6 Overview of Numerical Methods in Environmental Modeling

Up to this point, this book has dealt primarily with idealized models of environmental systems. These models assume uniform geometry, constant system properties, and inputs and boundary conditions that are either constant over time or are described by special time-varying equations such as exponential or sinusoidal functions. The environmental system is represented in many of these models by a single reactor, such as a 0D completely stirred tank reactor, a 1D plug-flow reactor, or a 2D advective–dispersive reactor, with the reaction kinetics usually assumed to be first order. For these idealized conditions, analytical solutions, that is, closed-form equations, can often be derived for the model state variables: concentrations, mass fluxes, or (as introduced in Chapter 13) human exposure and risk. However, when models are developed to provide more detailed and realistic representations of systems exhibiting spatial heterogeneity, temporal variation, and nonlinear reaction kinetics, analytical solutions are generally not available. Numerical solution methods are then needed.

Numerical methods are commonly used to integrate the mass-balance differential equation for contaminant concentration C, with the objective of determining the concentration C at a specific location (x, y, z) or at a specific time t. When the derivative of C appears in the equation with respect to only one variable, usually time (i.e., dC/dt) or a single spatial dimension (such as dC/dx), the equation is referred to as an *ordinary differential equation*, or ODE. A set of coupled equations for multiple contaminants (e.g., describing dC_1/dt, dC_2/dt, etc.) is a *system* of ordinary differential equations. Section 6.1 presents methods for the numerical solution of systems of ODEs. When the concentration derivatives appear in more than one dimension, the equations are referred to as *partial differential equations*, or PDEs. Methods for solving PDEs, which are usually implemented over a spatial system of grid points or cells, are presented in Section 6.2. This section also presents methods for solving systems of linear algebraic equations that arise in numerical techniques for PDEs. Simultaneous nonlinear algebraic equations are often required to describe equilibrium chemistry in environmental models; numerical methods for these systems are described in Section 6.3. The numerical methods described in this chapter are used to develop deterministic models of pollutant transport employing point estimates for various input parameters that appear in the mass-balance equations, for example, fluid flow rates or reaction rate constants. Chapter 7 provides an introduction to random variables and random processes, which are used in stochastic models to simulate both variability in the environment and the uncertainty of model predictions.

This chapter is intended to provide only an overview of the computational methods considered necessary for understanding many of the full-scale models presented in subsequent chapters. Texts devoted to numerical methods (e.g., Chapra and Canale, 1988) provide more detailed discussions. Press et al. (1994) explain and outline computer algorithms for implementing a wide variety of numerical techniques. A more in-depth study of these techniques is recommended for anyone developing models in which they are used or for a full appreciation of the methods used by others.

6.1 ORDINARY DIFFERENTIAL EQUATIONS

Consider the mass-balance equation for the indoor air pollution problem depicted in Chapter 1 (Fig. 1.5), as given by Eq. (1.8):

$$V\frac{dC_i}{dt} = (Q_{\text{in}})\,C_{i,\text{amb}} - (Q_{\text{out}})\,C_i + S - kVC_i \tag{6.1}$$

where C_i represents the pollutant concentration within a well-mixed indoor air compartment, i. Because the indoor air compartment is well-mixed, C_i is independent of location and varies as a function of time only, that is, $C_i = C_i(t)$; $C_{i,\text{amb}}$ represents the influent concentration entering the air compartment from the surroundings. Dividing both sides of the equation by the volume of the room, V, the equation for the concentration derivative is obtained:

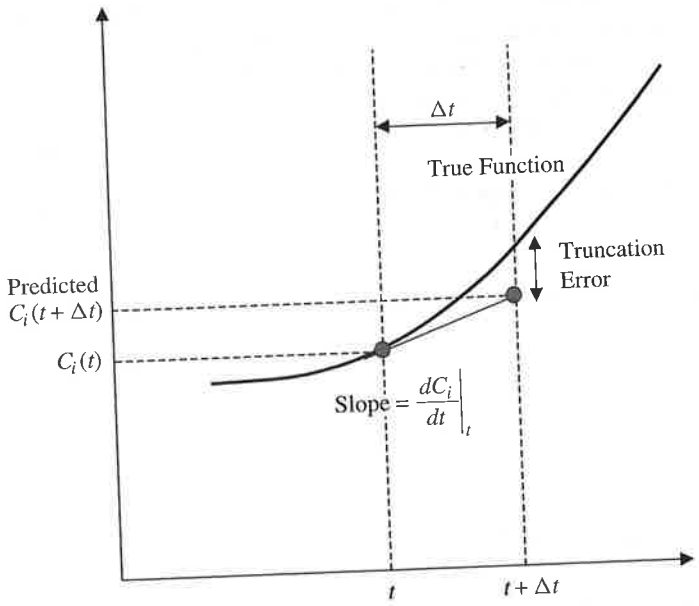
$$\frac{dC_i}{dt} = \frac{(Q_{\text{in}})\,C_{i,\text{amb}} + S}{V} - \left(\frac{Q_{\text{out}}}{V} + k\right)C_i \tag{6.2}$$

Equation (6.2) is a first-order ODE. The order of the ODE refers to the order of the derivative. For example, an equation with derivatives up to and including d^2C_i/dt^2 is a second-order differential equation. We limit ourselves in this section to solutions for first-order ODEs. Methods for solving higher-order ODEs are described elsewhere (Hoffman, 1992, p. 296).

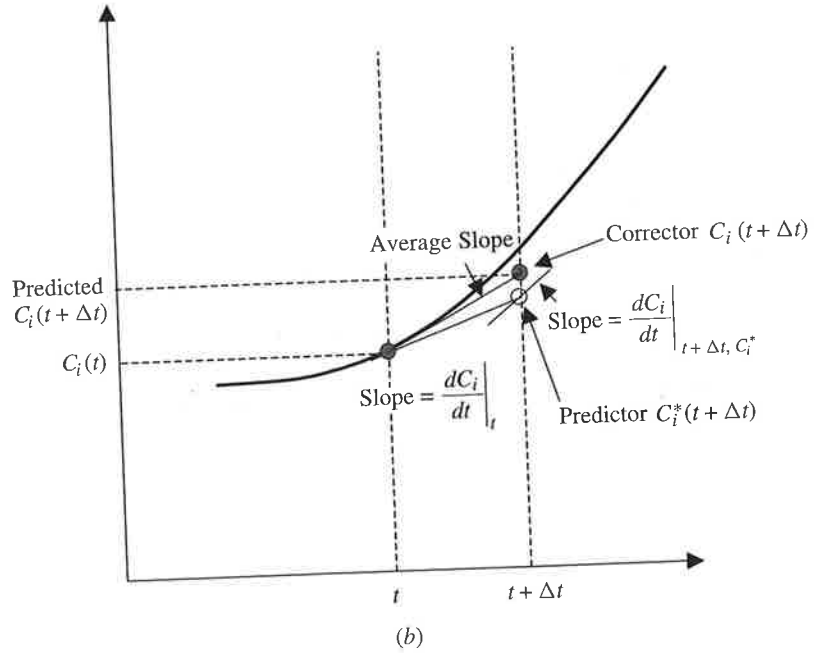
For a given set of inputs that define the rhs of Eq. (6.2), we seek a procedure for moving from a known value of C_i at time t to an unknown value of C_i at time $t + \Delta t$.[1] This problem is depicted in Figure 6.1, which shows a true (but unknown to us) value of $C_i(t)$ that we are attempting to reproduce through numerical integration. Figure 6.1*a* shows perhaps the simplest logical solution to this problem (the one you might come up with yourself if you were stuck on a deserted island with no previous knowledge of numerical methods): Use the known slope of the curve at time t to extrapolate the value forward to time $t + \Delta t$:

$$C_i(t + \Delta t) = C_i(t) + dC_i/dt|_t\,\Delta t \tag{6.3}$$

[1] This can be solved for analytically in the case of Eq. (6.2). In the more general case of nonlinear kinetics, multiple contaminants, and the like, it usually cannot.



(a)



(b)

Figure 6.1 Numerical integration of unknown function $C_i(t)$ using (*a*) the Euler method and (*b*) the predictor–corrector method.

Equation (6.3) is known as the Euler or Euler–Cauchy formula. As indicated in Figure 6.1*a*, Eq. (6.3) is likely to introduce some error, especially when the true solution is highly nonlinear and when it is implemented over large time steps, Δt.

How might the error indicated in Figure 6.1*a* be reduced? Figure 6.1*b* suggests an option. The derivative could be recomputed at time $t + \Delta t$, and then the linear extrapolation from time t recomputed using a slope that is the *average* of the initial derivative at time t and that subsequently computed at time $t + \Delta t$. This procedure, known as the predictor–corrector method, is implemented in two steps:

$$\text{Predictor} \qquad C_i^*(t+\Delta t) = C_i(t) + \left.\frac{dC_i}{dt}\right|_t \Delta t$$

$$\text{Corrector} \qquad C_i(t+\Delta t) = C_i(t) + \frac{\left.\frac{dC_i}{dt}\right|_t + \left.\frac{dC_i}{dt}\right|_{t+\Delta t, C_i^*}}{2} \Delta t \tag{6.4}$$

The first predictor step is identical to the Euler method in Eq. (6.3). Note that when multiple model outputs, $C_j(t)$, $j = 1, \ldots, J$, are required either to model J coupled compartments across which a contaminant is distributed or the coupled behavior of J chemical pollutants within a single compartment, a system of J ODEs is obtained. In this case, the predictor step is implemented for each of the J state variables to compute each of the $C_j^*(t)$, before computing the correctors. The second corrector step yields modified estimates, which are expected to be closer to the true values than are the predictors. However, they are still not completely accurate, due to the nonlinearity of the true solutions and the fact that the derivatives at time $(t + \Delta t)$ are computed using only estimates of the $C_j(t + \Delta t)$'s.

A more formal approach for describing the accuracy of numerical integration recognizes that two types of error can occur in the transition from time t to $t+\Delta t$. The first, depicted in Figure 6.1, is known as *truncation error*. This is the inherent error of the method. For the Euler method, the truncation error is proportional to $(\Delta t)^2$, the time step raised to the second power. As such, cutting the time step in half reduces the error by a factor of 4. For the predictor–corrector method, the truncation error is proportional to $(\Delta t)^3$, so that reducing the time step by a factor of 2 reduces the error by a factor of 8. More accurate integration is therefore expected using smaller time steps and the predictor–corrector method, than can be achieved with the Euler method.

If less truncation error occurs with smaller time steps, why not use smaller and smaller values of Δt until the desired accuracy is achieved? A first, practical reason is that the computation time increases as Δt is reduced. (More calculations are needed to integrate over the same time interval.) A second, more fundamental reason involves the second type of error: *round-off error*. The second terms that are added to $C_i(t)$ on the rhs of Eq. (6.3) or (6.4) become smaller and smaller as the time step is reduced. With a limited number of significant digits used for the calculation by the computer (typically 8, or 16 if double precision is used), a greater relative error can occur as round-off eliminates a higher fraction of the addend. While the truncation error

is reduced as Δt is made smaller, the round-off error increases. Furthermore, this increased relative error is repeated over more, shorter time steps. The net effect of both truncation error and round-off error over *many* time steps is referred to as the overall *propagation error*.

