore the lag constraint to get, $h_\tau(n) = 1$, for $n = 0$, and all τ):

$$\mathcal{R} = \sum_{n=0}^{N-1} h_\tau^2(n) = \min, \quad \text{subject to} \quad \sum_{n=0}^{N-1} h_\tau(n) = 1, \quad \sum_{n=0}^{N-1} n h_\tau(n) = -\tau \quad (6.16.4)$$

This was solved in Sec. 6.4. The noise-reduction-ratio of these filters is,

$$\mathcal{R}_\tau = \sum_{n=0}^{N-1} h_\tau^2(n) = \frac{1}{N} + \frac{3(N - 1 + 2\tau)^2}{N(N^2 - 1)} \quad (6.16.5)$$

We note the two special cases, first for the SMA filter having $\tau = -(N - 1)/2$, and second, for the zero-lag filter $h_a(n)$ having $\tau = 0$,

$$\mathcal{R}_{\text{SMA}} = \frac{1}{N}, \quad \mathcal{R}_a = \frac{4N - 2}{N(N + 1)}$$

The transfer functions of the FIR filters $h_a(n)$, $h_b(n)$ are not particularly illuminating, however, they are given below in rational form,

$$\begin{aligned} H_a(z) &= \sum_{n=0}^{N-1} h_a(n) z^{-n} = \frac{2}{N(N + 1)} \frac{N(1 - z^{-1})(2 + z^{-N}) - (1 + 2z^{-1})(1 - z^{-N})}{(1 - z^{-1})^2} \\ H_b(z) &= \sum_{n=0}^{N-1} h_b(n) z^{-n} = \frac{6}{N(N^2 - 1)} \frac{N(1 - z^{-1})(1 + z^{-N}) - (1 + z^{-1})(1 - z^{-N})}{(1 - z^{-1})^2} \end{aligned}$$

By a proper limiting procedure, one can easily verify the unity-gain and zero-lag properties, $H_a(z)|_{z=1} = 1$, and, $\tilde{n} = -H'_a(z)|_{z=1} = 0$.

The ILRS filter mentioned in the previous section is defined as the *integration*, or cumulative sum, of the slope filter, which can be evaluated explicitly resulting in (6.15.9),

$$h(n) = \sum_{k=0}^n h_b(k) = \sum_{k=0}^n \frac{6(N - 1 - 2k)}{N(N^2 - 1)} = \frac{6(n + 1)(N - 1 - n)}{N(N^2 - 1)} \quad (6.16.6)$$

where $0 \leq n \leq N - 1$. For $n > N$, since $h_b(k)$ has duration N , the above sum remains constant and equal to zero, i.e., equal to its final value,

$$\sum_{k=0}^{N-1} h_b(k) = 0$$

The corresponding transfer function is the integrated (accumulated) form of $H_b(z)$ and is easily verified to be as in Eq. (6.15.9),

$$H(z) = \frac{H_b(z)}{1 - z^{-1}}$$

The following MATLAB function, **pma**, implements Eq. (6.16.2) and the related indicators, where the input array **y** represents the financial data to be filtered. The function, **pmaimp**, implements the impulse response of Eq. (6.16.3).

```

at = pma(y,N,tau,yin);           % at = a + tau*b, prediction distance tau
a = pma(y,N,0,yin);             % local-level indicator
b = pma(y,N,1,yin)-pma(y,N,0,yin); % local-slope indicator
af = pma(y,N,1,yin);            % time-series forecast indicator, af = a + b
ht = pmaimp(N,tau);             % impulse response of predictive filter
ha = pmaimp(N,0);               % impulse response of local level filter
hb = pmaimp(N,1)-pmaimp(N,0);   % impulse response of local slope filter
    
```

and again, the string variable **yin** specifies the way the filters are initialized and can take on the following values,

```

yin = 'f',    % progressive filtering (default method)
yin = 'n',    % initial transients are NaNs (facilitates plotting)
yin = 'c',    % standard convolutional transients
    
```

A few examples of impulse responses are as follows, for $N = 5, 8, 11$,

$$\begin{aligned}
 N = 5, \quad \mathbf{h}_a &= \frac{1}{5}[3, 2, 1, 0, -1] && \text{(local level)} \\
 \mathbf{h}_b &= \frac{1}{10}[2, 1, 0, -1, -2] && \text{(local slope)} \\
 \mathbf{h}_1 &= \frac{1}{10}[8, 5, 2, -1, -4] && \text{(time-series forecast)} \\
 \\
 N = 8, \quad \mathbf{h}_a &= \frac{1}{12}[5, 4, 3, 2, 1, 0, -1, -2] \\
 \mathbf{h}_b &= \frac{1}{84}[7, 5, 3, 1, -1, -3, -5, -7] \\
 \mathbf{h}_1 &= \frac{1}{28}[14, 11, 8, 5, 2, -1, -4, -7] \\
 \\
 N = 11, \quad \mathbf{h}_a &= \frac{1}{22}[7, 6, 5, 4, 3, 2, 1, 0, -1, -2, -3] \\
 \mathbf{h}_b &= \frac{1}{110}[5, 4, 3, 2, 1, 0, -1, -2, -3, -4, -5] \\
 \mathbf{h}_1 &= \frac{1}{55}[20, 17, 14, 11, 8, 5, 2, -1, -4, -7, -10]
 \end{aligned}$$

Some comparisons of PMA with other moving average indicators are shown in Figures 6.18.2 and 6.21.1.

6.17 Single, Double, and Triple EMA Indicators

As discussed in Sec. 6.6, the single EMA (SEMA), double EMA (DEMA), and triple EMA (TEMA) steady-state exponential smoothing recursions are as follows,

$$\begin{aligned}
 e_{n/n-1} &= y_n - \hat{y}_{n/n-1} = y_n - a_{n-1} && \text{(SEMA)} && (6.17.1) \\
 a_n &= a_{n-1} + (1 - \lambda)e_{n/n-1}
 \end{aligned}$$

$$\begin{aligned}
 e_{n/n-1} &= y_n - \hat{y}_{n/n-1} = y_n - (a_{n-1} + b_{n-1}) \\
 \begin{bmatrix} a_n \\ b_n \end{bmatrix} &= \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a_{n-1} \\ b_{n-1} \end{bmatrix} + \begin{bmatrix} 1 - \lambda^2 \\ (1 - \lambda)^2 \end{bmatrix} e_{n/n-1} && \text{(DEMA)} && (6.17.2)
 \end{aligned}$$

$$\begin{aligned}
 e_{n/n-1} &= y_n - \hat{y}_{n/n-1} = y_n - (a_{n-1} + b_{n-1} + c_{n-1}) \\
 \begin{bmatrix} a_n \\ b_n \\ c_n \end{bmatrix} &= \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_{n-1} \\ b_{n-1} \\ c_{n-1} \end{bmatrix} + \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} e_{n/n-1} && \text{(TEMA)} && (6.17.3)
 \end{aligned}$$

where

$$\alpha_1 = 1 - \lambda^3, \quad \alpha_2 = \frac{3}{2}(1 - \lambda)(1 - \lambda^2), \quad \alpha_3 = \frac{1}{2}(1 - \lambda)^3$$

and $\hat{y}_{n/n-1}$ represents the forecast of y_n based on data up to time $n-1$. More generally, the forecast ahead by a distance τ is given by,

$$\begin{aligned}
 \text{(SEMA)} \quad \hat{y}_{n+\tau/n} &= a_n && \hat{y}_{n/n-1} &= a_{n-1} \\
 \text{(DEMA)} \quad \hat{y}_{n+\tau/n} &= a_n + b_n\tau && \Rightarrow \hat{y}_{n/n-1} &= a_{n-1} + b_{n-1} \\
 \text{(TEMA)} \quad \hat{y}_{n+\tau/n} &= a_n + b_n\tau + c_n\tau^2 && \hat{y}_{n/n-1} &= a_{n-1} + b_{n-1} + c_{n-1}
 \end{aligned} \tag{6.17.4}$$

We saw in Sec. 6.8 that an alternative way of computing the local level and local slope signals a_n, b_n in the DEMA case is in terms of the outputs of the cascade of two single EMAs, that is, with $\alpha = 1 - \lambda$,

$$\begin{aligned}
 y_n \rightarrow \text{EMA} \rightarrow a_n^{[1]} \rightarrow \text{EMA} \rightarrow a_n^{[2]} \rightarrow \begin{cases} a_n^{[1]} = \lambda a_{n-1}^{[1]} + \alpha y_n \\ a_n^{[2]} = \lambda a_{n-1}^{[2]} + \alpha a_n^{[1]} \end{cases} && (6.17.5)
 \end{aligned}$$


$$\begin{aligned}
 a_n &= 2a_n^{[1]} - a_n^{[2]} = \text{local level DEMA indicator} \\
 b_n &= \frac{\alpha}{\lambda} (a_n^{[1]} - a_n^{[2]}) = \text{local slope DEMA indicator} && (6.17.6)
 \end{aligned}$$

The transfer functions from y_n to the signals a_n, b_n were given in Eq. (6.8.5), and are expressible as follows in terms of the transfer function of a single EMA, $H(z) = \alpha / (1 - \lambda z^{-1})$,

$$H_a(z) = \frac{\alpha(1 + \lambda - 2\lambda z^{-1})}{(1 - \lambda z^{-1})^2} = 2H(z) - H^2(z) = 1 - [1 - H(z)]^2 \tag{6.17.7}$$

$$H_b(z) = \frac{\alpha^2(1 - z^{-1})}{(1 - \lambda z^{-1})^2} = \frac{\alpha}{\lambda} [H(z) - H^2(z)]$$

Similarly, in the TEMA case, the signals a_n, b_n, c_n can be computed from the outputs of three successive single EMAs via the following relationships,



$$\begin{aligned} a_n^{[1]} &= \lambda a_{n-1}^{[1]} + \alpha y_n \\ a_n^{[2]} &= \lambda a_{n-1}^{[2]} + \alpha a_n^{[1]} \\ a_n^{[3]} &= \lambda a_{n-1}^{[3]} + \alpha a_n^{[2]} \end{aligned} \tag{6.17.8}$$

$$\begin{bmatrix} a_n \\ b_n \\ c_n \end{bmatrix} = \frac{1}{2\lambda^2} \begin{bmatrix} 6\lambda^2 & -6\lambda^2 & 2\lambda^2 \\ \alpha(1 + 5\lambda) & -2\alpha(1 + 4\lambda) & \alpha(1 + 3\lambda) \\ \alpha^2 & -2\alpha^2 & \alpha^2 \end{bmatrix} \begin{bmatrix} a_n^{[1]} \\ a_n^{[2]} \\ a_n^{[3]} \end{bmatrix} \tag{6.17.9}$$

where $\alpha = 1 - \lambda$. See also Eqs. (6.8.9)–(6.8.13). In particular, we have,

$$a_n = 3a_n^{[1]} - 3a_n^{[2]} + a_n^{[3]} \quad (\text{local level TEMA indicator}) \tag{6.17.10}$$

Initialization issues for the single EMA, DEMA, and TEMA recursions are discussed in Sec. 6.19. The following MATLAB functions implement the corresponding filtering operations, where the input array y represents the financial data to be filtered.

```

a = sema(y,N,yin); % single exponential moving average
[a,b,a1,a2] = dema(y,N,yin); % double exponential moving average
[a,b,c,a1,a2,a3] = tema(y,N,yin); % triple exponential moving average
    
```

The variable yin specifies the way the filters are initialized and can take on the following possible values,

```

yin = y(1) % default for SEMA
yin = 'f' % fits polynomial to first N samples, default for DEMA, TEMA
yin = 'c' % cascaded initialization for DEMA, TEMA, described in Sect. 6.19
yin = any vector of initial values of [a], [a;b], or [a;b;c] at n=-1
yin = [0], [0;0], or [0;0;0] for standard convolutional output
    
```

Even though the EMA filters are IIR filters, traders prefer to specify the parameter λ of the EMA recursions through the SMA-equivalent length N defined as in Eq. (6.1.16),

$$\lambda = \frac{N - 1}{N + 1} \Leftrightarrow N = \frac{1 + \lambda}{1 - \lambda} \tag{6.17.11}$$

The use of DEMA and TEMA as market indicators with less lag was first advocated by Mulloy [297,298]. Some comparisons of these with other moving average indicators are shown in Fig. 6.18.2.

6.18 Linear Regression and R-Square Indicators

In the literature of technical analysis, the PMA indicators of Eq. (6.16.1) are usually not implemented as FIR filters, but rather as successive fits of straight lines to the past N data from the current data point, that is, over the time span, $[n - N + 1, n]$, for each n . This is depicted Fig. 6.18.1 below.

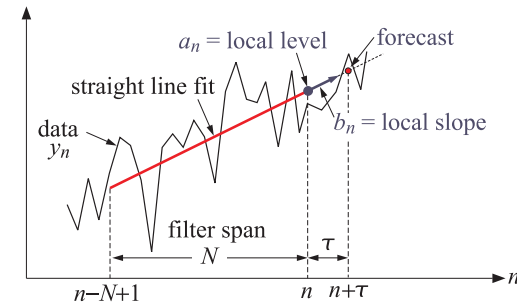


Fig. 6.18.1 Local linear regression and prediction.

They have been rediscovered many times in the past and different names given to them. For example, Lafferty [300] calls them “end-point moving averages”, while Raftar [303] refers to them as “moving trends.” Their application as a forecasting tool was discussed first by Chande [299].

Because of the successive fitting of straight lines, the signals a_n, b_n are known as the “linear regression” indicator and the “linear regression slope” indicator, respectively. The a_n indicator is also known as “least-squares moving average” (LSMA).

For each $n \geq N - 1$, the signals a_n, b_n can be obtained as the *least-squares solution* of the following $N \times 2$ *overdetermined* system of linear equations in two unknowns:

$$a_n - kb_n = y_{n-k}, \quad k = 0, 1, \dots, N - 1 \tag{6.18.1}$$

which express the fitting of a straight line, $a + b\tau$, to the data $[y_{n-N+1}, \dots, y_{n-1}, y_n]$, that is, over the time window, $[n - N + 1, n]$, where a is the intercept at the end of the line. The overdetermined system (6.18.1) can be written compactly in matrix form by defining the length- N column vectors,

$$\mathbf{u} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \\ \vdots \\ 1 \end{bmatrix}, \quad \mathbf{k} = \begin{bmatrix} 0 \\ 1 \\ 2 \\ \vdots \\ k \\ \vdots \\ N - 1 \end{bmatrix}, \quad \mathbf{y}_n = \begin{bmatrix} y_n \\ y_{n-1} \\ y_{n-2} \\ \vdots \\ y_{n-k} \\ \vdots \\ y_{n-N+1} \end{bmatrix} \Rightarrow [\mathbf{u}, -\mathbf{k}] \begin{bmatrix} a_n \\ b_n \end{bmatrix} = \mathbf{y}_n \tag{6.18.2}$$

with the least-squares solution expressed in the following MATLAB-like vectorial notation using the backslash operator,

$$\begin{bmatrix} a_n \\ b_n \end{bmatrix} = [\mathbf{u}, -\mathbf{k}] \setminus \mathbf{y}_n \tag{6.18.3}$$

Indeed, this is the solution for the local level and local slope parameters a, b that minimize the following least-squares performance index, defined for each n ,

$$\mathcal{J}_n = \sum_{k=0}^{N-1} (a - bk - y_{n-k})^2 = \min \tag{6.18.4}$$

In order to account also for the initial transient period, $0 \leq n \leq N - 1$, we may change the upper limit of summation in Eq. (6.18.4) to,

$$\mathcal{J}_n = \sum_{k=0}^{\min(n, N-1)} (a - bk - y_{n-k})^2 = \min \tag{6.18.5}$$

which amounts to fitting a straight line to a progressively longer and longer data vector until its length becomes equal to N , that is, starting with $a_0 = y_0$ and $b_0 = 0$,[†] we fit a line to $[y_0, y_1]$ to get a_1, b_1 , then fit a line to $[y_0, y_1, y_2]$ to get a_2, b_2 , and so on until $n = N - 1$, and beyond that, we continue with a length- N data window.

