4.7 Nonlinear Models

Fluid flows are nonlinear by nature, and one must address the data/model combination problem where the model is nonlinear. (There are also, as noted above, instances in which the data are nonlinear combinations of the state vector elements.) Nonetheless, the focus here on linear models is hardly wasted effort. As with more conventional systems, there are not many general methods for solving nonlinear estimation or control problems; rather, as with forward modeling, each situation has to be analyzed as a special case. Much insight is derived from a thorough understanding of the linear case, and indeed it is difficult to imagine tackling any nonlinear situation without a thorough grasp of the linear one. Not unexpectedly, the most accessible approaches to nonlinear estimation/control are based upon linearizations.

A complicating factor in the use of nonlinear models is that the objective functions need no longer have unique minima. There can be many nearby, or distant, minima, and the one chosen by the usual algorithms may depend upon exactly where one starts in the parameter space and how the search for the minimum is conducted. Indeed, the structure of the cost function may come to resemble a chaotic function, filled with hills, plateaus, and valleys into which one may stumble, never to get out again.[137] The combinatorial methods described in Chapter 3 are a partial solution.

4.7.1 The Linearized and Extended Kalman Filter

If one employs a nonlinear model,

{66001} $$\mathbf{x}(t) = \mathbf{L}\big(\mathbf{x}(t-1),\, \mathbf{B}\mathbf{q}(t-1),\, \mathbf{\Gamma}(t)\mathbf{u}(t-1)\big)\,, \tag{4.154}$$

then reference to the Kalman filter recursion shows that the forecast step can be taken as before,

{66002} $$\tilde{\mathbf{x}}(t,-) = \mathbf{L}\big(\tilde{\mathbf{x}}(t-1),\, \mathbf{B}\mathbf{q}(t-1),\, 0\big)\,, \tag{4.155}$$

but it is far from clear how to propagate the uncertainty from $\mathbf{P}(t-1)$ to $\mathbf{P}(t,-)$, the previous derivation being based upon the assumption that the error propagates linearly, independent of the true value of $\mathbf{x}(t)$ (or equivalently, that if the initial error is Gaussian, then so is the propagated error). With a nonlinear system one cannot simply add the propagated initial condition error to that arising from the unknown controls. A number of approaches exist to finding approximate solutions to this problem, but they can no longer be regarded as strictly optimal, representing different linearizations.

Suppose that we write,

$$\mathbf{x}(t) = \mathbf{x}_o(t) + \Delta\mathbf{x}(t)\,, \qquad \mathbf{q} = \mathbf{q}_0(t) + \Delta\mathbf{q}(t)\,, \tag{4.156}$$ {66003}

$$\begin{aligned}\mathbf{L}\big(\mathbf{x}(t),\, \mathbf{Bq}(t),\, \boldsymbol{\Gamma}\mathbf{u}(t),\, t\big) = &\\ &\mathbf{L}_o\big(\mathbf{x}_o(t),\, \mathbf{q}_o(t), 0, t\big) + \mathbf{L}_x\big(\mathbf{x}_o(t), 0\big)^T \Delta\mathbf{x}(t) \\ &+ \mathbf{L}_q\big(\mathbf{x}_o(t),\, \mathbf{q}_o(t)\big)^T \Delta\mathbf{q}(t) + \mathbf{L}_u\big(\mathbf{x}_o(t),\, \mathbf{q}_o(t)\big)^T \mathbf{u}(t)\end{aligned} \tag{4.157}$$ {66004}

where

$$\begin{aligned}&\mathbf{L}_x\big(\mathbf{x}_o(t),\, \mathbf{q}_o(t)\big) = \frac{\partial \mathbf{L}}{\partial \mathbf{x}(t)}\,, \quad \mathbf{L}_q\big(\mathbf{x}_o(t),\, \mathbf{q}_o(t)\big) = \frac{\partial \mathbf{L}}{\partial \mathbf{q}(t)}\,, \\ &\mathbf{L}_u\big(\mathbf{x}_o(t),\, \mathbf{q}_o(t)\big) = \frac{\partial \mathbf{L}}{\partial \mathbf{u}(t)}\,.\end{aligned}$$

Then,

$$\mathbf{x}_o(t) = \mathbf{L}_o\big(\mathbf{x}_o(t-1),\, \mathbf{Bq}_o(t-1),\, 0,\, t-1\big), \tag{4.158}$$ {66006}

defines a nominal solution, or trajectory, $\mathbf{x}_o(t)$. The model is assumed to be differentiable in this manner; all discrete models are so differentiable, numerically, barring a division by zero somewhere. Note that all discrete models are by definition discontinuous, and discrete differentiation automatically accomodates such discontinuities. Numerical models often have switches, typically given by "if xx, then yy" statements. Even these models are differentiable in the sense we need, except at the isolated point where the "if" statement is evaluated; typically the code representing the derivatives will also have a switch at this point.

A non-linear models, in particular, can have trajectories which bifurcate in a number of different ways, so that subject to slight differences in state, the trajectory can take widely differing pathways as time increases. This sensitivity can be a very serious problem for a Kalman filter forecast, because a linearization may take the incorrect branch, leading to divergences well-beyond any formal error estimate. Note, however, that the problem is much less serious in a smoothing problem, as one then has observations available indicating the branch actually taken.

