9

Initial Value Problems of Ordinary Differential Equations

9.1 INTRODUCTION

Problems of solving ordinary differential equations (ODE) are classified into initial value problems and boundary value problems. Many initial value problems are time-dependent, in which all the conditions for the solution are specified at the initial time. The numerical methods for initial value problems are significantly different from those for boundary value problems. Therefore, the present chapter discusses the numerical solution methods for the former type only, and Chapter 10 describes the numerical methods for the latter.

The initial value problem of a first-order ODE may be written in the form

$$y'(t) = f(y, t), \quad y(0) = y_0 \tag{9.1.1}$$

where $f(y, t)$ is a function of y and t, and the second equation is an initial condition. In Eq. (9.1.1), the first derivative of y is given by a known function of y and t, and we desire to compute the unknown function y by numerically integrating $f(y, t)$. If f were independent of y, the computation would be one of the straightforward integrations discussed in Chapter 4. However, the fact that f is a function of the unknown function y makes the integration different.

The initial condition is always a part of the problem definition because the solution of an initial value problem can be uniquely determined only if an initial condition is given.

More examples for initial value problems of first-order ordinary differential equations follow:

(a) $y'(t) = 3y + 5, \quad y(0) = 1$

(b) $y'(t) = ty + 1, \quad y(0) = 0$

(c) $y'(t) = -\dfrac{1}{1 + y^2}, \quad y(0) = 1$

(e) $y' = z, z' = -y, \quad y(0) = 1, z(0) = 0$

Numerical methods for ordinary differential equations calculate the solution on the points, $t_n = t_{n-1} + h$, where h is the step size (or time interval).

Three types of numerical integration methods for initial value problems are described in this chapter: Euler methods, Runge-Kutta methods, and predictor-corrector methods. Major aspects of the methods are summarized in Table 9.1.

Table 9.1 Summary of the methods for initial value problems of ODEs

Name of methods	Relevant formula	Error: Local	Error: Global	Other features[a]
Nonstiff Equation:				
Euler methods				
Forward	Forward difference	$O(h^2)$	$O(h)$	SS, EC
Modified	Trapezoidal rule	$O(h^3)$	$O(h^2)$	SS, EC, NL
Backward	Backward difference	$O(h^2)$	$O(h)$	SS, EC, NL
Runge-Kutta				
Second-order	Trapezoidal rule	$O(h^3)$	$O(h^2)$	SS, EC
Third-order	Simpson's 1/3 rule	$O(h^4)$	$O(h^3)$	SS, EC
Fourth-order	Simpson's 1/3 or 3/8	$O(h^5)$	$O(h^4)$	SS, EC
Predictor-Corrector				
Second-order	(identical with second-order Runge-Kutta)			SS, EC
Third-order	Newton backward	$O(h^4)$	$O(h^3)$	NS, DC
Fourth-order	Newton backward	$O(h^5)$	$O(h^4)$	NS, DC
Stiff Equations:				
Implicit Methods	Backward difference: Gear method			SS/NS
Exponential Transformation	Exponential transformation			SS

[a] SS: self-starting capability.
NS: no self-starting capability.
EC: step size can be changed easily in the middle of solution.
DC: difficult to change the step size.
NL: solution of nonlinear equations may be necessary in each step.

Once we learn the numerical methods to solve first-order differential equations, they can be easily applied to higher-order ODEs because a higher-order ODE can be decomposed to a set of first-order differential equations. For example, consider

$$y'''' + ay''' + by'' + cy' + ey = g \tag{9.1.2}$$

where a, b, c, e, and g are constants or known functions of t. The initial conditions are given as

$$y(0) = y_0, \quad y'(0) = y'_0,$$
$$y''(0) = y''_0, \quad y'''(0) = y'''_0$$

and where y_0, y'_0, y''_0, and y'''_0 are prescribed values. By defining u, v, and w as

$$u = y', \quad v = y'', \quad w = y'''$$

Eq. (9.1.2) can be written as

$$w' + aw + bv + cu + ey = g \tag{9.1.3}$$

So, Eq. (9.1.2) is equivalent to the set of four first-order ordinary differential equations:

$$\begin{aligned} y' &= u, & y(0) &= y_0 \\ u' &= v, & u(0) &= y'_0 \\ v' &= w, & v(0) &= y''_0 \\ w' &= g - aw - bv - cu - ey, & w(0) &= y'''_0 \end{aligned}$$

The numerical methods for first-order ordinary differential equations are then applicable to the foregoing set.

The numerical methods may be applied to integro-differential equations, too. For example, consider the equation given by

$$y'' + ay + \int_0^t y(s)\,ds = g, \quad y(0) = y_0, \quad y'(0) = y'_0 \tag{9.1.4}$$

By defining u and v as

$$u = y', \quad v = \int_0^t y(s)\,ds$$

Eq. (9.1.4) becomes

$$\begin{aligned} u' &= -ay - v + g, & u(0) &= y'_0 \\ v' &= y, & v(0) &= 0 \\ y' &= u, & y(0) &= y_0 \end{aligned} \tag{9.1.5}$$

The foregoing set of first-order ordinary differential equations can be solved by a numerical method.

9.2 EULER METHODS

We start our study with the Euler methods, which are suitable for a quick programming because of their great simplicity. It should be pointed out that, as the system of equations becomes more complicated, the Euler methods are more often used. Indeed, a large fraction of numerical methods for parabolic and hyperbolic partial differential equations, which are far more complicated than ordinary differential equations, are based on Euler methods rather than the Runge-Kutta or predictor-corrector methods.

Euler methods consist of three versions: (a) forward Euler, (b) modified Euler, and (c) backward Euler methods.

9.2.1 Forward Euler Method

The forward Euler method for $y' = f(y, t)$ is obtained by rewriting the forward difference approximation,

$$\frac{y_{n+1} - y_n}{h} \simeq y'_n \tag{9.2.2}$$

to

$$y_{n+1} = y_n + hf(y_n, t_n) \tag{9.2.3}$$

where $y'_n = f(y_n, t_n)$ is used. Using Eq. (9.2.3), y_n is recursively calculated as

$$\begin{aligned} y_1 &= y_0 + hy'_0 = y_0 + hf(y_0, t_0) \\ y_2 &= y_1 + hf(y_1, t_1) \\ y_3 &= y_2 + hf(y_2, t_2) \\ &\vdots \\ y_n &= y_{n-1} + hf(y_{n-1}, t_{n-1}) \end{aligned}$$

Example 9.1

(a) Solve $y' = -20y + 7\exp(-0.5t)$, $y(0) = 5$, using the forward Euler method with $h = 0.01$ for $0 < t \leqslant 0.02$. Do this part by hand calculation.

(b) Repeat the same for $h = 0.01$, 0.001, and 0.0001 on a computer for $0 \leqslant t \leqslant 0.09$. Evaluate errors of the three calculations by comparison to the analytical solution given by

$$y = 5e^{-20t} + (7/19.5)(e^{-0.5t} - e^{-20t})$$

⟨Solution⟩

(a) The first few steps of calculations with $h = 0.1$ are shown next:

$t_0 = 0$, $y_0 = y(0) = 5$

$t_1 = 0.01$, $y_1 = y_0 + hy'_0 = 5 + (0.01)(-20(5) + 7\exp(0))$
$= 4.07$

$t_2 = 0.02$, $y_2 = y_1 + hy'_1 = 4.07 + (0.01)(-20(4.07) + 7\exp(-0.005))$
$= 3.32565$

$\vdots$

$t_n = nh$, $y_n = y_{n-1} + hy'_{n-1}$

The computational results for selected values of t with three values of time interval (grid spacing) are shown in Table 9.2.

Table 9.2 Forward Euler method

t	$h = 0.01$	$h = 0.001$	$h = 0.0001$
0.01	4.07000 (8.693)[a]	4.14924 (0.769)[a]	4.15617 (0.076)[a]
0.02	3.32565 (14.072)	3.45379 (1.259)	3.46513 (0.124)
0.03	2.72982 (17.085)	2.88524 (1.544)	2.89915 (0.153)
0.04	2.25282 (18.440)	2.42037 (1.684)	2.43554 (0.167)
0.05	1.87087 (18.658)	2.04023 (1.722)	2.05574 (0.171)
0.06	1.56497 (18.125)	1.72932 (1.690)	1.74454 (0.168)
0.07	1.31990 (17.119)	1.47496 (1.613)	1.48949 (0.169)
0.08	1.12352 (15.839)	1.26683 (1.507)	1.28041 (0.150)
0.09	0.96607 (14.427)	1.09646 (1.387)	1.10895 (0.138)
0.10	0.83977 (12.979)	0.95696 (1.261)	0.96831 (0.126)

[a](error) × 100

Comments: Accuracy of the forward Euler method increases with a decrease in time interval h. In effect, magnitudes of errors are approximately proportional to h. However, further reduction of h without using double precision is not advantageous because it increases numerical error caused by round-off (see Chapter 1).

Although the forward Euler method is simple, it has to be used carefully for two kinds of errors. The first is the truncation errors that we have already seen in Example 9.1. The second is a potential of instability, which occurs when the time constant of the equation is negative unless time interval h is sufficiently small. A typical equation for a diminishing solution is $y' = -\alpha y$, with $y(0) = y_0 > 0$, where $\alpha > 0$. The exact solution is $y = y_0 \exp(-\alpha t)$ that approaches zero as t increases. The forward Euler method for this problem becomes

$$y_{n+1} = (1 - \alpha h)y_n$$

If $\alpha h < 1$, the numerical solution is diminishing and positive, but if $\alpha h > 1$, the sign of the solution alternates. Furthermore, if $\alpha h > 2$, the magnitude of the solution increases after each step, and the solution oscillates. This is the *instability*.

The forward Euler method is applicable to a set of first-order ODEs. Consider a set of first-order ODEs given by

$$\begin{aligned} y' &= f(y, z, t), \quad y(0) = y_0 \\ z' &= g(y, z, t), \quad z(0) = z_0 \end{aligned} \tag{9.2.4}$$

The Euler method for Equation (9.2.4) is written as

$$\begin{aligned} y_{n+1} &= y_n + hy'_n = y_n + hf(y_n, z_n, t_n) \\ z_{n+1} &= z_n + hz'_n = z_n + hg(y_n, z_n, t_n) \end{aligned} \tag{9.2.5}$$

A higher-order ordinary differential equation may be broken into a set of coupled first-order differential equations as mentioned earlier.

Example 9.2

Using the forward Euler method with $h = 0.5$, find the values of $y(1)$ and $y'(1)$ for

$$y''(t) - 0.05y'(t) + 0.15y(t) = 0, \quad y'(0) = 0, \quad y(0) = 1$$

⟨Solution⟩

Let $y' = z$, then the second-order ODE becomes

$$\begin{aligned} y' &= z, & y(0) &= 1 \\ z' &= 0.05z - 0.15y, & z(0) &= 0 \end{aligned}$$

We will denote $y_n = y(nh)$ and $z_n = z(nh)$. The initial conditions are expressed as $y_0 = y(0) = 1$ and $z_0 = y'(0) = 0$. Using the forward Euler method, y and z at $n = 1$ and $n = 2$ become:

$t = 0.5$:

$$\begin{aligned} y'_0 &= z_0 = 0 \\ z'_0 &= 0.05z_0 - 0.15y_0 = -0.15 \\ y_1 &= y_0 + hy'_0 = 1 + (0.5)(0) = 1 \\ z_1 &= z_0 + hz'_0 = 0 + (0.5)(-0.15) = -0.075 \end{aligned}$$

$t = 1$:

$$\begin{aligned} y'_1 &= z_1 = -0.075 \\ z'_1 &= 0.05z_1 - 0.15y_1 = (0.05)(-0.075) - (0.15)(1) = -0.15375 \\ y_2 &= y_1 + hy'_1 = 1 + (0.5)(-0.075) = 0.96250 \\ z_2 &= z_1 + hz'_1 = -0.075 + (0.5)(-0.15375) = -0.15187 \end{aligned}$$

Therefore

$$\begin{aligned} y(1) &= y_2 = 0.96250 \\ y'(1) &= z(1) = z_2 = -0.15187 \end{aligned}$$

Example 9.3

Solve the following set of first-order ODEs by the forward Euler method with $h = 0.005\pi$ and $h = 0.0005\pi$:

$$\begin{aligned} y' &= z, & y(0) &= 1 \\ z' &= -y, & z(0) &= 0 \end{aligned} \tag{A}$$

⟨Solution⟩

The calculations for the first few steps with $h = 0.0005\pi$ are shown next.

$t_0 = 0$: $y_0 = 1$, $z_0 = 0$

$t_1 = 0.0005\pi$: $y_1 = y_0 + hz_0 = 1 + (0.0005\pi)(0) = 1.0$, $z_1 = z_0 - hy_0 = 0 - (0.0005\pi)(1) = -0.00157$

$t_2 = 0.001\pi$: $y_2 = y_1 + hz_1 = 1 + (0.0005\pi)(-0.00157) = 0.99999$, $z_2 = z_1 - hy_1 = -0.00157 - (0.0005\pi)(1) = -0.00314$

In Table 9.3 the results of the present calculations for selected values of t are compared to the exact solution, $y = \cos(t)$ and $z = -\sin(t)$.

Table 9.3

	Exact		$h = 0.005\pi$		$h = 0.0005\pi$	
t	$y = \cos(t)$	$z = -\sin(t)$	y	z	y	z
0.5π	0	−1	1.32E − 4	−1.01241	2.62E − 6	−1.00123
π	−1	0	−1.02497	−2.67E − 4	−1.00247	−5.25E − 6
1.5π	0	1	−4.01E − 4	1.03770	−7.88E − 6	1.00371
2π	1	0	1.05058	5.48E − 4	1.00495	1.05E − 5
3π	−1	0	−1.07682	−8.43E − 4	−1.00743	−1.58E − 5
6π	1	0	1.15954	1.82E − 3	1.01491	3.19E − 5
8π	1	0	1.21819	2.54E − 3	1.01994	4.27E − 5

It is observed in the results shown in Table 9.3 that the error of y increases with increase in t, and in proportion to h. (See the y values for $t = \pi$, 2π, 3π, 6π and 8π: z values for these t values do not follow the same trend because, when z is close to zero, the errors of z are significantly affected by phase shift.)

9.2.3 Modified Euler Method

The motivation for the modified Euler method is twofold. First, the modified Euler method is more accurate than the forward Euler method. Second, it is more stable than the forward Euler method.

The modified Euler method is derived by applying the trapezoidal rule to integrating $y' = f(y, t)$:

$$y_{n+1} = y_n + \frac{h}{2}[f(y_{n+1}, t_{n+1}) + f(y_n, t_n)] \tag{9.2.6}$$

If f is linear in y, both Eqs.(9.2.6) may be easily solved for y_{n+1}. For example, if the ODE is given by

$$y' = ay + \cos(t)$$

Eq. (9.2.6) becomes

$$y_{n+1} = y_n + \frac{h}{2}[ay_{n+1} + \cos(t_{n+1}) + ay_n + \cos(t_n)]$$

Therefore, solving for y_{n+1} yields

$$y_{n+1} = \frac{1 + ah/2}{1 - ah/2} y_n + \frac{h/2}{1 - ah/2}[\cos(t_{n+1}) + \cos(t_n)] \tag{9.2.7}$$

If f is a nonlinear function of y, Eq. (9.2.6) becomes a nonlinear function of y_{n+1}, so an algorithm to solve nonlinear equations is necessary. A widely used method for solving nonlinear equations is the successive substitution method (Section 3.6):

$$y_{n+1}^{(k)} - y_n = \frac{h}{2}[f(y_{n+1}^{(k-1)}, t_{n+1}) + f(y_n, t_n)] \tag{9.2.8}$$

where $y_{n+1}^{(k)}$ is the kth iterative approximation for y_{n+1}, and $y_{n+1}^{(0)}$ is an initial guess for y_{n+1}. The above iteration is terminated when $|y_{n+1}^{(k)} - y_{n+1}^{(k-1)}|$ becomes less than a prescribed tolerance. The initial guess is set to y_n. Then, the first iteration step becomes identical with the forward Euler method. If only one more iteration step is used, the scheme becomes the second-order Runge-Kutta method or, equivalently, the Euler predictor-corrector method. But, in the modified Euler method, iteration is continued until the tolerance of convergence is satisfied.

Example 9.4 shows an application of the modified Euler method to a nonlinear first-order ODE.

Example 9.4

By the modified Euler method with $h = 0.1$, solve

$$y' = -y^{1.5} + 1, \quad y(0) = 10$$

for $0 \leqslant t \leqslant 1$. Print the results up to $t = 1$.

