CHAPTER 8

ITERATIVE LEAST-SQUARES METHODS

In Chapter 7, least-squares methods were developed for model structures that are linear in their parameters. These methods recast the identification problem as a least-squares regression, and then they solved that regression explicitly. This resulted in dramatic increases in model accuracy, as compared to the earlier, correlation-based methods presented in Chapter 6, since the solution used the data's statistics directly, rather than assuming that they followed a theoretical ideal.

The explicit least-squares methods presented in Chapter 7 have two principal limitations. First, they are computationally expensive, limiting them to systems with relatively short memories and low nonlinearity orders. Second, they are limited to model structures such as the Wiener and Volterra series, which are linear in their parameters.

This chapter discusses methods that can be used to find the minimum mean-square error (MMSE) model for structures whose outputs are nonlinear functions of some or all of their parameters. Except in a very few special cases, the MMSE solution cannot be found using a linear regression or any other closed-form expression. Rather, an iterative search is required to find the optimal model.

This chapter discusses iterative identification methods that seek to minimize the mean-square error (MSE) between the outputs of the system and its model. These methods bring the advantages of an MMSE solution, discussed in Chapter 7, to model structures that cannot be identified using linear-regression-based methods.

8.1 OPTIMIZATION METHODS

This section considers identification methods that use parametric optimization techniques. In all cases the model will be defined by an $M \times 1$ vector of parameters, although the particular choice of parameters will depend on the structure of the model being identified.

Identification of Nonlinear Physiological Systems, By David T. Westwick and Robert E. Kearney
ISBN 0-471-27456-9 © 2003 Institute of Electrical and Electronics Engineers

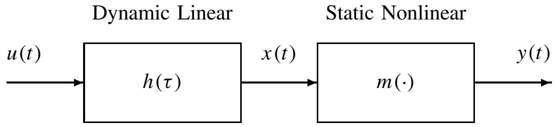

Figure 8.1 Block diagram of a Wiener system.

Consider, for example, the identification of a Wiener system, comprising a linear dynamic element followed by a static nonlinearity, as shown in Figure 8.1. Suppose that the linear dynamic subsystem is to be modeled as a FIR filter with T lags and that the static nonlinearity will be modeled using a polynomial of order Q. A straightforward parametric representation of this system would contain the T filter weights followed by the $Q + 1$ polynomial coefficients.

As another example, consider the Volterra kernels identified in Section 7.2 by estimating a vector of polynomial coefficients, labeled $\boldsymbol{\theta}$ in equation (7.6). For that model, $\boldsymbol{\theta}$ was a vector containing the unique elements of the Volterra kernels. Therefore, in the context of this chapter, $\boldsymbol{\theta}$ can be regarded as a parameter vector describing the Volterra series.

Whatever the model structure and parameterization, the objective is to find the parameter vector, $\hat{\boldsymbol{\theta}}$, that minimizes the MSE between the experimental and model outputs. In the discussion of iterative optimization techniques, it is useful to denote explicitly the dependence of the model output on the parameter vector. Thus, the model output will be written as $\hat{y}(t, \boldsymbol{\theta})$. Similarly, the MSE is written as a function of the parameter vector, $\boldsymbol{\theta}$:

$$\begin{aligned} V_N(\boldsymbol{\theta}) &= \frac{1}{2N} \sum_{t=1}^{N} \big(z(t) - \hat{y}(t, \boldsymbol{\theta})\big)^2 \\ &= \frac{1}{2N} \sum_{t=1}^{N} \epsilon^2(t, \boldsymbol{\theta}) \end{aligned} \tag{8.1}$$

where $\epsilon(t, \boldsymbol{\theta}) = z(t) - \hat{y}(t, \boldsymbol{\theta})$ is the model error. The cost function, $V_N(\boldsymbol{\theta})$, may be visualized as a surface, the so-called error surface, defined on an M-dimensional domain, where each dimension corresponds to the possible values of one model parameter. The "height" of this error surface is equal to the mean square error.

8.1.1 Gradient Descent Methods

Classical optimization methods start with an initial guess for the parameter vector, $\boldsymbol{\theta}_0$, and then apply a correction, $\mathbf{d_k}$, at each iteration. Thus, the update after the kth iteration is

$$\boldsymbol{\theta}_{\mathbf{k+1}} = \boldsymbol{\theta}_{\mathbf{k}} + \mathbf{d_k}$$

The simplest optimization method follows the direction of steepest descent on the error surface given by the (negative) gradient,

$$\mathbf{d_k} = -\mu_k \left. \frac{\partial V_N(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \right|_{\boldsymbol{\theta}=\boldsymbol{\theta}_{\mathbf{k}}}$$

where μ_k is the size of the kth step. Many implementations adjust the step size during the optimization to adapt to the local shape of the error surface. If the error surface is relatively smooth and simple, then relatively large steps may be taken. In contrast, if the gradient changes quickly, smaller steps may be necessary. One approach is to adjust the step size based on the result of each update. If the update is successful (i.e., the error is reduced), then the step size is increased. On the other hand, if a proposed update causes the error to increase, the update is rejected, the step size is reduced, the update is repeated, and the error is reevaluated. If the error still increases, the step size is reduced further.

The gradient can be computed from equation (8.1) using the chain rule:

$$\frac{\partial V_N(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = -\frac{1}{N}\sum_{t=1}^{N} \epsilon(t,\boldsymbol{\theta})\frac{\partial \hat{y}(t,\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$$

Note that the factor of 2 in the denominator of equation (8.1) has been canceled by a factor of 2 in the derivative computation.

The Jacobian is defined as the matrix of the partial derivatives of the model output with respect to the parameters,

$$\mathbf{J}(t,i) = \frac{\partial \hat{y}(t,\boldsymbol{\theta})}{\partial \boldsymbol{\theta}(i)}$$

where $\boldsymbol{\theta}(i)$ is the ith element of the parameter vector. Thus, the gradient is given by

$$\frac{\partial V_N(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = -\frac{1}{N}\mathbf{J}^T \boldsymbol{\epsilon} \tag{8.2}$$

where $\boldsymbol{\epsilon}$ is a vector containing the error signal $\epsilon(t,\theta)$.

8.1.2 Identification of Block-Structured Models

As an example, let us use parametric optimization methods to identify the elements of a Hammerstein cascade, a static nonlinearity followed by a dynamic linear system, as shown in Figure 8.2 and described in Section 4.3.2. The output of a Hammerstein system is given by equation (4.39):

$$\hat{y}(t) = \sum_{\tau=0}^{T-1} h(\tau)\left\{\sum_{q=0}^{Q} c^{(q)}u^q(t-\tau)\right\}$$

Implementation of a gradient descent search algorithm proceeds as follows. First construct a parameter vector, $\boldsymbol{\theta}$, describing the model. One possibility for a Hammerstein system would be the $Q+1$ polynomial coefficients, $c^{(q)}$, describing the static nonlinearity, $m(\cdot)$, and the T weights of the impulse response of the linear filter, $h(\tau)$. The

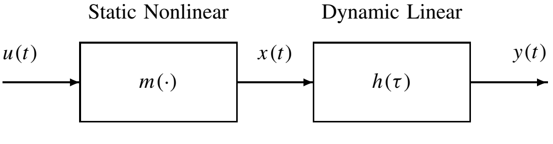

Figure 8.2 Block diagram of a Hammerstein system.

corresponding parameter vector would be

$$\boldsymbol{\theta} = \left[c^{(0)}\ c^{(1)}\ \dots\ c^{(Q)}\ h(0)\ h(1)\ \dots\ h(T-1)\right]^T \tag{8.3}$$

Next, compute the Jacobian matrix by differentiating the model output with respect to each parameter. For a Hammerstein system, this will generate two distinct types of columns.

The first $Q+1$ columns of the Jacobian matrix will contain the partial derivatives of the model output with respect to the polynomial coefficients.

$$\frac{\partial \hat{y}(t,\boldsymbol{\theta})}{\partial c^{(q)}} = \sum_{\tau=0}^{T-1} h(\tau)u^q(t-\tau) \tag{8.4}$$

Note that these Jacobian columns are computed by convolving the input, raised to the qth power, with the current linear filter estimate.

The remaining T columns of the Jacobian will contain the partial derivatives of the model output with respect to each of the T weights of the filter IRF.

$$\begin{aligned}\frac{\partial \hat{y}(t,\boldsymbol{\theta})}{\partial h(\tau)} &= \sum_{q=0}^{Q} c^{(q)}u^q(t-\tau)\\ &= x(t-\tau)\end{aligned} \tag{8.5}$$

where $x(t)$ is the output of the current estimate of the polynomial nonlinearity. Thus, these remaining columns are simply delayed copies of the output of the current estimate of the static nonlinearity.

8.1.2.1 Implementation Details

Computing the Jacobian matrix is at the heart of the optimization, but there are several practical issues that must be considered before presenting the overall algorithm.

Dealing with the Redundant Gain As noted in Section 4.3.2.1, there is one extra degree of freedom in the Hammerstein model structure. Thus, if the filter weights are multiplied by a gain, κ, and the polynomial coefficients divided by the same gain, there will be no effect on the input–output behavior of the system and hence no effect on the cost function, $V_N(\boldsymbol{\theta})$. Consequently, as noted before, it is customary to normalize the elements of a Hammerstein model in some way.

Note that changing the relative gains of the linear filter and static nonlinearity corresponds to moving along a straight line in parameter space. Movement along this line will not change the cost function, so the slope, curvature, and all higher-order derivatives of the error surface will be zero in this direction. Therefore, the gradient will be zero along this line, and the search direction will always be orthogonal to it. Consequently, the extra degree of freedom will have no effect on the simple gradient descent scheme. Therefore, the normalization can be applied once, after the search has converged.

Use of Orthogonal Polynomials There are substantial advantages in representing static nonlinearities with orthogonal polynomials (see Sections 2.5.2, 4.2.1, 6.2.1.3, and 6.2.2.3). Consequently, the iterative search algorithm for the Hammerstein system should be reformulated to represent the nonlinearity with an orthogonal polynomial system. If the Tchebyshev polynomials, $\mathcal{T}^{(q)}$ defined in Section 2.5.4, are employed, then the model output will be given by

$$\hat{y}(t) = \sum_{\tau=0}^{T-1} h(\tau) \left\{ \sum_{q=0}^{Q} \gamma^{(q)} \mathcal{T}^{(q)}(u(t-\tau)) \right\} \tag{8.6}$$

where $\gamma^{(q)}$ is the qth-order (Tchebyshev) polynomial coefficient.

The parameter vector now contains the $Q+1$ Tchebyshev polynomial coefficients, $\gamma^{(q)}$, followed by the T filter IRF weights, $h(\tau)$. The first $Q+1$ columns of the Jacobian matrix will contain derivatives with respect to the $Q+1$ Tchebyshev polynomial coefficients.

$$\frac{\partial \hat{y}(t, \boldsymbol{\theta})}{\partial \gamma^{(q)}} = \sum_{\tau=0}^{T-1} h(\tau) \mathcal{T}^{(q)}(u(t-\tau)) \tag{8.7}$$

This is similar to equation (8.4), for a power series representation of the nonlinearity, except that the powers of the input have been replaced with Tchebyshev polynomials in the input.

The remaining T columns of the Jacobian matrix contain derivatives with respect to the filter weights.

$$\begin{aligned} \frac{\partial \hat{y}(t, \boldsymbol{\theta})}{\partial h(\tau)} &= \sum_{q=0}^{Q} \gamma_q \mathcal{T}_q(u(t-\tau)) \\ &= x(t-\tau) \end{aligned} \tag{8.8}$$

This expression, a delayed copy of the nonlinearity output, is again similar to that derived using the power series except that the nonlinearity output is computed using Tchebyshev polynomials.

Initialization The success of an iterative minimization technique often depends on the initial guess at the parameter vector. A poor initial guess may result in the minimization requiring many iterations, and hence a long computation time, before the optimum is reached. Indeed, in some cases, a poor initial estimate may cause the algorithm to converge to a suboptimal local minimum. Thus, a good initial estimate of the model is very desirable. For this example, the initial parameters will be estimated using the correlation-based method described in Section 6.2.2.4. After accounting for these practical issues, the final algorithm proceeds as follows.

Summary of Algorithm

1. Create an initial estimate of the parameters using the Hunter–Korenberg algorithm, described in Section 6.2.2.4. Place the polynomial coefficients and filter weights in the initial parameter vector, $\boldsymbol{\theta}_0$.

2. Set the initial step size to a small number, say $\mu_0 = 0.001$. Choose an acceleration rate, $\kappa_a > 1$, and a deceleration rate, $0 < \kappa_d < 1$.* Initialize the iteration counter, $k = 0$.
3. Compute the output of the current model, $\hat{y}(\boldsymbol{\theta}_\mathbf{k}, t)$, model error, $\epsilon(t) = z(t) - \hat{y}(\boldsymbol{\theta}_\mathbf{k}, t)$, and cost function, $V_N(\boldsymbol{\theta}_\mathbf{k}) = \frac{1}{2N}\|\boldsymbol{\epsilon}\|_2^2$.
4. Compute the first $Q + 1$ columns of the Jacobian matrix, $\mathbf{J}(:, 1 : Q + 1)$, using equation (8.7).
5. Compute the remaining T columns, $\mathbf{J}(:, Q + 2 : Q + T + 1)$, using equation (8.8).
6. Update the parameter vector, $\boldsymbol{\theta}_\mathbf{k+1} = \boldsymbol{\theta}_\mathbf{k} + \mu_k \mathbf{J}^T \boldsymbol{\epsilon}$.
7. Compute the new model output, error, and cost function, $V_N(\boldsymbol{\theta}_\mathbf{k+1})$.
8. If $V_N(\boldsymbol{\theta}_\mathbf{k+1}) > V_N(\boldsymbol{\theta}_\mathbf{k})$, reject the new parameter vector. If the step size is very small, the search has converged, so set $\hat{\boldsymbol{\theta}} = \boldsymbol{\theta}_\mathbf{k}$ and exit. Otherwise, decrease the step size $\mu_k = \kappa_d \cdot \mu_k$, and go to step 6.
9. The iteration was successful, $V_N(\boldsymbol{\theta}_\mathbf{k+1}) < V_N(\boldsymbol{\theta}_\mathbf{k})$, so increase the step size, $\mu_{k+1} = \kappa_a \cdot \mu_k$, set $k = k + 1$, and return to step 4.

The adaptive step-size algorithm, above, uses three parameters: an initial step size, μ_0, an acceleration rate, κ_a, and a deceleration rate, κ_d. These three parameters are highly problem-dependent.

8.1.3 Second-Order Optimization Methods

Gradient descent optimizations are relatively simple and easy to implement, but they can be slow to converge under some conditions. For example, consider what happens when the error surface contains a long, narrow valley, as illustrated in Figure 8.3 for

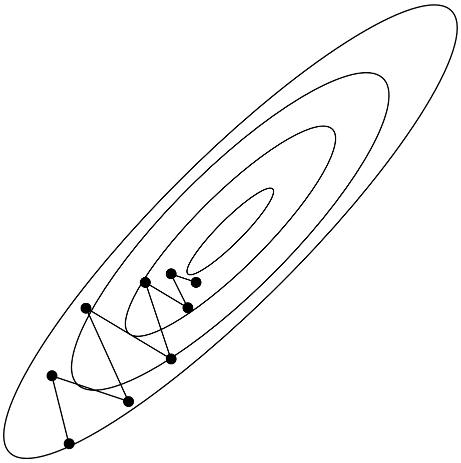

Figure 8.3 When a simple gradient descent procedure enters a long valley, the search path often oscillates from side to side, rather than traveling along the length of the valley.

