CHAPTER 7

EXPLICIT LEAST-SQUARES METHODS

7.1 INTRODUCTION

Section 5.2.2 showed how a least-squares regression could be used to identify the IRF of a linear system. This was then replaced by a much more efficient computation based on correlation functions. Since the Wiener and Volterra series are essentially nonlinear generalizations of the linear IRF, one might expect that least-squares regressions could be used to identify them as well.

The correlation-based techniques described in Chapter 6 identified the kernels by solving a least-squares regression, but used the statistical properties of the input, assumed to be Gaussian white noise, to simplify the computations. This, however, meant that the accuracy of the kernel estimates depended on how closely the statistics of the finite-length data records approached those of the infinitely long, white Gaussian input assumed in the derivation. In this chapter, the least-squares regression will be solved explicitly, eliminating many of the assumptions on the input statistics and increasing the accuracy of the resulting models.

However, computational requirements and storage considerations make the direct application of least-squares techniques impractical for most systems. This chapter presents two different approaches that reduce the computational intensity of the least-squares solution to manageable proportions while maintaining the accuracy gained by solving the regression exactly.

7.2 THE ORTHOGONAL ALGORITHMS

The nature of the least-squares approach to identification may be understood by considering the identification of the simple, second-order Wiener–Bose model illustrated in

Identification of Nonlinear Physiological Systems, By David T. Westwick and Robert E. Kearney
ISBN 0-471-27456-9

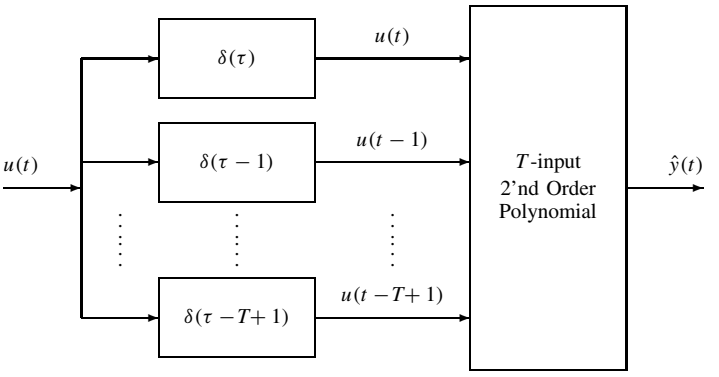

Figure 7.1 A Wiener–Bose model with a filter bank of pure delays. The orthogonal algorithms estimate the polynomial coefficients in the static nonlinearity using a linear least-squares regression.

Figure 7.1. The filter bank contains pure delays between 0 and $T-1$ samples so the output of the kth filter is $u(t-k+1)$. Then, from equation (4.63), the system output is given by

$$\hat{y}(t) = c^{(0)} + \sum_{\tau=0}^{T-1} c_{\tau}^{(1)} u(t-\tau) + \sum_{\tau_1=0}^{T-1} \sum_{\tau_2=\tau_1}^{T-1} c_{\tau_1,\tau_2}^{(2)} u(t-\tau_1)u(t-\tau_2) \tag{7.1}$$

Consequently, once the polynomial coefficients are known, the system will have been identified. Furthermore, equation (7.1) can be recognized as the "polynomial" form of a second-order Volterra series, [see equation (4.14)]. Thus, once the polynomial coefficients, $c^{(q)}$, have been estimated, the Volterra kernels can be determined using the procedure of Section 4.5.3 to yield a generalized nonlinear model.

The procedure described in Section 2.5.1 can be used to estimate the polynomial coefficients as follows. First, form the regression matrix, $\mathbf{U}$, with columns corresponding to the polynomial terms—that is, lagged input values and products of pairs of lagged input values. This matrix can be partitioned as

$$\mathbf{U} = \begin{bmatrix} \mathbf{U_0} & \mathbf{U_1} & \mathbf{U_2} \end{bmatrix} \tag{7.2}$$

where $\mathbf{U_0}$, $\mathbf{U_1}$, and $\mathbf{U_2}$ correspond to the zero-, first-, and second-order polynomial terms, respectively, applied to the input, $u(t)$. The submatrix $\mathbf{U_0}$ will comprise a single column containing the zero-order polynomial output,

$$\mathbf{U_0}(t) = u^0(t) = 1 \tag{7.3}$$

The submatrix $\mathbf{U_1}$ will have T columns, each containing a delayed copy of the input

$$\mathbf{U_1}(t,:) = \begin{bmatrix} u(t) & u(t-1) & \dots & u(t-T+1) \end{bmatrix} \tag{7.4}$$

In generating this matrix, $u(t)$ is assumed to be 0 for $t \leq 0$, so that column k, which contains $u(t-k+1)$, will have $k-1$ leading zeros.

The submatrix $\mathbf{U_2}$ will contain the second-order polynomial terms,

$$\begin{aligned}\mathbf{U_2}(t,:) = \big[&u^2(t)\ u(t)u(t-1)\ \ldots\ u(t)u(t-T+1)\\ &u^2(t-1)\ u(t-1)u(t-2)\ \ldots\ u^2(t-T+1)\big]\end{aligned} \tag{7.5}$$

where all but the first column will have one or more leading zeroes since $u(t)$ is assumed to be zero for $t \leq 0$.

The polynomial coefficients are stored in the vector,

$$\boldsymbol{\theta} = [c^{(0)}, c_0^{(1)}, \ldots, c_{T-1}^{(1)}, c_{0,0}^{(2)}, c_{0,1}^{(2)}, \ldots, c_{T-1,T-1}^{(2)}]^T \tag{7.6}$$

Therefore if $\hat{\boldsymbol{\theta}}$ contains estimates of the polynomial coefficients, the output of the model will given by $\hat{\mathbf{y}} = \mathbf{U}\hat{\boldsymbol{\theta}}$. This is a linear function of the parameter vector; thus the methods developed in Section 2.32, and in particular the *normal equations* (2.33), can be used to solve for the optimal $\hat{\boldsymbol{\theta}}$ and hence identify the system.

Formally, the identification problem to be solved can be stated as

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \left\{ (\mathbf{y} - \mathbf{U}\boldsymbol{\theta})^T (\mathbf{y} - \mathbf{U}\boldsymbol{\theta}) \right\} \tag{7.7}$$

that is, $\hat{\boldsymbol{\theta}}$ is the vector of polynomial coefficients that minimizes the sum of squared errors.

7.2.1 The Orthogonal Algorithm

Korenberg (1987; Korenberg et al., 1988a) proposed an (ordinary) Orthogonal Algorithm (OOA) that solved the normal equations by using modified Gram–Schmidt (MGS) orthogonalization (Golub and Van Loan, 1989) to compute the QR factorization of the regression matrix. The MGS factors U as

$$\mathbf{U} = \mathbf{QR} \tag{7.8}$$

where $\mathbf{Q}$ has orthonormal columns and $\mathbf{R}$ is upper triangular. The solution to the normal equations is then obtained from

$$\mathbf{R}\hat{\boldsymbol{\theta}} = \mathbf{Q^T z} \tag{7.9}$$

as can be seen by substituting equation (7.8) into the normal equation solution in equation (2.33). Thus,

$$\begin{aligned}\hat{\boldsymbol{\theta}} &= (\mathbf{U^T U})^{-1}\mathbf{U^T Z}\\ &= (\mathbf{R^T Q^T Q R})^{-1}\mathbf{R^T Q^T z}\\ &= \mathbf{R}^{-1}\mathbf{Q^T z}\end{aligned}$$

since $\mathbf{Q^T Q} = \mathbf{I}$. Finally, multiplying both sides by $\mathbf{R}$ produces equation (7.9), which can be easily solved using back substitution, since $\mathbf{R}$ is upper triangular.

The Volterra kernels can then be constructed from $\hat{\boldsymbol{\theta}}$ using the procedure outlined in Section 4.5.3. First, from equation (7.6), it is evident that $\hat{\boldsymbol{\theta}}(1)$ corresponds to $\hat{c}^{(0)}$ and thus determines the zero-order Volterra kernel, $\hat{\mathbf{h}}_\mathbf{0}$. Next note that, $\hat{\boldsymbol{\theta}}(2 \ldots T+1)$

contains the estimated first-order polynomial coefficients, $\hat{c}_0^{(1)}$ through $\hat{c}_{T-1}^{(1)}$. Thus, from equation (4.74), first-order kernel estimate will be

$$\hat{\mathbf{h}}_\mathbf{1}(\tau) = \hat{\boldsymbol{\theta}}(\tau + 1), \qquad 0 \leq \tau < T \tag{7.10}$$

The remaining elements of $\hat{\boldsymbol{\theta}}$ correspond to the second-order polynomial coefficients and can be used to construct the estimated second-order Volterra kernel. The procedure for extracting the second-order kernel elements from the parameter vector is more complex and is most easily explained by the following snippet of MATLAB code:

```
% T is the memory length, so offset points to the element
% just before the second-order polynomial coefficients
offset = T + 1;

% storage for the kernel
K2 = zeros(T,T);

% loop over the columns in K2,
% tri_length contains the length of the lower-triangular
% part of the column
tri_length = T;
for i = 1:T
  K2(i:T,i) = theta(offset+1:offset+tri_length);
  % make offset point at the next set of coefficients
  offset = offset + tri_length;
  tri_length = tri_length - 1;
end
%  now force the kernel to be symmetric
K2 = (K2 + K2')/2;
```

Each pass through the loop loads the subdiagonal part of one column in the kernel matrix with the appropriate polynomial coefficients. When the loop terminates, the matrix is added to its transpose and divided by two. Thus, each off-diagonal polynomial coefficient contributes to two symmetric kernel positions.

7.2.1.1 Computational and Storage Requirements

The total number of parameters that must be evaluated is

$$M = \binom{T+Q}{T} = \frac{(T+Q)!}{T!Q!} \tag{7.11}$$

where T is the system memory length, and Q is the maximum kernel order in the expansion. For second-order systems, $M = (T^2 + 3T + 2)/2$.

Storage is required for $\mathbf{U}$, the $N \times M$ element regression matrix, where N is the length of the data records. An efficiently written QR factorization routine could overwrite $\mathbf{U}$ with $\mathbf{Q}$.* Thus, for a second-order system, approximately $NT^2/2$ storage elements will be required.

*The implementation in the NLID toolbox (Kearney and Westwick, 2003), which is only intended as an illustration of the algorithm, stores both matrices and thus requires approximately twice as much storage.

The most-time consuming part of OOA is the Modified Gram–Schmidt QR factorization, which requires approximately $2NM^2$ flops (Golub and Van Loan, 1989). For a second-order kernel this becomes:

$$\tfrac{1}{2}N(T^2 + 3T + 2)^2$$

so the computational burden is approximately $NT^4/2$ flops provided that $N \gg T$.

For a third-order system the storage and computational requirements would be approximately $NT^3/3$ elements and $NT^6/18$ flops, respectively. This explosive growth in computational and storage requirements makes it impractical to apply the method to systems higher than second order and/or with long memory.

7.2.2 The Fast Orthogonal Algorithm

The OOA is quite general and makes no assumptions about the structure of the regression matrix, $\mathbf{U}$, defined by equations (7.2)–(7.5). Korenberg (1987) extended the algorithm to exploit the internal structure of $\mathbf{U}$ and eliminate redundant computations and storage. The resulting fast orthogonal algorithm (FOA) is much faster than the original OOA and requires much less storage.

The FOA does not use the QR factorization employed by the OOA, [see equation (7.9)]. Rather FOA solves the normal equations directly:

$$(\mathbf{U^T U})\hat{\boldsymbol{\theta}} = \mathbf{U^T z} \tag{7.12}$$

This direct approach makes it possible to:

1. Compute the Hessian, $\mathbf{U^T U}$, and projection coefficients, $\mathbf{U^T z}$ from high-order cross-correlation functions, without forming $\mathbf{U}$ explicitly.
2. Solve equation (7.12) efficiently via Cholesky decomposition of $\mathbf{U^T U}$ and back substitution.

The implementations of each of these will be discussed in the following sections.

7.2.2.1 Implicit Computation of $\mathbf{U^T U}$ The Hessian, $\mathbf{H} = \mathbf{U^T U}$, can be computed directly from the input, $u(t)$, without forming the regression matrix, $\mathbf{U}$. This results from its structure, given in equation (7.2),

$$\mathbf{U} = \begin{bmatrix}\mathbf{U_0}\ \mathbf{U_1}\ \mathbf{U_2}\end{bmatrix}$$

where $\mathbf{U_0}$, $\mathbf{U_1}$, and $\mathbf{U_2}$, are defined in equations (7.3) through (7.5). Note that $\mathbf{U_0}$ is the first column of $\mathbf{U}$, $\mathbf{U_1}$ takes up the next T columns (columns 2 through $T + 1$), and $\mathbf{U_2}$ fills the remaining $(T^2 + T)/2$ columns.

This Hessian may be partitioned similarly as

$$\mathbf{H} = \mathbf{U^T U} = \left[\begin{array}{c|c|c} \mathbf{U_0^T U_0} & \mathbf{U_0^T U_1} & \mathbf{U_0^T U_2} \\ \hline \mathbf{U_1^T U_0} & \mathbf{U_1^T U_1} & \mathbf{U_1^T U_2} \\ \hline \mathbf{U_2^T U_0} & \mathbf{U_2^T U_1} & \mathbf{U_2^T U_2} \end{array}\right] \tag{7.13}$$

In principle, this partitioning scheme may be extend to the regression matrices and Hessians for third- and higher-order systems, although in practice the computations may

not be feasible. Consequently, the following discussion will be limited to second-order systems.

