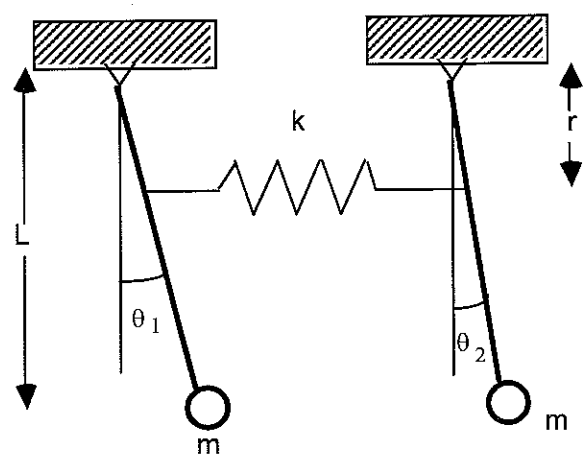
LAM **ENGINEERING 12: PHYSICAL SYSTEMS ANALYSIS** **Spring 2005**

LABORATORY 4: NUMERICAL ANALYSIS

In this laboratory, you will learn two numerical methods to approximate the solution to differential equations. Such methods are especially useful when the number of equations needed to model the system becomes high, or when closed form solutions cannot be found. Generally speaking, in methods of this type, discrete difference equations are found from continuous differential equations that model the behavior of a system. It is important to realize that these numerical methods are approximate. Any results obtained from these methods must be checked to verify that they actually are reasonable results, and error analysis is of particular interest.

Please look over the attached handouts from *Applied Numerical Methods*, by Carnahan, Luther, and Wilkes (Wiley, 1969). In the first of these (pp. 341-352), an introduction to using numerical method to solve differential equations is provided, and that is followed by a description of Euler's method. The second section (pp. 361-380) provides a description of Runge Kutta methods.

In this laboratory, you will use both Euler's Method and a fourth order Runge Kutta Method to model the coupled pendulum system that we studied in the previous lab. You can then compare the results from these analyses with the exact solutions you obtained analytically. You may write a program in a computer language of your choice (my recommendation), or you can use a spreadsheet or other package to perform this analysis.



CHAPTER 6

The Approximation of the Solution of Ordinary Differential Equations

6.1 Introduction

The behavior of many physical processes, particularly those in systems undergoing time-dependent changes (transients), can be described by ordinary differential equations. Thus, methods of solution for these equations are of great importance to engineers and scientists. Although many important differential equations can be solved by well-known analytical techniques, a greater number of physically significant differential equations cannot be so solved. Fortunately, the solutions of these equations can usually be generated numerically. This chapter will describe the more important of these numerical procedures.

***N*th Order Ordinary Differential Equations.** Consider the solution of nth-order ordinary differential equations of the form

$$F\left(x,y,\frac{dy}{dx},\frac{d^2y}{dx^2},\frac{d^3y}{dx^3},\ldots,\frac{d^ny}{dx^n}\right)=0 \qquad (6.1)$$

An equation of type (6.1) is termed *nth-order* because the highest derivative is of order n, and *ordinary* because only total derivatives appear (no partial derivatives are present, or alternatively, there is only one independent variable, x). A function $y(x)$ that satisfies this equation, implying that it is at least n times differentiable, is said to be a *solution* of the equation. To obtain a *unique solution* [in general, there are many functions $y(x)$ that satisfy (6.1)], it is necessary to supply some additional information, namely, values of $y(x)$ and/or of its derivatives at some specific values of x. For an nth-order equation, n such conditions are normally sufficient to determine a unique solution $y(x)$. If all n conditions are specified at the same value of x (x_0, for example), then the problem is termed an *initial-value problem*. When more than one value of x is involved, the problem is termed a *boundary-value problem*.

An nth-order ordinary differential equation may be written as a system of n first-order equations by defining $n-1$ new variables. For example, consider the second-order equation (Bessel's equation),

$$x^2\frac{d^2y}{dx^2}+x\frac{dy}{dx}+(x^2-p^2)y=0, \qquad (6.2)$$

where p is a constant. By defining one new variable, $z=dy/dx$, the second-order equation can be rewritten as a pair of first-order equations:

$$\begin{aligned}\frac{dy}{dx}-z&=0\\ x^2\frac{dz}{dx}+xz+(x^2-p^2)y&=0\end{aligned} \qquad (6.3)$$

Since most higher-order equations (or a system of such equations) can be rewritten in similar fashion, the numerical solution of first-order equations only will be described.

6.2 Solution of First-Order Ordinary Differential Equations

A first-order equation is, by definition, of the form

$$F\left(x,y,\frac{dy}{dx}\right)=0,$$

or, alternatively,

$$\frac{dy}{dx}=f(x,y) \qquad (6.4)$$

We desire a solution $y(x)$ which satisfies both (6.4) and one specified initial condition. In general, it is impossible to determine $y(x)$ in functional (analytical) form. Instead, the interval in the independent variable x over which a solution is desired, $[a,b]$, is divided into subintervals or *steps*. The value of the true solution $y(x)$ is approximated at $n+1$ evenly spaced values of x, $(x_0,x_1,\ldots,x_n)$, so that h, the *step size*, is given by

$$h=\frac{b-a}{n},$$

and

$$x_i=x_0+ih, \qquad i=0,1,\ldots,n \qquad (6.5)$$

Thus the solution is given in *tabular* form for $n+1$ *discrete* values of x only (see Fig. 6.1). This table of values contains sampled values of one particular approximation of the solution of the equation.

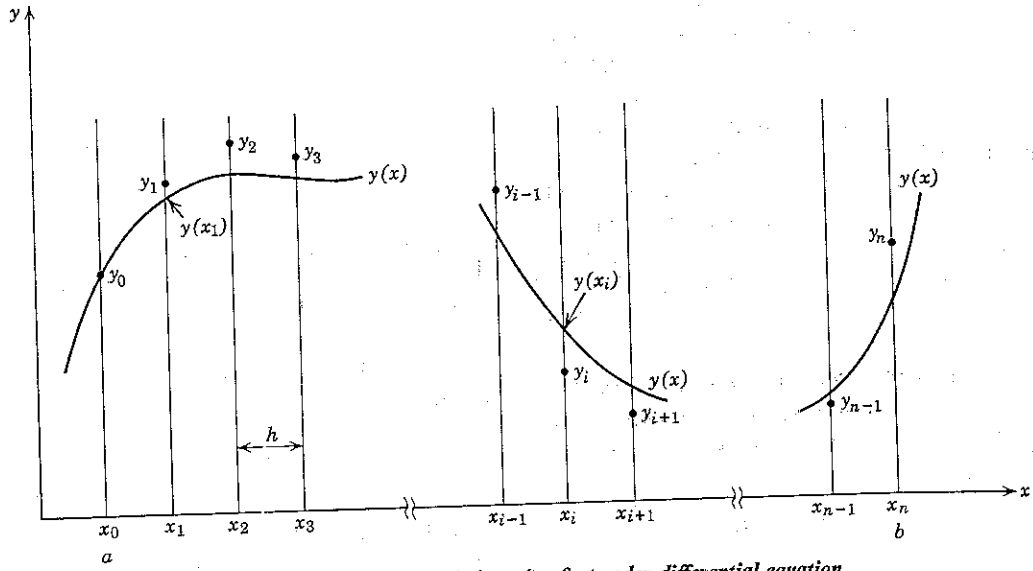


Figure 6.1 Numerical solution of a first-order differential equation

Let the true solution $y(x)$ at the indicated base points be denoted by $y(x_i)$, and the computed approximation of $y(x)$ at these same points be denoted by y_i, so that

$$y_i \doteq y(x_i) \tag{6.6}$$

The true derivative dy/dx at the base points will be approximated by $f(x_i,y_i)$, abbreviated as f_i, so that

$$f_i = f(x_i,y_i) \doteq f(x_i,y(x_i)) \tag{6.7}$$

When the requisite numerical calculations are done exactly, that is, without round-off error (see below), the difference between the computed value y_i and the true value $y(x_i)$ is termed the *discretization* or *truncation error*, ε_i:

$$\varepsilon_i = y_i - y(x_i) \tag{6.8}$$

The discretization error encountered in integrating a differential equation across one step is sometimes called the *local truncation error*. The discretization error is determined solely by the particular numerical solution procedure selected, that is, by the nature of the approximations present in the method; this type of error is independent of computing equipment characteristics.

An inherently different kind of error results from computing-machine design. In practice, computers have only a finite memory and therefore a finite number size (scientific computers usually have a *fixed word-length*, that is, the number of digits retained for any computed result is fixed, usually 7–12 significant digits). Thus any irrational number, or indeed any number with more significant digits than can be retained, that occurs in a sequence of calculations must be approximated by "rounded" values. The error involved is termed *round-off error*, and for any particular numerical method is determined by the computing characteristics of the machine which does the calculations, the order of the machine operations used to implement the algorithm, etc. Some upper bound can usually be found for the discretization error for a particular method; on the other hand, round-off error-generation is extremely complex and unpredictable. However, because of this very unpredictability, numerical analysts have been fairly successful in developing a probabilistic theory of round-off error, on the assumption that local round-off error, that is, the error caused by round-off in integrating a differential equation across one step, is a random variable (see [2] for an excellent description of this work). The only errors which will be examined in any detail here are those related to discretization, that is, those inherent in the numerical procedures.