*Setting $\kappa_a = 1.1$ and $\kappa_d = 0.9$ results in relatively slow changes to the step size, μ_k. Once the algorithm finds a suitable step size, convergence should be relatively quick.

two parameters. The direction of steepest descent when approaching the valley, which will be exactly orthogonal to its contours, will be nearly orthogonal to its long axis. Consequently, except very close to the middle of the valley, the direction of steepest descent will point across, rather than along, the valley. The step size will therefore be limited by the width of the valley. If too large a step is made, the parameter vector will begin to climb the far side of the valley's walls and the cost function will increase. As a result, the gradient descent optimization will be forced to take only very small steps and will be slow to converge.

Second-order methods attempt to avoid this problem by using the local curvature of the error surface to speed convergence. Thus, if the search enters a long, narrow valley, the curvature will be highest in the direction orthogonal to the valley and will be lowest in the direction along the valley. This curvature information can be used to adjust the direction of the step so that it points along the valley.

Second-order search methods assume that the local shape of the error surface can be approximated by a quadratic function. Therefore, computing the curvature of the error surface should define the quadratic surface and make it possible to locate the minimum analytically.

Consider what happens if the error surface is actually a quadratic function of the parameter vector. Then, the error may be represented exactly by a second-order Taylor expansion about any point as follows:

$$V_N(\boldsymbol{\theta} + \boldsymbol{\delta}) = V_N(\boldsymbol{\theta}) + \left(\frac{\partial V_N}{\partial \boldsymbol{\theta}}\right)^T \boldsymbol{\delta} + \frac{1}{2!}\boldsymbol{\delta}^T \frac{\partial^2 V_N}{\partial \boldsymbol{\theta}^2}\boldsymbol{\delta}$$

Note that the second term on the right-hand side is the gradient of the error surface, which can be computed using equation (8.2).

Now, define the Hessian, **H**, to be an $M \times M$ matrix containing the second-order partial derivatives of V_N with respect to its parameters. The (i, j)th element of the Hessian will be

$$\mathbf{H}(i, j) = \frac{\partial^2 V_N(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}(i) \partial \boldsymbol{\theta}(j)} \tag{8.9}$$

Using this, the second-order Taylor expansion of the error surface becomes

$$V_N(\boldsymbol{\theta} + \boldsymbol{\delta}) = V_N(\boldsymbol{\theta}) - \frac{1}{N}(\boldsymbol{\epsilon}^T \mathbf{J})\boldsymbol{\delta} + \frac{1}{2!}\boldsymbol{\delta}^T \mathbf{H}\boldsymbol{\delta} \tag{8.10}$$

Furthermore, the minimum of the error surface can be found exactly by differentiating equation (8.10) with respect to $\boldsymbol{\delta}$:

$$\frac{\partial V_N(\boldsymbol{\theta} + \boldsymbol{\delta})}{\partial \boldsymbol{\delta}} = -\frac{1}{N}\boldsymbol{\epsilon}^T \mathbf{J} + \boldsymbol{\delta}^T \mathbf{H}$$

and setting the result to zero. Since **H** is a symmetric matrix, the minimum value of the error surface will occur at

$$\boldsymbol{\delta} = \frac{1}{N}\mathbf{H}^{-1}\mathbf{J}^T \boldsymbol{\epsilon} \tag{8.11}$$

If the error surface were actually quadratic, equation (8.11) would give the exact minimum. However, this will only be the case when the model is linear in its parameters; equation (8.11) would then be the solution of the normal equation for a linear regression.

In practice, these approaches are used when the model is nonlinear in the variables so that the error surface includes higher-order terms. Consequently, equation (8.10) will only approximate the error surface, and solving equation (8.11) will not reach the exact minimum in a single step. However, applying equation (8.11) iteratively will converge on the true minimum since the error in the second-order Taylor expansion decreases as the parameter vector approaches the minimum.

8.1.3.1 The Newton Method Conceptually, the simplest second-order nonlinear optimization technique is the Newton method, which computes the Hessian and quadratic minimum exactly. Thus it uses the relation below, obtained by substituting equation (8.2) into the Hessian definition given in equation (8.9):

$$\begin{aligned}\mathbf{H}(i,j) &= \frac{-1}{N}\frac{\partial}{\partial\boldsymbol{\theta}(j)}\left(\mathbf{J}^T(:,i)\boldsymbol{\epsilon}\right)\\ &= \frac{1}{N}\mathbf{J}(:,i)^T\mathbf{J}(:,j) + \frac{1}{N}\left(\frac{\partial^2\hat{y}(\boldsymbol{\theta},t)}{\partial\boldsymbol{\theta}(i)\partial\boldsymbol{\theta}(j)}\right)^T\boldsymbol{\epsilon} \qquad (8.12)\end{aligned}$$

Since the error surface is not exactly quadratic, equation (8.11) will not find the minimum. Rather, it must be applied iteratively using an adjustable step size to help ensure convergence. The parameter update is given by

$$\mathbf{d_k} = \frac{\mu_k}{N}\mathbf{H}^{-1}\mathbf{J}^T\boldsymbol{\epsilon}$$

Redundant Parameters The Newton method requires the inversion of the exact Hessian and is therefore not suited to the block-structured identification problem, or indeed for the identification of any model structure with redundant parameters. This is because the redundant parameters will make the Hessian singular and hence impossible to invert.

For example, suppose that $\boldsymbol{\theta}(i)$ and $\boldsymbol{\theta}(j)$ are two redundant parameters so that

$$\frac{\partial\hat{y}(\boldsymbol{\theta},t)}{\partial\boldsymbol{\theta}(i)} = \alpha\frac{\partial\hat{y}(\boldsymbol{\theta},t)}{\partial\boldsymbol{\theta}(j)}$$

for some nonzero constant, α. The same proportionality will hold for the second-order derivatives:

$$\frac{\partial^2\hat{y}(\boldsymbol{\theta},t)}{\partial\boldsymbol{\theta}(i)\partial\boldsymbol{\theta}(k)} = \alpha\frac{\partial^2\hat{y}(\boldsymbol{\theta},t)}{\partial\boldsymbol{\theta}(j)\partial\boldsymbol{\theta}(k)}$$

Since, $\mathbf{H}(:,i) = \alpha\mathbf{H}(:,j)$, the Hessian will be singular and cannot be inverted.

For the Hammerstein cascade, there are two redundant linear combinations of parameters. Specifically, for any set of polynomial coefficients and filter weights, equations (8.4) and (8.5) can be used to show that

$$\sum_{q=0}^{Q} c^{(q)}\frac{\partial\hat{y}(\boldsymbol{\theta},t)}{\partial c^{(q)}} = \sum_{\tau=0}^{T-1} h(\tau)\frac{\partial\hat{y}(\boldsymbol{\theta},t)}{\partial h(\tau)} = \hat{y}(\boldsymbol{\theta},t)$$

Thus there is no unique minimum in the error surface; the Hessian will be singular and the Newton method cannot be used. If the redundant degree of freedom could be eliminated, the Hessian would become nonsingular, and the Newton method could be used.

One way to eliminate this extra degree of freedom would be to fix the value of one of the (nonzero) parameters. However, the optimum value of this parameter is not known at the start of iteration, so there is no way to guarantee that it is not zero. If this happens, rescaling the remaining parameters will involve a division by zero so their "optimal values" will be undefined.

Another approach would be to normalize either the filter weights or the polynomial coefficients. This bypasses the difficult problem of selecting a single, nonzero parameter to fix. The simplest way to implement this is to use a "constrained optimization" in which a term is added to the cost function to force the normalization. For example, the cost function

$$V_N(\boldsymbol{\theta}) = \frac{1}{2N}\sum_{t=1}^{N}\epsilon^2(\boldsymbol{\theta}, t) + K\left(1 - \sum_{\tau=0}^{T-1} h^2(\tau)\right)^2 \tag{8.13}$$

where K is a large positive constant, will force the filter weights to be normalized. This modified cost function eliminates the redundancy in the parameter space and will have a unique minimum where the MSE is minimized *and* the norm of the IRF weights is one.

8.1.3.2 The Gauss–Newton Method The Hessian is often dominated by the first term in equation (8.12), which is proportional to the square of the Jacobian. This is especially true near the minimum of the cost function, where the error is small. Moreover, the second term, containing the mixed partial derivatives of the model output, is much more expensive to compute than the first term. Thus, the Hessian is often approximated using only the first term:

$$\hat{\mathbf{H}} = \frac{1}{N}\mathbf{J}^T\mathbf{J} \tag{8.14}$$

The step size must be kept small with this approximation since the error surface is not exactly quadratic and the Hessian is not computed exactly. The resulting algorithm, known as the Gauss–Newton iteration, updates the parameter vector according to

$$\mathbf{d_k} = \frac{\mu_k}{N}\hat{\mathbf{H}}^{-1}\mathbf{J}^T\boldsymbol{\epsilon}$$

As with the Newton method, the Gauss–Newton method cannot identify block-structured models directly because the approximation to the Hessian will be singular because of the redundant parameters. As before, the redundancy can be eliminated either by fixing the value of a nonzero parameter or by adding a constraint to the cost function, as in equation (8.13).

8.1.3.3 The Levenberg–Marquardt Method The Levenberg–Marquardt algorithm uses a different approximation for the Hessian,

$$\hat{\mathbf{H}} = \frac{1}{N}\mathbf{J}^T\mathbf{J} + \mu_k\mathbf{I_M} \tag{8.15}$$

where $\mathbf{I_M}$ is the $M \times M$ identity matrix. Thus, the parameter update will be given by

$$\mathbf{d_k} = \left(\frac{1}{N}\mathbf{J}^T\mathbf{J} + \mu_k\mathbf{I_M}\right)^{-1}\mathbf{J}^T\boldsymbol{\epsilon} \tag{8.16}$$

Notice that, in contrast to the Newton and Gauss–Newton methods, the step-size parameter is included in the approximate Hessian definition. For small values of μ_k, the first term in equation (8.15) dominates, and the Levenberg–Marquardt algorithm behaves like the Gauss–Newton iteration. For large values of μ_k, $\hat{\mathbf{H}}$ is nearly diagonal. Thus, $\hat{\mathbf{H}}^{-1}$ will also be nearly diagonal, and the Levenberg–Marquardt algorithm will behave like a simple gradient descent with the step size equal to $1/\mu_k$.

An added benefit of the diagonal term, $\mu_k \mathbf{I}_\mathbf{M}$, is that it guarantees that $\hat{\mathbf{H}}$ will be nonsingular. Consequently, block-structured models, such as the Hammerstein cascade, can be identified directly without resorting to constrained optimization.

8.1.4 Jacobians for Other Block Structures

The Jacobian is all that is required to identify a block structured model using most of these iterative methods (i.e., simple gradient descent, the Gauss–Newton, or Levenberg–Marquardt algorithms). Consequently, it will be useful to derive the Jacobians for three other common block structures: the Wiener, the LNL, and the NLN cascades.

8.1.4.1 Wiener Cascade The output of the Wiener cascade, shown in Figure 8.1, is given by equation (4.34), which is repeated here:

$$\hat{y}(t) = \sum_{q=0}^{Q} c^{(q)} \left(\sum_{\tau=0}^{T-1} h(\tau) u(t-\tau) \right)^q$$

A straightforward parameter vector for this model contains the T filter IRF weights and the $Q+1$ polynomial coefficients.

$$\boldsymbol{\theta} = \left[h(0)\ h(1) \ldots h(T-1)\ c^{(0)}\ c^{(1)} \ldots c^{(Q)} \right]^T \tag{8.17}$$

Hence, the first T columns of the Jacobian will be given by

$$\begin{aligned} \mathbf{J}(t, i+1) &= \frac{\partial \hat{y}(\boldsymbol{\theta}, t)}{\partial h(i)}, \qquad i = 0, \ldots, T-1 \\ &= \sum_{q=0}^{Q} q c^{(q)} \left(\sum_{\tau=0}^{T-1} h(i) u(t-i) \right)^{q-1} u(t-i) \\ &= m'(x(t)) u(t-i) \end{aligned} \tag{8.18}$$

where $x(t)$ is the output of the current linear filter estimate, and $m'(\cdot)$ is the derivative of the static nonlinearity with respect to its input.

$$m'(w) = \frac{dm(w)}{dw}$$

The remaining $Q+1$ columns of the Jacobian will be

$$\begin{aligned} \mathbf{J}(t, i+T+1) &= \frac{\partial \hat{y}(\boldsymbol{\theta}, t)}{\partial c^{(i)}}, \qquad i = 0, \ldots, Q \\ &= \left(\sum_{\tau=0}^{T-1} h(\tau) u(t-\tau) \right)^q \\ &= x^q(t) \end{aligned} \tag{8.19}$$

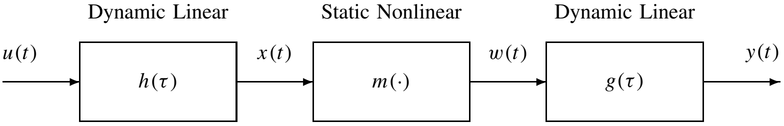

Figure 8.4 Block diagram of an LNL cascade model.

This parameter vector contains one redundant degree of freedom, as for the Hammerstein system. Thus, if the Gauss–Newton method is used, either one parameter must be fixed or one constraint must be added to the cost function.

8.1.4.2 LNL Cascade The LNL, or sandwich, model, discussed in Section 4.3.3, is illustrated in Figure 8.4. The model consists of an initial linear filter, represented by its impulse response, $h(\tau)$, followed by a static nonlinearity, represented by the coefficients, $c^{(q)}$, of a polynomial, followed by a second linear element, whose impulse response is $g(\tau)$. There are two intermediate signals for this structure: $x(t)$, the output of $h(\tau)$, and $w(t)$, the input to $g(\tau)$. The model's output, derived in equation (4.42), is

$$\begin{aligned} y(t) &= \sum_{\sigma=0}^{T-1} g(\sigma) w(t-\sigma) \\ &= \sum_{\sigma=0}^{T-1} g(\sigma) \sum_{q=0}^{Q} c^{(q)} \left(\sum_{\tau=0}^{T-1} h(\tau) u(t-\sigma-\tau) \right)^{q} \end{aligned}$$

The parameter vector will contain the IRF weights of the two linear filters plus the polynomial coefficients.

$$\boldsymbol{\theta} = \left[h(0) \ldots h(T-1)\; c^{(0)} \ldots c^{(Q)}\; g(0) \ldots g(T-1) \right]^{T} \tag{8.20}$$

There are two redundant degrees of freedom in this parameterization since the overall gain of the system can be distributed arbitrarily among three elements.

The first T columns of the Jacobian, derived from the weights of $h(\tau)$, will be

$$\begin{aligned} \frac{\partial \hat{y}(\boldsymbol{\theta}, t)}{\partial h(i)} &= \sum_{\sigma=0}^{T-1} g(\sigma) \frac{\partial w(t)}{\partial h(i)} \\ &= \sum_{\sigma=0}^{T-1} g(\sigma) m'(x(t-\sigma)) u(t-\sigma-i) \end{aligned} \tag{8.21}$$

where $m'(x(t-\sigma))$ is the derivative

$$m'(x(t-\sigma)) = \frac{\partial m(x(t-\sigma))}{\partial x} = \sum_{q=0}^{Q} q c^{(q)} w \left(\sum_{\tau=0}^{T-1} h(\tau) u(t-\sigma-\tau) \right)^{q-1}$$

Notice that equation (8.21) is similar to equation (8.18), the relation for equivalent columns in the Jacobian of a Wiener model, except that the result is filtered by $g(\tau)$, the IRF of the final element.