Assuming a nominal solution is available, we have an equation for the solution perturbation:

$$\Delta\mathbf{x}(t) = \mathbf{L}_x\big(\mathbf{x}_o(t-1),\, \mathbf{q}_o(t-1)\big)^T \Delta\mathbf{x}(t-1) + \mathbf{L}_q^T \Delta\mathbf{q}(t-1) + \mathbf{L}_u^T \mathbf{u}(t-1)\,, \tag{4.159}$$ {66007}

which is linear—called the "tangent linear model," and of the form already used for the Kalman filter, but with redefinitions of the governing matrices. The full solution would be the sum of the nominal solution, $\mathbf{x}_o(t)$, and the perturbation $\Delta\mathbf{x}(t)$. This form of estimate is sometimes known as the "linearized Kalman filter," or the "neighboring optimal estimator." Its usage depends upon the existence of a nominal solution, differentiability of the model, and the presumption that the controls $\Delta\mathbf{q}$, $\mathbf{u}$ do not drive the system too far from the nominal trajectory.

The so-called "extended Kalman filter" is nearly identical, except that the linearization is taken instead about the most recent estimate $\tilde{\mathbf{x}}(t)$; that is, the partial derivatives in (4.157) are evaluated using not $\mathbf{x}_o(t-1)$, but $\tilde{\mathbf{x}}(t-1)$. This latter form is more prone to instabilities, but if the system drifts very far from the nominal trajectory, it could well be more accurate than the linearized filter. Linearized smoothing algorithms can be developed in analogous ways, and as already noted, the inability to track strong model nonlinearities is much less serious with a smoother than with a filter. The references go into these questions in great detail. Problems owing to multiple minima in the cost function[138] can always be overcome by having enough observations to keep the estimates close to the true state. The usual posterior checks of model and data residuals are also a very powerful precaution against a model failing to track the true state adequately.

It is possible to define physical systems for which sufficiently accurate or useful derivatives of the system cannot be defined[139] so that neither Lagrange multiplier nor linerearized sequential methods can be used. Whether such systems occur in practice, or whether they are somewhat like the mathematical pathologies used by mathematicians to demonstrate the limits of conventional mathematical tools (e.g., the failure to exist of the derivative of $\sin(1/t)$, $t \to 0$, or of the existence of space-filling curves) is not so clear. It is clear that all linearization approaches do have limits of utility, but they are and will likely remain, the first choice of practitioners necessarily aware that no universal solution methods exist.

4.7.2 Parameter Estimation and Adaptive Estimation

Often models contain parameters whose values are poorly known. In fluid flow problems, these often concern parameterized turbulent mixing, with empirical parameters which the user is willing to adjust to provide the best fit of the model to the observations. Sometimes, this approach is the only way to determine the parameter values.

Suppose that the model is linear in $\mathbf{x}(t)$ and that it contains a vector of parameters, $\mathbf{p}$, whose nominal values, $\mathbf{p}_0$, we wish to improve, while also estimating the state vector. Write the model as

{66008} $$\mathbf{x}(t) = \mathbf{A}\big(\mathbf{p}(t-1)\big)\mathbf{x}(t-1) + \mathbf{B}\mathbf{q}(t-1) + \mathbf{\Gamma}\mathbf{u}(t-1), \tag{4.160}$$

where the time dependence in $\mathbf{p}(t)$ to the changing estimate of their value rather than a true physical time dependence. A general approach to solving this problem is to augment the state vector. That is,

{66009}
$$\mathbf{x}_A(t) = \begin{bmatrix} \mathbf{x}(t) \\ \mathbf{p}(t) \end{bmatrix} . \tag{4.161}$$

Then write a model for this augmented state as,

$$\mathbf{x}_A(t) = \mathbf{L}_A \left[\mathbf{x}_A(t-1), \, \mathbf{q}(t-1), \, \mathbf{u}(t-1) \right] , \tag{4.162}$$
{66010}

where

$$\mathbf{L}_A = \left\{ \begin{matrix} \mathbf{A}\big(\mathbf{p}(t-1)\big) & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{matrix} \right\} \mathbf{x}_A\,(t-1) + \mathbf{B}\mathbf{q}(t-1) + \boldsymbol{\Gamma}\mathbf{u}(t-1) \, . \tag{4.163}$$
{66011}

The observation equation is augmented simply as

$$\begin{gathered} \mathbf{y}_A(t) = \mathbf{E}_A(t)\mathbf{x}_A(t) + \mathbf{n}_A(t) \, , \\ \mathbf{E}_A(t) = \{ \mathbf{E}(t) \quad \mathbf{0} \} \ , \quad \mathbf{n}_A(t) = \mathbf{n}(t) \, , \end{gathered}$$

assuming that there are no direct measurements of the parameters. The evolution equation for the parameters can be made more complex than indicated here. A solution can be found by using the linearized Kalman filter, for example, linearizing about the nominal parameter values. Parameter estimation is a very large subject.[140]