The common numerical algorithms for solving a first-order ordinary differential equation with initial condition $y(x_0)$ are based on one of two approaches:

1. Direct or indirect use of Taylor's expansion of the *object* or *solution function* $y(x)$.
2. Use of open or closed integration formulas similar to those already developed in Chapter 2.

The various procedures can be classified roughly into two groups, the so-called one-step and multistep methods. *One-step methods* permit calculation of y_{i+1} given the differential equation and information at x_i only, that is, a value y_i. *Multistep methods* require, in addition, values of y_j and/or f_j at other (usually several) values x_j outside the integration interval under consideration, $[x_i,x_{i+1}]$.

One disadvantage of the multistep methods is that more information is required to start the procedure than is normally directly available. Usually an initial condition, say $y(x_0)$, is given; subsequent values, $y(x_1)$, $y(x_2)$, etc., are not known. Some other method (usually a

one-step method) must be used to get started. Another difficulty encountered in the multistep methods is that it is rather difficult to change the step-size h once the calculation is under way. On the other hand, since each new application of a one-step method is equivalent to restarting the procedure, such a change of step-size causes no trouble. The multistep methods require considerably less computation, compared with the one-step methods, to produce results of comparable accuracy. The advantages and disadvantages of each group of methods will become more apparent with the development of the numerical procedures.

Taylor's Expansion Approach. One method of approximating the solution of (6.4) numerically is to express the solution $y(x)$ about some starting point x_0 by using a Taylor's expansion:

$$y(x_0 + h) = y(x_0) + hf(x_0,y(x_0)) + \frac{h^2}{2!}f'(x_0,y(x_0)) + \frac{h^3}{3!}f''(x_0,y(x_0)) + \cdots \tag{6.9}$$

Here, $f'(x,y(x))$ denotes $(d/dx)f(x,y(x))$, $f''(x,y(x))$ denotes $(d^2/dx^2)f(x,y(x))$, etc. If $y(x_0)$ is specified as the initial condition, $f(x_0,y(x_0))$ can be computed directly from the differential equation (6.4)

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

To evaluate the higher-order derivatives of (6.9), we must differentiate $f(x,y)$ by using the chain rule, since f is a function of both x and y:

$$\frac{df}{dx} = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y}\frac{dy}{dx} \tag{6.10}$$

Example. Consider a case for which $f(x,y)$ is a function of x alone:

$$\frac{dy}{dx} = f(x,y) = x^2, \tag{6.11}$$

subject to the initial condition $y(x_0) = y_0$. From (6.10),

$$\begin{aligned} f'(x,y) &= 2x, & f'(x_0,y_0) &= 2x_0; \\ f''(x,y) &= 2, & f''(x_0,y_0) &= 2; \\ f'''(x,y) &= 0, & f'''(x_0,y_0) &= 0; \\ &\vdots & &\vdots \\ f^{(n)}(x,y) &= 0, & f^{(n)}(x_0,y_0) &= 0, \; n > 3. \end{aligned} \tag{6.12}$$

Expansion of $y(x)$ about x_0 by substitution of (6.12) into the Taylor's series of (6.9) yields

$$y(x_0 + h) = y(x_0) + hx_0^2 + h^2x_0 + \frac{h^3}{3}. \tag{6.13}$$

Analytic integration of (6.11) after separation of variables gives

$$\int_{y(x_0)}^{y(x_0+h)} dy = \int_{x_0}^{x_0+h} x^2dx = \frac{x^3}{3}\Big]_{x_0}^{x_0+h},$$

$$y(x_0 + h) = y(x_0) + hx_0^2 + h^2x_0 + \frac{h^3}{3}, \tag{6.14}$$

which is identical to (6.13). There is no truncation error for the algorithm of (6.13) because all high-order derivatives $f^{(n)}(x,y)$, $n > 3$, vanish.

Example. Consider a case for which $f(x,y)$ is a function of y alone:

$$\frac{dy}{dx} = f(x,y) = 2y, \tag{6.15}$$

subject to the initial condition $y(x_0) = y_0$. Differentiating (6.15) by using the chain rule (6.10) yields:

$$\begin{aligned} f'(x,y) &= 4y, & f'(x_0,y_0) &= 4y_0; \\ f''(x,y) &= 8y, & f''(x_0,y_0) &= 8y_0; \\ f'''(x,y) &= 16y, & f'''(x_0,y_0) &= 16y_0; \\ &\vdots & &\vdots \\ f^{(n)}(x,y) &= 2^{n+1}y, & f^{(n)}(x_0,y_0) &= 2^{n+1}y_0. \end{aligned} \tag{6.16}$$

The expansion of $y(x)$ about x_0, using (6.16) in the Taylor's series of (6.9), gives

$$y(x_0 + h) = y(x_0) + 2hy(x_0) + \frac{4h^2y(x_0)}{2!} + \frac{8h^3y(x_0)}{3!} + \cdots = y(x_0)\left[1 + 2h + \frac{(2h)^2}{2!} + \frac{(2h)^3}{3!} + \frac{(2h)^4}{4!} + \cdots\right] \tag{6.17}$$

After separation of variables, direct integration of (6.15) gives

$$\int_{y(x_0)}^{y(x_0+h)} \frac{dy}{y} = 2\int_{x_0}^{x_0+h} dx,$$

and produces the solution

$$y(x_0 + h) = y(x_0)e^{2h}. \tag{6.18}$$

Since $\{1 + 2h + [(2h)^2/2!] + \cdots\}$ is the Taylor's expansion of e^{2h} about $h = 0$, the two solutions (6.17) and (6.18) agree with an accuracy determined by the number of terms retained in the series. If terms up to that including $f^{(n-1)}$ are retained, then the error is given by

$$\epsilon = y(\xi)\frac{(2h)^{n+1}}{(n+1)!}, \qquad \xi \text{ in } (x_0, x_0 + h).$$

Example. Consider a more general, but still simple, example for which $f(x,y)$ is a function of both variables,

$$\frac{dy}{dx} = f(x,y) = x + y, \tag{6.19}$$

subject to the initial condition $y(x_0) = y_0$. Differentiation of (6.19) yields:

$$\begin{aligned} f'(x,y) &= 1 + x + y, & f'(x_0,y_0) &= x_0 + y_0 + 1; \\ f''(x,y) &= 1 + x + y, & f''(x_0,y_0) &= x_0 + y_0 + 1; \\ &\vdots & &\vdots \\ f^{(n)}(x,y) &= 1 + x + y, & f^{(n)}(x_0,y_0) &= x_0 + y_0 + 1. \end{aligned} \tag{6.20}$$

Expansion of $y(x)$ in a Taylor's series about $x = x_0$ yields

$$y(x_0 + h) = y(x_0) + \frac{h(x_0 + y(x_0))}{1!} + \frac{h^2(1 + x_0 + y(x_0))}{2!} + \frac{h^3(1 + x_0 + y(x_0))}{3!} + \frac{h^4(1 + x_0 + y(x_0))}{4!} + \cdots$$

By adding and subtracting $(x_0 + h + 1)$ on the right-hand side,

$$\begin{aligned} y(x_0 + h) &= -x_0 - h - 1 + (1 + x_0 + y(x_0)) \\ &\quad + h(1 + x_0 + y(x_0)) \\ &\quad + \frac{h^2(1 + x_0 + y(x_0))}{2!} + \cdots \\ &= -x_0 - h - 1 + (1 + x_0 + y(x_0)) \\ &\quad \times \left[1 + h + \frac{h^2}{2!} + \frac{h^3}{3!} + \frac{h^4}{4!} + \cdots\right] \end{aligned} \tag{6.21}$$

The differential equation is linear and can be solved analytically by using the integrating factor e^{-x}. Multiplying (6.19) by e^{-x}, integrating both sides, and solving for y yields

$$y = -(x + 1) + Ce^x, \tag{6.22}$$

where C is the integration constant. Evaluation of the integration constant at $(x_0, y(x_0))$ gives

$$C = e^{-x_0}(1 + x_0 + y(x_0)),$$

so that the solution of the initial-value problem is given by

$$y = -x - 1 + (1 + x_0 + y(x_0))e^{-x_0}e^x. \tag{6.23}$$

For $y = y(x_0 + h)$, (6.23) becomes

$$y(x_0 + h) = -x_0 - h - 1 + (1 + x_0 + y(x_0))e^h. \tag{6.24}$$

Since $[1 + h + (h^2/2!) + \cdots]$ of (6.21) is Taylor's expansion of e^h, the analytical solution again agrees with that found by the Taylor's expansion approach. The error caused by truncation after the term containing $f^{(n-1)}$ is given by

$$\frac{(1 + \xi + y(\xi))}{(n+1)!}h^{n+1}, \qquad \xi \text{ in } (x_0, x_0 + h).$$

A procedure for stepping from one value of x to another, that is, from x_0 to $x_0 + h$, follows from expansion of $y(x)$ about x_0. Similarly, algorithms for stepping* from x_i to $x_{i+1} = x_i + h$ can be based upon the Taylor's expansion of $y(x)$ about x_i:

$$y(x_{i+1}) = y(x_i + h) = y(x_i) + hf(x_i,y(x_i)) + \frac{h^2f'(x_i,y(x_i))}{2!} + \frac{h^3f''(x_i,y(x_i))}{3!} + \frac{h^4f'''(x_i,y(x_i))}{4!} + \cdots + \frac{h^nf^{(n-1)}(x_i,y(x_i))}{n!} + \frac{h^{n+1}f^{(n)}(\xi,y(\xi))}{(n+1)!}, \qquad \xi \text{ in } (x_i, x_{i+1}). \tag{6.25}$$

The algorithms, formed by dropping the last term on the right-hand side of (6.25) and replacing $y(x_{i+1})$ on the left-hand side by y_{i+1}, are said to be of *order* h^n. The error is of order h^{n+1}. The *local truncation error* e_t, introduced by one application is therefore bounded as follows:

$$|e_t| \leqslant \frac{h^{n+1}}{(n+1)!}M, \tag{6.26}$$

where

$$M \geqslant |f^{(n)}(\eta,y(\eta))|_{\max}, \qquad \eta \text{ in } (x_i, x_{i+1}).$$

Unfortunately, in the general case, the differentiation of $f(x,y)$ becomes enormously complicated. Except for the simplest case,

$$y(x_{i+1}) = y(x_i) + hf(x_i,y(x_i)) + O(h^2), \tag{6.27}$$

the direct Taylor's expansion of (6.25) is not often used to solve first-order differential equations. Here "$O(\ )$" means "terms of order $(\ )$." (In view of the recent development of computer programs which formally differentiate arbitrary symbolic expressions [1], this argument against the direct Taylor's expansion, with high-order terms included, may well vanish at some future time.)