Korenberg showed that each partition of the Hessian may be expressed in terms of an input autocorrelation functions of order 0 to 3 (the input mean to $\phi_{uuuu}(\tau_1, \tau_2, \tau_3)$), plus a correction term.

Top Left Block The top left block in the Hessian, equation (7.13), is given by

$$\mathbf{H}(1, 1) = \mathbf{U_0^T U_0} = \mathbf{U}(:, 1)^T \mathbf{U}(:, 1) = N$$

since $\mathbf{U_0}$, the first column of the regression matrix, contains all ones [see equation (7.3)]. This point is the exception, in that it does not depend on the input.

Top Center Block Next, consider the top middle block of the Hessian, $\mathbf{U_0^T U_1}$. The first element in this block is

$$\begin{aligned}\mathbf{H}(1, 2) &= \mathbf{U_0^T U_1}(:, 1) \\ &= \mathbf{U}(:, 1)^T \mathbf{U}(:, 2)\end{aligned}$$

Recall, from equation (7.4), that the first column of $\mathbf{U_1}$, which is also the second column of the regression matrix, $\mathbf{U}(:, 2)$, contains the input, $u(t)$. Thus,

$$\begin{aligned}\mathbf{H}(1, 2) &= \sum_{t=1}^{N} 1 \cdot u(t) \\ &= N\mu_u\end{aligned}$$

Note that μ_u may be regarded as the zero-order autocorrelation of the signal.

The remaining columns of $\mathbf{U_1}$ contain delayed copies of the input. For example, the second column of $\mathbf{U_1}$, which is $\mathbf{U}(:, 3)$, contains a leading 0, followed by the first $N - 1$ elements of $u(t)$. Thus,

$$\begin{aligned}\mathbf{H}(1, 3) &= \mathbf{U}(:, 1)^T \mathbf{U}(:, 3) \\ &= 0 + \sum_{t=1}^{N-1} 1 \cdot u(t) \\ &= N\mu_u - u(N) \\ &= \mathbf{H}(1, 2) - u(N)\end{aligned}$$

Similarly, $\mathbf{U}(:, 4)$ contains two leading zeros, followed by the first $N - 2$ values of $u(t)$. This is also equivalent to a single leading zero, followed by the first $N - 1$ entries in $\mathbf{U}(:, 3)$. Consequently,

$$\begin{aligned}\mathbf{H}(1, 4) &= \sum_{t=1}^{N-2} 1 \cdot u(t) \\ &= \mathbf{H}(1, 3) - u(N - 1)\end{aligned}$$

The remaining $T + 1$ elements in top middle block of the Hessian can be computed using the recursive relation

$$\mathbf{H}(1, k + 1) = \mathbf{H}(1, k) - u(N - k + 1), \qquad 2 \leq k \leq T$$

Note also that the Hessian is symmetric. In particular, the middle left block in equation (7.13) is

$$\mathbf{U_1^T U_0} = (\mathbf{U_0^T U_1})^T$$

which is the transpose of the top middle block.

Top Right Block The top right block in the Hessian contains $\mathbf{U_0^T U_2}$. Since $\mathbf{U_0}$ is a single column containing all ones, this block will contain the average values of the second-order terms: products of pairs of delayed copies of the input. These time averages, in turn, may be computed by applying small corrections to the first-order input autocorrelation function.

For example, the first element in the upper right block is

$$\begin{aligned}\mathbf{H}(1, T+2) &= \sum_{t=1}^{N} 1 \cdot u^2(t) \\ &= N\hat{\phi}_{uu}(0)\end{aligned}$$

where $\hat{\phi}_{uu}(\tau)$ is a biased estimate of the input autocorrelation obtained using equation (2.18).

The next element, $\mathbf{H}(1, T+3)$, is the product of $\mathbf{U_0}$ and the second column of $\mathbf{U_2}$. From equation (7.5), this column contains a leading zero, followed by $u(t)u(t-1)$, for $t = 2 \ldots N$. Thus, $\mathbf{H}(1, T+3)$ is given by

$$\begin{aligned}\mathbf{H}(1, T+3) &= \sum_{t=2}^{N} 1 \cdot u(t)u(t-1) \\ &= N\hat{\phi}_{uu}(1)\end{aligned}$$

Similarly, the third column of $\mathbf{U_2}$ contains two leading zeros, followed by $u(t)u(t-2)$, for $t = 3 \ldots N$. Thus, the third element in the upper right block of the Hessian is given by

$$\begin{aligned}\mathbf{H}(1, T+4) &= \sum_{t=3}^{N} 1 \cdot u(t)u(t-2) \\ &= N\hat{\phi}_{uu}(2)\end{aligned}$$

Given the structure of $\mathbf{U}$, this pattern applies to the first T elements of the Hessian block,

$$\mathbf{H}(1, T+k+2) = N\hat{\phi}_{uu}(k), \qquad k = 0 \ldots T-1$$

The next $T-1$ columns of $\mathbf{U_2}$ can be obtained by delaying its first $T-1$ columns by one time step [see equation (7.5)]. This requires adding a leading zero to each column and deleting the Nth element. Thus, the corresponding elements in the Hessian can be obtained by subtraction:

$$\begin{aligned}\mathbf{H}(1, 2T+2) &= \sum_{t=1}^{N-1} u^2(t) \\ &= \mathbf{H}(1, T+2) - u^2(N)\end{aligned}$$

Similarly, the next $T-2$ columns are obtained by delaying the first $T-2$ columns by two time steps, producing corrections involving the product of two elements. For example,

$$\begin{aligned}\mathbf{H}(1, 2T+3) &= \sum_{t=2}^{N-1} u(t)u(t-1) \\ &= \mathbf{H}(1, T+3) - u(N)u(N-1)\end{aligned}$$

Similar recursive procedures, based on correcting biased correlation estimates, can be used to fill in the rest of the Hessian. The middle block, $\mathbf{U_1^T U_1}$ in equation (7.13), contains products of the form $u(t-i+2)u(t-j+2)$, which can be computed from $\hat{\phi}_{uu}$. Moreover, these computations have already been used to compute the elements in the $\mathbf{U_0^T U_2}$ block. For example,

$$\begin{aligned}\mathbf{H}(2,2) &= \mathbf{U}(:,2)^T \mathbf{U}(:,2) \\ &= \mathbf{U_1}(:,1)^T \mathbf{U_1}(:,1) \\ &= \sum_{t=1}^{N} u^2(t) \\ &= N\hat{\phi}_{uu}(0)\end{aligned}$$

A horizontal movement across the Hessian gives

$$\begin{aligned}\mathbf{H}(2,3) &= \mathbf{U}(:,2)^T \mathbf{U}(:,3) \\ &= \sum_{t=2}^{N} u(t)u(t-1) \\ &= N\hat{\phi}_{uu}(1)\end{aligned}$$

Whereas a diagonal movement results in

$$\begin{aligned}\mathbf{H}(3,3) &= \mathbf{U}(:,3)^T \mathbf{U}(:,3) \\ &= \sum_{t=2}^{N} u^2(t-1) \\ &= N\hat{\phi}_{uu}(0) - u^2(N)\end{aligned}$$

Thus, all entries in the middle block can be computed by correcting the biased autocorrelation estimate, $\hat{\phi}_{uu}$.

The remainder of rows 2 through $T+2$ contain products of first-order and second-order terms. These values can be obtained by applying similar end corrections to $\hat{\phi}_{uuu}$. In the same way, terms involving the product of two second-order regressors may be computed from $\hat{\phi}_{uuuu}$ with appropriate corrections.

As a result of these manipulations, the complete Hessian can be computed by first calculating the mean and first-, second-, and third-order autocorrelations of the input and then applying small corrections, to compensate for end effects. This requires substantially fewer computations than generating the Hessian directly by forming the regression matrix, $\mathbf{U}$, and then multiplying it by its transpose.

7.2.2.2 Implicit Computation of U^{Tz} The projection coefficients, $U^T z$, that make up the right-hand side of equation (7.12) may also be computed without explicitly generating the regression matrix. To see this, partition the regression matrix,

$$\mathbf{U^T z} = \left[\begin{array}{c} \mathbf{U_0^T z} \\ \hline \mathbf{U_1^T z} \\ \hline \mathbf{U_2^T z} \end{array}\right] \tag{7.14}$$

The first block in equation (7.14) is

$$\begin{aligned} \mathbf{U_0^T z} &= \sum_{t=1}^{N} 1 \cdot z(t) \\ &= N\mu_z \end{aligned}$$

The second block, $\mathbf{U_1^T z}$, can be obtained from the first-order input–output cross-correlation, as follows:

$$\begin{aligned} \mathbf{U_1}(:,k)^T \mathbf{z} &= \sum_{t=1}^{N} u(t-k+1)z(t), \qquad k = 1 \ldots T \\ &= N\hat{\phi}_{uz}(k-1) \end{aligned}$$

Similarly, the values in the third block, $\mathbf{U_2^T z}$, may be obtained from the second-order input–output cross-correlation. For example,

$$\begin{aligned} \mathbf{U_2}(:,k)^T \mathbf{z} &= \sum_{t=1}^{N} u(t)u(t-k-1)z(t), \qquad k = 1 \ldots T \\ &= N\hat{\phi}_{uuz}(0,k-1) \end{aligned}$$

and

$$\begin{aligned} \mathbf{U_2}(:,T+k)^T \mathbf{z} &= \sum_{t=1}^{N} u(t-1)u(t-k)z(t), \qquad k = 1 \ldots T-1 \\ &= N\hat{\phi}_{uuz}(1,k) \end{aligned}$$

Note that there are no corrections required in these computations. The projection coefficients are calculated directly from the output mean and first- and second-order input–output cross-correlations.

7.2.2.3 Efficient Computation of Correlations The cost of computing the Hessian and projection coefficients may be reduced further by using efficient methods to compute the auto- and cross-correlations. Typically, the first-order cross-correlation is computed most efficiently by FFT-based methods (Bendat and Piersol, 1986; Press et al., 1992). However, the FFT approach has not been extended to higher-order correlation functions, and so these must be computed in the time domain. For example, a straightforward method of computing the second-order cross-correlation is illustrated by the

following MATLAB code*:

```
for i = 0:T-1
  for j = 1:i
    sum = 0;
    for n = i+1:N
      sum = sum + z(n)*u(n-i)*u(n-j);
    end
    phi(i+1,j+1) = sum/N;
    phi(j+1,i+1) = phi(i+1,j+1);
  end
end
```

Note that the two outer loops index through $T(T+1)/2$ unique values of the second-order cross-correlation. Each element in the correlation is computed in the innermost loop that comprises $2N$ multiplications and N additions. Consequently, the entire computation requires approximately $\frac{3}{2}NT^2$ flops.

Korenberg (1991) demonstrated that by rearranging the loops it is possible to cut the number of computations almost in half. He noted that the innermost loop computes the product `z(n)*u(n-i)` repeatedly, once for each value of `j`. Reversing the order of the loops eliminates these redundant multiplications. Thus, with this scheme, the second-order cross-correlation would be computed as follows:

```
for n = 1:N
  for i = 0:T-1
    if n > i
      temp = z(n)*u(n-i);
      for j = 0:i
        phi(i+1,j+1) = phi(i+1,j+1) + temp*u(n-j);
      end
    end
  end
end
%  Divide all entries by N, and place in symmetric positions
for i = 1:T
  for j = 1:i
    phi(i,j) = phi(i,j)/N;
    phi(j,i) = phi(i,j);
  end
end
```

This algorithm does add some extra "overhead," but reduces the cost of the main step by a factor of $\frac{3}{2}$, since the innermost loop now only involves a single multiplication and addition. This approach can also be applied to higher-order cross-correlations, where the savings are even greater.

*For practical applications, MATLAB is not well-suited for this computation since only the innermost loop can be vectorized. The outer two loops are unavoidable. The implementation in the NLID toolbox was written in C, compiled and the executable linked to MATLAB.

7.2.2.4 Solving for $\hat{\theta}$ via Cholesky Factorization

Once the Hessian, $\mathbf{H} = \mathbf{U}^{\mathbf{T}}\mathbf{U}$, and projection coefficients, $\mathbf{U}^{\mathbf{T}}\mathbf{z}$, have been computed using the efficient, cross-correlation-based techniques discussed above, it remains to solve the normal equations (7.12):

$$\mathbf{H}\hat{\boldsymbol{\theta}} = \mathbf{U}^{\mathbf{T}}\mathbf{z} \tag{7.15}$$

The Hessian is an $M \times M$, positive definite matrix and therefore its Cholesky factorization,

$$\mathbf{H} = \mathbf{R}^{\mathbf{T}}\mathbf{R} \tag{7.16}$$

where $\mathbf{R}$ is an upper triangular matrix, can be computed in about $M^3/3$ flops. Once this has been accomplished, equation (7.15) can be solved using two back-substitution steps as follows. Let

$$\mathbf{R}\boldsymbol{\psi} = \mathbf{R}^{\mathbf{T}}\mathbf{R}\hat{\boldsymbol{\theta}} \tag{7.17}$$

so that equation (7.15) becomes

$$\mathbf{R}\boldsymbol{\psi} = \mathbf{U}^{\mathbf{T}}\mathbf{z} \tag{7.18}$$

Solve this for $\boldsymbol{\psi}$ and then solve

$$\mathbf{R}^{\mathbf{T}}\hat{\boldsymbol{\theta}} = \boldsymbol{\psi} \tag{7.19}$$

for $\hat{\boldsymbol{\theta}}$. Since $\mathbf{R}$ is upper triangular, the back-substitution steps required to solve equations (7.18) and (7.19) will require about M^2 flops each.