A major point of concern in estimation procedures based upon Gauss-Markov type methods lies in specification of the various covariance matrices, especially those describing the model error—here lumped into $\mathbf{Q}(t)$. The reader will probably have concluded that there is, however, nothing precluding deduction of the covariance matrices from the model and observations, given that adequate numbers of observations are available. For example, it is straightforward to show that if a Kalman filter is operating properly, then the so-called innovation, $\mathbf{y}(t) - \mathbf{E}\tilde{\mathbf{x}}(t,-)$, should be uncorrelated with all previous measurements:

$$\langle \mathbf{y}(t') \big(\mathbf{y}(t) - \mathbf{E}\tilde{\mathbf{x}}(t,-) \big) \rangle = 0 \, , \quad t' < t, \tag{4.164}$$
{66013}

(recall Eq. (2.431)). To the extent that (5.29) is not satisfied, the covariances need to be modified, and algorithms can be formulated for driving the system toward this condition. The possibilities for such procedures have an extended literature under the title "adaptive estimation."[141]

4.7.3 Nonlinear Adjoint Equations; Searching for Solutions

Consider now a nonlinear model in the context of the Lagrange multipliers approach. Let the model be nonlinear in either the state vector, or the model parameters, or both, so that a typical objective function is,

$$\begin{aligned}J &= [\mathbf{x}(0)-\mathbf{x}_0]^T\,\mathbf{P}(0)^{-1}\,[\mathbf{x}(0)-\tilde{\mathbf{x}}_0]\\&+\sum_{t=1}^{t_f}[\mathbf{E}(t)\mathbf{x}(t)-\mathbf{y}(t)]^T\,\mathbf{R}(t)^{-1}\,[\mathbf{E}(t)\mathbf{x}(t)-\mathbf{y}(t)]\\&+\sum_{t=0}^{t_f-1}\mathbf{u}(t)^T\mathbf{Q}(t)^{-1}\mathbf{u}(t)\\&-2\sum_{t=1}^{t_f}\boldsymbol{\mu}(t)^T\,[\mathbf{x}(t)-\mathbf{L}\,[\mathbf{x}(t-1),\,\mathbf{Bq}(t-1),\,\mathbf{\Gamma u}(t-1)]]\;.\end{aligned} \tag{4.165}$$

The observations continue to be treated as linear in the state vector, but even this assumption can be relaxed. The normal equations are:

{66015} $$\frac{1}{2}\frac{\partial J}{\partial\mathbf{u}(t)}=\mathbf{Q}\,(t)^{-1}\,\mathbf{u}(t)+\left(\frac{\partial\mathbf{L}\,(\mathbf{x}(t),\,\mathbf{Bq}(t),\,\mathbf{\Gamma u}(t))}{\partial\mathbf{u}\,(t)}\right)^T\,\mathbf{\Gamma}^T\boldsymbol{\mu}(t+1)=\mathbf{0}\,, \tag{4.166}$$

$$0\le t\le t_f-1$$

{66016} $$\frac{1}{2}\frac{\partial J}{\partial\boldsymbol{\mu}(t)}=\mathbf{x}(t)-\mathbf{L}\,[\mathbf{x}(t-1),\,\mathbf{Bq}(t-1),\,\mathbf{\Gamma u}(t-1)]=\mathbf{0}\,,\quad 1\le t\le t_f \tag{4.167}$$

{66017} $$\frac{1}{2}\frac{\partial J}{\partial\mathbf{x}(0)}=\mathbf{P}(0)^{-1}\,[\mathbf{x}(0)-\mathbf{x}_0]+\left(\frac{\partial\mathbf{L}\,(\mathbf{x}(0),\,\mathbf{Bq}(0),\,\mathbf{\Gamma u}(0))}{\partial\mathbf{x}(0)}\right)^T\,\boldsymbol{\mu}(1)=\mathbf{0}\,, \tag{4.168}$$

{66018} $$\begin{aligned}\frac{1}{2}\frac{\partial J}{\partial\mathbf{x}(t)}&=\mathbf{E}(t)^T\mathbf{R}(t)^{-1}\,[\mathbf{E}(t)\mathbf{x}(t)-\mathbf{y}(t)]-\boldsymbol{\mu}(t)\\&+\left(\frac{\partial\mathbf{L}\,(\mathbf{x}(t),\,\mathbf{Bq}(t),\,\mathbf{\Gamma u}(t))}{\partial\mathbf{x}(t)}\right)^T\,\boldsymbol{\mu}(t+1)=\mathbf{0}\,,\quad 1\le t\le t_f-1\end{aligned} \tag{4.169}$$

{66019} $$\frac{1}{2}\frac{\partial J}{\partial\mathbf{x}(t_f)}=\mathbf{E}(t_f)^T\mathbf{R}(t_f)^{-1}\,[\mathbf{E}(t_f)\mathbf{x}(t_f)-\mathbf{y}(t_f)]-\boldsymbol{\mu}(t_f)=\mathbf{0}\,. \tag{4.170}$$

These are nonlinear because of the nonlinear model (4.167)—although the adjoint model (4.169) remains linear in $\boldsymbol{\mu}(t)$—and the linear methods used thus far will not work directly. The operators,