Thus, we may state the complete solution for all $0 \leq n \leq L - 1$, where L is the length of the data vector y_n , using the backslash notation,[‡]

for each, $n = 0, 1, 2, \dots, L - 1$, do:

$$K_n = \min(n, N - 1) + 1 = \text{fitting length, } K_n = N \text{ when } n \geq N - 1$$

$$\mathbf{k} = [0 : K_n - 1]' = \text{column vector, length } K_n$$

$$\mathbf{y}_n = \mathbf{y}(n - \mathbf{k}) = \text{column vector, } [y_n, y_{n-1}, \dots, y_{n-K_n+1}]^T$$

$$\mathbf{u} = \text{ones}(K_n, 1) = \text{column vector}$$

$$\begin{bmatrix} a_n \\ b_n \end{bmatrix} = [\mathbf{u}, -\mathbf{k}] \setminus \mathbf{y}_n = \text{linear regression indicators}$$

$$R^2(n) = (\text{corr}(-\mathbf{k}, \mathbf{y}_n))^2 = 1 - \det(\text{corrcoef}(-\mathbf{k}, \mathbf{y}_n)) = R^2 \text{ indicator}$$

(6.18.6)

where we also included the so-called *R-square indicator*,* which is the *coefficient of determination* for the linear fit, and quantifies the strength of the linear relationship, that is, higher values of R^2 suggest that the linear fit is statistically significant with a certain degree of confidence (usually taken to be at the 95% confidence level).

The MATLAB function, `r2crit` in the OSP toolbox, calculates the critical values R_c^2 of R^2 for a given N and given confidence level p , such that if $R^2(n) > R_c^2$, then the linear fit is considered to be statistically significant for the n th segment. Some typical critical values of R_c^2 at the $p = 0.95$ and $p = 0.99$ levels are listed below in Eq. (6.18.7), and were computed with the following MATLAB commands (see also [280]),

[†] $b_0 = 0$ is an arbitrary choice since b_0 is indeterminate for $N = 1$.

[‡]the backslash solution also correctly generates the case $n = 0$, i.e., $a_0 = y_0$ and $b_0 = 0$.

*where, `corr`, `det`, and `corrcoef`, are built-in MATLAB functions.

```
N = [5, 10, 14, 20, 25, 30, 50, 60, 120];
R2c = r2crit(N,0.95);
R2c = r2crit(N,0.99);
```

N	$p = 0.95$	$p = 0.99$
5	0.7711	0.9180
10	0.3993	0.5846
14	0.2835	0.4374
20	0.1969	0.3152
25	0.1569	0.2552
30	0.1303	0.2143
50	0.0777	0.1303
60	0.0646	0.1090
120	0.0322	0.0549

(6.18.7)

The *standard errors* for the successive linear fits, as well as the standard errors for the quantities a_n, b_n , can be computed by including the following lines within the for-loop in Eq. (6.18.6),

$$\mathbf{e}_n = \mathbf{y}_n - [\mathbf{u}, -\mathbf{k}] \begin{bmatrix} a_n \\ b_n \end{bmatrix} = \text{fitting error, column vector}$$

$$\sigma_e(n) = \sqrt{\frac{\mathbf{e}_n^T \mathbf{e}_n}{K_n - 2}} = \text{standard error}$$

$$\sigma_a(n) = \sqrt{\frac{2(2K_n - 1)}{K_n(K_n + 1)}} \sigma_e(n) = \text{standard error for } a_n$$

$$\sigma_b(n) = \sqrt{\frac{12}{K_n(K_n^2 - 1)}} \sigma_e(n) = \text{standard error for } b_n$$

(6.18.8)

The derivation of the expressions for $\sigma_e, \sigma_a, \sigma_b$ follows from the standard theory of least-squares linear regression. For example, linear regression based on the K pairs, $(x_k, y_k), k = 0, 1, \dots, K - 1$, results in the estimates, $\hat{y}_k = a + bx_k$, and error residuals, $e_k = y_k - \hat{y}_k$, from which the standard errors can be calculated from the following expressions [349],

$$\sigma_e^2 = \frac{1}{K - 2} \sum_{k=0}^{N-1} e_k^2, \quad \sigma_a^2 = \sigma_e^2 \frac{\bar{x}^2}{K\sigma_x^2}, \quad \sigma_b^2 = \frac{\sigma_e^2}{K\sigma_x^2} \tag{6.18.9}$$

For our special case of equally-spaced data, $x_k = -k$, we easily find,

$$\bar{x} = -\bar{k} = -\frac{1}{K} \sum_{k=0}^{K-1} k = -\frac{K-1}{2}$$

$$\bar{x}^2 = \bar{k}^2 = \frac{1}{K} \sum_{k=0}^{K-1} k^2 = \frac{(K-1)(2K-1)}{6} \Rightarrow$$

$$\sigma_a^2 = \frac{2(2K-1)}{K(K+1)} \sigma_e^2$$

$$\sigma_b^2 = \frac{12}{K(K^2-1)} \sigma_e^2$$

$$\sigma_x^2 = \sigma_k^2 = \bar{k}^2 - \bar{k}^2 = \frac{K^2-1}{12}$$

Standard error bands [329], as well as other types of bands and envelopes, and their use as market indicators, are discussed further in Sec. 6.22. The MATLAB function, `lreg`, implements Eqs. (6.18.6) and (6.18.8) with usage,

```
[a,b,R2,se,sa,sb] = lreg(y,N,init); % linear regression indicators
```

```
y = data          a = local level    se = standard error
N = window length b = local slope    sa = standard error for a
init = initialization R2 = R-square    sb = standard error for b
```

where `init` specifies the initialization scheme and takes on the following values,

```
init = 'f', progressive linear fitting of initial N-1 samples, default
init = 'n', replacing initial N-1 samples of a,b,R2,se,sa,sb by NaNs
```

The local level and local slope outputs a_n, b_n are identical to those produced by the function `pma` of Sec. 6.16.

Generally, these indicators behave similarly to the DEMA indicators, but both indicator types should be used with some caution since they are too quick to respond to price changes and sometimes tend to produce false buy/sell signals. In other words, some delay may miss the onset of a trend but provides more safety.

Example 6.18.1: Fig. 6.18.2 compares the SMA, EMA, WMA, PMA/linear regression, DEMA, TEMA indicators. The data are from [305] and represent daily prices for Nicor-Gas over 130 trading days starting on Sept. 1, 2006. The included excel file, `nicor.xls`, contains the open-high-low-close prices in its first four columns. The left graphs were produced by the following MATLAB code, in which the function, `ohlc`, from the OSP toolbox, makes an OHLC[†] bar chart,

```
Y = xlsread('nicor.xls'); % read Nicor-Gas data
Y = Y(1:130,:); % keep only 130 trading days
y = Y(:,4); % closing prices
t = 0:length(y)-1; % trading days

N = 20; % filter length

figure; % SMA, EMA, WMA
plot(t,sma(y,N),'r-', t,sema(y,N),'k--', t,wma(y,N),'g-'); hold on;
ohlc(t,Y(:,1:4)); % add OHLC bar chart

figure; % PMA/lreg, DEMA, TEMA
plot(t,pma(y,N,0),'r-', t,dema(y,N),'k--', t,tema(y,N),'g-'); hold on;
ohlc(t,Y(:,1:4)); % add OHLC bar chart
```

The filter length was $N = 20$. The right graphs are an expanded view of the range [45, 90] days and show more clearly the reduced lag of the PMA, DEMA, and TEMA indicators. At about the 57th trading day, these indicators turn downwards but still lie above the data, therefore, they would correctly issue a “sell” signal. By contrast, the SMA, EMA, and WMA indicators are rising and lie below the data, and they would issue a “buy” signal.

[†]Open-High-Low-Close

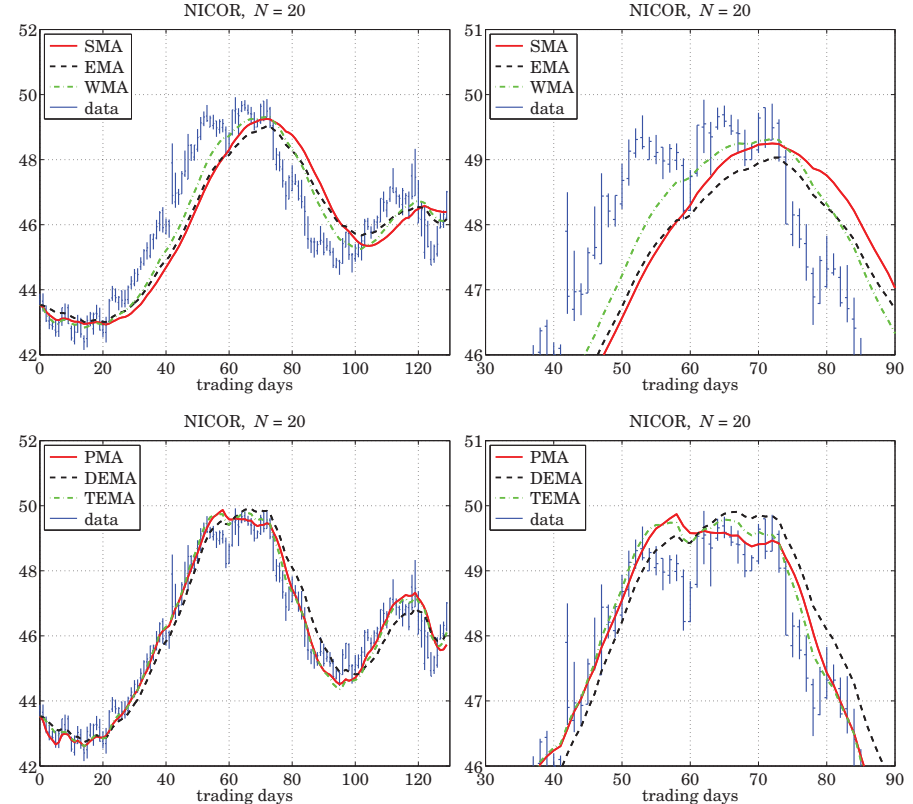


Fig. 6.18.2 Comparison of SMA, EMA, WMA with PMA/LREG, DEMA, TEMA indicators.

Fig. 6.21.1 in Sec. 6.21 compares the PMA with two other indicators of reduced lag, namely, the Hull moving average (HMA), and the exponential Hull moving average (EHMA).

The R -squared and slope indicators are also useful in determining the direction of trend. Fig. 6.18.3 shows the PMA/linear regression indicator, a_n , for the same Nicor data, together with the corresponding $R^2(n)$ signal, and the slope signal b_n , using again a filter length of $N = 20$. They were computed with the MATLAB code:

```
[a,b,R2] = lreg(y,N); % local level, local slope, and R-squared

% equivalent calculation:
% a = pma(y,N,0);
% b = pma(y,N,1)-pma(y,N,0);
```

For $N = 20$, the critical value of R^2 at the 95% confidence level is $R_c^2 = 0.1969$, determined in Eq. (6.18.7), and is displayed as the horizontal dashed line on the R^2 graph.

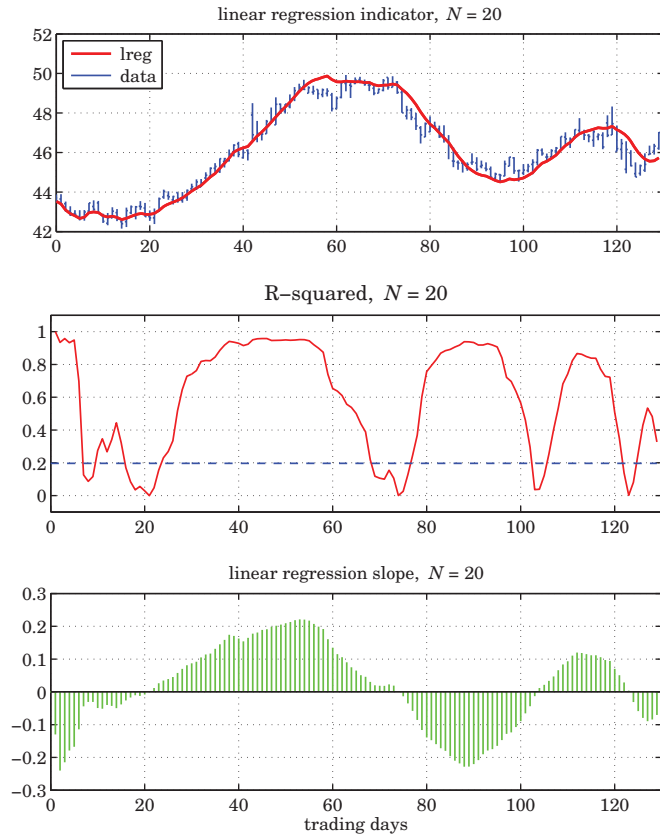


Fig. 6.18.3 PMA/linear regression, R-squared, and linear regression slope indicators.

When $R^2(n)$ is small, below R_c^2 , it indicates lack of a trend with the data moving sideways, and corresponds to slope b_n near zero.

When $R^2(n)$ rises near unity, it indicates a strong trend, but it does not indicate the direction, upwards or downwards. This is indicated by the slope indicator b_n , which is positive when the signal is rising, and negative, when it is falling. More discussion on using these three indicators in conjunction may be found in [305]. □

6.19 Initialization Schemes

In Eq. (6.18.6), one solves a shorter linear fitting problem of progressively increasing length during the transient period, $0 \leq n < N - 1$, and then switches to fixed length N for $n \geq N - 1$.

The same idea can be applied to all FIR filters, such as the SMA, WMA, TMA, and the PMA filter, $h_\tau(n)$, that is, to use the same type of filter, but of progressively increasing length, during the period $0 \leq n < N - 1$, and then switch to using the filters of fixed length N for $n \geq N - 1$. The first $N - 1$ outputs computed in this manner are not the same as the standard convolutional outputs obtained from the built-in function `filter`, because the latter uses the same length- N filter and assumes zero initial internal states.

To clarify this, consider the SMA case with $N = 5$, then the above procedure and the standard convolutional one compute the outputs in the following manner, agreeing only after $n \geq N - 1 = 4$,

progressive	convolutional
$a_0 = y_0$	$a_0 = \frac{1}{5} y_0$
$a_1 = \frac{1}{2} (y_1 + y_0)$	$a_1 = \frac{1}{5} (y_1 + y_0)$
$a_2 = \frac{1}{3} (y_2 + y_1 + y_0)$	$a_2 = \frac{1}{5} (y_2 + y_1 + y_0)$
$a_3 = \frac{1}{4} (y_3 + y_2 + y_1 + y_0)$	$a_3 = \frac{1}{5} (y_3 + y_2 + y_1 + y_0)$
$a_4 = \frac{1}{5} (y_4 + y_3 + y_2 + y_1 + y_0)$	$a_4 = \frac{1}{5} (y_4 + y_3 + y_2 + y_1 + y_0)$
$a_5 = \frac{1}{5} (y_5 + y_4 + y_3 + y_2 + y_1)$	$a_5 = \frac{1}{5} (y_5 + y_4 + y_3 + y_2 + y_1)$
...	...