7.2.2.5 Computational and Memory Requirements

The total number of parameters to be estimated with the FOA is the same as for the OOA:

$$M = \binom{T+Q}{T} = \frac{(T+Q)!}{T!Q!}$$

where T is the system memory length, and Q is the maximum kernel order in the expansion. What resources are required to estimate these parameters?

The storage requirements will be dominated by two components:

1. The input–output data itself, requiring $2N$ elements.
2. The Hessian matrix, $\mathbf{H}$, with $M \times M$ elements. For a second-order system, this will be approximately $T^4/4$ elements.

For second-order systems, the computational cost will be determined by two steps:

1. The third-order autocorrelation, ϕ_{uuuu}, used to compute the Hessian, $\mathbf{H}$. Computing this with the procedure outlined in the previous section (Korenberg, 1991) will require approximately $NT^3/3$ flops.
2. The Cholesky factorization required to extract the $\mathbf{R}$ matrix from $\mathbf{H}$. Since the Hessian is a $M \times M$ matrix, Cholesky factorization requires $M^3/3$ flops (Golub and Van Loan, 1989).

Thus, the dominant computations in the second-order FOA require about

$$\frac{M^3}{3} + \frac{NT^3}{3}$$

flops. However, for second-order systems, $M = (T^2+3T+2)/2$ so that provided $N \gg T$, the overall computational cost will be approximately $NT^3/3$, a factor of $\frac{3}{2}T$ smaller than that for the explicit orthogonal algorithm. Note, however, that the computational cost will also increase with T^6; this term will be significant if T^3 is similar in size to N. Consider, for example, a system with $T = 16$ and $N = 10{,}000$ data points; here $T^3 = 13{,}824$, and both terms will be significant.

7.2.3 Variance of Kernel Estimates

A major benefit of estimating kernels with least-squares regression is that the extensive theory regarding the statistical properties of regression, summarized in Section 2.4.2, can be applied directly to the kernel estimates. This section will demonstrate how these results can be used to determine confidence bounds for kernel estimates and predictions.

For this analysis, it is assumed that the system can be represented exactly by a set of kernels with the same maximum order and memory length as the estimates. Thus, the underlying system must have fading memory, and the kernel order and memory length must be large enough to describe the system's response. In particular, there must be a parameter vector, $\boldsymbol{\theta}$, such that the *noise-free* output of the system is

$$\mathbf{y} = \mathbf{U}\boldsymbol{\theta}$$

Furthermore, it is assumed that the output is corrupted by a zero-mean, white Gaussian noise process, $v(t)$,

$$z(t) = y(t) + v(t)$$

with variance σ_v^2, which is independent of the input, $u(t)$. Under these conditions, equation (2.41) can be used to estimate the parameter covariance matrix:

$$\mathbf{C}_{\hat{\boldsymbol{\theta}}} = \hat{\sigma}_v^2(\mathbf{U}^{\mathbf{T}}\mathbf{U})^{-1} = \hat{\sigma}_v^2\mathbf{H}^{-1} \tag{7.20}$$

where

$$\hat{\sigma}_v^2 = \frac{1}{N-M}\sum_{t=1}^{N}(z(t) - \hat{y}(t))^2 \tag{7.21}$$

is an unbiased estimate of the residual variance.

7.2.3.1 Variance of First-Order Kernel Estimates The elements of the first-order kernel estimate are given by elements $2 \rightarrow T+1$ of the parameter estimate vector, $\hat{\boldsymbol{\theta}}$. Thus, the variance in the first-order kernel estimate is given by elements $2 \rightarrow T+1$ of $\mathbf{C}_{\hat{\boldsymbol{\theta}}}$, defined in equation (7.20).

$$\mathrm{Var}(\hat{\mathbf{h}}^{(\mathbf{1})}(\tau)) = \mathbf{C}_{\hat{\boldsymbol{\theta}}}(\tau+1) \tag{7.22}$$

7.2.3.2 Variance of the Second-Order Kernel Estimate The relation between the variances of the elements of the second-order kernel and that of the parameter vector is more complex. Diagonal elements in the kernel correspond directly to elements in the

parameter vector. The variance of these kernel elements can be read directly from $\mathbf{C}_{\hat{\boldsymbol{\theta}}}$. However, each remaining parameter contributes equally to two symmetric, off-diagonal kernel values. The corresponding kernel values are half the parameter values, and consequently their variance is only one-fourth that of the parameter. Thus, to obtain the variance of the off diagonal kernel elements, the corresponding entries in $\mathbf{C}_{\hat{\boldsymbol{\theta}}}$ must be divided by 4. The following MATLAB code illustrates the computation.

```
% H is the Hessian, Cov is the covariance,
% v-hat are the residuals N is the data length,
% M is the number of parameters
ssresid = sum(vhat.^2)/(N-M);
Cov = (ssresid/N)*diag(inv(H));
% m points to the entry in the parameter vector
% just before the second-order kernel.
m = T+1;
% Now loop through the kernel, and step
% through the covariance,
for i = 1:hlen
  % point to the next element in the parameter vector
  % (which will correspond to the next value on the
  % kernel diagonal.
  m = m + 1;
  h2var(i,i) = Cov(m);
  for j = i+1:hlen
    % point to the next element in the parameter vector.
    % We are now stepping away from the kernel diagonal.
    m = m + 1;
    h2var(i,j) = Cov(m)/4;
    h2var(j,i) = Cov(m)/4;
  end
end
```

7.2.3.3 Variance of the Model Output The model output is a linear function of its parameters and so its variance may also be estimated using least-squares theory. Consider a linear combination of parameters given by

$$f(\hat{\boldsymbol{\theta}}) = \boldsymbol{\gamma}^T \hat{\boldsymbol{\theta}}$$

for any $\boldsymbol{\gamma} \in \Re^M$, where M is the number of parameters. The variance of the function $f(\hat{\boldsymbol{\theta}})$, due to the uncertainty in the parameters, $\hat{\boldsymbol{\theta}}$, can be computed from

$$\begin{aligned} \text{Var}\,(\boldsymbol{\gamma}^T \hat{\boldsymbol{\theta}}) &= E[\boldsymbol{\gamma}^T \hat{\boldsymbol{\theta}} - E[\boldsymbol{\gamma}^T \hat{\boldsymbol{\theta}}])(\boldsymbol{\gamma}^T \hat{\boldsymbol{\theta}} - E[\boldsymbol{\gamma}^T \hat{\boldsymbol{\theta}}])^T] \\ &= \boldsymbol{\gamma}^T E[(\hat{\boldsymbol{\theta}} - E[\hat{\boldsymbol{\theta}}])(\hat{\boldsymbol{\theta}} - E[\hat{\boldsymbol{\theta}}])^T]\boldsymbol{\gamma} \\ &= \boldsymbol{\gamma}^T \mathbf{C}_{\hat{\boldsymbol{\theta}}} \boldsymbol{\gamma} \end{aligned} \tag{7.23}$$

To see how this result is used, consider $u(t)$, an arbitrary input signal, and $\mathbf{U}$, the corresponding regression matrix. The predicted model output at time t will be given by

$$\hat{y}(t) = \mathbf{U}(t, :)\hat{\boldsymbol{\theta}}$$

which may be seen to be a linear combination of the model parameters. Consequently, its variance will be

$$\sigma^2_{\hat{y}(t)} = \mathbf{U}(t,:)\mathbf{C}_{\hat{\boldsymbol{\theta}}}\mathbf{U}(t,:)^T$$

7.2.4 Example: Fast Orthogonal Algorithm Applied to Simulated Fly Retina Data

The application of the FOA will now be demonstrated by applying it to simulated data from a model of signal processing in the fly retina. This example will be used throughout the remainder of the book to illustrate key techniques. The simulation is based on a model identified experimentally by Juusola et al. (1995); it consists of an LNLN block-structured model that relates fluctuations in the light intensity incident on the fly retina to the transmembrane voltage of a large monopolar cell (LMC). Results from the experimental study will be summarized at the end of Chapter 8 and will provide an interesting comparison to these simulations.

Figure 7.2 shows the model structure comprising a linear IRF, representing photoreceptor transduction, followed by an NLN cascade, characterizing the transformation between the photoreceptor and LMC nucleus. The two static nonlinearities were described by fourth-order Tchebyshev polynomials, and the second linear element was described by another IRF.

Two datasets were generated, each consisting of 8192 points sampled at 1 kHz. The first simulation used a white Gaussian noise input to represent ideal conditions. The second dataset corresponded more closely to an actual experiment; the input was colored noise generated by low-pass filtering a white Gaussian signal with a fourth-order elliptical filter having a cutoff frequency of 250 Hz. In both cases, white Gaussian noise was added to the output to represent measurement noise; this was scaled so that the SNR was approximately 13 dB. Figure 7.3 shows 200-ms segments of simulated data from the white- and colored-input datasets.

In all simulations, the first 8000 points were used for identification, reserving the remaining 192 points for cross-validation. To avoid producing transients in the relatively short validation segment, the whole 8192-point input record was processed by the identified models. The last 192 points of the resulting output were then removed and used for validation.

The first three Volterra kernels, of orders 0 through 2, were computed for both the white and colored datasets described above. Application of the FOA requires setting the kernel memory length, T, a priori. This was estimated by fitting a long (100-ms) linear IRF between the input and output; the resulting IRF decayed to zero after about 40 ms. Consequently, to be conservative the memory length of the Volterra series was set to 50 ms. The kernels and their variances were then estimated as described above.

Figure 7.4 shows estimates of the first- and second-order kernels using the FOA with the white-input dataset. Figure 7.4A shows the first-order kernel; note that it decays to zero, indicating that the 50-ms memory length was adequate. The dotted lines surrounding the first-order kernel correspond to bounds of three standard deviations from the mean. If the errors in the kernel estimates are zero-mean and Gaussian, then there is a 99.7% probability that the true kernel lies between these bounds. These bounds are very close to the kernel estimate indicating that it is very accurate.

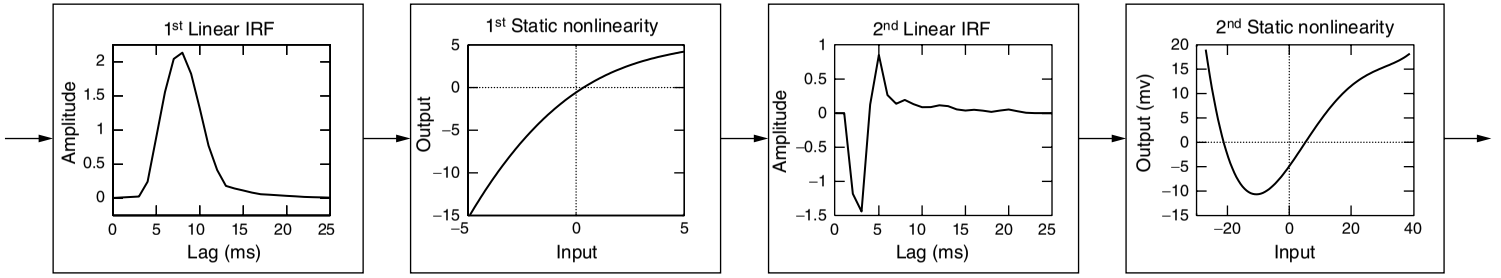

Figure 7.2 Elements of the LNLN model of the fly retina, used in the examples in Chapters 7 and 8.

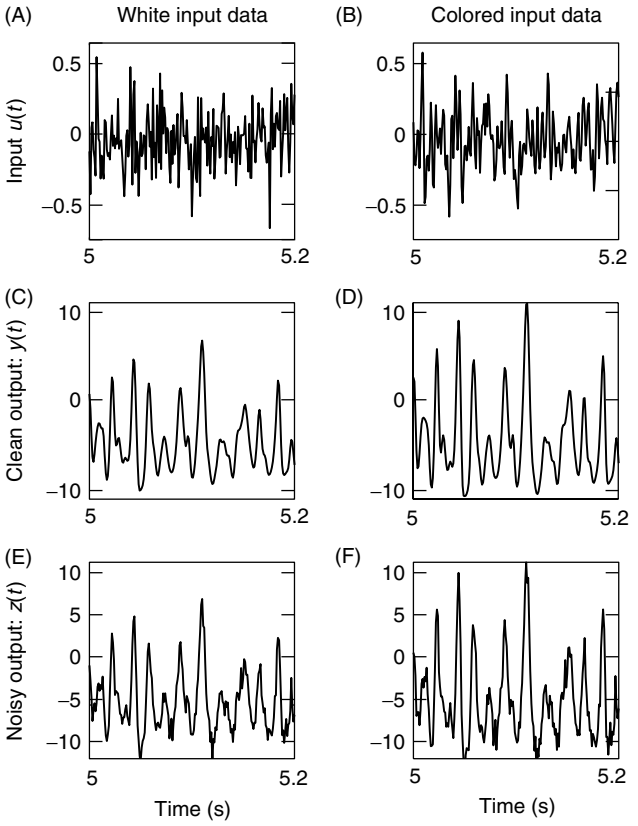

Figure 7.3 Two hundred milliseconds of simulated data from the fly retina model, used in the examples in Chapters 7 and 8.