{adjoint1} $$\left(\frac{\partial\mathbf{L}\,(\mathbf{x}\,(t)\,,\mathbf{Bq}\,(t)\,,\mathbf{\Gamma u}\,(t)\,,t)}{\partial\mathbf{u}\,(t)}\right)\,,\,\left(\frac{\partial\mathbf{L}\,(\mathbf{x}\,(t)\,,\mathbf{Bq}\,(t)\,,\mathbf{\Gamma u}\,(t)\,,t)}{\partial\mathbf{x}(t)}\right), \tag{4.171}$$

appearing in the above equations are, as in Eq. (4.157), the derivatives of the model with respect to the control and statevectors. Assuming they exist, they represent a linearization of the model about the state and control vectors and again are the tangent linaer model. Their transposes are, in this context, the adjoint model. There is some ambiguity about the terminology: the form of (4.171) or the transposes are definable independent of the form of J. Otherwise, Eq. (4.169) and its boundary condition (4.170) depend upon the actual observations and the details of J; one might call this pair the "adjoint evolution" equation to distinguish it from the adjoint model.

If the nonlinearity is not too large, perturbation methods may work. This notion leads to what is usually called "neighboring optimal control"[142] Where the nonlinearity is large, the approach to solution is an iterative one. Consider what one is trying to do. At the optimum, if we can find it, J will reach a stationary value in which the terms multiplying the $\boldsymbol{\mu}(t)$ will vanish. Essentially, one uses *search* methods that are able to find a solution (there may well be multiple such solutions, each corresponding to a local minimum of J).

There are many known ways to seek approximate solutions to a set of simultaneous equations, linear or nonlinear, using various search procedures. Most such methods are based upon what are usually called "Newton" or "quasi-Newton" methods, or variations on steepest descent. The most popular approach to tackling the set (4.166)–(4.170) has been a form of conjugate gradient or modified steepest descent algorithm. The iteration cycles are commonly carried out by making a first estimate of the initial conditions and the boundary conditions–for example, setting $\mathbf{u} = \mathbf{0}$. One integrates (4.167) forward in time to produce a first guess for $\mathbf{x}(t)$. A first guess set of Lagrange multipliers is obtained by integrating (4.169) backward in time. Normally, (4.166) is not then satisfied, but because the values obtained provide information on the gradient of the objective function with respect to the controls, one knows the sign of the changes to make in the controls to reduce J. Perturbing the original guess for $\mathbf{u}(t)$ in this manner, one does another forward integration of the model and backward integration of the adjoint. Because the Lagrange multipliers provide the partial derivatives of J with respect to the solution, (Eq. (4.168) permits calculation of the direction in which to shift the current estimate of $\mathbf{x}(0)$ to decrease J), one can employ a conjugate gradient or steepest descent method to modify $\tilde{\mathbf{x}}(0)$ and carry out another iteration.

In this type of approximate solution, the adjoint solution, $\tilde{\boldsymbol{\mu}}(t)$, is really playing two distinct roles. On the one hand, it is a mathematical device to impose the model constraints; on the other, it is being used as a numerical convenience for determining the direction and step size to best reduce the objective function. The two roles are obviously intimately related, but as we have seen for the linear models, the first role is the primary one. The problem of possibly falling into the wrong minimum of the objective function remains here, too.

In practice, $\mathbf{L}(\mathbf{x}(t-1), \mathbf{Bq}(t-1), \boldsymbol{\Gamma}\mathbf{u}(t-1), t-1)$ is represented as many lines of computer code. Generating the derivatives in Eq. (4.171) can be a major undertaking. Fortunately, and remarkably, the automatic differentiation (AD) tools mentioned above can convert the code for $\mathbf{L}[\mathbf{x}(t), \mathbf{Bq}(t), \boldsymbol{\Gamma}\mathbf{u}(t), t]$ into the appropriate code for the derivatives. While still requiring a degree of manual intervention, this AD software renders Lagrange multiplier methods a practical approach for model codes running to many thousands of lines.[143] The basic ideas are {pagead2} sketched in the next subsection and the Chapter Appendix.

4.7.4 Automatic Differentiation, Linearization, and Sensitivity

The linearized and extended filters and smoothers (e.g., Eq. 4.158) and the normal equations (4.166-4.170) involve derivatives such as $\partial \mathbf{L}/\partial \mathbf{x}(t)$. One might wonder how these are to be obtained. Several procedures exist, but to motivate what is perhaps the most elegant method, let us begin with a simple example of a two-dimensional non-linear model.

Example.