Similarly, the local level PMA filters, \mathbf{h}_a , of lengths up to $N = 5$ can be determined from Eq. (6.16.3), leading to the following progressive initializations,

$$\begin{aligned}
 N = 1, \quad \mathbf{h}_a &= [1], & a_0 &= y_0 \\
 N = 2, \quad \mathbf{h}_a &= [1, 0], & a_1 &= y_1 \\
 N = 3, \quad \mathbf{h}_a &= \frac{1}{6} [5, 2, -1], & a_2 &= \frac{1}{6} (5y_2 + 2y_1 - y_0) \\
 N = 4, \quad \mathbf{h}_a &= \frac{1}{10} [7, 4, 1, -2], & a_3 &= \frac{1}{10} (7y_3 + 4y_2 + y_1 - 2y_0) \\
 N = 5, \quad \mathbf{h}_a &= \frac{1}{5} [3, 2, 1, 0, -1], & a_4 &= \frac{1}{5} (3y_4 + 2y_3 + y_2 - y_0)
 \end{aligned}$$

and for the local slope filters \mathbf{h}_b ,

$$\begin{aligned}
 N = 1, \quad \mathbf{h}_b &= [0], & b_0 &= 0 \\
 N = 2, \quad \mathbf{h}_b &= [1, -1], & b_1 &= y_1 - y_0 \\
 N = 3, \quad \mathbf{h}_b &= \frac{1}{2} [1, 0, -1], & b_2 &= \frac{1}{2} (y_2 - y_0) \\
 N = 4, \quad \mathbf{h}_b &= \frac{1}{10} [3, 1, -1, -3], & b_3 &= \frac{1}{10} (3y_3 + y_2 - y_1 - 3y_0) \\
 N = 5, \quad \mathbf{h}_b &= \frac{1}{10} [2, 1, 0, -1, -2], & b_4 &= \frac{1}{10} (2y_4 + y_3 - y_1 - 2y_0)
 \end{aligned}$$

where, we arbitrarily set $\mathbf{h}_b = [0]$ for the case $N = 1$, since the slope is meaningless for a single data point. To see the equivalence of these with the least-square criterion of Eq. (6.18.5) consider, for example, the case $N = 5$ and $n = 2$,

$$\mathcal{J}_2 = (a - y_2)^2 + (a - b - y_1)^2 + (a - 2b - y_0)^2 = \min$$

with minimization conditions,

$$\begin{aligned} \frac{\partial \mathcal{J}_2}{\partial a} &= 2(a - y_2) + 2(a - b - y_1) + 2(a - 2b - y_0) = 0 & \Rightarrow & \quad 3a - 3b = y_2 + y_1 + y_0 \\ \frac{\partial \mathcal{J}_2}{\partial b} &= -2(a - b - y_1) - 4(a - 2b - y_0) = 0 & \Rightarrow & \quad 3a - 5b = y_1 + 2y_0 \end{aligned}$$

resulting in the solution,

$$a = \frac{1}{6}(5y_2 + 2y_1 - y_0), \quad b = \frac{1}{2}(y_2 - y_0)$$

Similarly we have for the cases $n = 0$ and $n = 1$,

$$\begin{aligned} \mathcal{J}_0 &= (a - y_0)^2 = \min & \Rightarrow & \quad a = y_0, \quad b = \text{indeterminate} \\ \mathcal{J}_1 &= (a - y_1)^2 + (a - b - y_0)^2 = \min & \Rightarrow & \quad a = y_1, \quad b = y_1 - y_0 \end{aligned}$$

EMA Initializations

The single, double, and triple EMA difference equations (6.17.1)–(6.17.3), also need to be properly initialized at $n = -1$. For the single EMA case, a good choice is $a_{-1} = y_0$, which leads to the same value at $n = 0$, that is,

$$a_0 = \lambda a_{-1} + \alpha y_0 = \lambda y_0 + \alpha y_0 = y_0 \quad (6.19.1)$$

This is the default initialization for our function, **sema**. Another possibility is to choose the mean of the first N data samples, $a_{-1} = \text{mean}([y_0, y_1, \dots, y_{N-1}])$.

For DEMA, if we initialize both the first and the second EMAs as in Eq. (6.19.1), then we must choose, $a_{-1}^{[1]} = y_0$, which leads to $a_0^{[1]} = y_0$, which then would require that, $a_{-1}^{[2]} = a_0^{[1]} = y_0$, thus, in this scheme, we would choose,

$$\begin{bmatrix} a_{-1}^{[1]} \\ a_{-1}^{[2]} \end{bmatrix} = \begin{bmatrix} y_0 \\ y_0 \end{bmatrix} \Rightarrow \begin{bmatrix} a_{-1} \\ b_{-1} \end{bmatrix} = \begin{bmatrix} 2 & -1 \\ \alpha/\lambda & -\alpha/\lambda \end{bmatrix} \begin{bmatrix} a_{-1}^{[1]} \\ a_{-1}^{[2]} \end{bmatrix} = \begin{bmatrix} y_0 \\ 0 \end{bmatrix} \quad (6.19.2)$$

This is the default initialization method for our function, **dema**. Another possibility is to fit a straight line to a few initial data [297,348], such as the first N data, where N is the equivalent SMA length, $N = (1 + \lambda)/(1 - \lambda)$, and then extrapolate the line backwards to $n = -1$. This can be accomplished in MATLAB-like notation as follows,

$$\begin{aligned} \mathbf{n} &= [1 : N]' = \text{column vector} \\ \mathbf{y} &= [y_0, y_1, \dots, y_{N-1}]' = \text{column vector} \\ \mathbf{u} &= \text{ones}(\text{size}(\mathbf{n})) \\ \begin{bmatrix} a_{-1} \\ b_{-1} \end{bmatrix} &= [\mathbf{u}, \mathbf{n}] \setminus \mathbf{y} \end{aligned} \quad (6.19.3)$$

6.19. Initialization Schemes

If one wishes to use the cascade of two EMAs, then the EMA signals, $a_n^{[1]}, a_n^{[2]}$, must be initialized by first applying Eq. (6.19.3), and then using the inverse matrix relationship of Eq. (6.17.6), i.e.,

$$\begin{bmatrix} a_{-1}^{[1]} \\ a_{-1}^{[2]} \end{bmatrix} = \begin{bmatrix} 1 & -\lambda/\alpha \\ 1 & -2\lambda/\alpha \end{bmatrix} \begin{bmatrix} a_{-1} \\ b_{-1} \end{bmatrix} \quad (6.19.4)$$

A third possibility [280] is to initialize the first EMA with $a_{-1}^{[1]} = y_0$, then calculate the output at the time instant $n = N - 1$ and use it to initialize the second EMA at $n = N$, that is, define $a_{N-1}^{[2]} = a_{N-1}^{[1]}$. This value can be iterated backwards to $n = -1$ to determine the proper initial value $a_{-1}^{[2]}$ such that, if iterated forward, it would arrive at the chosen value $a_{N-1}^{[2]} = a_{N-1}^{[1]}$. Thus, the steps in this scheme are as follows,

$$\begin{aligned} a_{-1}^{[1]} &= y_0 & a_{N-1}^{[2]} &= a_{N-1}^{[1]} \\ \text{for } n &= 0, 1, \dots, N-1, & \Rightarrow & \quad \text{for } n = N-1, \dots, 1, 0, \\ a_n^{[1]} &= \lambda a_{n-1}^{[1]} + \alpha y_n & & \quad a_n^{[2]} = \frac{1}{\lambda}(a_n^{[2]} - \alpha a_n^{[1]}) \\ \text{end} & & & \quad \text{end} \end{aligned} \quad (6.19.5)$$

Upon exit from the second loop, one has $a_{-1}^{[2]}$, then, one can transform the calculated $a_{-1}^{[1]}, a_{-1}^{[2]}$ to the a_n, b_n basis in order to get the DEMA recursion started,

$$\begin{bmatrix} a_{-1} \\ b_{-1} \end{bmatrix} = \begin{bmatrix} 2 & -1 \\ \alpha/\lambda & -\alpha/\lambda \end{bmatrix} \begin{bmatrix} a_{-1}^{[1]} \\ a_{-1}^{[2]} \end{bmatrix}$$

Such cascaded initialization scheme for DEMA (and TEMA below) is somewhat ad hoc since the EMA filters are IIR and there is nothing special about the time $n = N$; one, could just as well wait until about $n = 6N$ when typically all transient effects have disappeared. We have found that the schemes described in Eqs. (6.19.2) and (6.19.3) work the best.

Finally, we note that for ordinary convolutional output, one would choose zero initial values,

$$\begin{bmatrix} a_{-1} \\ b_{-1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} a_{-1}^{[1]} \\ a_{-1}^{[2]} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

All of the above initialization choices are incorporated in the function, **dema**. For TEMA, the default initialization is similar to that of Eq. (6.19.2), that is,

$$\begin{bmatrix} a_{-1}^{[1]} \\ a_{-1}^{[2]} \\ a_{-1}^{[3]} \end{bmatrix} = \begin{bmatrix} y_0 \\ y_0 \\ y_0 \end{bmatrix} \Rightarrow \begin{bmatrix} a_{-1} \\ b_{-1} \\ c_{-1} \end{bmatrix} = \begin{bmatrix} y_0 \\ 0 \\ 0 \end{bmatrix} \quad (6.19.6)$$

Alternatively, one can fit a second-order polynomial to the first few data samples, such as the first $2N$ samples [297], and extrapolate them back to $n = -1$. The fitting can be done with the following MATLAB-like code,

$$\begin{aligned} \mathbf{n} &= [1 : 2N - 1]' = \text{column vector} \\ \mathbf{y} &= [y_0, y_1, \dots, y_{2N-1}]' = \text{column vector} \\ \mathbf{u} &= \text{ones}(\text{size}(\mathbf{n})) \\ \begin{bmatrix} a_{-1} \\ b_{-1} \\ c_{-1} \end{bmatrix} &= [\mathbf{u}, \mathbf{n}, \mathbf{n}^2] \setminus \mathbf{y} \end{aligned}$$

The cascaded initialization scheme is also possible in which the output of the first EMA at time $n = N - 1$ serves to initialize the second EMA at $n = N$, and the output of the second EMA at $n = 2N - 1$ serves to initialize the third EMA at $n = 2N$. This, as well as the second-order polynomial fitting initialization schemes are incorporated in the function, **tema**.

A special case of the EMA indicator is “Wilder’s Exponential Moving Average” [281], known as WEMA. It is used widely and appears in several other indicators, such as the “Relative Strength Index” (RSI), the “Average True Range” (ATR), and the “Directional Movement System” (\pm DMI and ADX), discussed in Sec. 6.23. An N -point WEMA is defined to be an ordinary EMA with λ, α parameters,

$$\alpha = \frac{1}{N}, \quad \lambda = 1 - \alpha = 1 - \frac{1}{N} \quad (\text{WEMA parameters}) \quad (6.19.7)$$

It is equivalent to an EMA with effective length, N_e , determined as follows,

$$\lambda = \frac{N_e - 1}{N_e + 1} = 1 - \frac{1}{N} \Rightarrow N_e = 2N - 1 \quad (6.19.8)$$

The corresponding filtering equation for calculating the smoothed local-level signal a_n from the input data y_n , will be,

$$a_n = \lambda a_{n-1} + \alpha y_n = a_{n-1} + \alpha (y_n - a_{n-1})$$

or, for $n \geq 0$,

$$a_n = a_{n-1} + \frac{1}{N} (y_n - a_{n-1}) \quad (\text{WEMA}) \quad (6.19.9)$$

The required initial value a_{-1} can be chosen in a variety of ways, just as in EMA. However, by convention [281], the default way of fixing it is similar to that in Eq. (6.19.5). It is defined by choosing the value of a_n at time $n = N - 1$ to be the mean of first N input values, then, a_{N-1} is back-tracked to time $n = -1$, thus, we have,

$$\begin{aligned} a_{N-1} &= \frac{1}{N} (y_0 + y_1 + \dots + y_{N-1}) \\ \text{for } n &= N-1, \dots, 1, 0, \\ a_{n-1} &= \frac{1}{\lambda} (a_n - \alpha y_n) \\ \text{end} \end{aligned} \quad (6.19.10)$$

Upon exit from the loop, one has the proper starting value of a_{-1} . The following MATLAB function, **wema**, implements WEMA with such default initialization scheme,

```
a = wema(y,N,ain); % Wilder's EMA

y = signal to be smoothed
N = effective length, (EMA alpha = 1/N, lambda = 1-1/N)
ain = any initial value
     = 'm', default, as in Eq.(6.19.10)
     = 0, for standard convolutional output

a = smoothed version of y
```

6.20 Butterworth Moving Average Filters

Butterworth moving average (BMA) lowpass filters, are useful alternatives [285] to the first-order EMA filters, and have comparable smoothing properties and shorter lag. Here, we summarize their properties and filtering implementation, give explicit design equations for orders $M = 1, 2, 3$, and derive a general expression for their lag.

Digital Butterworth filters are characterized by two parameters, the filter order M , and the 3-dB cutoff frequency f_0 in Hz, or, the corresponding digital frequency in units of radians per sample, $\omega_0 = 2\pi f_0 / f_s$, where f_s is the sampling rate in Hz. We may also define the period of f_0 in units of samples/cycle, $N = f_s / f_0$, so that, $\omega_0 = 2\pi / N$.

We follow the design method of Ref. [30] based on the bilinear transformation, although the matched z-transform method has also been used [285]. If the filter order is even, say, $M = 2K$, then, there are K second-order sections, and if it is odd, $M = 2K + 1$, there is an additional first-order section. Both cases can be combined into one by writing,

$$M = 2K + r, \quad r = 0, 1 \quad (6.20.1)$$

Then, the transfer function can be expressed in the following cascaded and direct forms,

$$\begin{aligned} H(z) &= \left[\frac{G_0(1+z^{-1})}{1+a_{01}z^{-1}} \right]^r \prod_{i=1}^K \left[\frac{G_i(1+z^{-1})^2}{1+a_{i1}z^{-1}+a_{i2}z^{-2}} \right] \\ &= \frac{G(1+z^{-1})^M}{1+a_1z^{-1}+a_2z^{-2}+\dots+a_Mz^{-M}} \end{aligned} \quad (6.20.2)$$

where the notation $[]^r$ means that the first-order factor is absent if $r = 0$ and present if $r = 1$. The corresponding first-order coefficients are,

$$G_0 = \frac{\Omega_0}{\Omega_0 + 1}, \quad a_{01} = \frac{\Omega_0 - 1}{\Omega_0 + 1} \quad (6.20.3)$$

The second-order coefficients are, for $i = 1, 2, \dots, K$,

$$G_i = \frac{\Omega_0^2}{1 - 2\Omega_0 \cos \theta_i + \Omega_0^2} \quad (6.20.4)$$

$$a_{i1} = \frac{2(\Omega_0^2 - 1)}{1 - 2\Omega_0 \cos \theta_i + \Omega_0^2}, \quad a_{i2} = \frac{1 + 2\Omega_0 \cos \theta_i + \Omega_0^2}{1 - 2\Omega_0 \cos \theta_i + \Omega_0^2}$$

where the angles θ_i are defined by,

$$\theta_i = \frac{\pi}{2M} (M - 1 + 2i), \quad i = 1, 2, \dots, K \quad (6.20.5)$$

and the quantity Ω_0 is the equivalent analog 3-dB frequency defined as,

$$\Omega_0 = \tan\left(\frac{\omega_0}{2}\right) = \tan\left(\frac{\pi f_0}{f_s}\right) = \tan\left(\frac{\pi}{N}\right) \quad (6.20.6)$$

We note that the filter sections have zeros at $z = -1$, that is, at the Nyquist frequency, $f = f_s/2$, or, $\omega = \pi$. Setting $\Omega = \tan(\omega/2)$, the magnitude response of the designed digital filter can be expressed simply as follows:

$$|H(\omega)|^2 = \frac{1}{1 + (\Omega/\Omega_0)^{2M}} = \frac{1}{1 + (\tan(\omega/2)/\Omega_0)^{2M}} \quad (6.20.7)$$