Figures 7.4B and 7.4C show estimates of second-order kernel and its standard deviation. (Note that the standard deviation could be used to construct confidence bounds on the values of the second-order kernel estimate, but the resulting display is overly complex.) The standard deviation of the second-order kernel is small and almost constant everywhere except along the diagonal, where it is about $\sqrt{2}$ larger, corresponding to a doubling of the estimation variance. This is to be expected, since the diagonal values are obtained directly from the estimated coefficients, whereas symmetric pairs of off-diagonal elements are obtained by dividing a single polynomial coefficient by two. The derivation of this effect is left as an exercise to the reader, in Problem **4**.

The second-order Volterra series model, identified using the FOA, had a prediction accuracy of 95.8% variance accounted for (% VAF) in the identification dataset and had a 94.6% VAF in the cross-validation segment. Furthermore, its VAF was 98.7% for the noise-free output in the cross-validation segment. These are impressive results, since the actual model was an NLN cascade with two fourth-order polynomial nonlinearities and should therefore require eighth-order Volterra series representation. Despite this, most of the output power was accounted for by the first- and second-order kernels. Note, however, that this would change if the input power was increased, because the higher-order kernels would then become more significant.

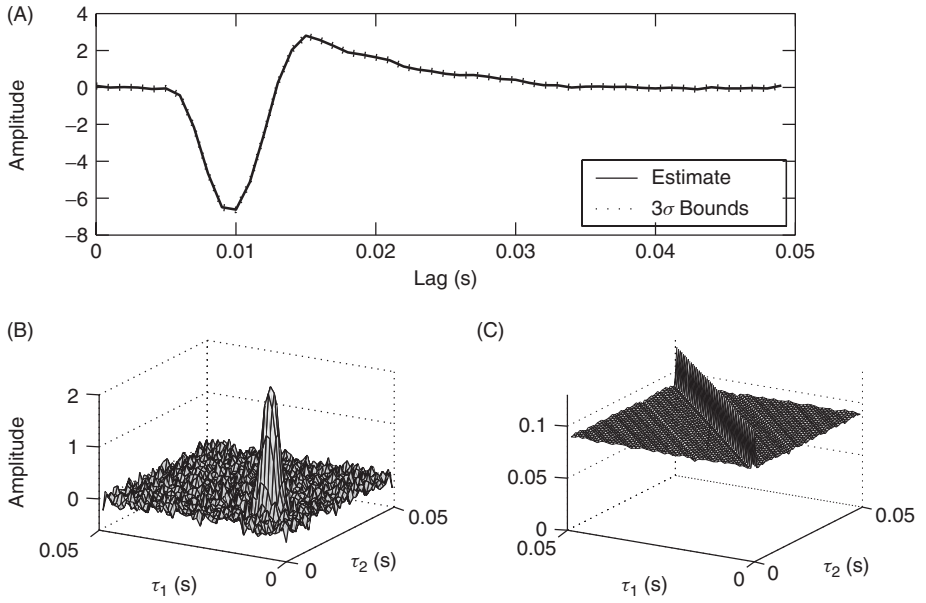

Figure 7.4 White-noise estimates of the first- and second-order Volterra kernels of the fly retina model estimated using the fast orthogonal algorithm. (A) The first-order kernel estimate (solid) bracketed by 3σ confidence bounds. (B) Second-order kernel estimate. (C) Estimate of the standard deviation for the second-order kernel estimate in B.

Next, the same analysis was applied to the colored input data. Figure 7.5 shows the resulting estimates of the first- and second-order kernels and their variance. Both kernel estimates display high-frequency noise. In the first-order kernel (Figure 7.5A), these are evident as small fluctuations that are most visible where the kernel does not change rapidly. This uncertainty in the kernel estimate is reflected in the error bounds, shown as dotted lines surrounding the estimate, which are much larger than those for the white input.

The effects are more dramatic in the second-order kernel estimate shown in Figure 7.5B. The shape of the kernel has been completely obscured by high-frequency noise (compare Figures 7.4B and 7.5B). Indeed, from the standard deviations in the kernel estimates, shown in Figures 7.4C and 7.5C, the uncertainty appears to have increased by a factor of at least 10. Indeed, for the colored input, the maximum value of the standard deviation was 1.5 times larger than that of the kernel estimated from the white input. Furthermore, the distribution of uncertainty is not uniform, as for the white noise data; the standard deviation is still greatest along the diagonal but decreases progressively with movement toward the edges.

Although the kernel estimates were noisy, they still predicted the system output very accurately; the VAF was 94.6% and 92.8% for the identification and validation segments, respectively. Apparently, the bandwidth of the input was sufficient to excite all system dynamics. On the other hand, the noise in the kernels occurred at frequencies where there was little input power or significant system dynamics. Thus, the noise in the kernel estimates had little effect on the model accuracy, at least for inputs having the same spectra as that used for the identification.

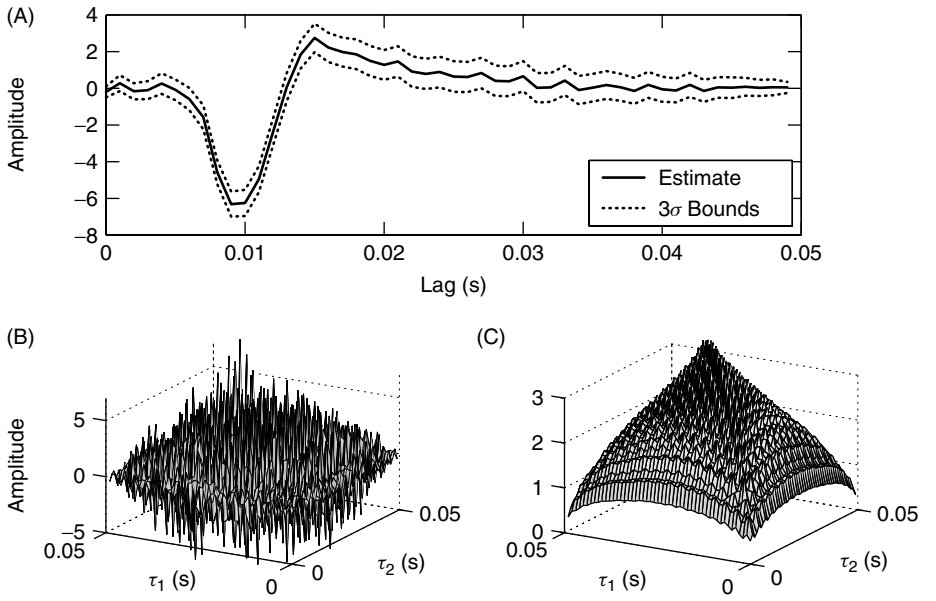

Figure 7.5 Colored-noise estimates of the first- and second-order Volterra kernels of the fly retina model estimated using the fast orthogonal algorithm. (A) The first-order kernel estimate and confidence bounds. (B) Second-order kernel estimate. (C) Estimated standard deviation of the second-order kernel estimate shown in B.

Noise in a kernel estimate often obscures its shape. Consequently, smoothing is often used to reduce noise and reveal the underlying structure. The first-order kernel is often smoothed with a three-point, zero-phase filter, with the symmetric, two-sided impulse response

$$h_{smo}(\tau) = \begin{cases} 0.5, & \tau = 0 \\ 0.25, & \tau = \pm 1 \\ 0, & \text{otherwise} \end{cases}$$

Similarly, the appearance of the second-order kernel may be improved by applying this filter to both the rows and columns. Figure 7.6 shows FOA kernels estimated from the colored noise data after one and two passes of this smoothing filter. Smoothing certainly improves the appearance of the both kernels, although the effect on the second kernel is more dramatic. Note, however, that smoothing may change the model's response and reduce its predictive abilities. Indeed, after two smoothing passes, the model's VAF dropped to 92.1% VAF for both the training and cross-validation segments.

7.2.5 Application: Dynamics of the Cockroach Tactile Spine

French and co-workers used the FOA to examine the dynamics of the cockroach tactile spine. This is a mechanoreceptor with a single sensory neuron that normally fires in response to movement of the tactile spine. However, in their experiments, the sensory neuron was stimulated by applying broadband electrical current directly to its membrane.

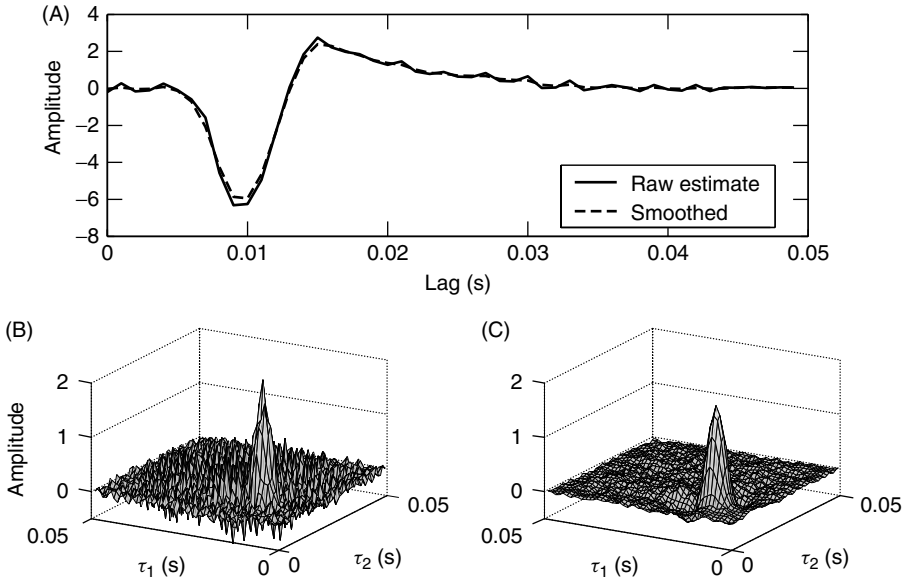

Figure 7.6 Effect of smoothing on the first- and second-order Volterra kernel estimates, obtained by the FOA from the colored-noise data. (A) Unsmooothed first-order kernel (solid) and after a single smoothing (dashes). (B) Second-order kernel estimate smoothed once. (C) Second-order kernel estimate smoothed twice.

The resulting action potentials were detected with a Schmidt trigger resulting in a binary output signal with a 1 indicating an action potential somewhere in the sampling interval.

Their first study (Korenberg et al., 1988b) used an input with a 50-Hz bandwidth and a sampling rate of 125 Hz. The FOA was used estimate the first three Volterra kernels between the input (membrane) current and output (action-potential train). Figure 7.7 shows the resulting second-order kernel. The only significant values appear to lie along the diagonal, suggesting that the underlying system has a Hammerstein structure, a memoryless nonlinearity followed by a dynamic linear system. Further support for this inference was provided by a comparison of the first-order kernel with the diagonal of the second-order kernel. As Figure 7.8 shows, they were very similar as would be expected for a Hammerstein system.

Subsequently, French and Korenberg (1989, 1991) repeated their experiments with a higher temporal resolution by increasing both the input bandwidth (250 Hz) and the sampling rate (500 Hz). Figures 7.9 and 7.10 show the first- and second-order kernels estimated with this higher resolution. The second-order kernel no longer appears to be diagonal as it did in their earlier lower-resolution study. Rather, the kernel shapes are consistent with those of a LNL system.

7.3 EXPANSION BASES

Amorocho and Brandstetter (1971) were actually the first to recast the kernel estimation problem as a linear least-squares regression. They felt that applying the approach directly would require computations that were prohibitively expensive for most practical

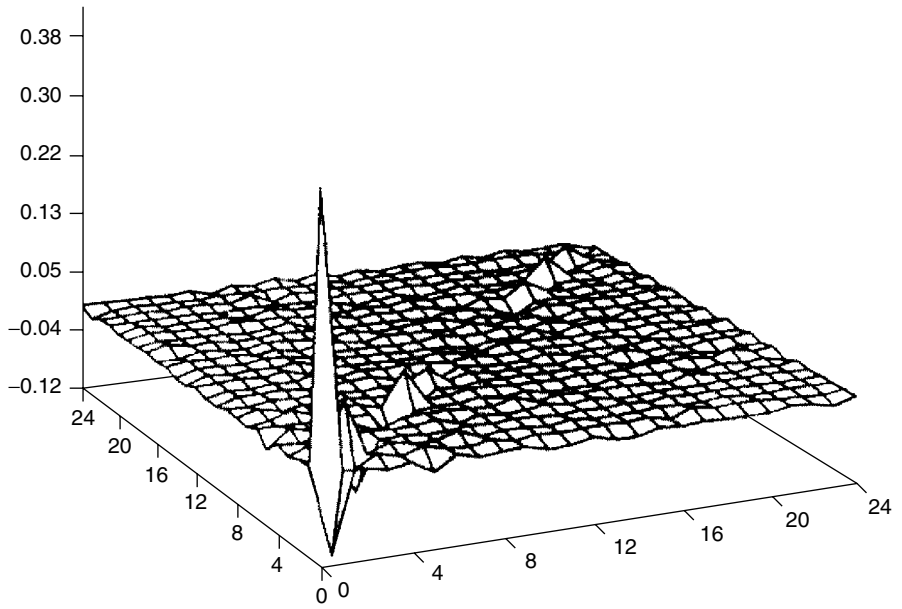

Figure 7.7 Second-order kernel of the cockroach tactile spine estimated using the fast orthogonal algorithm. From Korenberg et al. (1988b). Used with permission.