How can you tell whether a numerical integration is accurate? You do not know the true solution (i.e., from an analytical solution); if you did, you would use it! However, you might be able to simplify your model to a special case, for example, with constant spatial and temporal properties, simplified kinetics, and the like, for which a known analytical solution *is* available. The numerical method should be able to reasonably reproduce the known solution for this special case. Comparison of numerical solutions to known analytical solutions for simplified, idealized cases is thus a common and important first step in testing for accuracy. It does not guarantee that the numerical solution will be accurate for the real, more complex cases that you really care about. However, it does provide some degree of comfort and assurance. If the model cannot reproduce the analytical solution for simplified cases, then something is clearly wrong—either inherently with the method or in its computer implementation.

A second way to diagnose the accuracy of a numerical integration procedure is to evaluate the model with varying time steps. Initially, the time step is chosen based on the time scales of variation in model inputs and responses. For example, a model with variations in emissions, transport terms, reaction rates, and resulting concentrations over time scales of minutes and hours will typically require time steps of seconds for numerical integration; models with variations over weeks, months, and years typically require time steps of days or fractions of a day. A high estimate of the time step is first selected, the model executed, and the results recorded. A second, smaller time step (e.g., one-half of the value of Δt used for the first test run) is used and the results compared to those from the first case. If the initial time step was indeed too large, the results should be different. Successively smaller time steps are tested and the differences between runs should diminish, until reducing the time step further no longer yields a change in the results. This indicates that a sufficiently small time step has been selected and that accurate numerical integration has most likely been achieved. Eventually, reducing the time step further should once again yield changes in model predictions, as round-off error comes into play. The assumption is that initially reductions in Δt act to reduce the truncation error and that round-off error does not become significant until the time step is reduced to a very small value. This is usually the case, especially if the initial time step is chosen to be "conservatively large" for the problem under consideration. Other methods are available for diagnosing model accuracy, such as checking the model for mass balance. These and other quality control procedures for model evaluation are illustrated in the examples presented in this chapter and discussed further in Chapter 14.

The Euler and the predictor–corrector methods are among the simplest of the available procedures for numerical integration of ODEs but are not especially accurate. For some problems, they can yield unstable results. Instability occurs when the results deviate so far from the correct values that wildly diverging or oscillating predictions are made. Instability is not easy to define, but you know it when you see it. More

sophisticated numerical methods may be able to maintain stability and achieve significantly greater accuracy. Some of these methods utilize model results prior to t (e.g., at time $t - \Delta t$ and $t - 2\,\Delta t$) in making the transition from t to $t + \Delta t$. Use of these multiple values allows the higher-order shape and associated derivatives of the function to be taken into consideration. While such multistep methods are more accurate, they are not self-starting, since at time $t = 0$, values at $t - \Delta t$ and $t - 2\,\Delta t$ are not available. A self-starting method is thus needed for the initial calculations over the first few time steps. Some of the stability and accuracy of multistep methods can be achieved by self-starting methods if each time step is broken up into partial steps. Among the most widely used of these partial step procedures is the Runge–Kutta method.

The Runge–Kutta procedure is actually a family of methods, each with different "order" depending on how many partial steps are utilized within each time step. The fourth-order Runge–Kutta method is especially popular, due to its very high accuracy and stability, yet relative simplicity. The accuracy of the fourth-order Runge–Kutta method is related to that of Simpson's method for numerical integration, illustrated in Figure 6.2. An accurate estimate of $A = \int f(x)\,dx$ (i.e., the shaded area under the curve in Fig. 6.2) is computed using Simpson's rule as follows:

$$A = (\Delta x/6)\,[f(a) + 4f(b) + f(c)] \tag{6.5}$$

Where a, b, and c are evenly-spaced points in the interval $(x, x + \Delta x)$. In numerical solution of chemical mass-balance ODEs, the function $f(x)$ corresponds with the time derivative dC_i/dt that is integrated over a time step, Δt, to determine the unknown function $C_i(t)$.

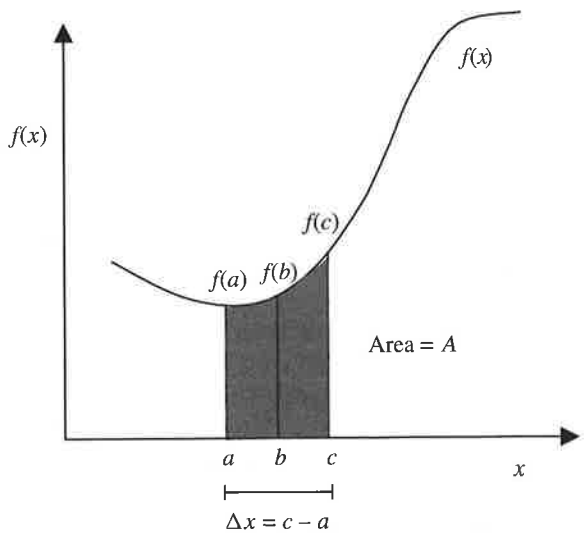


Figure 6.2 Formulation of Simpson's method for integration [Eq. (6.5)].

As in Simpson's method, the fourth-order Runge–Kutta method similarly divides the time step in half and computes values of $C(t)$ at the next time step from the derivatives, F_1, F_2, F_3, and F_4, as:

$$C_i(t + \Delta t) = C_i(t) + \tfrac{1}{6}\left[F_1^i + 2F_2^i + 2F_3^i + F_4^i\right] \tag{6.6}$$

where

$$F_1^i = \left[\left.\frac{dC_i}{dt}\right|_{t, C_i = C_i(t)}\right] \Delta t$$

$$F_2^i = \left[\left.\frac{dC_i}{dt}\right|_{t+(\Delta t/2), C_i = C_i(t)+0.5\,F_1^i}\right] \Delta t$$

$$F_3^i = \left[\left.\frac{dC_i}{dt}\right|_{t+(\Delta t/2), C_i = C_i(t)+0.5\,F_2^i}\right] \Delta t$$

$$F_4^i = \left[\left.\frac{dC_i}{dt}\right|_{t+\Delta t, C_i = C_i(t)+F_3^i}\right] \Delta t$$

As with the predictor–corrector method, when the Runge–Kutta method is applied to a system of ODEs for a suite of constituents, the F_1^j's must be computed for each constituent $j = 1, \ldots, J$, before moving on to calculate each of the F_2^j's, and so on. The derivative of C_i used to calculate each of the F's is evaluated with model inputs set at the indicated times (t, $t + \Delta t/2$, $t + \Delta t/2$ and $t + \Delta t$, for F_1, F_2, F_3 and F_4, respectively) and with "predictor" values of C_i computed as shown. The fourth-order Runge–Kutta method has an error proportional to $(\Delta t)^5$, so very high accuracy can be achieved as the time step is reduced.

EXAMPLE 6.1 NUMERICAL INTEGRATION OF A SIMPLE FOOD CHAIN MODEL

To illustrate procedures for numerical integration and the sensitivity of numerical results to different methods and time steps, the idealized system for nutrient uptake and growth of phytoplankton and zooplankton shown in Figure 6.3 is considered. [This example is based on Section 14.1.3 of Chapra and Reckhow (1983).] The model simulates the cycling of phosphorus between three species: inorganic phosphorus, p_1; phytoplankton, p_2; and zooplankton, p_3. The phytoplankton grow via uptake of inorganic phosphorus and are subsequently consumed by the zooplankton. The zooplankton grow as a result of this consumption, but die and degrade

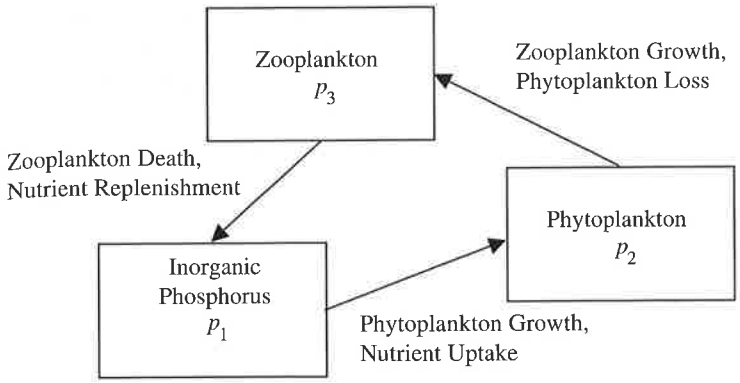


Figure 6.3 Simple phosphorus food chain cycle for Example 6.1.

back into inorganic P. Lotka–Volterra predator–prey relationships are defined for the phytoplankton and zooplankton, with nutrient uptake and growth dependent on both the "prey" (inorganic P for the phytoplankton; phytoplankton for the zooplankton) and predator concentrations. The concentrations of each of the three species are represented in terms of their phosphorus content [M(P) L^{-3}].

The model assumes nutrient (inorganic phosphorus)-limited uptake and growth of the phytoplankton, described by Michaelis–Menten kinetics (see Chapter 12):

$$\text{Phytoplankton growth rate} = k_m\left(\frac{p_1}{K_s + p_1}\right) p_2$$

where k_m (T^{-1}) is the maximum growth rate and K_s [M(P) L^{-3}] is the half saturation constant, equal to the nutrient concentration at which the growth rate of phytoplankton is half of its maximum value. The phytoplankton are consumed by zooplankton grazing:

$$\text{Zooplankton growth rate} = k_{23}\, p_2\, p_3$$

where k_{23} [$\{M(\text{P})\ L^{-3}\}^{-1}\ T^{-1}$] is a second-order rate constant, referred to as the zooplankton feeding rate. The rate of zooplankton death and consequent nutrient replenishment is given by:

$$\text{Nutrient replenishment rate} = k_z\, p_3$$

where k_z (T^{-1}) is the first-order zooplankton death rate.

The phosphorus–phytoplankton–zooplankton food chain is simulated for a batch reactor, considering only kinetic processes with assumed constant rate coefficients. In real aquatic systems these kinetic processes are supplemented by a seasonal pattern of loadings and discharge from the water body, with temperature- and light-driven variations in the rate constants. Examination of the growth patterns

and nutrient cycling predicted to occur in a closed system is nonetheless useful to begin to understand the dynamics and cyclical nature of the food chain. This idealized system also provides a good illustration of the behavior of numerical solutions for ODEs.