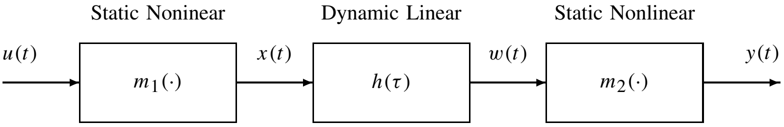

Figure 8.5 Block diagram of an NLN cascade model, comprising two static nonlinearities separated by a dynamic linear system.

The remainder of the Jacobian may be derived by treating the LNL cascade as a Hammerstein system driven by $x(t)$. As such, the terms due to the polynomial coefficients are given by equation (8.4), but with $x(t)$ substituted for $u(t)$,

$$\frac{\partial \hat{y}(k,\boldsymbol{\theta})}{\partial c^{(q)}} = \sum_{\tau=0}^{T-1} h(\tau)x^q(k-\tau)$$

and those due to the weights of the final linear element are obtained from equation (8.5) with $w(t)$ substituted for $x(t)$:

$$\frac{\partial \hat{y}(k,\boldsymbol{\theta})}{\partial h(\tau)} = w(t-\tau)$$

8.1.4.3 NLN Cascade A similar strategy can be used to determine the Jacobian for the NLN cascade model, shown in Figure 8.5. The model output, derived in equation (4.47), is

$$\hat{y}(\theta,t) = \sum_{q_2=0}^{Q_2} c_2^{(q_2)} \left(\sum_{q_1=0}^{Q_1} c_1^{(q_1)} \sum_{\tau=0}^{T-1} h(\tau)u^{q_1}(t-\tau) \right)^{q_2}$$

The derivatives with respect to the coefficients of the first polynomial nonlinearity are computed using the chain rule,

$$\begin{aligned}\frac{\partial \hat{y}(k,\boldsymbol{\theta})}{\partial c_1^{(q)}} &= \sum_{q_2=0}^{Q_2} c_2^{(q_2)} w^{q_2-1}(t) \frac{\partial w(t)}{\partial c_1^{(q)}} \\ &= m_2'(w(t)) \sum_{\tau=0}^{T-1} h(\tau)u^q(t-\tau)\end{aligned}$$

Note that this is obtained by computing the Jacobian for the Hammerstein system comprising the first two elements and then multiplying it by the local slope of the final nonlinearity.

The remainder of the Jacobian is determined by considering the final two elements as a Wiener cascade with input $x(t)$, the output of the first static nonlinear element $m_1(\cdot)$. This gives

$$\frac{\partial \hat{y}(\boldsymbol{\theta},t)}{\partial h(\tau)} = m_2'(w(t))x(t-\tau)$$

and

$$\frac{\partial \hat{y}(\boldsymbol{\theta},t)}{\partial c_2^{(q)}} = w^q(t)$$

8.1.5 Optimization Methods for Parallel Cascade Models

How would optimization methods be used to identify a parallel cascade model, such as that shown in Figure 8.6? To so do, the Jacobian matrix for the specific model structure must be determined. Notice, however, that the output of the parallel cascade model is the sum of the outputs of all the paths. Furthermore, changing the parameters in one path will have no effect on the outputs of the other paths. Consequently, the overall Jacobian matrix can be assembled from the Jacobians of the individual paths. Thus, the Jacobian of any parallel cascade model, made up of any combination of Hammerstein, Wiener, LNL, and NLN paths, can be computed from the expressions derived in the previous sections. Once the Jacobian is available, the gradient descent and/or Levenberg–Marquardt method may be used to find the optimal parameter set.

Marmarelis (1997; Marmarelis and Zhao, 1994, 1997) proposed using back-propagation to identify models with the structure, shown in Figure 8.7, that he called a "polynomial function neural network" or a "separable Volterra network." The appearance of this figure is patterned after those used to describe feedforward neural networks. In this network, the nodes in the hidden layer, B_1 through B_P, are summing junctions. Each of the hidden nodes is connected to each of the inputs with a multiplicative weight. For example, the input $u(t-i)$ is multiplied by the weight $b_j(i)$, before being sent to node B_j. The outputs of the hidden layer nodes are transformed by a multiple-input polynomial, $m(x_1, \ldots, x_P)$, to produce the model output, $y(t)$.

Notice that $x_1(t)$, the output of the first neuron, is given by the sum

$$x_1(t) = \sum_{i=0}^{T} b_1(i)u(t-i)$$

which is just a linear convolution. Thus, the neurons in the hidden layer are just linear FIR filters. Consequently, the SVN structure shown in Figure 8.7 is equivalent to the Wiener–Bose model, shown in Figure 4.20. In applying the SVN, Marmarelis (1997) excluded all cross-terms from the nonlinearity. Thus, the structure of the separable Volterra network was reduced to that of the parallel Wiener cascade, shown in Figure 8.6.

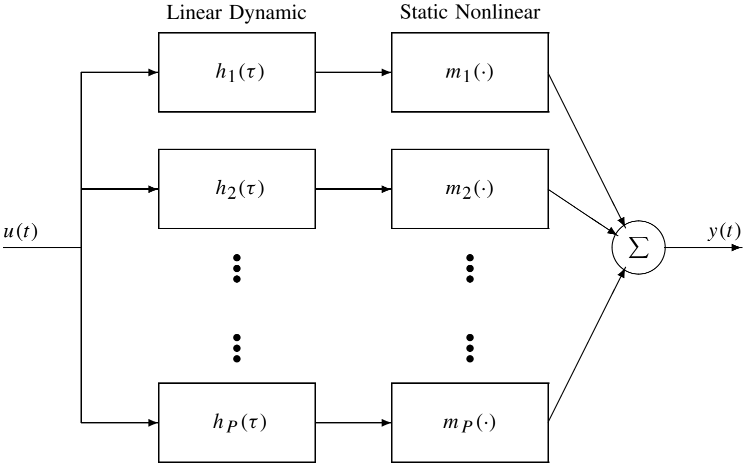

Figure 8.6 Parallel cascade made up of Wiener systems.

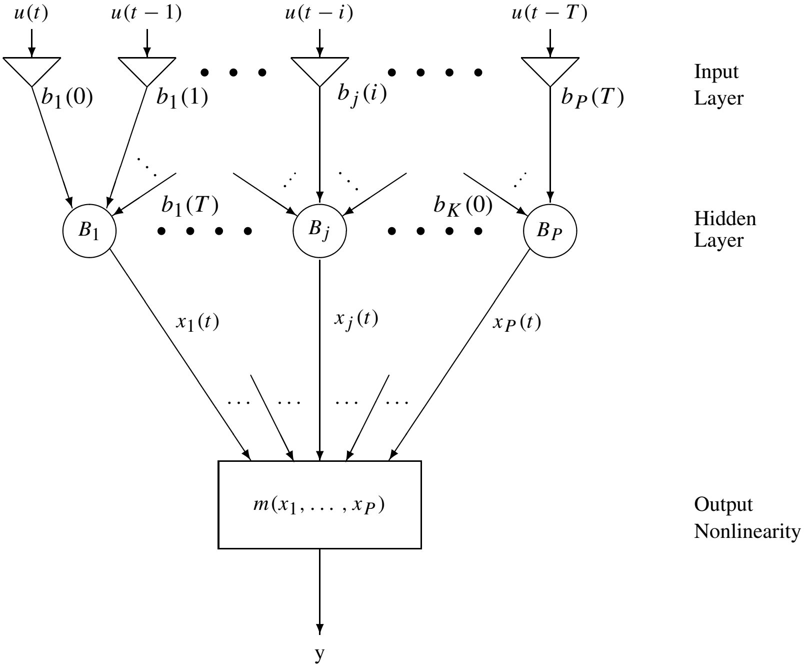

Figure 8.7 Block diagram of a polynomial function neural network (separable Volterra network). The nodes B_1 through B_L simply sum their inputs. The input $u(t-i)$ is multiplied by the weight $b_j(i)$, before being sent to node B_j. The outputs of the nodes are transformed by a multiple-input polynomial, $m(x_1, \dots, x_P)$.

The back-propagation algorithm, in its simplest form, is really just a gradient descent optimization (Rojas, 1996). The name "back-propagation" arises because each element in the gradient is computed from the inner product of the error vector and one column of the Jacobian [see equation (8.2)]. The name "back-propagation" is somewhat misleading, because it suggests that the error somehow travels backward through the net, which it does not. It does, however, enter into the computation of gradient, which in turn modifies the parameter values for the elements that are "behind" the error. Therefore the SVN approach may be regarded as equivalent to the identification of a parallel Wiener cascade by an iterative optimization.

8.1.6 Example: Using a Separable Volterra Network

Consider once more the example system, introduced in Section 7.2.4, where a LNLN cascade represents the relationship between the incident light and the electrical activity in the large mononuclear cells (LMC) in the fly retina. This section will use iterative optimization to fit a separable Volterra network (SVN) (a.k.a. parallel Wiener cascade) to the same simulated datasets used throughout Chapter 7.

The datasets consisted of 8192 points, sampled at 1 kHz, of input–output data, as described in Section 7.2.4. The optimal separable Volterra network was determined from the first 8000 data points using the Levenberg–Marquardt algorithm, see equation (8.15).

The remaining 192 points were used for model validation. Note, however, that for validation purposes, the model output was computed for the whole 8192 point input record and then subdivided into identification and validation segments. As a result, the 192-point cross-validation segment contained no transients.

8.1.6.1 Structure Selection

One difficulty with neural networks is that the network structure must be specified a priori. The structure of a separable Volterra network is determined by the number of parallel pathways, the length of the IRFs in the filter bank, and the order of the nonlinearity.

In Section 7.4.1 a Wiener–Bose model was constructed for this system using the method of "Principal Dynamic Modes" and determined to have three significant basis functions. This suggested that a SVN model would have three paths as well. However, Marmarelis (1997) suggested that convergence of SVN models could be improved by including additional pathways since this helps to avoid suboptimal local minima. Consequently, we chose a SVN model with five parallel paths for this example.

The length of the IRFs in the filter bank was determined by examining the Volterra kernels estimated for the system using the fast orthogonal algorithm (see Figure 7.4) and the Laguerre expansion technique (Figure 7.14). These indicate that the system memory was less than 50 ms, and so the IRF length was set to this value.

The nonlinearity order was chosen by trial and error. Initially, a second-order nonlinearity was used. The network was "trained" using a Levenberg–Marquardt optimization. The nonlinearity order was then increased. The IRFs and polynomial coefficients from the previously identified network were retained, while the higher-order coefficients were set to zero. The optimization was restarted using this expanded model structure. This procedure was repeated, until increasing the nonlinearity order did not significantly improve the prediction accuracy, as measured by the MDL criterion [see equation (5.17)]. On this basis, a fourth-order nonlinearity was selected for the network.

For each search, the values of the IRF weights and the first- and second-order polynomial coefficients were initialized from random samples of a white Gaussian distribution. The remaining polynomial coefficients were initialized to zero. One hundred iterations of the Levenberg–Marquardt algorithm were then used to find the parameter values for the network.

8.1.6.2 Results

Figure 8.8 shows the IRFs and polynomial nonlinearities estimated for the SVN model using a white noise input. This model accounted for 94.95% of the output variance for the identification segment and 93.38% in the validation segment. These results are slightly worse than those obtained by least-squares fitting the first- and second-order kernels, as reported in Chapter 7, where the FOA kernels predicted 94.58% VAF and the LET kernels predicted 93.70% VAF in the validation segment. The first and second-order Volterra kernels computed from this SVN model are shown in Figures 8.9A and 8.9B, respectively. Compare these with the kernels estimated by the FOA and LET from the same data (see Figures 7.4 and 7.15, respectively).

Figure 8.10 shows the IRFs and polynomials estimated using the colored noise input. It is evident that IRFs have more high-frequency noise than the IRFs estimated using white inputs, shown in Figure 8.8. This high-frequency noise is typical of least-squares estimates of systems when colored noise inputs are used (see, for example, Figure 5.4). Beyond this observation, it is difficult to compare the two models; the IRFs and nonlinearities found for

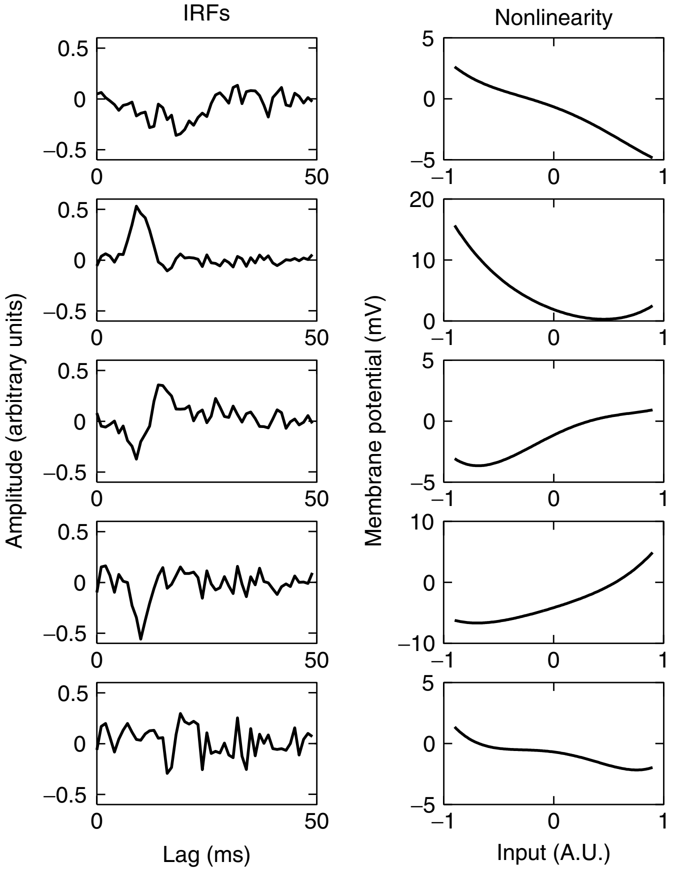

Figure 8.8 White-noise results: Elements of a separable Volterra network representation of the fly retina model. The five IRFs and polynomial nonlinearities were identified using the white-noise data.

each pathway depended on the initial values, which were selected randomly. Consequently, there is no relationship between corresponding elements in the two models.

The increased noise is also apparent in the first- and second-order Volterra kernels computed from this SVN model, shown in Figure 8.11. These kernels clearly have more high-frequency noise than those of Figure 8.9.

However, this high-frequency noise in the SVN model had little effect on its predictive power. The model has a 95.10% VAF for the training data and has a 93.40% VAF for the validation segment. This compares favorably with the kernel estimates obtained in the previous chapter, where the models identified by the FOA and LET accounted for 92.85% and 93.46% VAF, respectively. Furthermore, the SVN predictions remained excellent as is evident in Figure 8.12, which shows the noise-free system output, the additive output noise, and the output error (as compared to the *noise-free* output), for the validation segment. The output error is significantly smaller than the additive noise, indicating that the model prediction is actually closer to the true system output than is the measured output, which contains 13 dB of additive noise.