Let

{nonlin1}
$$\mathbf{x}(t)=\mathbf{L}(\mathbf{x}(t-1))=\begin{bmatrix} a\mathbf{x}^T(t-1)\,\mathbf{x}(t-1)+c \\ b\mathbf{x}^T(t-1)\,\mathbf{x}(t-1)+d \end{bmatrix} \tag{4.172}$$

where a,b,c,d are fixed constants. Time stepping from $t=0$,

$$\mathbf{x}(1) = \begin{bmatrix} a\mathbf{x}^T(0)\,\mathbf{x}(0)+c \\ b\mathbf{x}^T(0)\,\mathbf{x}(0)+d \end{bmatrix}, \tag{4.173}$$

$$\mathbf{x}(2) = \begin{bmatrix} a\mathbf{x}^T(1)\,\mathbf{x}(1)+c \\ b\mathbf{x}^T(1)\,\mathbf{x}(1)+d \end{bmatrix}$$
$$\cdots$$

Consider the dependence, $\partial\mathbf{x}(t)/\partial\mathbf{x}(0)$,

{nonlin4}
$$\frac{\partial\mathbf{x}(t)}{\partial\mathbf{x}(0)}=\begin{Bmatrix} \frac{\partial x_1(t)}{\partial x_1(0)} & \frac{\partial x_2(t)}{\partial x_1(0)} \\ \frac{\partial x_1(t)}{\partial x_2(0)} & \frac{\partial x_2(t)}{\partial x_2(0)} \end{Bmatrix} \tag{4.174}$$

For $t=2$, by the definitions and rules of Chapter 2, we have,

{chain4}
$$\frac{\partial\mathbf{x}(2)}{\partial\mathbf{x}(0)}=\begin{Bmatrix} \frac{\partial x_1(2)}{\partial x_1(0)} & \frac{\partial x_2(2)}{\partial x_1(0)} \\ \frac{\partial x_1(2)}{\partial x_2(0)} & \frac{\partial x_2(2)}{\partial x_2(0)} \end{Bmatrix}=\frac{\partial\mathbf{L}(\mathbf{L}(\mathbf{x}(0)))}{\partial\mathbf{x}(0)}=\mathbf{L}'(\mathbf{x}(0))\,\mathbf{L}'(\mathbf{L}(\mathbf{x}(0))). \tag{4.175}$$

Noting,

{nonlin3}
$$\frac{\partial\mathbf{x}(2)}{\partial\mathbf{x}(0)}=\frac{\partial\mathbf{L}(\mathbf{x}(1))}{\partial\mathbf{x}(0)}=\frac{\partial\mathbf{x}(1)}{\partial\mathbf{x}(0)}\frac{\partial\mathbf{L}(\mathbf{x}(1))}{\partial\mathbf{x}(1)}=\frac{\partial\mathbf{L}(\mathbf{x}(0))}{\partial\mathbf{x}(0)}\frac{\partial\mathbf{L}(\mathbf{x}(1))}{\partial\mathbf{x}(1)} \tag{4.176}$$

where we have used the "chainrule" for differentiation. Substituting into (4.176) we have,

$$\begin{aligned}\frac{\partial \mathbf{x}(2)}{\partial \mathbf{x}(0)} &= \begin{Bmatrix} 2ax_1(0) & 2bx_1(0) \\ 2ax_2(0) & 2bx_2(0) \end{Bmatrix} \begin{Bmatrix} 2ax_1(1) & 2bx_1(1) \\ 2ax_2(1) & 2bx_2(1) \end{Bmatrix} \\ &= \begin{Bmatrix} 2ax_1(0) & 2bx_1(0) \\ 2ax_2(0) & 2bx_2(0) \end{Bmatrix} \begin{Bmatrix} 2a^2 & 2ab \\ 2ab & 2b^2 \end{Bmatrix} \mathbf{x}^T(0)\,\mathbf{x}(0),\end{aligned}$$

*which it may be confirmed is identical to Eq. (**??**). By direct calculation from Eq. (4.173), we have*

$$\frac{\partial \mathbf{x}(2)}{\partial \mathbf{x}(0)} = 4\left(a^2+b^2\right)\mathbf{x}(0)^T\mathbf{x}(0)\begin{Bmatrix} ax_1(0) & bx_1(0) \\ ax_2(0) & bx_2(0) \end{Bmatrix}. \tag{4.177}$$ {chain3}

Now let us use the chain rule on,

$$\mathbf{x}(2) = \mathbf{L}(\mathbf{L}(\mathbf{x}(0))).$$

$$\begin{aligned}\frac{\partial \mathbf{x}(2)}{\partial \mathbf{x}(0)} &= \mathbf{L}'(\mathbf{x}(0))\,\mathbf{L}'(\mathbf{L}(\mathbf{x}(0))), \qquad (4.178)\\ \mathbf{L}'(\mathbf{x}(0)) &= \begin{Bmatrix} 2ax_1(0) & 2bx_1(0) \\ 2ax_2(0) & 2bx_2(0) \end{Bmatrix}.\end{aligned}$$

$$\mathbf{L}'(\mathbf{L}(\mathbf{x}(0))) = \mathbf{L}'(\mathbf{x}(1)) = \begin{Bmatrix} 2ax_1(1) & 2bx_1(1) \\ 2ax_2(1) & 2bx_2(1) \end{Bmatrix} = \begin{Bmatrix} 2a^2\mathbf{x}(0)^T\mathbf{x}(0) & 2ab\mathbf{x}(0)^T\mathbf{x}(0) \\ 2ab\mathbf{x}(0)^T\mathbf{x}(0) & 2b^2\mathbf{x}(0)^T\mathbf{x}(0) \end{Bmatrix}.$$