The three simultaneous, nonlinear ordinary differential equations for the system are

$$\frac{dp_1}{dt} = k_z\, p_3 - k_m \frac{p_1}{K_s + p_1} p_2$$

$$\frac{dp_2}{dt} = k_m \frac{p_1}{K_s + p_1} p_2 - k_{23}\, p_2\, p_3 \qquad (6.7)$$

$$\frac{dp_3}{dt} = k_{23}\, p_2\, p_3 - k_z\, p_3$$

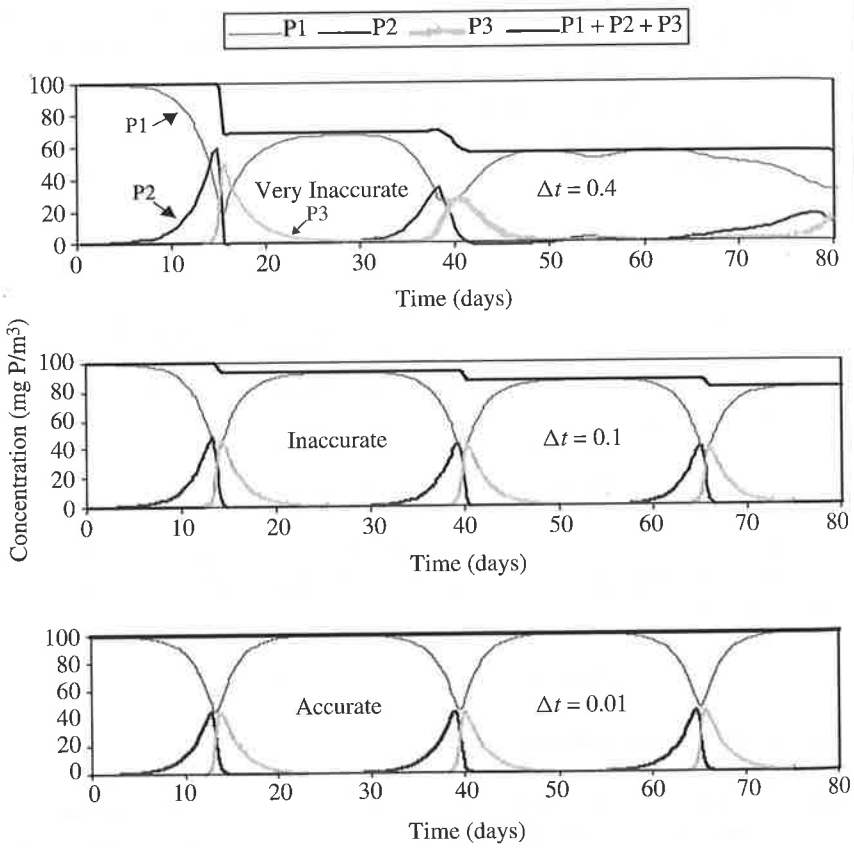


Figure 6.4 Numerical solution of the phosphorus food chain model using the predictor–corrector method.

Following Chapra and Reckhow (1983), the following coefficients are assumed for the model: $k_m = 0.5\ \text{day}^{-1}$; $K_s = 2\ \text{mg}\ P/\text{m}^{-3}$; $k_{23} = 0.1\ (\text{mg}\ P/\text{m}^{-3})^{-1}\ \text{day}^{-1}$; and $k_z = 0.5\ \text{day}^{-1}$; with initial conditions: $p_1(0) = 99.8$ and $p_2(0) = p_3(0) = 0.1\ \text{mg}\ P/\text{m}^{-3}$.

Numerical simulation results for this model are calculated using the predictor–corrector method and the Runge–Kutta method, with results shown in Figures 6.4 and 6.5, respectively. For each method, time steps of $\Delta t = 0.4$, 0.1, and 0.01 days are utilized. Figures 6.4 and 6.5 also show the computed total P concentration: $p_T(t) = p_1(t) + p_2(t) + p_3(t)$; which, because the system is closed (and constant volume), should remain equal to the initial value of 100 mgP/m^{-3}, as long as an accurate mass balance is maintained for the system.

As indicated in the figures, the results are sensitive to the time step chosen. With too large a time step, inaccurate results are obtained, especially with the less

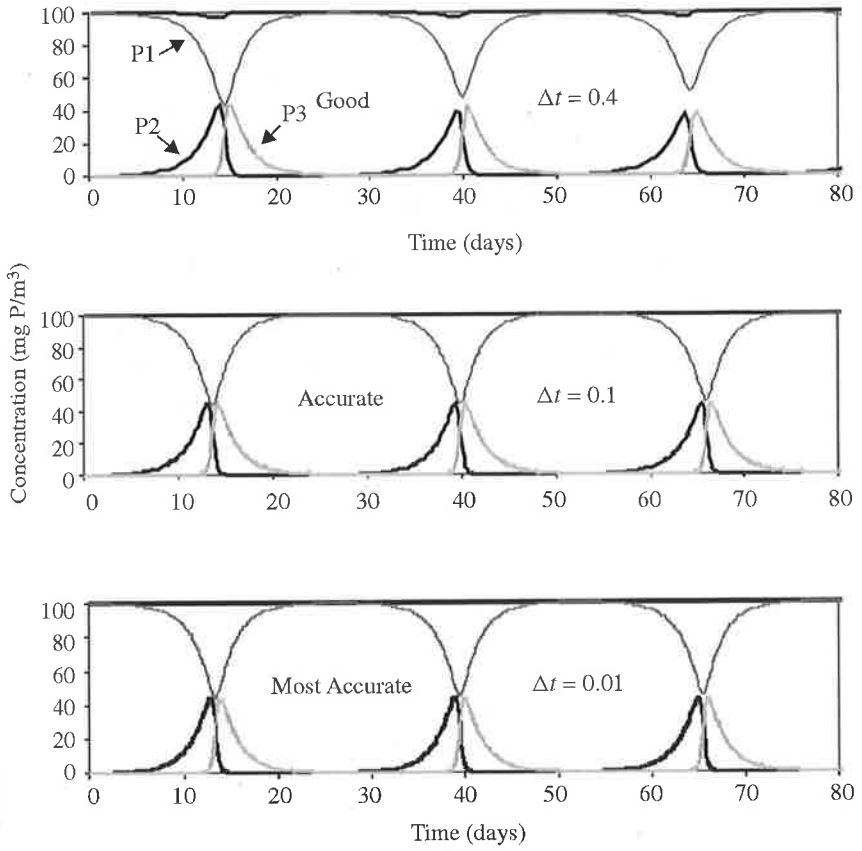


Figure 6.5 Numerical solution of the phosphorus food chain model using the fourth-order Runge–Kutta method.

accurate predictor–corrector method.[2] If too large a time step is chosen, irregular results can be generated, with significant deterioration of the mass balance and (though not evident in these simulations) even predictions of negative concentrations. When state variables that must be nonnegative violate this physical requirement, this is a clear indication of error and usually a precursor to instability. It is tempting in this situation to consider simple corrections to the algorithm, such as setting all predicted negative concentrations equal to zero or to a very small number. However, such ad hoc corrections generally introduce further mass-balance errors into the solution. Rather, the preferred approach is to try smaller time steps or utilize an alternative, more accurate solution method.[3] These steps and the results in Figures 6.4 and 6.5 illustrate the type of trial-and-error testing that typically must be done when developing and implementing a numerical solution.

Finding the right time step to use in solving systems of ordinary differential equations is particularly difficult if the system is *stiff*, meaning that changes in the magnitude of some of the state variables occur orders of magnitude more quickly than for other variables. The mathematical consequence of stiffness is that the solution to a system of ordinary differential equations requires inverting a matrix that is nearly singular. Chapter 11 addresses solution techniques for stiff systems of ODEs because the rate equations for ozone formation in the atmosphere are a prime example of a stiff system.

Over the past decade, a number of convenient and powerful mathematics software packages have become widely available. Mathcad (MathSoft, 1997), Matlab (MathWorks, 1995), and Mathematica (Wolfram, 1991) all include functions that will numerically integrate systems of ODEs. In addition to Runge–Kutta schemes, Mathcad and Mathematica provide specialized functions with adaptive time steps that can handle stiff systems. The primary limitation of these software packages is that they offer relatively little flexibility for formatting model inputs and outputs. However, when such flexibility is not required, they offer a useful shortcut for numerical analysis.

[2] The "correct" results are understood to be those achieved with an accurate method at small time steps (though not *too* small, due to round-off error), exhibiting stable and repeatable behavior, and maintaining overall mass balance. See also Figure 14.5d of Chapra and Reckrow (1983, p. 262).

[3] Another trick that can be used when negative concentrations are simulated is to transform the problem to one described in terms of variable(s) that ensure that the positivity requirement is met. For example, dividing both sides of Eq. (6.2) by C_i yields an equation in terms of the transformed variable $Y_i = \ln(C_i)$, since

$$\frac{dC_i}{C_i\,dt} = \frac{d\ln C_i}{dt} = \frac{dY_i}{dt} = \frac{(Q_{\text{in}})C_{\text{amb}} + S}{V\exp(Y_i)} - \left(\frac{Q_{\text{out}}}{V} + k\right)$$

This model may be solved in terms of Y_i and subsequently transformed back to the targeted variable, $C_i(t) = \exp[Y_i(t)]$. No matter what the value of $Y_i(t)$, $C_i(t)$ remains nonnegative.

6.2 PARTIAL DIFFERENTIAL EQUATIONS

When mass-balance equations are specified with derivatives in more than one dimension, such as time, and one or more spatial dimensions for dynamic models or more than one spatial dimension for steady-state models, the system is then described by *partial* differential equations. Consider the equation for two-dimensional advective–dispersive transport with first-order decay:

$$\frac{\partial C}{\partial t} = -u\frac{\partial C}{\partial x} + D_x\frac{\partial^2 C}{\partial x^2} + D_y\frac{\partial^2 C}{\partial y^2} - kC \tag{6.8}$$

In this application concentration variations are considered over time and in the x and y directions, that is, $C = C(x, y, t)$. While analytical solutions to this equation are available when u, D_x, D_y, and k are constant over time and space (as presented in Chapter 5, dependent on the assumed boundary and initial conditions), numerical methods are required when these parameters vary temporally and/or spatially. The methods involve discretized calculations over both the temporal and spatial dimensions. Two of the most common methods for implementing this type of solution, finite difference and finite element methods, are described in this section.

6.2.1 Finite Difference Method

The finite difference method solves the mass-balance equation(s) by forcing them to be satisfied at a set of discrete points in space. Figure 6.6 illustrates a two-dimensional grid that might be used to solve Eq. (6.8). Nodes in the longitudinal (x) direction are indexed by i, while nodes in the transverse (y) direction are indexed by j, with $C(i, j, t)$ indicating the concentration at grid point i, j at time t. The key step in the finite difference method is to express the derivatives for concentration at each point i, j as appropriate differences between concentrations at adjacent nodes. For example, for the first (advection) term on the rhs of Eq. (6.8), the first derivative of $C(i, j, t)$ with respect to x is needed and could be expressed using either:

$$\frac{\partial C(i, j, t)}{\partial x} = \frac{C(i+1, j, t) - C(i, j, t)}{\Delta x} \tag{6.9}$$

or

$$\frac{\partial C(i, j, t)}{\partial x} = \frac{C(i, j, t) - C(i-1, j, t)}{\Delta x} \tag{6.10}$$

where Δx is the distance between nodes in the x direction. Equation (6.9) uses *forward differencing* for the advection term. It approximates the derivative at point i, j using the difference between the concentration one node *downstream* of the point (in the direction of advection) and $C(i, j, t)$. Alternatively, Eq. (6.10) uses *backward differencing* since the first derivative is computed by taking the difference between $C(i, j, t)$ and the concentration at the node one step *upstream*. Neither approach is expected to provide an especially accurate estimate of the concentration derivative

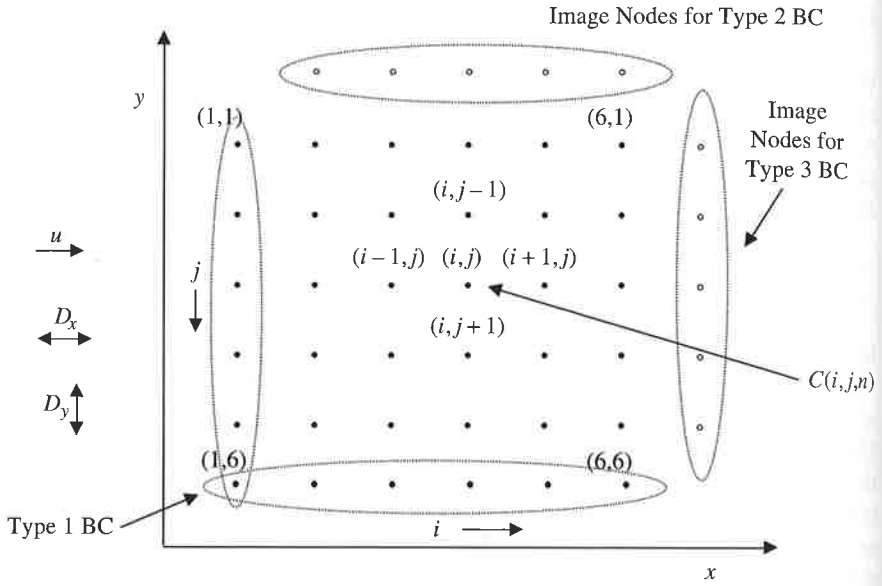


Figure 6.6 Finite difference grid with 6 × 6 (solid) node system. Image nodes (open) added for subsequent implementation of boundary conditions.

at point i, j, especially if its value is changing rapidly as a function of x. A third alternative is achieved by taking the average of the forward and backward differencing equations:

$$\frac{\partial C(i,j,t)}{\partial x} = \frac{\text{forward} + \text{backward}}{2} = \frac{C(i+1,j,t) - C(i-1,j,t)}{2\,\Delta x} \tag{6.11}$$

This approach, referred to as *central differencing* for the advection term, does not even use the value of the concentration at point i, j; rather, it calculates the derivative by drawing a straight line between the concentrations immediately up- and downstream of the target location.