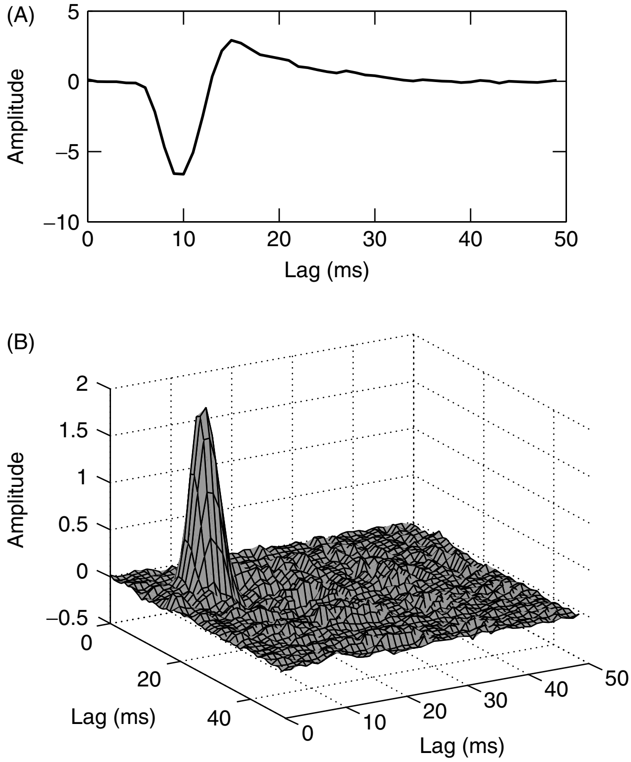

Figure 8.9 White-noise results: Estimates of the first- and second-order Volterra kernels of the fly retina system identified from the white noise data. These kernels were computed from the separable Volterra network shown in Figure 8.8. (A) First-order kernel. (B) Second-order kernel.

8.2 PARALLEL CASCADE METHODS

Section 8.1 described methods whereby gradient-based optimization methods can be used to construct iterative identification methods for model structures that are nonlinear in their parameters. The general procedure is first to select an appropriate model structure and then to group the variable parameters together in a parameter vector. Then, an iterative search is used to find the parameter set that minimizes the variance of the error between the system and model. Selecting an appropriate model structure is crucial since it remains fixed throughout the optimization; only the parameters are adjusted to fit the data.

This section will consider an alternate approach, illustrated in Figure 8.13, where the model structure becomes more complex with each iteration. First, fit a Wiener cascade between the input and output. Denote this Wiener cascade as $(h_1(\tau), m_1(\cdot))$, where $h_1(\tau)$ is the IRF of the linear element and $m_1(\cdot)$ is the nonlinearity. Usually, the nonlinearity will be represented by a polynomial, whose coefficients are denoted as $c_1^{(0)}, c_1^{(1)}, \ldots, c_1^{(q)}$. Compute the output, $\hat{y}_1(t)$, of this first cascade and subtract it from the measured output to give the first residue:

$$v_1(t) = z(t) - \hat{y}_1(t)$$

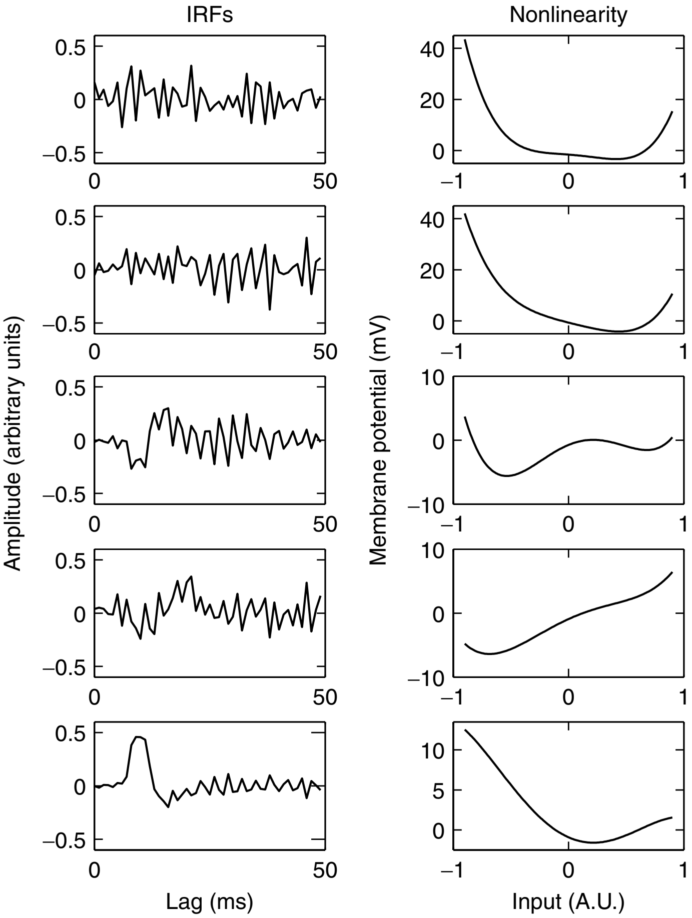

Figure 8.10 Colored-noise results: Elements of a separable Volterra network representation of the fly retina model. The five IRFs and polynomial nonlinearities were identified using colored-noise inputs.

Next, fit a second Wiener cascade, $(h_2(\tau), m_2(\cdot))$, between the input and the first residue. Compute the second residue, $v_2(t)$, by subtracting the output of the second path, $\hat{y}_2(t)$, from the first residue:

$$v_2(t) = v_1(t) - \hat{y}_2(t)$$

Continue this procedure until the residue cannot be distinguished from noise.

Throughout this section, subscripts will be used to indicate paths in the model. Thus, the kth path added to a model will be denoted as $(h_k(\tau), m_k(\cdot))$, and its output will be $\hat{y}_k(t)$. Furthermore, the output of the model comprising the first k paths of a parallel cascade will be

$$\hat{y}(k, t) = \sum_{i=1}^{k} \hat{y}_i(t)$$

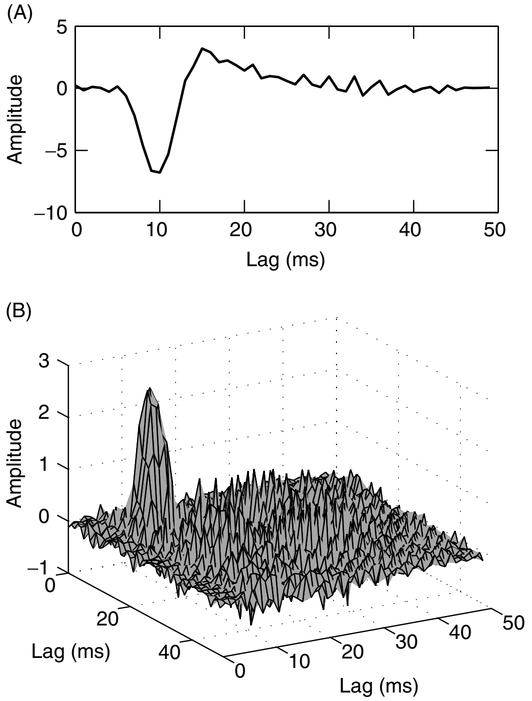

Figure 8.11 Colored-noise results: Estimates of the first and second-order Volterra kernels of the fly retina system. These were computed from the separable Volterra network identified from colored-noise data, shown in Figure 8.10. (A) First-order kernel. (B) Second-order kernel.

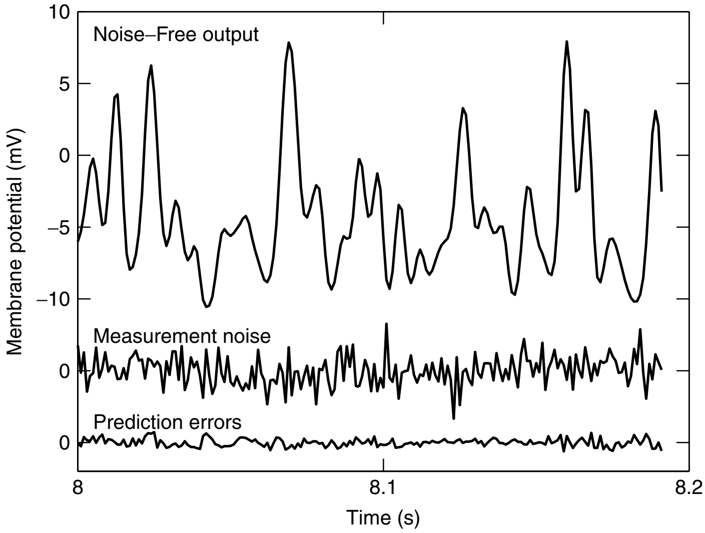

Figure 8.12 Model accuracy in the validation segment. The top trace shows the noise-free output of the fly retina model. The middle trace (displaced, but at the same scale) shows the additive output noise. The lowest trace shows the difference between the SVN output and the noise-free output of the simulation model.

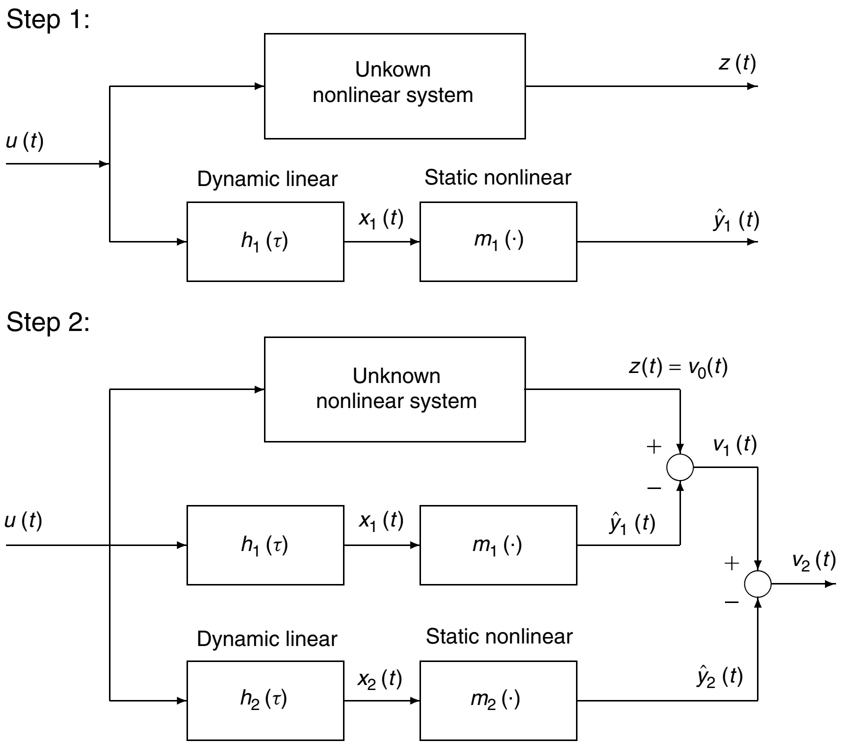

Figure 8.13 Constructing a parallel cascade model.

and the kth residue will be

$$\begin{aligned} v_k(t) &= v_{k-1}(t) - \hat{y}_k(t) \\ &= z(t) - \hat{y}(k, t) \end{aligned}$$

In some cases, it will be useful to think of the measured output as the "zeroth" residue:

$$v_0(t) = z(t)$$

This approach will generate a parallel Wiener cascade model. Section 4.4.1 showed that any system having a finite Volterra series representation can be described exactly by a parallel Wiener cascade. Therefore the question arises: "Will this iterative procedure actually construct an equivalent parallel cascade for an arbitrary finite Volterra series model?"

8.2.1 Parameterization Issues

In seeking to answer this question, it will be useful to relate the parallel cascade method to the gradient-based methods presented earlier. To do so, define the MSE cost function for the parallel cascade model,

$$V_N(k) = \frac{1}{2N} \sum_{t=1}^{N} (z(t) - \hat{y}(k, t))^2$$

Note that V_N is a function of the number of paths, k, which can only have positive integer values. This is quite different from the classical optimization methods, where the cost function depends on a vector of continuously variable parameters. Therefore, before the tools from the classical optimization framework can be used, V_N must be expressed as a function of a continuously variable parameter vector.

One possibility would be to use the parameters of the parallel cascade model directly. However, the parallel cascade procedure adds structural elements with each iteration, and the parameter values for each pathway are not altered once they have been added to the model. Consequently, directly parameterizing the parallel cascade model makes little sense, since to draw parallels with classical optimization methods the parameterization structure must remain constant throughout the identification. This suggests that the model's Volterra kernels, or more precisely the unique elements of its Volterra kernels, might be an appropriate choice. In particular, the parameter vector introduced in equation (7.6) provides a convenient representation for the parallel cascade model.

This parameterization has two major advantages. First, its structure will not change even though the parallel cascade structure grows with each iteration. Second, it is a unique parameterization since all equivalent representations of the system must have the same Volterra kernels. This bypasses problems caused by the nonuniqueness of parallel cascade models.

Recall, from Section 7.2, that the Volterra series model is linear in its parameters. Thus, it should be theoretically possible to determine the optimal Volterra series model by solving the normal equation in closed form. However, as noted in Chapter 7, this is not practical because the number of parameters in a Volterra series model is so large that computing the linear regression solution is computationally intractable. Indeed, the majority of Chapter 7 was devoted to developing solution strategies that were computationally practical. Nevertheless, because the Volterra series model is linear in its parameters, it will have a quadratic error surface with a single minimum. Consequently, iterative searches over its error surface can never get caught in local minima.

Therefore, the Volterra series parameterization is a useful theoretical framework for the analysis of parallel cascade models. In this framework, each parallel cascade model corresponds to a single point in the "parameter space" defined by its Volterra kernels. This point also defines a vector from the origin to the model parameters. Section 4.4 showed that the Volterra kernels for a parallel cascade model could be obtained by adding the Volterra kernels, of corresponding orders, of the individual pathways. Consequently, the point in parameter space corresponding to a parallel cascade model can be found by adding the vectors corresponding to the individual pathways.

The parallel cascade methods can be regarded similarly. Each pathway corresponds to a vector in the parameter space. The model comprising the first k pathways corresponds to the vector addition of the parameters defining the first k pathways. Thus, in this Volterra series framework the parallel cascade method can be regarded as an iterative search following a path defined by the vectors corresponding to the different pathways.

8.2.1.1 Convergence It is useful to consider the addition of each Wiener cascade to the model as comprising two steps. In the first step, a new linear element is chosen using methods to be specified later. Then in the second step, a polynomial is fit between the output of this new linear element and the current residue using linear regression.

The polynomial output is linear in its parameters, so the polynomial coefficients determined by linear regression will minimize the MSE for the particular linear element used in the Wiener cascade. Consequently, the resulting MSE must be less than or equal to that for any other choice of coefficients. Since setting all coefficients to zero would result in no change to the model, this implies that

$$V_N(k) \leq V_N(k-1)$$

That is, the value of the cost function decreases or stays constant with each additional pathway. Consequently, the parameter vector will descend toward the minimum in the cost function, just as in a classical optimization.

8.2.2 Testing Paths for Significance

One potential problem with the parallel cascade method is that insignificant pathways, which fit the measurement noise rather than the system's output, may be incorporated into the model. This may occur if inappropriate linear elements are chosen; it will always occur once the parallel cascade has modeled all the system's dynamics. This section will develop methods for detecting such spurious pathways so that they may be eliminated from parallel cascade models.

8.2.2.1 Hypothesis Testing One approach (Korenberg, 1991) is to test each new pathway against the hypothesis that it fits only noise. If the hypothesis cannot be rejected, the pathway is assumed to be modeling noise and is not included in the model. To construct this test, assume that the residue, $v_{k-1}(t)$, is an N point sample of white Gaussian noise with variance σ_v^2. Furthermore, assume that the kth pathway, fit between the input and this residue, has the output

$$\hat{y}_k(t) = \alpha w(t)$$

where $w(t)$ is an N-point sample of a unit variance Gaussian noise sequence that may or may not be white. Finally, assume that the processes $v_{k-1}(t)$ and $\hat{y}_k(t)$ are independent of each other.