〈Solution〉

The modified Euler method becomes

$$y_{n+1} = y_n + \frac{h}{2}[-(y_{n+1})^{1.5} - (y_n)^{1.5} + 2] \tag{A}$$

For $n = 0$:

$$y_1 = y_0 + \frac{h}{2}[-(y_1)^{1.5} - (y_0)^{1.5} + 2]$$

The best estimate for y_1 on the right side is y_0. By introducing $y_1 \simeq y_0$ to the right side, the equation becomes

$$y_1 \simeq 10 + \frac{0.1}{2}[-(10)^{1.5} - (10)^{1.5} + 2] = 6.93772$$

Introducing the above result to y_1 on the right side of Eq. (A) again yields

$$y_1 \simeq 10 + \frac{0.1}{2}[-(6.93772)^{1.5} - (10)^{1.5} + 2] = 7.60517$$

Repeating the substitution a few more times, we get

$$y_1 \simeq 10 + \frac{0.1}{2}[-(7.60517)^{1.5} - (10)^{1.5} + 2] = 7.47020$$

$$y_1 \simeq 10 + \frac{0.1}{2}[-(7.47020)^{1.5} - (10)^{1.5} + 2] = 7.49799$$

$$y_1 \simeq 10 + \frac{0.1}{2}[-(7.49799)^{1.5} - (10)^{1.5} + 2] = 7.49229$$

$$\vdots$$

$$y_1 = 10 + \frac{0.1}{2}[-(7.49326)^{1.5} - (10)^{1.5} + 2] = 7.49326$$

The computed results for ten time steps follow:

t	y
0.0	10.0
0.1	7.4932
0.2	5.8586
0.3	4.7345
0.4	3.9298
0.5	3.3357
0.6	2.8859
0.7	2.5386
0.8	2.2658
0.9	2.0487
1.0	1.8738

Why is the accuracy of the modified Euler method higher than that of the forward Euler method? To explain the reason analytically, let us consider a test equation, $y' = \alpha y$. Equation (9.2.6) for this problem may then be written as

$$y_{n+1} = y_n + \frac{\alpha h}{2}(y_{n+1} + y_n) \tag{9.2.9}$$

or equivalently, solving for y_{n+1},

$$y_{n+1} = \left(1 + \frac{\alpha h}{2}\right)\left(1 - \frac{\alpha h}{2}\right)^{-1} y_n$$

Expanding the coefficient of the foregoing equation yields

$$y_{n+1} = \left(1 + \alpha h + \frac{1}{2}(\alpha h)^2 + \frac{1}{4}(\alpha h)^3 + \cdots\right) y_n$$

Comparing this expansion to the Taylor expansion of the exact solution $y(t_{n+1}) = \exp(\alpha h) y_n$, it is found that Eq. (9.2.9) is accurate to the second-order term. Thus, the modified Euler method is a second-order (accurate) method. On the other hand, a similar analysis for the forward Euler method indicates that the forward Euler method is first-order accurate.

The local error (error generated in each step) of the forward Euler method is proportional to h^2 and its global error is proportional to h, whereas the local error of the modified Euler method is proportional to h^3 and its global error is proportional to h^2. The order of errors of the backward Euler method is the same as in the forward Euler method.

If the modified Euler method is applied to a set of ODEs, the whole equations must be solved simultaneously or "implicitly." However, the advantage of the implicit solution is that the method is more stable than the forward Euler method and thus allows a larger time step.

9.2.4 Backward Euler Method

The backward Euler method is based on the backward difference approximation and is written as

$$y_{n+1} = y_n + hf(y_{n+1}, t_{n+1}) \tag{9.2.10}$$

The accuracy of this method is the same as that of the forward Euler method. Besides, if f is a nonlinear function of y, an iterative scheme has to be used in each step just as in the modified Euler method. However, the advantages are (a) the method is stable for stiff problems, and (b) positivity of solution is guaranteed when the exact solution is positive. See applications of the backward Euler method in Section 9.5 and Chapter 12.

Summary of This Section

(a) The forward Euler method is based on the forward difference approximation. Its error in one interval is proportional to h^2 and its global error to h. The forward Euler method may become unstable if the ODE has a negative time constant unless a small h is used.

(b) The modified Euler method is based on the trapezoidal rule. If the ODE is not linear, an iterative method is necessary for each interval. Its error in one interval is proportional to h^3 and its global error to h^2.

(c) The backward Euler method is based on the backward difference approximation. Its errors are comparable to those of the forward Euler method. The method is stable so it is used to solve stiff problems that are difficult to solve by other methods.

9.3 RUNGE-KUTTA METHODS

A major drawback of the Euler methods is that the orders of accuracy are low. This disadvantage is twofold. To maintain a high accuracy requires very small h, which increases computational time and causes round-off errors.

In Runge-Kutta methods, the order of accuracy is increased by using intermediate points in each step interval. A higher accuracy also implies that errors decrease more quickly than in lower-order accuracy methods when h is reduced.

Consider an ordinary differential equation

$$y' = f(y, t), \quad y(0) = y_0 \tag{9.3.1}$$

To calculate y_{n+1} at $t_{n+1} = t_n + h$ with a known value of y_n, we integrate Eq. (9.3.1) over the interval $[t_n, t_{n+1}]$ as

$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} f(y, t)\, dt \tag{9.3.2}$$

Runge-Kutta methods are derived by applying a numerical integration method to the integral on the right side of Eq. (9.3.2) [Fox/Mayers]. In the remainder of this section, the second-, third-, and fourth-order Runge-Kutta methods are explained.

9.3.1 Second-Order Runge-Kutta Method

Here we examine an application of the trapezoidal rule to the right side of Eq. (9.3.2):

$$\int_{t_n}^{t_{n+1}} f(y, t)\, dt \simeq \frac{1}{2} h[f(y_n, t_n) + f(y_{n+1}, t_{n+1})] \tag{9.3.3}$$

In Eq. (9.3.3) y_{n+1} is not known, so the second term is approximated by $f(\bar{y}_{n+1}, t_{n+1})$, where $\bar{y}_{n+1}$ is the first estimate for y_{n+1} calculated by the forward Euler method. The scheme derived here is called the second-order Runge-Kutta method and summarized as

$$\bar{y}_{n+1} = y_n + hf(y_n, t_n)$$

$$y_{n+1} = y_n + \frac{h}{2}[f(y_n, t_n) + f(\bar{y}_{n+1}, t_{n+1})]$$

or in a more standard form as

$$k_1 = hf(y_n, t_n)$$
$$k_2 = hf(y_n + k_1, t_{n+1}) \tag{9.3.4}$$
$$y_{n+1} = y_n + \frac{1}{2}[k_1 + k_2]$$

The second-order Runge-Kutta method is identical to the Euler predictor-corrector method, which is the simplest predictor-corrector method (See Section 9.4). It is also equivalent to the modified Euler method applied with only one iteration step.

Example 9.5

The circuit shown in Figure E9.5 has a self-inductance of $L = 50$H, a resistance of $R = 20$ ohm, and a voltage source of $V = 10$ volt. If the switch is closed at $t = 0$, the current $I(t)$ satisfies

$$L\frac{d}{dt}I(t) + RI(t) = E, \quad I(0) = 0 \tag{A}$$

Find the electric current for $0 < t \leqslant 10$ sec by using the second-order Runge-Kutta method with $h = 0.1$.

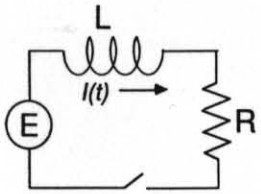


Figure E9.5 An electric circuit

⟨Solution⟩

We first rewrite Eq. (A) as

$$\frac{d}{dt}I = -\frac{R}{L}I + \frac{E}{L} \equiv f(I, t)$$

Then, the second-order Runge-Kutta method becomes

$$k_1 = h\left[-\frac{R}{L}I_n + \frac{E}{L}\right]$$
$$k_2 = h\left[-\frac{R}{L}(I_n + k_1) + \frac{E}{L}\right]$$
$$I_{n+1} = I_n + \frac{1}{2}(k_1 + k_2)$$

The calculations for the first two steps are shown next:

$n = 0$ $(t = 0.1)$: $k_1 = 0.1[(-0.4)(0) + 0.2] = 0.02$

$k_2 = 0.1[(-0.4)(0 + 0.02) + 0.2] = 0.0192$

$$I_1 = I_0 + \frac{1}{2}(k_1 + k_2) = 0 + \frac{1}{2}(0.02 + 0.0192) = 0.0196$$

$n = 1$ $(t = 0.2)$: $k_1 = 0.1[(-0.4)(0.0196) + 0.2] = 0.019216$

$k_2 = 0.1[(-0.4)(0.0196 + 0.019216) + 0.2] = 0.018447$

$$I_2 = I_1 + \frac{1}{2}(k_1 + k_2)$$
$$= 0.0196 + \frac{1}{2}(0.019216 + 0.018447) = 0.038431$$

The final result of computation (at multiples of 10 steps) is as follows:

t (sec)	I (amp)
0	0
1	0.1648
2	0.2752
3	0.3493
4	0.3990
5	0.4332
6	0.4546
7	0.4695
8	0.4796
9	0.4863
10	0.4908
(∞)	(0.5000)

The accuracy of the second-order Runge-Kutta method may be analyzed by using the test equation $y' = \alpha y$ as described toward the end of Section 9.2. However, to show a more formal approach, we consider here a generic form, $y' = f(y, t)$. We first expand the exact value of y_{n+1} in the Taylor series:

$$y_{n+1} = y_n + hf + \frac{h^2}{2}[f_t + f_y f]$$
$$+ \frac{h^3}{6}[f_{tt} + 2f_{ty}f + f_{yy}f^2 + f_t f_y + f_y^2 f] + 0(h^4) \tag{9.3.5}$$

where all the derivatives of y are expressed in terms of f and the partial derivatives of f at t_n.

Next, we expand the third equation in Eq. (9.3.4) in a Taylor series:

$$y_{n+1} = y_n + hf + \frac{h^2}{2}[f_t + f_y f] + \frac{h^3}{4}[f_{tt} + 2f_{ty}f + f_{yy}f^2] + 0(h^4) \tag{9.3.6}$$

By comparing Eq. (9.3.6) to Eq. (9.3.5), Eq. (9.3.4) is found to be accurate to the order of h^2 and the discrepancy (error generated in one step) is proportional to h^3. Notice that the second-order Runge-Kutta method is identical to the modified Euler method given by Eq. (9.2.8) with two iteration steps. However, the order of accuracy of the former is identical to that of the latter, which requires iterative convergence. This indicates that strict convergence of the iteration in the modified Euler method is meaningless. (Indeed, using the second-order Runge-Kutta method with a smaller h is far more effective in improving accuracy than using the modified Euler method with strict iterative convergence.) The foregoing analysis may be done more easily by considering the test equation $y' = \alpha y$, but this approach is left as a student's exercise.

Application of the second-order Runge-Kutta method to a higher-order ordinary differential equation is easy. For illustration, we consider the second-order differential equation:

$$y''(t) + ay'(t) + by(t) = q(t), \quad y(0) = 1, y'(0) = 0 \tag{9.3.7}$$

where a and b are coefficients and $q(t)$ is a known function, and two initial conditions are given. By defining

$$z(t) = y'(t) \tag{9.3.8}$$

Eq. (9.3.7) can be reduced to coupled first-order differential equations:

$$\begin{aligned} y' &= f(y, z, t) \equiv z, & y(0) &= 1 \\ z' &= g(y, z, t) \equiv -az - by + q, & z(0) &= 0 \end{aligned} \tag{9.3.9}$$

The second-order Runge-Kutta method for the foregoing equations is written as

$$\begin{aligned} k_1 &= hf(y_n, z_n, t_n) = hz_n \\ l_1 &= hg(y_n, z_n, t_n) = h(-az_n - by_n + q_n) \\ k_2 &= hf(y_n + k_1, z_n + l_1, t_{n+1}) = h(z_n + l_1) \\ l_2 &= hg(y_n + k_1, z_n + l_1, t_{n+1}) = h(-a(z_n + l_1) - b(y_n + k_1) + q_{n+1}) \end{aligned} \tag{9.3.10}$$

$$y_{n+1} = y_n + \frac{1}{2}(k_1 + k_2)$$

$$z_{n+1} = z_n + \frac{1}{2}(l_1 + l_2)$$

Example 9.6

A cubic material of mass $M = 0.5$ kg is fixed to the lower end of a massless spring. The upper end of the spring is fixed to a structure at rest. The cube receives resistance $R = -B\,dy/dt$ from the air, where B is a damping constant.

(See Figure E9.6.) The equation of motion is

$$M\frac{d^2}{dt^2}y + B\frac{d}{dt}y + ky = 0, \quad y(0) = 1, y'(0) = 0 \tag{A}$$

where y is the displacement from the static position, k is the spring constant equal to 100 kg/s^2, and $B = 10$ kg/s.

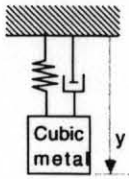


Figure E9.6 A spring-mass system

(a) Calculate $y(t)$ for $0 < t < 0.05$ using the second-order Runge-Kutta method with $h = 0.025$ by hand calculations.

(b) Calculate $y(t)$ for $0 < t < 10$ sec using the second-order Runge-Kutta method with $h = 0.001$.

(c) Repeat (b) for $B = 0$.

⟨Solution⟩

Equation (A) may be written as

$$\begin{aligned} y' &= z \equiv f(y, z, t), & y(0) &= 1 \\ z' &= -\frac{B}{M}z - \frac{k}{M}y \equiv g(y, z, t), & z(0) &= 0 \end{aligned} \tag{B}$$

By setting $a = B/M = 20$ and $b = k/M = 200$ and $g = 0$, the second-order Runge-Kutta method for Eq. (A) becomes the form of Eq. (9.3.9).

(a) For $n = 1$: $t = 0.025$

$$k_1 = hf(y_0, z_0, t_0) = hz_0 = 0.025(0) = 0$$

$$l_1 = hg(y_0, z_0, t_0) = h(-20z_0 - 200y_0) = 0.025(-20(0) - 200(1)) = -5$$

$$k_2 = hf(y_0 + k_1, z_0 + l_1, t_0) = h(z_0 + l_1) = 0.025(0 - 5) = -0.125$$

$$\begin{aligned} l_2 &= hg(y_0 + k_1, z_0 + l_1, t_1) = h[-20(z_0 + l_1) - 200(y_0 + k_1)] \\ &= 0.025[-20(0 - 5) - 200(1 + 0)] = -2.5 \end{aligned}$$

$$y_1 = y_0 + \frac{1}{2}(0 - 0.125) = 0.9375$$

$$z_1 = z_0 + \frac{1}{2}(-5 - 2.5) = -3.75$$

For $n = 2$: $t = 0.05$

$$k_1 = hf(y_1, z_1, t_1) = hz_1 = 0.025(-3.75) = -0.09375$$

$$\begin{aligned} l_1 &= hg(y_1, z_1, t_1) = h(-20z_1 - 200y_1) \\ &= 0.025[-20(-3.75) - 200(0.9375)] = -2.8125 \end{aligned}$$

$$k_2 = hf(y_1 + k_1, z_1 + l_1, t_1) = h(z_1 + l_1)$$
$$= 0.025(-3.75 - 2.8125) = 0.1640625$$
$$l_2 = hg(y_1 + k_1, z_1 + l_1, t_1) = h[-20(z_1 + l_1) - 200(y_1 + k_1)]$$
$$= 0.025[-20(-3.75 - 2.8125) - 200(0.9375 - 0.093750)]$$
$$= -0.9375$$
$$y_2 = y_1 + \frac{1}{2}(-0.09375 - 0.1640625) = 0.80859$$
$$z_2 = z_1 + \frac{1}{2}(-2.8125 - 0.9375) = -5.625$$

(b) and (c) This part of the computations was performed by using PROGRAM 9–1. The computational results after every 50 steps up to 0.75 sec are shown below:

t (sec)	(b) y (meter)	(c) y (meter)
	($B = 10$)	($B = 0$)
0	1.000	1.000
0.05	0.823	0.760
0.1	0.508	0.155
0.15	0.238	−0.523
0.2	0.066	−0.951
0.25	−0.016	−0.923
0.3	−0.042	−0.45
0.35	−0.038	0.235
0.4	−0.025	0.810
0.45	−0.013	0.996
0.5	−0.004	0.705
0.55	0.000	0.075
0.6	0.001	−0.590
0.65	0.001	−0.973
0.7	0.001	−0.889
0.75	0.000	−0.378

9.3.2 Third-Order Runge-Kutta Method

A Runge-Kutta method that is more accurate than the second-order Runge-Kutta method may be derived by using a higher-order numerical integration scheme for the second term of Eq. (9.3.2). Using the Simpson's 1/3 rule, Eq. (9.3.2) is approximately

$$y_{n+1} = y_n + \frac{h}{6}[f(y_n, t_n) + 4f(\bar{y}_{n+\frac{1}{2}}, t_{n+\frac{1}{2}}) + f(\bar{y}_{n+1}, t_{n+1})] \tag{9.3.11}$$

where $\bar{y}_{n+1}$ and $\bar{y}_{n+\frac{1}{2}}$ are estimates because $y_{n+\frac{1}{2}}$ and y_{n+1} are not known.