Since $y(x_0)$ is usually the only value of $y(x_i)$ that is known exactly (assuming that the initial condition is free of error), $y(x_i)$ in (6.27) must in general be replaced by y_i. The algorithm then assumes the form

$$y_1 = y(x_0) + hf(x_0,y(x_0)) \tag{6.28a}$$

$$y_{i+1} = y_i + hf(x_i,y_i) = y_i + hf_i, \qquad i \geqslant 1, \tag{6.28b}$$

which is called *Euler's method*.

6.3 Euler's Method

Because it is the simplest method and the most amenable to an analysis of error propagation, Euler's one-step method of (6.28) will be discussed in some detail, even though accuracy limitations preclude its use for most practical problems. There is a simple geometric interpretation for (6.28a). The solution across the interval $[x_0, x_1]$ is assumed to follow the line tangent to $y(x)$ at x_0 (see

* Throughout the remainder of this chapter, we shall consider the step-size h to be a positive number, that is, integration will be carried out for increasing values of the independent variable x. The algorithms to be developed, however, apply for negative h as well.

Fig. 6.2). When Euler's method is applied repeatedly across several intervals in sequence, the numerical solution traces out a polygon segment with sides of slopes f_i, $i = 0,1,2, \ldots, n-1$.

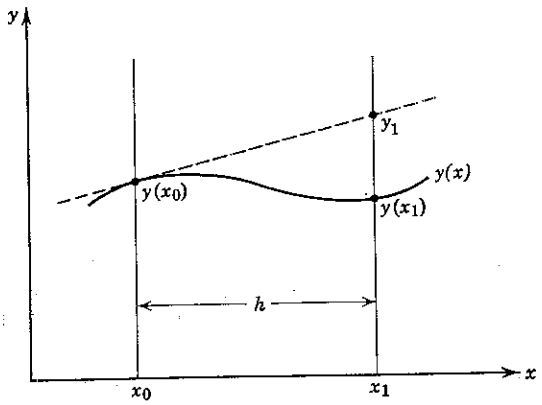


Figure 6.2 Euler's method.

As an example, consider the differential equation of (6.19),

$$\frac{dy}{dx} = x + y,$$

subject to the initial condition $x_0 = 0$, $y(x_0) = y_0 = 0$ (see Example 6.1). The solution already developed in (6.23) for this initial condition is

$$y = e^x - x - 1.$$

The Euler solution is shown in Table 6.1, using a step-size $h = 0.1$ and an upper limit of integration $x_{10} = 1.0$. Column 3 contains the values of y_i computed from (6.28) with all figures retained in the calculation (no round-off error). Column 4 contains the computed derivatives, again with all figures retained. Column 5 shows the true solutions rounded to four figures. Column 6 contains the overall truncation error (rounded to four figures) at each value of x_i. Columns 7 and 8 contain the results which are computed when only four figures are retained at each stage of the calculations. In column 7, the four figures retained are truncated, that is, less significant figures are simply dropped. In column 8, the four figures retained are rounded values. Column 9 shows the values which would be computed by using true y values, that is, $y(x_i)$ rather than y_i, at the beginning of each new step [see (6.27)]. Column 10 shows the local truncation error for each interval when $y(x_i)$ is used to begin each step. Column 11 shows the value of the maximum local truncation error computed from the Taylor's expansion error term with $n = 1$ (see (6.25)),

$$|e_t| = \frac{h^2}{2!}|f'(\xi, y(\xi))|, \quad \xi \text{ in } (x_i, x_{i+1}). \tag{6.29}$$

For the given equation, $f(x,y) = x + y$, the derivative in (6.29) is given by

$$f'(x,y) = \frac{d^2y}{dx^2} = e^x.$$

Since the maximum value of e^x, $x_i \leqslant x \leqslant x_{i+1}$ occurs at $x = x_{i+1}$, the maximum value of the local truncation error (6.29) is given by

$$|e_t|_{\max} = \frac{h^2}{2!} e^{x_{i+1}}. \tag{6.30}$$

Unfortunately, this error bound is valid only for the algorithm of (6.28a) and that obtained from (6.27),

$$y_{i+1} = y(x_i) + hf(x_i, y(x_i)). \tag{6.31}$$

Table 6.1 Solution of Equation (6.19) by Euler's Method

		$y(0) = 0$				$h = 0.1$				
i (1)	x_i (2)	y_i (3)	$f(x_i,y_i)$ (4)	True y Rounded $y(x_i)$ (5)	Overall Truncation Error Rounded $\varepsilon_i = y_i - y(x_i)$ (6)	Truncated at Four Figures y_i (7)	Rounded to Four Figures y_i (8)	y_i Value Computed Using $y(x_{i-1})$ (9)	Local Truncation Error Using $y(x_{i-1})$ (10)	$e^x_{\max}\frac{h^2}{2!}$ (11)
0	0.0	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	—	—
1	0.1	0.0000	0.1000	0.0052	−0.0052	0.0000	0.0000	0.0000	−0.0052	0.0055
2	0.2	0.0100	0.2100	0.0214	−0.0114	0.0100	0.0100	0.0157	−0.0057	0.0061
3	0.3	0.0310	0.3310	0.0499	−0.0189	0.0310	0.0310	0.0435	−0.0064	0.0067
4	0.4	0.0641	0.4641	0.0918	−0.0277	0.0641	0.0641	0.0849	−0.0069	0.0075
5	0.5	0.11051	0.61051	0.1487	−0.0382	0.1105	0.1105	0.1410	−0.0077	0.0082
6	0.6	0.171561	0.771561	0.2221	−0.0505	0.1715	0.1716	0.2136	−0.0085	0.0091
7	0.7	0.2487171	0.9487171	0.3138	−0.0651	0.2486	0.2488	0.3043	−0.0095	0.0101
8	0.8	0.34358881	1.14358881	0.4255	−0.0819	0.3434	0.3437	0.4152	−0.0103	0.0111
9	0.9	0.457947691	1.357947691	0.5596	−0.1017	0.4577	0.4581	0.5480	−0.0116	0.0123
10	1.0	0.5937424601	1.5937424601	0.7183	−0.1246	0.5934	0.5939	0.7056	−0.0127	0.0136

For the general integration between x_i and x_{i+1}, that is, the results of column 3, the actual algorithm is given by (6.28b) as

$$y_{i+1} = y_i + hf(x_i,y_i)$$

The value of y_i used for every interval except the first, where $y_0 = y(x_0)$, is inexact, being the result of previous calculations which involved earlier truncation errors.

6.4 Error Propagation in Euler's Method

Let us examine the *propagation* of local discretization errors in Euler's method of (6.28) applied to the integration of the initial-value problem

$$\frac{dy}{dx} = f(x,y), \qquad y(x_0) = y_0 \tag{6.32}$$

Assume that $f(x,y)$ and its first-order partial derivatives are continuous and bounded in the region $a \leqslant x \leqslant b$, $-\infty < y < \infty$, and that $a < x_0 < x_n < b$. Assume also that a solution $y(x)$ exists. Then there must exist constants M and K such that

$$|y''(x)| = \left| \frac{\partial f(x,y)}{\partial x} + f(x,y)\frac{\partial f(x,y)}{\partial y} \right| \leqslant M \tag{6.33}$$

and

$$|f(x,y^*) - f(x,y)| = \left| \frac{\partial f(x,\alpha)}{\partial y} \right| |y^* - y| \leqslant K|y^* - y|, \tag{6.34}$$

where $y^* < \alpha < y$ for (x,y) and (x,y^*) in the region. Relationship (6.34) follows directly from the differential mean-value theorem of page 9.