Notice that the path output and residue are described in terms of ergodic processes. In practice, with finite length records, both signals will contain transients and hence be nonstationary. However, if the linear elements of the Wiener cascades are represented by FIR filters of length T, the transients will only effect the first $T-1$ samples. Thus, only the last $N_s = N - T + 1$ samples of the two signals can be used, since these will not contain any transients.

Since $\hat{y}_k(t)$ was obtained by a least-squares fit to the residue, α is given by

$$\begin{aligned}\alpha &= \frac{\displaystyle\sum_{t=T}^{N} w(t) v_{k-1}(t)}{\displaystyle\sum_{t=T}^{N} w^2(t)} \\ &= \frac{1}{N_s}\sum_{t=T}^{N} w(t) v_{k-1}(t)\end{aligned}$$

If $w(t)$ and $v_{k-1}(t)$ are independent, then the expected value of α will be zero, and its variance will be given by

$$\begin{aligned} E[\alpha^2] &= \frac{1}{N_s^2} E\left[\sum_{t=T}^{N} w(t)v_{k-1}(t) \sum_{s=T}^{N} w(s)v_{k-1}(s)\right] \\ &= \frac{1}{N_s^2} \sum_{t=T}^{N}\sum_{s=T}^{N} E\big[w(t)w(s)\big]E\big[v_{k-1}(t)v_{k-1}(s)\big] \\ &= \frac{1}{N_s^2} \sum_{t=T}^{N} E[w^2(t)]E[v_{k-1}^2(t)] \\ &= \frac{\sigma_v^2}{N_s} \end{aligned} \tag{8.22}$$

Thus, if the residue and path output are independent Gaussian noise sequences, α will be a zero-mean Gaussian random variable with variance given by equation (8.22). Hence, if the cascade path is modeling noise, there is approximately a 95% chance that the value of α will lie within a 2 standard deviation range of 0.

$$|\alpha| < \frac{2\sigma_v}{\sqrt{N_s}}$$

To apply this test, note that α^2 is the variance of the pathway output:

$$\alpha^2 = \frac{1}{N_s} \sum_{t=T}^{N} \hat{y}_k^2(t)$$

Thus, if

$$\frac{\sum_{t=T}^{N} \hat{y}_k^2(t)}{\sum_{t=T}^{N} v_{k-1}^2(t)} \geq \frac{2}{\sqrt{N-T+1}} \tag{8.23}$$

there will be a greater than 95% probability that the kth path is fitting system dynamics rather than noise, and therefore should be included in the model.

8.2.2.2 Parametric Model Structure Tests Another approach to determining the significance of a parallel cascade pathway is to use parametric model order/structure tests (Westwick and Kearney, 1997). These involve computing a cost function that incorporates factors that reward improvements in predictive ability and penalize increases in model complexity. A cascade path is only retained if it decreases the cost function.

A useful cost function is the minimum description length (MDL), defined in equation (5.17), where the sum of squared errors (SSE) is multiplied by a penalty term that depends on the number of model parameters. The relation is

$$\mathrm{MDL}(M) = \left(1 + \frac{M \log(N)}{N}\right) \sum_{t=1}^{N} (z(t) - \hat{y}(t, M))^2$$

where $\hat{y}(t, M)$ is an estimate of $z(t)$, produced by a model that has M free parameters.

To compute the MDL criterion for a parallel cascade model, the number of free parameters must be determined. Each path in a parallel cascade model consists of an IRF of length T in series with a polynomial of order Q. Thus, it would appear that a k path parallel Wiener cascade model would have $M = k(T + Q + 1)$ parameters. However, the parallel Wiener cascade contains redundant parameters that must be eliminated from the total. First, all but one of the zero-order polynomial coefficients are redundant since they each add a constant to the model output. Second, each Wiener cascade contains one redundant parameter, as described in Section 4.3.1; these must also be eliminated. Consequently, a parallel cascade model with k paths will have

$$M_k = 1 + k \cdot (T + Q - 1)$$

free parameters.

Therefore, the MDL for a parallel cascade model may be expressed as

$$\mathrm{MDL}(M_k) = \left(1 + \frac{(1 + k(T + Q - 1)) \log(N)}{N}\right) V_N(k) \tag{8.24}$$

The kth path should only be incorporated into the model if it reduces the MDL cost function—that is, if $\mathrm{MDL}(M_k) < \mathrm{MDL}(M_{k-1})$.

8.2.3 Choosing the Linear Elements

In terms of the underlying Volterra kernel parameterization, adding a path to a parallel cascade model can be thought of as the vector addition of the parameter vectors of the initial and new paths.

Consider the Volterra kernels, and hence parameter vector, of a Wiener element comprising an IRF, $h(\tau)$, and a Qth-order nonlinearity with polynomial coefficients $c^{(0)}, c^{(1)}, \dots, c^{(Q)}$. The first-order Volterra kernel will be

$$\mathbf{h}^{(1)}(\tau) = c^{(1)} h(\tau)$$

Thus, the set of possible first-order kernels corresponds to a line passing through the origin in the direction of the IRF, $h(\tau)$. Choosing $c^{(1)}$ determines the distance from the origin, along that line, of the first-order kernel.

Similarly, the second-order kernel of the Wiener cascade is

$$\mathbf{h}^{(2)}(\tau_1, \tau_2) = c^{(2)} h(\tau_1) h(\tau_2)$$

Again, the set of possible second-order kernels is a line, passing through the origin, in the Volterra kernel (parameter) space. The direction of this line is given by the second-order kernel, $h(\tau_1)h(\tau_2)$.

The same is true for each kernel and its corresponding polynomial coefficient. Thus, each kernel order corresponds to a subspace in the Volterra kernel space. The IRF of the linear part of the Wiener cascade defines Q vectors within this subspace. The span of these vectors defines a Q-dimensional subspace corresponding to the set of all possible Volterra kernels for a Wiener cascade with a given linear IRF. Using a linear regression to fit the polynomial finds the optimal point *on that particular subspace*.

Because each nonlinearity is fitted using linear regression, the sum of squared residues can never increase, whatever subspace is selected by the linear element. If all relevant directions are searched by the linear elements, then the residue will be minimized. The problem then is to ensure that all relevant directions are searched. The following sections will consider a variety of ways to achieve this.

8.2.3.1 Complete Expansion Bases A straightforward approach is to use an expansion basis that spans the complete search space. In principle, any complete expansion basis for the kernels could be used. For purposes of discussion, this section will consider basis filters whose IRFs are the standard unit vectors:

$$e_k(\tau) = \delta_{k,\tau}$$

where $\delta_{k,\tau}$ is a Kronecker delta.

Clearly, any first-order kernel of length T, $\mathbf{h}^{(1)}(\tau)$, can be represented as a weighted sum of these basis elements:

$$\mathbf{h}^{(1)}(\tau) = \sum_{k=0}^{T-1} \mathbf{h}^{(1)}(k) e_k(\tau)$$

Now, for a parallel Wiener cascade model, the first-order kernel is generated by the first-order polynomial coefficients from the static nonlinearities in all pathways. Thus, the first-order kernel of any system can be produced by a parallel Wiener cascade with T paths, one for each basis vector, having first-order polynomial coefficients equal to the corresponding first-order kernel values.

Next, consider the parallel cascade representation of a $T \times T$ second-order kernel, $\mathbf{h}^{(2)}(\tau_1, \tau_2)$. The second-order kernel of a parallel Wiener cascade is generated by the second-order polynomial coefficients. Therefore, the contribution to the second-order kernel of a single pathway having unit vector $e_k(\tau)$ and a polynomial with second-order coefficient $c_k^{(2)}$ is given by

$$\mathbf{h}_{\mathbf{k}}^{(2)}(\tau_1, \tau_2) = c_k^{(2)} e_k(\tau_1) e_k(\tau_2)$$

Therefore, a parallel cascade Wiener model with pathways corresponding to the T standard unit vectors can represent only the diagonal elements of a kernel. Additional Wiener pathways must be added to produce off-diagonal values.

To generate the off-diagonal elements, consider the effects of adding a Wiener cascade whose linear element is the sum of two unit vectors, $h(\tau) = e_j(\tau) + e_k(\tau)$. The second-order kernel contribution from this pathway would be

$$\mathbf{h}_{\mathbf{jk}}^{(2)}(\tau_1, \tau_2) = c_{jk}^{(2)}(e_j(\tau_1)e_j(\tau_2) + e_j(\tau_1)e_k(\tau_2) + e_k(\tau_1)e_j(\tau_2) + e_k(\tau_1)e_k(\tau_2))$$

Thus, this Wiener path contributes to two diagonal kernel elements, $\mathbf{h}^{(2)}(j, j)$ and $\mathbf{h}^{(2)}(k, k)$, and to two symmetric, off-diagonal elements, $\mathbf{h}^{(2)}(j, k)$ and $\mathbf{h}^{(2)}(k, j)$. Therefore, to fit an arbitrary second-order kernel completely, the parallel-cascade Wiener model must include pathways with IRFs equal to each basis element and to the sum of every possible pair of basis elements. Similarly, the exact representation of an arbitrary third-order kernel will require pathways with IRFs equal to every possible sum of one, two, or three distinct basis elements. In general, the order-q kernel will require Wiener cascades having linear elements that are sums of all possible combinations of between 1 and q basis elements. Thus, using a complete set of basis functions will often result in parallel cascade models having an intractably large number of pathways.

8.2.3.2 Random Basis Vectors Another approach would be to select the filters randomly, rather than sequentially. In principle, it is possible to use vectors of random numbers as the IRFs in the parallel cascade. Indeed, French et al. (2001) used this approach to identify the dynamics of mechano-receptor neurons. The resulting model included over 100,000 parallel Wiener cascades. The individual pathways were not retained. Rather, each new path was used to refine estimates of the first- and second-order Volterra kernels and then discarded.

This points out the major disadvantage of the approach: Many Wiener paths are required to capture the dynamics of even simple systems. This is likely to be very inefficient since many of the paths will account for only very small fractions of the system's output. Indeed, since random basis vectors need not be orthogonal, it may require more pathways than would be necessary if a complete basis set were used.

8.2.3.3 Incomplete Basis Set One approach to reducing the complexity of a parallel cascade model is to use an incomplete basis for the linear filters. For example, the set of Laguerre filters as described in Chapter 7 is one such possibility. Using an incomplete basis expansion will reduce the number of Wiener cascades included in the expansion. However, the paths used to construct the model depend only on the expansion basis and not on the dynamics of the system being modeled. Consequently, if the basis set selected cannot describe the dynamics of the system fully, the resulting model may be badly biased.

8.2.3.4 Slices of Correlation Functions It seems evident that the efficiency of parallel cascade models would be improved and the risk of bias decreased, if the IRFs in the individual Wiener paths were related to the dynamics of the unknown system. One possibility, proposed by Korenberg (1991), is to use slices of input-residue correlation functions of various orders. These correlation functions will contain information about the unmodeled dynamics and thus should improve the convergence of the parallel cascade expansion.

Indeed, Korenberg (1991) developed lower bounds on the reduction in the residual variance due to the addition of these cascade paths. His analysis proceeds as follows. First, fit the kth pathway between the input and the $(k-1)$th residue, $v_{k-1}(t)$. Let the IRF of the kth linear element be equal to the first-order, input-residue cross-correlation*

$$h_k(\tau) = \phi_{uv_{k-1}}(\tau) \tag{8.25}$$

*Note that the first pathway will be fitted between the input and the "zeroth" residue $v_0(t) = z(t)$.

Find the optimal linear gain between $x_k(t)$, the output of this IRF, and the current residue using least-squares. The resulting reduction in the residual variance will be less than, or equal to, that produced by fitting a higher-order polynomial nonlinearity. Consequently, the variance reduction produced by the linear gain provides a lower bound on the reduction produced by the cascade path.

The optimal linear gain, $c^{(1)}$, is found by least-squares fitting between the IRF output, $x_k(t)$, and $v_{k-1}(t)$, the current residue:

$$c^{(1)} = \frac{\sum_{t=1}^{N} x_k(t) v_{k-1}(t)}{\sum_{t=1}^{N} x_k^2(t)}$$

Since the result is a least-squares fit, the reduction in the residual variance is equal to the variance of the term. Thus,

$$\begin{aligned}
\Delta_{V_N} &= \frac{1}{N} \sum_{t=1}^{N} \left(c^{(1)} x_k(t) \right)^2 \\
&= \frac{1}{N} \frac{\left(\sum_{t=1}^{N} x_k(t) v_{k-1}(t) \right)^2}{\sum_{t=1}^{N} x_k^2(t)} \\
&= \frac{1}{N} \frac{\left(\sum_{t=1}^{N} \sum_{i=0}^{T-1} h_k(i) u(t-i) v_{k-1}(t) \right)^2}{\sum_{t=1}^{N} \sum_{i,j=0}^{T-1} h_k(i) h_k(j) u(t-i) u(t-j)} \\
&= \frac{\left(\sum_{i=0}^{T-1} \phi_{uv_{k-1}}^2(i) \right)^2}{\sum_{i,j=0}^{T-1} \phi_{uv_{k-1}}(i) \phi_{uv_{k-1}}(j) \phi_{uu}(i-j)}
\end{aligned}$$

To find a lower bound on Δ_{V_N}, find a convenient *upper* bound on the denominator. Since $i = j$ maximizes $\phi_{uu}(i-j)$, let $i = j$ and choose the value of i which maximizes $\phi_{uv_{k-1}}(i)$. Expanding the cross-correlations, the denominator becomes

$$\frac{T^2}{N^2} \phi_{uu}(0) \left(\sum_{t=1}^{N} u(t-i) v_{k-1}(t) \right)^2$$

Then, using the Cauchy–Schwartz inequality,

$$\left(\sum_{t=1}^{N} u(t-i)v_{k-1}(t)\right)^2 \le \left(\sum_{t=1}^{N} u^2(t-i)\right)\left(\sum_{t=1}^{N} v_{k-1}^2(t)\right)$$

produces an upper bound on the denominator of Δ_{V_N},

$$T^2\phi_{uu}^2(0)\phi_{v_{k-1}v_{k-1}}(0)$$

and hence a lower bound on the variance reduction.

Thus, if $h_k(\tau)$ is determined using equation (8.25), the reduction in the residual variance will be at least

$$V_N(k-1) - V_N(k) \ge \frac{\left(\sum_{i=0}^{T-1} \phi_{uv_{k-1}}^2(i)\right)^2}{T^2\sigma_u^4\sigma_{v_{k-1}}^2}$$

This lower bound depends only on the variances of the input and residue, and on the absolute value of the input-residual cross-correlation. The reduction in residual variance will be greater than zero unless the residue is completely uncorrelated from the input.*

Once the first-order correlation has been driven to zero (i.e., the residue is completely uncorrelated with the input), IRFs must be obtained from slices of the second- and higher-order input-residual cross-correlation functions. Deriving the bounds on the reduction in variance for these paths is more difficult. Indeed, in order to achieve a guaranteed reduction in the residual variance, Korenberg (1991) had to add impulses to the slices of higher-order correlation functions. For example, the IRF of Wiener cascade path based on the lth slice of the second-order cross-correlation was constructed as

$$h_k(\tau) = \phi_{uuv_{k-1}}(l,\tau) \pm \alpha_k\delta(\tau - l) \tag{8.26}$$

where α_k is a constant, chosen such that the sequence goes to zero as the sequence of residual variances. For example, it could be defined to be

$$\alpha_k = \frac{\|v_{k-1}\|_2^2}{\|z\|_2^2}$$

As with the first-order pathway, a lower bound on the variance reduction may be obtained from the least-squares fit of a single term in the polynomial nonlinearity. If the IRF was taken from the second-order cross-correlation, the lower bound would be obtained by least-squares fitting the quadratic term in the nonlinearity. Thus, the optimal quadratic gain is

$$c^{(2)} = \frac{\sum_{t=1}^{N} x_k^2(t)v_{k-1}(t)}{\sum_{t=1}^{N} x_k^4(t)}$$

*If the input is white, then it can be shown that this procedure will reduce the first-order input-residual correlation to zero (Korenberg, 1982).