Referring back to the finite *cell* method introduced in Section 5.7, the same terminology—backward, forward, or central differencing—was used to describe the weighting of concentrations passing with the advective flow across the interface of two cells. There is a clear parallel between the finite cell method, where concentrations are averaged over volumetric compartments and mass transport occurs between them, and the finite difference method in which mass-balance equations are satisfied and concentrations computed at points in space. In the latter, the points are still thought to be representative of the spatial domain around them, that is, the degree of spatial representation is still limited by the coarseness of the grid. Likewise, many of the same issues that arise with respect to accuracy, stability, and numerical dispersion in the finite cell method apply to the finite difference method, especially with regard to the use of backward, forward, or central differencing. These issues are addressed in more detail below. First, equations for the remaining derivatives in Eq. (6.8) are developed.

The dispersion terms in Eq. (6.8) require a differencing expression for second derivatives. Recognizing that the second derivative describes the rate of change in the first derivative with distance, it can be computed by taking the difference between the forward difference estimate of the first derivative (which best describes the value of the first derivative midway between node i, j and node $i+1, j$) and the backward difference estimate (for the point midway between node i, j and node $i-1, j$), and dividing by the distance between these points, Δx:

$$\frac{\partial^2 C(i,j,t)}{\partial x^2} = \frac{\partial\left(\frac{\partial C}{\partial x}\right)}{\partial x} = \frac{\left[\frac{C(i+1,j,t) - C(i,j,t)}{\Delta x}\right] - \left[\frac{C(i,j,t) - C(i-1,j,t)}{\Delta x}\right]}{\Delta x}$$

$$= \frac{C(i+1,j,t) - 2C(i,j,t) + C(i-1,j,t)}{(\Delta x)^2} \tag{6.12}$$

A similar expression describes the second derivative for the dispersion term in the y direction:

$$\frac{\partial^2 C(i,j,t)}{\partial y^2} = \frac{C(i,j+1,t) - 2C(i,j,t) + C(i,j-1,t)}{(\Delta y)^2} \tag{6.13}$$

The second derivative in each case is estimated by adding the concentrations at the adjacent nodes and subtracting twice the concentration at the target node, and then dividing by the square of the internode distance.

The final derivative that must be specified for dynamic solution of Eq. (6.8) is the time derivative. Indexing discrete points in time by $t = 1, 2, \ldots n, n+1, \ldots$, where n is the current time in the simulation, the time derivative is first expressed using a simple Euler expression (forward differencing in time):

$$\frac{\partial C(i,j,n)}{\partial t} = \frac{C(i,j,n+1) - C(i,j,n)}{\Delta t} \tag{6.14}$$

Rearranging to solve for $C(i, j, n+1)$:

$$C(i,j,n+1) = C(i,j,n) + \frac{\partial C(i,j,n)}{\partial t}\,\Delta t \tag{6.15}$$

Finally, substituting for the terms on the rhs of Eq. (6.8), including the central difference expression for the advection term, the following equation is obtained:

$$C(i,j,n+1) = C(i,j,n) + \Delta t \left\{ \begin{array}{l} -u\left[\dfrac{C(i+1,j,n) - C(i-1,j,n)}{2\Delta x}\right] \\ +D_x\left[\dfrac{C(i+1,j,n) - 2C(i,j,n) + C(i-1,j,n)}{(\Delta x)^2}\right] \\ +D_y\left[\dfrac{C(i,j+1,n) - 2C(i,j,n) + C(i,j-1,n)}{(\Delta y)^2}\right] - kC(i,j,n) \end{array} \right\} \tag{6.16}$$

Equation (6.16) is *explicit*, that is, $C(i, j, n+1)$ at node i, j at time step $n+1$ is computed solely from values of C available at the current time n, at node i, j and the four surrounding nodes. While relatively easy to implement, explicit solutions may exhibit problems with stability. These problems are discussed in the following section, followed by presentation of an implicit solution method that addresses them.

Stability, Numerical Dispersion, and Implicit Solution Methods As with the finite cell method, finite difference solutions using central differencing are prone to instability when applied to highly advective, low-dispersion systems. Central differencing exhibits inherent "static" instability whenever the grid size is too large relative to the ratio of the dispersion to velocity. Stability is maintained when

$$\Delta x \leq \frac{2D}{u} \tag{6.17}$$

This is equivalent to requiring that the Peclet number be less than or equal to 2, where the Peclet number is defined as $\text{Pe} = \Delta x\, u/D$. Highly advective (high Peclet number) domains, such as the riverine portion of a coastal water system, thus require finer grid spacing. This requirement applies when implementing either a steady-state or a dynamic solution. When a dynamic solution such as Eq. (6.16) is implemented, a further requirement is placed on the time step, Δt. To avoid dynamic instability, the time step must be chosen so that (for a one-dimensional problem):

$$\Delta t \leq \frac{(\Delta x)^2}{2D} \tag{6.18}$$

For a two-dimensional problem [such as Eq. (6.16)], the restriction is greater still, with the two in the denominator of Eq. (6.18) replaced by 4.

One approach for avoiding the restrictions on Δx imposed by the static stability requirement [Eq. (6.17)] is to use backward, instead of central, differencing. However, as with cell models, this introduces numerical dispersion. Backward differencing computes the advective transport from a value of the derivative further upstream. The resulting error has the same effect as increasing the dispersion coefficient, moving additional mass from nodes with high concentration to nodes with low concentration. In the dynamic solution the magnitude of the numerical dispersion is given by:

$$D_n = \frac{1}{2} u\, \Delta x \left(1 - \frac{u\, \Delta t}{\Delta x}\right) \tag{6.19}$$

When backward differencing is employed to address stability problems, the extra numerical dispersion must be recognized. It may be necessary to reduce the input values of the dispersion coefficient(s), that is, reduce assigned values of D (D_x and/or D_y) so that the assigned values *plus* the numerical dispersion introduced through the numerical method equals the level of dispersion targeted for the problem. If the numerical dispersion exceeds that targeted for the application, this clearly will not work (negative dispersion coefficients cannot be assigned). Furthermore, in problems

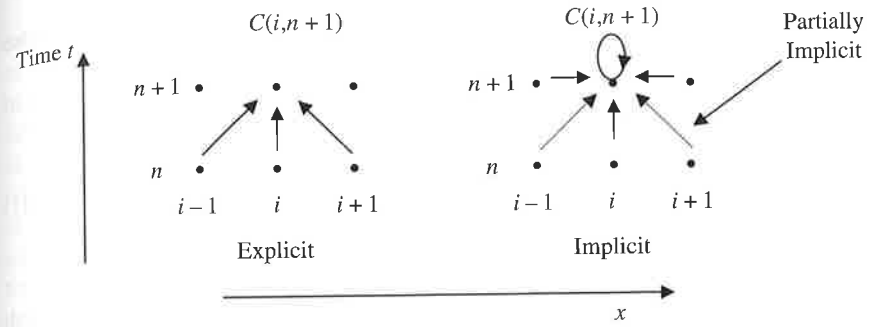


Figure 6.7 Explicit versus implicit solution of a finite difference problem over time for a spatially one-dimensional system. The explicit method uses only information available at time n to compute derivatives and move forward to time $n+1$. (Arrows show flow of information.) Implicit methods compute derivatives from the model state at time $n+1$ (and at time n if partially implicit) so that solution of simultaneous equations is required to move forward to time $n+1$.

with variable time steps and complex grid systems, it may be difficult to ascertain the magnitude of the numerical dispersion. The use of backward differencing is thus an imperfect solution to stability problems, but one that in some cases cannot be avoided, especially when solving steady-state models.

For dynamic models, an alternative approach is available for maintaining stability. The explicit approach depicted in Eq. (6.16) may be replaced by an *implicit* method. The key difference between explicit and implicit time integration is illustrated in Figure 6.7. The explicit method uses available node concentrations from time n to compute all concentrations at time $n+1$. The implicit method expresses the equations for concentration at each node at time $n+1$ in terms of concentrations at other nodes at time $n+1$. The equations are implicit because the information needed to move forward from time n to time $n+1$ is not all available at time n; rather the equations for the concentrations at the different nodes at time $n+1$ must be solved simultaneously.

A *fully* implicit solution defines all terms in the time derivative, $\partial C/\partial t$, using concentrations (and model inputs and parameters) at time $n+1$. The fully implicit equivalent of Eq. (6.16) is given by:

$$C(i, j, n+1) = C(i, j, n) + \Delta t \left\{ \begin{aligned} &-u\left[\frac{C(i+1, j, n+1) - C(i-1, j, n+1)}{2\,\Delta x}\right] \\ &+D_x\left[\frac{C(i+1, j, n+1) - 2C(i, j, n+1) + C(i-1, j, n+1)}{(\Delta x)^2}\right] \\ &+D_y\left[\frac{C(i, j+1, n+1) - 2C(i, j, n+1) + C(i, j-1, n+1)}{(\Delta y)^2}\right] - kC(i, j, n+1) \end{aligned} \right\} \tag{6.20}$$

If model inputs u, D_x, D_y, and k vary with time, their values at time $n+1$ would also be used in Eq. (6.20).