Each section has unity gain at DC. Indeed, setting $z = 1$ in Eq. (6.20.2), we obtain the following condition, which can be verified from the given definitions,

$$\frac{4G_i}{1 + a_{i1} + a_{i2}} = 1 \quad \text{and} \quad \frac{2G_0}{1 + a_{01}} = 1$$

Moreover, the filter lag can be shown to be (cf. Problem 6.12), for any $M \geq 1$ and $N > 2$,

$$\tilde{n} = \frac{1}{2\Omega_0 \sin\left(\frac{\pi}{2M}\right)} = \frac{1}{2 \tan\left(\frac{\pi}{N}\right) \sin\left(\frac{\pi}{2M}\right)} \quad (\text{lag}) \quad (6.20.8)$$

For $M \geq 2$ and $N \geq 5$, it can be approximated well by [284],

$$\tilde{n} = \frac{MN}{\pi^2}$$

The overall numerator gain in the direct form is the product of gains,

$$G = G_0^r G_1 G_2 \cdots G_K$$

and the direct-form numerator coefficients are the coefficients of the binomial expansion of $(1 + z^{-1})^M$ times the overall gain G . The direct-form denominator coefficients are obtained by convolving the coefficients of the individual sections, that is, setting, $\mathbf{a} = [1]$ if M is even, and, $\mathbf{a} = [1, a_{01}]$ if M is odd, then the vector, $\mathbf{a} = [1, a_1, a_2, \dots, a_M]$, can be constructed recursively by,

$$\text{for } i = 1, 2, \dots, K \quad (6.20.9)$$

$$\mathbf{a} = \text{conv}(\mathbf{a}, [1, a_{i1}, a_{i2}])$$

For example, we have,

$$M = 2, \quad \mathbf{a} = [1, a_{11}, a_{12}]$$

$$M = 3, \quad \mathbf{a} = \text{conv}([1, a_{01}], [1, a_{11}, a_{12}]) = [1, a_{01} + a_{11}, a_{12} + a_{01}a_{11}, a_{01}a_{12}]$$

From these, we obtain the following explicit expressions, for $M = 2$,

$$G = \frac{\Omega_0^2}{\Omega_0^2 + \sqrt{2}\Omega_0 + 1}, \quad a_1 = \frac{2(\Omega_0^2 - 1)}{\Omega_0^2 + \sqrt{2}\Omega_0 + 1}, \quad a_2 = \frac{\Omega_0^2 - \sqrt{2}\Omega_0 + 1}{\Omega_0^2 + \sqrt{2}\Omega_0 + 1} \quad (6.20.10)$$

$$H(z) = \frac{G(1 + 2z^{-1} + z^{-2})}{1 + a_1z^{-1} + a_2z^{-2}}, \quad \tilde{n} = \frac{1}{\sqrt{2}\Omega_0}$$

and, for $M = 3$,

$$G = \frac{\Omega_0^3}{(\Omega_0 + 1)(\Omega_0^2 + \Omega_0 + 1)}, \quad a_1 = \frac{(\Omega_0 - 1)(3\Omega_0^2 + 5\Omega_0 + 3)}{(\Omega_0 + 1)(\Omega_0^2 + \Omega_0 + 1)}$$

$$a_2 = \frac{3\Omega_0^2 - 5\Omega_0 + 3}{\Omega_0^2 + \Omega_0 + 1}, \quad a_3 = \frac{(\Omega_0 - 1)(\Omega_0^2 - \Omega_0 + 1)}{(\Omega_0 + 1)(\Omega_0^2 + \Omega_0 + 1)} \quad (6.20.11)$$

$$H(z) = \frac{G(1 + 3z^{-1} + 3z^{-2} + z^{-3})}{1 + a_1z^{-1} + a_2z^{-2} + a_3z^{-3}}, \quad \tilde{n} = \frac{1}{\Omega_0}$$

We note also that the $M = 1$ case has lag, $\tilde{n} = 1/(2\Omega_0)$, and is equivalent to the modified EMA of Eq. (2.3.5). This can be seen by rewriting $H(z)$ in the form,

$$H(z) = \frac{G_0(1 + z^{-1})}{1 + a_{01}z^{-1}} = \frac{\frac{1}{2}(1 - \lambda)(1 + z^{-1})}{1 - \lambda z^{-1}}, \quad \lambda = -a_{01} = \frac{1 - \Omega_0}{1 + \Omega_0}$$

where $0 < \lambda < 1$ for $\Omega_0 < 1$, which requires $N > 4$.

The MATLAB function, **bma**, implements the design and filtering operations for any filter order M and any period $N > 2$,[†] with usage,

```
[y,nlag,b,a] = bma(x,N,M,yin); % Butterworth moving average
[y,nlag,b,a] = bma(x,N,M);
```

where

```
x = input signal
N = 3-dB period, need not be integer, but N>2
M = filter order
yin = any Mx1 vector of initial values of the output y
      default, yin = repmat(x(1),M,1)
      yin = 'c' for standard convolutional output

y = output signal
nlag = filter lag
b = [b0, b1, b2, ..., bM], numerator filter coefficients
a = [1, a1, a2, ..., aM], denominator filter coefficients
```

[†] the sampling theorem requires, $f_0 < f_s/2$, or, $N = f_s/f_0 > 2$

Fig. 6.20.1 shows the BMA output for Butterworth orders $M = 2, 3$ applied to the same Nicor-Gas data of Fig. 6.18.2. It has noticeably shorter lag than SMA. The graphs were produced by the MATLAB code,

```

Y = xlsread('nicor.xls'); % load Nicor-Gas data
Y = Y(1:130,1:4); % 130 trading days, and [O,H,L,C] prices
y = Y(:,4); % closing prices
t = 0:length(y)-1; % trading days

N = 20; % period, SMA lag = (N-1)/2 = 9.50
[y2,n2] = bma(y,N,2); % order-2 Butterworth, lag n2 = 4.46
[y3,n3] = bma(y,N,3); % order-3 Butterworth, lag n3 = 6.31

figure; plot(t,sma(y,N), t,y2, t,y3); % plot SMA, y2, y3
hold on; ohlc(t,Y,'color','b'); % add OHLC bar chart
    
```

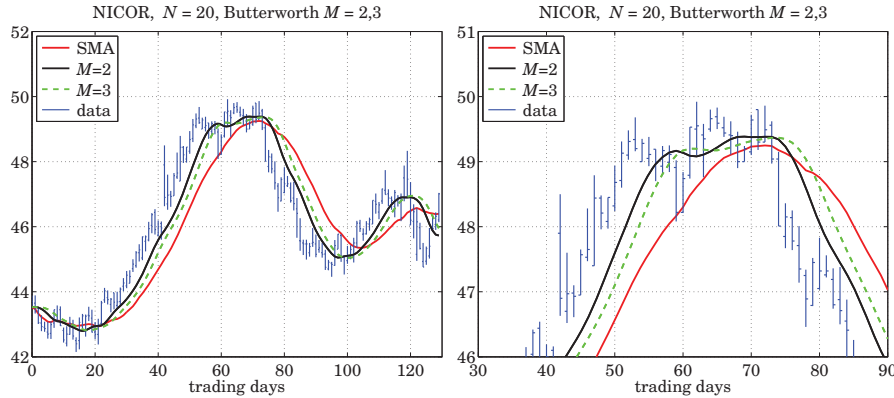


Fig. 6.20.1 Comparison of SMA and Butterworth filters of orders $M = 2, 3$.

6.21 Moving Average Filters with Reduced Lag

The PMA/linear regression and the DEMA/TEMA indicators have zero lag by design. There are other useful indicators that are easily implemented and have zero or very much reduced lag. Examples are twicing and Kaiser-Hamming (KH) filter sharpening [60], the Hull moving average (HMA) [309], the zero-lag EMA indicator (ZEMA) [284], the generalized DEMA (GDEMA) [307], and their variants. Here, we discuss a general procedure for constructing such reduced-lag filters, including the corresponding local-slope filters.

Consider three lowpass filters $H_1(z), H_2(z), H_3(z)$ with unity DC gains and lags, $\tilde{n}_1, \tilde{n}_2, \tilde{n}_3$, respectively, and define the following filters for estimating the local level and local slope of the data, generalizing the twicing operations of Eq. (6.10.9),

6.21. Moving Average Filters with Reduced Lag

$$\begin{aligned}
 H_a(z) &= H_1(z) [(1 + \nu)H_2(z) - \nu H_3(z)] = \text{local level} \\
 H_b(z) &= \frac{1}{\tilde{n}_3 - \tilde{n}_2} H_1(z) [H_2(z) - H_3(z)] = \text{local slope}
 \end{aligned}
 \tag{6.21.1}$$

where ν is a positive constant. One may view $H_a(z)$ as the smoothed, by $H_1(z)$, version of $(1 + \nu)H_2(z) - \nu H_3(z)$. The filter $H_a(z)$ will still have unity DC gain as follows by evaluating Eq. (6.21.1) at $z = 1$, that is,

$$H_a(1) = (1 + \nu)H_1(1)H_2(1) - \nu H_1(1)H_3(1) = (1 + \nu) - \nu = 1$$

Using the fact that the lag of a product of filters is the sum of the corresponding lags (cf. Problem 6.2), we find that the lag of $H_a(z)$ is,

$$\tilde{n}_a = (1 + \nu)(\tilde{n}_1 + \tilde{n}_2) - \nu(\tilde{n}_1 + \tilde{n}_3), \quad \text{or,}$$

$$\tilde{n}_a = \tilde{n}_1 + (1 + \nu)\tilde{n}_2 - \nu\tilde{n}_3 \tag{6.21.2}$$

By appropriately choosing the parameters $\nu, \tilde{n}_1, \tilde{n}_2, \tilde{n}_3$, the lag \tilde{n}_a can be made very small, even zero. Indeed, the following choice for ν will generate any particular \tilde{n}_a ,

$$\nu = \frac{\tilde{n}_1 + \tilde{n}_2 - \tilde{n}_a}{\tilde{n}_3 - \tilde{n}_2} \tag{6.21.3}$$

Below we list a number of examples that are special cases of the above constructions. In this list, the filter $H(z)$, whose lag is denoted by \tilde{n} , represents any unity-gain lowpass filter, such as WMA, EMA, or SMA and similarly, $H_N(z)$ represents either a length- N FIR filter such as WMA or SMA, or an EMA filter with SMA-equivalent length N . Such filters have a lag related to N via a relationship of the form, $\tilde{n} = r \cdot (N - 1)$, for example, $r = 1/3$, for WMA, and, $r = 1/2$, for EMA and SMA.

	reduced-lag filters	lag
(twicing)	$H_a(z) = 2H(z) - H^2(z)$,	$\tilde{n}_a = 0$
(GDEMA)	$H_a(z) = (1 + \nu)H(z) - \nu H^2(z)$,	$\tilde{n}_a = (1 - \nu)\tilde{n}$
(KH)	$H_a(z) = (1 + \nu)H^2(z) - \nu H^3(z)$,	$\tilde{n}_a = (2 - \nu)\tilde{n}$
(HMA)	$H_a(z) = H_{\sqrt{N}}(z) [2H_{N/2}(z) - H_N(z)]$,	$\tilde{n}_a = r[\sqrt{N} - 2]$
(ZEMA)	$H_a(z) = 2H(z) - z^{-d}H(z)$,	$\tilde{n}_a = \tilde{n} - d$
(ZEMA)	$H_a(z) = (1 + \nu)H(z) - \nu z^{-d}H(z)$,	$\tilde{n}_a = \tilde{n} - \nu d$

The corresponding local-slope filters are as follows (they do not depend on ν),

local-slope filters	
(DEMA/GDEMA)	$H_b(z) = \frac{1}{\tilde{n}} [H(z) - H^2(z)]$
(KH)	$H_b(z) = \frac{1}{\tilde{n}} [H^2(z) - H^3(z)]$
(HMA)	$H_b(z) = \frac{2}{rN} H_{\sqrt{N}}(z) [H_{N/2}(z) - H_N(z)]$
(ZEMA)	$H_b(z) = \frac{1}{d} [H(z) - z^{-d}H(z)]$

(6.21.5)

The standard twicing method, $H_a(z) = 2H(z) - H^2(z)$, coincides with DEMA if we choose $H(z)$ to be a single EMA filter,

$$H_{EMA}(z) = \frac{\alpha}{1 - \lambda z^{-1}}, \quad \alpha = 1 - \lambda, \quad \lambda = \frac{N-1}{N+1}, \quad \tilde{n} = \frac{N-1}{2} \tag{6.21.6}$$

but the filter $H(z)$ can also be chosen to be an SMA, WMA, or BMA filter, leading to what may be called, “double SMA,” or, “double WMA,” or, ‘double BMA.’”

The generalized DEMA, $H_{GDEMA}(z) = (1 + \nu)H(z) - \nu H^2(z)$, also has, $H = H_{EMA}$, and is usually operated in practice with $\nu = 0.7$. It reduces to standard DEMA for $\nu = 1$. The so-called Tillson’s *T3 indicator* [307] is obtained by cascading GDEMA three times,

$$H_{T3}(z) = [H_{GDEMA}(z)]^3 \quad (\text{T3 indicator}) \tag{6.21.7}$$

The Kaiser-Hamming (KH) filter sharpening case is not currently used as an indicator, but it has equally smooth output as GDEMA and T3. It reduces to the standard filter sharpening case with zero lag for $\nu = 2$.

In the original Hull moving average [309], $H_a(z) = H_{\sqrt{N}}(z) [2H_{N/2}(z) - H_N(z)]$, the filter H_N is chosen to be a length- N weighted moving average (WMA) filter, as defined in Eqs. (6.15.1) and (6.15.2), and similarly, $H_{N/2}$ and $H_{\sqrt{N}}$ are WMA filters of lengths $N/2$ and \sqrt{N} respectively. Assuming for the moment that these filter lengths are integers, then the corresponding lags of the three WMA filters, $H_{\sqrt{N}}, H_{N/2}, H_N$, will be,

$$\tilde{n}_1 = \frac{\sqrt{N}-1}{3}, \quad \tilde{n}_2 = \frac{N/2-1}{3}, \quad \tilde{n}_3 = \frac{N-1}{3},$$

and setting $\nu = 1$ in Eq. (6.21.2), we find,

$$\tilde{n}_a = \tilde{n}_1 + 2\tilde{n}_2 - \tilde{n}_3 = \frac{\sqrt{N}-1}{3} + \frac{N-2}{3} - \frac{N-1}{3} = \frac{\sqrt{N}-2}{3} \tag{6.21.8}$$

Thus, for larger N s, the lag is effectively reduced by a factor of \sqrt{N} . The extra filter factor $H_{\sqrt{N}}(z)$ provides some additional smoothing. In practice, the filter lengths $N_1 = \sqrt{N}$ and $N_2 = N/2$ are replaced by their rounded values. This changes the lag \tilde{n}_a somewhat. If one wishes to maintain the same lag as that given by Eq. (6.21.8), then one can compensate for the replacement of N_1, N_2 by their rounded values by using a

slightly different value for ν . It is straightforward to show that the following procedure will generate the desired lag value, where the required ν is evaluated from Eq. (6.21.3),

$$\begin{aligned} N_1 &= \text{round}(\sqrt{N}), \quad \varepsilon_1 = N_1 - \sqrt{N} = \text{rounding error} \\ N_2 &= \text{round}\left(\frac{N}{2}\right), \quad \varepsilon_2 = N_2 - \frac{N}{2} = \text{rounding error} \\ \nu &= \frac{\tilde{n}_1 + \tilde{n}_2 - \tilde{n}_a}{\tilde{n}_3 - \tilde{n}_2} = \frac{N/2 + \varepsilon_1 + \varepsilon_2}{N/2 - \varepsilon_2} \\ \tilde{n}_a &= \frac{N_1 - 1}{3} + (1 + \nu) \frac{N_2 - 1}{3} - \nu \frac{N - 1}{3} = \frac{\sqrt{N} - 2}{3} \\ \tilde{n}_3 - \tilde{n}_2 &= \frac{N - N_2}{3} \end{aligned} \tag{6.21.9}$$

with transfer functions,

$$\begin{aligned} H_a(z) &= H_{N_1}(z) [(1 + \nu)H_{N_2}(z) - \nu H_N(z)] = \text{local level} \\ H_b(z) &= \frac{1}{\tilde{n}_3 - \tilde{n}_2} H_{N_1}(z) [H_{N_2}(z) - H_N(z)] = \text{local slope} \end{aligned} \tag{6.21.10}$$

The WMA filters in the HMA indicator can be replaced with EMA filters resulting in the so-called “exponential Hull moving average” (EHMA), which has been found to be very competitive with other indicators [347]. Because N does not have to be an integer in EMA, it is not necessary to round the lengths $N_1 = \sqrt{N}$ and $N_2 = N/2$, and one can implement the indicator as follows, where H_N denotes the single EMA of Eq. (6.21.6),

$$\begin{aligned} \tilde{n}_a &= \frac{\sqrt{N}-2}{2}, \quad \tilde{n}_3 - \tilde{n}_2 = \frac{N}{4} \\ H_a(z) &= H_{\sqrt{N}}(z) [2H_{N/2}(z) - H_N(z)] \\ H_b(z) &= \frac{4}{N} H_{\sqrt{N}}(z) [H_{N/2}(z) - H_N(z)] \end{aligned}$$

One can also replace the WMA filters by SMAs leading to the “simple Hull moving average” (SHMA). The filter H_N now stands for a length- N SMA filter, resulting in $\tilde{n}_a = (\sqrt{N} - 1)/2$, and $\tilde{n}_3 - \tilde{n}_2 = (N - N_2)/2$. Except for these changes, the computational procedure outlined in Eq. (6.21.9) remains the same.