As in (6.8), denote the error between the approximate and exact solutions by ε_i, that is, let

$$\varepsilon_i = y_i - y(x_i)$$

Then the additional error, $\Delta\varepsilon_i$, generated in traversing the ith step, is

$$\Delta\varepsilon_i = \Delta y_i - \Delta y(x_i), \tag{6.35}$$

or

$$\varepsilon_{i+1} - \varepsilon_i = y_{i+1} - y_i - [y(x_{i+1}) - y(x_i)], \tag{6.36}$$

subject to the condition $\varepsilon_0 = 0$. From Euler's algorithm of (6.28b),

$$y_{i+1} - y_i = hf(x_i,y_i),$$

whereas, from Taylor's expansion (6.25),

$$y(x_{i+1}) - y(x_i) = hf(x_i,y(x_i)) + \frac{h^2}{2!}f'(\xi,y(\xi)), \qquad x_i < \xi < x_{i+1} \tag{6.37}$$

Thus (6.35) is equivalent to

$$\Delta\varepsilon_i = h[f(x_i,y_i) - f(x_i,y(x_i))] - \frac{h^2}{2}f'(\xi,y(\xi)) \tag{6.38}$$

Then, from (6.33) and (6.34),

$$|\Delta\varepsilon_i| = |\varepsilon_{i+1} - \varepsilon_i| \leqslant hK|y_i - y(x_i)| + \frac{M}{2!}h^2,$$

or

$$|\varepsilon_{i+1} - \varepsilon_i| \leqslant hK|\varepsilon_i| + \frac{M}{2!}h^2 \tag{6.39}$$

Since $|\varepsilon_{i+1}| \leqslant |\varepsilon_{i+1} - \varepsilon_i| + |\varepsilon_i|$, (6.39) can be rewritten as

$$|\varepsilon_{i+1}| \leqslant (1 + hK)|\varepsilon_i| + \frac{M}{2}h^2, \qquad i \geqslant 0 \tag{6.40}$$

To determine $|\varepsilon_i|$, (6.40) can be applied i times with the starting value $\varepsilon_0 = 0$. However, (6.40) has a general solution of the form

$$|\varepsilon_i| \leqslant \frac{Mh}{2K}[(1 + hK)^i - 1] \tag{6.41}$$

That (6.41) is a solution of (6.40) is not immediately apparent, but this can be proved by induction as follows. Consider a general inequality of the form

$$|\varepsilon_{i+1}| \leqslant A|\varepsilon_i| + B, \qquad i \geqslant 0, \tag{6.42}$$

where A and B are independent of i. The proposed solution of (6.42), for $A > 0$, $B \geqslant 0$, $A \neq 1$, is

$$|\varepsilon_i| \leqslant A^i|\varepsilon_0| + \left(\frac{A^i - 1}{A - 1}\right)B, \qquad i > 0 \tag{6.43}$$

For $i = 1$, (6.43) is identical with (6.42). For a general value of i, substitution of (6.43) into (6.42) yields

$$\begin{aligned} |\varepsilon_{i+1}| &\leqslant A\left\{A^i|\varepsilon_0| + \left(\frac{A^i - 1}{A - 1}\right)B\right\} + B \\ &= A^{i+1}|\varepsilon_0| + \left\{A\left(\frac{A^i - 1}{A - 1}\right) + 1\right\}B \\ &= A^{i+1}|\varepsilon_0| + \left(\frac{A^{i+1} - 1}{A - 1}\right)B, \end{aligned}$$

which is just (6.43) with i incremented by unity. Thus, by induction, (6.43) is a solution of (6.42). The error for Euler's method (6.40) is a special form of (6.43), since $\varepsilon_0 = 0$. In this case, $A = (1 + hK)$, $B = Mh^2/2$, so that the solution is

$$\begin{aligned} |\varepsilon_i| \leqslant A^i|\varepsilon_0| + \left(\frac{A^i - 1}{A - 1}\right)B &= \left(\frac{A^i - 1}{A - 1}\right)B \\ &= \left[\frac{(1 + hK)^i - 1}{1 + hK - 1}\right]\frac{M}{2}h^2 \\ &= \frac{Mh}{2K}[(1 + hK)^i - 1], \end{aligned}$$

which is the given solution (6.41).

The solution of (6.41) can be simplified further, since

$$(1 + hK) < e^{hK}, \tag{6.44}$$

which follows from Taylor's expansion of e^{hK}. Substitution of (6.44) into (6.41) gives

$$|\varepsilon_i| \leqslant \frac{Mh}{2K}(e^{ihK} - 1) \leqslant \frac{Mh}{2K} e^{ihK} \tag{6.45}$$

For $nh = x_n - x_0 = L$, the constant total interval of integration, (6.45) can be written as

$$|\varepsilon_n| \leqslant \frac{Mh}{2K} e^{LK}$$

Then, as h approaches zero, the error approaches zero, because

$$\lim_{h \to 0} |\varepsilon_n| \leqslant \lim_{h \to 0} \frac{Mh}{2K} e^{LK} = 0$$

Notice that this is true despite the fact that $n \to \infty$ as $h \to 0$. A numerical procedure for which, when $0 \leqslant i \leqslant n$,

$$\lim_{h \to 0} |\varepsilon_i| = 0,$$

is said to be *convergent*. Thus, Euler's method converges with an overall truncation error for which

$$|\varepsilon_i| = |y_i - y(x_i)| = O(h) \tag{6.46}$$

Note that although the *local* truncation error (6.29) for Euler's method is of order h^2, (6.46) shows that the *total* truncation error is of order h.

For the equation of (6.19)

$$f(x,y) = x + y,$$

subject to the initial condition $y(x_0) = y(0) = 0$, the analytical solution is

$$y = e^x - x - 1.$$

Since M can be taken as an upper bound of the magnitude of $f'(x,y)$ on the interval $[x_0, x_i]$, let

$$M = |f'(x,y)|_{\max} = |e^x|_{\max} \tag{6.47}$$

Also, K can be taken as an upper bound on the partial derivative of $f(x,y)$ with respect to y, that is,

$$K = |f_y(x,y)|_{\max} = 1. \tag{6.48}$$

Using these values for M and K, and with $h = 0.1$ as before, an upper bound on the truncation error $|\varepsilon_i|$ is, from (6.41),

$$|\varepsilon_i| \leqslant \frac{0.1e^{x_i}}{2}[(1 + 0.1)^i - 1] \tag{6.49}$$

Values of $|\varepsilon_i|_{\max}$ computed from the right-hand side of (6.49) are listed below. As expected, all these error bounds (Table 6.2) are greater than the true truncation error (from Table 6.1, column 6).

Table 6.2 Total Truncation Error for Euler's Method Solution of Equation (6.19)

i	x_i	$\lvert\varepsilon_i\rvert_{\max}$	$\varepsilon_i = y_i - y(x_i)$
0	0.0	—	0.0000
1	0.1	0.0055	−0.0052
2	0.2	0.0128	−0.0114
3	0.3	0.0222	−0.0189
4	0.4	0.0349	−0.0277
5	0.5	0.0502	−0.0382
6	0.6	0.0704	−0.0505
7	0.7	0.0960	−0.0651
8	0.8	0.1226	−0.0819
9	0.9	0.1678	−0.1017
10	1.0	0.2175	−0.1246

EXAMPLE 6.1

EULER'S METHOD

Problem Statement

Write a program that uses Euler's method to solve the first-order equation (6.19),

$$f(x, y) = x + y,$$

with the initial condition $y(x_0) = y(0) = 0$. Integrate the equation on the interval $0 \leqslant x \leqslant x_{max}$ using several different step sizes, h. Print the results after every k steps (i.e., for $x = x_0, x_k, x_{2k}, \ldots$) and compare the results with the true solution (6.23):

$$y(x_i) = e^{x_i} - x_i - 1, \qquad i = 0, k, 2k, 3k, \ldots$$

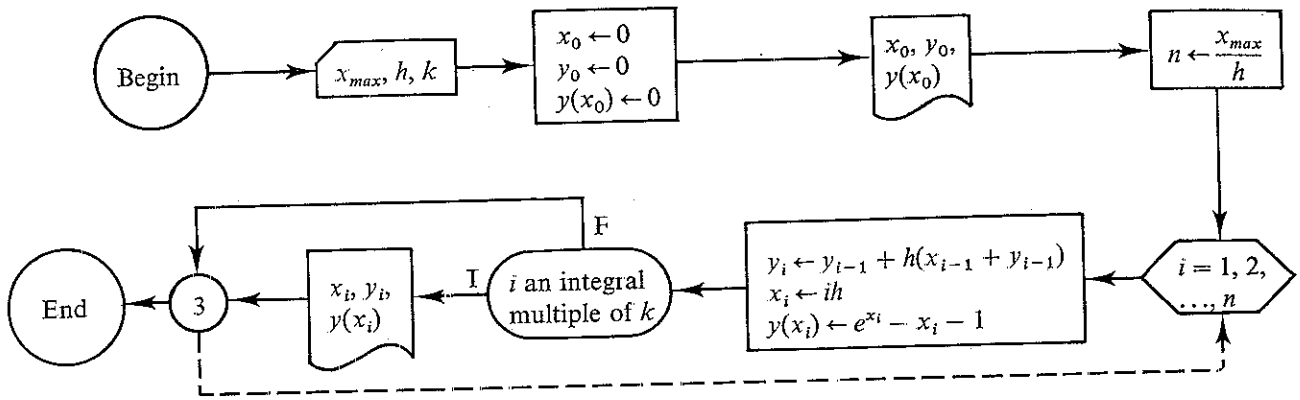
Method of Solution

Euler's algorithm (6.28),

$$\begin{aligned} y_i &= y_{i-1} + hf(x_{i-1}, y_{i-1}) \\ &= y_{i-1} + h(x_{i-1} + y_{i-1}), \end{aligned}$$

is used to implement the solution of (6.19).

Flow Diagram



FORTRAN *Implementation*

List of Principal Variables

Program Symbol	*Definition*
H	Step size, h.
I	Step counter, i.
IPRINT	Number of steps between printout, k.
NSTEPS	Total number of steps, x_{max}/h.
TRUEY	True solution, $y(x_i)$.
X	Independent variable, x_i.
XMAX	Maximum value of the independent variable, x_{max}.
Y	Computed solution, y_i.