A *partially* implicit solution defines the time derivative as a weighted average of values computed using concentrations (and model inputs) at times n and $n+1$:

$$C(i, j, n+1) = C(i, j, n) + \Delta t \left\{ \alpha \left. \frac{\partial C}{\partial t} \right|_{n+1} + (1-\alpha) \left. \frac{\partial C}{\partial t} \right|_{n} \right\} \tag{6.21}$$

The case where $\alpha = 0$ corresponds to the explicit method, $\alpha = 1$ to the fully implicit method, and $0 < \alpha < 1$ to a partially implicit solution. The special case in which $\alpha = 0.5$ is known as the Crank–Nicolson method. The partially implicit solution corresponding to the explicit Eq. (6.16) and the fully implicit Eq. (6.20) is

$$\begin{aligned} C(i, j, n+1) = C(i, j, n) \\ + \Delta t\, \alpha \left\{ \begin{array}{l} -u\left[\dfrac{C(i+1, j, n+1) - C(i-1, j, n+1)}{2\,\Delta x}\right] \\ +D_x\left[\dfrac{C(i+1, j, n+1) - 2C(i, j, n+1) + C(i-1, j, n+1)}{(\Delta x)^2}\right] \\ +D_y\left[\dfrac{C(i, j+1, n+1) - 2C(i, j, n+1) + C(i, j-1, n+1)}{(\Delta y)^2}\right] - kC(i, j, n+1) \end{array} \right\} \\ + \Delta t\,(1-\alpha) \left\{ \begin{array}{l} -u\left[\dfrac{C(i+1, j, n) - C(i-1, j, n)}{2\,\Delta x}\right] \\ +D_x\left[\dfrac{C(i+1, j, n) - 2C(i, j, n) + C(i-1, j, n)}{(\Delta x)^2}\right] \\ +D_y\left[\dfrac{C(i, j+1, n) - 2C(i, j, n) + C(i, j-1, n)}{(\Delta y)^2}\right] - kC(i, j, n) \end{array} \right\} \end{aligned} \tag{6.22}$$

Implicit or partially implicit solution methods are very effective at maintaining stability. However, they increase the computational burden significantly. For a system with N total nodes, a set of N simultaneous equations must be solved *at each time step*. Efficient methods for implementing this type of solution are discussed below. First though, the remaining building blocks of the solution, the initial and boundary conditions, must be specified.

Specification of Initial and Boundary Conditions With any of the finite difference solution methods thus far described, initial and boundary conditions are needed. For dynamic simulations, initial conditions are needed for the model state variables (e.g., contaminant concentrations) at *all nodes* at time $t = 0$ in order to initiate the computations. In some problems these conditions are unknown or highly uncertain, especially when the model is used for long-term simulations involving historic reconstruction. Even when the primary interest is in recent, current, or future values predicted by the model, historic conditions, perhaps beginning many years earlier, may be needed to initiate the calculations. In some cases accurate selection of these initial conditions may not be necessary—the model equations may have a short enough "memory" that the initial conditions do not influence the solution. This is fortunate, especially if you have little information for specifying the initial conditions. For other problems, the initial conditions do matter, and they must be determined as part of the calibration or parameter estimation process. The only way to tell is to try solving the equations with different initial conditions and see how the solutions differ.

Boundary conditions are required to specify the system state variables and/or their derivatives (e.g., concentrations and mass flux conditions) at all system boundaries, whether a dynamic or a steady-state solution is implemented. Consider the set of boundary nodes identified in Figure 6.6. Along the left-hand side of the domain (where $i = 1$), values of $C(i-1, j,)$ are off the grid and unavailable. How then can the backward or central differencing expressions for the first derivative [Eqs. (6.10) and (6.11), respectively] or the expression for the second derivative [Eq. (6.12)] be included in the mass-balance equation since these both include $C(i-1, j, t)$? Similar problems arise with the identification of $C(i, j-1, t)$ for nodes along the top row, $C(i+1, j, t)$ for nodes in the last (right-hand) column, and $C(i, j+1, t)$ for nodes along the bottom row. A creative solution is needed for this dilemma.

Three types of boundary conditions may apply:

Type 1, or *Dirichlet* boundary conditions: Here the model state variables are specified, for example, the concentrations along the boundary are known (at all times);

Type 2, or *Neumann* boundary conditions: The derivatives of the model state variables are known. In the case of advective–dispersive transport, this involves specification of the spatial derivative of the concentration normal to the boundary, either $\partial C/\partial x$, $\partial C/\partial y$, or $\partial C/\partial z$.

Type 3, *Cauchy,* or *mixed* boundary conditions: Linear combinations of the state variables and their derivatives are specified.

When the concentrations along a boundary are known, a Dirichlet boundary condition is used. Given the fluid flow rate, $Q = uA$, this is equivalent to specifying the advective flux across the boundary, $= uAC$, where A is the cross-sectional area associated with the grid point (normal to the boundary). A Neumann boundary condition is used to specify the dispersive flux across a boundary $[-DA(\partial C/\partial x)]$, since it involves fixing the value of the derivative. The total (advective + dispersive) flux across a boundary, $uAC - DA(\partial C/\partial x)$, is specified using a mixed boundary condition.[4]

[4] Confusion may arise in recognizing that advection at the boundary is determined by the concentration at the boundary and not the concentration derivative since the term representing advection in the mass-balance equation includes the derivative. However, in the derivation of the mass-balance equation, this term actually expressed the *change* in advection that occurs at the point, leading to mass accumulation or loss, and not the advection across the point itself. Similarly, specifications related to dispersion are implemented by fixing the first derivative, even though the second derivative appears in the dispersion term of the mass-balance equation.

To implement a Dirichlet boundary condition, the concentrations along the appropriate boundary row or column are set to their known values throughout the calculation. To implement a Neumann or a mixed boundary condition, image nodes are added as an extra row or column beyond the grid boundary. Concentrations at these grid points are set equal to the values necessary to maintain the known flux (total or dispersive) across the boundary. The exact equation used depends upon the type of differencing used to represent the derivatives. The steps taken to specify boundary conditions and associated node calculations are illustrated in the following example.

EXAMPLE 6.2 BOUNDARY CONDITION SPECIFICATION FOR A FINITE DIFFERENCE MODEL

To illustrate different types of boundary conditions and how they might be implemented in a finite difference solution, consider once again the system depicted in Figure 6.6. The following assumptions are made for each boundary, with the indicated implications for numerical computation.

1. Concentrations are known at all times along the lhs of the domain (where $i = 1$) and along the bottom boundary ($j = 6$). The mass-balance equation [Eq. (6.22)] is not needed for these nodes. However, the known values at these boundary points are used when implementing Eq. (6.22) for the nodes in the second column ($i = 2$) and for the nodes in the next-to-the-last row ($j = 5$), since these known concentrations constitute the respective values of $C(i-1, j, n)$ (for calculations in the second column) and $C(i, j+1, n)$ (for calculations in the next-to-the-last row). As a result, given the 6×6 grid in Figure 6.6, the number of equations that must be solved is reduced by $6 + 5 = 11$. The use of type 1 boundary nodes thus reduces the number of equations that must be solved. Note that the total fluxes across the upstream and bottom boundaries are *not* specified since the derivatives (and with them, the dispersive fluxes) across these boundaries are not input, but are computed as part of the model solution.

2. Assume that zero fluxes occur across the top boundary and the downstream (right-hand) boundary. Since there is no flow across the top boundary and the advective flux is therefore zero (by definition), this is equivalent to stating that the dispersive flux across the top boundary is zero. At the downstream boundary, where fluid flow leaves the system, the *total* flux is set equal to zero.[5] In both cases a set of fictitious image nodes is needed along the boundary, at a distance Δy above the top row and Δx to the right of the last column. Assuming that the central difference expression is used to define the first derivative [as it is in Eqs. (6.16), (6.20), and (6.22)], the concentrations at the image nodes are computed as follows:

For the top boundary: Beginning with the equation for the dispersive flux,

$$-D_y A \frac{\partial C}{\partial y} = 0$$

and substituting the appropriate node concentrations into the central difference expression gives

$$-D_y A \left[\frac{C(i, 2, n) - C(i, 0, n)}{2\,\Delta y}\right] = 0$$

where $j = 0$ refers to the image nodes in the row above the top boundary.

This equation can be solved for the image node concentrations at each time step. In particular, the zero-dispersive flux requirement can only be satisfied when the image node concentrations are set equal to those of their "partner" nodes in the second row:

$$C(i, 0, n) = C(i, 2, n)$$

For the downstream boundary: Beginning with the *total* flux equation

$$uAC - D_x A \frac{\partial C}{\partial x} = 0$$

and substituting the appropriate node concentrations:

$$uAC(6, j, n) - D_x A \left[\frac{C(7, j, n) - C(5, j, n)}{2\,\Delta x}\right] = 0$$

yields the desired expression for the image node concentrations $[C(7, j, n)]$ at each time step:

$$C(7, j, n) = \left[\frac{2\,\Delta x\, u}{D_x}\right] C(6, j, n) - C(5, j, n)$$

Setting the top and downstream image nodes as indicated above implements their respective no-flux boundary conditions but does nothing to reduce (or increase) the number of equations that must be solved. Thus, for the 6×6 grid system in Figure 6.6, a total of $36 - 11 = 25$ equations must be solved at each time step. With the explicit method, each of the 25 equations is solved individually at each time step, as in Eq. (6.16). With the implicit or partially implicit method, the 25 equations must be solved simultaneously at each time step.

[5] Like the semi-infinite column boundary condition used for one-dimensional analytical solutions ($\partial C/\partial x = 0$ at $x = \infty$), this boundary condition only makes sense when the downstream boundary is "far downstream," so that the concentrations computed there are not really of concern and have little effect on the computed concentrations that are of interest, further upstream. A bigger grid system, extending further downstream in the x direction, may be necessary to accomplish this, and this assumption should be tested by varying the length of extension to ensure that the downstream boundary does indeed not affect

This example does not exhaust the different types of boundary conditions that might arise in environmental models. Variations can occur in different physical systems and at different media interfaces (in problems involving mass transport across media). A further example of boundary condition selection and implementation is found in Chapter 9, where a finite difference model is used to simulate groundwater flow and contaminant transport. In each case, the boundary condition and its implementation in the model must be deduced from the underlying mass or momentum conservation and transport conditions assumed at the boundary. When poorly understood or highly uncertain boundary conditions have a significant impact on the overall mass balance of the system and predicted concentrations at points of interest, it may be necessary to determine at least some of the boundary conditions through a model calibration or parameter estimation effort. Methods for parameter estimation of unknown or uncertain model coefficients and inputs are presented in Chapter 14.

Steady-State Solution Consider now the steady-state two-dimensional advective-dispersive transport problem. The concentration time derivative in Eq. (6.8) is set equal to zero:

$$0 = -u\frac{\partial C}{\partial x} + D_x\frac{\partial^2 C}{\partial x^2} + D_y\frac{\partial^2 C}{\partial y^2} - kC \tag{6.23}$$

This equation must be satisfied at all nodes. The same differencing expressions are used for the spatial derivatives and the boundary conditions as implemented above. Using central differencing for the advection term, Eq. (6.23) becomes

$$0 = -u\left[\frac{C(i+1,j) - C(i-1,j)}{2\,\Delta x}\right] + D_x\left[\frac{C(i+1,j) - 2C(i,j) + C(i-1,j)}{(\Delta x)^2}\right]$$
$$+ D_y\left[\frac{C(i,j+1) - 2C(i,j) + C(i,j-1)}{(\Delta y)^2}\right] - kC(i,j) \tag{6.24}$$

Note that the time dimension (n) is no longer included. The equation is rewritten to isolate $C(i, j)$ on the lhs:

$$C(i,j) = \left\langle [-u]\left[\frac{C(i+1,j) - C(i-1,j)}{2\,\Delta x}\right] + D_x\left[\frac{C(i+1,j) + C(i-1,j)}{(\Delta x)^2}\right]\right.$$
$$\left. + D_y\left[\frac{C(i,j+1) + C(i,j-1)}{(\Delta y)^2}\right]\right\rangle \Big/ \left\{\frac{2D_x}{(\Delta x)^2} + \frac{2D_y}{(\Delta y)^2} + k\right\} \tag{6.25}$$

For the example system above, the boundary conditions again determine known concentrations for the first column and the bottom row. Similarly, image nodes are added above the top row and beyond the last column with appropriately fixed concentrations

to implement the no-flux assumptions. This leaves a set of $25 (= 36 - 11)$ simultaneous linear algebraic equations that must be solved *once*. In general, steady-state problems using finite difference or finite cell methods require solution of a set of simultaneous equations once. Dynamic models with explicit solution methods require (the much simpler) solution of a sequence of independent equations at each time step, while dynamic models with implicit methods require solution of a set of simultaneous equations at each time step. We next illustrate a particular set of iterative methods that are especially useful when solving "sparse" systems of equations of the type considered thus far.