Multiplying, as in (4.178),

$$\begin{Bmatrix} 2ax_1(0) & 2bx_1(0) \\ 2ax_2(0) & 2bx_2(0) \end{Bmatrix} \begin{Bmatrix} 2a^2\mathbf{x}(0)^T\mathbf{x}(0) & 2ab\mathbf{x}(0)^T\mathbf{x}(0) \\ 2ab\mathbf{x}(0)^T\mathbf{x}(0) & 2b^2\mathbf{x}(0)^T\mathbf{x}(0) \end{Bmatrix} =$$

$$4\left(a^2+b^2\right)\mathbf{x}(0)^T\mathbf{x}(0)\begin{Bmatrix} ax_1(0) & bx_1(0) \\ ax_2(0) & bx_2(0) \end{Bmatrix}$$

consistent with (4.177). Hence, as required,

$$d\mathbf{x}(2) = d\mathbf{x}(0)^T\begin{Bmatrix} 2ax_1(0) & 2bx_1(0) \\ 2ax_2(0) & 2bx_2(0) \end{Bmatrix} \begin{Bmatrix} 2a^2\mathbf{x}(0)^T\mathbf{x}(0) & 2ab\mathbf{x}(0)^T\mathbf{x}(0) \\ 2ab\mathbf{x}(0)^T\mathbf{x}(0) & 2b^2\mathbf{x}(0)^T\mathbf{x}(0) \end{Bmatrix}.$$

As a computational point note that this last equation involves a matrix-matrix multiplication on the right. But if written as a transpose,

$$d\mathbf{x}(2)^T = \left\{ \begin{matrix} 2a^2\mathbf{x}(0)^T\mathbf{x}(0) & 2ab\mathbf{x}(0)^T\mathbf{x}(0) \\ 2ab\mathbf{x}(0)^T\mathbf{x}(0) & 2b^2\mathbf{x}(0)^T\mathbf{x}(0) \end{matrix} \right\}^T \left\{ \begin{matrix} 2ax_1(0) & 2bx_1(0) \\ 2ax_2(0) & 2bx_2(0) \end{matrix} \right\}^T d\mathbf{x}(0).$$

$d\mathbf{x}(2)^T$ *can be found from matrix-vector multiplications alone, and which for large matrices is a vastly reduced computation. This reduction in computational load lies behind the use of so-called reverse mode methods described below.*

Going much beyond these simple statements takes us too far into the technical details. [144]

The chain rule can be extended, such that,

{chainrule1}
$$\begin{aligned}\frac{\partial\mathbf{x}(t)}{\partial\mathbf{x}(0)} &= \frac{\partial\mathbf{L}(\mathbf{x}(t-1))}{\partial\mathbf{x}(0)} = \frac{\partial\mathbf{x}(t-1)}{\partial\mathbf{x}(0)}\frac{\partial\mathbf{L}(\mathbf{x}(t))}{\partial\mathbf{x}(t-1)} = \frac{\partial\mathbf{L}(\mathbf{x}(t-2))}{\partial\mathbf{x}(0)}\frac{\partial\mathbf{L}(\mathbf{x}(t-1))}{\partial\mathbf{x}(t-1)} \qquad (4.179)\\ &= \frac{\partial\mathbf{L}(\mathbf{x}(t-3))}{\partial\mathbf{x}(0)}\frac{\partial\mathbf{L}(\mathbf{x}(t-2))}{\partial\mathbf{x}(t-2)}\frac{\partial\mathbf{L}(\mathbf{x}(t-1))}{\partial\mathbf{x}(t-1)} = \cdots \\ &= \frac{\partial\mathbf{L}(\mathbf{x}(0))}{\partial\mathbf{x}(0)}\cdots\frac{\partial\mathbf{L}(\mathbf{x}(t-2))}{\partial\mathbf{x}(t-2)}\frac{\partial\mathbf{L}(\mathbf{x}(t-1))}{\partial\mathbf{x}(t-1)}\end{aligned}$$

Although this result is formally correct, such a calculation could be quite cumbersome to code and carryout for a more complicated model (examples of such codes do exist). An alternative, of course, is to systematically and separately perturb each of the elements of $\mathbf{x}(0)$, and integrate the model forward from $t=0$ to t, thus numerically evaluating $\partial\mathbf{x}(t)/\partial x_i(0)$, $1\leq i\leq N$. The model would thus have to be run N times, and there might be issues of numerical accuracy. (The approach is similar to the determination of numerical Green functions considered above.)

Practical difficulties such as these have given rise to the idea of "automatic differentiation" in which one accepts from the beginning that a computer code will be used to define $\mathbf{L}(\mathbf{x}(t), t, \mathbf{q}(t))$ (reintroducing the more general definition of $\mathbf{L}$).[145] One then seeks automatic generation of a second code, capable of evaluating the elements in Eq. (4.179), that is terms of the form $\partial\mathbf{L}(\mathbf{x}(n))/\partial\mathbf{x}(n)$, for any n. Automatic differentiation (AD) tools (sometimes called "compilers" take the computer codes (typically in Fortran, or C or Matlab) and generate new codes for the *exact* partial derivatives. Various packages go under names like ADIFOR, TAF, ADiMAT, etc. The possibility of using such procedures has already been alluded to, where it was noted that for a linear model, the first derivative would be the state transition matrix $\mathbf{A}$, which may not otherwise be explicitly available. That is,

$$\mathbf{A}(t) = \frac{\partial\mathbf{L}(\mathbf{x}(t), t, \mathbf{q}(t))}{\partial\mathbf{x}(t)}.$$

Actual implementation of AD involves one deeply in the structures of computer coding languages, and is not within the scope of this book. Note that the existing implementations are not restricted to such simple models as we used in the particular example, but deal with the more general $\mathbf{L}(\mathbf{x}(t), t, \mathbf{q}(t))$.

