1

Initial-Value Problems for Ordinary Differential Equations

INTRODUCTION

The goal of this book is to expose the reader to modern computational tools for solving differential equation models that arise in chemical engineering, e.g., diffusion-reaction, mass-heat transfer, and fluid flow. The emphasis is placed on the understanding and proper use of software packages. In each chapter we outline numerical techniques that either illustrate a computational property of interest or are the underlying methods of a computer package. At the close of each chapter a survey of computer packages is accompanied by examples of their use.

BACKGROUND

Many problems in engineering and science can be formulated in terms of differential equations. A differential equation is an equation involving a relation between an unknown function and one or more of its derivatives. Equations involving derivatives of only one independent variable are called ordinary differential equations and may be classified as either initial-value problems (IVP) or boundary-value problems (BVP). Examples of the two types are:

$$\text{IVP:} \quad y'' = -yx \tag{1.1a}$$

$$y(0) = 2, \qquad y'(0) = 1 \tag{1.1b}$$

$$\text{BVP:} \quad y'' = -yx \tag{1.2a}$$

$$y(0) = 2, \qquad y(1) = 1 \tag{1.2b}$$

where the prime denotes differentiation with respect to x. The distinction between the two classifications lies in the location where the extra conditions [Eqs. (1.1b) and (1.2b)] are specified. For an IVP, the conditions are given at the same value of x, whereas in the case of the BVP, they are prescribed at two different values of x.

Since there are relatively few differential equations arising from practical problems for which analytical solutions are known, one must resort to numerical methods. In this situation it turns out that the numerical methods for each type of problem, IVP or BVP, are quite different and require separate treatment. In this chapter we discuss IVPs, leaving BVPs to Chapters 2 and 3.

Consider the problem of solving the mth-order differential equation

$$y^{(m)} = f(x, y, y', y'', \ldots, y^{(m-1)}) \tag{1.3}$$

with initial conditions

$$\begin{aligned} y(x_0) &= y_0 \\ y'(x_0) &= y_0' \\ &\vdots \\ y^{(m-1)}(x_0) &= y_0^{(m-1)} \end{aligned}$$

where f is a known function and $y_0, y_0', \ldots, y_0^{(m-1)}$ are constants. It is customary to rewrite (1.3) as an equivalent system of m first-order equations. To do so, we define a new set of dependent variables $y_1(x), y_2(x), \ldots, y_m(x)$ by

$$\begin{aligned} y_1 &= y \\ y_2 &= y' \\ y_3 &= y'' \\ &\vdots \\ y_m &= y^{(m-1)} \end{aligned} \tag{1.4}$$

and transform (1.3) into

$$\begin{aligned} y_1' &= y_2 && = f_1(x, y_1, y_2, \ldots, y_m) \\ y_2' &= y_3 && = f_2(x, y_1, y_2, \ldots, y_m) \\ &\vdots && \vdots \\ y_m' &= f(x, y_1, y_2, \ldots, y_m) && = f_m(x, y_1, y_2, \ldots, y_m) \end{aligned} \tag{1.5}$$

with

$$\begin{aligned} y_1(x_0) &= y_0 \\ y_2(x_0) &= y_0' \\ &\vdots \\ y_m(x_0) &= y_0^{(m-1)} \end{aligned}$$

In vector notation (1.5) becomes

$$\begin{aligned} \mathbf{y}'(x) &= \mathbf{f}(x, \mathbf{y}) \\ \mathbf{y}(x_0) &= \mathbf{y}_0 \end{aligned} \tag{1.6}$$

where

$$\mathbf{y}(x) = \begin{bmatrix} y_1(x) \\ y_2(x) \\ \vdots \\ y_m(x) \end{bmatrix}, \qquad \mathbf{f}(x, \mathbf{y}) = \begin{bmatrix} f_1(x, \mathbf{y}) \\ f_2(x, \mathbf{y}) \\ \vdots \\ f_m(x, \mathbf{y}) \end{bmatrix}, \qquad \mathbf{y}_0 = \begin{bmatrix} y_0 \\ y_0' \\ \vdots \\ y_0^{(m-1)} \end{bmatrix}$$

It is easy to see that (1.6) can represent either an mth-order differential equation, a system of equations of mixed order but with total order of m, or a system of m first-order equations. In general, subroutines for solving IVPs assume that the problem is in the form (1.6). In order to simplify the analysis, we begin by examining a single first-order IVP, after which we extend the discussion to include systems of the form (1.6).

Consider the initial-value problem

$$\begin{aligned} y' &= f(x, y), \qquad y(x_0) = y_0 \\ x_0 &\leq x \leq x_N \end{aligned} \tag{1.7}$$

We assume that $\partial f/\partial y$ is continuous on the strip $x_0 \leq x \leq x_N$, thus guaranteeing that (1.7) possesses a unique solution [1]. If $y(x)$ is the exact solution to (1.7), its graph is a curve in the xy-plane passing through the point (x_0, y_0). A discrete numerical solution of (1.7) is defined to be a set of points $[(x_i, u_i)]_{i=0}^{N}$, where $u_0 = y_0$ and each point (x_i, u_i) is an approximation to the corresponding point $(x_i, y(x_i))$ on the solution curve. Note that the numerical solution is only a set of points, and nothing is said about values between the points. In the remainder of this chapter we describe various methods for obtaining a numerical solution $[(x_i, u_i)]_{i=0}^{N}$.

EXPLICIT METHODS

We again consider (1.7) as the model differential equation and begin by dividing the interval $[x_0, x_N]$ into N equally spaced subintervals such that

$$\begin{aligned} h &= \frac{x_N - x_0}{N} \\ x_i &= x_0 + ih, \qquad i = 0, 1, 2, \ldots, N \end{aligned} \tag{1.8}$$

The parameter h is called the step-size and does not necessarily have to be uniform over the interval. (Variable step-sizes are considered later.)

If $y(x)$ is the exact solution of (1.7), then by expanding $y(x)$ about the point x_i using Taylor's theorem with remainder we obtain:

$$y(x_{i+1}) = y(x_i) + (x_{i+1} - x_i)y'(x_i) + \frac{(x_{i+1} - x_i)^2}{2!} y''(\xi_i), \qquad x_i \leq \xi_i \leq x_{i+1} \tag{1.9}$$

The substitution of (1.7) into (1.9) gives

$$y(x_{i+1}) = y(x_i) + hf(x_i, y(x_i)) + \frac{h^2}{2!} f'(\xi_i, y(\xi_i)) \tag{1.10}$$

The simplest numerical method is obtained by truncating (1.10) after the second term. Thus with $u_i \simeq y(x_i)$,

$$\begin{aligned} u_{i+1} &= u_i + hf(x_i, u_i), \qquad i = 0, 1, \ldots, N - 1, \\ u_0 &= y_0 \end{aligned} \tag{1.11}$$

This method is called the Euler method.

By assuming that the value of u_i is exact, we find that the application of (1.11) to compute u_{i+1} creates an error in the value of u_{i+1}. This error is called the local truncation error, e_{i+1}. Define the local solution, $z(x)$, by

$$z'(x) = f(x, z), \qquad z(x_i) = u_i \tag{1.12}$$

An expression for the local truncation error, $e_{i+1} = z(x_{i+1}) - u_{i+1}$, can be obtained by comparing the formula for u_{i+1} with the Taylor's series expansion of the local solution about the point x_i. Since

$$z(x_i + h) = z(x_i) + hf(x_i, z(x_i)) + \frac{h^2}{2!} z''(\bar{\xi}_i)$$

or

$$z(x_i + h) = u_i + hf(x_i, u_i) + \frac{h^2}{2!} z''(\bar{\xi}_i), \qquad x_i \leq \bar{\xi}_i \leq x_{i+1} \tag{1.13}$$

it follows that

$$e_{i+1} = \frac{h^2}{2!} z''(\bar{\xi}_i) = 0(h^2) \tag{1.14}$$

The notation $0(\ )$ denotes terms of order $(\ )$, i.e., $f(h) = 0(h^L)$ if $|f(h)| \leq Ah^l$ as $h \to 0$, where A and l are constants [1]. The global error is defined as

$$\mathcal{E}_{i+1} = y(x_{i+1}) - u_{i+1} \tag{1.15}$$

and is thus the difference between the true solution and the numerical solution at $x = x_{i+1}$. Notice the distinction between e_{i+1} and $\mathcal{E}_{i+1}$. The relationships between e_{i+1} and $\mathcal{E}_{i+1}$ will be discussed later in the chapter.

We say that a method is pth-order accurate if

$$e_{i+1} = 0(h^{p+1}) \tag{1.16}$$

and from (1.14) and (1.16) the Euler method is first-order accurate. From the previous discussions one can see that the local truncation error in each step can be made as small as one wishes provided the step-size is chosen sufficiently small.

The Euler method is explicit since the function f is evaluated with known information (i.e., at the left-hand side of the subinterval). The method is pictured in Figure 1.1. The question now arises as to whether the Euler method is able to provide an accurate approximation to (1.7). To partially answer this question, we consider Example 1, which illustrates the properties of the Euler method.

EXAMPLE 1

Kehoe and Butt [2] have studied the kinetics of benzene hydrogenation on a supported Ni/kieselguhr catalyst. In the presence of a large excess of hydrogen, the reaction is pseudo-first-order at temperatures below 200°C with the rate given by

$$-r = P_{H_2} k_0 K_0 T \exp\left[\frac{(-Q - E_a)}{R_g T}\right] C_B \qquad \text{mole/(g of catalyst·s)}$$

where

$$\begin{aligned} R_g &= \text{gas constant, 1.987 cal/(mole·K)} \\ -Q - E_a &= \text{2700 cal/mole} \\ P_{H_2} &= \text{hydrogen partial pressure (torr)} \\ k_0 &= \text{4.22 mole/(gcat·s·torr)} \\ K_0 &= 2.63 \times 10^{-6}\ \text{cm}^3/(\text{mole·K}) \\ T &= \text{absolute temperature (K)} \\ C_B &= \text{concentration of benzene (mole/cm}^3\text{).} \end{aligned}$$

Price and Butt [3] studied this reaction in a tubular reactor. If the reactor is assumed to be isothermal, we can calculate the dimensionless concentration profile of benzene in their reactor given plug flow operation in the absence of inter- and intraphase gradients. Using a typical run,

$$\begin{aligned} P_{H_2} &= 685 \text{ torr} \\ \rho_B &= \text{density of the reactor bed, 1.2 gcat/cm}^3 \\ \theta &= \text{contact time, 0.226 s} \\ T &= 150°\text{C} \end{aligned}$$

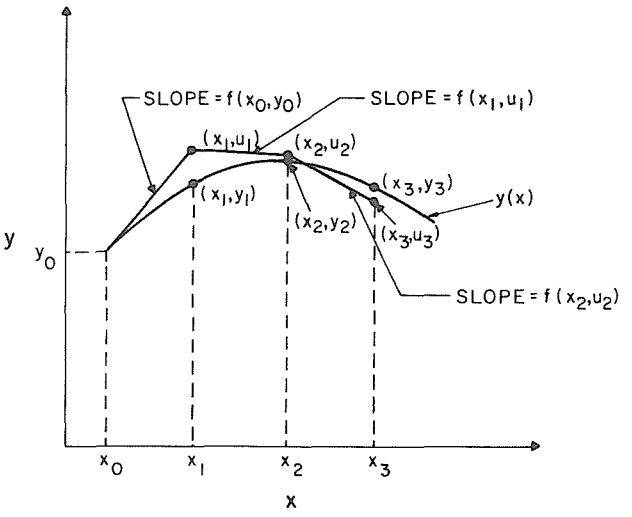

FIGURE 1.1 Euler method.

SOLUTION

Define

C_B^0 = feed concentration of benzene (mole/cm^3)

z = axial reactor coordinate (cm)

L = reactor length

y = dimensionless concentration of benzene (C_B/C_B^0)

x = dimensionless axial coordinate (z/L).

The one-dimensional steady-state material balance for the reactor that expresses the fact that the change in the axial convection of benzene is equal to the amount converted by reaction is

$$\frac{d}{dx}\left(\frac{C_B}{\theta}\right) = r$$

with

$$C_B = C_B^0 \quad \text{at} \quad x = 0$$

Since θ is constant,

$$\frac{dy}{dx} = -\rho_B \theta P_{H_2} k_0 K_0 T \exp\left[\frac{(-Q - E_a)}{R_g T}\right] y$$

Let

$$\phi = \rho_B \theta P_{H_2} k_0 K_0 T \exp\left[\frac{(-Q - E_a)}{R_g T}\right]$$

Using the data provided, we have $\phi = 21.6$. Therefore, the material balance equation becomes

$$\frac{dy}{dx} = -21.6y$$

with

$$y = 1 \quad \text{at} \quad x = 0$$

and analytical solution

$$y = \exp(-21.6x)$$

Now we solve the material balance equation using the Euler method [Eq. (1.11)]:

$$u_{i+1} = u_i - 21.6hu_i, \qquad i = 0, 1, 2, \ldots, N - 1$$

where

$$h = \frac{1}{N}$$

Table 1.1 shows the generated results. Notice that for $N = 10$ the differences between the analytical solution and the numerical approximation increase with x. In a problem where the analytical solution decreases with increasing values of the independent variable, a numerical method is unstable if the global error grows with increasing values of the independent variable (for a rigorous definition of stability, see [4]). Therefore, for this problem the Euler method is unstable when $N = 10$. For $N = 20$ the global error decreases with x, but the solution oscillates in sign. If the error decreases with increasing x, the method is said to be stable. Thus with $N = 20$ the Euler method is stable (for this problem), but the solution contains oscillations. For all $N > 20$, the method is stable and produces no oscillations in the solution.

From a practical standpoint, the "effective" reaction zone would be approximately $0 \leq x \leq 0.2$. If the reactor length is reduced to $0.2L$, then a more realistic problem is produced. The material balance equation becomes

$$\frac{dy}{dx} = -4.32y$$

$$y = 1 \quad \text{at} \quad x = 0$$

Results for the "short" reactor are shown in Table 1.2. As with Table 1.1, we see that a large number of steps are required to achieve a "good" approximation to the analytical solution. An explanation of the observed behavior is provided in the next section.

Physically, the solutions are easily rationalized. Since benzene is a reactant, thus being converted to products as the fluid progresses toward the reactor outlet ($x = 1$), y should decrease with x. Also, a longer reactor would allow for greater conversion, i.e., smaller y values at $x = 1$.