The estimate $\bar{y}_{n+\frac{1}{2}}$ is obtained by the forward Euler method as

$$\bar{y}_{n+\frac{1}{2}} = y_n + \frac{h}{2}f(y_n, t_n) \tag{9.3.12}$$

The estimate $\bar{y}_{n+1}$ may be obtained by

$$\bar{y}_{n+1} = y_n + hf(y_n, t_n)$$

or

$$\bar{y}_{n+1} = y_n + hf(\bar{y}_{n+\frac{1}{2}}, t_{n+\frac{1}{2}})$$

or a linear combination of both

$$\bar{y}_{n+1} = y_n + h[\theta f(y_n, t_n) + (1 - \theta)f(\bar{y}_{n+\frac{1}{2}}, t_{n+\frac{1}{2}})] \tag{9.3.13}$$

Here θ is an undetermined parameter, which will be determined to maximize accuracy of the numerical method. With Eq. (9.3.13), the whole scheme is written in the following form:

$$\begin{aligned} k_1 &= hf(y_n, t_n) \\ k_2 &= hf\left(y_n + \frac{1}{2}k_1, t_n + \frac{h}{2}\right) \\ k_3 &= hf(y_n + \theta k_1 + (1 - \theta)k_2, t_n + h) \\ y_{n+1} &= y_n + \frac{1}{6}(k_1 + 4k_2 + k_3) \end{aligned} \tag{9.3.14}$$

To optimize θ, we expand k_1, k_2, and k_3 in Taylor series as

$$k_1 = hf \tag{9.3.15a}$$

$$k_2 = hf + \frac{1}{2}h^2(f_t + f_y f) + \frac{1}{8}h^3(f_{tt} + 2f_{ty}f + f_{yy}f^2) \tag{9.3.15b}$$

$$k_3 = hf + h^2(f_t + f_y f) + \frac{1}{2}h^3[f_{tt} + 2f_{ty}f + f_{yy}f^2 + (1 - \theta)(f_t + f_y f)f_y] \tag{9.3.15c}$$

where f and its derivatives are evaluated at t_n. By introducing Eq. (9.3.15) into Eq. (9.3.14) and comparing it to Eq. (9.3.5), we find that $\theta = -1$ is the optimum because Eq. (9.3.14) then agrees with Eq. (9.3.5) to the third-order term.

The foregoing derivation may be more easily understood if it is applied to the test equation $y' = \alpha y$.

In summary, the third-order-accurate Runge-Kutta method is written as

$$\begin{aligned} k_1 &= hf(y_n, t_n) \\ k_2 &= hf\left(y_n + \frac{1}{2}k_1, t_n + \frac{h}{2}\right) \\ k_3 &= hf(y_n - k_1 + 2k_2, t_n + h) \\ y_{n+1} &= y_n + \frac{1}{6}(k_1 + 4k_2 + k_3) \end{aligned} \tag{9.3.16}$$

9.3.3 Fourth-Order Runge-Kutta Method

Derivation of the fourth-order Runge-Kutta method is similar to that of the third-order method except one more intermediate step of evaluating the derivative is used. There are several alternative choices for the numerical integration scheme to be used in Eq. (9.3.2). The fourth-order Runge-Kutta method is accurate to the fourth-order term of the Taylor expansion, so the local error is proportional to h^5.

The following two versions of the fourth-order Runge-Kutta method are most popularly used. The first version is based on the Simpson's 1/3 rule and is written as

$$\begin{aligned} k_1 &= hf(y_n, t_n) \\ k_2 &= hf\left(y_n + \frac{k_1}{2}, t_n + \frac{h}{2}\right) \\ k_3 &= hf\left(y_n + \frac{k_2}{2}, t_n + \frac{h}{2}\right) \\ k_4 &= hf(y_n + k_3, t_n + h) \\ y_{n+1} &= y_n + \frac{1}{6}[k_1 + 2k_2 + 2k_3 + k_4] \end{aligned} \tag{9.3.17}$$

The second version is based on the Simpson's 3/8 rule and is written as

$$\begin{aligned} k_1 &= hf(y_n, t_n) \\ k_2 &= hf\left(y_n + \frac{k_1}{3}, t_n + \frac{h}{3}\right) \\ k_3 &= hf\left(y_n + \frac{k_1}{3} + \frac{k_2}{3}, t_n + \frac{2h}{3}\right) \\ k_4 &= hf(y_n + k_1 - k_2 + k_3, t_n + h) \\ y_{n+1} &= y_n + \frac{1}{8}[k_1 + 3k_2 + 3k_3 + k_4] \end{aligned} \tag{9.3.18}$$

Example 9.7

Calculate $y(1)$ by solving

$$y' = -1/(1 + y^2), \quad y(0) = 1$$

using the fourth-order Runge-Kutta method with $h = 1$.

⟨Solution⟩

We set

$$f(y, t) = -\frac{1}{1 + y^2}$$

and $y_0 = 1$ and $t_0 = 0$. Because we have only one interval, the whole calculations are:

$$k_1 = hf(y_0, t_0) = -\frac{1}{(1 + 1)} = -\frac{1}{2}$$

$$k_2 = hf\left(y_0 + \frac{k_1}{2}, t_0 + \frac{h}{2}\right) = -\frac{1}{(1 + (0.75)^2)} = -0.64$$

$$k_3 = hf\left(y_0 + \frac{k_2}{2}, t_0 + \frac{h}{2}\right) = -\frac{1}{(1 + (0.68)^2)} = -0.6838$$

$$k_4 = hf(y_0 + k_3, t_0 + h) = -\frac{1}{(1 + (0.3161)^2)} = -0.9091$$

$$y_1 = y_0 + \frac{1}{6}[k_1 + 2k_2 + 2k_3 + k_4]$$

$$= 1 + \frac{1}{6}[-0.5 - 2(0.64) - 2(0.6838) - 0.9091] = 0.3238$$

Example 9.8

Solve

$$y' = ty + 1, \quad y(0) = 0$$

using the fourth-order Runge-Kutta method, Eq. (9.3.17), with $h = 0.2$, 0.1, and 0.05, respectively, and evaluate the error for each h at $t = 1$, 2, 3, 4, and 5.

⟨Solution⟩

Computations for this example were performed by using PROGRAM 9–2. The results follow:

	$h = 0.2$		$h = 0.1$		$h = 0.05$	
t	y	p.e.	y	p.e.	y	p.e.[a]
1	1.41067	(0.00)	1.41069	(0.00)	1.41068	(0.00)
2	8.83839	(0.01)	8.83937	(0.00)	8.83943	(0.00)
3	112.394	(0.11)	112.506	(0.01)	112.514	(0.00)
4	3716.42	(0.52)	3734.23	(0.04)	3735.72	(0.00)
5	330549.	(1.71)	335798.	(0.15)	336273.	(0.01)

[a] p.e.: percentage error

A comparison of these results to those of Euler methods reveals that the error of the fourth-order Runge-Kutta method with $h = 0.1$ is comparable to that of the modified Euler method with $h = 0.01$. Also, the fourth-order Runge-Kutta method with $h = 0.2$ is comparable to the forward Euler method with $h = 0.001$.

Application of the fourth-order Runge-Kutta method to a set of ordinary differential equations is very similar to that of the second-order Runge-Kutta method. For simplicity of explanation, we consider a set of two equations:

$$\begin{aligned} y' &= f(y, z, t) \\ z' &= g(y, z, t) \end{aligned} \tag{9.3.19}$$

The fourth-order Runge-Kutta method for the set of two equations becomes

$$\begin{aligned} k_1 &= hf(y_n, z_n, t_n) \\ l_1 &= hg(y_n, z_n, t_n) \\ k_2 &= hf\left(y_n + \frac{k_1}{2}, z_n + \frac{l_1}{2}, t_n + \frac{h}{2}\right) \\ l_2 &= hg\left(y_n + \frac{k_1}{2}, z_n + \frac{l_1}{2}, t_n + \frac{h}{2}\right) \\ k_3 &= hf\left(y_n + \frac{k_2}{2}, z_n + \frac{l_2}{2}, t_n + \frac{h}{2}\right) \\ l_3 &= hg\left(y_n + \frac{k_2}{2}, z_n + \frac{l_2}{2}, t_n + \frac{h}{2}\right) \\ k_4 &= hf(y_n + k_3, z_n + l_3, t_n + h) \\ l_4 &= hg(y_n + k_3, z_n + l_3, t_n + h) \end{aligned} \tag{9.3.20}$$

$$y_{n+1} = y_n + \frac{1}{6}[k_1 + 2k_2 + 2k_3 + k_4] \tag{9.3.21}$$

$$z_{n+1} = z_n + \frac{1}{6}[l_1 + 2l_2 + 2l_3 + l_4] \tag{9.3.22}$$

Even when the number of equations in a set is greater than two, the derivation of the fourth-order Runge-Kutta method is essentially the same. A program to solve a set of equations using the fourth-order Runge-Kutta method is given as PROGRAM 9–3.

Example 9.9

Repeat the problem in Example 9.3 by using the fourth-order Runge-Kutta method with $h = 0.2\pi$ and $h = 0.05\pi$.

⟨Solution⟩

PROGRAM 9–3 is used to produce the following results:

	Exact		$h = 0.2\pi$		$h = 0.05\pi$	
t	$y = \cos(t)$	$z = -\sin(t)$	y	z	y	z
0.5π	0	−1	1.23E−4	−0.99997	1.32E−6	−0.99999
π	−1	0	−0.99993	−2.48E−4	−0.99999	−2.65E−6
1.5π	0	1	−3.72E−4	0.99990	−3.96E−6	0.99999
2π	1	0	0.99987	4.95E−4	0.99999	5.29E−6
3π	−1	0	−0.99989	−7.43E−4	−0.99999	−7.94E−6
6π	1	0	0.99960	1.49E−3	0.99999	1.57E−5
8π	1	0	0.99947	1.98E−3	0.99999	2.11E−5

Comparing these values to the results of the forward Euler solution in Example 9.3, the accuracy of the fourth-order Runge-Kutta method even with $h = 0.2\pi$ is significantly better than the forward Euler method with $h = 0.01\pi$.

9.3.5 Error, Stability, and Grid Interval Optimization

The Runge-Kutta methods are subject to two kinds of errors—truncation error and instability. As discussed earlier, the truncation error is due to the discrepancy between the Taylor expansion of the numerical method and the Taylor expansion of the exact solution. The amount of error decreases as the order of the method becomes higher. On the other hand, instability is an accumulated effect of the local error such that the error of the solution grows unboundedly as the time steps are advanced.

To analyze the instability of a Runge-Kutta method, let us consider the test equation

$$y' = \alpha y \tag{9.3.23}$$

where $\alpha < 0$. For a given value of y_n, the exact value for y_{n+1} is analytically given as

$$y_{n+1} = \exp(\alpha h) y_n \tag{9.3.24}$$

Notice that because $\alpha < 0$, $|y_{n+1}|$ decreases as n (or time) increases.

The numerical solution of Eq. (9.3.23) by the fourth-order Runge-Kutta method becomes

$$\begin{aligned} k_1 &= \alpha h y_n \\ k_2 &= \alpha h\left(y_n + \frac{k_1}{2}\right) = \alpha h\left(1 + \frac{1}{2}\alpha h\right) y_n \\ k_3 &= \alpha h\left(y_n + \frac{k_2}{2}\right) = \alpha h\left(1 + \frac{1}{2}\alpha h\left(1 + \frac{1}{2}\alpha h\right)\right) y_n \\ k_4 &= \alpha h(y_n + k_3) = \alpha h\left(1 + \alpha h\left(1 + \frac{1}{2}\alpha h\left(1 + \frac{1}{2}\alpha h\right)\right)\right) y_n \end{aligned} \tag{9.3.25}$$

$$y_{n+1} = \left[1 + \alpha h + \frac{1}{2}(\alpha h)^2 + \frac{1}{6}(\alpha h)^3 + \frac{1}{24}(\alpha h)^4\right] y_n \tag{9.3.26}$$

Equation (9.3.26) equals the first five terms of the Taylor expansion for the right side of Eq. (9.3.24) about t_n. The factor

$$\gamma = 1 + \alpha h + \frac{1}{2}(\alpha h)^2 + \frac{1}{6}(\alpha h)^3 + \frac{1}{24}(\alpha h)^4 \tag{9.3.27}$$

in Eq. (9.3.26) is approximating $\exp(\alpha h)$ of Eq. (9.3.24), so the truncation error and instability of Eq. (9.3.26) both originate in this approximation.

Equation (9.3.27) and $\exp(\alpha h)$ are plotted together in Figure 9.1 for comparison. The figure indicates that if $\alpha < 0$ and the modulus (absolute value) of αh increases, the deviation of γ from $\exp(\alpha h)$ increases, so that the error of the Runge-Kutta method increases. Particularly, if $\alpha h \leqslant -2.785$, the method becomes unstable because the modulus of the numerical solution grows in each step whereas the modulus of the true solution decreases by a factor, $\exp(\alpha h)$, in each step.

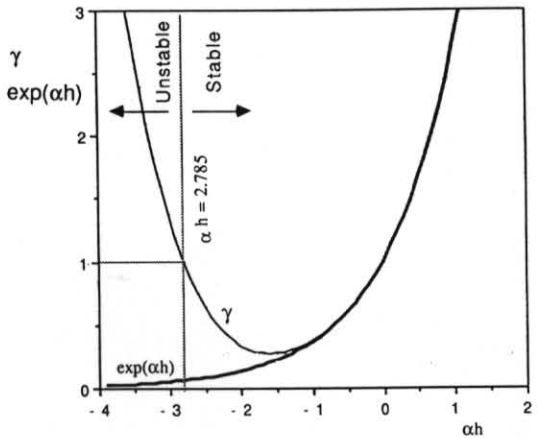


Figure 9.1 Domain of stability for the fourth-order Runge-Kutta method

In practical applications of the Runge-Kutta method, the size of an optimal grid interval can be determined in the following way. For illustration purposes, suppose we desire to keep the local error of the third-order Runge-Kutta method less than ξ. The local error of the third-order Runge-Kutta method for a test interval h is proportional to h^4, so we express the local error in the form

$$E_h = Bh^4 \tag{9.3.28}$$

where B is a constant that depends on the given problem. If we apply the same Runge-Kutta method in two steps with $h/2$ as the time interval, the error becomes proportional to $(h/2)^4$ times 2, where the factor 2 is due to the accumulation of error in two steps. Thus, we have another relation,

$$2E_{h/2} = 2B\left(\frac{h}{2}\right)^4 = \frac{1}{8}Bh^4 \tag{9.3.29}$$

By subtracting Eq. (9.3.29) from Eq. (9.3.28), we get

$$E_h - 2E_{h/2} = Bh^4 - \frac{1}{8}Bh^4 = \frac{7}{8}Bh^4 \tag{9.3.30}$$

The left side of the foregoing equation may be evaluated by a numerical experiment—that is, by running the scheme twice starting from the same initial value. In the first run, only one time step is advanced using a trial value for h as the time interval. We denote the result of this calculation as $[y_1]_h$. In the second run, $[y_2]_{h/2}$ is calculated in two time steps using $h/2$ as the time interval. Using the results of those two calculations, the left side of Eq. (9.3.30) is evaluated as

$$E_h - 2E_{h/2} = [y_1]_h - [y_2]_{h/2} \tag{9.3.31}$$

Introducing Eq. (9.3.31) into Eq. (9.3.30) and solving for B yields

$$B = \frac{8}{7}([y_1]_h - [y_2]_{h/2})/h^4 \tag{9.3.32}$$

Once B is determined, the maximum (or optimum) h that satisfies the criterion $E_h \leqslant \xi$ may be found by introducing $E_h = \xi$ into Eq. (9.3.28) and solving for h, as follows:

$$h = \left(\frac{\xi}{B}\right)^{0.25} \tag{9.3.33}$$

The theory we have just described is reminiscent of the Romberg integration explained in Section 3.2.

Example 9.10

Assume that a fourth-order Runge-Kutta method is applied to

$$y' = -\frac{y}{1+t^2}, \quad y(0) = 1$$

find an optimal step interval satisfying $E_h \leqslant 0.00001$.

⟨Solution⟩

For the fourth-order Runge-Kutta method, the local error is expressed by

$$E_h = Bh^5 \tag{A}$$

The approach is very similar to Eqs. (9.3.28) through (9.3.33) except that the order of error is five. The error accumulated in two steps using $h/2$ is $2E_{h/2} = 2B(h/2)^5$. The difference between the errors of one-step and two-step calculations, namely $E_h - 2E_{h/2}$, is numerically evaluated by

$$2E_h - 2E_{h/2} = [y_1]_h - [y_2]_{h/2}. \tag{B}$$

In Eq. (B), $[y_1]_h$ is the result of the fourth-order Runge-Kutta method for only one step with h, and $[y_2]_{h/2}$ is the result of the same for two steps with $h/2$. Introducing Eq. (A) into Eq. (B) and solving for B, we have

$$B = \frac{16}{15}([y_1]_h - [y_1]_{h/2})/h^5 \qquad \text{(C)}$$

Now we actually run the fourth-order Runge-Kutta method for only one step with $h = 1$ starting with the given initial condition. Then we run it for two steps with $h/2 = 1/2$. The results are

$$[y_1]_1 = 0.4566667 \quad \text{(one interval only)}$$
$$[y_2]_{1/2} = 0.4559973 \quad \text{(two intervals)}$$

From Eq. (C), we obtain B as

$$B = \frac{16}{15}(0.4566667 - 0.4559973)/(1)^5 = 6.3 \times 10^{-4} \qquad \text{(D)}$$

By introducing this into Eq. (A), the local error for any h is expressed by

$$E_h = 6.3 \times 10^{-4} h^5$$

The maximum h that satisfies the given criterion, $E_h < 0.00001$, is

$$h = (0.00001/6.3 \times 10^{-4})^{1/5} = 0.44 \qquad \text{(E)}$$

Summary of This Section

(a) The Runge-Kutta methods are derived by integrating the first order ODE with numerical integration methods. The second-order Runge-Kutta method is identical to the modified Euler method with two iteration cycles as well as to the second-order predictor-corrector method.