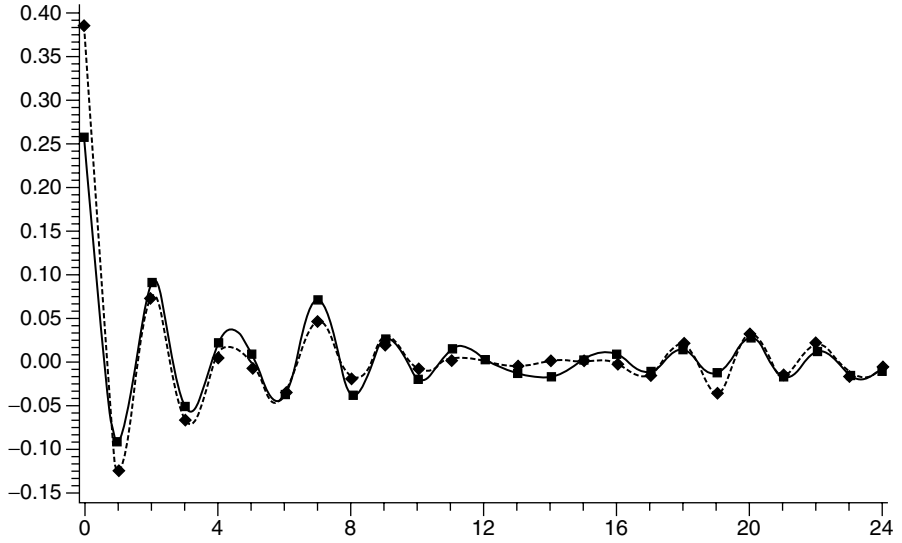

Figure 7.8 The square of the first-order kernel superimposed on the diagonal of the second-order kernel. From Korenberg et al. (1988b). Used with permission.

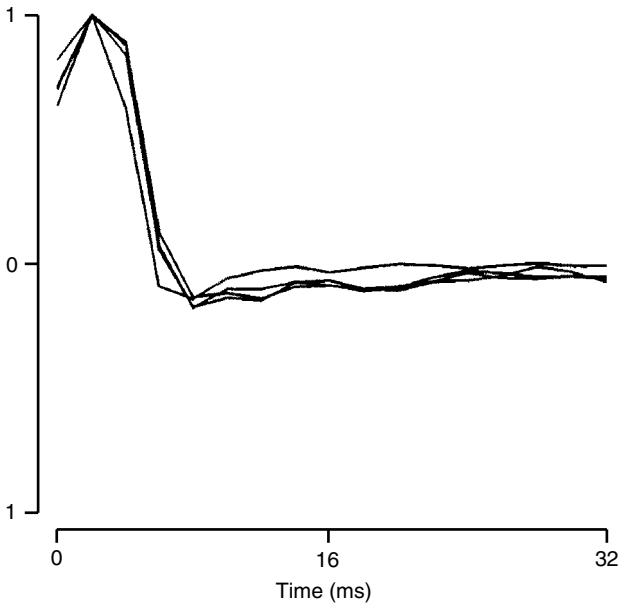

Figure 7.9 First-order kernel estimates of the cockroach tactile spine obtained at a sampling frequency of 500 Hz. The four curves were obtained from four separate cockroaches. From French and Korenberg (1991). Used with permission of BMES.

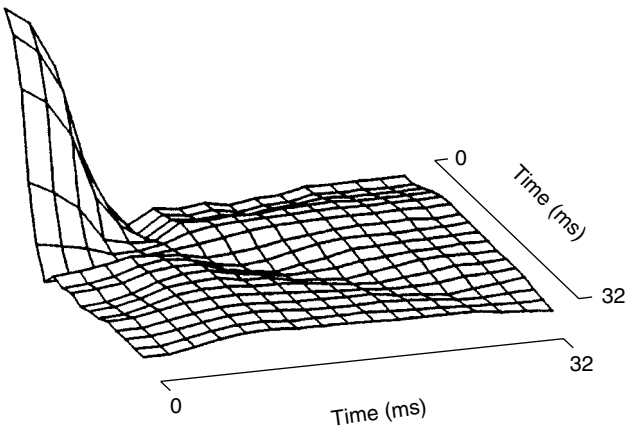

Figure 7.10 Second-order kernel of a cockroach tactile spine. Since the system is causal and free of feedback loops, only the first quadrant ($\tau_1, \tau_2 \geq 0$) is shown. From French and Korenberg (1991). Used with permission of BMES.

applications. They suggested that these difficulties could be alleviated by using a basis expansion to reduce the number of polynomial coefficients in the regression.

Recall, from Section 7.2, that the orthogonal algorithms were based on the identification of a Wiener–Bose model. As shown in Figure 7.1, the Wiener–Bose model had a

filter bank consisting of pure delays, $h_k(\tau) = \delta(\tau - 1 - k)$. This led to a linear least-squares problem in which the regressors were lagged samples of the input and products of lagged samples of the input. In computing the Volterra kernels from the polynomial coefficients, each polynomial coefficient contributed to a kernel either a delta function along the diagonal or a set of symmetric off-diagonal delta functions. This provided a very general, but not particularly efficient, expansion basis.

Section 4.5.1 showed that the Wiener–Bose model was unique to within a (nonsingular) linear transformation of the matrix $\mathbf{G}$, containing the IRFs in the filter-bank defined in equation (4.64). In the orthogonal algorithms, where the filter bank consists of pure delays, $\mathbf{G}$ is the $T \times T$ identity matrix. However, any nonsingular $T \times T$ matrix could be used. This raises the question, Is it possible to choose a set of filters, or equivalently a matrix $\mathbf{G}$, such that all polynomial coefficients associated with one or more of the filters are small enough to be insignificant? If so, filters that do not contribute significantly to the output can be removed, thereby simplifying the model without reducing its accuracy.

7.3.1 The Basis Expansion Algorithm

The following algorithm estimates the zero- through second-order Volterra kernels of a system. Identification using the basis expansion algorithm (BEA) proceeds much as for the orthogonal algorithm, except that basis elements replace the delayed impulses in the filter bank. As with the orthogonal algorithm, the generalization to higher-order kernels is straightforward and involves adding third- and higher-order polynomial terms to the regression matrix constructed in step 3.

1. Choose a suitable set of filters, $h_k(\tau)$, $k = 1 \ldots P$, to use as an expansion basis.
2. Compute the outputs of the filters via convolution.

$$x_k(t) = \sum_{\tau=0}^{T-1} h_k(\tau) u(t - \tau) \qquad \text{for } k = 1 \ldots P$$

3. Construct the regression matrix,

$$\mathbf{X}(t, :) = \left[1 \; x_1(t) \; \ldots \; x_P(t) \; x_1^2(t) \; x_1(t)x_2(t) \; \ldots \; x_P^2(t)\right] \tag{7.24}$$

4. Find the MMSE solution to the linear regression

$$\mathbf{z} = \mathbf{X}\boldsymbol{\theta}$$

5. Use equations (4.74) and (4.75) to construct the kernels from $\hat{\boldsymbol{\theta}}$.

Step 1 of this algorithm leads to a classical "chicken and egg" problem. The objective is to choose the "best" basis expansion to identify the system. However, doing so requires a knowledge of what the system is. Consequently, making the proper choice becomes a matter of experience, trial and error, and some luck.

7.3.1.1 Computational Requirements Assuming that an appropriate expansion basis has been chosen, consider the computational cost of the basis expansion algorithm. Computing the filter outputs (see step 2) requires $2NT$ flops to compute for each of P convolutions for a total of $2NTP$ flops.

The bulk of the computational burden is associated with solving the least-squares problem. The regression matrix has one row per data point and has one column for each parameter in the model. The total number of model parameters is

$$M = \frac{(P+Q)!}{P!Q!} \tag{7.25}$$

Recall that the MGS orthogonalization, used to compute the basis expansion, requires approximately $2NM^2$ flops (Golub and Van Loan, 1989). Thus for kernels of order 0 through 2, the computational cost is approximately $\frac{1}{2}NP^4$. This formula is similar to that for the orthogonal algorithm, except that the memory length, T, has been replaced with the number of basis elements, P. Hence this technique is most useful where $P \ll T$.

7.3.2 The Laguerre Expansion

This section will discuss an expansion basis that has proven to be useful in practice—the orthogonal Laguerre filters. The basis of discrete Laguerre filters comprises the IRFs defined by

$$h_k(\tau) = \alpha^{(\tau-k)/2}(1-\alpha)^{1/2}\sum_{i=0}^{k}(-1)^i\binom{\tau}{i}\binom{k}{i}\alpha^{k-i}(1-\alpha)^i, \qquad \tau \geq 0 \tag{7.26}$$

These IRFs are dependent on a single parameter, α, often termed the "decay parameter." Figure 7.11 shows the IRFs of the first five Laguerre filters for $\alpha = 0.25$, a value often used in system identification.

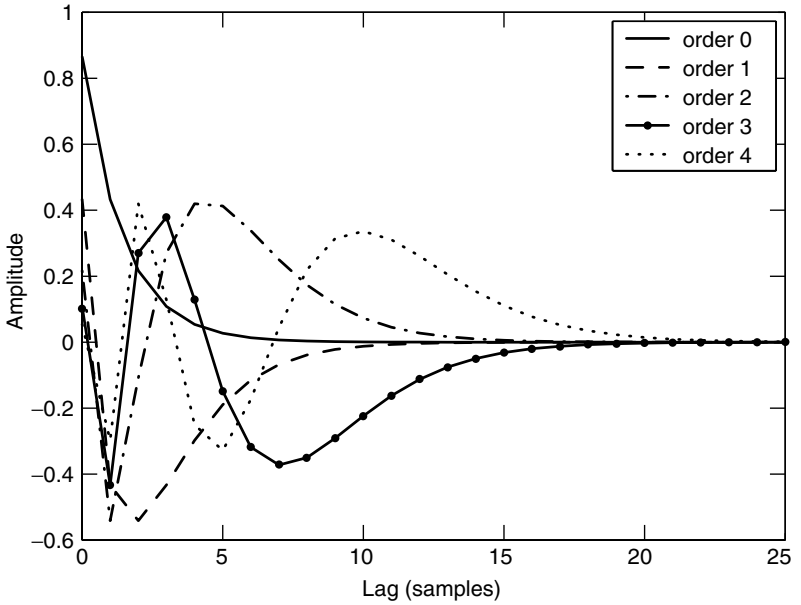

Figure 7.11 The IRFs of the Laguerre filters of order 0 through 4 with decay parameter $\alpha = 0.25$.

The Laguerre filters have three desirable properties.

1. The IRFs are orthogonal and thus should lead to a well-conditioned estimation problem.
2. The IRFs decay exponentially to zero as τ goes to infinity. Thus, if the system kernels decay smoothly to zero, as is often the case, it may be possible to capture their dynamics with only a few basis functions.
3. The outputs of the Laguerre filters can be computed efficiently (Ogura, 1986), by recursively applying the same filter. Thus, the zero-order filter output is obtained using the relation

$$x_0(t) = \sqrt{\alpha}x_0(t-1) + \sqrt{1-\alpha}u(t), \qquad x_0(0) = 0 \tag{7.27}$$

where $u(t)$ is the input. The output of filter k can be obtained by filtering the output filter $k-1$ filter with

$$x_k(t) = \sqrt{\alpha}x_k(t-1) + \sqrt{\alpha}x_{k-1}(t) - x_{k-1}(t-1), \qquad x_k(0) = 0 \tag{7.28}$$

Identification using the Laguerre expansion technique (LET) uses the basis expansion algorithm, described in Section 7.3.1, with a basis of Laguerre filters, described in equation (7.26), as the filter bank. In practice, the filter outputs, computed in step 2 of the algorithm, are often computed recursively, using equations (7.27) and (7.28), instead of with convolutions. The rest of the algorithm proceeds unchanged.

Practical application of the Laguerre expansion technique requires the selection of two variables: the decay parameter, α, and the number of basis elements, P. Methods for choosing these parameters are discussed in the next two subsections.

7.3.3 Limits on α

A well-chosen expansion basis can dramatically reduce the number of polynomial coefficients to be determined. However, selecting the appropriate basis set is not trivial. For the Laguerre filters, two parameters must be chosen a priori: α, the decay parameter, and P, the number of basis elements. Together these completely determine the IRFs of the elements in the filter bank.

Consider the zero-order Laguerre filter, whose output is computed using equation (7.27). In the z domain, this filter can be represented as a first-order low-pass filter,

$$H_0(z) = \frac{\sqrt{1-\alpha}}{1-\sqrt{\alpha}z^{-1}}$$

From equation (7.28), it is evident that the output of the kth Laguerre filter may be obtained by filtering the output of the $(k-1)$th filter with the z-domain transfer function

$$H_{ap}(z) = \frac{\sqrt{\alpha}-z^{-1}}{1-\sqrt{\alpha}z^{-1}}$$

which has unit gain at all frequencies. Hence, the transfer function of the order k Laguerre filter is

$$H_k(z) = \big(H_{ap}(z)\big)^k H_0(z)$$

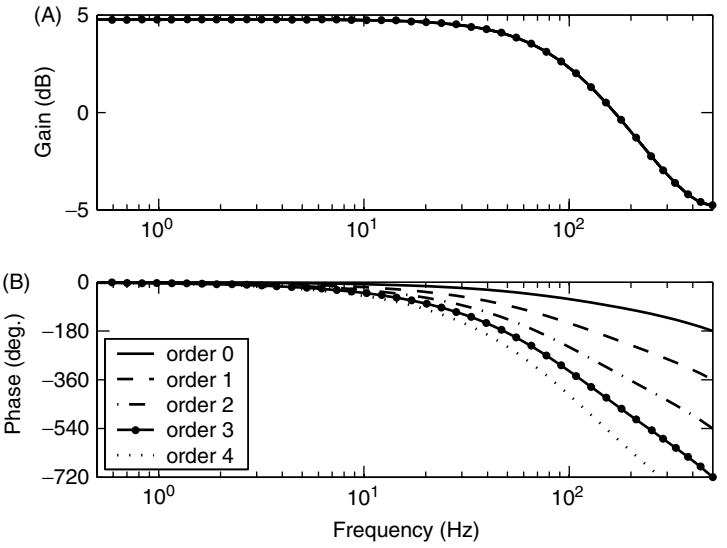

Figure 7.12 Bode diagrams for the discrete Laguerre filters of order 0 through 4 with decay parameter $\alpha = 0.25$, and sampling frequency f_s = 1000 Hz.