Program Listing

```
C       APPLIED NUMERICAL METHODS, EXAMPLE 6.1
C       EULER'S METHOD
C
C       THIS PROGRAM USES EULERS METHOD TO COMPUTE THE SOLUTION OF THE
C       ORDINARY DIFFERENTIAL EQUATION  DY/DX = X+Y  ON THE INTERVAL
C       (0,XMAX) WITH STEPSIZE H AND INITIAL CONDITION Y(0) = 0.
C       I IS THE COUNTER FOR THE NUMBER OF APPLICATIONS OF THE METHOD.
C       NSTEPS IS THE TOTAL NUMBER OF STEPS FOR EULER'S METHOD.
C       TRUEY IS THE ANALYTICAL SOLUTION  Y(X) = EXP(X) - X - 1.
C       SOLUTIONS ARE PRINTED AFTER EVERY IPRINT STEPS.
C
      IMPLICIT  REAL*8(A-H, O-Z)
C
C     ..... READ DATA AND SET INITIAL CONDITIONS .....
    1 READ (5,100)  XMAX, H, IPRINT
      WRITE (6,200)  XMAX, H, IPRINT
      X = 0.
      Y = 0.
      TRUEY = 0.
      WRITE (6,201)  X, Y, TRUEY
C
C     ..... EULERS METHOD INTEGRATION .....
      NSTEPS = (XMAX + H/2.)/H
      DO 3  I = 1, NSTEPS
      Y = Y + H*(X + Y)
      X = FLOAT(I)*H
      TRUEY = DEXP(X) - X - 1.
    3 IF (I/IPRINT*IPRINT.EQ.I)  WRITE (6,201)  X, Y, TRUEY
      GO TO 1
C
C     ..... FORMATS FOR INPUT AND OUTPUT STATEMENTS .....
  100 FORMAT ( 10X,F10.6,17X,F10.6,20X,I5 )
  200 FORMAT ( 10H1XMAX    = , F12.6/ 10H H       = , F12.6/ 10H IPRINT =
     1 I5/ 1H0, 6X, 1HX, 15X, 1HY, 13X, 5HTRUEY/ 1H )
  201 FORMAT ( 1H , F10.6, 2F16.6 )
C
      END
```

Data

```
XMAX   =    1.000000        H   =    1.000000        IPRINT   =     1
XMAX   =    1.000000        H   =    0.500000        IPRINT   =     1
XMAX   =    1.000000        H   =    0.250000        IPRINT   =     1
XMAX   =    1.000000        H   =    0.100000        IPRINT   =     1
XMAX   =    1.000000        H   =    0.010000        IPRINT   =    10
XMAX   =    1.000000        H   =    0.001000        IPRINT   =   100
XMAX   =    1.000000        H   =    0.000100        IPRINT   =  1000
XMAX   =    1.000000        H   =    0.000010        IPRINT   = 10000
```

Computer Output

Results for the 1st Data Set

```
XMAX   =     1.000000
H      =     1.000000
IPRINT =     1

      X                Y              TRUEY

  0.0              0.0              0.0
  1.000000         0.0              0.718282
```

Computer Output (*Continued*)

Results for the 2nd Data Set

```
XMAX   =     1.000000
H      =     0.500000
IPRINT =     1

        X              Y              TRUEY

     0.0            0.0            0.0
     0.500000       0.0            0.148721
     1.000000       0.250000       0.718282
```

Results for the 3rd Data Set

```
XMAX   =     1.000000
H      =     0.250000
IPRINT =     1

        X              Y              TRUEY

     0.0            0.0            0.0
     0.250000       0.0            0.034025
     0.500000       0.062500       0.148721
     0.750000       0.203125       0.367000
     1.000000       0.441406       0.718282
```

Results for the 4th Data Set

```
XMAX   =     1.000000
H      =     0.100000
IPRINT =     1

        X              Y              TRUEY

     0.0            0.0            0.0
     0.100000       0.0            0.005171
     0.200000       0.010000       0.021403
     0.300000       0.031000       0.049859
     0.400000       0.064100       0.091825
     0.500000       0.110510       0.148721
     0.600000       0.171561       0.222119
     0.700000       0.248717       0.313753
     0.800000       0.343589       0.425541
     0.900000       0.457948       0.559603
     1.000000       0.593742       0.718282
```

Results for the 5th Data Set

```
XMAX   =     1.000000
H      =     0.010000
IPRINT =     10

        X              Y              TRUEY

     0.0            0.0            0.0
     0.100000       0.004622       0.005171
     0.200000       0.020190       0.021403
     0.300000       0.047849       0.049859
     0.400000       0.088864       0.091825
     0.500000       0.144632       0.148721
     0.600000       0.216697       0.222119
     0.700000       0.306763       0.313753
     0.800000       0.416715       0.425541
     0.900000       0.548633       0.559603
     1.000000       0.704814       0.718282
```

Computer Output (*Continued*)

Results for the 6th Data Set

```
XMAX   =      1.000000
H      =      0.001000
IPRINT =    100
```

X	Y	TRUEY
0.0	0.0	0.0
0.100000	0.005116	0.005171
0.200000	0.021281	0.021403
0.300000	0.049656	0.049859
0.400000	0.091527	0.091825
0.500000	0.148309	0.148721
0.600000	0.221573	0.222119
0.700000	0.313048	0.313753
0.800000	0.424651	0.425541
0.900000	0.558497	0.559603
1.000000	0.716924	0.718282

Results for the 7th Data Set

```
XMAX   =      1.000000
H      =      0.000100
IPRINT =   1000
```

X	Y	TRUEY
0.0	0.0	0.0
0.100000	0.005165	0.005171
0.200000	0.021391	0.021403
0.300000	0.049839	0.049859
0.400000	0.091795	0.091825
0.500000	0.148680	0.148721
0.600000	0.222064	0.222119
0.700000	0.313682	0.313753
0.800000	0.425452	0.425541
0.900000	0.559492	0.559603
1.000000	0.718146	0.718282

Results for the 8th Data Set

```
XMAX   =      1.000000
H      =      0.000010
IPRINT =  10000
```

X	Y	TRUEY
0.0	0.0	0.0
0.100000	0.005170	0.005171
0.200000	0.021402	0.021403
0.300000	0.049857	0.049859
0.400000	0.091822	0.091825
0.500000	0.148717	0.148721
0.600000	0.222113	0.222119
0.700000	0.313746	0.313753
0.800000	0.425532	0.425541
0.900000	0.559592	0.559603
1.000000	0.718268	0.718282

Discussion of Results

Equation (6.19) has been solved on the interval [0,1] using Euler's method with several different step sizes, $h = 1.0, 0.5, 0.25, 0.1, 0.01, 0.001, 0.0001$, and 0.00001. The errors in the computed solution at $x = 1$ as a function of the step size are shown in Table 6.1.1 and Fig. 6.1.1. As expected from the error analysis of Section 6.4, the error decreases linearly with h for small h (see (6.46)).

Table 6.1.1 Error in Euler's Method Solution at $x = 1$

Step size, h	Error at $x = 1$
1.0	0.718282
0.5	0.468282
0.25	0.276876
0.1	0.124540
0.01	0.013468
0.001	0.001358
0.0001	0.000136
0.00001	0.000014

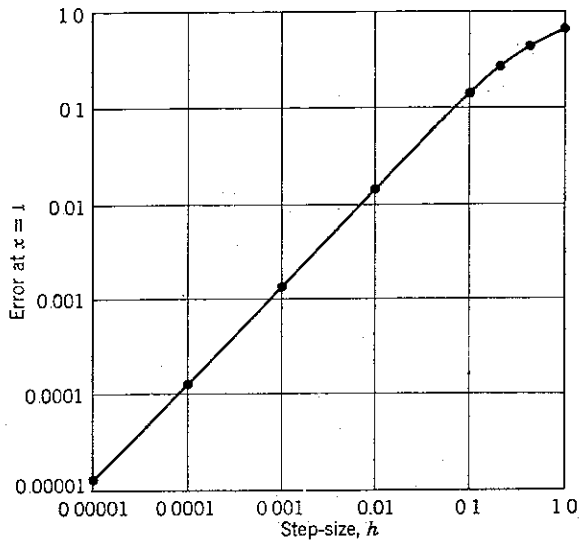


Figure 6.1.1 Error in Euler's method solution at $x = 1$

6.5 Runge-Kutta Methods

The solution of a differential equation by direct Taylor's expansion of the object function is generally not practical if derivatives of higher order than the first are retained. For all but the simplest equations, the necessary higher-order derivatives tend to become quite complicated. In addition, as shown by the preceding examples, each problem results in a specific series for its solution. Thus, when higher-order error terms are desired, no simple algorithm analogous to Euler's method can be developed directly from the Taylor's expansion.

Fortunately, it is possible to develop one-step procedures which involve only first-order derivative evaluations, but which also produce results equivalent in accuracy to the higher-order Taylor formulas. These algorithms are called the *Runge-Kutta* methods. Approximations of the second, third, and fourth orders (that is, approximations with accuracies equivalent to Taylor's expansions of $y(x)$ retaining terms in h^2, h^3, and h^4, respectively) require the estimation of $f(x, y)$ at two, three, and four values, respectively, of x on the interval $x_i \leqslant x \leqslant x_{i+1}$. Methods of order m, where m is greater than four, require derivative evaluations at more than m nearby points [14]

All the Runge-Kutta methods have algorithms of the form

$$y_{i+1} = y_i + h\phi(x_i, y_i, h). \tag{6.50}$$

Here, ϕ, termed the *increment* function by Henrici [2], is simply a suitably chosen approximation to $f(x,y)$ on the interval $x_i \leqslant x \leqslant x_{i+1}$. Because there is a considerable amount of algebra involved in the development of the higher-order Runge-Kutta formulas, only the simplest of these procedures (the second-order algorithm) will be developed in detail. The derivation of other Runge-Kutta formulas is analogous to the one which follows.