(b) A higher-order ODE can be solved by a Runge-Kutta method after it is transformed to a set of first-order ODEs.

(c) Each Runge-Kutta method becomes unstable if α is negative and $|\alpha h|$ exceeds a certain criterion.

(d) The local error of a Runge-Kutta method can be found by running the same method twice: the first time for one interval with a value of h, and the second time for two intervals with $h/2$.

9.4 PREDICTOR-CORRECTOR METHODS

9.4.1 Third-Order Adams Predictor-Corrector Method

A predictor-corrector method consists of a predictor step and a corrector step in each interval. The predictor estimates the solution for the new point, and then the corrector improves its accuracy. Predictor-corrector methods use the solutions for previous points instead of using intermediate points in each interval.

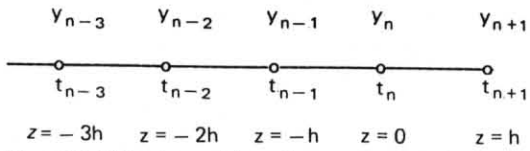


Figure 9.2 Grid points used in predictor-corrector methods

To explain the methods, let us consider an equispaced time interval and assume that the solution has been calculated up to time point n so that the values of y and y' on the previous time points may be used for the calculation of y_{n+1}.

Both predictor and corrector formulas are derived by introducing an appropriate polynomial approximation for $y'(t)$ into Eq. (9.3.2). The most primitive member of the predictor-corrector methods is the second-order predictor-corrector method, which is identical to the second-order Runge-Kutta method.

Let us derive a third-order predictor by approximating $y' = f(y, t)$ with a quadratic interpolation polynomial fitted to f'_n, y'_{n-1} and y'_{n-2}:

$$y'(z) = \frac{1}{2h^2}[(z + h)(z + 2h)y'_n - 2z(z + 2h)y'_{n-1} + z(z + h)y'_{n-2}] + E(z) \qquad (9.4.1)$$

where z is a local coordinate defined by

$$z = t - t_n$$

and $E(z)$ is the error (see Section 2.3). Equation (9.4.1) is the Lagrange interpolation fitted to the values y'_n, y'_{n-1} and y'_{n-2}. The error of the polynomial is

$$E(z) = \frac{1}{3!} z(z + h)(z + 2h)y^{(iv)}(\xi), \quad t_{n-2} < \xi < t_{n+1} \qquad (9.4.2)$$

Here, the derivative in the error term is of the fourth order because a quadratic polynomial is fitted to y'.

Equation (9.3.2) can be rewritten in terms of the local coordinate $z = t - t_n$ as

$$y_{n+1} = y_n + \int_0^h y'(z)\,dz \qquad (9.4.3)$$

Introducing Eq. (9.4.1) into Eq. (9.4.3) yields

$$y_{n+1} = y_n + \frac{h}{12}(23y'_n - 16y'_{n-1} + 5y'_{n-2}) + 0(h^4) \qquad (9.4.4)$$

Equation (9.4.4) is called the third-order Adams-Bashforth predictor formula. The error of Eq. (9.4.4) is attributable to Eq. (9.4.2) and is evaluated by integrating Eq.

(9.4.2) in $[0, h]$, as follows:

$$0(h^4) = \frac{3}{8}h^4 y^{(iv)}(\xi), \quad t_{n-2} < \xi < t_{n+1}$$

In deriving Eq. (9.4.4), notice that Eq. (9.4.1) has been used as an extrapolation. As pointed out in Section 2.9, extrapolation is less accurate than interpolation (See Section 2.9 and Appendix A). Therefore, Eq. (9.4.4) is used only as a predictor and is written as

$$\bar{y}_{n+1} = y_n + \frac{h}{12}(23y'_n - 16y'_{n-1} + 5y'_{n-2}) + 0(h^4) \tag{9.4.5}$$

where the overbar indicates a predictor.

To derive a corrector formula, a predicted value of y'_{n+1} denoted by $\bar{y}'_{n+1}$ is necessary, which is calculated by introducing $\bar{y}_{n+1}$ into $y'(t) = f(y, t)$ as

$$\bar{y}'_{n+1} = f(\bar{y}_{n+1}, t_{n+1})$$

The quadratic polynomial fitted to $\bar{y}'_{n+1}$, y'_n, and y'_{n-1} is written as

$$y'(z) = \frac{1}{2h^2}[z(z+h)\bar{y}'_{n+1} - 2(z-h)(z+h)y'_n + z(z-h)y'_{n-1}] + E(z) \tag{9.4.6}$$

where z is the local coordinate defined after Eq. (9.4.1). The error of this equation is

$$E(z) = \frac{1}{3!}(z-h)z(z+h)y^{(iv)}(\xi), \quad t_{n-1} < \xi < t_{n+1}$$

Introducing Eq. (9.4.6) into Eq. (9.4.3) yields a corrector formula as

$$y_{n+1} = y_n + \frac{h}{12}(5\bar{y}'_{n+1} + 8y'_n - y'_{n-1}) + 0(h^4) \tag{9.4.7}$$

The error is

$$0(h^4) = -\frac{1}{24}h^4 y^{(iv)}(\xi), \quad t_{n-1} < \xi < t_{n+1}$$

Equation (9.4.7) is named the *Adams-Moulton corrector formula of order 3*. The set of Eqs. (9.4.5) and (9.4.7) is called the *third-order Adams predictor-corrector method.*

As seen from the preceding derivation, numerous formulas can be derived by changing the choice of the extrapolating and interpolating polynomials.

In discussing the predictor-corrector methods, we have assumed that the solutions for previous points are available. The third-order predictor-corrector method needs three previous values of y as explained earlier. Therefore, to start up the method, the solutions for $n = 0$, $n = 1$, and $n = 2$ are necessary, the first of which is given by an initial condition, but the second and third should be provided by some other means than the predictor-corrector method, such as a Runge-Kutta method.

Example 9.11

Repeat the problem of Example 9.8 using the third-order Adams predictor-corrector method with $h = 0.1$ and 0.01.

⟨Solution⟩

Since the predictor-corrector methods cannot be self-started, we use the fourth-order Runge-Kutta method to provide the solution for the first two intervals. For the present computation, PROGRAM 9–3 is modified by incorporating the predictor-corrector method, so y_1 and y_2 are calculated by the Runge-Kutta method, and then the remainder is calculated by the predictor-corrector method. The program used is given as PROGRAM 9–4. The computational results are as follows:

	$h = 0.1$		$h = 0.01$	
t	y	(p.e.)[a]	y	(p.e.)[a]
1	1.41091	(−0.01)	1.41069	(0.0000)
2	8.84404	(−0.05)	8.83943	(−0.0001)
3	112.644	(−0.12)	112.514	(−0.0004)
4	3740.07	(−0.11)	3736.00	(0.0004)
5	335593	(0.22)	336344.	(0.0009)

[a] p.e.: percentage error

9.4.2 Fourth-Order Adams Predictor-Corrector Method

The interpolation polynomial fitted to y' at points, n, $n-1$, $n-2, \ldots, n-m$ may be written in the Newton backward formula [see Eq. (2.4.14)] as

$$g_m(t) = \sum_{k=0}^{m} (-1)^k \binom{s+k-1}{k} \Delta^k y'_{n-k} \tag{9.4.8}$$

where

$$s = \frac{t - t_n}{h}$$

By introducing Eq. (9.4.8) into Eq. (9.3.2) as an approximation for $f(y, t)$, we obtain the Adams-Bashforth predictor formula of order $m + 1$:

$$\bar{y}_{n+1} = y_n + h[b_0 y'_n + b_1 \Delta y'_{n-1} + \cdots + b_m \Delta^m y'_{n-m}] \tag{9.4.9}$$

where

$$b_k = \int_0^1 \binom{s+k-1}{k} ds \tag{9.4.10}$$

The first few values of b_k follow:

$$b_0 = 1$$

$$b_1 = \frac{1}{2}$$

$$b_2 = \frac{5}{12}$$

$$b_3 = \frac{3}{8}$$

$$b_4 = \frac{251}{720}$$

If we set $m = 2$ in Eq. (9.4.9) for example, we obtain the third-order predictor given by Eq. (9.4.4). By following the same procedure for $m = 3$, the fourth-order predictor formula is derived as

$$\bar{y}_{n+1} = y_n + \frac{9h}{24}(55y'_n - 59y'_{n-1} + 37y'_{n-2} - 9y'_{n-3}) + 0(h^5) \tag{9.4.11}$$

where

$$0(h^5) = \frac{251}{720} h^5 y^{(v)}(\xi), \quad t_{n-3} < \xi < t_{n+1}$$

The corrector formulas may be derived by using the polynomial fitted to y' at grid points, $n + 1, n, n - 1, \ldots, n - m + 1$. The Newton backward interpolation formula fitted to y' at these points (see Section 2.5) is

$$g_m(t) = \sum_{k=0}^{m} \binom{s+k-2}{k} \Delta^k y'_{n+1-k} \tag{9.4.12}$$

Introducing this equation into Eq. (9.3.2) yields the Adams-Moulton corrector formula:

$$y_{n+1} = y_n + h[c_0 y'_{n+1} + c_1 \Delta y'_n + \cdots + c_m \Delta^m y'_{n-m}] \tag{9.4.13}$$

where

$$c_k = \int_0^1 \binom{s+k-2}{k} ds$$

The first few values of c_k follow:

$$c_0 = 1$$

$$c_1 = -\frac{1}{2}$$

$$c_2 = -\frac{1}{12}$$

$$c_3 = -\frac{1}{24}$$

$$c_4 = -\frac{19}{720}$$

By setting $m = 3$ in Eq. (9.4.13), we obtain the fourth-order Adams-Moulton corrector:

$$y_{n+1} = y_n + \frac{h}{24}(9\bar{y}'_{n+1} + 19y'_n - 5y'_{n-1} + y'_{n-2}) + 0(h^5) \tag{9.4.14}$$

where $y'_n = f(y_n, t_n)$ and

$$0(h^5) = -\frac{19}{720} h^5 y^{(v)}(\xi), \quad t_{n-2} < \xi < t_{n+1}$$

The set of Eqs. (9.4.11) and (9.4.14) is called the *fourth-order Adams predictor-corrector method.*

9.4.3 Advantages and Disadvantages of Predictor-Corrector Methods

An advantage of predictor-corrector methods is their computational efficiency: they use information from previous steps. Indeed, the function $f(y, t)$ is evaluated only twice in each step regardless of the order of the predictor-corrector method, whereas the fourth-order Runge-Kutta method evaluates $f(y, t)$ four times in each interval. Another advantage is that the local error can be detected at each step with a small additional computing effort. The technique of detecting local error is discussed in Subsection 9.4.4. On the other hand, there are some disadvantages as follows:

(a) The method cannot be started by itself because of the use of previous points. Until the solutions for enough points are determined, another method such as a Runge-Kutta method must be used.

(b) Because previous points are used, changing the interval size in the middle of solution is not easy. Although the predictor-corrector formulas may be derived on nonuniformly spaced points, the coefficients of the formulas change for each interval, so programming becomes very cumbersome.

(c) The predictor-corrector method cannot be used if y' becomes discontinuous. This can happen when one of the coefficients of the differential equation changes discontinuously in the middle of the domain.

However, the last two difficulties can be overcome as follows: Because the predictor-corrector program must contain a self-starting method such as a Runge-Kutta method anyway, the computation can be restarted whenever the step interval has to be changed or when y' becomes discontinuous.

9.4.4 Analysis of Local Error and Instability of Predictor-Corrector Methods

One advantage of the predictor-corrector methods is that local error may be evaluated easily by observing the difference between the predictor and the corrector in each step. For illustration of analysis, we consider the third-order Adams predictor-corrector method. Equations (9.4.5) and (9.4.7) indicate that, assuming that $y_n, y_{n-1}, y_{n-2}, \ldots$ are given, the predictor and corrector values become

$$\bar{y}_{n+1} = y_{n+1,\text{exact}} - \frac{3}{8}h^4 y^{(\text{iv})}(\xi) \tag{9.4.15}$$

$$y_{n+1} = y_{n+1,\text{exact}} + \frac{1}{24}h^4 y^{(\text{iv})}(\xi) \tag{9.4.16}$$

If we assume further that the values of the fourth derivative in Eqs. (9.4.15) and (9.4.16) take the same value, then subtracting Eq. (9.4.16) from Eq. (9.4.15) yields

$$\bar{y}_{n+1} - y_{n+1} = -\frac{10}{24}h^4 y^{(\text{iv})}(\xi) \tag{9.4.17}$$

Backsubstituting Eq. (9.4.17) into Eq. (9.4.16) yields

$$y_{n+1,\text{exact}} - y_{n+1} = \frac{1}{10}(\bar{y}_{n+1} - y_{n+1}) \tag{9.4.18}$$

The right side of Eq. (9.4.18) is the local error of the corrector. Because it is expressed in terms of the difference between the predictor and the corrector, the calculation is simple. By using this algorithm at every step interval, the local error of the method can be automatically monitored in a program.

Now we study the stability of a predictor-corrector method by considering again the third-order Adams predictor-corrector method given by Eq. (9.4.5) and Eq. (9.4.7). Suppose we apply the method to the test equation given by

$$y' = \alpha y \tag{9.4.19}$$

Introducing Eq. (9.4.19) into Eqs. (9.4.5) and (9.4.7) yields

$$\bar{y}_{n+1} = y_n + \frac{\alpha h}{12}(23y_n - 16y_{n-1} + 5y_{n-2})$$

$$y_{n+1} = y_n + \frac{\alpha h}{12}(5\bar{y}_{n+1} + 8y_n - y_{n-1})$$

Eliminating $\bar{y}_{n+1}$ in the foregoing two equations and reorganizing the terms lead to

$$y_{n+1} = -a_2 y_n - a_1 y_{n-1} - a_0 y_{n-2} \tag{9.4.20}$$

where

$$a_2 = -(1 + 13b + 115b^2)$$
$$a_1 = b + 80b^2$$
$$a_0 = -25b^2$$
$$b = \frac{\alpha h}{12}$$

Equation (9.4.20) may be considered as the initial value problem of a difference equation, for which the analytical solution may be obtained in a similar way as for a third-order linear ordinary differential equation. Indeed, the analytical solution of Eq. (9.4.20) may be found in the form,

$$y_n = c\gamma^n \tag{9.4.21}$$

where γ is a characteristic value and c is a constant. By introducing Eq. (9.4.21) into Eq. (9.4.20), we get the characteristic equation:

$$\gamma^3 + a_2\gamma^2 + a_1\gamma + a_0 = 0 \tag{9.4.22}$$

Equation (9.4.22) is a third-order polynomial equation, so it has three roots although two of them can be complex values. We denote the three roots by

$$\gamma_1, \quad \gamma_2, \quad \text{and} \quad \gamma_3$$

Because each of γ_1, γ_2, and γ_3 satisfies Eq. (9.4.20), a linear combination of all the solutions is also a solution of Eq. (9.4.20). The general solution of Eq. (9.4.20) may

now be written as

$$y_n = c_1(\gamma_1)^n + c_2(\gamma_2)^n + c_3(\gamma_3)^n \tag{9.4.23}$$

where c_1, c_2, and c_3 are determined when the initial values of y_0, y_1, and y_2 are given (remember that the third-order predictor-corrector method needs three starting values).

The exact solution to the original problem, Eq. (9.4.19), is given by

$$y_n = y(0) \exp(\alpha n h) \tag{9.4.24}$$

where $y(0)$ is the initial value of $y(t)$. The question is how each of the three terms in Eq. (9.4.23) is related to Eq. (9.4.24). The answer is that one term in Eq. (9.4.23) is an approximation for Eq. (9.4.24), but the other two are irrelevant to the true solution and constitute a part of the error of the scheme. We assume that the first term is the approximation and that the second and the third are the errors. Instability of the method is then related to the second and third terms. If these error terms vanish as n increases, there is no instability. If the magnitude of these error terms becomes greater than unity, erratic behavior of the numerical solution occurs. This is the instability, and it happens if

$$|\gamma_2| > 1 \quad \text{or} \quad |\gamma_3| > 1 \quad \text{or both}$$

When the predictor-corrector method is applied to Eq. (9.4.19), both α and h affect the instability. However, because α and h always appear as a product [see Eq. (9.4.20)], we can consider αh as one parameter. The roots of Eq. (9.4.22) for various values of αh are shown in Table 9.2.