As before, $c^{(2)}x_k^2(t)$ is a least-squares fit to the residual, $v_{k-1}(t)$. Thus, the reduction in the mean-square error is equal to the variance of $c^{(2)}x_k^2(t)$. Thus,

$$\begin{aligned}\Delta_{V_N} &= \frac{1}{N}\sum_{t=1}^{N}\left(c^{(2)}x_k^2(t)\right)^2 \\ &= \frac{1}{N}\sum_{t=1}^{N}\frac{\left(\sum_{t=1}^{N}x_k^2(t)v_{k-1}(t)\right)^2}{\sum_{t=1}^{N}x^4(t)}\end{aligned}$$

As for the first-order case, an upper bound on the denominator which depends only on the even moments of the input and residue can be found using the Cauchy–Schwartz inequality. For example, if the input is Gaussian, then $x_k(t)$ will also be Gaussian, and

$$E[x^4] = 3E[x^2]^2$$

Thus

$$3T^4\sigma_u^8\sigma_{v_{k-1}}^4$$

will be an upper bound on the denominator of Δ_{V_N}.

Next, consider the numerator of Δ_{V_N}.

$$\begin{aligned}\sum_{t=1}^{N}x_k^2(t)v_{k-1}(t) &= \sum_{t=1}^{N}\sum_{i,j=0}^{T-1}h_k(i)k_k(j)u(t-i)u(t-j)v_{k-1}(t) \\ &= \sum_{i,j=0}^{T-1}\phi_{uuv_{k-1}}(l,i)\phi_{uuv_{k-1}}(l,j)\phi_{uuv_{k-1}}(i,j) \\ &\quad \pm 2\alpha_k\sum_{i=0}^{T-1}\phi_{uuv_{k-1}}^2(l,i) + \alpha_k^2\phi_{uuv_{k-1}}(l,l)\end{aligned}$$

Squaring this produces the numerator. Note that the middle term will be nonzero if any element in the slice of the second-order cross-correlation is nonzero. Thus, constructing Wiener cascade paths using equation (8.26) will reduce the residual variance until the second-order input-residue correlation has been reduced to zero.

In general, the IRF of the linear elements of a parallel cascade can be chosen to be a one-dimensional slice of a higher-order correlation function with discrete deltas added to its diagonal points. For example, to base a Wiener cascade on slices of the third-order cross-correlation, the expressions would be

$$h_k(\tau) = \phi_{uuuv_{k-1}}(\tau, l_1, l_2) \pm \alpha_1\delta(\tau - l_1) \pm \alpha_2\delta(\tau - l_2) \tag{8.27}$$

where l_1 and l_2 are randomly selected indices between 0 and $T-1$, and $\delta(\tau)$ is a Kronecker delta. The constants α_1 and α_2 would have randomly chosen signs and magnitudes which would go to zero with the variance of the residue. Korenberg (1991) suggested using

$$\alpha = \frac{\sum_{t=1}^{N} v_{k-1}^2(t)}{\sum_{t=1}^{N} z^2(t)} \tag{8.28}$$

to determine these constants. Analogous expressions can be derived for higher-order correlation functions.

This approach guarantees convergence, but it often finds many insignificant paths that must be rejected using significance tests such as inequality (8.23) or equation (8.24). Furthermore, it is difficult to establish whether a particular expansion has captured all the dynamics of the system. Only by testing all possible pathways can the expansion be said to be complete.

8.2.3.5 The Eigenvector Method Westwick and Kearney (1994, 1997) extended Korenberg's analysis to decrease the number of pathways required to achieve a given accuracy and simplify the stopping criterion. Their approach was to find the cascade paths, from a particular high-order input-residual cross-correlation function, that reduced the residual variance the most.

Paths Based on the First-Order Correlation The first pathway is estimated from the first-order correlation as follows. Consider a "Wiener cascade" consisting of a linear filter followed by a linear, unity-gain element. The general approach is to first find the linear filter that minimizes the residual variance for this first-order "nonlinearity" and then fit a high-order polynomial between the output of this optimal filter and the residue.

The initial minimization is achieved by identifying the optimal linear filter between the input and output. This can be accomplished using equation (5.10),

$$\mathbf{h_1} = \mathbf{\Phi_{uu}^{-1}} \boldsymbol{\phi}_{\mathbf{uv_0}} \tag{8.29}$$

If the first-order "nonlinearity" is used, the residue would be

$$v_1(t) = v_0(t) - x_1(t)$$

with variance,

$$\begin{aligned}\sigma_{v_1}^2 &= E(v_0(t) - x_1(t))^2 \\ &= \sigma_{v_0}^2 - 2E[v_0(t)x_1(t)] + E[x_1^2(t)]\end{aligned}$$

However, since

$$\begin{aligned}E[x_1^2(t)] &= E\left[\left(\sum_{\tau=0}^{T-1} h_1(\tau)u(t-\tau)\right)^2\right] \\ &= \sum_{\tau_1=0}^{T-1}\sum_{\tau_2=0}^{T-1} h_1(\tau_1)h_1(\tau_2)E[u(t-\tau_1)u(t-\tau_2)] \\ &= \mathbf{h_1^T \Phi_{uu} h_1}\end{aligned}$$

and

$$\begin{aligned}E[v_0(t)x_1(t)] &= E\left[v_0(t)\sum_{\tau=1}^{T-1} h_1(\tau)u(t-\tau)\right]\\ &= \sum_{\tau=1}^{T-1} h_1(\tau)E[v_0(t)u(t-\tau)]\\ &= \sum_{\tau=1}^{T-1} h_1(\tau)\phi_{uv_0}(\tau)\\ &= \mathbf{h_1^T\Phi_{uu}h_1}\end{aligned}$$

using only the first-order "nonlinearity" reduces the residual variance by

$$\sigma_{v_0}^2 - \sigma_{v_1}^2 = \mathbf{h_1^T\Phi_{uu}h_1} = \boldsymbol{\phi}_{\mathbf{uy}}^{\mathbf{T}}\mathbf{\Phi_{uu}^{-1}}\boldsymbol{\phi}_{\mathbf{uy}} \tag{8.30}$$

Thus, if the first-order input-residue cross-correlation is nonzero, equation (8.30) gives the reduction in the residual variance due to a linear pathway, identified using equation (8.29).

Fitting a Higher-Order Nonlinearity Now, use a linear regression to fit a high-order polynomial between $x_1(t)$ and $v_0(t)$. Update the first Wiener cascade to use $m_1(\cdot)$ as its nonlinearity. Since the high-order polynomial was fit using least-squares, including it will either reduce the residual variance or leave it unchanged.

Suppose that, after $m_1(\cdot)$ was fitted, the first-order cross-correlation between $u(t)$ and $v_1(t)$ was nonzero.

$$\begin{aligned}\phi_{uv_1}(\tau) &= E\left[u(t-\tau)\left(v_0(t) - m_1(x_1(t))\right)\right]\\ &= \phi_{uv_0}(\tau) - E[u(t-\tau)m_1(x_1(t))]\end{aligned}$$

Note that, by Bussgang's theorem [see equation (6.46)], $\phi_{uv_1}(\tau)$ will be proportional to $\phi_{uv_0}(\tau)$, provided that the input, $u(t)$, is Gaussian.

Thus, if $\phi_{uv_1}(\tau) \neq 0$, then applying equation (8.29) will produce an IRF, $h_2(\tau)$, which is a scaled copy of $h_1(\tau)$, and equation (8.30) will guarantee a reduction in the residual variance. However, this reduction in the residual variance could also be obtained by changing the first-order polynomial coefficient of $m_1(\cdot)$, which clearly isn't possible, since $m_1(\cdot)$ was fitted using least-squares.

Consequently, the first pathway, whose IRF is fitted using equation (8.29), will reduce the first-order input-residue cross-correlation to zero. Furthermore, fitting a high-order polynomial between $x_1(t)$ and $v_0(t)$ may reduce the residual variance, but will not change $\phi_{uv_1}(\tau)$, which will remain equal to zero. This procedure finds the optimal first cascade path in the sense it minimizes the norm of the first-order cross-correlation between the input and residue. There is, however, no guarantee that this will minimize the variance of the residue.

Use of a Pseudo-Inverse Chapter 5 showed that the deconvolution of the input autocorrelation from the input–output cross-correlation can encounter severe numerical conditioning problems when the input signal is band-limited. The same problem exists when estimating the IRF of a Wiener cascade. Indeed, numerical conditioning problems are likely to be more important since the signal-to-noise ratio will decrease progressively as additional pathways are added to the model.

The treatment of linear IRF estimation given in Chapter 5 argued that robust estimates could be obtained by replacing the matrix inversion in equation (5.10) with a pseudo-inverse (5.16) to eliminate ill-conditioned terms. Efficient selection of the terms to retain was made possible by computing the MDL implicitly, as described in equation (5.22). Using this approach, the output variance accounted for by each term was computed, the terms were sorted in decreasing order of significance, and then only those terms that reduced the MDL cost function were retained.

Ideally, the first pathway should reduce the first-order cross-correlation between the input and residue to zero. However, with noisy, finite-length records, all that can be achieved is to reduce the first-order correlation to noise. Thus, rather than using the exact, MMSE solution given in equation (5.10) as the IRF of the linear element, the pseudo-inverse-based solution, described in Section 5.2.3.4, should be used. The terms eliminated from the pseudo-inverse, whose contributions were indistinguishable from noise, will remain in the first-order cross-correlation. This will result in the first Wiener cascade, $(h_1(\tau), m_1(\cdot))$, reducing the first-order input-residue cross-correlation, $\phi_{uv_1}(\tau)$, so that it contains only noise. Consequently, even though $\phi_{uv_1}(\tau)$ is not exactly zero, it cannot be used to construct the second Wiener cascade, $(h_2(\tau), m_2(\cdot))$.

Second-Order Correlations Additional pathways must be obtained from the second-order input-residue correlations. The optimal pathway may be determined by extending the analysis used to derive the optimal first path as follows. Consider a Wiener cascade whose static nonlinearity is a second-order polynomial. If $u(t)$ is Gaussian, as assumed, and the linear element, $h_2(\tau)$, is scaled so that its output, $x_2(t)$, has unit variance, then the (un-normalized) Hermite polynomials will be orthogonal for this input. Thus, the natural choice for the second-order nonlinearity would be a second-order Hermite polynomial. The problem is to find the second-order Hermite polynomial coefficient, $\gamma_2^{(2)}$, and normalized impulse response, $h_2(\tau)$, that minimize the residual variance.

If $h_2(\tau)$ is followed by a second-order Hermite polynomial, the residue will be

$$v_2(t) = v_1(t) - \gamma_2^{(2)}(x_2^2(t) - 1) \tag{8.31}$$

with variance

$$\begin{aligned}
\sigma_{v_2}^2 &= E\left[(v_1(t) - \gamma_2^{(2)} x_2^2(t) + \gamma_2^{(2)})^2\right] \\
&= E[v_1^2] - 2\gamma_2^{(2)} E[x_2^2 v] + 2\gamma_2^{(2)} E[v_1] + \left(\gamma_2^{(2)}\right)^2 E[x_2^4] \\
&\quad -2\left(\gamma_2^{(2)}\right)^2 E[x_2^2] + \left(\gamma_2^{(2)}\right)^2 \\
&= \sigma_{v_1}^2 - 2\gamma_2^{(2)} E[v_1(t) x_2^2(t)] + 2\left(\gamma_2^{(2)}\right)^2
\end{aligned} \tag{8.32}$$

Minimizing equation (8.32) with respect to the second-order polynomial coefficient, $\gamma_2^{(2)}$, yields

$$\gamma_2^{(2)} = \frac{E[v_1(t) x_2^2(t)]}{2}$$

Substituting this into equation (8.32) shows that the optimal polynomial coefficient will reduce the residual variance by

$$\sigma_{v_1}^2 - \sigma_{v_2}^2 = \frac{1}{2}\left(E[v_1(t)x_2^2(t)]\right)^2$$

The residual variance reduction can be expressed in terms of the second-order input-residue cross-correlation since

$$\begin{aligned} E[v_1(t)x_2^2(t)] &= E\left[v(t)\left(\sum_{\tau=0}^{T-1} h_2(\tau)u(t-\tau)\right)^2\right] \\ &= \sum_{\tau_1=0}^{T-1}\sum_{\tau_2=0}^{T-1} h_2(\tau_1)h_2(\tau_2)E[u(t-\tau_1)u(t-\tau_2)v_1(t)] \\ &= \mathbf{h_2^T}\boldsymbol{\phi}_{\mathbf{uuv_1}}\mathbf{h_2} \end{aligned}$$

To maximize $\mathbf{h_2^T}\boldsymbol{\phi}_{\mathbf{uuv_1}}\mathbf{h_2}$, first express $h_2(\tau)$ as a weighted sum,

$$\mathbf{h} = \sum_{k=1}^{T} a_k \mathbf{g_k} \qquad (8.33)$$

where (λ_k, g_k), with $|\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_T|$, are the generalized eigenvalues and eigenvectors (Golub and Van Loan, 1989) of the matrix pencil $(\boldsymbol{\phi}_{\mathbf{uuv}}, \boldsymbol{\Phi}_{\mathbf{uu}})$. Thus, (λ_k, g_k) satisfy

$$\boldsymbol{\phi}_{\mathbf{uuy}}\mathbf{g_k} = \lambda_k \boldsymbol{\Phi}_{\mathbf{uu}}\mathbf{g_k} \qquad (8.34)$$

$$\mathbf{g_j^T}\boldsymbol{\Phi}_{\mathbf{uu}}\mathbf{g_k} = \delta_{jk} \qquad (8.35)$$

Now, the variance reduction obtained using the second-order Hermite polynomial may be expressed in terms of the expansion coefficients, a_k, and the generalized eigenvalues, λ_k.

$$\begin{aligned} \mathbf{h_2^T}\boldsymbol{\phi}_{\mathbf{uuv_1}}\mathbf{h_2} &= \sum_{i,j=1}^{T} a_i a_j \mathbf{g_i^T}\boldsymbol{\phi}_{\mathbf{uuv_1}}\mathbf{g_j} \\ &= \sum_{i,j=1}^{T} a_i a_j \lambda_j \mathbf{g_i^T}\boldsymbol{\Phi}_{\mathbf{uu}}\mathbf{g_j} \\ &= \sum_{i,=1}^{T} a_i^2 \lambda_i \end{aligned} \qquad (8.36)$$

where the last equality depends on equation (8.35). Using equation (8.36), the variance reduction can be expressed in terms of the expansion weights, a_i, and the generalized eigenvalues, λ_i:

$$\sigma_{v_1}^2 - \sigma_{v_2}^2 = \frac{1}{2}\left(\sum_{i,=1}^{T} a_i^2 \lambda_i\right)^2 \tag{8.37}$$