Let ϕ be a weighted average of two derivative evaluations k_1 and k_2 on the interval $x_i \leqslant x \leqslant x_{i+1}$, that is,

$$\phi = ak_1 + bk_2, \tag{6.51}$$

which leads to the Runge-Kutta algorithm

$$y_{i+1} = y_i + h(ak_1 + bk_2). \tag{6.52}$$

Let

$$\begin{aligned} k_1 &= f(x_i, y_i), \\ k_2 &= f(x_i + ph,\ y_i + qhf(x_i, y_i)) \\ &= f(x_i + ph,\ y_i + qhk_1), \end{aligned} \tag{6.53}$$

where p and q are constants to be established later. The quantities hk_1 and hk_2 have a simple geometric interpretation, which will become apparent once p and q are determined.

First, expand k_2 in a Taylor's series for a function of two variables*, and drop all terms in which the exponent of h is greater than one:

$$\begin{aligned} k_2 &= f(x_i + ph,\ y_i + qhf(x_i, y_i)) \\ &= f(x_i, y_i) + phf_x(x_i, y_i) \\ &\quad + qhf(x_i, y_i) f_y(x_i, y_i) + O(h^2). \end{aligned} \tag{6.54}$$

From (6.53) and (6.54), (6.52) becomes

$$\begin{aligned} y_{i+1} = y_i &+ h[af(x_i, y_i) + bf(x_i, y_i)] \\ &+ h^2[bpf_x(x_i, y_i) + bqf(x_i, y_i) f_y(x_i, y_i)] + O(h^3). \end{aligned} \tag{6.55}$$

Next, expand the object function $y(x)$ about x_i by using Taylor's series (6.25), as before:

$$\begin{aligned} y(x_i + h) = y(x_{i+1}) &= y(x_i) + hf(x_i, y(x_i)) \\ &+ \frac{h^2}{2} f'(x_i, y(x_i)) + \frac{h^3}{3!} f''(\xi, y(\xi)), \\ &\xi \text{ in } (x_i, x_{i+1}). \end{aligned} \tag{6.56}$$

By chain-rule differentiation (6.10), $f'(x_i, y(x_i))$ is given by

$$f'(x_i, y(x_i)) = f_x(x_i, y(x_i)) + f_y(x_i, y(x_i)) f(x_i, y(x_i)).$$

Finally, equate terms in like powers of h in (6.55) and (6.56):

Power of h	*Expansion of y(x)*	*Runge-Kutta Algorithm*
0	$y(x_i)$	y_i
1	$f(x_i, y(x_i))$	$(a + b) f(x_i, y_i)$
2	$\frac{1}{2}[f_x(x_i, y(x_i)) + f_y(x_i, y(x_i)) f(x_i, y(x_i))]$	$[bpf_x(x_i, y_i) + bqf_y(x_i, y_i) f(x_i, y_i)]$

Assuming that $y_i = y(x_i)$ and that we want equality of the coefficients of h^2 for all suitably differentiable functions $f(x,y)$, we find that $a + b = 1$, $bp = 1/2$, and $bq = 1/2$. Thus

$$\begin{aligned} a &= 1 - b, \\ p &= q = \frac{1}{2b}. \end{aligned} \tag{6.57}$$

Since the three equations of (6.57) contain four unknowns, the system is underdetermined, that is, there is one variable, say b, which may be chosen arbitrarily. The two common choices are $b = \frac{1}{2}$ and $b = 1$.

* The first few terms of the two-variable Taylor's series are:

$$\begin{aligned} f(x + r, y + s) = f(x,y) &+ rf_x(x,y) + sf_y(x,y) + r^2 f_{xx}(x,y)/2 \\ &+ rsf_{xy}(x,y) + s^2 f_{yy}(x,y)/2 + O[(|r| + |s|)^3]. \end{aligned}$$

For b = $\frac{1}{2}$. $a = \frac{1}{2}$, $p = 1$, $q = 1$. Then (6.52) becomes

$$y_{i+1} = y_i + \frac{h}{2}[f(x_i,y_i) + f(x_i + h, y_i + hf(x_i,y_i))], \tag{6.58}$$

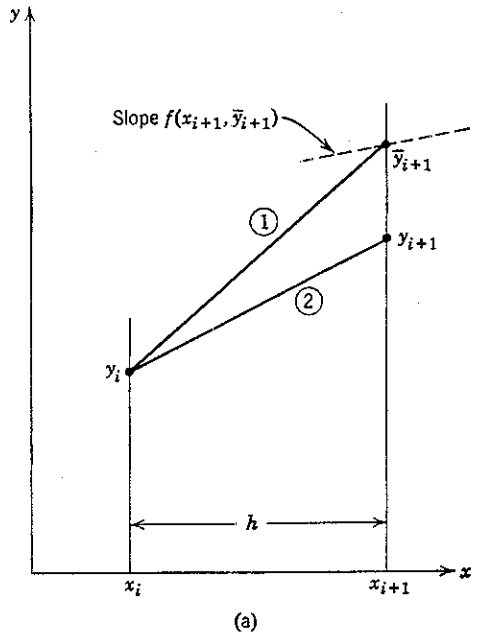
which can also be written

$$y_{i+1} = y_i + \frac{h}{2}[f(x_i,y_i) + f(x_{i+1},\bar{y}_{i+1})], \tag{6.59}$$

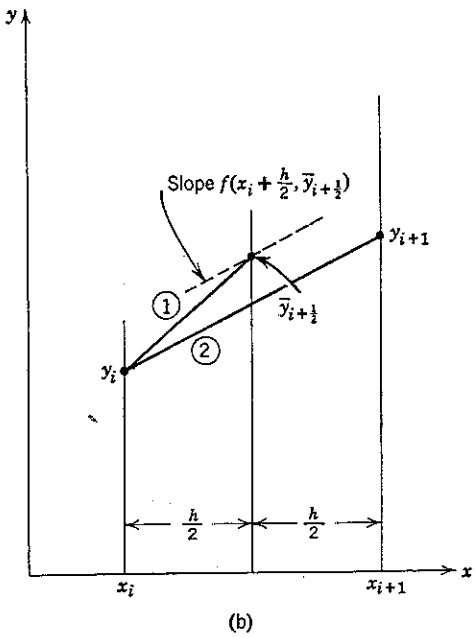
where

$$\bar{y}_{i+1} = y_i + hf(x_i,y_i). \tag{6.60}$$

The one-step algorithm of (6.59) and (6.60) is known as the *improved Euler's method* or *Heun's method*, and has the geometric interpretation shown in Fig. 6.3*a*.



(a)



(b)

Figure 6.3 Second-order Runge-Kutta procedures.

Essentially, Euler's method is employed twice in sequence. First, equation (6.60) is used to predict $\bar{y}_{i+1}$, a preliminary estimate of y_{i+1}. That is, $\bar{y}_{i+1}$ is the ordinate, at $x = x_{i+1}$, of the straight line ①, passing through (x_i,y_i) with slope $f(x_i,y_i) = k_1$. Second, an improved estimate y_{i+1} is obtained from equation (6.59); the slope of line ②, used for this purpose, is the weighted average of approximations to f at the two ends of the interval. Note that although the true derivative at x_{i+1} is $f(x_{i+1}, y(x_{i+1}))$, it is approximated by $f(x_{i+1},\bar{y}_{i+1})$, since $y(x_{i+1})$ is unknown.

Euler's algorithm of (6.60) may be viewed as a *predicting* equation for $\bar{y}^{(1)}_{i+1}$ (the first approximation to y_{i+1}), whereas (6.59) may be considered a *correcting* equation to produce an improved estimate of y_{i+1}. Equation (6.59) may be used iteratively to produce a sequence of corrected y_{i+1} values, $\bar{y}^{(2)}_{i+1}, \bar{y}^{(3)}_{i+1}, \ldots, \bar{y}^{(m)}_{i+1}$. In this case, the pair of equations, (6.60) and (6.59), leads to the simplest of the so-called *predictor-corrector* methods which are described in more detail in Section 6.11.

For b = 1. $a = 0, p = 1/2, q = 1/2$. Then (6.52) becomes

$$y_{i+1} = y_i + hf\left(x_i + \frac{h}{2}, \bar{y}_{i+1/2}\right), \tag{6.61}$$

where

$$\bar{y}_{i+1/2} = y_i + \frac{h}{2}f(x_i,y_i). \tag{6.62}$$

The one-step algorithm of (6.61) and (6.62) is called the *improved polygon method* or the *modified Euler's method*, illustrated in Fig. 6.3*b*. Again, Euler's method is employed twice in sequence. First, from (6.62), an approximation $\bar{y}_{i+1/2}$ is obtained at the halfway point $x_i + h/2$. Second, (6.61) evaluates $f(x,y)$ for $x = x_i + h/2$, $y = \bar{y}_{i+1/2}$, and uses this as the average derivative for proceeding over the whole interval.