Iterative Solution Methods for Simultaneous Linear Algebraic Equations In the example above, a set of 25 simultaneous equations must be solved at each time step to implement a (partially) implicit solution, or once to determine the steady-state solution. While matrix inversion or related procedures can be used to accomplish this, this problem is especially suited to solution using iterative techniques. These techniques are relatively easy to implement when the problem matrix is sparse, as occurs when the mass-balance equation for a node includes concentrations at only a few other system nodes: typically the upstream and downstream nodes for a one-dimensional problem, the four surrounding nodes for a two-dimensional problem, or the six surrounding nodes for a three-dimensional problem.

The first step in an iterative solution is to isolate the targeted unknown state variables on the lhs of each equation, with one equation for each of the unknowns. Thus, for the dynamic model with implicit solution described above, Eq. (6.22) is rewritten for each of the 25 nodes by moving all terms involving $C(i, j, n+1)$ to the lhs of the equation:

$$C(i,j,n+1)\left\{1 + \Delta t\,\alpha\left[\frac{2D_x}{(\Delta x)^2} + \frac{2D_y}{(\Delta y)^2} + k\right]\right\} = C(i,j,n)$$
$$+ \Delta t\,\alpha\left\{\begin{array}{l} -u\left[\dfrac{C(i+1,j,n+1) - C(i-1,j,n+1)}{2\,\Delta x}\right] \\ +D_x\left[\dfrac{C(i+1,j,n+1) + C(i-1,j,n+1)}{(\Delta x)^2}\right] \\ +D_y\left[\dfrac{C(i,j+1,n+1) + C(i,j-1,n+1)}{(\Delta y)^2}\right] \end{array}\right\}$$
$$+ \Delta t(1-\alpha)\left\{\begin{array}{l} -u\left[\dfrac{C(i+1,j,n) - C(i-1,j,n)}{2\,\Delta x}\right] \\ +D_x\left[\dfrac{C(i+1,j,n) - 2C(i,j,n) + C(i-1,j,n)}{(\Delta x)^2}\right] \\ +D_y\left[\dfrac{C(i,j+1,n) - 2C(i,j,n) + C(i,j-1,n)}{(\Delta y)^2}\right] - kC(i,j,n) \end{array}\right\}$$

Dividing both sides of the equation by

$$\left\{1 + \Delta t\,\alpha\left[\frac{2D_x}{(\Delta x)^2} + \frac{2Dy}{(\Delta y)^2} + k\right]\right\}$$

yields the targeted expression, with $C(i, j, n+1)$ only found on the lhs:

$$C(i,j,n+1) = \left[\frac{1}{1+\Delta t\,\alpha\left(\frac{2D_x}{(\Delta x)^2}+\frac{2D_y}{(\Delta y)^2}+k\right)}\right] \times \left((C(i,j,n) + \Delta t\,\alpha \left\{ \begin{array}{l} -u\left[\dfrac{C(i+1,j,n+1) - C(i-1,j,n+1)}{2\,\Delta x}\right] \\ +D_x\left[\dfrac{C(i+1,j,n+1) + C(i-1,j,n+1)}{(\Delta x)^2}\right] \\ +D_y\left[\dfrac{C(i,j+1,n+1) + C(i,j-1,n+1)}{(\Delta y)^2}\right] \end{array} \right\} \right. + \Delta t(1-\alpha) \left\{ \begin{array}{l} -u\left[\dfrac{C(i+1,j,n) - C(i-1,j,n)}{2\,\Delta x}\right] \\ +D_x\left[\dfrac{C(i+1,j,n) - 2C(i,j,n) + C(i-1,j,n)}{(\Delta x)^2}\right] \\ +D_y\left[\dfrac{C(i,j+1,n) - 2C(i,j,n) + C(i,j-1,n)}{(\Delta y)^2}\right] - kC(i,j,n) \end{array} \right\} \left. \right) \tag{6.26}$$

Equation (6.26) looks daunting but is easy to implement in the iterative solution framework. To do this, the full set of 25 equations corresponding to each unknown value of $C(i, j, n+1)$ is written, an initial guess for each of the $C(i, j, n+1)$ is made, and the equations are solved repeatedly over many iterations until convergence is achieved. That is, beginning with an initial guess, $C(i, j, n+1)^0$ for each of the 25 nodes where solution is required, values of $C(i, j, n+1)^1$ are computed, values of $C(i, j, n+1)^2$ computed from these, and eventually $C(i, j, n+1)^{m+1}$ computed from the $C(i, j, n+1)^m$ until:

$$C(i,j,n+1)^{m+1} - C(i,j,n+1)^m < \varepsilon \tag{6.27}$$

where ε is a very small value chosen to test for convergence. When the convergence criterion is satisfied at all nodes, the values of the target concentrations at iteration number m (or $m+1$) are accepted and used to move forward to the next time step.

Within each iteration, the calculation proceeds across the grid, and Eq. (6.26) is evaluated for all of the target nodes. At iteration m, the unknown concentration values

at time $n+1$ on the rhs of Eq. (6.26) [$C(i-1, j, n+1)$, $C(i+1, j, n+1)$, $C(i, j-1, n+1)$ and $C(i, j+1, n+1)$] can all be evaluated using their values from iteration $m-1$:

$$C(i,j,n+1)^m = \left[\frac{1}{1+\Delta t\,\alpha\left(\frac{2D_x}{(\Delta x)^2}+\frac{2D_y}{(\Delta y)^2}+k\right)}\right] \times \left((C(i,j,n) + \Delta t\,\alpha \left\{ \begin{array}{l} -u\left[\dfrac{C(i+1,j,n+1)^{m-1} - C(i-1,j,n+1)^{m-1}}{2\,\Delta x}\right] \\ +D_x\left[\dfrac{C(i+1,j,n+1)^{m-1} + C(i-1,j,n+1)^{m-1}}{(\Delta x)^2}\right] \\ +D_y\left[\dfrac{C(i,j+1,n+1)^{m-1} + C(i,j-1,n+1)^{m-1}}{(\Delta y)^2}\right] \end{array} \right\} \right. + \Delta t(1-\alpha) \times \left\{ \begin{array}{l} -u\left[\dfrac{C(i+1,j,n) - C(i-1,j,n)}{2\,\Delta x}\right] \\ +D_x\left[\dfrac{C(i+1,j,n) - 2C(i,j,n) + C(i-1,j,n)}{(\Delta x)^2}\right] \\ +D_y\left[\dfrac{C(i,j+1,n) - 2C(i,j,n) + C(i,j-1,n)}{(\Delta y)^2}\right] - kC(i,j,n) \end{array} \right\} \left. \right) \tag{6.28}$$

This calculation, known as *Jacobi iteration*, can proceed in any order through the grid system since all the unknown concentrations at iteration $m-1$ are computed, stored, and used in the calculation for iteration m. An alternative approach, the *Gauss–Seidel* iteration, uses unknown concentrations that have already been calculated in iteration m for the calculation of subsequent unknown concentrations during the same iteration. To envision this, imagine sequentially sweeping across each row from left to right, beginning each iteration at the upper left-hand corner (at $i = 1$, $j = 1$), moving across the first row until it is completed, then moving to the beginning of the second row ($i = 1$, $j = 2$), continuing to the right, and so on, until the full set of unknown nodes is evaluated. With this order of calculation, values of $C(i-1, j, n+1)^m$ (from the column to the left) and $C(i, j-1, n+1)^m$ (from the row above) will already be available when it is time to compute $C(i, j, n+1)^m$. Why not use them in Eq. (6.26) instead of the old values of $C(i-1, j, n+1)^{m-1}$ and $C(i, j-1, n+1)^{m-1}$ from the previous iteration? If (as we hope!) each iteration brings the concentration estimates closer to their correct values, the latter values should be more accurate and allow for more rapid convergence to the correct values. Gauss–Seidel iteration does indeed allow for more rapid convergence. It has the further advantage that it is *easier* to program since values of the unknown concentrations from the previous iteration no longer need to be stored once $C(i-1, j, n+1)^m$ and $C(i, j-1, n+1)^m$ are computed

and their tests for convergence are implemented. The resulting Gauss–Seidel iterative equation is given by:

$$C(i,j,n+1)^m = \left[\frac{1}{1+\Delta t\,\alpha\left(\frac{2D_x}{(\Delta x)^2}+\frac{2D_y}{(\Delta y)^2}+k\right)}\right] \times \left((C(i,j,n)+\Delta t\,\alpha \left\{ \begin{array}{l} -u\left[\frac{C(i+1,j,n+1)^{m-1}-C(i-1,j,n+1)^m}{2\,\Delta x}\right] \\ +D_x\left[\frac{C(i+1,j,n+1)^{m-1}+C(i-1,j,n+1)^m}{(\Delta x)^2}\right] \\ +D_y\left[\frac{C(i,j+1,n+1)^{m-1}+C(i,j-1,n+1)^m}{(\Delta y)^2}\right] \end{array} \right\} \right. \\ \left. + \Delta t(1-\alpha)\left\{ \begin{array}{l} -u\left[\frac{C(i+1,j,n)-C(i-1,j,n)}{2\,\Delta x}\right] \\ +D_x\left[\frac{C(i+1,j,n)-2C(i,j,n)+C(i-1,j,n)}{(\Delta x)^2}\right] \\ +D_y\left[\frac{C(i,j+1,n)-2C(i,j,n)+C(i,j-1,n)}{(\Delta y)^2}\right]-kC(i,j,n) \end{array} \right\} \right) \tag{6.29}$$

This same approach can be used to solve the steady-state model equations. To do this, note that the working equation has already been modified to isolate $C(i, j)$ on the lhs [see Eq. 6.25)]. Equation (6.25) is then written for Gauss–Seidel iteration as follows:

$$C(i,j)^m = \left\langle [-u]\left[\frac{C(i+1,j)^{m-1}-C(i-1,j)^m}{2\,\Delta x}\right] + D_x\left[\frac{C(i+1,j)^{m-1}+C(i-1,j)^m}{(\Delta x)^2}\right] + D_y\left[\frac{C(i,j+1)^{m-1}+C(i,j-1)^m}{(\Delta y)^2}\right]\right\rangle \Big/ \left\{\frac{2D_x}{(\Delta x)^2}+\frac{2D_y}{(\Delta y)^2}+k\right\} \tag{6.30}$$

With any of the iterative solution methods, initial guesses are required for each of the unknowns. For the steady-state model, these may be estimated by interpolating between boundary conditions, if they are known. For the dynamic model with implicit solution, the initial guesses for each time step may be set to the computed values from the previous time step, or to estimates for the new time step computed using the explicit solution [e.g., Eq. 6.16)].