Table 9.2 Characteristic values of the third-order predictor-corrector method applied to $y'(t) = \alpha y(t)$

αh	exp (αh)	γ_1	Percentage Error	γ_2, γ_3	$\lvert\gamma_2\rvert, \lvert\gamma_3\rvert$
0.1	1.1051	1.1051	0	$0.006 \pm 0.039j$	0.040
0.2	1.2214	1.2214	0	$0.014 \pm 0.074j$	0.075
0.5	1.6487	1.6477	0.06	$0.047 \pm 0.155j$	0.162
1.0	2.7183	2.6668	1.90	$0.108 \pm 0.231j$	0.255
1.5	4.4816	4.1105	8.3	$0.155 \pm 0.266j$	0.308
2.0	7.3891	5.9811	23.	$0.190 \pm 0.283j$	0.341
2.5	12.1825	8.2705	32.	$0.215 \pm 0.292j$	0.362
−0.1	0.9048	0.9048	0	$-0.003 \pm 0.043j$	0.043
−0.2	0.8187	0.8189	0.02	$-0.002 \pm 0.092j$	0.092
−0.3	0.7408	0.7416	0.1	$-0.003 \pm 0.145j$	0.145
−0.4	0.6703	0.6732	0.43	$-0.011 \pm 0.203j$	0.203
−0.5	0.6065	0.6147	1.35	$0.022 \pm 0.265j$	0.266
−1.0	0.3678	0.4824	31.2	$0.116 \pm 0.588j$	0.600
−1.5	0.2231	0.4944	121.	$0.338 \pm 0.821j$	0.889
−2.0	0.1353	0.5650	419.	$0.731 \pm 0.833j$	1.109

$j = \sqrt{-1}$

In Table 9.2, γ_1 is the root relevant to the true solution; indeed, it is an approximation to exp (αh). The other two γs are irrelevant roots. The last column shows the magnitude of the second and the third roots. It is seen that when $\alpha h > 0$ (that is, $\alpha > 0$), the magnitudes of γ_2 and γ_3 are always smaller than γ_1. Therefore, as n increases, the magnitude of the second and third terms relative to the first term diminishes. Thus, there is no instability if $\alpha > 0$.

In the second half of Table 9.2 where $\alpha < 0$, the relevant root γ_1 is always smaller than 1, and it decreases as αh becomes more negative. When the magnitude of αh is very small, the irrelevant roots are smaller than the relevant root, but the magnitude of the irrelevant roots keeps increasing and exceeds the relevant roots before αh reaches -1. The magnitude of the irrelevant root exceeds unity approximately when $\alpha h = -1.8$. If the magnitude of the irrelevant root exceeds unity, the second and third terms of Eq. (9.4.20) will show an erratic behavior. That is, while the first term approaches 0, the second and the third terms diverge with oscillatory behavior. Therefore, instability of the third-order predictor-corrector occurs if $\alpha h < -1.8$.

Table 9.2 also provides important information on the accuracy of the third-order Adams predictor-corrector method. As described previously, the first root γ_1 in Table 9.2 approximates exp (αh). The discrepancy between the γ_1 and exp (αh) is a direct measure of the local error. The table shows that, for $\alpha > 0$, the percentage of error is small until αh reaches 0.5. For $\alpha < 0$, the percentage of error increases quickly as $|\alpha h|$ increases, and for $\alpha h = -0.5$, which is still far from the instability domain, the percentage of error is significant.

Summary of This Section

(a) A predictor-corrector method consists of a predictor and a corrector.

(b) The predictors of the Adams predictor-corrector methods are named Adams-Bashforth predictors. They are derived by integrating a polynomial extrapolation of y' for the previous points.

(c) The correctors of the Adams predictor-corrector methods are named Adams-Moulton predictors, and they are derived by integrating a polynomial interpolation of y' for the previous points plus $\bar{y}'$ (the predicted value for the new point).

(d) The second-order predictor-corrector method is identical with the second-order Runge-Kutta method.

(e) The third- and fourth-order predictor-corrector methods cannot be self-started. However, once started, their computational efficiency is higher than that of the Runge-Kutta method. Error check in each interval is easier than for the Runge-Kutta method.

9.5 MORE APPLICATIONS

In this section, five applications of the numerical methods for initial value problems are shown. Although the fourth-order Runge-Kutta method is used throughout this

section, it can be replaced by any other method for ordinary differential equations described in this chapter.

Example 9.12

A metal piece of 0.1 kg mass and 200° C (or 473° K) is suddenly placed in a room of temperature 25° C, where it is subject to both natural convection cooling and radiation heat transfer. Assuming that the temperature distribution in the metal is uniform, the equation for the temperature may be written as

$$\frac{dT}{dt} = \frac{A}{\rho c v}[\varepsilon\sigma(297^4 - T^4) + h_c(297 - T)], \quad T(0) = 473 \tag{A}$$

where T is the temperature in degrees Kelvin, and we assume that the constants are given by

$\rho = 300$ kg/m^3	(density of the metal)
$v = 0.001$ m^3	(volume of the metal)
$A = 0.25$ m^2	(surface area of the metal)
$c = 900$ J/kgK	(specific heat of the metal)
$h_c = 30$ J/m^{2}K	(heat transfer coefficient)
$\varepsilon = 0.8$	(emissivity of the metal)
$\sigma = 5.67 \times 10^{-8}$ w/m^{2}K^4	(Stefan-Boltzmann constant)

⟨Solution⟩

This problem can be solved by modifying PROGRAM 9–2, which uses the fourth-order Runge-Kutta method. Temperatures calculated by the fourth-order Runge-Kutta method with $h = 1$ sec follow for selected values of t:

t (sec)	T (° K)
0	473
10	418.0
20	381.7
30	356.9
60	318.8
120	300.0
180	297.4

Example 9.13

The electric current of the circuit shown in Figure E9.13a satisfies the integro-differential equation

$$L\frac{di}{dt} + Ri + \frac{1}{C}\int_0^t i(t')\,dt' + \frac{1}{C}q(0) = E(t), \quad t > 0 \tag{A}$$

where the switch is closed at $t = 0$; $i = i(t)$ is the current (amp); R is a resistance (ohm); L, C, and E are given by

$$L = 200 \text{ henry}$$
$$C = 0.001 \text{ farad}$$
$$E(t) = 1 \text{ volt for } t > 0$$

Initial conditions are $q(0) = 0$ (capacitor's initial charge) and $i(0) = 0$. Calculate the current for $0 \leqslant t \leqslant 5$ sec after closing the switch ($t = 0$) for the following four values of R:

(a) $R = 0$ ohm
(b) $R = 50$ ohm
(c) $R = 100$ ohm
(d) $R = 300$ ohm

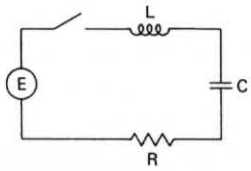


Figure E9.13a Electric circuit

⟨Solution⟩

We first define

$$q(t) = \int_0^t i(t')\,dt' \tag{B}$$

Differentiating Eq. (B) yields

$$\frac{d}{dt}q(t) = i(t), \quad q(0) = 0 \tag{C}$$

Introducing Eq. (B) into Eq. (A) and rewriting give

$$\frac{d}{dt}i(t) = -\frac{R}{L}i(t) - \frac{1}{LC}q(t) + \frac{1}{LC}q(0) + \frac{E(t)}{L}, \quad i(0) = 0 \tag{D}$$

Thus, Eq. (A) is transformed to a set of two first-order ODEs, Eqs. (C) and (D). PROGRAM 9–4 was modified for the present problem in two respects (see note below). The result of the computation is shown in a graphic form in Figure E9.13b.

Note: (a) To perform the calculations for all four cases in one run, four pairs of coupled first-order ODEs are incorporated, the first pair corresponding to the first case, the second pair to the second case, and so on. This is possible because not all equations in the program have to be mathematically coupled. (b) A graphic plotting routine is added, so all four cases are plotted on one graphic output.

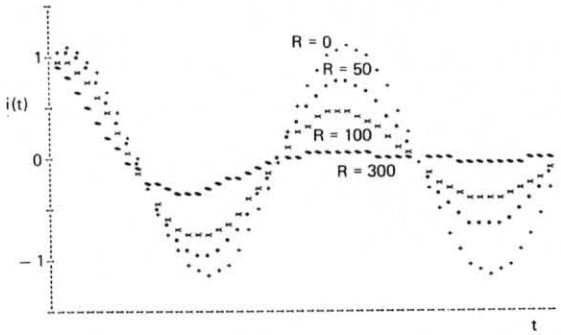


Figure E9.13b Graph of computed results

Example 9.14

The three-mass system is shown in the figure below. The displacements of the three masses satisfy the equations given by

$$\begin{aligned} M_1y_1'' + B_1y_1' + K_1y_1 - B_1y_2' - K_2y_2 &= F_1(t) \\ -B_1y_1' - K_1y_1 + M_2y_2'' + B_1y_2' + (K_1 + K_2)y_2 - K_2y_3 &= 0 \\ -K_2y_2 + M_3y_3'' + B_2y_3' + (K_2 + K_3)y_3 &= F_3(t) \end{aligned} \tag{A}$$

Constants and initial conditions are as follows:

$K_1 = K_2 = K_3 = 1$	(spring constants, kgm/s^2)
$M_1 = M_2 = M_3 = 1$	(mass, kg)
$F_1(t) = 1, F_3(t) = 0$	(force, Newton)
$B_1 = B_2 = 0.1$	(damping coefficients, kg/s)
$y_1(0) = y_1'(0) = y_2(0) = y_2'(0) = y_3(0) = y_3'(0) = 0$	(initial conditions)

Solve the foregoing equations by using the fourth-order Runge-Kutta method for $0 \leqslant t \leqslant 30$ sec with $h = 0.1$

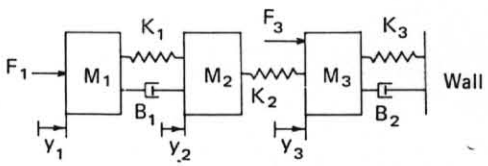


Figure E9.14a Masses-springs system

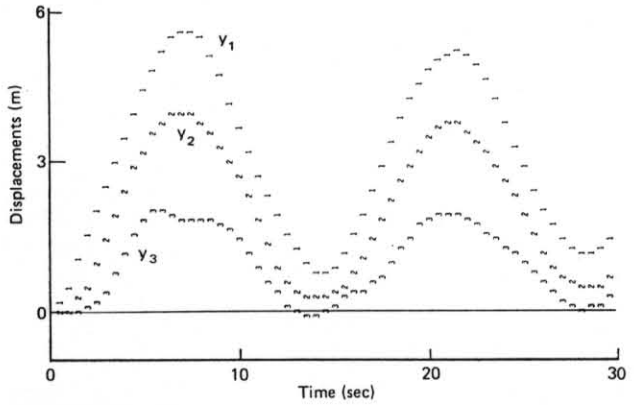


Figure E9.14b Results of computations

⟨Solution⟩

By defining

$$y_4 = y_1', \quad y_5 = y_2', \quad \text{and} \quad y_6 = y_3' \tag{B}$$

Eq. (A) is written as a set of six first-order ODEs as follows:

$$y_1' = y_4 \tag{C1}$$

$$y_2' = y_5 \tag{C2}$$

$$y_3' = y_6 \tag{C3}$$

$$y_4' = \frac{1}{M_1}[-B_1y_4 - K_1y_1 + B_1y_5 + K_2y_2 + F_1] \tag{C4}$$

$$y_5' = \frac{1}{M_2}[B_1y_4 + K_1y_1 - B_1y_5 - (K_1 + K_2)y_2 + K_2y_3] \tag{C5}$$

$$y_6' = \frac{1}{M_3}[K_2y_2 - B_2y_6 - (K_2 + K_3)y_3 + F_3] \tag{C6}$$

These equations are solved by modifying PROGRAM 9–3. The computational results are shown in Figure E9.14b.

Example 9.15

A rod 1.0 m long placed in a vacuum is heated by an electric current through the rod. The temperature at both ends is fixed at 273° K. The heat is dissipated from the surface by radiation heat transfer to the environment whose

temperature is 273° K. Using the following constants, determine the temperature distribution in the axial direction:

$k = 60$ W/mK	(thermal conductivity)
$Q = 50$ W/m	(heat generation rate per unit length of the bar)
$\sigma = 5.67 \times 10^{-8}$ W/m^{2}K^4	(Stefan-Boltzmann constant)
$A = 0.0001$ m^2	(the cross sectional area)
$P = 0.01$ m	(perimeter of the rod)

⟨Solution⟩

The heat conduction equation in the axial direction x is written as

$$-Ak\frac{d^2}{dx^2}T + P\sigma(T^4 - 273^4) = Q \quad 0 < x < 1.0 \tag{A}$$

with the boundary conditions

$$T(0) = T(1.0) = 273 \text{ K}$$

where T is temperature in degrees Kelvin.

The present problem is a boundary value problem (boundary conditions are specified at $x = 0$ and $x = 1$), but it can be solved as an initial value problem on the trial-and-error basis. By defining y_1 and y_2 as

$$y_1(x) = T(x)$$
$$y_2(x) = T'(x)$$

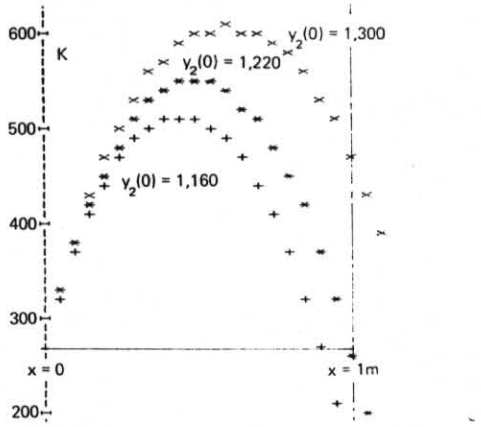


Figure E9.15 Computed results of the shooting method

Eq. (A) may be rewritten as a set of two first-order ODEs as

$$y_1' = y_2 \tag{B}$$
$$y_2' = \frac{P}{Ak}\sigma(y^4 - 273^4) - \frac{Q}{kA}$$

Only one initial condition, $y_1(0) = 273$, is known from the boundary conditions (but $y_2(0)$ is not known). So we solve Eq. (A) with trial values for $y_2(0)$ until the boundary condition for the right end, namely $y_1(1) = 273$, is satisfied. This approach is called the *shooting method* [Rieder/Busby].

For the present example, PROGRAM 9–3 is used with some modifications. The results are directly plotted on a printer and shown in Figure E9.15. It is seen that $y_2(0) = 1160$ is too small as an initial guess, whereas $y_2(0) = 1300$ is too large. Some $y_2(0)$ in between these values should give the best result. After a few more trials, $y_2(0) = 1220$ is found to satisfy almost exactly the right boundary condition.

Example 9.16

The temperature of a perfectly insulated iron bar 55 cm long is initially at 200° C. The temperature of the left edge is suddenly reduced and fixed to 0° C at $t = 0$ sec. Calculate the temperature distribution at every 100 sec until 1000 sec is reached. The property constants are

$k = 80.2$ W/mK	(thermal conductivity)
$\rho = 7870$ kg/m^3	(density)
$c = 0.447$ kJ/kg° K	(specific heat unit)

⟨Solution⟩

We first divide the rod into eleven control volumes as shown in Figure E9.16a. Denoting the average temperature of control volume i by $T_i(t)$, the heat balance equation for control volume i is written as

$$\rho c\Delta xA(dT_i/dt) = (q_{i-1} - q_i)A \tag{A}$$

In Eq. (A), q_i is the heat flux (rate of conduction of heat transfer per unit cross-sectional area) at the boundary of the control volumes i and $i + 1$, and written

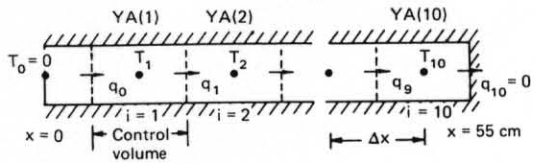


Figure E9.16a An insulated bar

```
INPUT PRINTING OR PLOTTING INTERVAL(P.I.)
 100
INPUT NUMBER OF STEPS IN ONE P.I. OF X
 5
INPUT MAXIMUM X TO STOP CALCULATION
 1005
H= 20      T0  T1  T2  ........                          T10
t=  100    0  109  170  192  198  200  200  200  200  200  200
  (sec)
    200    0   81  141  175  191  197  199  200  200  200  200
    300    0   67  122  160  182  193  197  199  200  200  200
    400    0   58  108  146  172  186  194  198  199  200  200
    500    0   52   99  136  163  180  190  195  198  199  200
    600    0   48   91  127  155  173  186  193  196  198  199
    700    0   44   85  120  147  167  181  190  195  197  198
    800    0   41   80  114  141  162  176  186  192  196  197
    900    0   39   76  108  135  156  172  183  190  194  196
   1000    0   37   72  104  130  152  168  179  187  192  194
```

Figure E9.16b Result of computations

by

$$q_i = -\frac{k}{\Delta x}(T_{i+1} - T_i) \quad \text{for } i = 0, 1, 2, \ldots, 9 \tag{B}$$

and

$$q_{10} = 0 \tag{C}$$

Introducing Eq. (B) into Eq. (A) and rearranging yield

$$\frac{dT_i}{dt} = \frac{k}{\rho c \Delta x^2}(T_{i-1} - 2T_i + T_{i+1}) \tag{D-1}$$

for $i = 1, 2, 3, \ldots, 9$, and

$$\frac{dT_{10}}{dt} = \frac{k}{\rho c \Delta x^2}(T_9 - T_{10}) \tag{D-2}$$

Equation (D) may be considered as a set of first-order ODEs and solved by using one of the Runge-Kutta methods. The set of equations is solved by PROGRAM 9–3 with some modifications. The computed results are shown in Figure E9.16b.