TABLE 1.1 Results of Euler Method on $\frac{dy}{dx} = -21.6y$, $y = 1$ at $x = 0$

x	Analytical Solution†	$N = 10$	$N = 20$	$N = 100$	$N = 8000$
0.00	1.00000	1.0000	1.00000	1.00000	1.00000
0.05	0.33960	—	−0.80000(−1)	0.29620	0.33910
0.10	0.11533	−1.1600	0.64000(−2)	0.87733(−1)	0.11499
0.15	0.39164(−1)	—	−0.51200(−3)	0.25986(−1)	0.38993(−1)
0.20	0.13300(−1)	1.3456	0.40960(−4)	0.76970(−2)	0.13222(−1)
0.25	0.45166(−2)	—	−0.32768(−5)	0.22798(−2)	0.44837(−2)
0.30	0.15338(−2)	−1.5609	0.26214(−6)	0.67528(−3)	0.15204(−2)
0.35	0.52088(−3)	—	−0.20972(−7)	0.20000(−3)	0.51558(−3)
0.40	0.17689(−3)	1.8106	0.16777(−8)	0.59244(−4)	0.17483(−3)
0.45	0.60070(−4)	—	−0.13422(−9)	0.17548(−4)	0.59286(−4)
0.50	0.20400(−4)	−2.1003	0.10737(−10)	0.51976(−5)	0.20104(−4)
0.55	0.69276(−5)	—	−0.85899(−12)	0.15395(−5)	0.68172(−5)
0.60	0.23526(−5)	2.4364	0.68719(−13)	0.45600(−6)	0.23117(−5)
0.65	0.79892(−6)	—	−0.54976(−14)	0.13507(−6)	0.78390(−6)
0.70	0.27131(−6)	−2.8262	0.43980(−15)	0.40006(−7)	0.26582(−6)
0.75	0.92136(−7)	—	−0.35184(−16)	0.11850(−7)	0.90139(−7)
0.80	0.31289(−7)	3.2784	0.28147(−17)	0.35098(−8)	0.30566(−7)
0.85	0.10626(−7)	—	−0.22518(−18)	0.10396(−8)	0.10365(−7)
0.90	0.36084(−8)	−3.8030	0.18014(−19)	0.30793(−9)	0.35148(−8)
0.95	0.12254(−8)	—	−0.14412(−20)	0.91207(−10)	0.11919(−8)
1.00	0.41614(−9)	4.4114	0.11529(−21)	0.27015(−10)	0.40416(−9)

† (−3) denotes 1.0×10^{-3}.

STABILITY

In Example 1 it was seen that for some choices of the step-size, the approximate solution was unstable, or stable with oscillations. To see why this happens, we will examine the question of stability using the test equation

$$\frac{dy}{dx} = \lambda y \qquad y(0) = y_0 \tag{1.17}$$

where λ is a complex constant. Application of the Euler method to (1.17) gives

$$u_{i+1} = u_i + \lambda h u_i \tag{1.18}$$

or

$$u_{i+1} = (1 + h\lambda)u_i = (1 + h\lambda)^2 u_{i-1} = \ldots = (1 + h\lambda)^{i+1} u_0 \tag{1.19}$$

The analytical solution of (1.17) is

$$y(x_{i+1}) = y_0 e^{\lambda x_{i+1}} = y_0 e^{(i+1)h\lambda} \tag{1.20}$$

Comparing (1.20) with (1.19) shows that the application of Euler's method to (1.17) is equivalent to using the expression $(1 + h\lambda)$ as an approximation for

TABLE 1.2 Results of Euler Method on $\frac{dy}{dx} = -4.32y$, $y = 1$ at $x = 0$

x	Analytical Solution	$N = 100$	$N = 1000$	$N = 8000$
0.0	1.00000	1.00000	1.00000	1.00000
0.1	0.64921	0.64300	0.64860	0.64913
0.2	0.42147	0.41345	0.42068	0.42137
0.3	0.27362	0.26585	0.27286	0.27353
0.4	0.17764	0.17094	0.17698	0.17756
0.5	0.11533	0.10992	0.11479	0.11526
0.6	0.07487	0.07067	0.07445	0.07481
0.7	0.04860	0.04544	0.04828	0.04856
0.8	0.03155	0.02922	0.03132	0.03152
0.9	0.02048	0.01878	0.02031	0.02046
1.0	0.01330	0.01208	0.01317	0.01328

$e^{\lambda h}$. Now suppose that the value y_0 is not exactly representable by a machine number (see Appendix A), then $e_0 = y_0 - u_0$ will be nonzero. From (1.19), with u_0 replaced by $y_0 - e_0$,

$$u_{i+1} = (1 + h\lambda)^{i+1}(y_0 - e_0)$$

and the global error $\mathcal{E}_{i+1}$ is

$$\mathcal{E}_{i+1} = y(x_{i+1}) - u_{i+1} = y_0 e^{(i+1)h\lambda} - (1 + h\lambda)^{i+1}(y_0 - e_0)$$

or

$$\mathcal{E}_{i+1} = [e^{(i+1)\lambda h} - (1 + h\lambda)^{i+1}]\, y_0 + (1 + h\lambda)^{i+1} e_0 \qquad \textbf{(1.21)}$$

Hence, the global error consists of two parts. First, there is an error that results from the Euler method approximation $(1 + h\lambda)$ for $e^{\lambda h}$. The second part is the propagation effect of the initial error, e_0. Clearly, if $|1 + h\lambda| > 1$, this component will grow and, no matter what the magnitude of e_0 is, it will become the dominant term in $\mathcal{E}_{i+1}$. Therefore, to keep the propagation effects of previous errors bounded when using the Euler method, we require

$$|1 + h\lambda| \leq 1 \qquad \textbf{(1.22)}$$

The region of absolute stability is defined by the set of h (real nonnegative) and λ values for which a perturbation in a single value u_i will produce a change in subsequent values that does not increase from step to step [4]. Thus, one can see from (1.22) that the stability region for (1.17) corresponds to a unit disk in the complex $h\lambda$-plane centered at $(-1, 0)$. If λ is real, then

$$-2 \leq h\lambda \leq 0 \qquad \textbf{(1.23)}$$

Notice that if the propagation effect is the dominant term in (1.21), the global error will oscillate in sign if $-2 \leq h\lambda \leq -1$.

EXAMPLE 2

Referring to Example 1, find the maximum allowable step-size for stability and for nonoscillatory behavior for the material balance equations of the "long" and "short" reactor. Can you now explain the behavior shown in Tables 1.1 and 1.2?

SOLUTION

For the long reactor: $\lambda_L = -21.6$ (real)
For the short reactor: $\lambda_S = -4.32$ (real)
For stability: $0 \geqslant h\lambda \geqslant -2$
For nonoscillatory error: $0 \geqslant h\lambda > -1$

	Long Reactor	Short Reactor
Unstable	$0.0926 < h$	$0.4630 < h$
Stable, error oscillations	$0.0463 \leqslant h \leqslant 0.0926$	$0.2315 \leqslant h \leqslant 0.4630$
Stable, no error oscillations	$h < 0.0463$	$h < 0.2315$

For the short reactor, all of the presented solutions are stable and nonoscillatory since the step-size is always less than 0.2315. The large number of steps required for a "reasonably" accurate solution is a consequence of the first-order accuracy of the Euler method.

For the long reactor with $N > 20$ the solutions are stable and nonoscillatory since h is less than 0.0463. With $N = 10$, $h = 0.1$ and the solution is unstable, while for $N = 20$, $h = 0.05$ and the solution is stable and oscillatory. From the above table, when $N = 20$, the global error should oscillate if the propagation error is the dominant term in Eq. (1.21). This behavior is not observed from the results shown in Table 1.1. The data for $N = 10$ and $N = 20$ can be explained by examining Eq. (1.21):

$$\mathcal{E}_{i+1} = [e^{(i+1)\lambda h} - (1 + h\lambda)^{i+1}]y_0 + (1 + h\lambda)^{i+1}e_0 = (\mathrm{A})y_0 + (\mathrm{B})e_0$$

For $N = 10$, $h = 0.1$ and $\lambda h = -2.16$. Therefore,

i	(A)	(B)	Global Error Calculated from Results Shown in Table 1.1
0	1.2753	−1.160	1.2753
1	−1.3323	1.3456	−1.3323
2	1.5624	−1.5609	1.5624

Since $y_0 = 1$ and e_0 is small, the global error is dominated by term (A) and not the propagation term, i.e., term (B). For $N = 20$, $h = 0.05$ and $\lambda h = -1.08$. Therefore,

i	(A)	(B)	Global Error Calculated from Results Shown in Table 1.1
0	0.4196	-0.08	0.4196
1	0.1089	0.64×10^{-2}	0.1089
2	0.3967×10^{-1}	-0.512×10^{-3}	0.3967×10^{-1}

As with $N = 10$, the global error is dominated by the term (A). Thus no oscillations in the global error are seen for $N = 20$.

From (1.19) one can explain the oscillations in the solution for $N = 10$ and 20. If

$$(1 + h\lambda) < 0$$

then the numerical solution will alternate in sign. For $(1 + h\lambda)$ to be equal to zero, $h\lambda = -1$. When $N = 10$ or 20, $h\lambda$ is less than -1 and therefore oscillations in the solution occur.

For this problem, it was shown that for the long reactor with $N = 10$ or 20 the propagation error was not the dominant part of the global error. This behavior is a function of the parameter λ and thus will vary from problem to problem.

From Examples 1 and 2 one observes that there are two properties of the Euler method that could stand improvement: stability and accuracy. Implicit within these categories is the cost of computation. Since the step-size of the Euler method has strict size requirements for stability and accuracy, a large number of function evaluations are required, thus increasing the cost of computation. Each of these considerations will be discussed further in the following sections. In the next section we will show methods that improve the order of the accuracy.

RUNGE-KUTTA METHODS

Runge-Kutta methods are explicit algorithms that involve evaluation of the function f at points between x_i and x_{i+1}. The general formula can be written as

$$u_{i+1} = u_i + \sum_{j=1}^{\nu} \omega_j K_j \tag{1.24}$$

where

$$K_j = hf\left(x_i + c_j h,\ u_i + \sum_{l=1}^{j-1} a_{jl} K_l\right)$$
$$c_1 = 0 \tag{1.25}$$

Notice that if $\nu = 1$, $\omega_1 = 1$, and $K_1 = hf(x_i, u_i)$, the Euler method is obtained. Thus, the Euler method is the lowest-order Runge-Kutta formula. For higher-order formulas, the parameters ω, c, and a are found as follows. For example, if $\nu = 2$, first expand the exact solution of (1.7) in a Taylor's series,

$$y(x_{i+1}) = y(x_i) + hf(x_i, y(x_i)) + \frac{h^2}{2!} f'(x_i, y(x_i)) + 0(h^3) \tag{1.26}$$

Next, rewrite $f'(x_i, y(x_i))$ as

$$\frac{df_i}{dx} = \frac{\partial f_i}{\partial x} + \frac{\partial f_i}{\partial y}\frac{dy}{dx}\bigg|_{x=x_i} = (f_x + f_y f)_i \tag{1.27}$$

Substitute (1.27) into (1.26) and truncate the $0(h^3)$ term to give

$$u_{i+1} = u_i + hf_i + \frac{h^2}{2}(f_x + f_y f)_i \tag{1.28}$$

Expand each of the K_j's about the ith position. To do so, denote

$$K_1 = hf(x_i, u_i) = hf_i \tag{1.29a}$$

and

$$K_2 = hf(x_i + c_2 h, u_i + a_{21}K_1) \tag{1.29b}$$

Recall that for any two functions η and ϕ that are located near x_i and u_i, respectively,

$$f(\eta, \phi) \simeq f(x_i, u_i) + (\eta - x_i)f_x(x_i, u_i) + (\phi - u_i)f_y(x_i, u_i) \tag{1.30}$$

Using (1.30) on K_2 gives

$$K_2 = h(f_i + c_2 h f_x + a_{21}K_1 f_y)$$

or

$$K_2 = hf_i + h^2(c_2 f_x + a_{21} f_y f)_i \tag{1.31}$$

Substitute (1.29a) and (1.31) into (1.24):

$$u_{i+1} = u_i + \omega_1 hf_i + \omega_2 hf_i + \omega_2 h^2 c_2 (f_x)_i + a_{21}\omega_2 h^2 (f_y f)_i \tag{1.32}$$

Comparing like powers of h in (1.32) and (1.28) shows that

$$\omega_1 + \omega_2 = 1.0$$

$$\omega_2 c_2 = 0.5$$

$$\omega_2 a_{21} = 0.5$$

The Runge-Kutta algorithm is completed by choosing the free parameter; i.e., once either ω_1, ω_2, c_2, or a_{21} is chosen, the others are fixed by the above formulas.

If c_2 is set equal to 0.5, the Runge-Kutta scheme is

$$u_{i+1} = u_i + hf(x_i + \tfrac{1}{2}h,\ u_i + \tfrac{1}{2}hf_i), \qquad i = 0, 1, \ldots, N-1$$
$$u_0 = y_0 \tag{1.33}$$

or a midpoint method. For $c_2 = 1$,

$$u_{i+1} = u_i + \frac{h}{2}[f_i + f(x_i + h,\ u_i + hf_i)], \qquad i = 0, 1, \ldots, N-1$$
$$u_0 = y_0 \tag{1.34}$$

These two schemes are graphically interpreted in Figure 1.2. The methods are second-order accurate since (1.28) and (1.31) were truncated after terms of $0(h^2)$.

If a pth-order accurate formula is desired, one must take v large enough so that a sufficient number of degrees of freedom (free parameters) are available in order to obtain agreement with a Taylor's series truncated after terms in h^p. A table of minimum such v for a given p is

p	2	3	4	5	6	...
v	2	3	4	6	8	...

Since v represents the number of evaluations of the function f for a particular i, the above table shows the minimum amount of work required to achieve a desired order of accuracy. Notice that there is a jump in v from 4 to 6 when p goes from 4 to 5, so traditionally, because of the extra work, methods with $p > v$ have been disregarded. An example of a fourth-order scheme is the

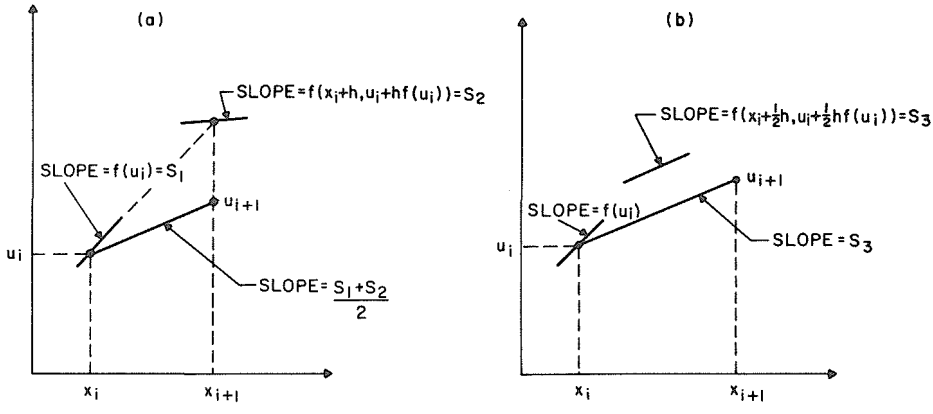

FIGURE 1.2 Runge-Kutta interpretations. (*a*) Eq. (1.34). (*b*) Eq. (1.33).

Runge-Kutta-Gill Method [41] and is:

$$
\begin{aligned}
u_{i+1} &= u_i + \tfrac{1}{6}(K_1 + K_4) + \tfrac{1}{3}(bK_2 + dK_3) \\
K_1 &= hf(x_i, u_i) \\
K_2 &= hf(x_i + \tfrac{1}{2}h, u_i + \tfrac{1}{2}K_1) \\
K_3 &= hf(x_i + \tfrac{1}{2}h, u_i + aK_1 + bK_2) \\
K_4 &= hf(x_i + h, u_i + cK_2 + dK_3) \\
a &= \frac{\sqrt{2} - 1}{2}, \qquad b = \frac{2 - \sqrt{2}}{2} \\
c &= -\frac{\sqrt{2}}{2}, \qquad d = 1 + \frac{\sqrt{2}}{2}
\end{aligned}
\tag{1.35}
$$

for

$$
i = 0, 1, \ldots, N - 1 \qquad \text{and} \qquad u_0 = y_0
$$

The parameter choices in this algorithm have been made to minimize round-off error.