The following MATLAB code illustrates the computation of the local-level output signal a_n from the data y_n , for the three versions of HMA and a given value of $N > 1$,

```

N1 = round(sqrt(N));   e1 = N1 - sqrt(N);
N2 = round(N/2);      e2 = N2 - N/2;

nu = (N/2 + e1 + e2) / (N/2 - e2);

a = wma((1+nu)*wma(y,N2) - nu*wma(y,N), N1);    % HMA
a = sma((1+nu)*sma(y,N2) - nu*sma(y,N), N1);    % SHMA
a = sema(2*sema(y,N/2) - sema(y,N), sqrt(N));  % EHMA
    
```

The functions, **hma**, **shma**, **ehma**, which are discussed below, implement these operations but offer more options, including the computation of the slope signals.

In the zero-lag EMA (ZEMA or ZLEMA) indicator [284], $H_a(z) = 2H(z) - z^{-d}H(z)$, the filter $H(z)$ is chosen to be a single EMA filter of the form of Eq. (6.21.6), and the delay d is chosen to coincide with the filter lag, that is, $d = \bar{n} = (N - 1)/2$. It follows from Eq. (6.21.4) that the lag will be exactly zero, $\bar{n}_a = \bar{n} - d = 0$. This assumes that \bar{n} is an integer, which happens only for odd N . For even N , the delay d may be chosen as the rounded-up version of \bar{n} , that is,

$$d = \text{round}(\bar{n}) = \text{round}\left(\frac{N - 1}{2}\right) = \frac{N}{2}, \quad N = \text{even}$$

Then, the lag \bar{n}_a can still be made to be zero by choosing the parameter ν such that $\bar{n}_a = \bar{n} - \nu d = 0$, or, $\nu = \bar{n}/d = \bar{n}/\text{round}(\bar{n})$. Thus, the generalized form of the ZEMA indicator is constructed by,

$$\begin{aligned} \bar{n} &= \frac{N - 1}{2}, \quad d = \text{round}(\bar{n}), \quad \nu = \frac{\bar{n}}{d} \\ H_a(z) &= (1 + \nu)H(z) - \nu z^{-d}H(z) \\ H_b(z) &= \frac{1}{d}[H(z) - z^{-d}H(z)] \end{aligned} \tag{6.21.11}$$

The code segment below illustrates the computation of the local-level ZEMA signal. It uses the function, **delay**, which implements the required delay.

```
nbar = (N-1)/2;
d = round(nbar);
v = nbar/d;
a = (1+v)*sema(y,N) - v*delay(sema(y,N), d); % ZEMA
```

The following MATLAB functions implement the reduced-lag filters discussed above, where the input array **y** represents the financial data to be filtered, and the outputs **a**, **b** represent the local-level and local-slope signals.

```
[a,b] = hma(y,N,yin); % Hull moving average
[a,b] = ehma(y,N,yin); % exponential Hull moving average
[a,b] = shma(y,N,yin); % simple Hull moving average
[a,b] = zema(y,N,yin); % zero-lag EMA
y = delay(x,d); % d-fold delay, y(n) = x(n-d)
a = gdema(y,N,v,yin); % generalized DEMA
a = t3(y,N,v,yin); % Tillson's T3
```

The input variable **yin** defines the initialization and defaults to progressive filtering for **hma**, **shma**, and **zema**, **yin**='f', and to, **yin** = **y**₀, for **ehma**.

Fig. 6.21.1 compares the PMA/linear regression indicator with HMA, EHMA, and ZEMA on the same Nicor-Gas data, with filter length $N = 20$. Fig. 6.21.2 compares the corresponding slope indicators. The MATLAB code below illustrates the computation.

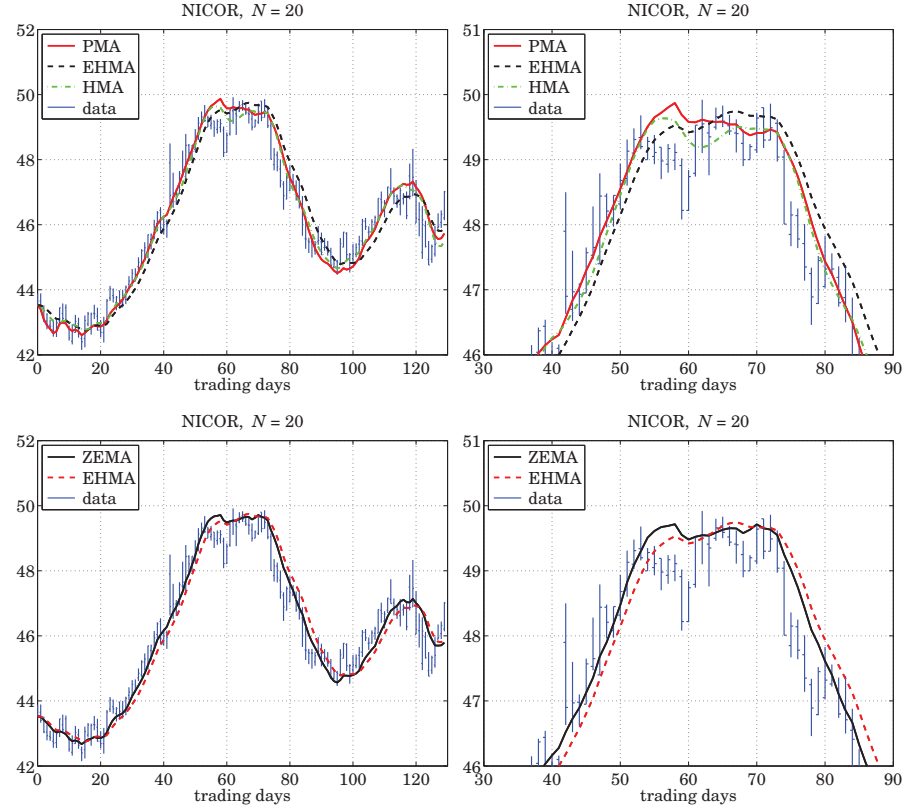


Fig. 6.21.1 Comparison of PMA/LREG, HMA, EHMA, and ZEMA indicators.

```
Y = xlsread('nicor.xls'); % load Nicor-Gas data
Y = Y(1:130,1:4); % keep 130 trading days, and [O,H,L,C] prices
y = Y(:,4); % closing prices
t = 0:length(y)-1; % trading days

N = 20; % filter length

[a1,b1] = lreg(y,N); % PMA/LREG
[ah,bh] = hma(y,N); % HMA
[ae,be] = ehma(y,N); % EHMA
[az,bz] = zema(y,N); % ZEMA

figure; plot(t,a1, t,ae, t,ah); % PMA/LREG, EHMA, HMA
hold on; ohlc(t,Y); % add OHLC chart

figure; plot(t,az, t,ae); % ZEMA, EHMA
hold on; ohlc(t,Y); % add OHLC chart
```

```
figure; plot(t,bh, t,be); hold on; % HMA, EHMA slopes
stem(t,b1,'marker','none'); % plot LREG slope as stem

figure; plot(t,bh, t,bz); hold on; % HMA, ZEMA slopes
stem(t,b1,'marker','none');
```

We note that the reduced-lag HMA, EHMA, and ZEMA local-level and local-slope filters have comparable performance as the PMA/linear regression and DEMA/TEMA filters, with a comparable degree of smoothness.

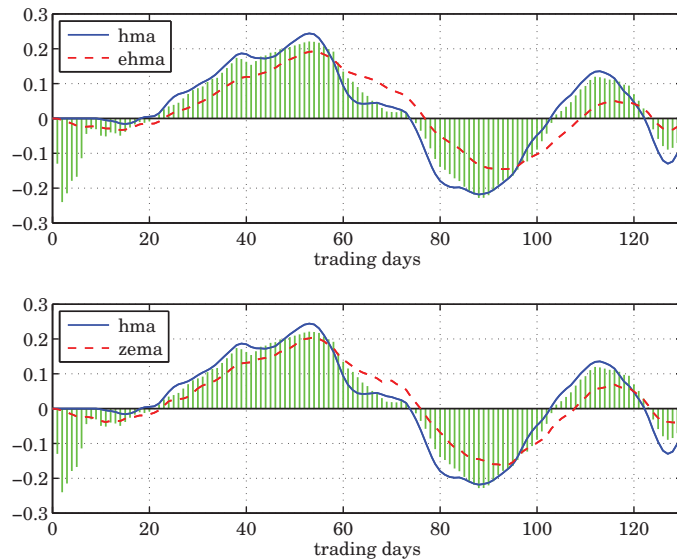


Fig. 6.21.2 Slope indicators, linear regression (stem) vs. HMA, EHMA, ZEMA.

6.22 Envelopes, Bands, and Channels

Moving averages help market traders discern trends by smoothing out variations in the data. However, such variations can provide additional useful information, such as gauging volatility, or, identifying extremes in the data that provide trading opportunities, or observing how prices settle into their trends.

Trading envelopes or bands or channels consist of two curves drawn above and below a moving average trendline. The two bounds define a zone of variation, or volatility, about the average, within which most of the price fluctuations are expected to lie.

The typical trading rule is that when a price closes near or above the upper bound, it signals that the stock is overbought and suggests trading in the opposite direction. Similarly, if a price moves below the lower bound it signals that the stock is oversold and suggests an opposite reaction.

6.22. Envelopes, Bands, and Channels

In this section we discuss the following types of bands and their computation,

- Bollinger bands
- Projection bands
- Fixed-width bands
- Starc bands
- Standard-error bands
- Donchian channels
- Keltner bands
- Parabolic SAR

Examples of these are shown in Figs. 6.22.1 and 6.22.2 applied to the same Nicor data that we used previously. Below we give brief descriptions of how such bands are computed. We use our previously discussed MATLAB functions, such as SMA, LREG, etc., to summarize the computations. Further references are given in [323–334].

Bollinger Bands

Bollinger bands [323–327] are defined relative to an N -day SMA of the closing prices, where typically, $N = 14$. The two bands are taken to be two standard deviations above and below the SMA. The following MATLAB code clarifies the computation,

```
M = sma(y,N); % N-point SMA of closing prices y
S = stdev(y,N); % std-dev relative to M
L = M - 2*S; % Lower bound
U = M + 2*S; % upper bound
```

where the function, `stdev`, uses the built-in function `std` to calculate the standard deviation over each length- N data window, and its essential code is,

```
for n=1:length(y),
    S(n) = std(y(max(1,n-N+1):n));
end
```

where the data window length is N for $n \geq N$, and n during the initial transients $n < N$.

Standard-Error Bands

Standard-error bands [329] use the PMA/linear-regression moving average as the middle trendline and shift it by two standard errors up and down. The calculation is summarized below with the help of the function `lreg`, in which the quantities, y , a , se , represent the closing prices, the local level, and the standard error,

```
[a,~,~,se] = lreg(y,N); % N-point linear regression
L = a - 2*se; % lower bound
U = a + 2*se; % upper bound
```

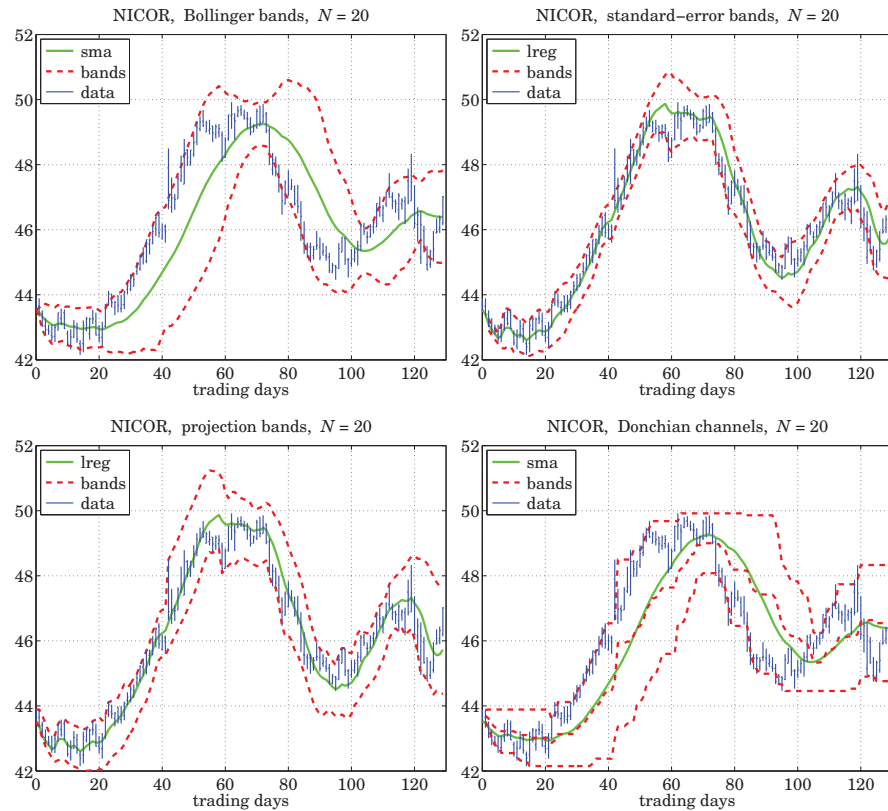


Fig. 6.22.1 Bollinger bands, standard-error bands, projection bands, and Donchian channels.