Other techniques, such as *successive over relaxation (SOR)*, are available to further speed the rate of conversion of an iterative solution (Hoffman, 1992, p. 56; Wang and Anderson, 1982, p. 27). With SOR, the change that occurs with each iteration is extended by first noting the change, multiplying it by a factor ω (generally between 1.0 and 2.0), and adding the product to the previous iteration. A two-step procedure (similar to a predictor–corrector method) thus results. For example, for modification of the Gauss–Seidel solution of the steady-state model [Eq. (6.30)], SOR is implemented by first computing a "predictor" value of $C(i, j)^{m*}$ using Eq. (6.30), then computing the "corrector" as:

$$C(i,j)^m = C(i,j)^{m-1} + \omega\left[C(i,j)^{m*} - C(i,j)^{m-1}\right] \tag{6.31}$$

Though convergence is generally faster with SOR, problems with overshooting the correct answer can occur, causing oscillation around the correct answer. A method that utilizes decreasing values of ω (decreasing toward 1.0) as m increases can be used if this problem occurs. In most situations it is safest to stick with the Gauss–Seidel method, unless the computation time resulting from the iterative solution is excessively large (as may be the case when, as with the implicit dynamic solution, the iteration is required at each time step).

6.2.2 Finite Element Methods

A second numerical approach for solving partial differential equations is the use of finite element methods. Finite element methods originated in the field of solid mechanics and are the standard methods for solving structural analysis problems. They are less widely used than finite difference methods for fluid mechanics and heat and mass transfer but offer significant advantages for some applications. Finite element methods are especially useful for problems with irregular geometry, inhomogeneous properties, and complicated loadings. It is relatively easy to vary the size of the elements over the model domain, whereas grid cell size variations can introduce significant complications with finite differences. Finite element methods are also more accurate and stable than finite difference methods for comparable element and grid cell sizes.

As discussed in the previous section, finite difference methods are developed by replacing spatial or temporal derivatives in a partial differential equation with differences defined between nodes on a regular rectangular space–time grid. Finite element methods also start with a discretized domain and produce algebraic equations from differential equations. However, with finite elements, the value of the state variable of interest across a small element of the domain is approximated by an interpolating function, which is usually a polynomial. Elements are regular polygons in one, two, or three dimensions. Values at the vertices or nodes of each element are determined to optimally satisfy the partial differential equations that apply to the system. The solution is assembled over the full domain by requiring that values of

the state variables along the edges of adjacent elements match and that boundary conditions be satisfied.

A description of the main steps required in finite element analysis, with an illustration for the 1D steady-state advection–diffusion equation, is found in the supplemental information for this chapter at www.wiley.com/college/ramaswami. More in-depth presentations of finite element methods are provided in textbooks dedicated to the subject (e.g., Desai, 1979; Wait and Mitchell, 1985; and Zienkiewicz and Taylor, 2000). Bear (1979), Wang and Anderson (1982), and Zheng and Bennett (1995) provide accessible introductions to finite elements in the context of groundwater flow and contaminant transport modeling; more in-depth presentations are found in (Guyman (1970), Guyman et al. (1970), Pinder and Gray (1977), Huyakorn and Pinder (1983), and Carey (1995).

6.3 SOLUTIONS TO NONLINEAR SYSTEMS OF EQUATIONS

Section 6.2.1 described the Jacobi and Gauss–Siedel methods for solving systems of linear algebraic equations, as employed for numerical solution of the finite difference representation of advective–dispersive transport phenomena. Indeed, these and other matrix inversion techniques are the basic numerical methods for simultaneously solving any general system of linear equations. However, linear relationships are often not adequate to represent real-world systems, particularly when chemical reactions are involved. Consideration of chemical equilibria, dependence of equilibrium constants and rate constants on temperature (often an exponential relationship), and inclusion of chemical kinetics of orders greater than 1, introduce a high degree of nonlinearity into the relationships between different variables in physicochemical systems. This section describes numerical methods used to solve the systems of nonlinear algebraic equations that may apply to these problems. As always, for any solution technique to work, the number of equations must be equal to the number of variables in the system. Typically, the variables in a system may include the concentrations of various chemical species involved in reactions, ambient temperature, pressure, and so forth. Kinetic and equilibrium relationships and charge and mass balances provide the framework for the nonlinear equations relating these variables to each other. At a single time step, the multispecies nonlinear system of equations is solved using the techniques described below.

Before addressing multispecies systems of equations, it is useful to first consider the Newton–Raphson method for solving a nonlinear equation involving just a single variable. Consider a polynomial equation, $f(x)$, of order greater than 1. Solving this equation involves finding values of x at which $f(x)$ will go to 0, that is, $f(x) = 0$. We do not know the roots of this equation a priori, and use an iterative scheme to approach the roots from an initial guess, x_0. If x were the real root of the equation, $f(x)$ would be exactly 0. In addition, if the initial guess, x_0, was sufficiently close to the real root, x, $f(x)$ could be computed from the first derivative $f'(x)$ evaluated at $x_0[=f'(x_0)]$ as shown below, and set to 0:

$$f(x) = f(x_0) + f'(x_0) \times [x - x_0] = 0 \tag{6.32}$$

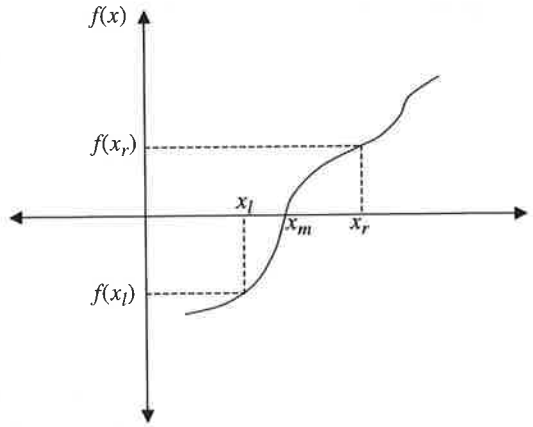
which, upon rearrangement, yields

$$x = x_0 - \frac{f(x_0)}{f'(x_0)} \tag{6.33}$$

In reality, since we have no a priori knowledge of the value of the roots of the equation, Eq. (6.33) is used iteratively to converge on the roots of the equation $f(x) = 0$, starting with an initial guess, x_0. If we specify each iteration number as m:

$$x_{m+1} = x_m - \frac{f(x_m)}{f'(x_m)} \tag{6.34}$$

Equation (6.34) is called Newton's method or the Newton–Raphson method for finding the roots of nonlinear equations. The method rapidly converges to the solution but is computationally expensive since the exact derivative $[f'(x)]$ needs to be first determined analytically and then evaluated at each iteration. Note that rearrangement of Eq. (6.34) yields the discrete definition of the slope $f'(x)$ when $f(x) = 0$, that is, $f'(x_m) = [0 - f(x_m)]/[x_{m+1} - x_m]$. The Newton–Raphson method is closely related to other slope-type iteration methods such as the bisection method and the method of Regula Falsi, as illustrated in Figure 6.8.



Bisection Method: Slope $= \left[\dfrac{0 - f(x_l)}{x_m - x_l}\right] = \left[\dfrac{f(x_r) - f(x_l)}{x_r - x_l}\right]$

Newton's Method: Slope $= f'(x) = \left[\dfrac{0 - f(x)}{x_m - x}\right]$ for any x near the root

Figure 6.8 Using the concept of slope in the bisection method where the root, x_m, is converged on from a point x_r to the right and a point x_l to the left, and in the Newton–Raphson method where the derivative function $f'(x)$ is used to converge toward the real root $x_0 = x_m$.

Problems that can arise with the Newton–Raphson method include the presence of an inflection point in the function near the location of the root, which can cause divergence from the real solution. The solution may oscillate around a local minimum or maximum in the function if one is encountered on the way to the root. The presence of multiple roots in the interval within which iterations are being commenced can also cause masking of one root by another, that is, depending upon the choice of the initial guess x_0, the solution may converge consistently toward one root, missing the other solution.

Next consider a set of n equations, F_1, F_2, F_3, ... F_n, involving n variables, x_1, x_2, x_3, ..., x_n. The set of equations may be written as:

$$\begin{aligned} F_1(x_1, x_2, x_3, \ldots, x_n) &= 0 \\ F_2(x_1, x_2, x_3, \ldots, x_n) &= 0 \\ F_3(x_1, x_2, x_3, \ldots, x_n) &= 0 \\ &\vdots \\ F_n(x_1, x_2, x_3, \ldots, x_n) &= 0 \end{aligned} \tag{6.35}$$

This set of equations is solved when the values of $(x_1, x_2, x_3, \ldots, x_n)$ simultaneously satisfy $F_n = 0$, for all n shown in Eq. (6.35). If our initial guesses of the solution set are $(x_{1,0}, x_{2,0}, x_{3,0}, \ldots, x_{n,0})$, an expression analogous to Eq. (6.32) for the multivariate problem specified in Eq. (6.35) can be written as:

$$\begin{aligned} F_1(x_1, x_2, x_3, \ldots, x_n) &= F_1(x_{1,0}, x_{2,0}, x_{3,0}, \ldots, x_{n,0}) + \frac{\partial F_1}{\partial x_1}\delta x_1 + \frac{\partial F_1}{\partial x_2}\delta x_2 + \ldots \frac{\partial F_1}{\partial x_n}\delta x_n = 0 \\ F_2(x_1, x_2, x_3, \ldots, x_n) &= F_2(x_{1,0}, x_{2,0}, x_{3,0}, \ldots, x_{n,0}) + \frac{\partial F_2}{\partial x_1}\delta x_1 + \frac{\partial F_2}{\partial x_2}\delta x_2 + \ldots \frac{\partial F_2}{\partial x_n}\delta x_n = 0 \\ F_3(x_1, x_2, x_3, \ldots, x_n) &= F_3(x_{1,0}, x_{2,0}, x_{3,0}, \ldots, x_{n,0}) + \frac{\partial F_3}{\partial x_1}\delta x_1 + \frac{\partial F_3}{\partial x_2}\delta x_2 + \ldots \frac{\partial F_3}{\partial x_n}\delta x_n = 0 \\ &\vdots \\ F_n(x_1, x_2, x_3, \ldots, x_n) &= F_n(x_{1,0}, x_{2,0}, x_{3,0}, \ldots, x_{n,0}) + \frac{\partial F_n}{\partial x_1}\delta x_1 + \frac{\partial F_n}{\partial x_2}\delta x_2 + \ldots \frac{\partial F_n}{\partial x_n}\delta x_n = 0 \end{aligned} \tag{6.36}$$

In Eq. (6.36), $\delta x_i = (x_i - x_{i,0})$. Note the similarity between the system of n equations in Eqs. (6.36) and (6.32), which was written for a single variable. As above, our first guess may not be very good, so multiple iterations may be required. In matrix notation, the expansion equation for iteration $m + 1$ may be written as follows:

$$\mathbf{F}(\mathbf{x}_{m+1}) = \mathbf{F}(\mathbf{x} + \boldsymbol{\delta}\mathbf{x}) = \mathbf{F}(\mathbf{x}_m) + J(\mathbf{x}_m)\boldsymbol{\delta}\mathbf{x} = \mathbf{0} \tag{6.37}$$

where $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}^T$, $\mathbf{F} = \{F_1, F_2, \ldots, F_n\}^T$, and $\boldsymbol{\delta}\mathbf{x} = \{\delta x_1, \delta x_2, \ldots, \delta x_n\}^T$, with $\delta x_i = (x_{m+1} - x_{i,m})$. The matrix J in Eq. (6.37) is known as the Jacobian matrix, and is defined as:

$$J(x_m) = \begin{bmatrix} \left[\frac{\partial F_1}{\partial x_1}\right] & \left[\frac{\partial F_1}{\partial x_2}\right] & \cdots & \left[\frac{\partial F_1}{\partial x_n}\right] \\ \left[\frac{\partial F_2}{\partial x_1}\right] & \left[\frac{\partial F_2}{\partial x_2}\right] & \cdots & \left[\frac{\partial F_2}{\partial x_n}\right] \\ \cdots & \cdots & \cdots & \cdots \\ \left[\frac{\partial F_n}{\partial x_1}\right] & \left[\frac{\partial F_n}{\partial x_2}\right] & \cdots & \left[\frac{\partial F_n}{\partial x_n}\right] \end{bmatrix} \tag{6.38}$$