Use of the explicit Runge-Kutta formulas improves the order of accuracy, but what about the stability of these methods? For example, if λ is real, the second-order Runge-Kutta algorithm is stable for the region $-2.0 \leq \lambda h \leq 0$, while the fourth-order Runge-Kutta-Gill method is stable for the region $-2.8 \leq \lambda h \leq 0$.

EXAMPLE 3

A thermocouple at equilibrium with ambient air at 10°C is plunged into a warm-water bath at time equal to zero. The warm water acts as an infinite heat source at 20°C since its mass is many times that of the thermocouple. Calculate the response curve of the thermocouple.

Data: Time constant of the thermocouple = $0.4\ \text{min}^{-1}$.

SOLUTION

Define

C_p = thermal capacity of the thermocouple
U = heat transfer coefficient of the thermocouple
A = heat transfer area of thermocouple
t = time (min)
T, T_p, T_0 = temperature of thermocouple, water, and ambient air

$$\theta = \frac{T_p - T}{T_p - T_0}$$

$$\eta = \frac{C_p}{UA} = \text{time constant of the thermocouple}$$

$$t^* = \frac{t}{10}$$

The governing differential equation is Newton's law of heating or cooling and is

$$C_p \frac{dT}{dt} = UA(T_p - T), \qquad T = 10°\text{C} \quad \text{at} \quad t = 0$$

If the response curve is calculated for $0 \leq t \leq 10$ min, then

$$\frac{d\theta}{dt^*} = -25\theta, \qquad \theta = 1 \quad \text{at} \quad t = 0$$

The analytical solution is

$$\theta = e^{-25t^*}, \qquad 0 \leq t^* \leq 1$$

Now solve the differential equation using the second-order Runge-Kutta method [Eq. (1.34)]:

$$u_0 = 1$$

$$u_{i+1} = u_i + \frac{h}{2}[f_i + f(t_i^* + h, u_i + hf_i)], \qquad i = 0, 1, \ldots, N - 1$$

where

$$f_i = -25u_i$$

$$f(t_i^* + h, u_i + hf_i) = -25(u_i + hf_i)$$

and using the Runge-Kutta-Gill method [Eq. (1.35)]:

$$u_0 = 1$$

$$u_{i+1} = u_i + \tfrac{1}{6}(K_1 + K_4) + \tfrac{1}{3}(bK_2 + dK_3), \qquad i = 0, 1, \ldots, N - 1$$

$$K_1 = -25hu_i$$

$$K_2 = -25h(u_i + \tfrac{1}{2}K_1)$$

$$K_3 = -25h(u_i + aK_1 + bK_2)$$

$$K_4 = -25h(u_i + cK_2 + dK_3)$$

Table 1.3 shows the generated results. Notice that for $N = 20$ the second-order Runge-Kutta method shows large discrepancies when compared with the analytical solution. Since $\lambda = -25$, the maximum stable step-size for this method is $h = 0.08$, and for $N = 20$, h is very close to this maximum. For the

TABLE 1.3 Comparison of Runge-Kutta Methods $\frac{d\theta}{dt^*} = -250, \theta = 1$ at $t^* = 0$

		Second-Order Runge-Kutta Method		Runge-Kutta-Gill Method	
t^*	Analytical Solution	$N = 20$	$N = 200$	$N = 20$	$N = 200$
0.00000	1.00000	1.00000	1.00000	1.00000	1.00000
0.20000	0.67379(−02)	0.79652(−01)	0.68350(−02)	0.89356(−02)	0.67380(−02)
0.40000	0.45400(−04)	0.63444(−02)	0.46717(−04)	0.79845(−04)	0.45401(−04)
0.60000	0.30590(−06)	0.50534(−03)	0.31931(−06)	0.71346(−06)	0.30591(−06)
0.80000	0.20612(−08)	0.40252(−04)	0.21825(−08)	0.63752(−08)	0.20612(−08)
1.00000	0.13888(−10)	0.32061(−05)	0.14917(−10)	0.56966(−10)	0.13889(−10)

Runge-Kutta-Gill method the maximum stable step-size is $h = 0.112$, and h never approaches this limit. From Table 1.3 one can also see that the Runge-Kutta-Gill method produces a more accurate solution than the second-order method, which is as expected since it is fourth-order accurate. To further this point, refer to Table 1.4 where we compare a first (Euler), a second, and a fourth-order method to the analytical solution. For a given N, the accuracy increases with the order of the method, as one would expect. Since the Runge-Kutta-Gill method (RKG) requires four function evaluations per step while the Euler method requires only one, which is computationally more efficient? One can answer this question by comparing the RKG results for $N = 100$ with the Euler results for $N = 800$. The RKG method ($N = 100$) takes 400 function evaluations to reach $t^* = 1$, while the Euler method ($N = 800$) takes 800. From Table 1.4 it can be seen that the RKG ($N = 100$) results are more accurate than the Euler ($N = 800$) results, and require half as many function evaluations. It is therefore shown that for this problem although more function evaluations per step are required by the higher-order accurate formulas, they are computationally more efficient when trying to meet a specified error tolerance (this result cannot be generalized to include all problems).

Physically, all the results in Tables 1.3 and 1.4 have significance. Since $\theta = (T_p - T)/(T_p - T_0)$, initially $T = T_0$ and $\theta = 1$. When the thermocouple is plunged into the water, the temperature will begin to rise and T will approach T_p, that is, θ will go to 0.

So far we have always illustrated the numerical methods with test problems that have an analytical solution so that the errors are easily recognizable. In a practical problem an analytical solution will not be known, so no comparisons can be made to find the errors occurring during computation. Alternative strategies must be constructed to estimate the error. One method of estimating the local error would be to calculate the difference between u^*_{i+1} and u_{i+1} where u_{i+1} is calculated using a step-size of h and u^*_{i+1} using a step-size of $h/2$. Since the accuracy of the numerical method depends upon the step-size to a certain power, u^*_{i+1} will be a better estimate for $y(x_{i+1})$ than u_{i+1}. Therefore,

$$|z_{i+1} - u^*_{i+1}| < |z_{i+1} - u_{i+1}|$$

TABLE 1.4 Comparison of Runge-Kutta Methods with the Euler Method

$\frac{d\theta}{dt^*} = -25\theta,\ \theta = 1$ at $t^* = 0$

		Second-Order Runge-Kutta Method		Runge-Kutta-Gill Method		Euler Method	
t^*	Analytical Solution	$N = 100$	$N = 800$	$N = 100$	$N = 800$	$N = 100$	$N = 800$
0.00000	1.000000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
0.20000	0.67379(−02)	0.71746(−02)	0.67436(−02)	0.67393(−02)	0.67379(−02)	0.31712(−02)	0.62212(−02)
0.40000	0.45400(−04)	0.51476(−04)	0.45476(−04)	0.45418(−04)	0.45400(−04)	0.10057(−04)	0.38703(−04)
0.60000	0.30590(−06)	0.36932(−04)	0.30667(−06)	0.30609(−06)	0.30590(−06)	0.31892(−07)	0.24078(−06)
0.80000	0.20612(−08)	0.26497(−08)	0.20680(−08)	0.20628(−08)	0.20612(−08)	0.10113(−09)	0.14980(−08)
1.00000	0.13888(−10)	0.19011(−10)	0.13946(−10)	0.13902(−10)	0.13888(−10)	0.32072(−12)	0.93191(−11)

and

$$e_{i+1} = z_{i+1} - u_{i+1} \simeq u^*_{i+1} - u_{i+1}$$

For Runge-Kutta formulas, using the one-step, two half-steps procedure can be very expensive since the cost of computation increases with the number of function evaluations. The following table shows the number of function evaluations per step for pth-order accurate formulas using two half-steps to calculate u^*_{i+1}:

p	2	3	4	5
Evaluations of f per step	5	8	11	14

Take for example the Runge-Kutta-Gill method. The Gill formula requires four function evaluations for the computation of u_{i+1} and seven for u^*_{i+1}. A better procedure is Fehlberg's method (see [5]), which uses a Runge-Kutta formula of higher-order accuracy than used for u_{i+1} to compute u^*_{i+1}. The Runge-Kutta-Fehlberg fourth-order pair of formulas is

$$u_{i+1} = u_i + \left[\tfrac{25}{216}k_1 + \tfrac{1408}{2565}k_3 + \tfrac{2197}{4104}k_4 - \tfrac{1}{5}k_5\right], \qquad e_{i+1} = 0(h^5) \tag{1.36}$$

$$u^*_{i+1} = u_i + \left[\tfrac{16}{135}k_1 + \tfrac{6656}{12825}k_3 + \tfrac{28561}{56430}k_4 - \tfrac{9}{50}k_5 + \tfrac{2}{55}k_6\right], \qquad e_{i+1} = 0(h^6),$$

where

$$\begin{aligned}
k_1 &= hf(x_i, u_i)\\
k_2 &= hf(x_i + \tfrac{1}{4}h, u_i + \tfrac{1}{4}k_1)\\
k_3 &= hf(x_i + \tfrac{3}{8}h, u_i + \tfrac{3}{32}k_1 + \tfrac{9}{32}k_2)\\
k_4 &= hf(x_i + \tfrac{12}{13}h, u_i + \tfrac{1932}{2197}k_1 - \tfrac{7200}{2197}k_2 + \tfrac{7296}{2197}k_3)\\
k_5 &= hf(x_i + h, u_i + \tfrac{439}{216}k_1 - 8k_2 + \tfrac{3680}{513}k_3 - \tfrac{845}{4104}k_4)\\
k_6 &= hf(x_i + \tfrac{1}{2}h, u_i - \tfrac{8}{27}k_1 + 2k_2 - \tfrac{3544}{2565}k_3 + \tfrac{1859}{4104}k_4 - \tfrac{11}{40}k_5)
\end{aligned}$$

On first inspection the system (1.36) appears quite complicated, but it can be programmed in a very straightforward way. Notice that the formula for u_{i+1} is fourth-order accurate but requires five function evaluations as compared with the four of the Runge-Kutta-Gill method, which is of the same order accuracy. However, if e_{i+1} is to be estimated, the half-step method using the Runge-Kutta-Gill method requires eleven function evaluations while Eq. (1.36) requires only six—a considerable decrease! The key is to use a pair of formulas with a common set of k_i's. Therefore, if (1.36) is used, as opposed to (1.35), the accuracy is maintained at fourth-order, the stability criteria remains the same, but the cost of computation is significantly decreased. That is why a number of commercially available computer programs (see section on Mathematical Software) use Runge-Kutta-Fehlberg algorithms for solving IVPs.

In this section we have presented methods that increase the order of accuracy, but their stability limitations remain severe. In the next section we discuss methods that have improved stability criteria.

IMPLICIT METHODS

If we once again consider Eq. (1.7) and expand $y(x)$ about the point x_{i+1} using Taylor's theorem with remainder:

$$y(x_i) = y(x_{i+1}) - hy'(x_{i+1}) + \frac{h^2}{2!} y''(\xi_i), \qquad x_i \leq \xi_i \leq x_{i+1} \tag{1.37}$$

Substitution of (1.7) into (1.37) gives

$$y(x_i) = y(x_{i+1}) - hf(x_{i+1}, y(x_{i+1})) + \frac{h^2}{2!} f_i'(\bar{\xi}, y(\bar{\xi})), \qquad x_i \leq \bar{\xi} \leq x_{i+1} \tag{1.38}$$

A numerical procedure of (1.7) can be obtained from (1.38) by truncating after the second term:

$$\begin{aligned} u_{i+1} &= u_i + hf_{i+1}, \qquad i = 0, 1, \ldots, N-1, \\ u_0 &= y_0 \end{aligned} \tag{1.39}$$

Equation (1.39) is called the implicit Euler method because the function f is evaluated at the right-hand side of the subinterval. Since the value of u_{i+1} is unknown, (1.39) is nonlinear if f is nonlinear. In this case, one can use a Newton iteration (see Appendix B). This takes the form

$$u_{i+1}^{[s+1]} = h\left[f\Big|_{u_{i+1}^{[s]}} + \frac{\partial f}{\partial y}\Big|_{u_{i+1}^{[s]}} (u_{i+1}^{[s+1]} - u_{i+1}^{[s]}) \right] + u_i \tag{1.40}$$

or after rearrangement

$$\left(1 - h\frac{\partial f}{\partial y}\right)\Big|_{u_{i+1}^{[s]}} (u_{i+1}^{[s+1]} - u_{i+1}^{[s]}) = hf\Big|_{u_{i+1}^{[s]}} + u_i - u_{i+1}^{[s]} \tag{1.41}$$

where $u_{i+1}^{[s]}$ is the sth iterate of u_{i+1}. Iterate on (1.41) until

$$|u_{i+1}^{[s+1]} - u_{i+1}^{[s]}| \leq \text{TOL} \tag{1.42}$$

where TOL is a specified absolute error tolerance.

One might ask what has been gained by the implicit nature of (1.39) since it requires more work than, say, the Euler method for solution. If we apply the implicit Euler scheme to (1.17) (λ real),

$$u_{i+1} = u_i + h\lambda u_{i+1}$$

or

$$u_{i+1} = \left(\frac{1}{1 - h\lambda}\right) u_i = \left(\frac{1}{1 - h\lambda}\right)^{i+1} y_0 \tag{1.43}$$

If $\lambda < 0$, then (1.39) is stable for all $h > 0$ or it is unconditionally stable, and never oscillates.

The implicit nature of the method has stabilized the algorithm, but unfortunately the scheme is only first-order accurate. To obtain a higher order of accuracy, combine (1.38) and (1.10) to give

$$2[y(x_{i+1}) - y(x_i)] = h[f_{i+1} + f_i] + 0(h^3) \tag{1.44}$$

The algorithm associated with (1.44) is

$$\begin{aligned} u_{i+1} &= u_i + \frac{h}{2}[f_{i+1} + f_i], \qquad i = 0, 1, \ldots, N - 1, \\ u_0 &= y_0 \end{aligned} \tag{1.45}$$

which is commonly called the trapezoidal rule. Equation (1.45) is second-order accurate, and the stability of the scheme can be examined by applying the method to (1.17), giving (λ real)

$$u_{i+1} = \left[\frac{\left(1 + \frac{\lambda h}{2}\right)}{\left(1 - \frac{\lambda h}{2}\right)}\right]^{i+1} y_0 \tag{1.46}$$

If $\lambda < 0$, then (1.45) is unconditionally stable, but notice that if $h\lambda < -2$ the method will produce oscillations in the sign of the error. A summary of the stability regions (λ real) for the methods discussed so far is shown in Table 1.5.