Projection Bands

Projection bands [328] also use the linear regression function `lreg` to calculate the local slopes for the high and low prices, H, L . The value of the upper (lower) band at the n -th time instant is determined by considering the values of the highs H (lows L) over the look-back period, $n - N + 1 \leq t \leq n$, extrapolating each of them linearly according to their slope to the current time instant n , and taking the maximum (minimum) among them. The following MATLAB code implements the procedure,

```
[~,bL] = lreg(L,N); % linear regression slope for Low
 [~,bH] = lreg(H,N); % linear regression slope for High
for n=0:length(H)-1,
    t = (max(0,n-N+1) : n)'; % look-back interval
    Lo(n+1) = min(L(t+1) + bL(n+1)*(n-t)); % lower band
    Up(n+1) = max(H(t+1) + bH(n+1)*(n-t)); % upper band
end
```

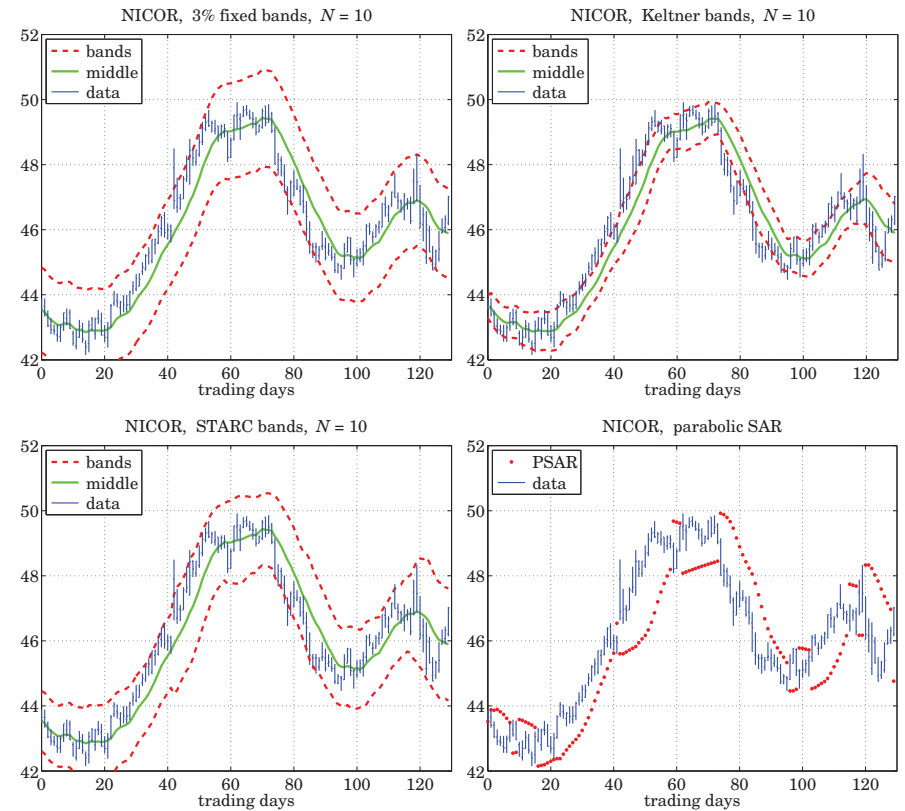


Fig. 6.22.2 Fixed-width bands, Keltner bands, STARC bands, and parabolic SAR.

Donchian Channels

Donchian channels [331] are constructed by finding, at each time instant n , the highest high (resp. lowest low) over the past time interval, $n - N \leq t \leq n - 1$, that is, the value of the upper bound at the n -th day, is the maximum of the highs over the previous N

days, not including the current day, i.e., $\max[H_{n-1}, H_{n-2}, \dots, H_{n-N}]$. The code below describes the procedure,

```
for n = 2:length(H)      % n is MATLAB index
    t = max(1,n-N) : n-1; % past N days
    Lo(n) = min(L(t));    % lower band
    Up(n) = max(H(t));    % upper band
end
Mid = (Up + Lo)/2;      % middle band
```

Fixed-Width Bands

Fixed-width bands or envelopes [330] shift an N -point SMA of the closing prices by a certain percentage, such as, typically, 3 percent,

```
M = sma(C,N);          % N-point SMA of closing prices C
L = M - p*M;           % lower band, e.g., p = 0.03
U = M + p*M;           % upper band
```

Keltner Bands

Keltner bands or channels [330], use as the middle trendline an N -point SMA of the average of the high, low, and closing prices, $(H + L + C)/3$, and use an N -point SMA of the difference $(H - L)$ as the bandwidth, representing a measure of volatility. The code below implements the operations,

```
M = sma((H+L+C)/3,N); % SMA of (H+L+C)/3
D = sma(H-L,N);       % SMA of (H-L)
L = M - D;             % lower band
U = M + D;             % upper band
```

The typical value of N is 10, and the trading rule is that a “buy” signal is generated when the closing price C lies above the upper band, and a “sell” signal when C lies below the lower band.

Starc Bands

In Starc[†] bands [330] all three prices, high, low, and closing, H, L, C , are used. The middle band is an N -point SMA of the closing prices C , but the bandwidth is defined in terms of an N_a -point of the so-called “average true range” (ATR), which represents another measure of volatility. The code below describes the computation,

```
M = sma(C,N);          % SMA of closing prices
R = atr([H,L,C],Na);   % ATR = average true range
L = M - 2*R;           % lower band
U = M + 2*R;           % upper band
```

[†]Stoller Average Range Channels

The ATR [281] is an N_a -point WEMA of the “true range”, defined as follows, at each time n ,

$$T_n = \max[H_n - L_n, H_n - C_{n-1}, C_{n-1} - L_n] = \text{true range in } n\text{-th day}$$

$$R_n = \text{wema}(T_n, N_a) = \text{ATR} \quad (6.22.1)$$

and is implemented by the function `atr`, with the help of the `delay` function. Its essential MATLAB code is as follows, where H, L, C are column vectors,

```
T = max([H-L, H-delay(C,1), delay(C,1)-L], [], 2); % row-wise max
R = wema(T,N);
```

MATLAB Functions

The following MATLAB functions implement the band indicators discussed above, where the various parameters are fully explained in the help files for these functions,

```
[L,U,M] = bbands(y,N,d); % Bollinger bands
[L,U,a] = sebands(y,N,d,init); % standard-error bands
[L,U,R,Rs] = pbands(Y,N,Ns); % projection bands & oscillator
[L,U,M] = donch(Y,N); % Donchian channels
[L,U,M] = fbands(Y,N,p); % fixed-width bands
[L,U,M] = kbands(Y,N); % Keltner bands
[L,U,M] = stbands(Y,N,Na); % Starc bands
S = stdev(y,N,flag); % standard deviation
[R,TR] = atr(Y,N); % average true range
```

The essential MATLAB code for generating Figs. 6.22.1 and 6.22.2 is as follows,

```
Y = xlsread('nicor.xls'); % load Nicor-Gas data
Y = Y(1:130,1:4); % 130 trading days, and [0,H,L,C] prices
y = Y(:,4); % closing prices
t = 0:length(y)-1; % trading days

N = 20; % used in Fig.6.22.1

[L,U,M] = bbands(y,N); % Bollinger
figure; ohlc(t,Y); hold on; % make OHLC bar chart
plot(t,M,'g-', t,U,'r--', t,L,'r--');

[L,U,a] = sebands(y,N); % standard-error
figure; ohlc(t,Y); hold on;
plot(t,a,'g-', t,U,'r--', t,L,'r--');

a = lreg(y,N);
[L,U] = pbands(Y,N); % projection
figure; ohlc(t,Y); hold on;
plot(t,a,'g-', t,U,'r--', t,L,'r--');

[L,U,M] = donch(Y,N); % Donchian
figure; ohlc(t,Y); hold on;
plot(t,M,'r--', t,L,'r--', t,U,'r--');
plot(t,sma(y,N),'g-');
```



```

N=10; % used in Fig.6.22.2

p=0.03; % fixed-width
[L,U,M] = fbands(Y,N,p);
figure; ohlc(t,Y); hold on;
plot(t,M,'g-', t,U,'r--', t,L,'r--');

[L,U,M] = kbands(Y,N); % Keltner
figure; ohlc(t,Y); hold on;
plot(t,M,'g-', t,U,'r--', t,L,'r--');

[L,U,M] = stbands(Y,N); % Starc
figure; ohlc(t,Y); hold on;
plot(t,M,'g-', t,U,'r--', t,L,'r--');

H = Y(:,2); L = Y(:,3); ni=1; Ri=1; % parabolic SAR
S = psar(H,L,Ri,ni);
figure; ohlc(t,Y); hold on;
plot(t,S,'r.');
```

Parabolic SAR

Wilder's parabolic stop & reverse (SAR) [281] is a trend-following indicator that helps a trader to switch positions from long to short or vice versa.

While holding a long position during a period of increasing prices, the SAR indicator lies *below* the prices and is also increasing. When the prices begin to fall and touch the SAR from above, then a “sell” signal is triggered with a reversal of position from long to short, and the SAR switches sides and begins to fall, lying *above* the falling prices. If subsequently, the prices begin to rise again and touch the SAR from below, then a “buy” signal is triggered and the position is reversed from short to long again, and so on.

The indicator is very useful as it keeps a trader constantly in the market and works well during trending markets with steady periods of increasing or decreasing prices, even though it tends to recommend buying relatively high and selling relatively low—the opposite of what is the ideal. It does not work as well during “sideways” or trading markets causing so-called “whipsaws.” It is usually used in conjunction with other indicators that confirm trend, such as the RSI or DMI. Some further references on the SAR are [335–340].

The SAR is computed in terms of the high and low price signals H_n, L_n and is defined as the exponential moving average of the *extreme price* reached within each trending period, but it uses a time-varying EMA parameter, $\lambda_n = 1 - \alpha_n$, as well as additional conditions that enable the reversals. Its basic EMA recursion from day n to day $n+1$ is,

$$S_{n+1} = \lambda_n S_n + \alpha_n E_n = (1 - \alpha_n) S_n + \alpha_n E_n, \quad \text{or,}$$

$$\boxed{S_{n+1} = S_n + \alpha_n (E_n - S_n)} \quad (\text{SAR}) \quad (6.22.2)$$

where E_n is the extreme price reached during the current trending position, that is, the highest high reached up to day n during an up-trending period, or the lowest low up to day n during a down-trending period. At the beginning of each trending period, S_n is initialized to be the extreme price of the previous trending period.

The EMA factor α_n increases linearly with time, starting with an initial value, α_i , at the beginning of each trending period, and then increasing by a fixed increment $\Delta\alpha$, but only every time a *new* extreme value is reached, that is,

$$\alpha_{n+1} = \begin{cases} \alpha_n + \Delta\alpha, & \text{if } E_{n+1} \neq E_n \\ \alpha_n, & \text{if } E_{n+1} = E_n \end{cases} \quad (6.22.3)$$

where we note that $E_{n+1} \neq E_n$ happens when E_{n+1} is strictly greater than E_n during an up-trend, or, E_{n+1} is strictly less than E_n during a down-trend. Moreover, an additional constraint is that α_n is not allowed to exceed a certain maximum value, α_m . The values recommended by Wilder [281] are,

$$\alpha_i = 0.02, \quad \Delta\alpha = 0.02, \quad \alpha_m = 0.2$$

Because of the increasing α_n parameter, the EMA has a time-varying decreasing lag,[†] thus, tracking more quickly the extreme prices as time goes by. As a result, S_n has a particular curved shape that resembles a parabola, hence the name “parabolic” SAR.

The essential steps in the calculation of the SAR are summarized in the following MATLAB code, in which the inputs are the quantities, H, L, representing the high and low price signals, H_n, L_n , while the output quantities, S, E, a, R, represent, S_n, E_n, α_n, R_n , where R_n holds the current position and is equal to ± 1 for long/short.

```

Hi = max(H(1:ni)); % initial highest high, default
Li = min(L(1:ni)); % initial lowest low

R(ni) = Ri; % initialize outputs at starting time n=ni
a(ni) = ai;
S(ni) = Li*(Ri==1) + Hi*(Ri==-1);
E(ni) = Hi*(Ri==1) + Li*(Ri==-1);

for n = ni : length(H)-1
    S(n+1) = S(n) + a(n) * (E(n) - S(n)); % SAR update
    r = R(n); % current position
    if (r==1 & L(n+1)<=S(n+1)) | (r==-1 & H(n+1)>=S(n+1)) % reversal
        r = -r; % reverse r
        S(n+1) = E(n); % reset new S
        E(n+1) = H(n+1)*(r==1) + L(n+1)*(r==-1); % reset new E
        a(n+1) = ai; % reset new a
    else % no reversal
        if n>2 % new S
            S(n+1) = min([S(n+1), L(n-1), L(n)])*(r==1) ... % additional
                + max([S(n+1), H(n-1), H(n)])*(r==-1); % conditions
        end
        E(n+1) = max(E(n), H(n+1))*(r==1) ... % new E
            + min(E(n), L(n+1))*(r==-1);
        a(n+1) = min(a(n) + (E(n+1)~=E(n)) * Da, am); % new a
    end
    R(n+1) = r; % new R
end % for-loop
```

[†]The EMA equivalent length decreases from, $N_i = 2/\alpha_i - 1 = 99$, down to, $N_m = 2/\alpha_m - 1 = 9$.

If the current trading position is long ($r = 1$), corresponding to an up-trending market, then, a reversal of position to short ($r = -1$) will take place at time $n+1$ if the low price L_{n+1} touches or becomes less than the SAR, that is, if, $L_{n+1} \leq S_{n+1}$. Similarly, if the current position is short, corresponding to a down-trending market, then, a reversal of position to long will take place at time $n+1$ if the high price H_{n+1} touches or becomes greater than the SAR, that is, if, $H_{n+1} \geq S_{n+1}$. At such reversal time points, the SAR is reset to be equal to the extreme price of the previous trend, that is, $S_{n+1} = E_n$, and the E_{n+1} is reset to be either L_{n+1} if reversing to short, or H_{n+1} if reversing to long, and the EMA parameter is reset to, $\alpha_{n+1} = \alpha_t$.

An additional condition is that during an up-trend, the SAR for tomorrow, S_{n+1} , is not allowed to become greater than either today's or yesterday's lows, L_n, L_{n-1} , and in such case it is reset to the minimum of the two lows. Similarly, during a down-trend, the S_{n+1} is not allowed to become less than either today's or yesterday's highs, H_n, H_{n-1} , and is reset to the maximum of the two highs. This is enforced by the code line,

$$S_{n+1} = \min([S_{n+1}, L_{n-1}, L_n]) \cdot (r==1) + \max([S_{n+1}, H_{n-1}, H_n]) \cdot (r==-1)$$

The parabolic SAR is implemented with the MATLAB function `psar`, with usage,

```
[S,E,a,R] = psar(H,L,Ri,ni,af,Hi,Li); % parabolic SAR

H = vector of High prices, column
L = vector of Low prices, column, same length as H
Ri = starting position, long Ri = 1, short Ri = -1
ni = starting time index, default ni = 1, all outputs are NaNs for n<ni
af = [ai,da,am] = [initial EMA factor, increment, maximum factor]
    default, af = [0.02, 0.02, 0.2]
Hi,Li = initial high and low used to initialize S(n),E(n) at n=ni,
    default, Hi = max(H(1:ni)), Li = min(L(1:ni))

S = parabolic SAR, same size as H
E = extremal price, same size as H
a = vector of EMA factors, same size as H
R = vector of positions, R = +1/-1 for long/short, same size as H
```

The SAR signal S_n is usually plotted with dots, as shown for example, in the bottom right graph of Fig. 6.22.2. Fig. 6.22.3 shows two more examples.