The higher-order Runge-Kutta methods are developed analogously. For example, the increment function for the third-order method is

$$\phi = ak_1 + bk_2 + ck_3,$$

where k_1, k_2, and k_3 approximate the derivative at various points on the integration interval $[x_i, x_{i+1}]$. In this case,

$$k_1 = f(x_i,y_i),$$

$$k_2 = f(x_i + ph, y_i + phk_1),$$

$$k_3 = f(x_i + rh, y_i + shk_2 + (r - s)hk_1)$$

The *third-order* Runge-Kutta algorithms are given by

$$y_{i+1} = y_i + h(ak_1 + bk_2 + ck_3) \tag{6.63}$$

To determine the constants a, b, c, p, r, and s, we first expand k_2 and k_3 about (x_i,y_i) in Taylor's series as functions of two variables. The object function $y(x)$ is expanded in a Taylor's series as before, (6.25). Coefficients of like powers of h through the h^3 terms in (6.63) and (6.25) are equated to produce a formula with a local truncation error of order h^4. Details of the argument are essentially the same as in the development of the second-order methods.

Again, there are fewer equations than unknowns:

$$\begin{aligned} a + b + c &= 1, \\ bp + cr &= \tfrac{1}{2}, \\ bp^2 + cr^2 &= \tfrac{1}{3}, \\ cps &= \tfrac{1}{6}. \end{aligned}$$

Two of the constants a, b, c, p, r, and s are arbitrary. For one set of constants, selected by Kutta, the third-order method is:

$$\begin{aligned} y_{i+1} &= y_i + \frac{h}{6}(k_1 + 4k_2 + k_3), \\ k_1 &= f(x_i,y_i), \\ k_2 &= f(x_i + \tfrac{1}{2}h,\ y_i + \tfrac{1}{2}hk_1), \\ k_3 &= f(x_i + h, y_i + 2hk_2 - hk_1). \end{aligned} \tag{6.64}$$

Note that if $f(x,y)$ is a function of x only, then (6.64) reduces to Simpson's rule (2.21b).

All the *fourth-order* formulas are of the form

$$y_{i+1} = y_i + h(ak_1 + bk_2 + ck_3 + dk_4), \tag{6.65}$$

where k_1, k_2, k_3, and k_4 are approximate derivative values computed on the interval $x_i \leqslant x \leqslant x_{i+1}$. Several fourth-order algorithms are used. The following is attributed to Kutta:

$$\begin{aligned} y_{i+1} &= y_i + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4), \\ k_1 &= f(x_i,y_i), \\ k_2 &= f(x_i + \tfrac{1}{2}h, y_i + \tfrac{1}{2}hk_1), \\ k_3 &= f(x_i + \tfrac{1}{2}h, y_i + \tfrac{1}{2}hk_2), \\ k_4 &= f(x_i + h, y_i + hk_3). \end{aligned} \tag{6.66}$$

Again, note that (6.66) reduces to Simpson's rule if $f(x,y)$ is a function of x only. Another fourth-order method, also ascribed to Kutta, is:

$$\begin{aligned} y_{i+1} &= y_i + \frac{h}{8}(k_1 + 3k_2 + 3k_3 + k_4), \\ k_1 &= f(x_i,y_i), \\ k_2 &= f(x_i + \tfrac{1}{3}h, y_i + \tfrac{1}{3}hk_1), \\ k_3 &= f(x_i + \tfrac{2}{3}h, y_i - \tfrac{1}{3}hk_1 + hk_2), \\ k_4 &= f(x_i + h, y_i + hk_1 - hk_2 + hk_3). \end{aligned} \tag{6.67}$$

This reduces to Simpson's second rule (2.21c) if $f(x,y)$ is a function of x only.

The most widely used fourth-order method (and very likely the most widely used single-step method for solving ordinary differential equations as well) is the one credited to Gill [6]:

$$\begin{aligned} y_{i+1} &= y_i + \frac{h}{6}\left[k_1 + 2\left(1 - \frac{1}{\sqrt{2}}\right)k_2 + 2\left(1 + \frac{1}{\sqrt{2}}\right)k_3 + k_4\right], \\ k_1 &= f(x_i,y_i), \\ k_2 &= f(x_i + \tfrac{1}{2}h, y_i + \tfrac{1}{2}hk_1), \\ k_3 &= f\left(x_i + \frac{1}{2}h, y_i + \left(-\frac{1}{2} + \frac{1}{\sqrt{2}}\right)hk_1 + \left(1 - \frac{1}{\sqrt{2}}\right)hk_2\right), \\ k_4 &= f\left(x_i + h, y_i - \frac{1}{\sqrt{2}}hk_2 + \left(1 + \frac{1}{\sqrt{2}}\right)hk_3\right). \end{aligned} \tag{6.68}$$

Originally, the constants were chosen to reduce the amount of temporary storage required in the solution of sizable systems of first-order equations. With the advent of machines having large memories, the necessity for saving a few memory locations has largely disappeared, but the Runge-Kutta subroutines in most computer-program libraries still employ the Gill constants.

Runge-Kutta formulas of higher order can be developed by extending the procedures outlined in this section (families of fifth-order formulas can be found in [25, 40, 41].)

6.6 Truncation Error, Stability, and Step-Size Control in the Runge-Kutta Algorithms

Since the mth-order Runge-Kutta algorithms of Section 6.5 were generated by requiring that (6.50) agree with the Taylor's expansion of the solution function $y(x)$ through terms of order h^m, the local truncation error e_t is of the form

$$e_t = Kh^{m+1} + O(h^{m+2}), \tag{6.69}$$

where K depends (in a complicated way usually) upon $f(x,y)$ and its higher-order partial derivatives. If one assumes that h is sufficiently small, so that the error is dominated by the first term in (6.69), it is possible, though not at all simple, to find bounds for K [7, 8]. In general, such bounds depend upon bounds for $f(x, y)$ and its various partial derivatives, and upon the particular Runge-Kutta method used. Ralston [8] shows that particular choices of the free parameters in the underdetermined equations for the constants in the algorithm [see P6.15] will tend to minimize the upper bound on K.

In order to choose a reasonable step size, one needs some estimate of the error being committed in integrating across one step. On the one hand, the step size should be small enough to achieve required accuracy (if possible); on the other, it should be as large as possible in order to keep rounding errors (a function of the number of arithmetic operations performed) under control and to avoid an excessive number of derivative evaluations. This latter consideration is very important, particularly when the differential equation is complicated and each derivative evaluation requires substantial computing time. For the mth-order methods of the previous section, the derivative must be evaluated m times for *each* integration step.

One approach to this problem is to assume that the local truncation errors have the form Kh^{m+1} with K constant, and that the local truncation error, committed in traversing one step, dominates the change in total error for the step. Then an estimate of the local truncation error can be found by integrating between two points, say x_n and x_{n+1}, using two different step sizes h_1 and h_2 to evaluate y_{n+1}; let the corresponding solutions be $y_{n+1,1}$ and $y_{n+1,2}$. Then if y^*_{n+1} is the "true" solution, we can employ Richardson's extrapolation technique described on page 78 as follows*:

$$y^*_{n+1} - y_{n+1,1} = Kh_1^{m+1}\frac{(x_{n+1} - x_n)}{h_1}, \qquad (6.70)$$

$$y^*_{n+1} - y_{n+1,2} = Kh_2^{m+1}\frac{(x_{n+1} - x_n)}{h_2}.$$

Dividing the first of these equations by the second, and solving for y^*_{n+1} yields

$$y^*_{n+1} = \frac{y_{n+1,1} - y_{n+1,2}(h_1/h_2)^m}{1 - (h_1/h_2)^m}. \qquad (6.71)$$

If we choose $h_2 = h_1/2$, (6.71) becomes

$$y^*_{n+1} = \frac{y_{n+1,1} - 2^m y_{n+1,2}}{1 - 2^m}, \qquad (6.72)$$

and an estimate of the local truncation error for the solution $y_{n+1,1}$, assuming that $(x_{n+1} - x_n) = h_1$, is given by (6.70) and (6.72) as

$$e_t = Kh_1^{m+1} = \frac{2^m(y_{n+1,2} - y_{n+1,1})}{2^m - 1} \qquad (6.73)$$

For the fourth-order Runge-Kutta method, $m = 4$ and (6.73) becomes

$$e_t = Kh_1^5 = \frac{16}{15}(y_{n+1,2} - y_{n+1,1}). \qquad (6.74)$$

Unfortunately, if we use (6.73) as a monitoring procedure for the integration step size on every integration interval, the total number of calculations is approximately trebled over the number required for integration using just one step size, h_1. As a compromise, we could use the monitoring procedure less frequently, for instance, for every kth step.

Another criterion, suggested by Collatz [9] for the Runge-Kutta method of (6.66), is to calculate $|(k_3 - k_2)/(k_2 - k_1)|$ after each integration step. If the ratio becomes large (more than a few hundredths [10]), then the step size should be decreased. This is only a very qualitative guideline, but has the virtue that the added computation is negligible.

If m is odd, Call and Reeves [11] suggest integrating in the reverse direction, from x_{n+1} to x_n with h replaced by $-h$, after having integrated across the step in the forward direction. The truncation error is estimated as half the difference between y_n and y^*_n, where y^*_n is the solution found as a result of the reverse integration. Unfortunately, when m is even, the method fails, since the truncation errors in one direction exactly cancel those in the other and, aside from rounding errors, $y_n = y^*_n$.

Determining a bound for the accumulated or propagated error for the Runge-Kutta algorithms is difficult [12,13]. Reported bounds are usually very conservative [7]; in addition, the parameters essential for their computation are only rarely available. In general, if the local truncation error for a one-step method is of $O(h^{m+1})$, then the accumulated error will be of $O(h^m)$, [3,4], that is, the reduction in the order of the error is similar to that observed for Euler's method.