From Table 1.5 we see that the Euler method requires a small step-size for stability. Although the criteria for the Runge-Kutta methods are not as

TABLE 1.5 Comparison of Methods Based upon $\frac{dy}{dx} = -\tau y$, $y(0) = 1$, $\tau > 0$ and is a real constant

Method	Stable Step-Size, No Oscillations	Stable Step-Size, Oscillations	Unstable Step-Size	Order of Accuracy
Euler (1.11)	$h\tau < 1$	$1 \leq h\tau \leq 2$	$2 < h\tau$	1
Second-order Runge-Kutta (1.33)	$h\tau \leq 2$	None	$2 < h\tau$	2
Runge-Kutta-Gill (1.35)	$h\tau \leq 2.8$	None	$2.8 < h\tau$	4
Implicit Euler (1.39)	$h\tau < \infty$	None	None	1
Trapezoidal (1.45)	$h\tau < 2$	$2 \leq h\tau \leq \infty$	None	2

stringent as for the Euler method, stable step-sizes for these schemes are also quite small. The trapezoidal rule requires a small step-size to avoid oscillations but is stable for any step-size, while the implicit Euler method is always stable. The previous two algorithms require more arithmetic operations than the Euler or Runge-Kutta methods when f is nonlinear due to the Newton iteration, but are typically used for solution of certain types of problems (see section on stiffness).

In Table 1.5 we once again see the dilemma of stability versus accuracy. In the following section we outline one technique for increasing the accuracy when using any method.

EXTRAPOLATION

Suppose we solve a problem with a step-size of h giving the solution u_i at x_i, and also with a step-size $h/2$ giving the solution ω_i at x_i. If an Euler method is used to obtain u_i and ω_i, then the error is proportional to the step-size (first-order accurate). If $y(x_i)$ is the exact solution at x_i, then

$$\begin{aligned} u_i &\simeq y(x_i) + \phi h \\ \omega_i &\simeq y(x_i) + \phi \frac{h}{2} \end{aligned} \tag{1.47}$$

where ϕ is a constant. Eliminating ϕ from (1.47) gives

$$y(x_i) = 2\omega_i - u_i \tag{1.48}$$

If the error formulas (1.47) are exact, then this procedure gives the exact solution. Since the formulas (1.47) usually only apply as $h \to 0$, then (1.48) is only an approximation, but it is expected to be a more accurate estimate than either ω_i or u_i. The same procedure can be used for higher-order methods. For the trapezoidal rule

$$\begin{aligned} u_i &\simeq y(x_i) + \phi h^2 \\ \omega_i &\simeq y(x_i) + \phi \left(\frac{h}{2}\right)^2 \\ y(x_i) &\simeq \frac{4\omega_i - u_i}{3} \end{aligned} \tag{1.49}$$

EXAMPLE 4

The batch still shown in Figure 1.3 initially contains 25 moles of n-octane and 75 moles of n-heptane. If the still is operated at a constant pressure of 1 atmosphere (atm), compute the final mole fraction of n-heptane, x_H^f, if the remaining solution in the still, S^f, totals 10 moles.

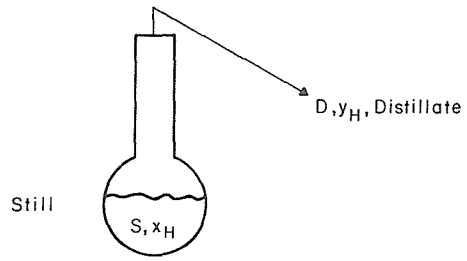

FIGURE 1.3 Batch still.

Data: At 1 atm total pressure, the relationship between x_H and the mole fraction of *n*-heptane in the vapor phase, y_H, is

$$y_H = \frac{2.16x_H}{1 + 1.16\,x_H}$$

SOLUTION

An overall material balance is

$$dS = -dD$$

A material balance of *n*-heptane gives

$$d(x_H S) = -y_H dD$$

Combination of these balances yields

$$\int_{S_0}^{S^f} \frac{dS}{S} = \int_{x_H^0}^{x_H^f} \frac{dx_H}{y_H - x_H}$$

where $S_0 = 100$, $S^f = 10$, $x_H^0 = 0.75$.
Substitute for y_H and integrate to give

$$\left(\frac{S^f}{S_0}\right) = \left(\frac{1 - x_H^0}{1 - x_H^f}\right)\left[\left(\frac{1 - x_H^0}{1 - x_H^f}\right)\left(\frac{x_H^f}{x_H^0}\right)\right]^{1/1.16}$$

and

$$x_H^f = 0.37521825$$

Physically, one would expect x_H to decrease with time since heptane is lighter than octane and would flash in greater amounts than would octane. Now compare the analytical solution to the following numerical solutions. First, reformulate the differential equation by defining

$$t = \frac{S_0 - S}{S_0 - S^f}$$

so that

$$0 \leq t \leq 1$$

Thus:

$$\frac{dx_H}{dt} = 1.16 \frac{(S^f - S_0)}{(S_0(1 - t) + S^f t)} \frac{x_H(1 - x_H)}{(1 + 1.16x_H)}, \qquad x_H = x_H^0 \quad \text{at} \quad t = 0$$

If an Euler method is used, the results are shown in Table 1.6. From a practical standpoint, all the values in Table 1.6 would probably be sufficiently accurate for design purposes, but we provide the large number of significant figures to illustrate the extrapolation method. A simple Euler method is first-order accurate, and so the truncation error should be proportional to $h(1/N)$. This is shown in Table 1.6. Also notice that the error in the extrapolated Euler method decreases faster than that in the Euler method with increasing N. The truncation error of the extrapolation is approximately the square of the error in the basic method. In this example one can see that improved accuracy with less computation is achieved by extrapolation. Unfortunately, the extrapolation is successful only if the step-size is small enough for the truncation error formula to be reasonably accurate. Some nonlinear problems require extremely small step-sizes and can be computationally unreasonable.

Extrapolation is one method of increasing the accuracy, but it does not change the stability of a method. There are commercial packages that employ extrapolation (see section on Mathematical Software), but they are usually based upon Runge-Kutta methods instead of the Euler or trapezoidal rule as outlined

TABLE 1.6 Errors in the Euler Method and the Extrapolated Euler Method for Example 4

Number of Steps	Total Number of Steps	Absolute Value of the Error
Euler Method		
50	50	0.01373
100	100	0.00675
200	200	0.00335
400	400	0.00166
800	800	0.00083
1,600	1,600	0.00041
Extrapolated Euler Method		
50–100	150	0.000220
100–200	300	0.000056
200–400	600	0.000013
400–800	1,200	0.000003
800–1600	2,400	0.000001

above. In the following section we describe techniques currently being used in software packages for which stability, accuracy, and computational efficiency have been addressed in detail (see, for example, [5]).

MULTISTEP METHODS

Multistep methods make use of information about the solution and its derivative at more than one point in order to extrapolate to the next point. One specific class of multistep methods is based on the principle of numerical integration. If the differential equation $y' = f(x, y)$ is integrated from x_i to x_{i+1}, we obtain

$$\int_{x_i}^{x_{i+1}} y' \, dx = \int_{x_i}^{x_{i+1}} f(x, y(x)) \, dx$$

or

$$y(x_{i+1}) = y(x_i) + \int_{x_i}^{x_{i+1}} f(x, y(x)) \, dx \tag{1.50}$$

To carry out the integration in (1.50), approximate $f(x, y(x))$ by a polynomial that interpolates $f(x, y(x))$ at k points, $x_i, x_{i-1}, \ldots, x_{i-k+1}$. If the Newton backward formula of degree $k-1$ is used to interpolate $f(x, y(x))$, then the Adams-Bashforth formulas [1] are generated and are of the form

$$u_{i+1} = u_i + h \sum_{j=1}^{k} b_j u'_{i-j+1} \tag{1.51}$$

where

$$u'_j = f(x_j, u_j)$$

This is called a k-step formula because it uses information from the previous k steps. Note that the Euler formula is a one-step formula ($k = 1$) with $b_1 = 1$. Alternatively, if one begins with (1.51), the coefficients b_j can be chosen by assuming that the past values of u are exact and equating like powers of h in the expansion of (1.51) and of the local solution z_{i+1} about x_i. In the case of a three-step formula

$$u_{i+1} = u_i + h[b_1 u'_i + b_2 u'_{i-1} + b_3 u'_{i-2}]$$

Substituting values of z into this and expanding about x_i gives

$$z_{i+1} = z_i + hz'_i[b_1 + b_2 + b_3] - h^2 z''_i[b_2 + 2b_3] + \frac{h^3}{2!} z'''_i[b_2 + 4b_3] + \ldots$$

where

$$z'_{i-1} = z'_i - hz''_i + \frac{h^2}{2!} z'''_i + \ldots$$

$$z'_{i-2} = z'_i - 2hz''_i + \frac{4h^2}{2!} z'''_i + \ldots$$

The Taylor's series expansion of z_{i+1} is

$$z_{i+1} = z_i + hz_i' + \frac{h^2}{2!} z_i'' + \frac{h^3}{3!} z_i''' + \ldots$$

and upon equating like power of h, we have

$$b_1 + b_2 + b_3 = 1$$

$$b_2 + 2b_3 = -\tfrac{1}{2}$$

$$b_2 + 4b_3 = \tfrac{1}{3}$$

The solution of this set of linear equations is $b_1 = \frac{23}{12}$, $b_2 = -\frac{16}{12}$, and $b_3 = \frac{5}{12}$. Therefore, the three-step Adams-Bashforth formula is

$$u_{i+1} = u_i + \frac{h}{12}[23u_i' - 16u_{i-1}' + 5u_{i-2}'] \tag{1.52}$$

with an error $e_{i+1} = 0(h^4)$ [generally $e_{i+1} = 0(h^{k+1})$ for any value of k; for example, in (1.52) $k = 3$].

A difficulty with multistep methods is that they are not self-starting. In (1.52) values for u_i, u_i', u_{i-1}', and u_{i-2}' are needed to compute u_{i+1}. The traditional technique for computing starting values has been to use Runge-Kutta formulas of the same accuracy since they only require u_0 to get started. An alternative procedure, which turns out to be more efficient, is to use a sequence of s-step formulas with $s = 1, 2, \ldots, k$ [6]. The computation is started with the one-step formulas in order to provide starting values for the two-step formula and so on. Also, the problem of getting started arises whenever the step-size h is changed. This problem is overcome by using a k-step formula whose coefficients (the b_i's) depend upon the past step-sizes ($h_s = x_s - x_{s-1}$, $s = i, i - 1, \ldots, i - k + 1$) (see [6]). This kind of procedure is currently used in commercial multistep routines.

The previous multistep methods can be derived using polynomials that interpolated at the point x_i and at points backward from x_i. These are sometimes known as formulas of explicit type. Formulas of implicit type can also be derived by basing the interpolating polynomial on the point x_{i+1}, as well as on x_i and points backward from x_i. The simplest formula of this type is obtained if the integral is approximated by the trapezoidal formula. This leads to

$$u_{i+1} = u_i + \frac{h}{2}[f(x_i, u_i) + f(x_{i+1}, u_{i+1})]$$

which is Eq. (1.45). If f is nonlinear, u_{i+1} cannot be solved for directly. However, we can attempt to obtain u_{i+1} by means of iteration. Predict a first approximation $u_{i+1}^{[0]}$ to u_{i+1} by using the Euler method

$$u_{i+1}^{[0]} = u_i + hf_i \tag{1.53}$$

Then compute a corrected value with the trapezoidal formula

$$u_{i+1}^{[s+1]} = u_i + \frac{h}{2}[f_i + f(u_{i+1}^{[s]})], \qquad s = 0, 1, \ldots \tag{1.54}$$

For most problems occurring in practice, convergence generally occurs within one or two iterations. Equations (1.53) and (1.54) used as outlined above define the simplest predictor-corrector method.

Predictor-corrector methods of higher-order accuracy can be obtained by using the multistep formulas such as (1.52) to predict and by using corrector formulas of type

$$u_{i+1} = u_i + h \sum_{j=0}^{k} b_j u'_{i-j+1} \tag{1.55}$$

Notice that j now sums from zero to k. This class of corrector formulas is called the Adams-Moulton correctors. The b_j's of the above equation can be found in a manner similar to those in (1.52). In the case of $k = 2$,

$$u_{i+1} = u_i + \frac{h}{12}[5u'_{i+1} + 8u'_i - u'_{i-1}] \tag{1.56}$$

with a local truncation error of $0(h^4)$. A similar procedure to that outlined for the use of (1.53) and (1.54) is constructed using (1.52) as the predictor and (1.56) as the corrector. The combination (1.52), (1.56) is called the Adams-Moulton predictor-corrector pair of formulas.

Notice that the error in each of the formulas (1.52) and (1.56) is $0(h^4)$. Therefore, if e_{i+1} is to be estimated, the difference

$$|u^*_{i+1} - u_{i+1}|, \qquad u^*_{i+1} \text{ from (1.56)}, \quad u_{i+1} \text{ from (1.52)}$$

would be a poor approximation. More precise expressions for the errors in these formulas are [5]

$$e_{i+1} = \tfrac{3}{8}h^4 y''''(\xi), \qquad \text{for (1.52)}$$

$$e^*_{i+1} = -\tfrac{1}{24}h^4 y''''(\xi^*), \qquad \text{for (1.56)}$$

where $x_{i-2} < \xi$ and $\xi^* < x_{i+1}$. Assume that $\xi^* = \xi$ (this would be a good approximation for small h), then subtract the two expressions.

$$e^*_{i+1} - e_{i+1} = u^*_{i+1} - u_{i+1} = -\tfrac{5}{12}h^4 y''''(\xi)$$

Solving for $h^4 y''''(\xi)$ and substituting into the expression e^*_{i+1} gives

$$|e^*_{i+1}| \simeq \tfrac{1}{10}|u^*_{i+1} - u_{i+1}|$$

Since we had to make a simplifying assumption to obtain this result, it is better to use a more conservative coefficient, say $\frac{1}{8}$. Hence,

$$|e^*_{i+1}| \simeq \tfrac{1}{8}|u^*_{i+1} - u_{i+1}| \tag{1.57}$$

Note that this is an error estimate for the more accurate value so that u^*_{i+1} can be used as the numerical solution rather than u_{i+1}. This type of analysis is not used in the case of Runge-Kutta formulas because the error expressions are very complicated and difficult to manipulate in the above fashion.

Since the Adams-Bashforth method [Eq. (1.51)] is explicit, it possesses poor stability properties. The region of stability for the implicit Adams-Moulton method [Eq. (1.55)] is larger by approximately a factor of 10 than the explicit Adams-Bashforth method, although in both cases the region of stability decreases as k increases (see p. 130 of [4]). For the Adams-Moulton predictor-corrector pair, the exact regions of stability are not well defined, but the stability limitations are less severe than for explicit methods and depend upon the number of corrector iterations [4].