Thus, all discrete Laguerre filters have identical first-order, low-pass gain characteristics. However, as a result of the repeated action of the all-pass filter, each filter has a distinct phase characteristic. Figure 7.12 shows the gain and phase characteristics of the IRFs shown in Figure 7.11.

Choosing the decay parameter, α, appropriately, can avoid the numerical ill-conditioning that may occur when least-squares estimates are obtained using colored inputs (see Sections 5.2.3 and 7.2.4). Suppose the test input contains no significant power beyond a frequency $f = B$; further assume that this bandwidth is sufficient to excite all of the system's dynamics. Limiting the bandwidth of all components of the identified model to that of the input would avoid numerical problems since the model would then only have dynamic modes that were excited adequately.

The Laguerre expansion accomplishes this as follows. Recall that the Laguerre filters consist of a first-order, low-pass filter, with a pole at $z = \sqrt{\alpha}$, followed by zero or more all-pass filters. The corner frequency for the single, discrete-time pole is

$$\cos(\beta) = \cos\left(\frac{2\pi f}{f_s}\right) = \frac{-1 + 4\sqrt{\alpha} - \alpha}{2\sqrt{\alpha}}$$

where β is the normalized frequency ($0 \leq \beta \leq 2\pi$). Substituting for the input bandwidth, $f = B$, and solving for α suggests that for the identification to be well-conditioned numerically, α should be greater than

$$\alpha \geq \left((2 - \cos(\beta)) - \sqrt{\cos^2(\beta) - 4\cos(\beta) + 3}\right)^2 \tag{7.29}$$

where β is given by

$$\beta = 2\pi \frac{B}{f_s}$$

7.3.4 Choice of α and P

Marmarelis (1993) provided guidelines for choosing the values of the decay parameter, α, and the number of basis functions in the expansion, P. He observed that the Laguerre filters formed a "fan" whose size depends on α. Figure 7.13 shows the first 50 lags of the 50 Laguerre filters with $\alpha = 0.2$. Outside the fan, the filter IRFs are insignificant. Thus, the fan indicates the number of filters required to represent systems with a particular memory length. The memory of the filter bank, as a whole, must be at least as long as that of the system, but should not be significantly greater because this may lead to estimation problems. Thus, given the memory length of the system, T, and a decay parameter, α, chosen using equation (7.29), P should be chosen such that the point (T, P) is near the edge of the "fan." Thus, one could require that

$$h_P(T) \leq 0.01 \tag{7.30}$$

which means that the last filter in the basis has decayed nearly to zero at the anticipated memory length. Alternately, given a desired number of basis elements, this heuristic can be used to determine the value of α required to cover the perceived memory length. It has been implemented in the NLID toolbox (Kearney and Westwick, 2003) routine `ch_alpha.m`, which is based on the routine `calcAlpha` from the Lysis (Marmarelis, and Courellis, 2003) software package.

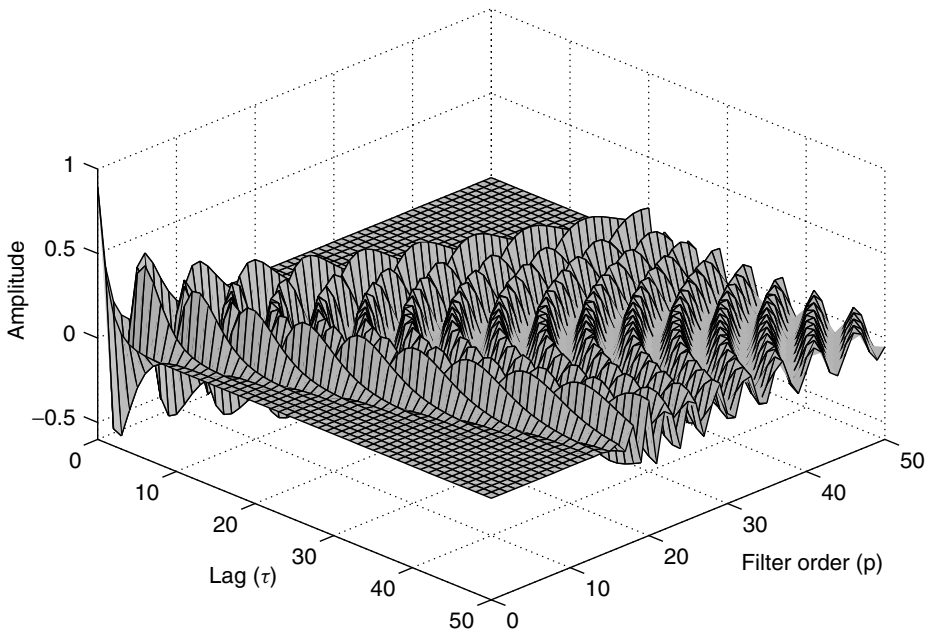

Figure 7.13 The first 50 lags of the first 50 Laguerre filters for $\alpha = 0.2$. Redrawn from Marmarelis, (1993) and used with permission of the Biomedical Engineering Society.

7.3.5 The Laguerre Expansion Technique

The Laguerre expansion technique (LET) can be summarized as follows:

1. Estimate the input bandwidth, and use inequality (7.29) to compute a minimum value for the α decay parameter.
2. Choose P, the number of basis elements to be used in the expansion, and then determine the corresponding value of α. If α is less than the minimum determined in step 1, reduce the number of basis elements.
3. Given P and α, use the BEA, described in Section 7.3.1, to generate a Laguerre kernel expansion.

If a priori information regarding the appropriate number of basis elements is not available, one could conceivably start at $P = 1$ and continue to increase P until the decay parameter becomes smaller than the minimum specified by inequality (7.29). A parametric model order selection test, such as minimizing the MDL criterion defined in equation (5.17), could then be used to determine the number of Laguerre filters.

7.3.6 Computational Requirements

If the number of basis elements is chosen a priori, then the computational burden will be equal to a single application of the BEA, about $\frac{1}{2}NP^4$ flops for a second-order nonlinearity. If several candidate bases are evaluated, the computational burden could be several times higher. To deal with this, an implicit basis expansion algorithm (Westwick and Lutchen, 2000) has been developed to accelerate the computation of multiple basis expansions. It uses the FOA to compute the Hessian and projection coefficients, on a basis of delayed impulses, and then projects these onto the expansion basis before solving the regression. This separates the steps that depend on the data length from those that depend on the number of basis elements.

7.3.7 Variance of Laguerre Kernel Estimates

Section 7.2.3, developed variance estimates for kernels estimated using the FOA. These estimates were based on the variance of linear regression parameters. Furthermore, equation (7.23) provided an expression for the variance of a linear function of the estimated parameters.

From equations (4.74) and (4.75), it is evident that any element of a kernel can be written as a weighted sum of the parameters. Consequently, equation (7.23) may be used to estimate the variance of Laguerre expansion kernels.

For example, the first-order kernel is given by

$$\hat{\mathbf{h}}^{(\mathbf{1})}(\tau) = \sum_{k=1}^{P} \hat{c}_k^{(1)} h_k(\tau)$$

where $c_k^{(1)}$ is the first-order polynomial coefficient associated with the kth Laguerre basis filter, $h_k(\tau)$. This may be rewritten as

$$\hat{\mathbf{h}}^{(\mathbf{1})}(\tau) = \boldsymbol{\gamma}(1, \tau)^T \hat{\boldsymbol{\theta}} \tag{7.31}$$

where $\boldsymbol{\gamma}(1, \tau)$ is the vector that generates the first-order kernel at lag τ and is given by

$$\boldsymbol{\gamma}(1, \tau)^T = \left[0\ h_1(\tau) \ldots h_p(\tau)\ 0 \ldots 0\right]$$

Substituting this into equation (7.23) gives the variance of the first-order kernel estimate at lag τ:

$$\text{Var}\ (\hat{\mathbf{h}}^{(1)}(\tau)) = \boldsymbol{\gamma}^T(1, \tau)\mathbf{C}_{\hat{\boldsymbol{\theta}}}\boldsymbol{\gamma}(1, \tau)$$

A similar expression may be derived for the variance of the second-order Laguerre expansion kernel, by rewriting equation (4.75) in the same form as equation (7.31) and then substituting the resulting vector, $\boldsymbol{\gamma}(2, \tau_1, \tau_2)$, into equation (7.23).

7.3.8 Example: Laguerre Expansion Kernels of the Fly Retina Model

The Laguerre Expansion Technique (LET), described in Section 7.3.5, will now be demonstrated by applying it to the white-noise dataset from the fly retina model. First, an upper bound on the memory length of the system must be determined. Section 7.2.4 demonstrated that a value of $T = 50$ ms was adequate when the FOA was used with the same data.

The input was white, with inequality (7.29) suggesting a minimum value of $\alpha_m = 0.03$ corresponding to $P = 28$. Thus, Laguerre expansions were evaluated for P ranging from 1 to 28 and the model that minimized the MDL was selected. This resulted in a model with $P = 12$ Laguerre basis elements and a decay parameter $\alpha = 0.2$. Figure 7.14 shows the resulting kernel estimates and their variances.

The variance in the second-order kernel is highest near zero lag (on both axes) and lowest at the longest lags. This occurs because of the shape of the Laguerre basis

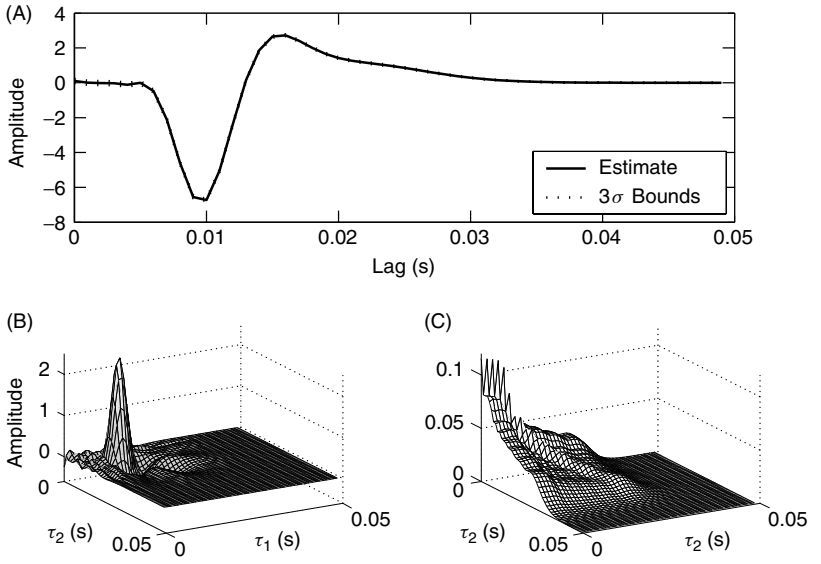

Figure 7.14 White-noise estimates of the first- and second-order Volterra kernels of the fly retina model estimated using the Laguerre expansion technique. (A) First-order kernel estimate (solid) between 3σ confidence bounds. (B) Second-order kernel estimate. (C) Estimated standard deviation of the second-order kernel estimate.

elements; as Figure 7.11 shows, all basis elements contribute significantly to the kernels at low lags, but the number of significant basis elements decreases as the lag increases. Therefore, the variance near the origin will have relatively large contributions from all polynomial coefficients. However, as the lag increases, fewer polynomial coefficients will make significant contributions to the kernel and hence to the estimation variance.

Note that for this example, indeed for any system with a delay, the greatest uncertainty in the kernel estimate occurs during the initial lags, where the actual kernel values are zero. This problem can be remedied by introducing a delay into the Laguerre basis elements. The delay can be chosen by examining the initial kernel estimates and the corresponding uncertainties.

For this example, the first-order kernel estimate (Figure 7.14), is close to zero for lags of 0–7 ms. Similarly, comparing the second-order kernel and the estimate of its uncertainty, shown in Figures 7.14B and 7.14C, it is evident that when both τ_1 and τ_2 are less than about 5 ms, the kernel value is within 3σ of zero. Thus, neither estimated kernel is significantly different from zero for lags between 0 and about 5 ms. Therefore at any given time, the output is not significantly dependent on the past 5 ms of the input, and setting the kernel to zero at these lags is unlikely to reduce the model accuracy significantly. In practice, it is safest to try several delays in this range and then to choose the longest delay that does not adversely affect the model accuracy. Once the delay has been determined, 6 ms in this case, the values of P and α must be recomputed to minimize the MDL. In this case, incorporating a delay of 6 ms into the Laguerre basis resulted in $P = 9$ and $\alpha = 0.3$. Figure 7.15 shows the kernels estimated with a delayed Laguerre basis.