Notes:

(a) Equation (D) may be viewed as a semidifference approximation for the heat conduction equation (parabolic partial differential equation)

$$\frac{\partial}{\partial x} k \frac{\partial}{\partial x} T(x, t) = \rho c \frac{\partial T(x, t)}{\partial t}$$

with the initial condition, $T(x, 0) = 200°\,C$, and the boundary conditions, $T(0, t) = T'(55, t) = 0$.

(b) The present solution technique for the partial differential equation using a numerical method for ODEs is called the *method of lines*.

(c) Space-dependent and time-dependent thermal conductivity can be implemented with a minor change: that is, to recalculate k for each boundary of the control volumes in each time step.

(d) The author's study indicates that the computations using $h = 50$ sec agree well with that of $h = 1$ sec, but the solution scheme becomes unstable with $h = 100$ sec.

9.6 STIFF ODEs

9.6.1 Why Stiff Equations Are Difficult

Stiffness refers to a very short time constant of an ODE. Consider, for example,

$$y' = -\alpha y + s(t), \quad y(0) = y_0 \tag{9.6.1}$$

where $\alpha > 0$. The solution of this equation is, if $s = 0$,

$$y(t) = y_0 e^{-\alpha t} \tag{9.6.2a}$$

and if $s(t) \neq 0$, then

$$y(t) = y_0 e^{-\alpha t} + e^{-\alpha t} \int_0^t s(\xi) e^{\alpha \xi} \, d\xi \tag{9.6.2b}$$

The response of the system to the initial condition as well as to the changes of $s(t)$ is characterized by $1/|\alpha|$ that is called the *time constant*.

Solution of a stiff problem with a standard Runge-Kutta or predictor-corrector method is difficult or, sometimes, impossible. For example, if the fourth-order Runge-Kutta method is used for Eq. (9.6.2a and b), the computation becomes unstable unless $h < 2.785/|\alpha|$ (see Subsection 9.3.5). As the time constant becomes shorter, one has to use a progressively smaller time step. For $\alpha = -100000 \text{ sec}^{-1}$ as an example, h must be smaller than $2.785/100000 = 0.000002785$ sec just to maintain stability. The predictor-corrector methods discussed earlier in this chapter are subject to similar constraints.

When very fast transients of a system are computed, the necessity of small time steps is understandable. On the other hand, when $s(t)$ is a slowly varying function or a constant, the solution changes vary slowly, so naturally we desire to use larger time steps. Nonetheless, the same small time steps are necessary to assure stability of the numerical solution, no matter how slow the actual change of the solution is.

Stiffness is particularly serious for a set of ODEs [Gear (1971); Shampine/Gear (1979); Hall/Watt; Fertziger; Kuo]. If the set of equations contains only one stiff

equation, the stability of a numerical method is governed by the short time constant of the stiffest equation.* For example, in the case of two equations,

$$\begin{aligned} y' &= -y + z + 3 \\ z &= -10^7 z + y \end{aligned} \tag{9.6.3}$$

the second equation has a significantly shorter time constant than the first.

A number of numerical methods that allow a large time step have been proposed including the implicit Runge-Kutta method and the rational Runge-Kutta method. Two such methods are introduced in the remainder of this section.

9.6.2 Implicit Methods

For simplicity, let us consider a set of two ODEs:

$$\begin{aligned} \frac{d}{dt} y &= f(y, z, t) \\ \frac{d}{dt} z &= g(y, z, t) \end{aligned} \tag{9.6.4}$$

Using the backward difference approximation to the left side, we can write an implicit scheme as

$$\begin{aligned} y_{n+1} - y_n &= hf(y_{n+1}, z_{n+1}, t_{n+1}) \equiv hf_{n+1} \\ z_{n+1} - z_n &= hg(y_{n+1}, z_{n+1}, t_{n+1}) \equiv hg_{n+1} \end{aligned} \tag{9.6.5}$$

where the f and g terms on the right side have unknowns, y_{n+1} and z_{n+1}.

If f and g are nonlinear functions, Eq. (9.6.5) cannot be solved in a closed form. However, the iterative solution explained in Subsection 9.2.3 can be applied very easily [Hall/Watt]. Indeed, the reader is encouraged to try it. Unfortunately, it is not computationally efficient for a large system of ODEs. A more efficient approach is to linearize the equations by Taylor expansions [Kubicek; Constantinites]. The Taylor expansion of $f_{k,\,n+1}$ about t_n becomes

$$\begin{aligned} f_{n+1} &= f_n + f_y \Delta y + f_z \Delta z + f_t h \\ g_{n+1} &= g_n + g_y \Delta y + g_z \Delta z + g_t h \end{aligned} \tag{9.6.6}$$

* More strictly speaking, the time constant is an eigenvalue of the system and therefore not a value associated with any single equation in the set. Nonetheless, if one of the equations is much stiffer than the others, that equation determines the smallest time constant and there is very little influence from other equations.

where

$$\Delta y = y_{n+1} - y_n, \quad \Delta z = z_{n+1} - z_n \tag{9.6.7}$$

Introducing Eq. (9.6.6) into Eq. (9.6.5) and using Eq. (9.6.7) yields

$$\begin{bmatrix} 1 - hf_y & -hf_z \\ -hg_y & 1 - hg_z \end{bmatrix} \begin{bmatrix} \Delta y \\ \Delta z \end{bmatrix} = \begin{bmatrix} hf_n + h^2 f_t \\ hg_n + h^2 g_t \end{bmatrix} \tag{9.6.8a}$$

or more compactly

$$(I - hJ)\Delta \bar{y} = RHS \tag{9.6.8b}$$

where

RHS = vector on the right side of Eq. (9.6.8a)
J = the Jacobian matrix defined by

$$J = \begin{bmatrix} f_y & f_z \\ g_y & g_z \end{bmatrix}$$

I = identity matrix
$\Delta \bar{y}$ = col $(\Delta y, \Delta z)$

Equation (9.6.8) is solved by Gauss elimination. The implicit method is unconditionally stable unless nonlinear effects cause instability.

The method has been extended to a larger set of coupled ODEs. The Gear methods [Gear, 1971], which are available in NAG library [NAG], use higher-order backward difference approximations with variable time steps.

9.6.3 Exponential Method

Exponential transformation and exponential fitting have been proposed and used by various researchers to solve stiff ODEs. This subsection gives only a brief introduction of the basic ideas in exponential methods.

To explain the principle, consider a single first-order ODE:

$$y' = f(y, t) \tag{9.6.9}$$

where, for simplicity of the discussions, we assume f does not include t explicitly.

Adding cy to both terms of Eq. (9.6.9) yields

$$y' + cy = f(y, t) + cy \tag{9.6.10}$$

where c is a constant. Using e^{-ct} as an integrating factor, Eq. (9.6.10) is integrated in the interval $[t_n, t_{n+1}]$ as

$$y(t_{n+1}) = y_n e^{-ch} + \int_0^h [f(y(t_n + \xi), t_n + \xi) + cy(t_n + \xi)] e^{c(\xi - h)}\, d\xi \tag{9.6.11}$$

where $t_{n+1} = t_n + h$. Equation (9.6.11) is exact regardless of the choice of c.

Several different numerical schemes may be derived by introducing an approximation for $f + cy$ in the integrand. However, the accuracy of an approximate integration is then affected by the value of c. To find an appropriate value of c, we write y as

$$y(t) = y_n + \delta y(t) \tag{9.6.12}$$

Introducing Eq. (9.6.12) into Eq. (9.6.9) yields

$$\begin{aligned} \delta y' &= f(y_n + \delta y) \\ &= f_n + (f_y)_n \delta y + 0(\delta y^2) \end{aligned} \tag{9.6.13}$$

By ignoring the second-order error term, Eq. (9.6.13) can be equivalently written as

$$y' - (f_y)_n y = f_n - (f_y)_n y_n \tag{9.6.14}$$

which is a linearized approximation for Eq. (9.6.9) about $t = t_n$. If c in Eq. (9.6.10) is set to

$$c = -(f_y)_n \tag{9.6.15}$$

then, Eq. (9.6.10) becomes identical with Eq. (9.6.14). By using

$$y'' = f' = f_y y' = f_y f \tag{9.6.16}$$

Eq. (9.6.15) may also be expressed by

$$c = -(f'/f)_n \tag{9.6.17}$$

An explicit numerical scheme is obtained by setting the terms in the brackets of Eq. (9.6.11) by

$$[f(y, t_n + \xi) + cy(t_n + \xi)] \simeq \hat{f}_n + cy_n \tag{9.6.18}$$

Because the right side of Eq. (9.6.18) is constant, Eq. (9.6.11) reduces to

$$\begin{aligned} y_{n+1} &= y_n e^{ch} + (1/c)(1 - e^{-ch})[f_n + cy_n] \\ &= y_n + hf_n \left[\frac{1 - e^{-ch}}{ch}\right] \end{aligned} \tag{9.6.19}$$

which is known as the *exponentially fitted method* [Bui; Oran; Hetric; Fergason/Hansen]. Not only this method is unconditionally stable but also positivity of the solution is guaranteed whenever the exact solution is expected to be positive.

The errors of Eq. (9.6.19) come from the approximation of Eq. (9.6.18). A more accurate method using an iterative procedure is developed in the remainder of this subsection. Based on Eq. (9.6.19), a predictor for $y(t)$ for $t_n < t < t_{n+1}$ can be set

$$\bar{y}(t) = y_n + \left[\frac{1 - e^{-c\xi}}{c}\right] f_n, \quad \xi = t - t_n \tag{9.6.20}$$

and for t_{n+1},

$$\bar{y}_{n+1} = y_n + \left[\frac{1 - e^{-ch}}{c}\right] f_n$$

By introducing Eq. (9.6.20) into Eq. (9.6.11) we obtain

$$y_{n+1} = \bar{y}_n + \int_{\xi=0}^{h} [f(\bar{y}(t_n + \xi), t_n + \xi) - f_n + cy(t_n + \xi) - cy_n] e^{c(\xi - h)}\, d\xi \tag{9.6.21}$$

The second term of Eq. (9.6.21) is a correction of Eq. (9.6.20), and can be evaluated by any one of the following:

(a) Analytical integration if it is possible.
(b) Approximating the terms in the brackets by a linear interpolation.
(c) Integrating by the trapezoidal rule.

Approach (a) is not easy unless f is a simple function, so we do not consider it any further. To pursue (b), the linear interpolation of the bracketed part is written as

$$[f(\bar{y}(t_n + \xi)) - f_n + cy(t_n + \xi) - cy_n] \cong B\xi \tag{9.6.22}$$

where

$$B = \frac{f_{n+1} - f_n + c(y_{n+1} - y_n)}{h}$$

Introducing Eq. (9.6.22) into Eq. (9.6.21), the corrector becomes

$$y_{n+1} = \bar{y}_{n+1} + \frac{Bh^2}{ch}\left(\frac{1 - e^{-ch}}{ch - 1}\right) \tag{9.6.23}$$

If the trapezoidal rule is used, the corrector becomes

$$y_{n+1} = \bar{y}_{n+1} + \frac{Bh^2}{2} \tag{9.6.24}$$

which agrees with Eq. (9.6.23) in the limit of $ch \to 0$.

The second term of Eq. (9.6.23) or Eq. (9.6.24) is a correction to Eq. (9.6.19). To compute the second term, Eq. (9.6.19) is first evaluated, and then the second term is computed.

An extension of the exponential method to a set of nonlinear equations is straightforward, and the procedure is essentially the same as for a single equation. That is, Eq. (9.6.19) is independently evaluated for all the equations. Once the predictors for all the variables are obtained, then the second term of Eq. (9.6.23) or Eq. (9.6.24) is evaluated.

Summary of This Section

(a) An ODE becomes stiff if its time constant is short and $f'/f < 0$ (if there is no homogeneous term, the solution approaches zero). If a standard numerical method such as one of the Runge-Kutta or predictor-corrector methods is used, a very small time step is required even when the solution is slowly changing.

(b) To alleviate the difficulty of the stiff ODEs, two methods, including an implicit method and exponential method, are introduced.

PROGRAMS

PROGRAM 9-1 Second-order Runge-Kutta Method

(A) Explanations

This program solves a second-order ordinary differential equation in Example 9.6.

$$M\frac{d^2}{dt^2}y(t) + B\frac{d}{dt}y(t) + ky = 0, \quad y(0) = 1, \quad y'(0) = 0 \tag{A}$$

which represents the motion of a mass attached to one end of a spring-damper system that is hung from the ceiling as shown in Fig. E9.6. In the foregoing equation, M, B, and k may be interpreted as mass, damping constant, and spring constant, respectively. To apply the second-order Runge-Kutta method to the second-order ODE, Eq. (A) is first split into two first-order equations as

$$\begin{aligned} \frac{d}{dt}y(t) &= f(y, z, t), \quad y(0) = 1 \\ \frac{d}{dt}z(t) &= g(y, z, t), \quad z(0) = 0 \end{aligned} \tag{B}$$

where

$$\begin{aligned} f(y, z, t) &= z(t) \\ g(y, z, t) &= -(B/M)z(t) - (k/M)y(t) \end{aligned}$$

The constants and initial conditions are specified within the code. No input is necessary.

(B) List

```
/* CSL/c9-1.c        Second Order Runge-Kutta Scheme
 *                   (Solving the problem II of Example 9.6) */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
/*          time : t
            y,z: y,y'
            kount: number of steps between two lines of printing
            k, m, b: k, M(mass), B(damping coefficient) in Example 9.6 */
main()
{
int kount, n, kstep=0;
float bm, k1, k2, km, l1, l2;
static float time,   k = 100.0, m = 0.5, b = 10.0, z = 0.0;
static float y = 1.0,   h = 0.001;
   printf( "CSL/C9-1   Second Order Runge-Kutta Scheme \n" );
   printf( "        t              y                z\n" );
   printf( "   %12.6f    %12.5e   %12.5e \n", time, y, z );
   km = k/m;
   bm = b/m;
   for( n = 1; n <= 20; n++ ){
      for( kount = 1; kount <= 50; kount++ ){
         kstep=kstep+1; time = h*kstep ;
         k1 = h*z;                l1 = -h*(bm*z + km*y);
         k2 = h*(z + l1);         l2 = -h*(bm*(z + l1) + km*(y + k1));
         y = y + (k1 + k2)/2;     z = z + (l1 + l2)/2;
      }
   printf( "   %12.6f    %12.5e   %12.5e \n", time, y, z );
   }
   exit(0);
}
```

(C) Sample Output

```
CSL/C9-1   Second Order Runge-Kutta Scheme
      t             y             z
      0.000000     1.00000e+00    0.00000e+00
      0.050000     8.23049e-01   -5.81545e+00
      0.100000     5.08312e-01   -6.19085e+00
      0.150000     2.38353e-01   -4.45118e+00
      0.200000     6.67480e-02   -2.46111e+00
      0.250000    -1.66253e-02   -9.82537e-01
      0.300000    -4.22529e-02   -1.40603e-01
      0.350000    -3.88646e-02    2.11764e-01
      0.400000    -2.58300e-02    2.77157e-01
      0.450000    -1.32004e-02    2.17147e-01
      0.500000    -4.55050e-03    1.29208e-01
      0.550000     1.17297e-05    5.76674e-02
      0.600000     1.68646e-03    1.38587e-02
```

(D) Discussions

The physical meaning of y is displacement of the mass and z is the velocity. The initial conditions $y(0) = 1$ and $z(0) = 0$ mean that initially the mass is held at $y = 1$ with zero velocity, and released at $t = 0$. The computed results show that the mass starts moving to the negative direction with increasing negative velocity, but the direction of motion is changed between $t = 0.3$ and 0.35. The motion of the mass is oscillatory but dies away quickly because of a strong damping effect.

If B is set to zero, the damping effect is removed and the motion theoretically becomes harmonic. If the program is run with $B = 0$, the computed results will show a more sustained oscillation. However, numerical error tends to damp the oscillation. The reader is encouraged to run the program with varying time steps. The accuracy of calculating a harmonic oscillation is significantly improved by a higher-order numerical method such as the fourth-order Runge-Kutta method.