Recall, however, that $h_2(\tau)$ was scaled to have unit output variance and that the variance of $x_2(t)$ can be written in terms of the expansion coefficients, a_j, as follows:

$$\begin{aligned}\sigma_{x_2}^2 &= \mathbf{h_2^T \Phi_{uu} h_2}\\ &= \sum_{i,j=1}^{T} a_i a_j \mathbf{g_i^T \Phi_{uu} g_j}\\ &= \sum_{j=1}^{T} a_j^2\end{aligned}$$

Thus, to obtain the maximum reduction in variance possible with the second-order Hermite polynomial term, maximize equation (8.37) subject to the constraint that

$$\sum_{j=1}^{T} a_k^2 = 1 \tag{8.38}$$

By assumption, the eigenvalues are arranged in decreasing order of magnitude. Therefore the minimum residual variance is achieved when $a_1 = 1$, with the rest of the a_i equal to zero. The corresponding residual variance is given by

$$\sigma_{v_2}^2 = \sigma_v^2 - \frac{\lambda_1^2}{2}$$

Therefore, adding the pathway to the model will reduce the variance by $\lambda_1^2/2$. Moreover, the other generalized eigenvectors of the correlation matrix, which represent the dynamics of less significant "system modes," are unaffected. This can be shown by examining the second-order cross-correlation between the input and residue, expressed as

$$\boldsymbol{\phi}_{\mathbf{uuv_2}} = \boldsymbol{\phi}_{\mathbf{uuv_1}} - \lambda_1 \mathbf{\Phi_{uu}}(\mathbf{g_1 g_1^T})\mathbf{\Phi_{uu}}$$

Now consider the matrix pencil $(\boldsymbol{\phi}_{\mathbf{uuv_2}}, \mathbf{\Phi_{uu}})$. From

$$\begin{aligned}\boldsymbol{\phi}_{\mathbf{uuv_2}}\mathbf{g_i} &= \boldsymbol{\phi}_{\mathbf{uuv_1}}\mathbf{g_i} - \lambda_1 \mathbf{\Phi_{uu} g_1 g_1^T \Phi_{uu} g_i}\\ &= \lambda_i \mathbf{\Phi_{uu} g_i} - \lambda_1 \mathbf{\Phi_{uu} g_1}\delta_{1,i}\\ &= \begin{cases} 0 & \text{for } i = 1\\ \lambda_i \mathbf{\Phi_{uu} g_i} & \text{otherwise}\end{cases}\end{aligned}$$

it is evident that it has the same generalized eigenvectors as the original pencil $(\boldsymbol{\phi}_{\mathbf{uuv_1}}, \mathbf{\Phi_{uu}})$, except that the eigenvalue associated with $\mathbf{g_1}$ is zero.

Incorporation of a Higher-Order Nonlinearity Replacing the second-order polynomial nonlinearity with a higher-order polynomial will scale the second-order cross-correlation across the Wiener path (Hunter and Korenberg, 1986). Thus, after re-computing the residue, $v_2(t)$, using a different nonlinearity, the g_i will continue to be the generalized eigenvectors of $(\boldsymbol{\phi}_{\mathbf{uuv}}, \boldsymbol{\Phi}_{\mathbf{uu}})$, and the only eigenvalue affected will be λ_1.

However, if the new nonlinearity is fitted using linear regression, then the principal generalized eigenvalue of $(\boldsymbol{\phi}_{\mathbf{uuv_1}}, \boldsymbol{\Phi}_{\mathbf{uu}})$ will be reduced to zero, just as it was by the second-order polynomial nonlinearity. If this were not the case, $\boldsymbol{\phi}_{\mathbf{uuv_2}}\mathbf{g_1} = \lambda\boldsymbol{\Phi}_{\mathbf{uu}}\mathbf{g_1}$, and the residual variance could be reduced by $\lambda^2/2$, using a second-order nonlinearity. Clearly, this is not possible since the nonlinearity was fitted using least-squares. Therefore, as with the second-order nonlinearity, $\boldsymbol{\phi}_{\mathbf{uuv_2}}$ will have the same eigenvectors and eigenvalues as $\boldsymbol{\phi}_{\mathbf{uuv_1}}$, with the exception of $\mathbf{g_1}$, whose eigenvalue will be zero in the subsequent correlation matrix.

The cascade estimated in this way is optimal in the sense that it reduces the largest eigenvalue to zero and thus produces the largest possible reduction in the norm of the second-order cross-correlation between the input and residue. However, as with the optimal first-order pathway, there is no guarantee that it will also minimize the residual variance.

Low Rank Projection If the input is band-limited, the generalized eigenvalue problem may become ill-conditioned, in a manner similar to other least-squares estimates developed in this text. As before, the ill-conditioning may be alleviated by projecting the solution onto the significant singular vectors of the input autocorrelation matrix. The difficulty is to determine which terms to retain and which ones to eliminate.

Previous pseudo-inverse applications used implicit, correlation-based computations of the residual variance to sort the terms in decreasing order of significance. These terms were then used to compute the MDL and thus determine which terms should be retained. Unfortunately, in the present case it is not practical to compute the residual variance implicitly. However, the variance of the contribution to the intermediate signal, $x_2(t)$, can be computed implicitly and used to sort the terms. These sorted terms may then be included, one by one, into the cascade path until the explicitly computed MDL starts to increase.

Stopping Conditions Pathways based on the second-order input-residue cross-correlation can be added until all significant eigenvalues have been reduced to zero. This is most easily determined by testing each new pathway for significance, using either the hypothesis test (8.23), or the MDL (8.24) as it is added to the model. Once a cascade path fails the significance test, it can be concluded that all possible pathways have been derived from the second-order correlation. This is a consequence of the selection procedure that proposes the most significant pathway based on the second-order input-residue cross-correlation at each iteration. If this does not account for a statistically significant fraction of the residual variance, then clearly none of the other, less significant ones will either. Additional paths must be derived from third- or higher-order correlations.

8.2.3.6 Optimization Methods Another possibility, mentioned by Korenberg (1991), would be to use an iterative optimization to fit each Wiener cascade path in turn. While no details were given, two practical issues must be resolved.

1. How would the cascade paths be initialized? There are many possibilities. For example, the initial cascade path could be computed using either of the correlation-based techniques discussed above. Alternately, randomly selected elements from an expansion basis could be used. However, depending on how the optimization was initialized, it might find the globally optimal Wiener cascade, or it might converge to a suboptimal local minimum. This is not as serious as with a classical optimization problem since convergence is nevertheless guaranteed. Thus, adding a suboptimal cascade to the model will only, at worst, increase the number of paths needed for convergence.
2. How would convergence be detected? Individual pathways can be tested for significance, using either of the tests described in this chapter. However, because iterative optimization cannot be guaranteed to find a global optimum, there can be no guarantee that another, more significant pathway does not exist. Consequently, finding a pathway that fails the significance test cannot be taken as evidence that the procedure has converged.

8.2.4 Parallel Wiener Cascade Algorithm

The following algorithm, based on the generalized eigenvector algorithm (Westwick and Kearney, 1997), summarizes how optimization methods (Korenberg, 1991) may be incorporated into a parallel cascade identification procedure.

1. Initialization: Set $v_0(t) = z(t)$, and set the path number $k = 1$.
2. Use the pseudo-inverse-based method, described in Section 5.2.3.4, to fit a linear impulse response, $h_1(\tau)$, between $u(t)$ and $v_0(t)$. Compute its output, $x_1(t)$, via convolution.
3. Use linear regression to fit the first polynomial nonlinearity between $x_1(t)$ and $v_0(t)$. Choose the polynomial order that minimizes the MDL criterion.
4. If no polynomial reduces the MDL, reject the first-order pathway and proceed to step 6.
5. Set the path number $k = 2$.
6. Use the generalized eigenvector method in conjunction with a low-rank projection, as described on pages 238–241, to fit an IRF, based on the second-order cross-correlation between $u(t)$ and $v_{k-1}(t)$.
7. Compute its output, $x_k(t)$, using convolution.
8. Use linear regression to fit the kth polynomial nonlinearity between $x_k(t)$ and $v_{k-1}(t)$. Choose the polynomial order that minimizes the MDL criterion.
9. Optionally, use an iterative optimization, such as the Levenberg–Marquardt algorithm, described in Section 8.1.3, to refine the pathway.
10. If the inclusion of the kth pathway causes the MDL to decrease, set $k = k + 1$ and return to step 6.
11. Since the kth path caused the MDL to increase, discard it and stop the identification.

8.2.5 Longer Cascades

So far, only parallel Wiener cascade models have been considered. Longer cascades paths (e.g., LNL, LNLN) may also be used (Korenberg, 1991). This can be shown by examining the convergence proof for the parallel Wiener cascade. Convergence depends primarily on fitting the final element in the cascade using least-squares regression. However, irrespective of whether a cascade finishes with a linear element (e.g., LNL) or a static nonlinearity (e.g., LNLN), the path's output is linear in the parameters that describe the last element. Thus, the final element can be fitted using least-squares regression. As a result, convergence is ensured for longer cascades, provided that the elements of the cascade paths search all possible directions in the Volterra kernel space.

The rationale for using longer pathways is that each path should be capable of representing a wider class of systems. Thus, the parallel cascade model should require fewer pathways to achieve a given accuracy. However, the increased time required to fit each pathway may offset the reduction in the number of pathways. We are only aware of one reference, an unpublished student project (Mo and Elkasabgy, 1984), that has demonstrated faster identification by using LNL as opposed to Wiener cascades.

8.2.6 Example: Parallel Cascade Identification

This section returns once more to the fly retina example, and it uses the parallel cascade method to construct a parallel Wiener cascade model of the system from the colored noise dataset. The overall model structure is the same as for the separable Volterra network example presented previously. However, in this case, the precise structure (the number of paths and polynomial orders) will not be set a priori. Rather, the parallel cascade method will determine it from the data. As before, the memory length of the dynamic elements was set to 50 ms based on low-order Volterra kernel estimates.

The IRF of the first pathway was determined from the first-order input–output cross-correlation using equation (5.10). There was a noticeable high-frequency component in the IRF, which was attributed to estimation noise since the input was not white. Consequently, the pseudo-inverse method, described in Section 5.2.3.4, was used to perform the deconvolution. Finally, a polynomial nonlinearity was fitted between the output of the first linear element and the system output. The order of the polynomial was chosen automatically, by minimizing the MDL, to be 6. The elements of the first pathway are shown in the top row of Figure 8.14.

Additional pathways were estimated from the principal generalized eigenvector of second-order input-residue cross-correlation, since this produces the greatest reduction in the residual variance, given the information in the second-order cross-correlation. As with the first pathway, the IRF was projected onto the significant singular vectors of the input autocorrelation matrix, to counter the ill-conditioning due to the nonwhite input spectrum.

The orders of the polynomial nonlinearities were chosen to minimize the MDL at each stage. Pathways were added to the model using this approach until adding a pathway resulted in an increase in the MDL. The pathway for which this occurred was not included in the model, and the parallel cascade expansion was halted.

The results of the identification are summarized in Figure 8.14. Note that the first pathway was obtained from the first-order input–output cross-correlation and that pathways 2–5 were derived from the second-order correlation. Table 8.1 presents the %VAF and MDL statistics for each path. Note that as the first four paths were added to the model, the MDL decreased, whereas it increased when the fifth path was added. Thus, after the first four paths were identified, no further significant pathways could be found from the first- and second-order cross-correlations.

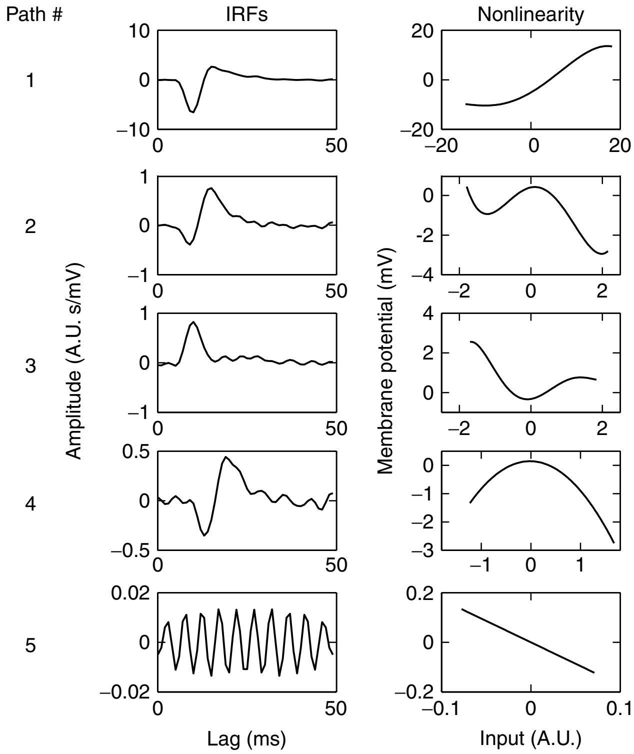

Figure 8.14 Elements of the first five Wiener pathways identified from the colored noise data by the eigenvector variant of the parallel cascade method. Path 1 was estimated from the first-order correlation while paths 2–5 were estimated from the second-order correlation.

TABLE 8.1 Results of the Parallel Cascade Identification[a]

Path Number	VAF (residue)	VAF (total)	MDL
1	92.81	92.81	1.510
2	15.74	93.94	1.309
3	13.37	94.75	1.166
4	3.91	94.96	1.147
5	0.12	94.96	1.148

[a] The fifth pathway added (bottom row) increased the MDL, and was therefore deemed to be insignificant.

Once the elements of the parallel cascade model were identified, its first- and second-order Volterra kernels were computed. Figure 8.15 shows how the first- and second-order kernels changed as the four significant pathways were added successively to the parallel cascade model. Figures 8.15A and 8.15B show the first- and second-order Volterra kernels of the first Wiener cascade. Figures 8.15C and 8.15D present the kernels due to the sum

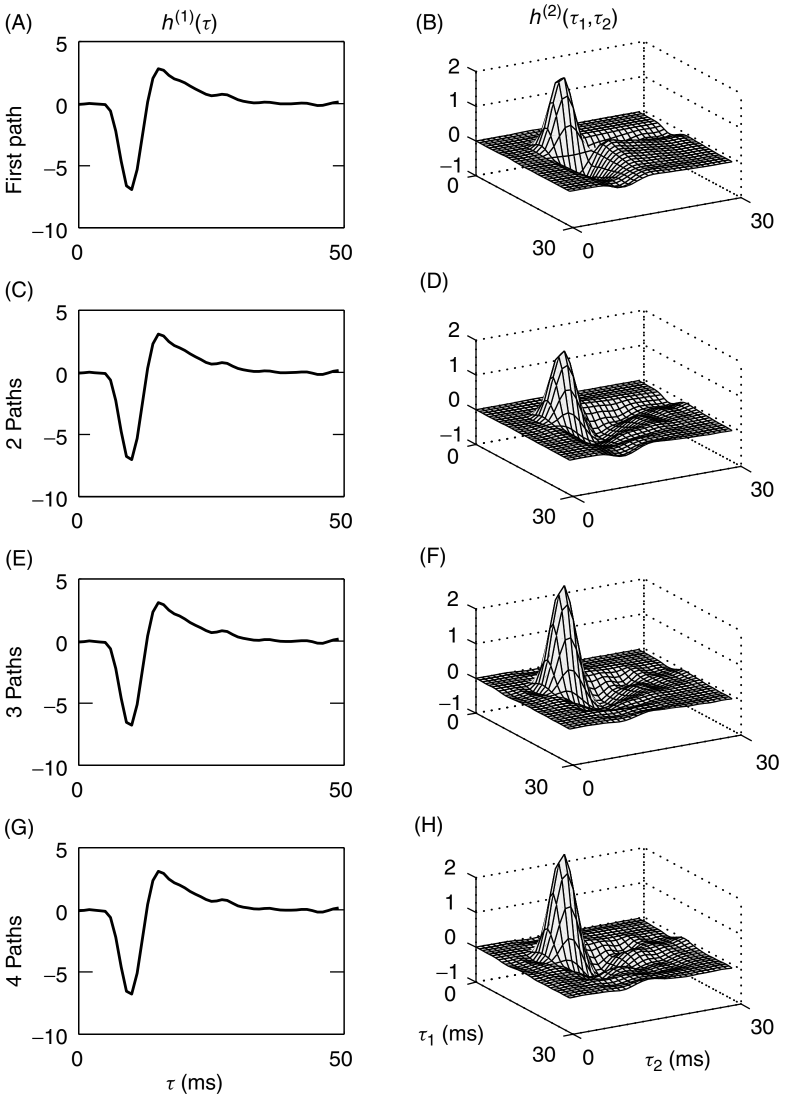

Figure 8.15 Evolution of the first- and second-order Volterra kernels of the parallel cascade model of the simulated fly retina, as pathways are added to the model. (A, B) Model consisting of a single Wiener cascade path. (C, D) Model comprising first two identified paths. (E, F) Model with three paths. (G, H) Model with four paths. Note the changes in the second-order kernels, B, D, F, and H.

of the first two pathways. The third and fourth rows continue this pattern. Notice that, as expected, there is no noticeable change in the first-order Volterra kernel from the first to the fourth model (see Figures 8.15A, 8.15C, 8.15E, and 8.15G), whereas the second-order kernel evolves as paths are added to the model, as shown in Figures 8.15B, 8.15D, 8.15F, and 8.15H. This contrasts with the PDM analysis, shown in Figure 7.18, where the first- and second-order kernels both changed as paths were added to the model.