The multistep integration formulas listed above can be represented by the generalized equation:

$$u_{i+1} = \sum_{j=1}^{k_1} a_{i+1,j}\, u_{i-j+1} + h_{i+1} \sum_{j=0}^{k_2} b_{i+1,j}\, u'_{i-j+1} \tag{1.58}$$

which allows for variable step-size through h_{i+1}, $a_{i+1,j}$, and $b_{i+1,j}$. For example, if $k_1 = 1$, $a_{i+1,1} = 1$ for all i, $b_{i+1,j} = b_{i,j}$ for all i, and $k_2 = q - 1$, then a qth-order implicit formula is obtained. Further, if $b_{i+1,0} = 0$, then an explicit formula is generated. Computationally these methods are very efficient. If an explicit formula is used, only a single function evaluation is needed per step. Because of their poor stability properties, explicit multistep methods are rarely used in practice. The use of predictor-corrector formulas does not necessitate the solution of nonlinear equations and requires $S + 1$ (S is the number of corrector iterations) function evaluations per step in x. Since S is usually small, fewer function evaluations are required than from an equivalent order of accuracy Runge-Kutta method and better stability properties are achieved. If a problem requires a large stability region (see section of stiffness), then implicit backward formulas must be used. If (1.58) represents an implicit backward formula, then it is given by

$$u_{i+1} = \sum_{j=1}^{k_1} a_{i+1,j}\, u_{i-j+1} + h_{i+1}\, b_{i+1,0}\, u'_{i+1}$$

or

$$u_{i+1} = b_{i+1,0}\, h_{i+1}\, f(u_{i+1}) + \phi_i \tag{1.59}$$

where ϕ_i is the grouping of all known information. If a Newton iteration is performed on (1.59), then

$$\left[1 - b_{i+1,0}\, h_{i+1} \left.\frac{\partial f}{\partial y}\right|_{u^{[s]}_{i+1}}\right] \left[u^{[s+1]}_{i+1} - u^{[s]}_{i+1}\right]$$

$$= b_{i+1,0}\, h_{i+1} \left. f\right|_{u^{[s]}_{i+1}} + \phi_i - u^{[s]}_{i+1}, \qquad s = 0, 1, \ldots \tag{1.60}$$

Therefore, the derivative $\partial f/\partial y$ must be calculated and the function f evaluated at each iteration. One must "pay" in computation time for the increased stability. The order of accuracy of implicit backward formulas is determined by the value of k_1. As k_1 is increased, higher accuracy is achieved, but at the expense of decreased stability (see Chapter 11 of [4]).

Multistep methods are frequently used in commercial routines because of their combined accuracy, stability, and computational efficiency properties (see section on Mathematical Software). Other high-order methods for handling problems that require large regions of stability are discussed in the following section.

HIGH-ORDER METHODS BASED ON KNOWLEDGE OF $\partial f/\partial y$

A variety of methods that make use of $\partial f/\partial y$ has been proposed to solve problems that require large stability regions. Rosenbrock [7] proposed an extension of the explicit Runge-Kutta process that involved the use of $\partial f/\partial y$. Briefly, if one allows the summation in (1.25) to go from 1 to j, i.e., an implicit Runge-Kutta method, then,

$$\bar{k}_j = hf\left(u_i + \sum_{l=1}^{j} a_{jl}\bar{k}_l\right) \tag{1.61}$$

If $\bar{k}_j$ is expanded,

$$\bar{k}_j = hf\left(u_i + \sum_{l=1}^{j-1} a_{jl}\bar{k}_l\right) + h\frac{\partial f}{\partial y}\left(u_i + \sum_{l=1}^{j-1} a_{jl}\bar{k}_l\right) a_{jj}\bar{k}_j \tag{1.62}$$

and rearranged to give

$$\left[1 - ha_{jj}\frac{\partial f}{\partial y}\left(u_i + \sum_{l=1}^{j-1} a_{jl}\bar{k}_l\right)\right]\bar{k}_j = hf\left(u_i + \sum_{l=1}^{j-1} a_{jl}\bar{k}_l\right) \tag{1.63}$$

the method is called a semi-implicit Runge-Kutta method. In the function f, it is assumed that the independent variable x does not appear explicitly, i.e., it is autonomous. Equation (1.63) is used with

$$u_{i+1} = u_i + \sum_{j=1}^{\nu} \omega_j \bar{k}_j \tag{1.64}$$

to specify the method. Notice that if the bracketed term in (1.63) is replaced by 1, then (1.63) is an explicit Runge-Kutta formula. Calahan [8], Allen [9], and Caillaud and Padmanabhan [10] have developed these methods into algorithms and have shown that they are unconditionally stable with no oscillations in the solution. Stabilization of these algorithms is due to the bracketed term in (1.63). We will return to this semi-implicit method in the section Mathematical Software.

Other methods that are high-order, are stable, and do not oscillate are the

second- and third-order semi-implicit methods of Norsett [11], more recently the diagonally implicit methods of Alexander [12], and those of Bui and Bui [13] and Burka [14].

STIFFNESS

Up to this point we have limited our discussion to a single differential equation. Before looking at systems of differential equations, an important characteristic of systems, called stiffness, is illustrated.

Suppose we wish to model the reaction path $A \underset{k_2}{\overset{k_1}{\rightleftharpoons}} B$ starting with pure A. The reaction path can be described by

$$\frac{dC_A}{dt} = -k_1 C_A + k_2 C_B \tag{1.65}$$

where

$$C_A = C_A^0 \text{ at } t = 0$$

$$C_A = \text{concentration of A}$$

$$t = \text{time}$$

One can define $y_1 = (C_A - C_A^{eq})/(C_A^0 - C_A^{eq})$ where C_A^{eq} is the equilibrium value of C_A $(t \to \infty)$. Equation (1.65) becomes

$$\frac{dy_1}{dt} = -(k_1 + k_2)\, y_1, \qquad y_1 = 1 \quad \text{at} \quad t = 0 \tag{1.66}$$

If $k_1 = 1000$ and $k_2 = 1$, then the solution of (1.66) is

$$y_1 = e^{-1001t} \tag{1.67}$$

If one uses the Euler method to solve (1.66), then

$$h < \tfrac{1}{1001}$$

for stability. The time required to observe the full evolution of the solution is very short. If one now wishes to follow the reaction path $B \overset{k_3}{\to} D$, then

$$\frac{dC_B}{dt} = -k_3 C_B, \qquad C_B = C_B^0 \quad \text{at} \quad t = 0 \tag{1.68}$$

If $k_3 = 1$ and $y_2 = C_B/C_B^0$, then the solution of (1.68) is

$$y_2 = e^{-t} \tag{1.69}$$

If the Euler method is applied to (1.68), then

$$h < 1$$

for stability. The time required to observe the full evolution of the solution is long when compared with that required by (1.66). Next suppose we wish to simulate the reaction pathway

$$A \underset{k_2}{\overset{k_1}{\rightleftharpoons}} B \xrightarrow{k_3} D$$

The governing differential equations are

$$\begin{aligned} \frac{dC_A}{dt} &= -k_1 C_A + k_2 C_B \\ \frac{dC_B}{dt} &= k_1 C_A - (k_2 + k_3) C_B, \\ C_A &= C_A^0,\ C_B = 0 \quad \text{at} \quad t = 0 \end{aligned} \tag{1.70}$$

This system can be written as

$$\frac{d\mathbf{y}}{dt} = \mathcal{Q}\mathbf{y} = \mathbf{f},\ \mathbf{y}(0) = [1, 0]^T \tag{1.71}$$

where

T designates the transpose

$$\mathbf{y} = \left[\frac{C_A}{C_A^0} \quad \frac{C_B}{C_A^0}\right]^T$$

$$\mathcal{Q} = \begin{bmatrix} -k_1 & k_2 \\ k_1 & -(k_2 + k_3) \end{bmatrix}$$

$$\mathbf{f} = [f_1, f_2]^T$$

The solution of (1.71) is

$$\begin{aligned} y_1 &= \tfrac{1000}{1001} e^{-1001t} + \tfrac{1}{1001} e^{-t} \\ y_2 &= -\tfrac{1000}{1001} e^{-1001t} + \tfrac{1000}{1001} e^{-t} \end{aligned} \tag{1.72}$$

A plot of (1.72) is shown in Figure 1.4. Notice that y_1 decays very rapidly, as would (1.67), whereas y_2 requires a long time to trace its full evolution, as would (1.69). If (1.71) is solved by the Euler method

$$h < \frac{1}{|\lambda_{\mathcal{Q}}|_{\max}} \tag{1.73}$$

where $|\lambda_{\mathcal{Q}}|_{\max}$ is the absolute value of the largest eigenvalue of $\mathcal{Q}$. We have the unfortunate situation with systems of equations that the largest step-size is governed by the largest eigenvalue while the integration time for full evolution of the solution is governed by the smallest eigenvalue (slowest decay rate). This property of systems is called stiffness and can be quantified by the stiffness ratio

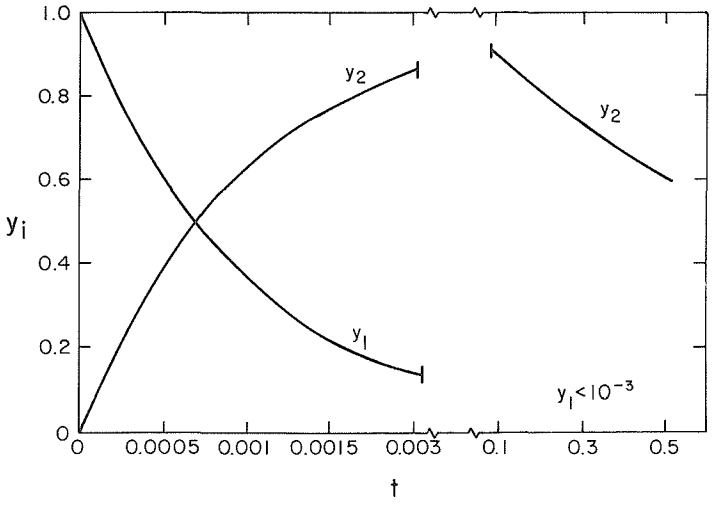

FIGURE 1.4 Results from Eq. (1.72).

[15] SR,

$$\text{SR} = \frac{\max_i |\text{real part of } \lambda_{\mathscr{E}_i}|}{\min_i |\text{real part of } \lambda_{\mathscr{E}_i}|}, \qquad \text{real part of } \lambda_{\mathscr{E}_i} < 0, \qquad i = 1, \ldots, m, \tag{1.74}$$

$$m = \text{number of equations in the system}$$

which allows for imaginary eigenvalues. Typically SR = 20 is not stiff, SR = 10^3 is stiff, and SR = 10^6 is very stiff. From (1.72) SR = $\frac{1001}{1} \simeq 10^3$, and the system (1.71) is stiff. If the system of equations (1.71) were nonlinear, then a linearization of (1.71) gives

$$\frac{d\mathbf{y}}{dt} = Q(t_i)\mathbf{y}(t_i) + J(t_i)(\mathbf{y} - \mathbf{y}(t_i)) \tag{1.75}$$

where

$$\begin{aligned} \mathbf{y}(t_i) &= \text{vector } \mathbf{y} \text{ evaluated at time } t_i \\ Q(t_i) &= \text{matrix } Q \text{ evaluated at time } t_i \\ J(t_i) &= \text{matrix } J \text{ evaluated at time } t_i \end{aligned}$$

$$J = \begin{bmatrix} \dfrac{\partial f_1}{\partial y_1} & \dfrac{\partial f_1}{\partial y_2} \\ \dfrac{\partial f_2}{\partial y_1} & \dfrac{\partial f_2}{\partial y_2} \end{bmatrix}$$

The matrix J is called the Jacobian matrix, and in general is

$$J = \begin{bmatrix} \dfrac{\partial f_1}{\partial y_1}, \dfrac{\partial f_1}{\partial y_2}, \ldots, \dfrac{\partial f_1}{\partial y_m} \\ \vdots \\ \dfrac{\partial f_m}{\partial y_1}, \dfrac{\partial f_m}{\partial y_2}, \ldots, \dfrac{\partial f_m}{\partial y_m} \end{bmatrix}$$

For nonlinear problems the stiffness is based upon the eigenvalues of J and thus applies only to a specific time, and it may change with time. This characteristic of systems makes a problem both interesting and difficult. We need to classify the stiffness of a given problem in order to apply techniques that "perform" well for that given magnitude of stiffness. Generally, implicit methods "outperform" explicit methods on stiff problems because of their less rigid stability criterion. Explicit methods are best suited for nonstiff equations.

SYSTEMS OF DIFFERENTIAL EQUATIONS

A straightforward extension of (1.11) to a system of equations is

$$\begin{aligned} \mathbf{u}_{i+1} &= \mathbf{u}_i + h\mathbf{f}(x_i, \mathbf{u}_i), \qquad i = 0, 1, \ldots, N-1 \\ \mathbf{u}_0 &= \mathbf{y}_0 \end{aligned} \tag{1.76}$$

Likewise, the implicit Euler becomes

$$\begin{aligned} \mathbf{u}_{i+1} &= \mathbf{u}_i + h\mathbf{f}(x_i, \mathbf{u}_{i+1}), \qquad i = 0, 1, \ldots, N-1 \\ \mathbf{u}_0 &= \mathbf{y}_0 \end{aligned} \tag{1.77}$$

while the trapezoid rule gives

$$\begin{aligned} \mathbf{u}_{i+1} &= \mathbf{u}_i + \frac{h}{2}[\mathbf{f}(x_i, \mathbf{u}_{i+1}) + \mathbf{f}(x_i, \mathbf{u}_i)], \qquad i = 0, 1, \ldots, N-1 \\ \mathbf{u}_0 &= \mathbf{y}_0 \end{aligned} \tag{1.78}$$

For a system of equations the Runge-Kutta-Fehlberg method is

$$\begin{aligned} \mathbf{u}_{i+1} &= \mathbf{u}_i + \left[\tfrac{25}{216}\mathbf{k}_1 + \tfrac{1408}{2565}\mathbf{k}_3 + \tfrac{2197}{4104}\mathbf{k}_4 - \tfrac{1}{5}\mathbf{k}_5\right] \\ \mathbf{u}^*_{i+1} &= \mathbf{u}_i + \left[\tfrac{16}{135}\mathbf{k}_1 + \tfrac{6656}{12825}\mathbf{k}_3 + \tfrac{28561}{56430}\mathbf{k}_4 - \tfrac{9}{50}\mathbf{k}_5 + \tfrac{2}{55}\mathbf{k}_6\right] \end{aligned} \tag{1.79}$$

where

$$\mathbf{k}_i = \left[k_i^{\{1\}}, k_i^{\{2\}}, \ldots, k_i^{\{m\}}\right]^T$$

and, for example,

$$
\begin{aligned}
k_1^{\{j\}} &= hf_j(x_i, u_i^{\{1\}}, u_i^{\{2\}}, \ldots, u_i^{\{m\}}), \qquad j = 1, \ldots, m \\
\phi_{ji} &= u_i^{\{j\}} + \tfrac{1}{4}k_1^{\{j\}} \\
k_2^{\{j\}} &= hf_j(x_i + \tfrac{1}{4}h, \phi_{1i}, \phi_{2i}, \ldots, \phi_{mi}), \qquad j = 1, \ldots, m \\
&\vdots
\end{aligned}
$$

The Adams-Moulton predictor-corrector formulas for a system of equations are

$$
\begin{aligned}
\mathbf{u}_{i+1} &= \mathbf{u}_i + \frac{h}{12}[23\mathbf{u}_i' - 16\mathbf{u}_{i-1}' + 5\mathbf{u}_{i-2}'] \\
\mathbf{u}_{i+1}^* &= \mathbf{u}_i + \frac{h}{12}[5\mathbf{u}_{i+1}' + 8\mathbf{u}_i' - \mathbf{u}_{i-1}']
\end{aligned}
\tag{1.80}
$$

An algorithm using the higher-order method of Caillaud and Padmanabhan [10] was formulated by Michelsen [16] by choosing the parameters in (1.63) so that the same factor multiplies each $\bar{\mathbf{k}}_i$, thus minimizing the work involved in matrix inversion. The final scheme is

$$
\begin{aligned}
\mathbf{u}_{i+1} &= \mathbf{u}_i + \omega_1\bar{\mathbf{k}}_1 + \omega_2\bar{\mathbf{k}}_1 + \omega_3\bar{\mathbf{k}}_3, \qquad i = 0, 1, \ldots, N-1 \\
\mathbf{u}_0 &= \mathbf{y}_0 \\
\bar{\mathbf{k}}_1 &= h\left[I - ha_1\frac{\partial \mathbf{f}}{\partial \mathbf{y}}(\mathbf{u}_i)\right]^{-1}\mathbf{f}(\mathbf{u}_i) \\
\bar{\mathbf{k}}_2 &= h\left[I - ha_1\frac{\partial \mathbf{f}}{\partial \mathbf{y}}(\mathbf{u}_i)\right]^{-1}\mathbf{f}(\mathbf{u}_i + b_2\bar{\mathbf{k}}_1) \\
\bar{\mathbf{k}}_3 &= h\left[I - ha_1\frac{\partial \mathbf{f}}{\partial \mathbf{y}}(\mathbf{u}_i)\right]^{-1}(b_{31}\bar{\mathbf{k}}_1 + b_{32}\bar{\mathbf{k}}_2)
\end{aligned}
\tag{1.81}
$$

where I is the identity matrix,

$$
\begin{aligned}
a_1 &= 0.43586659 \\
b_2 &= 0.75 \\
b_{31} &= \frac{-1}{6a_1}(8a_1^2 - 2a_1 + 1) \\
b_{32} &= \frac{2}{9a_1}(6a_1^2 - 6a_1 + 1) \\
\omega_1 &= \tfrac{11}{27} - b_{31} \\
\omega_2 &= \tfrac{16}{27} - b_{32} \\
\omega_3 &= 1.0
\end{aligned}
$$

As previously stated, the independent variable x must not explicitly appear in $\mathbf{f}$. If x does explicitly appear in $\mathbf{f}$, then one must reformulate the system of equations by introducing a new integration variable, t, and let

$$\frac{dx}{dt} = 1 \tag{1.82}$$

be the $(m + 1)$ equation in the system.