PROGRAM 9-2 Fourth-order Runge-Kutta Scheme

(A) Explanations

This program integrates a first-order ordinary differential equation,

$$y' = f(y, t), \quad y(0) = y_0$$

When executed, the user is asked interrogatively for the number of steps, nstep_pr, in each printing time interval denoted by t_pr. The step interval h for the Runge-Kutta scheme is then set to h = t_pr/nsteps_pr. The user is also asked for the maximum t to which the solution must proceed. The initial condition for y is set in the program.

(B) List

```
/* CSL/c9-2.c      Fourth-Order Runge-Kutta Scheme
 *                      (See Example 9.8) */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define TRUE 1
/*            time : t
              h : time step interval
              k1, k2, k3, and k4 : k1, k2, k3 and k4
              t_new and t_old : t for the new point and old point, resp.
              y : solution
              nstep_pr : number of steps in one printing t_pr
              t_pr : time interval of printing output              */
main()
{
int nstep_pr, j, j_, k;
float _f0, h, hh, k1, k2, k3, k4, t_old, t_limit,
                                          t_mid, t_new, t_pr, y, ya, yn;
double fun();
   printf( "\n CSL/C9-2  Fourth-Order Runge-Kutta Scheme \n" );
   while(TRUE){
      printf( "Interval of t for printing ?\n" );    scanf( "%f", &t_pr );
      printf( "Number of steps in one printing interval?\n" );
      scanf( "%d", &nstep_pr );
      printf( "Maximum t?\n" );          scanf( "%f", &t_limit );
      y = 0;    /*  Setting the initial value of the solution  */
      h = t_pr/nstep_pr;
      printf( "h=%g \n", h );
      t_new = 0;                        /* Time is initialized. */
      hh = h/2;
      printf( "-------------------------------------\n" );
      printf( "       t                y\n" );
      printf( "-------------------------------------\n" );
      printf( "  %12.5f      %15.6e \n", t_new, y );
      do{
         for( j = 1; j <= nstep_pr; j++ ){
            t_old = t_new;
            t_new = t_new + h;
            yn = y;
            t_mid = t_old + hh;
            yn = y;          k1 = h*fun( yn, t_old );
            ya = yn + k1/2; k2 = h*fun( ya, t_mid );
            ya = yn + k2/2; k3 = h*fun( ya, t_mid );
            ya = yn + k3  ; k4 = h*fun( ya, t_new );
            y = yn + (k1 + k2*2 + k3*2 + k4)/6;
         }
         printf( "  %12.5f      %15.6e \n", t_new, y );
      } while( t_new <= t_limit );
      printf( "-------------------------------------\n" );
      printf( " Maximum t limit exceeded \n" );
      printf( "Type 1 to continue, or 0 to stop.\n" );
      scanf( "%d", &k );
      if( k != 1 )  exit(0);

}
```

```
double fun(y, t)
float y, t;
{
float fun_v;
   fun_v = t*y  +1;      /* Definition of f(y,t)    */
   return( fun_v );
}
```

(C) Sample Output

```
CSL/C9-2  Fourth-Order Runge-Kutta Scheme

Interval of t for printing ?
1
Number of steps in one printing interval?
10
Maximum t?
5
h=0.1
--------------------------------------
       t                 y
--------------------------------------
       0.00000          0.000000e+00
       1.00000          1.410686e+00
       2.00000          8.839369e+00
       3.00000          1.125059e+02
       4.00000          3.734231e+03
       5.00000          3.357972e+05
       6.00000          8.194355e+07
--------------------------------------
 Maximum t limit exceeded
```

(D) Discussions

The equation solved in the foregoing output is

$$y'(t) = -\frac{y(t)}{t^2 + y^2(t)}$$

with the initial condition $y(0) = 1$. The interactive input specifies that the printing interval is 1, which is divided into ten time steps for numerical integration.

PROGRAM 9-3 Fourth-order Runge-Kutta Method for a Set of ODEs

(A) Explanations

This program is designed to solve by the fourth-order Runge-Kutta method a set of an arbitrary number of first-order ordinary differential equations written as

$$\frac{dy_i(t)}{dt} = f_i(y_j, j = 1, \ldots, I; t), \quad i = 1, 2, \ldots, I$$

where I is the number of equations coupled, and the initial conditions are

$$y_i(0) = y_{i,0}, \quad i = 1, \ldots, I$$

The functions on the right side of the differential equations are hard-coded in func(). Initial conditions are also specified within the program.

The printing time interval for the solution, the number of steps for integration per one printing time interval, and the maximum time for integration are given interactively by input.

(B) List

```
/* CSL/c9-3.c          Fourth-Order Runge-Kutta Scheme
 *                     for a Set of Equations
 *                      (See Example 9.9) */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define TRUE 1
/*             k1[], k2[], k3[], k4[] : k1, k2, k3 and k4 for j-th eq.
               t_new : t for the new point
               t_old : t for the previous point
               t_mid : t for the midpoint
               y[j] : solution for the j-th equation                */
main()
{
int i, No_of_eqs, j, k, n, ns;
float k1[11], k2[11], k3[11], k4[11],
     h, hh, pi, ta, t_old, t_limit, t_mid, t_new,
     y[11], ya[11], yn[11];
void f();
   printf( "\n CSL/C9-3      Fourth-Order Runge-Kutta Scheme \n" );
   printf( "                for a Set of Equations \n" );
   while (TRUE){
      No_of_eqs = 2;          /* Number of equations */
      y[1] = 1;       /* Initial condition for y1 at t=0. */
      y[2] = 0;       /* Initial condition for y2 at y=0. */
      printf( "Interval of t for printing ?  " );    scanf( "%f", &pi );
      printf( "Number of steps in one print interval ? " );
      scanf( "%d", &ns );
      printf("Maximum t to stop calculations ?");scanf("%f", &t_limit);
      h = pi/ns;
      printf( " h= %g \n", h );
      t_new = 0;
      hh = h/2;
      printf( "       t         y(1),          y(2), .....\n" );
      printf(  "   %10.4f ", t_new );
      for( i = 1; i <= No_of_eqs; i++ ) printf(  "%12.5e ", y[i] );
      printf(  "\n" );
      do{
         for( n = 1; n <= ns; n++ ){
            t_old = t_new; /* Old time */
            t_new = t_new + h; /* New time */
            t_mid = t_old + hh; /* Midpoint time */
            for( i = 1; i <= No_of_eqs; i++ )   ya[i] = y[i];
              f( k1,  ya, &t_old, &h );
            for( i = 1; i <= No_of_eqs; i++ )   ya[i] = y[i] + k1[i]/2;
              f( k2, ya, &t_mid, &h );
```

```
            for( i = 1; i <= No_of_eqs; i++ )   ya[i] = y[i] + k2[i]/2;
              f( k3, ya, &t_mid, &h );
            for( i = 1; i <= No_of_eqs; i++ )   ya[i] = y[i] + k3[i];
              f( k4, ya, &t_new, &h );
            for( i = 1; i <= No_of_eqs; i++ )
              y[i] = y[i] + (k1[i] + k2[i]*2 + k3[i]*2 + k4[i])/6;
          }
          printf(  "   %10.4f ", t_new );
          for( i = 1; i <= No_of_eqs; i++ )  printf(  "%12.5e ", y[i] );
          printf(  "\n" );
        }while ( t_new < t_limit );
        printf( "Type 1 to continue, or 0 to stop. \n" );
        scanf( "%d", &k );    if( k != 1 )  exit(0);
    }
}

void f(k, y, t, h)
float k[], y[], *t, *h;
{
    k[1] =  y[2]**h;
    k[2] = -y[1]**h;
    /* More equations come here if the number of equations are greater.*/
    return;
}
```

(C) Sample Output

```
CSL/C9-3        Fourth-Order Runge-Kutta Scheme
                  for a Set of Equations
Interval of t for printing ?  0.5
Number of steps in one print interval ?  2
Maximum t to stop calculations ?  5.1
 h= 0.25
        t          y(1),           y(2), .....
   0.0000  1.00000e+00  0.00000e+00
   0.5000  8.77587e-01 -4.79410e-01
   1.0000  5.40326e-01 -8.41448e-01
   1.5000  7.07842e-02 -9.97482e-01
   2.0000 -4.16083e-01 -9.09312e-01
   2.5000 -8.01083e-01 -5.98526e-01
   3.0000 -9.89959e-01 -1.41212e-01
   3.5000 -9.36474e-01  3.50671e-01
   4.0000 -6.53723e-01  7.56699e-01
   4.5000 -2.10930e-01  9.77470e-01
   5.0000  2.83500e-01  9.58937e-01
```

(D) Discussions

A set of two ODEs given by

$$\frac{d}{dt}y_1(t) = y_2(t), \quad y_1(0) = 1$$

$$\frac{d}{dt}y_2(t) = -y_1(t), \quad y_2(0) = 0$$

is solved by the fourth-order Runge-Kutta method.

Results are printed out for every increment of $\Delta t = 0.5$ as specified by input. The time step interval for the Runge-Kutta method is half of the printing time interval, namely, $h = 0.25$. The computation is stopped when t exceeds 5. If a smaller time step for integration is desired, a larger number can be given for "Number of steps in one print interval."

The problem solved here may be interpreted as a harmonic oscillation problem because, if y_2 is eliminated, the equation for y_1 becomes

$$\frac{d^2y_1}{dt^2} + y_1 = 0$$

The reader is encouraged to run this program with a finer time interval and plot the solution for a longer period, so a sustaining harmonic oscillation is seen. (See also Example 9.9.)

PROGRAM 9–4 Third-order Predictor-Corrector Method

(A) Explanations

This program solves a first-order ordinary differential equation,

$$\frac{dy}{dt} = f(y, t), \quad y(0) = y_0$$

using the third-order predictor-corrector method.

Since the predictor-corrector method cannot be self-started, the fourth-order Runge-Kutta method is used to start up the predictor-corrector method.

For the first two time steps, the fourth-order Runge-Kutta scheme is used. After then, the program ignores the Runge-Kutta method and starts using the predictor-corrector scheme. This program can be upgraded very easily to the fourth-order predictor-corrector method.

The user defines $f(y, t)$ and the initial condition in the program in a similar manner as PROGRAMS 9–2 and 9–3. Some input data are given interactively.

(B) List

```
/*CSL/c9-4.c   Third-Order Predictor-Corrector Method */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define TRUE 1
/*
              h : step interval
              h_half : 1/2 of h
              f_old : f at the previous step
              f_2step_old : f of 2 steps ago
```

```
                f_3step-old : f of 3 steps ago
                time_pr : time interval for printing output
                nstep_pr : number of steps in one printing interval
                y : solution
                yp : predictor                                    */
main()
{
int nstep_pr, j, n;
float _f0, f, fa, f_old, f_2step_old, f_3step_old, g, h, h_half,
     k1, k2, k3, k4, t_old, time_pr, t_mid,
     tmax, t_new, ta, y, ya, yb, yp;
void func();
   printf( "\n\nCSL/C9-4  Third-Order Predictor-Corrector Method \n" );
   printf( "Time interval for printing solution ?\n" );
   scanf( "%f", &time_pr );
   printf( "Number of steps in one print interval ?\n" );
   scanf( "%d", &nstep_pr );
   printf( "Limit of t ?\n" );
   scanf( "%f", &tmax );
   h = time_pr/nstep_pr;                                  /*  Time interval */
   h_half = h/2;
   printf( "Step size=%g \n", h );
   y = 0;                  /*  Initial condition for the solution */
   yb = y;
   ta =0;
   t_new = 0;
   f_2step_old = 0;
   f_old = 0;
   g = h/12;
   printf( " Solution:\n" );
   printf( "       t                  y\n" );
   func( &fa, ta, y);
   n = 0;                /* Initialize time step counter */
   while(TRUE){
      for( j = 1; j <= nstep_pr; j++ ){
         /*  nstep_pr time steps are advanced in this printing cycle */
         n = n + 1;  /*  Counting time steps */
         f_3step_old = f_2step_old;  f_2step_old = f_old;   f_old = fa;
         t_old = t_new;  t_new = t_old + h;  t_mid = t_old + h_half;
         if( n <= 2 ){                    /*  4-th order Runge-Kutta */
            func( &f, t_old, y );
            k1 = h*f; ya=y+k1/2;      func( &f, t_mid, ya );
            k2 = h*f; ya=y+k2/2;      func( &f, t_mid, ya );
            k3 = h*f; ya=y+k3;        func( &f, t_new, ya );
            k4 = h*f;
            y = y + (k1 + k2*2. + k3*2. + k4)/6.;
         }
         else{
            yp = y + g*(23*f_old - 16*f_2step_old + 5*f_3step_old);
            func( &fa, t_new, yp );
            y = y + g*(5*fa + 8*f_old - f_2step_old);
         }
         func( &fa, t_new, y );
      }
      printf("%10.4f          %12.5e \n", t_new, y );
      if( t_new > tmax )   exit(0);
   }
}
```



```
void  func(f, t, y)
float *f, t, y;
{
   *f = t*y + 1;      /* Defines the differential equation */
   return;
}
```

(C) Sample Output

```
CSL/C9-4  Third-Order Predictor-Corrector Method

Time interval for printing solution ?
1
Number of steps in one print interval ?
10
Limit of t ?
5.0
Step size=0.1
 Solution:
        t                y
   1.0000          1.41091e+00
   2.0000          8.84414e+00
   3.0000          1.12644e+02
   4.0000          3.74007e+03
   5.0000          3.35593e+05
   6.0000          8.11660e+07
```

(D) Discussions

The problem solved is

$$\frac{dy}{dt} = ty + 1, \quad y(0) = 1$$

The solution approaches infinity because the right side is positive and increases with t.

PROBLEMS

(9.1) Solve the following problems in $0 \leqslant t \leqslant 5$ using the forward Euler method with $h = 0.5$ by hand calculation. Repeat the same with $h = 0.01$ by using a computer (write a short program yourself). Evaluate the errors by comparing to the exact solutions that follow:

(a) $y' + ty = 1$, $y(0) = 1$

(b) $y' + 3y = e^{-t}$, $y(0) = 1$

(c) $y' = (t^2 - y)$, $y(0) = 0.5$

(d) $y' + y|y| = 0, \quad y(0) = 1$

(e) $y' + |y|^{1/2} = \sin(t), \quad y(0) = 1$

Exact Solution

	Case *a*	*b*[a]	*c*	*d*	*e*
t	*y*	*y*	*y*	*y*	*y*
0	1.0000	1.0000	0.5000	1.0000	1.0000
1	1.3313	0.2088	0.4482	0.5000	0.6147
2	0.7753	0.06890	1.7969	0.3333	0.7458
3	0.4043	2.4955*E*–2	4.9253	0.2500	0.4993
4	0.2707	9.1610*E*–3	9.9725	0.2000	−0.2714
5	0.2092	3.3692*E*–3	16.980	0.1666	−2.2495

[a]*Hint:* Solution of (b) may oscillate with $h = 0.5$, but you are encouraged to try anyway.

(9.2) Solve

$$y''(t) - 0.05y'(t) + 0.15y(t) = 0, \quad y'(0) = 0, \quad y(0) = 1$$

and find the values of $y(1)$ and $y(2)$ using the forward Euler method with $h = 0.5$.

(9.3) Solve the following problems in $0 \leqslant t \leqslant 5$ using the forward Euler method with $h = 0.1$ and $h = 0.01$ (write your own program).

(a) $y'' + 8y = 0, \; y(0) = 1, \; y'(0) = 0$

(b) $y'' - 0.01(y')^2 + 2y = \sin(t), \; y(0) = 0, \; y'(0) = 1$

(c) $y'' + 2ty' + ty = 0, \; y(0) = 1, \; y'(0) = 0$

(d) $(e^t + y)y'' = t, \; y(0) = 1, \; y'(0) = 0$

Evaluate the errors using the following exact solutions:

Exact Solution

	Case *a*	*b*	*c*	*d*
t	*y*	*y*	*y*	*y*
0	1.0	0.0000	1.0000	1.0000
1	−0.9514	0.8450	0.8773	1.0629
2	0.8102	0.9135	0.5372	1.3653
3	−0.5902	0.1412	0.3042	1.8926
4	0.3128	−0.7540	0.1763	2.5589
5	−0.0050	−0.9589	0.1035	3.2978

(9.4) Solve the following equations for $0 < t < 5$ by the modified Euler method:

$$4y' = -3y + 7z + 2t, \quad y(0) = 1$$
$$7z' = -2y + 8z, \quad z(0) = 0$$

Use both $h = 0.01$ and 0.001.

(9.5) A conical tank contains water up to 0.5 m high from the bottom. The tank has a hole of 0.02 m radius at the bottom. The radius of the tank at y is given by $r = 0.25y$, where r is the radius and y is the height measured from the bottom. The velocity of the water that

drains through the hole is given by $v^2 = 2gy$ where $g = 9.8$ m/sec^2. Using the forward Euler method (use $h = 0.001$ sec), find out how many minutes it will take until the tank becomes empty.

(9.6) A circuit shown in Figure P9.6 has self-inductance of $L = 100$ henry, a resistance of $R = 2$ ohm and a DC voltage source of 10 volt. If the switch is closed at $t = 0$, the current, $I(t)$, changes in accordance with

$$L\frac{d}{dt}I(t) + RI(t) = E, \quad I(0) = 0$$

(a) Find the current I at $t = 1, 2, 3, 4$, and 5 sec by using the forward Euler method with $h = 0.01$.