The accuracy of the parallel cascade and SVN models were comparable. In the cross-validation segment, the parallel cascade model predicted the validation segment slightly better than the SVN, accounting for 93.45% VAF compared to 93.38% for the SVN. Furthermore, the kernel estimates produced by the parallel cascade method were considerably smoother than those constructed from the separable Volterra network. This is likely due to the low-rank projection employed in the parallel cascade method, which eliminates much of the high-frequency estimation noise due to the nonwhite input spectrum. Finally, note that computing the parallel cascade model took 1/20 of the time required for the iterative optimization of the SVN.

8.3 APPLICATION: VISUAL PROCESSING IN THE LIGHT-ADAPTED FLY RETINA

Juusola et al. (1995) used several of the methods discussed in this chapter to construct models of the first synapse in the light-adapted fly retina. In these experiments, the test input was provided by a green, light-emitting diode and consisted of a constant background illumination to which a Gaussian white noise signal was added. Experiments were performed with eight different background levels, ranging logarithmically from 160 to 500,000 effective photons per photoreceptor cell per second.

The outputs were electrical signals recorded from two different cell types in the retina. Glass microelectrodes were inserted into the retina of the compound eye through a small lateral hole in the cornea, which was subsequently sealed with high-vacuum grease. All signals were filtered at 500 Hz and sampled at 1 kHz. Each recording consisted of 8192 points: The first 8000 points were used for system identification, whereas the remaining 192 points were used for model validation.

In the first set of experiments, outputs were measured from the photoreceptors. The parallel cascade method was used to fit zero- through second-order Volterra kernels to the data. For background levels 5–8 (16,000 to 500,000 photons per second), the first-order Volterra kernel accounted for most (from 93.0% to 97.3%) of the output variance in the validation segment. Adding the second-order kernel did not significantly improve the model accuracy. Indeed, in some cases the second-order kernel decreased the predictive power of the model, suggesting that the second-order kernel was primarily fitting noise. Thus, it was concluded that the photoreceptor could be represented by its first-order Volterra kernel, which corresponded to the linear impulse response since there were no other significant kernels.

Recordings were also made of the outputs of large monopolar cells (LMC) that were directly excited by the photoreceptor outputs. The parallel cascade method was used to estimate the zero- through second-order Volterra kernels between the light input and the LMC output. This system comprised the photoreceptor, the LMC, and the synapse between them.

The goal, however, was to characterize the LMC and synapse, without the photoreceptor dynamics. Unfortunately, it was not possible to place electrodes to record input and output of a single LMC directly; consequently an indirect approach was required.

It was noted that for a given background illumination level, there was little variation among the first-order Volterra kernels identified for various photoreceptors. Thus, the input to the LMC could be simulated by convolving the light input with the IRF of a typical photoreceptor. To estimate the LMC and synapse dynamics, the parallel cascade

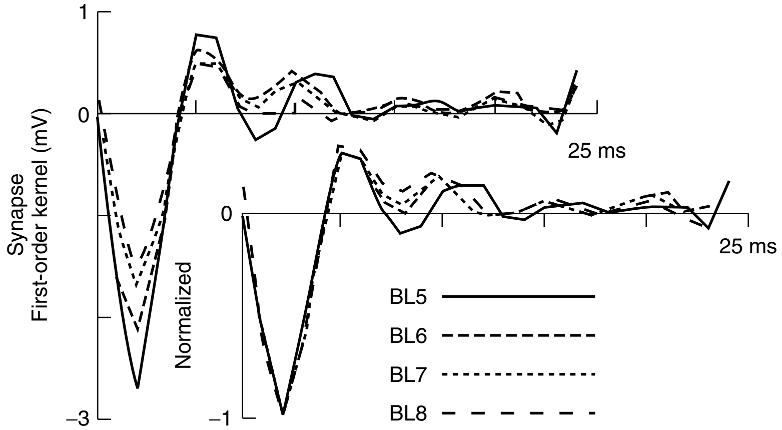

Figure 8.16 First-order kernels of the light adapted fly synapse, obtained at eight different background levels. Reprinted from Juusola et al. (1995). Used with permission of the American Physiological Society.

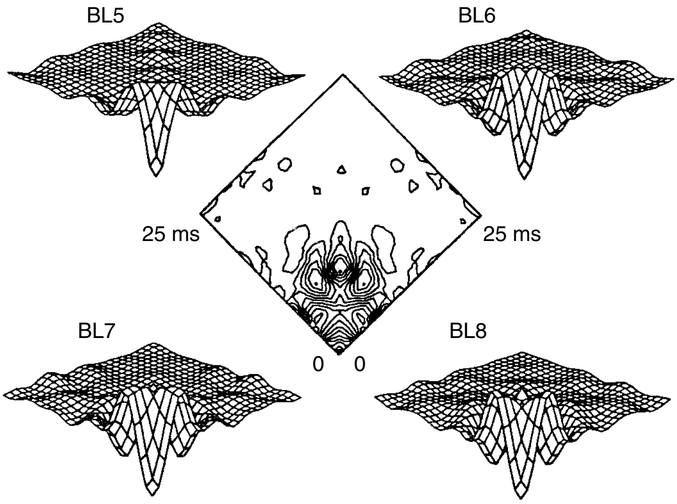

Figure 8.17 Second-order kernels of the light adapted fly synapse, obtained at eight different background levels. Reprinted from Juusola et al. (1995). Used with permission of the American Physiological Society.

method was used to fit the zero- through second-order Volterra kernels, with memory lengths of 25 ms, between the estimated input and measured output of the LMC. The first- and second-order kernel estimates, obtained at background levels 5–8, are shown in Figures 8.16 and 8.17, respectively. With these background levels, the first-order kernels accounted for 81.9% to 85.1% of the output variance in the validation segment. Adding the second-order kernels increased the prediction accuracy to between 91.5% and 95.0%. Thus, in contrast to the photoreceptors, the second-order kernels made a significant contribution to the LMC model.

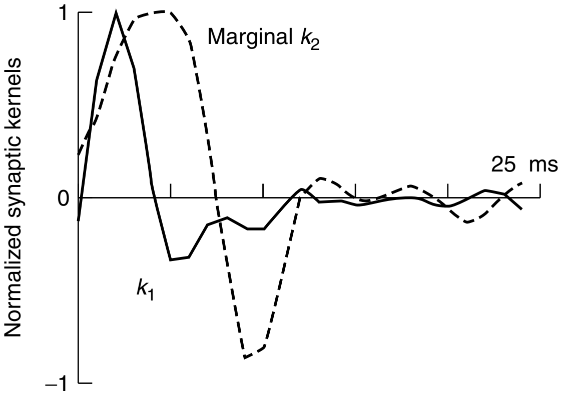

Figure 8.18 Testing the first- and second-order kernels for the LNL structure. Reprinted from Juusola et al. (1995). Used with permission of the American Physiological Society.

The LMC kernels were then tested to see if they could be generated by one of the simple block structures. Since the Wiener and Hammerstein cascades are both special cases of the LNL structure, the test described in equation (4.46) in Section 4.3.3 was applied first. Figure 8.18 shows the first-order Volterra kernel superimposed on the "marginal" second-order kernel—the sum of all nonzero slices of the second-order kernel, taken parallel to one of the axes. The two traces are clearly not proportional to each other, indicating that the first- and second-order Volterra kernels do not come from an LNL system. The Wiener and Hammerstein structures are special cases of the LNL cascade, so they too can be eliminated as potential structures for the first synapse in the fly retina.

There is no simple kernel-based test for the NLN structure. It is necessary to fit a NLN model to data and compare its predictive power to other, more general models (such as the Volterra series), or to the parallel cascade model. Juusola et al. attempted to fit an NLN model to their data. The two static nonlinearities were represented by fourth degree polynomials. The linear dynamic element was a 26-point FIR filter, giving the model a memory length of 25 ms, the same as that of the Volterra series model.

The Levenberg–Marquardt optimization algorithm, described in Section 8.1.3, was used to estimate the model parameters. Since this is an iterative algorithm, an initial estimate of the model was required. In this case, the two static nonlinearities were initialized as unity linear gains (i.e., all polynomial coefficients were zero, except for the first-order term, which was 1). The IRF of the linear dynamic element was initialized using the previously computed estimate of the first-order Volterra kernel. The predictive power of the resulting NLN models was comparable to that of the second-order Volterra series, between 92.3% and 94.1% VAF in the validation segment, even though the NLN models had far fewer parameters than the Volterra series. This suggested that the NLN cascade is an appropriate model structure for this system. Figure 8.19 shows the elements of the estimated NLN cascade.

After this study was completed, the photoreceptor IRF and LMC kernels were used in a model of the fly compound eye (Juusola and French, 1997). Simulations performed with this model were used to estimate the motion sensitivity of the fly's compound eye.

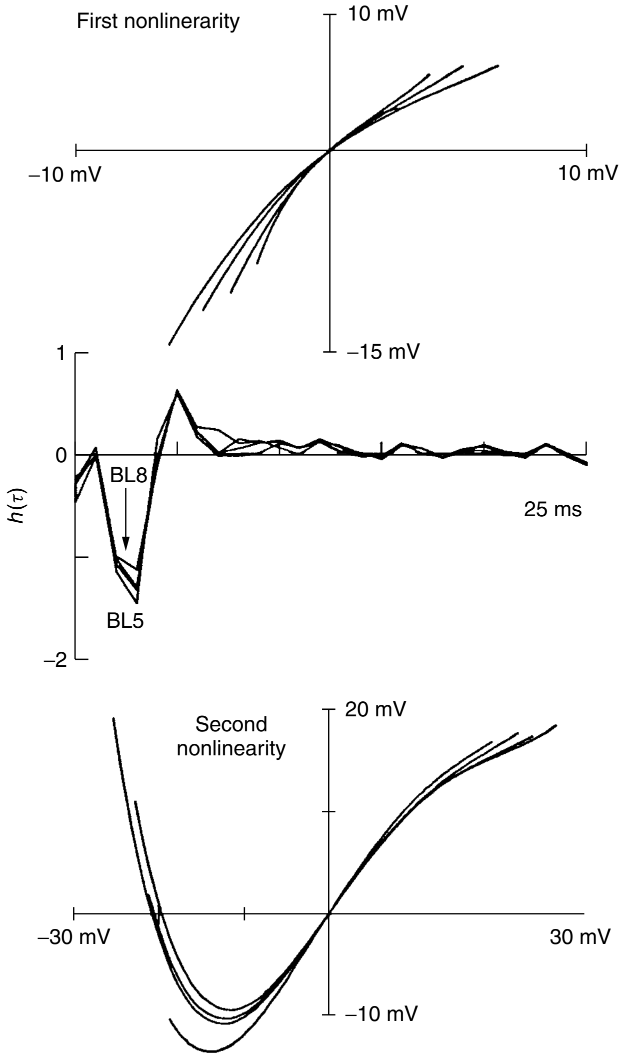

Figure 8.19 Elements of an NLN model of the first synapse in the light-adapted fly retina. Reprinted from Juusola et al. (1995). Used with permission of the American Physiological Society.

8.4 PROBLEMS

1. Compute the Jacobian for a Wiener system, in which the nonlinearity is expanded using Hermite polynomials.

2. Let $u(t)$ and $y(t)$ be the input and output of a nonlinear system. Let $h(t)$ be a linear system whose IRF is given by

$$\mathbf{h} = \mathbf{\Phi}_{\mathbf{uu}}^{-1}\boldsymbol{\phi}_{\mathbf{uy}}$$

and let its output be $x(t) = h(\tau) * u(t)$. Show that $u(t)$ is uncorrelated with $y(t) - x(t)$.

Next, suppose that a polynomial nonlinearity, $m(\cdot)$, is fit between $x(t)$ and $y(t)$, using a linear regression. Show that $u(t)$ and $y(t) - m(x(t))$ are uncorrelated.

3. Prove the second equality in equation (8.30).

4. Let $u(t)$ and $y(t)$ be the input and output of a nonlinear system, such that $\phi_{uy}(\tau) = 0$. Let $h(0, \tau)$ be a randomly chosen impulse response, and consider the following iteration for $k = 0, 1, \ldots$:

   - Compute $x(k, t)$, the output of the current IRF, $h(k, \tau)$.
   - Let $w(k, t)$ be the product of $x(k.t)$ and the output, $y(t)$.
   - Fit a linear IRF, $h(k + 1, \tau)$, between $u(t)$ and $w(k, t)$, normalize $h(k + 1, \tau)$, increment k, and repeat the iteration.

   Let $h(\tau)$ and λ be the principal generalized eigenvector and eigenvalue of the pencil $(\mathbf{\Phi_{uu}}, \boldsymbol{\phi}_{\mathbf{uuy}})$. Show that, provided that $h(0, \tau)$ is not orthogonal to $h(\tau)$, $h(k, \tau)$ will converge to $h(\tau)$.

5. Develop an iteration, similar to that in Problem **4**, which extracts IRFs from the third-order cross-correlation.

8.5 COMPUTER EXERCISES

1. The file `ch8/RunningEx.m` generates data similar to that used in the running example employed throughout this chapter. Use the FOA to estimate the first- and second-order Volterra kernels from this data. Now, generate a parallel Wiener cascade, using second-order polynomials, and compare the accuracy of the two identified models. Finally, identify a parallel Wiener cascade, but with fourth-order polynomial nonlinearities. Compare the predictive power of this model to the other two.

2. Implement the iterative scheme described in Problem **4**. Compare its performance to that of the explicit computation of the principal generalized eigenvector. Can you incorporate a pseudo-inverse into the algorithm, in order to improve the numerical conditioning?