EXAMPLE 5

Referring to Example 1, if we now consider the reactor to be adiabatic instead of isothermal, then an energy balance must accompany the material balance. Formulate the system of governing differential equations and evaluate the stiffness. Write down the Euler and the Runge-Kutta-Fehlberg methods for this problem.

Data

$$C_p = 12.17 \times 10^4 \text{ J/(kmole·°C)}$$

$$-\Delta H_r = 2.09 \times 10^8 \text{ J/kmole}$$

SOLUTION

Let $T^* = T/T°$, $T° = 423$ K (150°C). For the "short" reactor,

$$\frac{dy}{dx} = -0.1744 \exp\left[\frac{3.21}{T^*}\right] y \qquad \text{(material balance)}$$

$$\frac{dT^*}{dx} = 0.06984 \exp\left[\frac{3.21}{T^*}\right] y \qquad \text{(energy balance)}$$

$$y = 1, \; T^* = 1 \quad \text{at} \quad x = 0$$

First, check to see if stiffness is a problem. To do this the transport equations can be linearized and the Jacobian matrix formed.

$$\mathrm{J} = \begin{bmatrix} -0.1744 \exp\left(\dfrac{3.21}{T^*}\right) & \dfrac{0.56}{(T^*)^2} \exp\left(\dfrac{3.21}{T^*}\right) y \\ 0.06984 \exp\left(\dfrac{3.21}{T^*}\right) & \dfrac{-0.224}{(T^*)^2} \exp\left(\dfrac{3.21}{T^*}\right) y \end{bmatrix}$$

At the inlet $T^* = 1$ and $y = 1$, and the eigenvalues of $\boldsymbol{J}$ are approximately $(6.3, -7.6)$. Since T^* should increase as y decreases, for example, if $T^* = 1.12$ and $y = 0.5$, then the eigenvalues of $\boldsymbol{J}$ are approximately $(3.0, -4.9)$. From the stiffness ratio, one can see that this problem is not stiff.

Euler:

$$u_{i+1}^{\{1\}} = u_i^{\{1\}} - 0.1744 \exp\left[\frac{3.21}{u_i^{\{2\}}}\right] u_i^{\{1\}} h$$

$$u_{i+1}^{\{2\}} = u_i^{\{2\}} + 0.06984 \exp\left[\frac{3.21}{u_i^{\{2\}}}\right] u_i^{\{1\}} h$$

$$u_0^{\{1\}} = 1$$

$$u_0^{\{2\}} = 1$$

Runge-Kutta-Fehlberg:

$$u_{i+1}^{\{1\}} = u_i^{\{1\}} + [C1 \cdot k_1^{\{1\}} + C2 \cdot k_3^{\{1\}} + C3 \cdot k_4^{\{1\}} + C4 \cdot k_5^{\{1\}}]$$

$$u_{i+1}^{\{2\}} = u_i^{\{2\}} + [C1 \cdot k_1^{\{2\}} + C2 \cdot k_3^{\{2\}} + C3 \cdot k_4^{\{2\}} + C4 \cdot k_5^{\{2\}}]$$

$$u_{i+1}^{\{1\}*} = u_i^{\{1\}} + [C5 \cdot k_1^{\{1\}} + C6 \cdot k_3^{\{1\}} + C7 \cdot k_4^{\{1\}} + C8 \cdot k_5^{\{1\}} + C9 \cdot k_6^{\{1\}}]$$

$$u_{i+1}^{\{2\}*} = u_i^{\{2\}} + [C5 \cdot k_1^{\{2\}} + C6 \cdot k_3^{\{2\}} + C7 \cdot k_4^{\{2\}} + C8 \cdot k_5^{\{2\}} + C9 \cdot k_6^{\{2\}}]$$

$$C1 = \tfrac{25}{216}, \qquad C5 = \tfrac{16}{135}$$

$$C2 = \tfrac{1408}{2565}, \qquad C6 = \tfrac{6656}{12825}$$

$$C3 = \tfrac{2197}{4104}, \qquad C7 = \tfrac{28561}{56430}$$

$$C4 = -\tfrac{1}{5}, \qquad C8 = -\tfrac{9}{50}$$

$$C9 = \tfrac{9}{55}$$

Define

$$F1(\mathrm{A}, \mathrm{B}) = -0.1744 \exp\left[\frac{3.21}{\mathrm{B}}\right] \mathrm{A}$$

$$F2(\mathrm{A}, \mathrm{B}) = 0.06984 \exp\left[\frac{3.21}{\mathrm{B}}\right] \mathrm{A}$$

then

$$k_1^{\{1\}} = hF1(u_i^{\{1\}}, u_i^{\{2\}})$$

$$k_1^{\{2\}} = hF2(u_i^{\{1\}}, u_i^{\{2\}})$$

$$k_2^{\{1\}} = hF1(u_i^{\{1\}} + \tfrac{1}{4}k_1^{\{1\}}, u_i^{\{2\}} + \tfrac{1}{4}k_1^{\{2\}})$$

$$k_2^{\{2\}} = hF2(u_i^{\{1\}} + \tfrac{1}{4}k_1^{\{1\}}, u_i^{\{2\}} + \tfrac{1}{4}k_1^{\{2\}})$$

$$\vdots$$

STEP-SIZE STRATEGIES

Thus far we have only concerned ourselves with constant step-sizes. Variable step-sizes can be very useful for (1) controlling the local truncation error and (2) improving efficiency during solution of a stiff problem. This is done in all of the commercial programs, so we will discuss each of these points in further detail.

Gear [4] estimates the local truncation error and compares it with a desired error, TOL. If the local truncation error has been achieved using a step-size h_1,

$$e = \phi h_1^{p+1} \tag{1.83}$$

Since we wish the error to equal TOL,

$$\text{TOL} = \phi h_2^{p+1} \tag{1.84}$$

Combination of (1.83) and (1.84) gives

$$h_2 = h_1 \left[\frac{\text{TOL}}{e}\right]^{1/(p+1)} \tag{1.85}$$

Equation (1.83) is method-dependent, so we will illustrate the procedure with a specific method. If we solve a given problem using the Euler method,

$$u_{i+1} = u_i + h_1 f(u_i) \tag{1.86}$$

and the implicit Euler,

$$\omega_{i+1} = \omega_i + h_1 f(\omega_{i+1}) \tag{1.87}$$

and subtract (1.86) and (1.87) from (1.10) and (1.38), respectively (assuming $u_i = y_i$), then

$$\begin{aligned} u_{i+1} - y(x_{i+1}) &= -\tfrac{1}{2}h_1^2 f_i' + 0(h_1^3) \\ \omega_{i+1} - y(x_{i+1}) &= \tfrac{1}{2}h_1^2 f_i' + 0(h_1^3) \end{aligned} \tag{1.88}$$

The truncation error can now be estimated by

$$e_{i+1} = \omega_{i+1} - y(x_{i+1}) \simeq \tfrac{1}{2}(\omega_{i+1} - u_{i+1}) \tag{1.89}$$

The process proceeds as follows:

1. Equations (1.86) and (1.87) are used to obtain u_{i+1} and ω_{i+1}.
2. The truncation error is obtained from (1.89).
3. If the truncation error is less than TOL, the step is accepted; if not, the step is repeated.
4. In either case of step(3), the next step-size is calculated according to

$$h_2 = h_1 \left(\frac{\text{TOL}}{e_{i+1}}\right)^{1/2} \tag{1.90}$$

To avoid small errors, one can use an h_2 that is a certain percentage smaller than calculated by (1.90).

Michelsen [16] solved (1.81) with a step-size of h and then again with $h/2$. The semi-implicit algorithm is third-order accurate, so it may be written as

$$\mathbf{u}_{i+1} = \mathbf{y}(x_{i+1}) + \mathbf{g}h^4 + 0(h^5) \tag{1.91}$$

where $\mathbf{g}h^4$ is the dominant, but still unknown, error term. If $\mathbf{u}_{i+1}$ denotes the numerical solution for a step-size of h, and $\boldsymbol{\omega}_{i+1}$ for a step-size of $h/2$, then,

$$\begin{aligned} \mathbf{u}_{i+1} &= \mathbf{y}(x_{i+1}) + \mathbf{g}h^4 + 0(h^5) \\ \boldsymbol{\omega}_{i+1} &= \mathbf{y}(x_{i+1}) + 2\mathbf{g}\left(\frac{h}{2}\right)^4 + 0(h^5) \end{aligned} \tag{1.92}$$

where the $2\mathbf{g}$ in (1.92) accounts for error accumulation in each of the two integration steps. Subtraction of the two equations (1.92) from one another gives

$$\mathbf{e}_{i+1} = \boldsymbol{\omega}_{i+1} - \mathbf{u}_{i+1} = -\tfrac{7}{8}\mathbf{g}h^4 + 0(h^5) \tag{1.93}$$

Provided $\mathbf{e}_{i+1}$ is sufficiently small, the result is accepted. The criterion for step-size acceptance is

$$e^{\{j\}} < \mathrm{TOL}^{\{j\}}, \qquad j = 1, 2, \ldots, m \tag{1.94}$$

where

$$e^{\{j\}} = \text{local truncation error for the } j \text{ component}$$

If this criterion is not satisfied, the step-size is halved and the integration repeated. When integrating stiff problems, this procedure leads to small steps whenever the solution changes rapidly, often times at the start of the integration. As soon as the stiff component has faded away, one observes that the magnitude of $\mathbf{e}$ decreases rapidly and it becomes desirable to increase the step-size. After a successful step with h_i, the step-size h_{i+1} is adjusted by

$$h_{i+1} = h_i \min\left[\left\{4 \max\left|\frac{e^{\{j\}}}{\mathrm{TOL}^{\{j\}}}\right|\right\}^{-1/4}, 3\right], \qquad j = 1, 2, \ldots, m \tag{1.95}$$

For more explanation of (1.95) see [17]. A good discussion of computer algorithms for adjusting the step-size is presented by Johnston [5] and by Krogh [18].

We are now ready to discuss commercial packages that incorporate a variety of techniques for solving systems of IVPs.

MATHEMATICAL SOFTWARE

Most computer installations have preprogrammed computer packages, i.e., software, available in their libraries in the form of subroutines so that they can be accessed by the user's main program. A subroutine for solving IVPs will be designed to compute a numerical solution over $[x_0, x_N]$ and return the value u_N

given x_0, x_N, and u_0. A typical calling sequence could be

CALL DRIVE (FUNC, X, XEND, U, TOL),

where

FUNC = a user-written subroutine for evaluating $f(x, y)$
X = x_0
XEND = x_N
U = on input contains u_0 and on output contains u_N
TOL = an error tolerance

This is a very simplified call sequence, and more elaborate ones are actually used in commercial routines.

The subroutine DRIVE must contain algorithms that:

1. Implement the numerical integration
2. Adapt the step-size
3. Calculate the local error so as to implement item 2 such that the global error does not surpass TOL
4. Interpolate results to XEND (since h is adaptively modified, it is doubtful that XEND will be reached exactly)

Thus, the creation of a software package, from now on called a code, is a nontrivial problem. Once the code is completed, it must contain sufficient documentation. Several aspects of documentation are significant (from [24]):

1. Comments in the code identifying arguments and providing general instructions to the user (this is valuable because often the code is separated from the other documentation)
2. A document with examples showing how to use the code and illustrating user-oriented aspects of the code
3. Substantial examples of the performance of the code over a wide range of problems
4. Examples showing misuse, subtle and otherwise, of the code and examples of failure of the code in some cases.

Most computer facilities have at least one of the following mathematical libraries:

IMSL [19]
NAG [20]
HARWELL [21]

The Harwell library contains several IVP codes, IMSL has two (which will be discussed below), and NAG contains an extensive collection of routines. These large libraries are not the only sources of codes, and in Table 1.7 we provide a survey of IVP software (excluding IMSL, Harwell, and NAG). Since the production of software has increased tremendously during recent years, any survey of codes will need continual updating. Table 1.7 should provide the reader with an appreciation for the types of codes that are being produced, i.e., the underlying numerical methods. We do not wish to dwell on all of these codes but only to point out a few of the better ones. Recently, a survey of IVP software [33] concluded that RKF45 is the best overall explicit Runge-Kutta routine, while LSODE is quite good for solving stiff problems. LSODE is the update for GEAR/GEARB (versions of which are presently the most used stiff IVP solver) [34].