Note that the variance in both the first- and second-order kernels is exactly 0 for lags less than or equal to 6 ms, since all of the delayed basis elements are zero for these

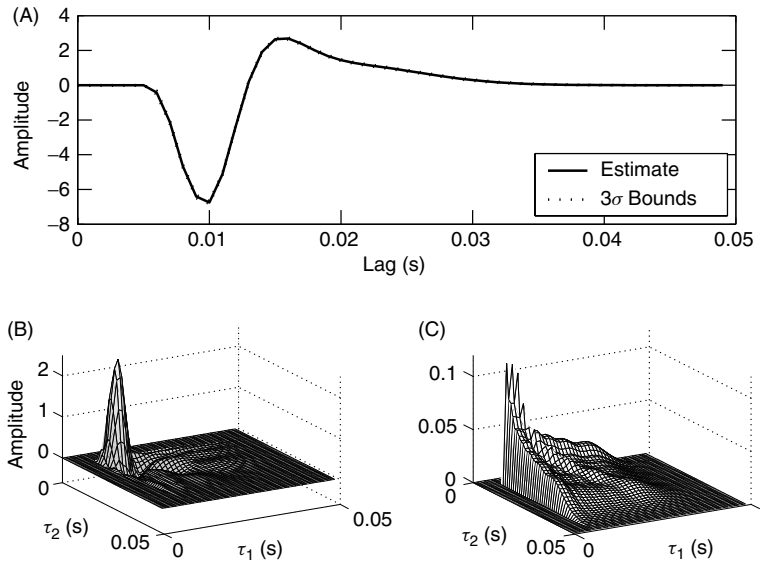

Figure 7.15 White-noise estimates of the first- and second-order Volterra kernels of the fly retina model estimated using the Laguerre expansion technique with a delay of 6 ms. (A) Estimate of the first-order kernel (solid) between 3σ confidence bounds. (B) Second-order kernel estimate. (C) Estimated standard deviation of the second-order kernel estimate shown in B.

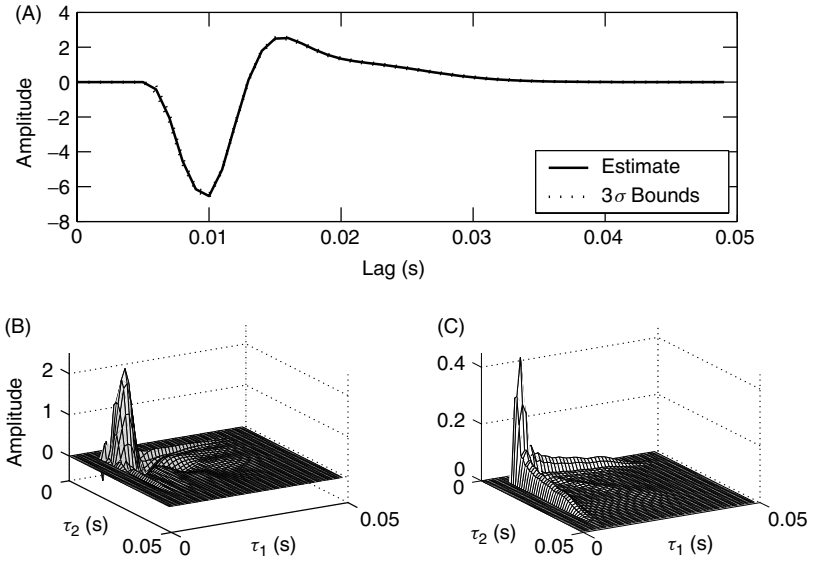

Figure 7.16 Colored-noise estimates of the first- and second-order Volterra kernels of the fly retina model estimated using the Laguerre expansion technique with a delay of 6 ms. (A) First-order kernel estimate (solid) between 3σ confidence bounds. (B) Estimate of the second-order kernel. (C) Estimated standard deviation of the second-order kernel estimate in B.

lags. In both cases, the variance is largest immediately after the delay since all Laguerre functions contribute.

LET was applied to the colored input data, which had a bandwidth of 250 Hz. This corresponds to a normalized corner frequency of

$$\beta = 2\pi \frac{250}{1000} = \frac{\pi}{2}$$

giving the minimum value of $\alpha = 0.07$ to ensure that the estimation problem remains well-conditioned. The parameters used for the white noise data ($\alpha = 0.3$, with a delay of 6 ms) easily satisfy this requirement; consequently they were used with the colored noise input as well.

Estimates of the kernels, and their uncertainties, derived from the colored-noise data are shown in Figure 7.16. Note that the standard deviation of the second-order kernel (Figure 7.16C) is about four times higher than for the white-noise dataset, shown in Figure 7.15C. However, because all dynamics in the filter bank were strongly excited by the input, the estimation problem remained well-conditioned. This is apparent by comparing the present results with those obtained with the FOA Figure 7.5C where the standard deviation is more than 10 times greater.

7.4 PRINCIPAL DYNAMIC MODES

Section 4.5.1 showed that the filter bank of a Wiener–Bose model is not unique. The matrix containing the filters may be multiplied by a (nonsingular) square matrix without

affecting its ability to represent the system since the inverse transformation may be incorporated into the nonlinearity. Indeed, this observation underlies the use of expansion bases as discussed in Section 7.3. This section presents another approach whereby the expansion basis is chosen *after* estimating the Volterra kernels; the objective is to find the minimal set of filters required to represent the system (Marmarelis, 1994; Marmarelis and Orme, 1993).

The development proceeds as follows. Let $\hat{\mathbf{W}}$ be a matrix containing estimates of the zero- through second-order Volterra kernels,

$$\hat{\mathbf{W}} = \begin{bmatrix} \hat{\mathbf{h}}_\mathbf{0} & \frac{1}{2}\hat{\mathbf{h}}_\mathbf{1}^T \\ \frac{1}{2}\hat{\mathbf{h}}_\mathbf{1} & \hat{\mathbf{h}}_\mathbf{2} \end{bmatrix} \tag{7.32}$$

Construct a vector containing lagged samples of the input, augmented with a leading 1:

$$\begin{aligned} \mathbf{u_a^T(t)} &= [1u(t)u(t-1)\ldots u(t-T+1)] \\ &= [1\mathbf{u^T(t)}] \end{aligned} \tag{7.33}$$

The output of the zero- through second-order kernels can be written as the quadratic:

$$\hat{y}(t) = \mathbf{u_a^T(t)}\hat{\mathbf{W}}\mathbf{u_a(t)} \tag{7.34}$$

$$= \hat{\mathbf{h}}_\mathbf{0} + \tfrac{1}{2}\hat{\mathbf{h}}_\mathbf{1}^T\mathbf{u(t)} + \tfrac{1}{2}\mathbf{u^T(t)}\hat{\mathbf{h}}_\mathbf{1} + \mathbf{u^T(t)}\hat{\mathbf{h}}_\mathbf{2}\mathbf{u(t)} \tag{7.35}$$

where equation (7.35) was obtained by expanding the matrix $\hat{\mathbf{W}}$ into its component blocks [equation (7.32)].

Let $\mathbf{T}$ be a matrix containing the eigenvectors of $\hat{\mathbf{W}}$, and let $\mathbf{\Lambda}$ be a diagonal matrix containing its eigenvalues. Then

$$\hat{\mathbf{W}} = \mathbf{T}^T\mathbf{\Lambda}\mathbf{T}$$

and the model output can be expressed as

$$\begin{aligned} \hat{y}(t) &= \mathbf{u_a^T(t)}\mathbf{T}^T\mathbf{\Lambda}\mathbf{T}\mathbf{u_a(t)} \\ &= \mathbf{x^T(t)}\mathbf{\Lambda}\mathbf{x(t)} \end{aligned} \tag{7.36}$$

where $\mathbf{x(t)} = \mathbf{Tu_a(t)}$, and let $x_k(t)$ be its kth entry at time t. Then, the output of the model can be expressed as the sum

$$\hat{y}(t) = \sum_{k=1}^{T+1} \lambda_k w_k^2(t) \tag{7.37}$$

where $|\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_{T+1}|$ are the eigenvalues of $\hat{\mathbf{W}}$. The terms $w_k(t)$ are generated by

$$\begin{aligned} w_k(t) &= \mathbf{T}(:,k)^T\mathbf{u_a(t)} \\ &= \mathbf{T}(1,k) + \mathbf{T}(2:T+1,k)^T\mathbf{u(t)} \end{aligned}$$

the sum of a constant, $\mathbf{T}(1, k)$, and the output of a linear filter, whose IRF is given by deleting the first element of $\mathbf{T}(:, k)$, thus removing the offset term:

$$h_k(\tau) = \mathbf{T}(\tau + 2, k), \qquad \tau = 0 \ldots T - 1 \tag{7.38}$$

Thus, each eigenvalue is associated with an eigenvector, a column in $\mathbf{T}$, that may be viewed as applying a linear filter plus an offset to the input, $u(t)$.

In many cases, only a few eigenvalues will be significant and equation (7.37) can be truncated to include only the P most significant eigenvalues in $\mathbf{\Lambda}$:

$$\hat{y}(t) \approx \sum_{k=1}^{P} \lambda_k w_k^2(t) \tag{7.39}$$

The output of this reduced system, equation (7.39), can be viewed as a Wiener–Bose model, with a filter bank comprising the P filters corresponding to the most significant eigenvalues.

Note that equation (7.38) discards the first element in each column of the transformation matrix, $\mathbf{T}$. This element generates a constant term in the vector $\mathbf{x}(\mathbf{t})$, which can be generated by the nonlinearity in the Wiener–Bose model rather than by the filter bank. Thus, truncation will have no effect on the output of the reduced model.

Since $\mathbf{T}$ contains the eigenvectors of $\hat{\mathbf{W}}$, it is a real symmetric matrix with orthogonal columns. However, the IRFs in the filter bank of the reduced Wiener–Bose model will not, in general, be orthogonal, since they are truncated copies of the orthogonal eigenvectors.

The resulting expansion is based on the dynamics of the system itself rather than some a priori *assumption*. Consequently, it has been termed a principal dynamic mode (PDM) expansion since each filter can be regarded as a dynamic mode.

7.4.1 Example: Principal Dynamic Modes of the Fly Retina Model

PDM decomposition was applied to the kernel estimates, for the simulated fly retina model, shown in Figure 7.16. The kernels were obtained by applying the Laguerre expansion technique to the colored-noise dataset. After computing the eigendecomposition of the matrix $\hat{\mathbf{W}}$, defined in equation (7.32), the IRFs of the PDM expansion were obtained by removing the first row of the eigenvector matrix, as shown in equation (7.38).

The basis expansion algorithm was then used to fit second-order, Wiener–Bose models to the colored noise data using a filter bank consisting of the principal dynamic modes. The mode associated with the largest eigenvalue was used first. Modes were added to the filter bank one at a time in order of decreasing eigenvalue. The fit was repeated after each mode was added and the model accuracy and MDL were evaluated. Table 7.1 summarizes the results of this analysis, showing the magnitude of the eigenvalues, the in-sample accuracy of the model, and the MDL cost function. The model with three dynamic modes minimized the MDL. This structure was taken to be optimal and subjected to further analysis. Figure 7.17 shows the IRFs of the first three principal dynamic modes.

Figure 7.18 shows the evolution of the Volterra kernels as modes were added to the PDM model. Figures 7.18A and 7.18B show the first- and second-order kernels, respectively, of a model with a single mode. In this case, the model structure is really a Wiener cascade, whose linear IRF is equal to that of the first dynamic mode, shown as a solid line in Figure 7.17. Figures 7.18C and 7.18D show the kernels of the model incorporating

TABLE 7.1 Results of the Principal Dynamic Mode Analysis of the Fly Retina Kernels[a]

Number of Modes	$\|\lambda\|$	% VAF (In-Sample)	MDL
1	10.73	88.10	2.436
2	5.80	93.92	1.249
3	1.70	94.20	1.199
4	1.01	94.21	1.201
5	0.51	94.22	1.208

[a] Examining the eigenvalues suggests either 2 or 4, since this is where the largest gaps (in a multiplicative sense) occur. Fitting the models reveals that choosing 3 modes minimizes the MDL. This model was chosen for further analysis.

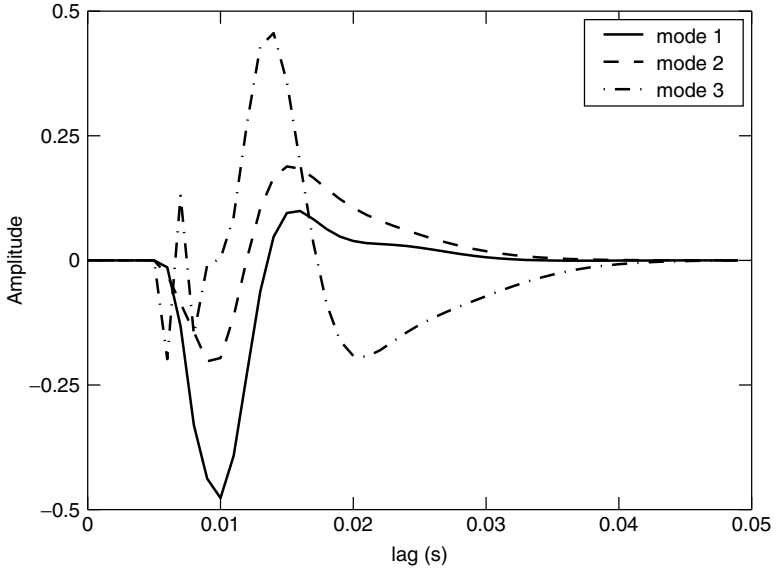

Figure 7.17 First three principal dynamic modes extracted from the kernels of the fly retina model.

only the first two dynamic modes. Figures 7.18E and 7.18F show the kernels generated by the first three PDMs. Comparing Figures 7.18A and 7.18C, it is evident that the second peak in the first-order kernel grows significantly when the second PDM is added to the model. Although the large, positive peak in the second-order kernel remains unchanged as modes are added to the model, Figures 7.18B, 7.18D, and 7.18F, the shapes of the depressions following the peak change noticeably.