In many cases, one cares primarily about some scalar quantity, $H(\tilde{\mathbf{x}}(t_f))$, e.g. the heat flux or pressure field in a flow, as given by the statevector at the end time, $\mathbf{x}(t_f)$, of a model computation. Suppose[146] one seeks the sensitivity of that quantity to perturbations in the initial conditions (any other control variable could be considered), $\mathbf{x}(0)$. Let $\mathbf{L}$ continue to be the operator defining the time-stepping of the model. Define $\Psi_t = \mathbf{L}(\mathbf{x}(t), t, \mathbf{q}(t))$. Then,

$$H = H\left(\Psi_{t_f}\left[\Psi_{t_f-1}\left[...\Psi_1\left[\mathbf{x}(0)\right]\right]\right]\right),$$

that is, the function of the final state of interest is a nested set of operators working on the control vector $\mathbf{x}(0)$. Then the derivative of H with respect to $\mathbf{x}(0)$ is again obtained from the chain rule,

$$\frac{\partial H}{\partial \mathbf{x}(0)} = H'\left(\Psi'_{t_f}\left[\Psi'_{t_f-1}\left[...\Psi'_1\left[\mathbf{x}(0)\right]\right]\right]\right), \tag{4.180}$$ {operator1}

where the prime denotes the derivative with respect to the argument of the operator $\mathbf{L}$ evaluated at that time,

$$\frac{\partial \Psi_t(\mathbf{p})}{\partial \mathbf{p}}.$$

Notice that these derivatives, are the Jacobians (matrices) of dimension $N \times N$ at each time-step, and are the same derivatives that appear in the operators in (4.171). The nested operator (4.180) can be written as a matrix product,

$$\frac{\partial H}{\partial \mathbf{x}(0)} = \nabla h^T \frac{\partial \Psi_1(\mathbf{p})}{\partial \mathbf{p}} \frac{\partial \Psi_2(\mathbf{p})}{\partial \mathbf{p}} ... \frac{\partial \Psi_{t_f}(\mathbf{p})}{\partial \mathbf{p}}. \tag{4.181}$$ {operator2}

∇h is the vector of derivatives of function H (the gradient) and so (4.181) is a column vector of dimension $N \times 1$. $\mathbf{p}$ represents the statevector at the prior timestep for each Ψ_t The adjoint compilers described above compute $\partial H / \partial \mathbf{x}(0)$ in what is called the "forward mode", producing an operator which runs from right to left, multiplying t_f-$N \times N$ matrices starting with $\partial \Psi_1(\mathbf{p}) / \partial \mathbf{p}$.

If however, Eq. (4.181) is transposed,

$$\left(\frac{\partial H}{\partial \mathbf{x}(0)}\right)^T = \left(\frac{\partial \Psi_{t_f}(\mathbf{p})}{\partial \mathbf{p}}\right)^T \left(\frac{\partial \Psi_{t_f-1}(\mathbf{p})}{\partial \mathbf{p}}\right)^T ... \left(\frac{\partial \Psi_1(\mathbf{p})}{\partial \mathbf{p}}\right)^T \nabla h, \tag{4.182}$$ {operator3}

where the first multiplication on the right involves multiplying the column vector ∇h by an $N \times N$ matrix, thus producing another $N \times 1$ vector. More generally, the set of products in (4.182), again taken from right to left, involves only multiplying a vector by a matrix, rather than a matrix by a matrix as in (4.181), with a potentially very large computational saving. Such evaluation is the *reverse* or *adjoint mode* calculation (the transposes generate the required adjoint operators. although a formal transpose is not actually formed) and have become available in the automatic differentiation tools only comparatively recently. In comparing the computation in the forward and reverse modes, one must be aware that there is a storage penalty in (4.182) not incurred in (4.181).[147] In practice, the various operators $\partial \Psi_i (\mathbf{p}) / \partial \mathbf{p}$ are not obtained explicitly, but are computed.

Historically, the forward mode was developed first, and remains the most common implementation of AD. It permits one to systematically linearize models, and by repeated application of the AD tool, to develop formal Taylor series for nonlinear models. With the rise in fluid state estimation problems of very large dimension, there has recently been a much greater emphasis on the reverse mode.