The comparison of computer codes is a difficult and tricky task, and the results should always be "taken with a grain of salt." Hull et al. [35] have compared nonstiff methods, while Enright et al. [36] compared stiff ones. Although this is an important step, it does not bear directly on how practical a code is. Shampine et al. [37] have shown that how a method is implemented

TABLE 1.7 IVP Codes

Name	Method Implemented	Comments	Reference
RKF45	Runge-Kutta-Fehlberg	—	[22]
GERK	Runge-Kutta-Fehlberg	—	[23]
DE/ODE	Variable-order Adams multistep	DE is limited to 20 equations or less: ODE has no size limit	[6]
DEROOT/ODERT	Variable-order Adams multistep	Same as DE/ODE except that nonlinear scalar equations can be coupled to the IVPs	[6]
GEAR/GEARB	Variable-order Adams multistep and backward multistep	Allow for nonstiff Adams and stiff backward formulas; GEARB allows for banded structure of the Jacobian	[24], [25]
LSODE	—	Replacement for GEAR/GEARB	[26]
EPISODE/EPISODEB	Same as GEAR/GEARB	Differ from GEAR/GEARB in how the variable step-size is performed	[27]
M3RK	Stabilized explicit Runge-Kutta*	Designed to solve systems arising from a method of lines discretization of partial differential equations	[28]
STRIDE	Implicit Runge-Kutta	—	[29]
STIFF3	Semi-implicit Runge-Kutta	See text; Eq. (1.81) with (1.95)	[17]
BLSODE	Blended multistep*	For stiff oscillatory problems	[30]
STINT	Cyclic composite multistep*	—	[31]
SECDER	Variable-order Enright formula*	—	[32]

*Method not covered in this chapter.

may be more important than the choice of method, even when dealing with the best codes. There is a distinction between the best methods and the best codes. In [31] various codes for nonstiff problems are compared, and in [38] GEAR and EPISODE are compared by the authors. One major aspect of code usage that cannot be tested is the user's attitude, including such factors as user time constraints, accessibility of the code, familiarity with the code, etc. It is typically the user's attitude which dictates the code choice for a particular problem, not the question of which is the best code. Therefore, no sophisticated code comparison will be presented. Instead, we illustrate the use of software packages by solving two problems. These problems are chosen to demonstrate the concept of stiffness.

The following codes were used in this study:

1. IMSL-DVERK: Runge-Kutta solver.
2. IMSL-DGEAR: This code is a modified version of GEAR. Two methods are available in this package: a variable-order Adams multistep method and a variable-order implicit multistep method. Implicit methods require Jacobian calculations, and in this package the Jacobian can be (a) user-supplied, (b) internally calculated by finite differences, or (c) internally calculated by a diagonal approximation based on the directional derivative (for more explanation see [24]). The various methods are denoted by the parameter MF, where

MF	Method	Jacobian
10	Adams	—
21	Implicit	User-supplied
22	Implicit	Finite differences
23	Implicit	Diagonal approximation

3. STIFF3: Implements (1.81) using (1.94) and (1.95) to govern the step-size and error.
4. LSODE: updated version of GEAR. The parameter MF is the same as for DGEAR. MF = 23 is not an option in this package.
5. EPISODE: A true variable step-size code based on GEAR. GEAR, DGEAR, and LSODE periodically change the step-size (not on every step) in order to decrease execution time while still maintaining accuracy. EPISODE adapts the step-size on every step (if necessary) and is therefore good for problems that involve oscillations. For decaying or linear problems, EPISODE would probably require larger execution times than GEAR, DGEAR, or LSODE.
6. ODE: Variable-order Adams multistep solver.

We begin our discussions by solving the reactor problem outlined in Example 5:

$$\frac{dy}{dx} = -0.1744 \exp\left[\frac{3.21}{T^*}\right] y$$

$$\frac{dT^*}{dx} = 0.06984 \exp\left[\frac{3.21}{T^*}\right] y \tag{1.96}$$

$$y = T^* = 1 \quad \text{at} \quad x = 0$$

Equations (1.96) are not stiff (see Example 1.5), and all of the codes performed the integration with only minor differences in their solutions. Typical results are shown in Table 1.8. Notice that a decrease in TOL when using DVERK did produce a change in the results (although the change was small). Practically speaking, any of the solutions presented in Table 1.8 would be acceptable. From the discussions presented in this chapter, one should realize that DVERK, ODE, DGEAR (MF = 10), LSODE (MF = 10), and EPISODE (MF = 10) use methods that are capable of solving nonstiff problems, while STIFF3, DGEAR (MF = 21, 22, 23), LSODE (MF = 21, 22), and EPISODE (MF = 21, 22, 23) implement methods for solving stiff systems. Therefore, all of the codes are suitable for solving (1.96). One might expect the stiff problem solvers to require longer execution times because of the Jacobian calculations. This behavior was observed, but since (1.96) is a small system, i.e., two equations, the execution times for all of the codes were on the same order of magnitude. For a larger problem the effect would become significant.

Next, we consider a stiff problem. Robertson [39] originally proposed the

TABLE 1.8 Typical Results from Software Packages Using Eq. (1.96)

	DVERK, TOL = (−4)		DVERK, TOL = (−6)		DGEAR (MF = 21), TOL = (−4)		STIFF3, TOL = (−4)	
x	y	T^*	y	T^*	y	T^*	y	T^*
0.0	1.000000	1.00000	1.000000	1.00000	1.000000	1.00000	1.000000	1.00000
0.1	0.699795	1.12021	0.700367	1.11999	0.700468	1.11994	0.700371	1.11998
0.2	0.528839	1.18868	0.529199	1.18853	0.529298	1.18849	0.529208	1.18853
0.3	0.413483	1.23487	0.413737	1.23477	0.413775	1.23475	0.413745	1.23477
0.4	0.329730	1.26841	0.329919	0.26833	0.329864	1.26836	0.329924	1.26833
0.5	0.266347	1.29379	0.266492	1.29373	0.266349	1.29379	0.266497	1.29373
0.6	0.217094	1.31352	0.217208	1.31347	0.217070	1.31353	0.217211	1.31347
0.7	0.178118	1.32912	0.178209	1.32909	0.178076	1.32914	0.178212	1.32909
0.8	0.146869	1.34164	0.146943	1.34161	0.146801	1.34167	0.146945	1.34161
0.9	0.121569	1.35177	0.121629	1.35175	0.121495	1.35180	0.121630	1.35175
1.0	0.100931	1.36003	0.100980	1.36002	0.100864	1.36006	0.100982	1.36001

following set of differential equations that arise from an autocatalytic reaction pathway:

$$\begin{aligned} \frac{dy_1}{dt} &= -0.04y_1 + 10^4 y_2 y_3 \\ \frac{dy_2}{dt} &= 0.04y_1 - 10^4 y_2 y_3 - 3 \times 10^7 y_2^2 \\ \frac{dy_3}{dt} &= 3 \times 10^7 y_2^2 \\ y_1(0) &= 1, \quad y_2(0) = 0, \quad y_3(0) = 0 \quad \text{at} \quad t = 0 \end{aligned} \tag{1.97}$$

The Jacobian matrix is

$$J = \begin{bmatrix} -0.04 & 10^4 y_3 & 10^4 y_2 \\ 0.04 & -10^4 y_3 - 6 \times 10^7 y_2 & -10^4 y_2 \\ 0 & 6 \times 10^7 y_2 & 0 \end{bmatrix} \tag{1.98}$$

When t varies from 0.0 to 0.02, one of the eigenvalues of J changes from -0.04 to $-2{,}450$. Over the complete range of t, $0 \leq t \leq \infty$, one of the eigenvalues varies from -0.04 to -10^4. Figure 1.5 shows the solution of (1.97) for $0 \leq t \leq 10$. Notice the steep gradient in y_2 at small values of t. Thus the problem is very stiff. Caillaud and Padmanabhan [10], Seinfeld et al. [40], Villadsen and Michelsen [17], and Finlayson [41] have discussed this problem. Table 1.9 shows the

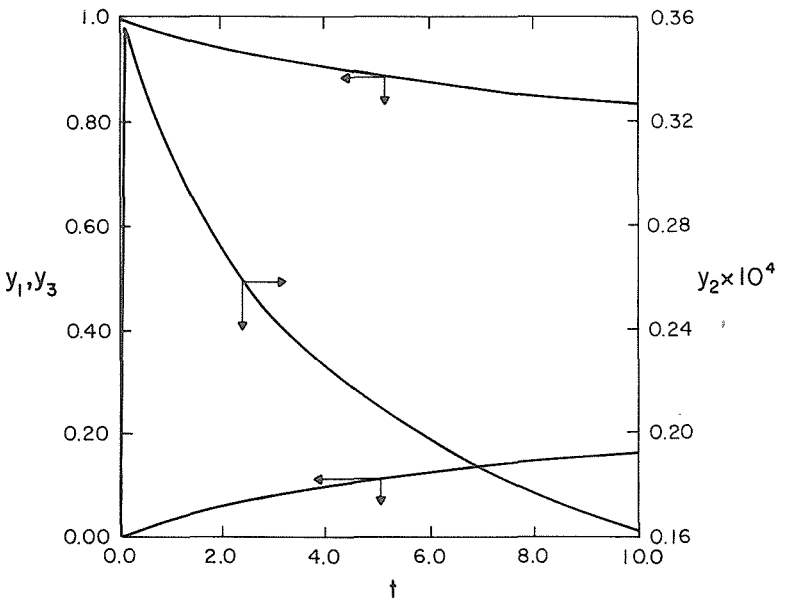

FIGURE 1.5 Results from Eq. (1.97).

results at $t = 10$. At a TOL of 10^{-4} all of the nonstiff methods failed to produce a solution. At smaller tolerance values, the nonstiff methods failed to produce a solution or required excessive execution times, i.e., two orders of magnitude greater than those of the stiff methods. This behavior is due to the fact that the tolerances were too large to achieve a stable solution (recall that the step-size is adapted to meet the error criterion that is governed by the value of TOL) or a solution was obtained but at a high cost (large execution time) because of the very small step-size requirements of nonstiff methods (see section on stiffness).

TABLE 1.9 Comparison of Software Packages on the Robertson Problem (Results at $t = 10$)

Code	MF	TOL	y_1	$y_2 \times 10^4$	y_3	Execution Time Ratio†
DVERK	—	(−4)	—	No solution	—	—
DVERK	—	(−6)	—	No solution	—	—
DVERK	—	(−8)	—	No solution	—	—
ODE	—	(−4)	—	No solution	—	—
ODE	—	(−6)	0.8411	0.1586	0.1589	339.0
ODE	—	(−9)	0.8414	0.1623	0.1586	347.0
DGEAR	10	(−4)	—	No solution	—	—
DGEAR	21	(−4)	0.8414	0.1624	0.1586	0.25
DGEAR	22	(−4)	0.8414	0.1624	0.1586	1.0
DGEAR	23	(−4)	—	No solution	—	—
DGEAR	10	(−6)	0.8414	0.1619	0.1586	261.0
DGEAR	21	(−6)	0.8414	0.1623	0.1586	1.0
DGEAR	22	(−6)	0.8414	0.1623	0.1586	1.0
DGEAR	23	(−6)	0.8414	0.1624	0.1586	2.5
LSODE	10	(−4)	—	No solution	—	—
LSODE	21	(−4)	—	No solution	—	—
LSODE	22	(−4)	—	No solution	—	—
LSODE‡	10	(−4)	—	No solution	—	—
LSODE‡	21	(−4)	0.8414	0.1623	0.1586	1.75
LSODE‡	22	(−4)	0.8414	0.1623	0.1586	1.75
LSODE	10	(−6)	—	No solution	—	—
LSODE	21	(−6)	0.8414	0.1623	0.1586	1.75
LSODE	22	(−6)	0.8414	0.1623	0.1586	1.75
EPISODE	10	(−4)	—	No solution	—	—
EPISODE	21	(−4)	—	No solution	—	—
EPISODE	22	(−4)	—	No solution	—	—
EPISODE	23	(−4)	—	No solution	—	—
EPISODE	10	(−6)	0.8414	0.1623	0.1586	530.0
EPISODE	21	(−6)	0.8414	0.1623	0.1586	1.5
EPISODE	22	(−6)	0.8414	0.1623	0.1586	1.5
EPISODE	23	(−6)	0.8414	0.1623	0.1586	3.8
STIFF3	—	(−4)	0.8414	0.1623	0.1586	1.25
STIFF3	—	(−6)	0.8414	0.1623	0.1586	3.0
"EXACT"§	—	—	0.841	0.162	0.159	—

†Execution time ratio = execution time/execution time of DGEAR [MF = 21, TOL = (−6)].

‡Tolerance for y_2 is (−8); for y_1 and y_3, (−4).

§Caillaud and Padmanabhan [10].

TABLE 1.10 Comparison of Code Results to the "Exact" Solution for Time = 1, 4, and 10

Code	MF	TOL	t	y_1	$y_2 \times 10^4$	y_3
"EXACT"	—	—	1.0	0.966	0.307	0.0335
			4.0	0.9055	0.224	0.0944
			10.0	0.841	0.162	0.159
STIFF3	—	(−6)	1.0	0.9665	0.3075	0.3351(−1)
			4.0	0.9055	0.2240	0.9446(−1)
			10.0	0.8414	0.1623	0.1586
EPISODE	21	(−6)	1.0	0.9665	0.3075	0.3351(−1)
			4.0	0.9055	0.2240	0.9446(−1)
			10.0	0.8414	0.1623	0.1586
DGEAR	10	(−6)	1.0	0.9665	0.3087	0.3350(−1)
			4.0	0.9055	0.2238	0.9445(−1)
			10.0	0.8414	0.1619	0.1586
DGEAR	21	(−6)	1.0	0.9665	0.3075	0.3351(−1)
			4.0	0.9055	0.2240	0.9446(−1)
			10.0	0.84414	0.1623	0.1586
ODE	—	(−6)	1.0	0.9665	0.3075	0.3351(−1)
			4.0	0.9055	0.2222	0.9452(−1)
			10.0	0.8411	0.1586	0.1589

All of the stiff algorithms were able to produce solutions with execution times on the same order of magnitude. Caillaud and Padmanabhan [10] have studied (1.97) using Runge-Kutta algorithms. Their "exact" results (fourth-order Runge-Kutta with step-size = 0.001) and the results obtained from various codes are presented in Table 1.10. Notice that when a solution was obtained from either a stiff or a nonstiff algorithm, the results were excellent. Therefore, the difference between the stiff and nonstiff algorithms was their execution times.

The previous two examples have illustrated the usefulness of the commercial software packages for the solution of practical problems. It can be concluded that generally one should use a package that incorporates an implicit method for stiff problems and an explicit method for nonstiff problems (this was stated in the section on stiffness, but no examples were given).

We hope to have eliminated the "blackbox" approach to the use of initial-value packages through the illustration of the basic methods and rationale behind the production of these programs. No code is infallible, and when you obtain spurious results from a code, you should be able to rationalize your data with the aid of the code's documentation and the material presented in this chapter.

PROBLEMS*

1. A tubular reactor for a homogeneous reaction has the following dimensions: $L = 2$ m, $R = 0.1$ m. The inlet reactant concentration is $c_0 = 0.03$ kmole/m^3, and the inlet temperature is $T_0 = 700$ K. Other

* See the Preface regarding classes of problems.

data is as follows: $-\Delta H = 10^4$ kJ/kmole, $c_p = 1$ kJ/(kg·K), $E_a = 100$ kJ/kmole, $\rho = 1.2$ kg/m^3, $u_0 = 3$ m/s, and $k_0 = 5s^{-1}$. The appropriate material and energy balance equations are (see [17] for further explanation):

$$\frac{dy}{dz} = -\mathrm{Da}\, y \exp\left[\delta\left(1 - \frac{1}{\theta}\right)\right], \qquad 0 \leq z \leq 1,$$

$$\frac{d\theta}{dz} = \beta \mathrm{Da}\, y \exp\left[\delta\left(1 - \frac{1}{\theta}\right)\right] - H_w(\theta - \theta_w)$$

where

$$\mathrm{Da} = \frac{Lk_0}{u_0}$$

$$\beta = \frac{c_0(-\Delta H)}{\rho c_p T_0}$$

$$\delta = \frac{E_a}{R_g T_0}$$

$$H_w = \frac{2\overline{U}}{R}\left(\frac{L}{\rho c_p u_0}\right)$$

$$y = \frac{c}{c_0}$$

$$\theta = \frac{T}{T_0}$$

If one considers the reactor to be adiabatic, $\overline{U} = 0$, the transport equations can be combined to

$$\frac{d}{dz}(\theta + \beta y) = 0$$

which gives

$$\theta = 1 + \beta(1 - y)$$

using the inlet conditions $\theta = y = 1$. Substitution of this equation into the material balance yields

$$\frac{dy}{dz} = -\mathrm{Da}\, y \exp\left[\frac{\delta\beta(1 - y)}{1 + \beta(1 - y)}\right], \qquad y = 1 \quad \text{at} \quad z = 0$$

(a) Compute y and θ if $\overline{U} = 0$ using an Euler method.