7.4.2 Application: Cockroach Tactile Spine

French and Marmarelis (1995) used the PDM approach to extend the analysis of action potential encoding on the tactile spine of the cockroach, originally described in

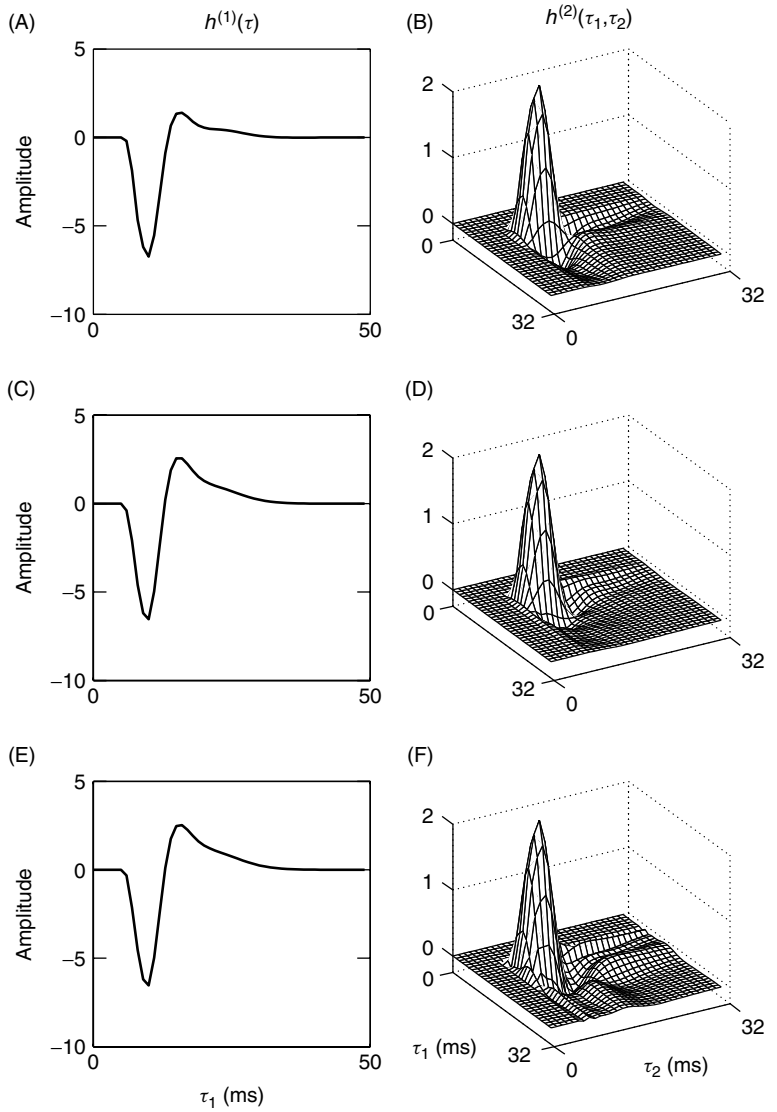

Figure 7.18 First- and second-order Volterra kernels constructed for PDM models of the fly retina, comprising the first, first two, and first three modes. The display of the second-order kernels was limited to lags of 0–32 ms for presentation purposes only. (A, B) First- and second-order kernels, respectively, generated by the first PDM. (C, D) Kernels generated by the first two modes. (E, F) kernels generated by the first three PDMs.

Section 7.2.5. They combined the Laguerre expansion technique with the PDM decomposition to construct Wiener–Bose models. In this study, the input spectrum was limited to a bandwidth of 100 Hz, and the input and output were sampled at 200 Hz. Again, spikes in the output were detected using a Schmidt trigger, resulting in a binary output signal. Two types of input were applied: a "high-power" input with a RMS level of 24.3 nA and a "low-power" input with an RMS level of 12.7 nA.

First, the Laguerre expansion technique was used to identify the zero- through second-order kernels; these closely resembled those estimated by the fast orthogonal algorithm (French and Korenberg, 1991). Principal dynamic mode analysis (Marmarelis, 1994, 1997; Marmarelis and Orme, 1993) was then used to construct a Wiener–Bose model from these kernels. The eigenvalues of the first four dynamic modes were [2.82, 2.27, 0.56, 0.21], suggesting that two dynamic modes might be sufficient to represent the system. Figures 7.19 and 7.20 show the IRFs of the two first two principal dynamic modes identified with both the high- and low-power inputs.

Figure 7.21 shows the output nonlinearity obtained with the high-power dataset. The amplitude of the nonlinearity corresponds to the probability of observing a spike in the output, given the values of the two PDM outputs. Note that the identification data did not probe the entire plane; the crosshatched area represents the portion of the plane for which there were enough input–output data to characterize the nonlinearity.

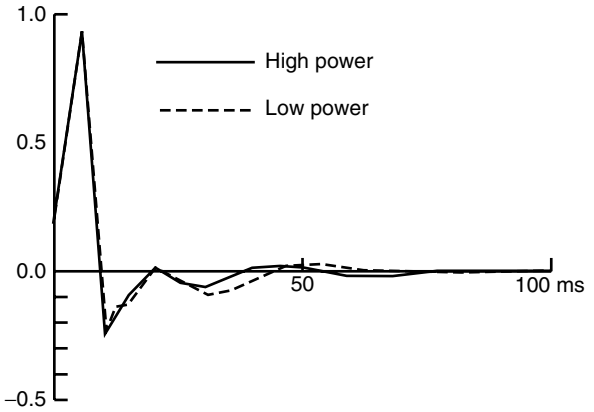

Figure 7.19 The IRF of the first principal dynamic mode of the cockroach tactile spine preparation. Note that there is little difference between the IRFs obtained from the low- and high-power inputs. From French and Marmarelis (1995). Used with permission.

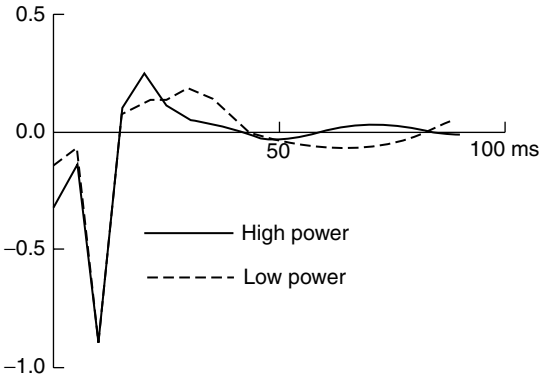

Figure 7.20 The IRF of the second principal dynamic mode of the cockroach tactile spine preparation. Note the variation between the IRFs obtained from the low- and high-power experiments. From French and Marmarelis (1995). Used with permission.

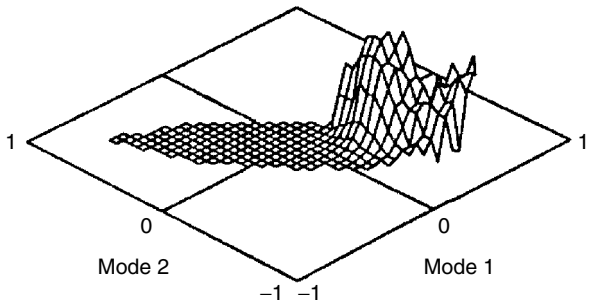

Figure 7.21 Static nonlinearity of the PDM model of the cockroach tactile spine. The cross-hatched area corresponds to the limits explored by the identification input. From French and Marmarelis (1995). Used with permission.

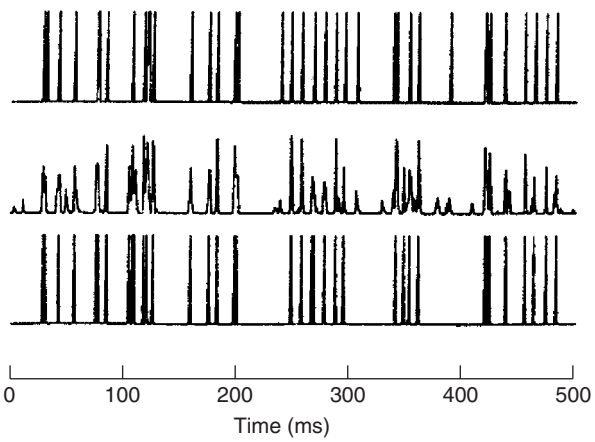

Figure 7.22 Cross-validation predictions from the PDM model of the cockroach tactile spine. The top trace shows the recorded output. The middle trace is the output of the two-input static nonlinearity. The lower trace is obtained by thresholding the nonlinearity output. From French and Marmarelis (1995). Used with permission.

Figure 7.22 illustrates the predictive power of the model. The upper trace shows a 500-ms segment of validation data, the middle trace shows the corresponding model output, and the lower trace shows the result of thresholding to the model output. The agreement between the experimental and model outputs is exceptionally good.

The nonlinearity, shown in Figure 7.21, indicates that the neuron is most likely to fire when the output of the first mode is positive, and that of the second mode is negative. Thus, the first mode appears to be excitatory, whereas the second is inhibitory. The IRF of the first mode, shown in Figure 7.19, is dominated by a positive spike at about 5 ms. Hence, the first mode apparently represents a short transport delay. French and Marmarelis proposed several potential explanations for the second, inhibitory mode.

It may not always be possible to identify the correspondences between the IRFs of the "principal dynamic modes" and the dynamics of underlying physiological processes. In particular, there is no guarantee that the various physiological process will produce

mutually orthogonal outputs. Thus, PDMs may contain contributions from several physiological processes, and vice versa. Computer Exercises **2** and **3** will explore these issues further.

7.5 PROBLEMS

1. Show that the least-squares solution found by the orthogonal algorithm, $\hat{\boldsymbol{\theta}} = \mathbf{R}^{-1}\mathbf{Q}^{\mathbf{T}}\mathbf{y}$, is equivalent to the explicit solution of the normal equations [see equation (2.33)], $(\mathbf{U}^{\mathbf{T}}\mathbf{U})^{-1}\mathbf{U}^{\mathbf{T}}\mathbf{y}$.

2. Write down the steps necessary to compute the IRF of a linear system using the FOA. Compare your algorithm to `fil.m`. What are the differences (if any)? When will they become significant?

3. What will be the computational and storage requirements be for the FOA applied to a third-order system?

4. The orthogonal algorithm may also be used to compute Wiener kernels, if the equivalent Wiener–Bose model uses a multiple-input Hermite polynomial, instead of a power series, as its nonlinearity. Let the input be white Gaussian noise, show that the expected value of the Hessian is diagonal, and then compute its inverse. Compare the values of the second-order coefficients corresponding to on- and off-diagonal kernel elements. Use the inverse Hessian to compute the variance of the resulting second-order kernel estimates, both on and off the diagonal.

5. Suppose that the basis expansion algorithm was applied using a basis of sinusoids. Show that the polynomial coefficients will be equal to the Fourier transforms of the kernels.

6. Re-derive equation (7.20), under the assumption that the measurement noise, $v(t)$, is the result of filtering a white Gaussian process with a FIR filter with impulse response, $h_v(\tau)$.

7. Show that the "neuronal modes" can also be computed from a matrix containing the polynomial coefficients.

7.6 COMPUTER EXERCISES

1. Load the file ../ch7/ear_model, and plot its components. Generate a white Gaussian input, and compute the resulting output from this system. Identify a second-order Volterra series using both the fast orthogonal algorithm and the Laguerre expansion technique. Experiment with different values of α and M, to determine appropriate values for this system. Convert the identified kernels into an LNL model.

Repeat the identification with output noise. Do 100 trials, and compute the mean and standard deviation of the ensemble of kernels identified by each technique. How does the choice of α and M affect these statistics? Also compute these statistics for the elements of the LNL model extracted from the kernels. Finally, repeat the simulation using a low-pass filtered input. *Hint*: It may be helpful to write a short program that

performs the series of 100 (or 1000 if you have time) trials and computes the ensemble statistics.

2. Load the file `ch7/pdm_models`, which contains the model `TwoModes`, and plot the components of the model. Generate a white Gaussian signal, with unit variance, and compute the model's response to this input. Extract the IRFs of the two filters in the model's filter bank, and generate an $X-Y$ plot of their outputs. Use the "principal dynamic modes" to identify the system, and generate an $X-Y$ plot of the outputs of the two principal dynamic modes. Compare this with the $X-Y$ plot generated from the simulation model. Compare the identified IRFs with those of the simulation model.

Repeat the problem, but use a low-pass filtered input (start with `butter(4,0.4)`, but try different filter types, orders, and cutoff frequencies). What happens to the shape of the $X-Y$ plot? What happens to the identified IRFs? Are the IRFs of the simulation model linear combinations of the identified IRFs? Can you transform the identified IRFs so that the major and minor axes of the $X-Y$ plot line up with the x and y axes? What can you say about the resulting IRFs?

3. Load the file `ch7/pdm_models`, which contains the model `TwoModes`, and plot the components of the model. Generate a white Gaussian signal, with unit variance, and compute the model's response to this input. Use the "principal dynamic modes" to identify the system, and generate an $X-Y$ plot of the outputs of the two principal dynamic modes. Use the "manual" option to select the number of modes. Examine the singular value plot, and choose the appropriate number of modes. Double the amplitude of the input, and repeat the identification. What happened? Compare the modes identified using the two differently scaled inputs with the two modes from the simulation model. Reduce the amplitude of the input, and repeat the identification. Can you explain the source of the "extra" mode?