Many fluid models rely on "if...then..." and similar branching statements, such as assignment of a variable to the maximum value of a list. For example, if some region is statically unstable owing to cooling at the surface, a test for instability may lead the model to homogenize the fluid column; otherwise, the stratification is unaffected. Objections to AD are sometimes raised, apparently based on the intuitive belief that such a model cannot be differentiated. In practice, once a branch is chosen, the statevector is well-defined, as is its derivative, the AD code itself then having corresponding branches or assignments to maxima or minima of a list. A brief example of this issue is given in the Chapter Appendix. Our employment so far of the adjoint model and the adjoint evolution equation, has been in the context of minimizing an objective function—and to some degree, the adjoint has been nothing but a numerical convenience for algorithms which find minima. As we have seen repeatedly however, Lagrange multipliers have a straightforward interpretation as the sensitivity of an objective function J, to perturbations in problem parameters. This use of the multipliers can be developed independently of the state estimation problem.

Example.

Consider the linear time invariant model

$$\mathbf{x}(n) = \mathbf{A}\mathbf{x}(n-1),$$

such that

$$\mathbf{x}(n) = \mathbf{A}^n \mathbf{x}(0).$$

Suppose we seek the dependence of $H = \mathbf{x}(n)^T \mathbf{x}(n)/2 = \left(\mathbf{x}(0)^T \mathbf{A}^{nT} \mathbf{A}^n \mathbf{x}(0)\right)/2$ *on the problem parameters. The sensitivity to the initial conditions is straightforward,*

$$\frac{\partial H}{\partial \mathbf{x}(0)} = \mathbf{A}^{nT} \mathbf{A}^n \mathbf{x}(0).$$

Suppose instead that $\mathbf{A}$ *depends upon an internal parameter,* k*, perhaps the spring constant in the example of the discrete mass-spring oscillator, for which* H *would be an energy. Then,*

$$\frac{\partial H}{\partial k}=\frac{\partial\left(\mathbf{x}\left(n\right)^{T}\mathbf{x}\left(n\right)\right)}{\partial k}=\left(\frac{\partial\left(\mathbf{x}\left(n\right)^{T}\mathbf{x}\left(n\right)\right)}{\partial\mathbf{x}\left(n\right)}\right)^{T}\frac{\partial\mathbf{x}\left(n\right)}{\partial k}=\mathbf{x}\left(n\right)^{T}\frac{\partial\mathbf{x}\left(n\right)}{\partial k}.$$

We have from Eq. (2.33),

$$\frac{d\mathbf{x}\left(n\right)}{dk}=\frac{d\mathbf{A}^{n}}{dk}\mathbf{x}\left(0\right)=\left[\frac{d\mathbf{A}}{dk}\mathbf{A}^{n-1}+\mathbf{A}\frac{d\mathbf{A}}{dk}\mathbf{A}^{n-2}+...+\mathbf{A}^{n-1}\frac{d\mathbf{A}}{dk}\right]\mathbf{x}\left(0\right),$$

and so,

$$\frac{\partial H}{\partial k}=\mathbf{x}\left(0\right)^{T}\mathbf{A}^{nT}\left[\frac{d\mathbf{A}}{dk}\mathbf{A}^{n-1}+\mathbf{A}\frac{d\mathbf{A}}{dk}\mathbf{A}^{n-2}+...+\mathbf{A}^{n-1}\frac{d\mathbf{A}}{dk}\right]\mathbf{x}\left(0\right).$$

and with evaluation of $d\mathbf{A}/dk$ *being straightforward, we are finished.*

The Chapter Appendix describes briefly how computer programs can be generated to carry out these operations.

4.7.5 Approximate Methods

All of the inverse problems discussed, whether time-independent or not, were reduced ultimately to finding the minimum of an objective function, either in unconstrained form [e.g., (2.350) or 4.61] or constrained by exact relationships (e.g., models) (2.352) or (4.97). Once the model is formulated, the objective function agreed on, and the data obtained in appropriate form (often the most difficult step), the formal solution is reduced to finding the constrained or unconstrained minimum. "Optimization theory" is a very large, very sophisticated subject directed at finding such minima, and the methods we have described here—sequential estimation and Lagrange multiplier methods—are only two of a number of possibilities.

As we have seen, some of the methods stop at the point of finding a minimum and do not readily produce an estimate of the uncertainty of the solution. One can distinguish inverse methods from optimization methods by the requirement of the former for the requisite uncertainty estimates. Nonetheless, as noted before in some problems, mere knowledge that there is at least one solution may be of intense interest, irrespective of whether it is unique or whether its stability to perturbations in the data or model is well understood.

The reader interested in optimization methods generally is referred to the literature on that subject.[148] Geophysical fluid problems often fall into the category of extremely large, nonlinear optimization, one which tends to preclude the general use of many methods that are attractive for problems of more modest size.

The continued exploration of ways to reduce the computational load without significantly degrading either the proximity to the true minimum or the information content (the uncertainties of the results) is a very high priority. Several approaches are known. The use of steady-state filters and smoothers has already been discussed. Textbooks discuss a variety of possibilities for simplifying various elements of the solutions. In addition to the steady-state assumption, methods include: (1) "state reduction"—attempting to remove from the model (and thus from the uncertainty calculation) elements of the state vector that are either of no interest or comparatively unchanging;[149] (2) "reduced-order observers",[150] in which some components of the model are so well observed that they do not need to be calculated; (3) proving or assuming that the uncertainty matrices (or the corresponding information matrices) are block diagonal or banded, permitting use of a variety of sparse algorithms. This list is not exhaustive.