(b) Repeat (a) using a Runge-Kutta method.

(c) Repeat (a) using an implicit method.

(d) Check algorithms (a) to (c) by comparing their solutions to the analytical solution by letting $\delta = 0$.

2. Write a subroutine called EULER such that the call is

CALL EULER (FUNC, XO, XOUT, H, TOL, N, Y),

where

FUNC = external subroutine to calculate the right-hand-side functions
XO = initial value of the independent variable
XOUT = final value of the independent variable
H = initial step-size
TOL = local error tolerance
N = number of equations to be integrated
Y = vector with N components for the dependent variable y. On input y is the vector initial values, on output it contains the computed values of y at XOUT.

The routine is to perform an Euler integration on

$$\frac{dy}{dx} = f(x, y)$$

$$y(0) = y_0, \qquad XO \leq X \leq XOUT$$

Create this algorithm such that it contains an error-checking routine and a step-size selection routine. Test your routine by solving Example 5. Hopefully, this problem will give the reader some feel for the difficulty in creating a general-purpose routine.

3.* Repeat Problem 1, but now allow for heat transfer by letting $\bar{U} = 70$ J/(m^2·s·K). Locate the position of the hot spot, θ_{max}, with $\theta_w = 1$.

4.* In Example 4 we evaluated a binary batch distillation system. Now consider the same system with recycle (R = recycle ratio) and a constant condenser hold-up M (see Figure 1.6).

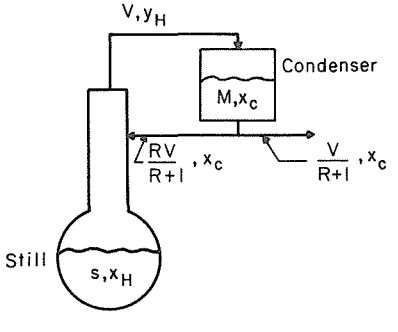

FIGURE 1.6 Batch still with recycle.

A mass balance on *n*-heptane for the condenser is

$$M\frac{dx_c}{dt} = V(y_H - x_c)$$

An overall balance on the still gives

$$\frac{ds}{dt} = \frac{-V}{R+1}$$

while an overall balance on *n*-heptane is

$$s\frac{dx_H}{dt} = \frac{-V}{R+1}(x_c - x_H)$$

Repeat the calculations of Example 1.4 with $s = 100$, $x_H = 0.75$, and $x_c = 0.85$ at $t = 0$. Let $R = 0.3$ and $M = 10$.

5.* Consider the following process where steam passes through a coil, is condensed, and is withdrawn as condensate at the same temperature in order to heat liquid in a well-stirred vessel (see Figure 1.7).
If

F_s = flow rate of steam
H_v = latent heat of vaporization of the steam
F = flow rate of liquid to be heated
T_0 = inlet liquid temperature
T = outlet liquid temperature

and the control valve is assumed to have linear flow characteristics such that instantaneous action occurs, i.e., the only lags in the control scheme

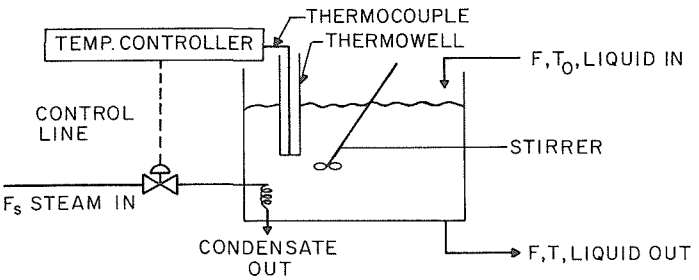

FIGURE 1.7 Temperature control process.

occur in the temperature measures, then the process can be described by

$$Mc_p \frac{dT}{dt} = Fc_p(T_0 - T) + F_s H_v \qquad \text{(liquid energy balance)}$$

$$C_1 \frac{dT_w}{dt} = U_1 A_1 (T - T_w) \qquad \text{(thermowell energy balance)}$$

$$C_2 \frac{dT_t}{dt} = U_2 A_2 (T_w - T_t) \qquad \text{(thermocouple energy balance)}$$

$$F_s = K_p(T_s - T_t) \qquad \text{(proportional control)}$$

For convenience allow

$$\frac{F}{M} = 1 \text{ min}^{-1}$$

$$\frac{H_v}{Mc_p} = 1 \text{ °C/kg}$$

$$T_0 = 50\text{°C}$$

$$10 \frac{U_1 A_1}{C_1} = \frac{U_2 A_2}{C_2} = 1 \text{ min}^{-1}$$

The system of differential equations becomes

$$\frac{dT}{dt} = F_s - T + T_0$$

$$\frac{dT_w}{dt} = 0.1(T - T_w)$$

$$\frac{dT_t}{dt} = T_w - T_t$$

$$F_s = K_p(T_t - T_s)$$

Initially $T = 50$°C. Investigate the temperature response, $T(t)$, to a 10°C step increase in the designed liquid temperature, $T_s = 60$°C, for $K_p = 2$ and $K_p = 6$. Recall that with proportional control there is offset in the response.

6.* In a closed system of three components, the following reaction path can occur:

$$A \xrightarrow{k_1} B$$

$$B + C \xrightarrow{k_2} A + C$$

$$2B \xrightarrow{k_3} C + B$$

with governing differential equations

$$\frac{dC_A}{dt} = -k_1 C_A + k_2 C_B C_C$$

$$\frac{dC_B}{dt} = k_1 C_A - k_2 C_B C_C - k_3 C_B^2$$

$$\frac{dC_C}{dt} = k_3 C_B^2$$

$$C_A(0) = 1, \qquad C_B(0) = C_C(0) = 0$$

Calculate the reaction pathway for $k_1 = 0.08$, $k_2 = 2 \times 10^4$, and $k_3 = 6 \times 10^7$.

7. Develop a numerical procedure to solve

$$\frac{d^2f}{dr^2} + \frac{2}{r}\frac{df}{dr} = \Phi^2 R(f), \qquad 0 \leq r \leq 1$$

$$\frac{df}{dr}(0) = 0, \qquad f(1) = 1$$

Hint: Let $df/dr(1) = \alpha$ and choose α to satisfy $(df/dr)(0) = 0$.

(Later in this text we will discuss this method for solving boundary-value problems. Methods of this type are called shooting methods.)

REFERENCES

1. Conte, S. D., and C. deBoor, *Elementary Numerical Analysis: An Algorithmic Approach,* 3rd Ed., McGraw-Hill, New York (1980).
2. Kehoe, J. P. G., and J. B. Butt, "Interactions of Inter- and Intraphase Gradients in a Diffusion Limited Catalytic Reaction," A.I.Ch.E. J., *18,* 347 (1972).
3. Price, T. H., and J. B. Butt, "Catalyst Poisoning and Fixed Bed Reactor Dynamics-II," Chem. Eng. Sci., *32,* 393 (1977).
4. Gear, C. W., *Numerical Initial-Value Problems in Ordinary Differential Equations,* Prentice-Hall, Englewood Cliffs, N.J. (1971).
5. Johnston, R. L., *Numerical Methods—A Software Approach,* Wiley, New York (1982).
6. Shampine, L. F., and M. K. Gordon, *Computer Solution of Ordinary Differential Equations: The Initial Value Problem,* Freeman, San Francisco (1975).

7. Rosenbrock, H. H., "Some General Implicit Processes for the Numerical Solution of Differential Equations," Comput. J., *5,* 329 (1963).

8. Calahan, D., "Numerical Considerations in the Transient Analysis and Optimal Design of Nonlinear Circuits," Digest Record of Joint Conference on Mathematical and Computer Aids to Design, ACM/SIAM/IEEE, Anaheim, Calif. 129 (1969).

9. Allen, R. H., "Numerically Stable Explicit Integration Techniques Using a Linearized Runge-Kutta Extension," Boeing Scientific Laboratories Document D1-82-0929 (1969).

10. Caillaud, J. B., and L. Padmanabhan, "An Improved Semi-implicit Runge-Kutta Method for Stiff Systems," Chem. Eng. J., *2,* 227 (1971).

11. Norsett, S. P., "One-Step Methods of Hermite-type for Numerical Integration of Stiff Systems," BIT, *14,* 63 (1974).

12. Alexander, R., "Diagonally Implicit Runge-Kutta Methods for Stiff ODES," SIAM J. Numer. Anal., *14,* 1006 (1977).

13. Bui, T. D., and T. R. Bui, "Numerical Methods for Extremely Stiff Systems of Ordinary Differential Equations," Appl. Math. Modelling, *3,* 355 (1979).

14. Burka, M. K., "Solution of Stiff Ordinary Differential Equations by Decomposition and Orthogonal Collocation," A.I.Ch.E.J., *28,* 11 (1982).

15. Lambert, J. D., "Stiffness," in *Computational Techniques for Ordinary Differential Equations,* I. Gladwell, and D. K. Sayers (eds.), Academic, London (1980).

16. Michelsen, M. L., "An Efficient General Purpose Method for the Integration of Stiff Ordinary Differential Equations," A.I.Ch.E.J., *22,* 594 (1976).

17. Villadsen, J., and M. L. Michelsen, *Solution of Differential Equation Models by Polynomial Approximation,* Prentice-Hall, Englewood Cliffs, N.J. (1978).

18. Krogh, F. T., "Algorithms for Changing Step Size," SIAM J. Numer. Anal., *10,* 949 (1973).

19. International Mathematics and Statistics Libraries Inc., Sixth Floor–NBC Building, 7500 Bellaire Boulevard, Houston, Tex.

20. Numerical Algorithms Group (USA) Inc., 1250 Grace Court, Downers Grove, Ill.

21. Harwell Subroutine Libraries, Computer Science and Systems Division of the United Kingdom Atomic Energy Authority, Harwell, England.

22. Forsythe, G. E., M. A. Malcolm, and C. B. Moler, *Computer Methods for Mathematical Computations,* Prentice-Hall, Englewood Cliffs, N.J. (1977).

23. Shampine, L. F., and H. A. Watts, "Global Error Estimation for Ordinary Differential Equations," ACM TOMS, *2,* 172 (1976).

24. Hindmarsh, A. C., "GEAR: Ordinary Differential Equation System Solver," Lawrence Livermore Laboratory Report UCID-30001 (1974).

25. Hindmarsh, A. C., "GEARB: Solution of Ordinary Differential Equations Having Banded Jacobians," Lawrence Livermore Laboratory Report UCID-30059 (1975).

26. Hindmarsh, A. C., "LSODE and LSODI, Two New Initial Value Ordinary Differential Equation Solvers," ACM SIGNUM Newsletter December (1980).

27. Byrne, G. D., and A. C. Hindmarsh, "EPISODEB: An Experimental Package for the Integration of Systems of Ordinary Differential Equations with Banded Jacobians," Lawrence Livermore Laboratory Report UCID-30132 (1976).

28. Verwer, J. G., "Algorithm 553. M3RK, An Explicit Time Integrator for Semidiscrete Parabolic Equations," ACM TOMS, *6,* 236 (1980).

29. Butcher, J. C., K. Burrage, and F. H. Chipman, "STRIDE Stable Runge-Kutta Integrator for Differential Equations," Report Series No. 150, Department of Mathematics, University of Auckland, New Zealand (1979).

30. Skeel, R., and A. Kong, "Blended Linear Multistep Methods," ACM TOMS, *3,* 326 (1977).

31. Tendler, J. M., T. A. Bickart, and Z. Picel, "Algorithm 534. STINT: STiff INTegrator," ACM TOMS, *4,* 399 (1978).

32. Addison, C. A., "Implementing a Stiff Method Based Upon the Second Derivative Formulas," University of Toronto Department of Computer Science, Technical Report No. 130/79 (1979).

33. Gladwell, I., J. A. I. Craigie, and C. R. Crowther, "Testing Initial-Value Problem Subroutines as Black Boxes," Numerical Analysis Report No. 34, Department of Mathematics, University of Manchester, Manchester, England.

34. Gaffney, P. W., "Information and Advice on Numerical Software," Oak Ridge National Laboratory Report No. ORNL/CSD/TM-147, May (1981).

35. Hull, T. E., W. H. Enright, B. M. Fellen, and A. E. Sedgwick, "Comparing Numerical Methods for Ordinary Differential Equations," SIAM J. Numer. Anal., *9,* 603 (1972).

36. Enright, W. H., T. E. Hull, and B. Lindberg, "Comparing Numerical Methods for Stiff Systems of ODEs," BIT, *15,* 10 (1975).

37. Shampine, L. F., H. A. Watts, and S. M. Davenport, "Solving Nonstiff Ordinary Differential Equations—The State of the Art," SIAM Rev., *18,* 376 (1976).

38. Byrne, G. D., A. C. Hindmarsh, K. R. Jackson, and H. G. Brown, "A Comparison of Two ODE CODES: GEAR and ERISODE," Comput. Chem. Eng., *1,* 133 (1977).

39. Robertson, A. H., "Solution of a Set of Reaction Rate Equations," in *Numerical Analysis,* J. Walsh (ed.), Thomson Brook Co., Washington (1967).

40. Seinfeld, J. H., L. Lapidus, and M. Hwang, "Review of Numerical Integration Techniques for Stiff Ordinary Differential Equations," Ind. Eng. Chem. Fund., *9,* 266 (1970).

41. Finlayson, B. A., *Nonlinear Analysis in Chemical Engineering,* McGraw-Hill, New York (1980).

BIBLIOGRAPHY

Only a brief overview of IVPs has been given in this chapter. For additional or more detailed information, see the following:

Finlayson, B. A., *Nonlinear Analysis in Chemical Engineering,* McGraw-Hill, New York (1980).

Forsythe, G. E., M. A. Malcolm, and C. B. Moler, *Computer Methods for Mathematical Computations,* Prentice-Hall, Englewood Cliffs, N.J. (1977).

Gear, C. W., *Numerical Initial-Value Problems in Ordinary Differential Equations,* Prentice-Hall, Englewood Cliffs, N.J. (1971).

Hall, G., and J. M. Watt (eds.), *Modern Numerical Methods for Ordinary Differential Equations,* Clarendon Press, Oxford (1976).

Johnston, R. L., *Numerical Methods—A Software Approach,* Wiley, New York (1982).

Lambert, J. D., *Computational Methods in Ordinary Differential Equations,* Wiley, New York (1973).

Shampine, L. F., and M. K. Gordon, *Computer Solution of Ordinary Differential Equations: The Initial Value Problem,* Freeman, San Francisco (1975).

Villadsen, J., and M. L. Michelsen, *Solution of Differential Equation Models by Polynomial Approximation,* Prentice-Hall, Englewood Cliffs, N.J. (1978).