The partial derivatives in this matrix are evaluated using values from the mth iteration, x_m. With $\mathbf{F}(\mathbf{x}_{m+1}) = \mathbf{0}$, Eq. (6.37) can be rearranged into the standard form for a system of linear algebraic equations:

$$J(x_m)[\boldsymbol{\delta}\mathbf{x}_m] = -\mathbf{F}(\mathbf{x}_m) \tag{6.39}$$

Rearrangement of Eq. (6.57) yields

$$\mathbf{x}_{m+1} = \mathbf{x}_m - [J(\mathbf{x}_m)]^{-1}\,\mathbf{F}(\mathbf{x}_m) \tag{6.40}$$

Once again, note the similarities between the iteration algorithm for Newton's method for a single variable shown in Eq. (6.34) and that represented in matrix form in Eq. (6.40) for the multivariate problem. While the matrix representation for multivariate nonlinear systems of equations provides convergence when certain matrix constraints are met, the evaluation and inversion of the Jacobian matrix can be difficult. Instead, a modified Newton–Raphson method is used for faster computations. In this method, the partial derivative matrix is simplified and approximated by using only the diagonal terms of the matrix shown in Eq. (6.38). Thus, the iteration algorithm becomes

$$\begin{aligned} x_{1,m+1} &= x_{1,m} - \frac{F_1(x_1, x_2, \ldots, x_n)_m}{\left[\frac{\partial F_1}{\partial x_1}\right]_{(X_1, X_2, \ldots, X_n)_m}} \\ &\vdots \\ x_{n,m+1} &= x_{n,m} - \frac{F_n(x_1, x_2, \ldots, x_n)_m}{\left[\frac{\partial F_n}{\partial x_n}\right]_{(X_1, X_2, \ldots, X_n)_m}} \end{aligned} \tag{6.41}$$

Example 6.3 illustrates the use of the modified Newton–Raphson method for a two-variable problem, mimicking (for a relatively simple case) some of the chemical equilibrium equations that are presented in Chapters 2 and 12.

EXAMPLE 6.3

Consider a reversible reaction in which a chemical species A1 is transformed into another species A2, with equilibrium constant $K = 2$. By stoichiometry, it is given that 2 [A1] $\Leftrightarrow$ [A2]. The total molar concentration of A introduced into the system is 1 mol/L, initially introduced as A1. Hence the two relevant equations linking the two variables A1 and A2 are

$$K = \frac{[A2]}{[A1]^2} = 2 \qquad \text{Equilibrium relationship}$$

$$[A1] + 2[A2] = 1 \qquad \text{Mass-balance relationship}$$

Substituting the first equation ($[A2] = 2[A1]^2$) into the second one, and solving the resulting quadratic equation, we get the exact solution (to the sixth decimal place!) as: A1 = 0.390388 and A2 = 0.304806. This indicates that, of 1 mol of A1 introduced into a 1-L reactor, 0.61 mol are converted to A2, yielding 0.305 mol of A2 (due to stoichiometry) with 0.39 mol of A1 remaining in solution.

Now, we shall see if the exact solution can be found by the modified Newton–Raphson iteration method. To be consistent with the notation used in the discussion of theory, we see that:

$$F_1(A1, A2) = -2[A1]^2 + [A2] = 0 \qquad \partial F_1/\partial A1 = -4[A1]$$

$$F_2(A1, A2) = [A1] + 2[A2] - 1 = 0 \qquad \partial F_2/\partial A2 = 2$$

We can begin with an initial guess that A1 = 1 and A2 = 0, that is, the situation at the point when the reaction had not yet commenced. The first 27 iterations are shown below:

Iteration	A1	A2	F_1	F_2	$\partial F_1/\partial A1$	$\partial F_2/\partial A2$
1	1	0	−2	0	−4	2
2	0.5	0	−0.5	−0.5	−2	2
3	0.25	0.25	0.125	−0.25	−1	2
4	0.375	0.375	0.09375	0.125	−1.5	2
5	0.4375	0.3125	−0.07031	0.0625	−1.75	2
6	0.397321	0.28125	−0.03448	−0.04018	−1.58929	2
7	0.375627	0.301339	0.019148	−0.02169	−1.50251	2
8	0.388371	0.312186	0.010522	0.012744	−1.55348	2
9	0.395144	0.305814	−0.00646	0.006773	−1.58058	2
10	0.391055	0.302428	−0.00342	−0.00409	−1.56422	2
11	0.388868	0.304473	0.002035	−0.00219	−1.55547	2

(*continued*)

Iteration	A1	A2	F_1	F_2	$\partial F_1/\partial A1$	$\partial F_2/\partial A2$
12	0.390177	0.305566	0.00109	0.001308	−1.56071	2
13	0.390875	0.304912	−0.00066	0.000698	−1.5635	2
14	0.390456	0.304562	−0.00035	−0.00042	−1.56182	2
15	0.390232	0.304772	0.000209	−0.00022	−1.56093	2
16	0.390366	0.304884	0.000112	0.000134	−1.56147	2
17	0.390438	0.304817	−6.7E-05	7.16E-05	−1.56175	2
18	0.390395	0.304781	−3.6E-05	−4.3E-05	−1.56158	2
19	0.390372	0.304802	2.15E-05	−2.3E-05	−1.56149	2
20	0.390386	0.304814	1.15E-05	1.38E-05	−1.56154	2
21	0.390393	0.304807	−6.9E-06	7.34E-06	−1.56157	2
22	0.390389	0.304803	−3.7E-06	−4.4E-06	−1.56156	2
23	0.390387	0.304806	2.2E-06	−2.4E-06	−1.56155	2
24	0.390388	0.304807	1.18E-06	1.41E-06	−1.56155	2
25	0.390389	0.304806	−7.1E-07	7.53E-07	−1.56155	2
26	0.390388	0.304806	−3.8E-07	−4.5E-07	−1.56155	2
27	0.390388	0.304806	2.26E-07	−2.4E-07	−1.56155	2

Notice that the solution converges to the exact solution within the third decimal place (A1 = 0.390 ± 0.001) by the 12th iteration, and to the sixth decimal place by the 26nd iteration. Also note how the iterative solutions oscillate above and below the true solution in consecutive iterations. This is a feature of the slope technique, in which we are approaching the true solution from the right, then from the left, and so on, as also occurs in the bisection method (Fig. 6.8) for a single variable nonlinear equation. The solutions obtained with the full Jacobian matrix would converge with fewer iterations, but with more intensive matrix inversions that would become increasingly demanding as the number of equations is increased.

Chapter 12 presents applications of nonlinear solution techniques to determine chemical equilibria involving multiple chemical species participating in acid–base dissociation, dissolution–precipitation, oxidation–reduction, and surface complexation reactions. Computer packages such as MICROQL (Westall, 1986) and MINTEQ (Allison et al., 1991) implement techniques such as the Newton–Raphson method and are readily available for solution of a wide array of nonlinear equations involving multispecies chemical equilibria in aqueous systems.

6.4 SUMMARY

Methods for numerically solving four classes of mathematical models have been described in this chapter. Numerical integration techniques, such as the Euler–Cauchy method, the predictor–corrector method, and the Runge–Kutta method are described in Section 6.1 and can be used to solve chemical mass-balances ODEs with concentration represented as a function of a single variable, typically time. Initial concentrations in the system must be known to initiate these integration techniques.

Methods for solving partial differential equations, applicable in solving mass-balance differential equations over time and multidimensional space, were presented in Section 6.2. Section 6.2.1 focused on finite difference methods for solving PDEs, formulating forward differencing, backward differencing, and central differencing schemes for the spatial derivative, and implicit and explicit methods to step forward in time. Finite element methods for solving PDEs were introduced in Section 6.2.2. Both techniques for solving PDEs require the statement of initial (temporal) and (spatial) boundary conditions to initiate the iterations. Techniques for simultaneously solving a set of linear algebraic equations are presented in Section 6.2.1 as tools for implementing finite difference schemes. Solution techniques that address systems of nonlinear algebraic equations are presented in Section 6.3 and are typically used to model complex chemical reactions with multispecies equilibria or higher-order kinetics.

The numerical methods briefly presented here form the foundation of many computer codes and simulation packages designed to model contaminant transport and fate in the environment. In the applications presented in this chapter, all the input parameters to the models are assumed to be "single valued," that is, they have fixed, or "point-estimate" values assumed to be known a priori by the user. Single-valued, point-estimate input parameter values, while easy to incorporate into models, are not often representative of our understanding of the real world. In particular, modelers must also consider systematic (seasonal or spatial) variability in model parameter values, random fluctuations in these values with no known systematic underlying cause, as well as uncertainty in model parameters associated with a lack of a priori knowledge of the system. To address this need, Chapter 7 presents probabilistic techniques for the incorporation of random variables and random processes into environmental models.

7 Overview of Probabilistic Methods and Tools for Modeling

Environmental systems are highly variable in their properties and response to inputs. Furthermore, there is a great deal of uncertainty about these properties, future inputs, and responses. In this chapter we introduce the basic tools of probability used to model variable and uncertain environmental systems.

Variability refers to the inherent differences in environmental properties that occur over space and time and from one sample to another (e.g., the differences in exposure, susceptibility, and risk that occur between one individual and another in a target population). *Uncertainty* reflects a lack of knowledge of environmental processes and properties. While many of the same tools of probability and statistics can be applied to characterize variability and uncertainty, the need to carefully distinguish between them is widely recognized (e.g., Bogen and Spear, 1987; Burmaster and Wilson, 1996; Cullen and Frey, 1999), and we are careful to do so in the applications that follow.

In Section 1.5.1 we characterized deterministic models as those that calculate a single value for each model output, in contrast to stochastic models that produce a distribution of values for each prediction. While probabilistic methods provide the basic building blocks for stochastic models, the distinction between deterministic and stochastic models is not always clear-cut. For example, the random motions of fluid elements described in Sections 5.3 and 5.5 that lead to Fickian and non-Fickian dispersion are typically aggregated over many fluid elements and treated as deterministic processes at the continuum scale. Similarly, chemical transformations that are stochastic at the scale of individual particles and molecules are usually aggregated, using kinetic models to provide deterministic representations of bulk reaction processes. Individual chemical and fluid elements can be statistically simulated to yield the same results as deterministic process models for transport and reaction; this is the basis for the *population balance* method that tracks discrete pollutant particles through the environment (Patterson et al., 1981; Koch and Prickett, 1993; Visser, 1997).

Probability models describe the likelihood, or probability, of different outcomes or events. A *random variable* is a quantity that can take on different values for a variety of reasons, but with no specific mechanism or underlying cause that allows an outcome to be predicted with certainty. Instead, the variable is described by a *probability distribution function*. When the quantity is *ordered* in space and/or time, the probability model must also describe the nature of this ordering. Such quantities are referred to as *random processes*. We begin this chapter by describing models for