The left graph reproduces Wilder's original example [281] and was generated by the following MATLAB code,

```
Y = xlsread('psarexa.xls'); % data from Wilder [281]
t = Y(:,1); H = Y(:,2); L = Y(:,3); % extract H,L signals

Ri = 1; ni = 4; % initialize SAR
[S,E,a,R] = psar(H,L,Ri,ni);

num2str([t, H, L, a, E, S, R], '%8.2f'); % reproduces table on p.13 of [281]

figure; ohlc(t,[H,L], t,S,'r.');
```

The right graph is from Achelis [280]. The SAR is plotted with filled dots, but at the end of each trending period and shown with open circles are the points that triggered the reversals. The MATLAB code for this example is similar to the above,

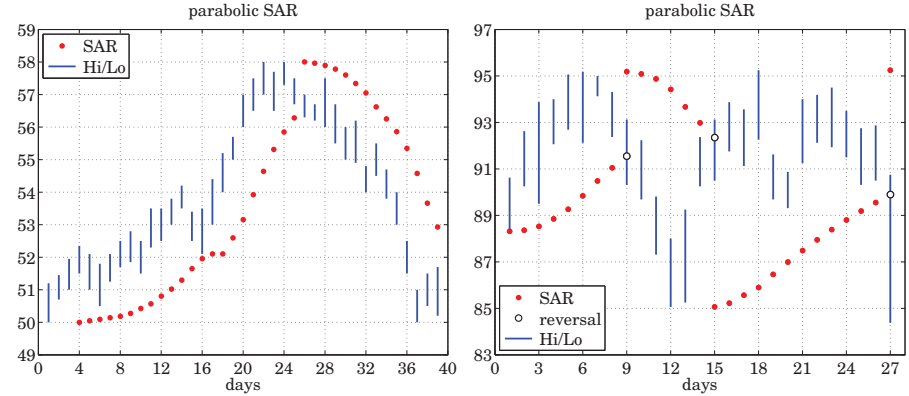


Fig. 6.22.3 Parabolic SAR examples from Wilder [281] and Achelis [280].

```
Y = xlsread('psarexb.xls'); % data from Ref. [208]
t = Y(:,1); H = Y(:,2); L = Y(:,3);

Ri = 1; ni = 1; % initialize
[S,E,a,R] = psar(H,L,Ri,ni); % compute SAR

num2str([t, H, L, a, E, S, R], '%9.4f'); % reproduces table from Ref. [280]
figure; ohlc(t,[H,L], t,S,'r.');
```

The first up-trending period ends at day $n = 9$ at which the would be SAR, shown as an opened-circle, lies above the low of that day, thus, causing a reversal to short and that SAR is then replaced with the filled-circle value that lies above the highs, and corresponds to the highest high of the previous period that had occurred on day $n = 6$.

The second down-trending period ends at $n = 15$ at which point the SAR, shown as an opened-circle, is breached by the high on that day, thus causing a reversal to long, and the SAR is reset to the filled-circle value on that day lying below the data, and corresponds to the lowest low during the previous period that had been reached on day $n = 12$. Finally, another such reversal takes place on day $n = 27$ and the SAR is reset to the highest high that had occurred on day $n = 18$. To clarify, we list below the values of the SAR at the reversal points before and after the reversal takes place,

n	$S_{\text{before}}(n)$	$S_{\text{after}}(n)$
9	91.5448	95.1875 = H_6
15	92.3492	85.0625 = L_{12}
27	89.8936	95.2500 = H_{18}

6.23 Momentum, Oscillators, and Other Indicators

There exist several other indicators that are used in technical analysis, many of them built on those we already discussed. The following MATLAB functions implement some

of the more popular ones, several are also included in MATLAB's financial toolbox. Additional references can be found in the Achelis book [280] and in [281–347].

```
R = rsi(y,N,type);           % relative strength index, RSI
R = cmo(y,N);               % Chande momentum oscillator, CMO
R = vhfilt(y,N);           % Vertical Horizontal Filter, VHF
[Dp,Dm,DX,ADX] = dirmov(Y,N); % directional movement system, +-DI,DX,ADX

-----

[y,yr,ypr] = mom(x,d,xin);  % momentum and price rate of change
[y,ys,ypr] = prosc(x,N1,N2,N3); % price oscillator & MACD
[pK,pD] = stoch(Y,K,Ks,D,M); % stochastic, %K, %D oscillators

-----

R = accdist(Y);            % accumulation/distribution line
R = chosc(Y,N1,N2);       % Chaikin oscillator
R = cmflow(Y,N);          % Chaikin money flow
R = chvol(Y,N);           % Chaikin volatility

-----

[P,N] = pnvi(Y,P0);        % positive/negative volume indices, PVI/NVI
R = cci(Y,N);              % commodity channel index, CCI
R = dpo(Y,N);              % detrended price oscillator, DPO
[R,N] = dmi(y,Nr,Ns,Nm);  % dynamic momentum index, DMI
[R,Rs] = forosc(y,N,Ns);  % forecast oscillator

-----

[R,Rs] = trix(y,N,Ns,yin); % TRIX oscillator
a = vema(y,N,Nv);         % variable-length EMA
```

Below we discuss briefly their construction. Examples of their use are included in their respective help files. Several online examples can be found in the Fidelity Guide [345] and in the TradingView Wiki [346].

Relative Strength Index, RSI

The relative strength index (RSI) was introduced by Wilder [281] to be used in conjunction with the parabolic SAR to confirm price trends. It is computed as follows, where y is the column vector of daily closing prices,

```
x = diff(y);           % price differences
xu = +x.*(x>0);        % upward differences
xd = -x.*(x<=0);      % downward differences
su = wema(xu,N);       % smoothed differences
sd = wema(xd,N);
RSI = 100*su/(su+sd); % RSI
```

Chande Momentum Oscillator, CMO

If the `wema` function is replaced by `sma`, one obtains the Chande momentum oscillator,

```
x = diff(y);           % price differences
xu = +x.*(x>0);        % upward differences
xd = -x.*(x<=0);      % downward differences
su = sma(xu,N);        % smoothed differences
sd = sma(xd,N);
CMO = 100*(su-sd)/(su+sd); % CMO
```

Thus, the SMA-based RSI is related to CMO via,

$$\text{CMO} = 2 \text{RSI} - 100 \Leftrightarrow \text{RSI} = \frac{\text{CMO} + 100}{2}$$

Vertical Horizontal Filter, VHF

The vertical horizontal filter (VHF) is similar to the RSI or CMO and helps to confirm a trend in trending markets. It is computed as follows, where the first N outputs are NaNs,

```
x = [NaN; diff(y)];      % y = column of closing prices
                          % x = price differences

for n=N+1:length(y),
    yn = y(n-N+1:n);     % length-N look-back window
    xn = x(n-N+1:n);
    R(n) = abs(max(yn)-min(yn)) / sum(abs(xn)); % VHF
end
```

Directional Movement System

The directional movement system was also proposed by Wilder [281] and consists of several indicators, the plus/minus directional indicators, (\pm DI), the directional index (DX), and the average directional index (ADX). These are computed as follows,

```
R = atr(Y,N);           % average true range
DH = [0; diff(H)];     % high price differences
DL = [0; -diff(L)];    % low price differences
Dp = DH .* (DH>DL) .* (DH>0); % daily directional movements
Dm = DL .* (DL>DH) .* (DL>0); %
Dp = wema(Dp,N);       % averaged directional movements
Dm = wema(Dm,N);
Dp = 100 * Dp ./ R;    % +DI,-DI directional indicators
Dm = 100 * Dm ./ R;
DX = 100*abs(Dp - Dm)./(Dp + Dm); % directional index, DI
ADX = wema(DX,N);     % average directional index, ADX
```

Momentum and Price Rate of Change

In its simplest form a momentum indicator is the difference between a price today, $x(n)$, and the price d days ago, $x(n-d)$, but it can also be expressed as a ratio, or as a

percentage, referred to as *price rate of change*,

$$y(n) = x(n) - x(n-d) = \text{momentum}$$

$$y_r(n) = 100 \cdot \frac{x(n)}{x(n-d)} = \text{momentum as ratio}$$

$$y_p(n) = 100 \cdot \frac{x(n) - x(n-d)}{x(n-d)} = \text{price rate of change}$$

It can be implemented simply with the help of the function, `delay`,

```
y = x - delay(x,d);
yr = x/delay(x,d) * 100;
yp = (x-delay(x,d))/delay(x,d) * 100;
```

Price Oscillator and MACD

The standard moving average convergence/divergence (MACD) indicator is defined as the difference between two EMAs of the daily closing prices: a length-12 shorter/faster EMA and a length-26 longer/slower EMA. A length-9 EMA of the MACD difference is also computed as a trigger signal.

Typically, a buy (sell) signal is indicated when the MACD rises above (falls below) zero, or when it rises above (falls below) its smoothed signal line.

The MACD can also be represented as a percentage resulting into the price oscillator, and also, different EMA lengths can be used. The following code segment illustrates the computation, where `x` are the closing prices,

```
y1 = sema(x,N1); % fast EMA, default N1=12
y2 = sema(x,N2); % slow EMA, default N2=26
y = y1 - y2; % MACD
ys = sema(y,N3); % smoothed MACD signal, default N3=9
ypr = 100 * y./y2; % price oscillator
```

Stochastic Oscillator

```
H = Y(:,1); L = Y(:,2); C = Y(:,3); % extract H,L,C inputs
Lmin = NaN(size(C)); Hmax = NaN(size(C)); % NaNs for n<K
for n = K:length(C), % look-back period K
    Lmin(n) = min(L(n-K+1:n)); % begins at n=K
    Hmax(n) = max(H(n-K+1:n));
end
pK = 100 * sma(C-Lmin, Ks) ./ sma(Hmax-Lmin, Ks); % percent-K
pD = sma(pK, D); % percent-D
```

Fast Stochastic has $K_s = 1$, i.e., no smoothing, and *Slow Stochastic* has, typically, $K_s = 3$.

Accumulation/Distribution

```
H=Y(:,1); L=Y(:,2); C=Y(:,3); V=Y(:,4); % extract H,L,C,V
R = cumsum((2*C-H-L)./(H-L).*V); % ACCDIST
```

Chaikin Oscillator

```
y = accdist(Y); % Y = [H,L,C,V] data matrix
R = sema(y,N1) - sema(y,N2); % CHOSC, default, N1=3, N2=10
```

Chaikin Money Flow

```
H=Y(:,1); L=Y(:,2); C=Y(:,3); V=Y(:,4); % extract H,L,C,V
R = sma((2*C-H-L)./(H-L).*V, N) ./ sma(V,N); % CMFLOW
```

Chaikin Volatility

```
S = sema(H-L,N); % H,L given
R = (S - delay(S,N)) ./ delay(S,N) * 100; % volatility
```

Positive/Negative Volume Indices, PNVI

These are defined recursively as follows, in MATLAB-like notation, where C_n, V_n are the closing prices and the volume,

$$P_n = P_{n-1} + (V_n > V_{n-1}) \cdot \frac{C_n - C_{n-1}}{C_{n-1}} \cdot P_{n-1} = P_{n-1} \left(\frac{C_n}{C_{n-1}} \right)^{(V_n > V_{n-1})} = \text{PVI}$$

$$N_n = N_{n-1} + (V_n < V_{n-1}) \cdot \frac{C_n - C_{n-1}}{C_{n-1}} \cdot N_{n-1} = N_{n-1} \left(\frac{C_n}{C_{n-1}} \right)^{(V_n < V_{n-1})} = \text{NVI}$$

and initialized to some arbitrary initial value, such as $P_0 = N_0 = 1000$. The MATLAB implementation uses the function, `delay`,

```
P = P0 * cumprod( (C./delay(C,1)) .^ (V>delay(V,1)) ); % PNVI
N = P0 * cumprod( (C./delay(C,1)) .^ (V<delay(V,1)) );
```

Commodity Channel Index, CCI

```
T = (H+L+C)/3; % H,L,C given
M = sma(T,N);
for n=N+1:length(C),
    D(n) = mean(abs(T(n-N+1:n) - M(n))); % mean deviation
end
R = (T-M) ./ D / 0.015; % CCI
```

Detrended Price Oscillator, DPO

```
S = sma(y,N,'n');           % y = column of closing prices
M = floor(N/2) + 1;        % advancing time
R = y - delay(S,-M,'n');    % DPO, i.e., R(n) = y(n) - S(n+M)
```

Dynamic Momentum Index, DMI

```
x = [NaN; diff(y)];        % y = column of closing prices
xu = x .* (x>0);          % upward differences
xd = -x .* (x<=0);        % downward differences
S = stdev(y,Ns);          % Ns-period stdev
V = S ./ sma(S,Nm);       % volatility measure
N = floor(Nr ./ V);       % effective length
N(N>Nmax) = Nmax;         % restrict max and min N
N(N<Nmin) = Nmin;
Nstart = Nr + Ns + Nm;
su1 = mean(xu(2:Nstart));  % initialize at start time
sd1 = mean(xd(2:Nstart));
switch lower(type)
case 'wema'                % WEMA type
    for n = Nstart+1:length(y),
        su(n) = su1 + (xu(n) - su1) / N(n); su1 = su(n);
        sd(n) = sd1 + (xd(n) - sd1) / N(n); sd1 = sd(n);
    end
case 'sma'                 % SMA type
    for n = Nstart+1:length(y),
        su(n) = mean(xu(n-N(n)+1:n));
        sd(n) = mean(xd(n-N(n)+1:n));
    end
end
R = 100 * su./(su+sd);     % DMI
```

Forecast Oscillator

```
yp = pma(y,N,1);          % time series forecast
x = y - delay(yp,1);
R = 100 * x./y;           % forecast oscillator
Rs = sma(R,Ns);           % trigger signal
```

TRIX Oscillator

```
[~,~,~,~,a3] = tema(y,N,cin); % triple EMA
R = 100*(a3 - delay(a3,1))./a3; % TRIX
Rs = sma(R,Ns);              % smoothed TRIX
```

Variable-Length EMA

```
la = (N-1)/(N+1); a1 = 1-la; % EMA parameter
switch lower(type)
case 'cmo'                  % CMO volatility
    V = abs(cmo(y,Nv))/100;
case 'r2'                   % R^2 volatility
    [~,~,V] = lreg(y,Nv);
end
for n=Nv+1:length(y),      % default si=y(Nv)
    s(n) = si + a1*V(n)*(y(n)-si); % EMA recursion
    si = s(n);
end
```

6.24 MATLAB Functions

We summarize the MATLAB functions discussed in this chapter:

```
% -----
% Exponential Moving Average Functions
% -----
% ema      - exponential moving average - exact version
% stema    - steady-state exponential moving average
% lpbasis  - fit order-d polynomial to first L inputs
% emap     - map equivalent lambdas's between d=0 and d=1 EMAs
% emaerr   - MSE, MAE, MAPE error criteria
% emat     - transformation matrix from polynomial to cascaded basis
% mema     - multiple exponential moving average
% holt     - Holt's exponential smoothing
% holtterr - MSE, MAE, MAPE error criteria for Holt
```