(b) Evaluate the error by comparing the numerical solution to the analytical solution given by $I(t) = (E/R)(1 - \exp^{-Rt/L})$.

(c) Investigate the effect of h by repeating the foregoing calculations with $h = 0.1$.

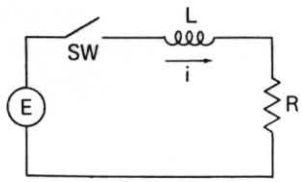


Figure P9.6 Electric circuit

(9.7) A U-tube of 0.05 m radius is initially filled with water but separated by a partition so that the water level of the left vertical part is 0.2 m higher than the water level of the right vertical part. At $t = 0$ the partition is suddenly removed. The water level of left vertical portion, y_A, measured from the midplane between two surfaces, satisfies

$$Ly_A'' = -2gy_A$$

where L is the total length of water in the U-tube, which is assumed to be 1 m, and $g = 9.8$ m/sec^2. Ignoring the friction in the tube, calculate the water level by the forward Euler method for $0 < t < 10$ sec, and find when y_A reaches minimums and maximums. Use $h = 0.001$.

(9.8) Repeat the previous problem assuming that there is friction in the pipe so that the equation of motion is given by

$$Ly_A'' = -2gy_A - \beta y_A'$$

where $\beta = 0.8$ m/sec. Use $h = 0.001$.

(9.9) The number density (number of atoms per cm^3) of iodine-135 (radioisotope) satisfies

$$\frac{d}{dt}N_i(t) = -\lambda_i N_i(t)$$

where $N(t)$ is the number density of iodine-135 and λ_i is its decay constant equal to 0.1044 hr^{-1}. If $N_i(0) = 10^5$ atoms/cm^3 at $t = 0$, compute $N_i(t)$ at $t = 1$ hr by the modified Euler method. Set h to 0.05 hr.

(9.10) The decay product of iodine-135 (considered in the previous problem) is xenon-135; it is also radioactive. Its decay constant is $\lambda_x = 0.0753\ \text{hr}^{-1}$. The number density of xenon satisfies

$$\frac{d}{dt} N_x(t) = -\lambda_x N_x(t) + \lambda_i N_i(t)$$

where N_x is the number density of xenon and N_i is the number density of iodine defined in the previous problem. Assuming that $N_x(0) = 0$, develop a program to compute N_i and N_x together based on the modified Euler method. (Because the differential equations are linear, use closed form solutions for each time step.) Print out the solution for every 5 hours until 50 hrs is reached. Use $h = 0.1$ hr.

(9.11) Calculate $y(1)$ by solving the following equation by the second-order Runge-Kutta method with $h = 0.5$:

$$y' = -\frac{y}{t + y^2}, \quad y(0) = 1$$

(9.12) Calculate $y(2)$ for the following equation using the second-order Runge-Kutta method with $h = 1$:

$$y'' + 0.2y' + 0.003y \sin(t) = 0, \quad y(0) = 0,\ y'(0) = 1$$

(9.13) Find the value of $y(1)$ by solving

$$y'' - 0.05y' + 0.15y = 0, \quad y(0) = 1, \quad y'(0) = 0$$

Use the second-order Runge-Kutta method with $h = 0.5$.

(9.14) Solve the following differential equation:

$$2y'' + (y')^2 + y = 0, \quad y(0) = 0,\ y'(0) = 1$$

by the second-order Runge-Kutta method with $h = 0.5$ and evaluate $y(1)$ and $y'(1)$.

(9.15) An initial value problem of an ordinary differential equation is given by

$$y''' = -y, \quad y(0) = 1$$
$$y'(0) = y''(0) = 0$$

Using the second-order Runge-Kutta method with $h = 0.2$, calculate $y(0.4)$ and $y(1)$.

(9.16) **(a)** A 50-gal tank of water contains salt at a concentration of 10 oz/gal. To dilute the salt content, fresh water is supplied at the rate of 2 gal/min. If the tank is well mixed, and the same amount of water leaves the tank every minute, the salt content satisfies

$$y_1'(t) = -\frac{2}{50} y_1, \quad y_1(0) = 10$$

where $y_1(t)$ is the salt concentration in oz/gal, and t is time in minutes. By the second-order Runge-Kutta method with $h = 1$ min, find out how long it takes until the salt concentration reaches 1/10 of its initial value.

(b) The water that leaves the tank enters another tank of 20 gal, into which fresh water is also poured at the rate of 3 gal/min and well mixed. The salt concentration in this tank satisfies

$$y_2'(t) = -\frac{3}{20} y_2(t) + \frac{2}{20} y_1(t), \quad y_2(0) = 0$$

where $y_1(t)$ is the salt concentration of the 50 gal tank of the previous problem. By using the second-order Runge-Kutta method, find when the salt concentration of the 20 gal tank reaches its maximum. Assume that the second tank has fresh water at $t = 0$.

(9.17) Repeat Problem 9.12 by the third-order Runge-Kutta method.

(9.18) A bullet is shot into the air at a 45° angle from the ground at $u = v = 150$ m/sec, where u and v are horizontal and vertical velocities, respectively. The equations of motion are given by

$$\begin{aligned} u' &= -cVu, \qquad u(0) = 150\ \text{m/sec} \\ v' &= -g - cVv, \quad v(0) = 150\ \text{m/sec} \end{aligned} \tag{A}$$

where u and v are functions of time, $u = u(t)$, and $v = v(t)$, and

$$V = \sqrt{u^2 + v^2}$$
$$c = 0.005 \quad \text{(coefficient of drag)}$$
$$g = 9.8\ \text{m/sec}^2 \quad \text{(gravity)}$$

The equations of motion may be solved by one of the Runge-Kutta methods. The trajectory of the bullet may be calculated by integrating,

$$x' = u \quad \text{and} \quad y' = v$$

or

$$\begin{aligned} x &= \int_0^t u(t')\,dt' \\ y &= \int_0^t v(t')\,dt' \end{aligned} \tag{B}$$

A program based on the second-order Runge-Kutta method to solve Eq. (A) and evaluate Eq. (B) follows:

```
/* CSL/bullet2.c     Bullet shot to the sky (2nd order R-K) */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
main()
{
int n;
float K1, K2, L1, L2, u, v, VEL1, VEL2, VEL3;
static float ub=150.0, vb=150.0, h=0.1, c=0.005, t=0.0, x=0.0, y=0.0;
   printf( "\n\n CSL/BLLET2     Bullet shot to the sky\n " );
   printf( " t            u              v              x              y");
```

```
    printf(  "\n%12.5e  %12.5e  %12.5e  %12.5e  %12.5e", t, ub, vb, x, y );
    for( n = 1; n <= 200; n++ ){
       t = t + h;
       VEL1 = sqrt( pow(ub, 2) + pow(vb, 2) );
       K1 = -c*VEL1*ub*h;
       L1 = (-9.8 - c*VEL1*vb)*h;
       VEL2 = sqrt( pow(ub + K1/2, 2) + pow(vb + L1/2, 2) );
       K2 = -c*VEL2*(ub + K1)*h;
       L2 = (-9.8 - c*VEL2*(vb + L1))*h;
       u = ub + (K1 + K2)/2;
       v = vb + (L1 + L2)/2;
       x = x + 0.5*(u + ub)*h;
       y = y + 0.5*(v + vb)*h;
       ub = u;
       vb = v;
       printf("\n%12.5e  %12.5e  %12.5e  %12.5e  %12.5e",t,ub,vb,x,y);
       if( y < 0 ) break ;
    }
    printf( "\n\n" );
}
```

(a) Run the program and plot the trajectory of the bullet.

(b) Rewrite the program using the third-order Runge-Kutta method.

(9.19) Calculate $y(1)$ by solving the following equation using the fourth-order Runge-Kutta method with $h = 1$:

$$y' = -\frac{y}{t + y^2}, \quad y(0) = 1 \text{ for } t = 0$$

(9.20) The solution of $y' = -1(1 + y^2)$ by the second-order Runge-Kutta method is shown for two different h values.

	h = 0.1	h = 0.2
t	y	y
0.0	1.0000000	1.0000000
0.1	0.9487188	
0.2	0.894672	0.8947514
0.3	0.8375606	
0.4	0.7770516	0.7772616
0.5	0.7127807	
0.6	0.6443626	0.6447898
0.7	0.5714135	
0.8	0.4935937	0.4943817
0.9	0.4106803	
1.0	0.3226759	0.3240404

(a) Estimate the local error with $h = 0.1$.

(b) Estimate a more accurate value of $y(1)$.

(9.21) By hand calculations, find the solution of

$$y'(t) = -\frac{1}{1 + y^2}, \quad y(0) = 1$$

for $t = 1$ and $t = 2$ using the fourth-order Runge-Kutta method with $h = 0.5$ and $h = 1$.

(9.22) Repeat Problem 9.1 with the fourth-order Runge-Kutta method with $h = 0.1$.

(9.23) For the equation given by

$$y' = 3y + \exp(1 - t), \quad y(0) = 1$$

find an optimal time step for the second-order Runge-Kutta method to satisfy the condition for the local error, $E(h) < 0.0001$. (Run the second-order Runge-Kutta method for one interval with a value of h, and rerun it for two intervals with $h/2$.)

(9.24) Repeat Problem (9.23) for the fourth-order Runge-Kutta method.

(9.25) By repeating the analysis of Eqs. (9.3.23) through (9.3.27), derive the equation corresponding to Eq. (9.3.27) for the third-order Runge-Kutta method.

(9.26) If the third-order Runge-Kutta method is applied to $y' = -\alpha y$, find in what range of h the method is unstable.

(9.27) The initial temperature of the metal piece described in Example 9.12 is now assumed to be 25° C. The metal piece is internally heated electrically at the rate of $q = 3000$ W. The equation for the temperature is written as

$$\frac{dT}{dt} = \frac{1}{\rho c v}[q - \epsilon\sigma A(T^4 - 298^4) - h_c A(T - 298)], \quad T(0) = 298$$

Calculate the temperature until $t = 10$ min, and print it out for every 0.5 min by the fourth-order Runge-Kutta method with $h = 0.1$ min. (Use the constants given in Example 9.12.)

(9.28) The motion of the mass system shown in Figure P9.28 is given by

$$y'' + 2\zeta\omega y' + \omega^2 y = F(t)/M$$

where

$\omega = (k/M)^{1/2}$ (undamped natural frequency, s^{-1})
$\zeta = c/(2M\omega) = 0.5$ (damping factor)
$k = 3.2$ (spring constant, kg/s^2)
$M = 5$ (mass, kg)
$F(t) = 1$ (kg force) for $0 < t < 1$, or 0 for $t > 1$ (1 kg force = 9.8 Newton)

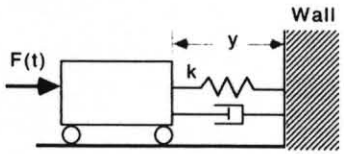


Figure P9.28 Spring-mass system

If $F(t)$ is a step function of magnitude $F_0 = 1$ kg force and time duration 1 sec, determine the motion of the mass for $0 < t < 10$ sec using the fourth-order Runge-Kutta method.

(9.29) Determine the response and dynamic load of the damping mass system of the previous problem subject to a triangular force pulse

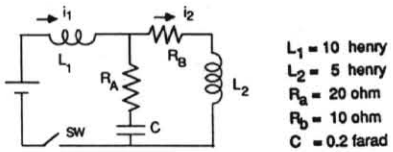
$$\begin{aligned} F(t) &= 2F_0 t, && 0 \leqslant t \leqslant 1 \text{ sec} \\ &= 2F_0(1-t), && 1 \leqslant t \leqslant 2 \text{ sec} \\ &= 0, && t > 2 \text{ sec} \end{aligned}$$

where $F_0 = 1$ kg (force). Use the fourth-order Runge-Kutta method.

(9.30) The differential equations for the circuit shown in Figure P9.30 is

$$L_1 \frac{d}{dt} i_1 + R_A(i_1 - i_2) + \frac{1}{C}\int_0^t (i_1(t') - i_2(t'))\,dt' = e(t)$$

$$-\frac{1}{C}\int_0^t (i_1(t') - i_2(t'))\,dt' - R_A(i_1 - i_2) + R_B i_2 + L_2 \frac{d}{dt} i_2 = 0$$



$L_1 = 10$ henry
$L_2 = 5$ henry
$R_a = 20$ ohm
$R_b = 10$ ohm
$C = 0.2$ farad

Figure P9.30 Electric circuit

The initial conditions are

$$i_1(0) = i_2(0) = 0,$$

and $e(t) = 1$. Using the fourth-order Runge-Kutta method with $h = 0.1$ sec, determine i_1 and i_2 for $0 < t < 10$ sec.

(9.31) The problem in Example 9.15 has a geometrical symmetry about $x = 0.5$. Therefore, the problem may be equivalently restated as

$$Ak \frac{d^2}{dx^2} T + P\sigma(T^4 - 273^4) = Q, \quad 0.5 < x < 1$$

$$T'(0.5) = 0$$

$$T(1) = 0$$

Using the fourth-order Runge-Kutta method, solve the foregoing equations by the shooting method. (*Hint:* change $T(0.5)$ by trial and error until $T(1) = 0$ is satisfied.)

(9.32) Repeat (a), (b), and (c) of Problem 9.1 with the Adams third-order predictor-corrector method using $h = 0.5$.

(9.33) Repeat (a), (b), and (c) of Problem 9.1 with the Adams fourth-order predictor-corrector method using $h = 1.0$.

(9.34) Explicitly write down the Adams-Bashforth predictors of orders 2, 3, 4, and 5 respectively.

(9.35) Explicitly write down the Adams-Moulton correctors of orders 2, 3, 4, and 5 respectively.

(9.36) Solve the problem in Example 9.11 by the Adams fourth-order predictor-corrector scheme. (*Hint:* Modify PROGRAM 9–4 so that the first three steps are calculated by the fourth-order Runge-Kutta method and the remainder is then calculated by the fourth-order predictor-corrector method.)

(9.37) The Euler predictor-corrector method is written as

$$\bar{y}_{n+1} = y_n + hy'_n$$

$$y_{n+1} = y_n + \frac{1}{2}h(\bar{y}'_{n+1} + y'_n)$$

If the Euler predictor-corrector method is applied to $y'(t) = \alpha y(t)$ where $\alpha < 0$, instability occurs if $\alpha h < -2$. Prove this.

(9.38) The Milne predictor-corrector method is given by

$$\bar{y}_{n+1} = y_{n-1} + \frac{4}{3}h(2y'_n - y'_{n-1} + 2y'_{n-2}) \quad \text{predictor}$$

$$y_{n+1} = y_{n-1} + \frac{1}{3}h(\bar{y}'_{n+1} + 4y'_n + y'_{n-1}) \quad \text{corrector}$$

Show that the Milne method becomes unstable for $y' = \alpha y$ if $\alpha < 0$ and

$$-\infty < \alpha h < -0.8 \quad \text{or} \quad -0.3 < \alpha h < 0$$

(*Hint:* Calculate the roots of the characteristic equation as illustrated in Section 9.4.4.)

REFERENCES

Bui, T. D., A. K. Oppenheim, and D. T. Pratt, "Recent advances in methods for numerical solution of O.D.E. initial value problems," *J. Comp. Math.*, Vol. 11, p. 283–296, 1984.

Constantinides, A., *Applied Numerical Methods with Personal Computers*, McGraw-Hill, 1987.

Creese, T. M., and R. M. Haralick, *Differential Equations for Engineers*, McGraw-Hill, 1978.

Ferziger, J. H., *Numerical Methods for Engineering Application*, Wiley-Interscience, 1981.

Fox, L., and D. F. Mayers, *Computing Methods for Scientists and Engineers*, Oxford, University Press 1968.

Furgason, D. R., and K. F. Hansen, "Solution of the space dependent reactor kinetics equations in three dimensions," *Nucl. Sci. Eng.*, Vol. 51, p. 189–205, 1973.

Gear, C. W., *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, 1971.

Habib, I. S., *Engineering Analysis Methods*, Lexington Books, 1975.

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

Hetric, D. L., *Dynamics of Nuclear Reactors*, University of Chicago Press, 1971.

Kubicek, M., and V. Hlavacek, *Numerical Solution of Nonlinear Boundary Value Problems with Applications*, Prentice-Hall, 1983.

Kuo, K. K., *Principles of Combustion*, Wiley-Interscience, 1986.

Lapidus, L., and J. H. Seinfeld, *Numerical Solution of Ordinary Differential Equations*, Academic Press, 1971.

NAG Fortran Library (1987), Algorithm Group Inc., 1101 31st, Suite 100, Downers Grove, IL 60515–1263.

Oran, E. S., and J. P. Boris, *Numerical Simulation of Reactive Flow*, Elsevier, 1987.

Rieder, W. G., and H. R. Busby, *Introductory Engineering Modeling*, Wiley, 1986.

Shampine, L. F. and C. W. Gear, "A User's View of Solving Stiff Ordinary Differential Equations," *SIAM Review*, Vol. 21, 1979.