The technical analysis functions are:

```
% -----
% Technical Analysis Functions
% -----
% accdist  - accumulation/distribution line
% atr      - true range & average true range
% cci      - commodity channel index
% chosc    - Chaikin oscillator
% cmflow   - Chaikin money flow
% chvol    - Chaikin volatility
% cmo      - Chande momentum oscillator
% dirmov   - directional movement system, +-DI, DX, ADX
% dmi      - dynamic momentum index (DMI)
```

```

% dpo - detrended price oscillator
% forosc - forecast oscillator
% pnvi - positive and negative volume indices, PVI, NVI
% prosc - price oscillator & MACD
% psar - Wilder's parabolic SAR
% rsi - relative strength index, RSI
% stdev - standard deviation index
% stoch - stochastic oscillator, %K, %D oscillators
% trix - TRIX oscillator
% vhfilt - Vertical Horizontal Filter
%
% ----- moving averages -----
%
% bma - Butterworth moving average
% dema - steady-state double exponential moving average
% ehma - exponential Hull moving average
% gdema - generalized dema
% hma - Hull moving average
% ilrs - integrated linear regression slope indicator
% delay - delay or advance by d samples
% mom - momentum and price rate of change
% lreg - linear regression, slope, and R-squared indicators
% pma - predictive moving average, linear fit
% pmaimp - predictive moving average impulse response
% pma2 - predictive moving average, polynomial order d=1,2
% pmaimp2 - predictive moving average impulse response, d=1,2
% sema - single exponential moving average
% shma - SMA-based Hull moving average
% sma - simple moving average
% t3 - Tillson's T3 indicator, triple gdema
% tema - triple exponential moving average
% tma - triangular moving average
% vema - variable-length exponential moving average
% wema - Wilder's exponential moving average
% wma - weighted or linear moving average
% zema - zero-lag EMA
%
% ----- bands -----
%
% bbands - Bollinger bands
% donch - Donchian channels
% fbands - fixed-envelope bands
% kbands - Keltner bands or channels
% pbands - Projection bands and projection oscillator
% sebands - standard-error bands
% stbands - STARC bands
%
% ----- misc -----
%
% ohlc - make Open-High-Low-Close bar chart
% ohlccy - OHLC with other indicators on the same graph
% yylim - adjust left/right ylim
%
% r2crit - R-squared critical values
% tcrit - critical values of Student's t-distribution
% tdistr - cumulative t-distribution

```

6.25 Problems

6.1 Consider a filter with a real-valued impulse response h_n . Let $H(\omega) = M(\omega)e^{-j\theta(\omega)}$ be its frequency response, where $M(\omega) = |H(\omega)|$ and $\theta(\omega) = -\arg H(\omega)$. First, argue that $\theta(0) = 0$ and $M'(0) = 0$, where $M'(\omega) = dM(\omega)/d\omega$. Then, show that the filter delay \tilde{n} of Eq. (6.1.18) is the group delay at DC, that is, show Eq. (6.1.19),

$$\tilde{n} = \left. \frac{d\theta(\omega)}{d\omega} \right|_{\omega=0}$$

6.2 The lag of a filter was defined by Eqs. (6.1.17) and (6.1.18) to be,

$$\tilde{n} = \frac{\sum_n n h_n}{\sum_n h_n} = - \left. \frac{H'(z)}{H(z)} \right|_{z=1}$$

If the filter $H(z)$ is the cascade of two filters, $H(z) = H_1(z)H_2(z)$, with individual lags, \tilde{n}_1, \tilde{n}_2 , then show that, regardless of whether $H_1(z), H_2(z)$ are normalized to unity gain at DC, the lag of $H(z)$ will be the sum of the lags,

$$\tilde{n} = \tilde{n}_1 + \tilde{n}_2$$

6.3 Consider a low-frequency signal $s(n)$ whose spectrum $S(\omega)$ is limited within a narrow band around DC, $|\omega| \leq \Delta\omega$, and therefore, its inverse DTFT representation is:

$$s(n) = \frac{1}{2\pi} \int_{-\Delta\omega}^{\Delta\omega} S(\omega) e^{j\omega n} d\omega$$

For the purposes of this problem, we may think of the above relationship as defining $s(n)$ also for non-integer values of n . Suppose that the signal $s(n)$ is filtered through a filter $H(\omega)$ with real-valued impulse response whose magnitude response $|H(\omega)|$ is approximately equal to unity over the $\pm\Delta\omega$ signal bandwidth. Show that the filtered output can be written approximately as the delayed version of the input by an amount equal to the group delay at DC, that is,

$$y(n) = \frac{1}{2\pi} \int_{-\Delta\omega}^{\Delta\omega} H(\omega) S(\omega) e^{j\omega n} d\omega \approx s(n - \tilde{n})$$

6.4 Show that the general filter-sharpening formula (6.9.5) results in the following special cases:

$$p = 0, q = d \Rightarrow H_{\text{impr}} = 1 - (1 - H)^{d+1}$$

$$p = 1, q = d \Rightarrow H_{\text{impr}} = 1 - (1 - H)^{d+1} [1 + (d + 1)H]$$

6.5 Prove the formulas in Eqs. (6.10.5) and (6.10.7).

6.6 Prove Eq. (6.4.10).

6.7 Consider the single and double EMA filters:

$$H(z) = \frac{1 - \lambda}{1 - \lambda z^{-1}}, \quad H_a(z) = 2H(z) - H^2(z) = \frac{(1 - \lambda)(1 + \lambda - 2\lambda z^{-1})}{(1 - \lambda z^{-1})^2}$$

a. Show that the impulse response of $H_a(z)$ is:

$$h_a(n) = (1 - \lambda) [1 + \lambda - (1 - \lambda)n] \lambda^n u(n)$$

b. Show the relationships:

$$\sum_{n=0}^{\infty} n h_a(n) = 0, \quad \sum_{n=0}^{\infty} n^2 h_a(n) = -\frac{2\lambda^2}{(1-\lambda)^2} \quad (6.25.1)$$

c. Show that the NRR of the filter $H_a(z)$ is:

$$\mathcal{R} = \sum_{n=0}^{\infty} h_a^2(n) = \frac{(1-\lambda)(1+4\lambda+5\lambda^2)}{(1+\lambda)^3}$$

d. Show that the magnitude response squared of $H_a(z)$ is:

$$|H_a(\omega)|^2 = \frac{(1-\lambda)^2 [1+2\lambda+5\lambda^2-4\lambda(1+\lambda)\cos\omega]}{[1-2\lambda\cos\omega+\lambda^2]^2} \quad (6.25.2)$$

e. Show that Eq. (6.25.2) has local minima at $\omega = 0$ and $\omega = \pi$, and a local maximum at $\omega = \omega_{\max}$:

$$\cos\omega_{\max} = \frac{1+4\lambda-\lambda^2}{2(1+\lambda)} \quad (6.25.3)$$

and that the corresponding extremal values are:

$$\begin{aligned} |H_a(0)|^2 &= 1, & |H_a(\pi)|^2 &= \frac{(1-\lambda)^2(1+3\lambda)^2}{(1+\lambda)^4} \\ |H_a(\omega_{\max})|^2 &= \frac{(1+\lambda)^2}{1+2\lambda} \end{aligned} \quad (6.25.4)$$

6.8 Consider the modified EMA of Eq. (2.3.5) and its twicing,

$$H(z) = \frac{(1-\lambda)(1+z^{-1})}{2(1-\lambda z^{-1})}, \quad H_a(z) = 2H(z) - H^2(z) = \frac{(1-\lambda)(1+z^{-1})(3+\lambda-z^{-1}(1+3\lambda))}{4(1-\lambda z^{-1})^2}$$

a. Show the relationships:

$$\sum_{n=0}^{\infty} n h_a(n) = 0, \quad \sum_{n=0}^{\infty} n^2 h_a(n) = -\frac{(1+\lambda)^2}{2(1-\lambda)^2}$$

b. Show that the NRR of the filter $H_a(z)$ is:

$$\mathcal{R} = \sum_{n=0}^{\infty} h_a^2(n) = \frac{1}{8}(1-\lambda)(3\lambda+7)$$

6.9 Consider the optimum length- N predictive FIR filter $h_\tau(k)$ of polynomial order $d = 1$ given by Eq. (6.4.1).

a. Show that its effective lag is related to the prediction distance τ by $\tilde{n} = -\tau$.

b. Show that its NRR is given by

$$\mathcal{R} = \sum_{k=0}^{N-1} h_\tau^2(k) = \frac{1}{N} + \frac{3(N-1+2\tau)^2}{N(N^2-1)}$$

Thus, it is minimized when $\tau = -(N-1)/2$. What is the filter $h_\tau(k)$ in this case?

c. Show that the second-derivative of its frequency response at DC is given by:

$$\frac{d^2}{d\omega^2} H(\omega)_{\omega=0} = -\sum_{n=0}^{\infty} k^2 h_\tau(k) = \frac{1}{6}(N-1)(N-2+6\tau)$$

Determine the range of τ s for which $|H(\omega)|^2$ is sloping upwards or downwards in the immediate vicinity of $\omega = 0$.

d. It is evident from the previous question that the value $\tau = -(N-2)/6$ corresponds to the vanishing of the second-derivative of the magnitude response. Show that in this case the filter is simply,

$$h(k) = \frac{3N(N-1)-2k(2N-1)}{N(N^2-1)}, \quad k = 0, 1, \dots, N-1$$

and verify explicitly the results:

$$\sum_{k=0}^{N-1} h(k) = 1, \quad \sum_{k=0}^{N-1} kh(k) = \frac{N-2}{6}, \quad \sum_{k=0}^{N-1} k^2 h(k) = 0, \quad \sum_{k=0}^{N-1} h^2(k) = \frac{7N^2-4N-2}{3N(N^2-1)}$$

e. Show that $h_\tau(k)$ interpolates linearly between the $\tau = 0$ and $\tau = 1$ filters, that is, show that for $k = 0, 1, \dots, N-1$,

$$h_\tau(k) = (1-\tau)h_a(k) + \tau h_1(k) = h_a(k) + [h_1(k) - h_a(k)]\tau$$

f. Another popular choice for the delay parameter is $\tau = -(N-1)/3$. Show that,

$$h(k) = \frac{2(N-k)}{N(N+1)}, \quad k = 0, 1, \dots, N-1$$

and that,

$$\sum_{k=0}^{N-1} h(k) = 1, \quad \sum_{k=0}^{N-1} kh(k) = \frac{N-1}{3}, \quad \sum_{k=0}^{N-1} k^2 h(k) = \frac{N(N-1)}{6}, \quad \sum_{k=0}^{N-1} h^2(k) = \frac{2(2N+1)}{3N(N+1)}$$

In financial market trading, the cases $\tau = -(N-1)/2$ and $\tau = -(N-1)/3$ correspond, respectively, to the so-called “simple” and “weighted” moving average indicators. The case $\tau = -(N-2)/6$ is not currently used, but it provides a useful compromise between reducing the lag while preserving the flatness of the passband. By comparison, the relative lags of three cases are:

$$\frac{1}{6}(N-2) < \frac{1}{3}(N-1) < \frac{1}{2}(N-1)$$

6.10 *Computer Experiment: Response of predictive FIR filters.* Consider the predictive FIR filter $h_\tau(k)$ of the previous problem. For $N = 9$, compute and on the same graph plot the magnitude responses $|H(\omega)|^2$ for the following values of the prediction distance:

$$\tau = -\frac{N-1}{2}, \quad \tau = -\frac{N-2}{6}, \quad \tau = 0, \quad \tau = 1$$

Using the calculated impulse response values $h_\tau(k)$, $0 \leq k \leq N-1$, and for each value of τ , calculate the filter lag, \tilde{n} , the NRR, \mathcal{R} , and the “curvature” parameter of Eq. (6.10.5). Recall from part (d) of the Problem 6.9 that the second τ should result in zero curvature.

Repeat all the questions for $N = 18$.

6.11 *Moving-Average Filters with Prescribed Moments.* The predictive FIR filter of Eq. (6.16.3) has lag equal to $\bar{n} = -\tau$ by design. Show that its second moment is not independently specified but is given by,

$$\overline{n^2} = \sum_{n=0}^{N-1} n^2 h(n) = -\frac{1}{6}(N-1)(N-2+6\tau) \quad (6.25.5)$$

The construction of the predictive filters (6.16.3) can be generalized to allow arbitrary specification of the first and second moments, that is, the problem is to design a length- N FIR filter with the prescribed moments,

$$\overline{n^0} = \sum_{n=0}^{N-1} h(n) = 1, \quad \overline{n^1} = \sum_{n=0}^{N-1} nh(n) = -\tau_1, \quad \overline{n^2} = \sum_{n=0}^{N-1} n^2 h(n) = \tau_2 \quad (6.25.6)$$

Show that such filter is given by an expression of the form,

$$h(n) = c_0 + c_1 n + c_2 n^2, \quad n = 0, 1, \dots, N-1$$

where the coefficients c_0, c_1, c_2 are the solutions of the linear system,

$$\begin{bmatrix} S_0 & S_1 & S_2 \\ S_1 & S_2 & S_3 \\ S_2 & S_3 & S_4 \end{bmatrix} \begin{bmatrix} \lambda_0 \\ \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} 1 \\ -\tau_1 \\ \tau_2 \end{bmatrix}$$

where

$$S_p = \sum_{n=0}^{N-1} n^p, \quad p = 0, 1, 2, 3, 4$$

Then, show that the S_p are given explicitly by,

$$S_0 = N, \quad S_1 = \frac{1}{2}N(N-1), \quad S_2 = \frac{1}{6}N(N-1)(2N-1) \\ S_3 = \frac{1}{4}N^2(N-1)^2, \quad S_4 = \frac{1}{30}N(N-1)(2N-1)(3N^2-3N-1)$$

and that the coefficients are given by,

$$c_0 = \frac{3(3N^2-3N+2)+18(2N-1)\tau_1+30\tau_2}{N(N+1)(N+2)} \\ c_1 = -\frac{18(N-1)(N-2)(2N-1)+12(2N-1)(8N-11)\tau_1+180(N-1)\tau_2}{N(N^2-1)(N^2-4)} \\ c_2 = \frac{30(N-1)(N-2)+180(N-1)\tau_1+180\tau_2}{N(N^2-1)(N^2-4)}$$

Finally, show that the condition $c_2 = 0$ recovers the predictive FIR case of Eq. (6.16.3) with second moment given by Eq. (6.25.5).

6.12 Consider the Butterworth filter of Eq. (6.20.2). Show that the lag of the first-order section and the lag of the i th second-order section are given by,

$$\bar{n}_0 = \frac{1}{2\Omega_0}, \quad \bar{n}_i = -\frac{\cos \theta_i}{\Omega_0}, \quad i = 1, 2, \dots, K$$

Using these results, prove Eq. (6.20.8) for the full lag \bar{n} , and show that it is valid for both even and odd filter orders M .

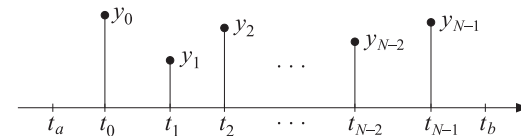
Smoothing Splines

7.1 Interpolation versus Smoothing

Besides their extensive use in drafting and computer graphics, splines have many other applications. A large online bibliography can be found in [350]. A small subset of references on interpolating and smoothing splines and their applications is [351-404].

We recall from Sec. 4.2 that the minimum- R_s filters had the property of maximizing the smoothness of the filtered output signal by minimizing the mean-square value of the s -differenced output, that is, the quantity $E[(\nabla^s \hat{x}_n)^2]$ in the notation of Eq. (4.2.11). Because of their finite span, minimum- R_s filters belong to the class of local smoothing methods. Smoothing splines are global methods in the sense that their design criterion involves the entire data signal to be smoothed, but their objective is similar, that is, to maximize smoothness.

We assume an observation model of the form $y(t) = x(t) + v(t)$, where $x(t)$ is a smooth trend to be estimated on the basis of N noisy observations $y_n = y(t_n)$ measured at N time instants t_n , for $n = 0, 1, \dots, N-1$, as shown below.



The times t_n , called the *knots*, are not necessarily equally-spaced, but are in increasing order and are assumed to lie within a slightly larger interval $[t_a, t_b]$, that is,

$$t_a < t_0 < t_1 < t_2 < \dots < t_{N-1} < t_b$$

A smoothing spline fits a continuous function $x(t)$, taken to be the estimate of the underlying smooth trend, by solving the optimization problem:

$$\mathcal{J} = \sum_{n=0}^{N-1} w_n (y_n - x(t_n))^2 + \lambda \int_{t_a}^{t_b} [x^{(s)}(t)]^2 dt = \min \quad (7.1.1)$$

where $x^{(s)}(t)$ denotes the s -th derivative of $x(t)$, λ is a positive “smoothing parameter,” and w_n are given non-negative weights.