All the Runge-Kutta methods can be shown [2] to be convergent, that is, $\lim_{h\to 0}(y_i - y(x_i)) = 0$ (see page 347). Another criterion for selecting an algorithm for the solution of a differential equation with given initial conditions is *stability*. Stability is a somewhat ambiguous term and appears in the literature with a variety of qualifying adjectives (inherent, partial, relative, weak, strong, absolute, etc.). In general, a solution is said to be *unstable* if errors introduced at some stage in the calculations (for example, from erroneous initial conditions or local truncation or round-off errors) are propagated without bound throughout subsequent calculations.

Certain equations with specified initial conditions cannot be solved by any step-by-step integration procedure without exhibiting instability, and are said to be *inherently* unstable. For example, consider the equation of (6.19),

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

for which the analytical solution is given by (6.23) as

$$y(x) = -x - 1 + [1 + x_0 + y(x_0)]e^{-x_0}e^x.$$

With the initial condition $y(0) = -1$, the analytical solution is

$$y(x) = -x - 1.$$

* Throughout this section, n should be considered as a general subscript, and not the subscript of the final base point as in (6.5)

Thus the exponential term in the general solution vanishes because of the particular choice of the initial condition. Even a very tiny change in the initial condition (for example, $y(0) = -0.99999$) will eventually cause a drastic change in the magnitude (even the sign in this case) of the solution for large values of x. Therefore, even though the coefficient of the exponential term is quite small, the contribution of the exponential term will eventually swamp the contribution of the linear terms in the solution. When such an equation is solved by using one of the one-step methods, each new step may be viewed as the solution of a new initial-value problem. Even if the initial condition is error-free for the first step, the initial conditions for subsequent steps will inevitably contain errors introduced by truncation and round-off in preceding steps; the calculated solution for large x will bear no resemblance to the true solution.

Inherent instability is associated with the equation being solved and the initial conditions specified, but does not depend on the particular algorithm being used. Depending on the equation being solved, its initial conditions, *and* the particular one-step method being used, another form of instability, *partial instability* [15], may be observed, even when the equation is not inherently unstable. This phenomenon is related to the step size chosen, and is perhaps seen most easily by examining the Euler's method of algorithm of (6.28b). From (6.38), the total error at x_{i+1} is related to the total error at x_i by

$$\varepsilon_{i+1} = \varepsilon_i + h[f(x_i,y_i) - f(x_i,y(x_i))] - \frac{h^2}{2} f'(\xi,y(\xi)), \tag{6.75}$$

where $x_i < \xi < x_{i+1}$. From the differential mean-value theorem of page 9, we may write

$$f(x_i,y_i) - f(x_i,y(x_i)) = (y_i - y(x_i)) \left.\frac{\partial f}{\partial y}\right|_{x_i,\alpha}$$

with α in $(y_i,y(x_i))$. Since $[y_i - y(x_i)]$ is just ε_i, (6.75) may be written

$$\varepsilon_{i+1} = \varepsilon_i \left(1 + h \left.\frac{\partial f}{\partial y}\right|_{x_i,\alpha}\right) - \frac{h^2}{2} f'(\xi,y(\xi)),$$

$$\xi \text{ in } (x_i,x_{i+1}), \quad \alpha \text{ in } (y_i,y(x_i)). \tag{6.76}$$

The first term on the right-hand side of (6.76) is the contribution of the propagated error to the error at x_{i+1} while the second term is the local truncation error. Clearly, if $\partial f/\partial y$ is negative, then a value of h can be found which will make $[1 + h(\partial f/\partial y)] < 1$, and the error will tend to diminish or die out: the solution will be stable. If $[1 + h(\partial f/\partial y)] > 1$, that is, for $\partial f/\partial y$ positive, the error at x_i will be amplified in traversing the ith step, and the solution will tend toward instability. Even in these cases, however, it may be possible to keep the propagation error under control, especially during the early course of the integration, by choosing a sufficiently small h, that is, by keeping the *propagation factor* $[1 + h(\partial f/\partial y)]$ close to 1.

Suppose that $\partial f/\partial y$ is positive and constant, so that the propagation factor is greater than one for all h and the error does increase without bound for increasing x as shown by (6.76). For example, consider equation (6.15),

$$\frac{dy}{dx} = 2y,$$

for which the solution (6.18) is

$$y(x) = y(x_0)e^{2(x-x_0)}.$$

Will the unbounded growth of the error invalidate the computed solution? Not necessarily, since the solution itself is unbounded for increasing x. The most important criterion is not that the *absolute* error ε_i be bounded, but that the *relative* error ε_i/y_i not grow appreciably.

Similar, though more complicated, propagation factors can be developed for higher-order one-step methods [3]. The quantity $h(\partial f/\partial y)$, sometimes called the *step factor*, contributes to these propagation factors in a manner comparable to that for Euler's method. Collatz [9] suggests that the step factor be kept essentially constant during the course of the integration, leading to another method of controlling the step size.

6.7 Simultaneous Ordinary Differential Equations

Consider the solution of the following system of n simultaneous first-order ordinary differential equations in the dependent variables $y_1, y_2, \ldots, y_n$:

$$\begin{aligned}
\frac{dy_1}{dx} &= f_1(x,y_1,y_2,\ldots,y_n),\\
\frac{dy_2}{dx} &= f_2(x,y_1,y_2,\ldots,y_n),\\
&\vdots\\
\frac{dy_n}{dx} &= f_n(x,y_1,y_2,\ldots,y_n),
\end{aligned} \tag{6.77}$$

with initial conditions given at a common point (x_0), that is,

$$\begin{aligned}
y_1(x_0) &= y_{1,0},\\
y_2(x_0) &= y_{2,0},\\
&\vdots\\
y_n(x_0) &= y_{n,0}.
\end{aligned} \tag{6.78}$$

The solution of such a system is, at least in principle, no more difficult than the solution of a single first-order equation. The algorithm selected is applied to each of the n equations in parallel at each step.

Since a single high-order equation

$$\frac{d^m y}{dx^m} = F\left(x, y, \frac{dy}{dx}, \frac{d^2y}{dx^2}, \ldots, \frac{d^{m-1}y}{dx^{m-1}}\right), \qquad (6.79)$$

with appropriate initial conditions

$$y(x_0), \frac{dy(x_0)}{dx}, \ldots, \frac{d^{m-1}y(x_0)}{dx^{m-1}},$$

can always be rewritten as a system of first-order equations of the form of (6.77) [see, for example, (6.2), (6.3)], the numerical methods developed in this chapter may be applied indirectly to solve higher-order initial-value problems as well. Initial-value problems involving systems of equations of mixed order may also be reduced to the form of (6.77) in most cases. Depending on how the differential equations are coupled, some of the derivatives in (6.77) may be functions of other derivatives, that is,

$$\frac{dy_j}{dx} = f_j(x, y_1, y_2, \ldots, y_n, f_1, f_2 \ldots, f_{j-1}, f_{j+1}, \ldots f_n).$$

If the equations can be ordered so that, for all j, $dy_j/dx = f_j(x, y_1, y_2, \ldots, y_n, f_1, f_2, \ldots, f_{j-1})$, then the integration schemes of the preceding sections can always be applied. If such an ordering is impossible, then one may compute the particular (dy_j/dx) which cannot be so ordered from

$$\frac{dy_j}{dx} = f_j(x, y_1, y_2, \ldots, y_n, f_1, f_2, \ldots, f_{j-1}, f_{j+1}^*, \ldots, f_n^*)$$

where the starred derivatives must be known or assumed at x_0. Thereafter, one can usually use the most recently computed values for them.

Error analyses comparable to those of Section 6.4 are virtually impossible to implement for the higher-order Runge-Kutta schemes for systems of differential equations. The step-size control mechanisms and stability considerations outlined in the preceding section carry over to the multiple-equation case without appreciable change. In practice, we often solve the equations using different step sizes and observe the behavior of the solutions with regard to apparent convergence and stability.

EXAMPLE 6.3

FOURTH-ORDER RUNGE-KUTTA METHOD
TRANSIENT BEHAVIOR OF A RESONANT CIRCUIT

Problem Statement

Write a general-purpose function named RUNGE that solves a system of n first-order ordinary differential equations

$$\begin{aligned}\frac{dy_1}{dx} &= f_1(x, y_1, y_2, \ldots, y_n)\\ \frac{dy_2}{dx} &= f_2(x, y_1, y_2, \ldots, y_n)\\ &\vdots\\ \frac{dy_n}{dx} &= f_n(x, y_1, y_2, \ldots, y_n),\end{aligned} \tag{6.3.1}$$

using the fourth-order Runge-Kutta method with Kutta's constants (6.66).

Consider the circuit of Fig. 6.3.1 containing a capacitor of C farads, a resistor of R ohms, and an inductance of L henries. Assume that the capacitor is initially charged

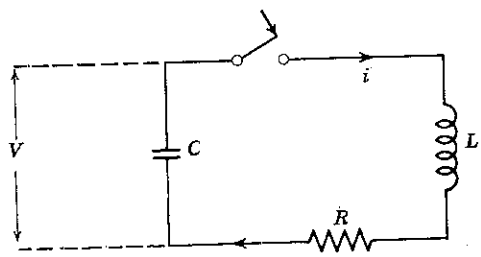


Figure 6.3.1 Electrical circuit

to a voltage V_0, and that suddenly the switch is closed at time $t = 0$. Show that the ordinary differential equation describing V, the damped oscillation of voltage across the capacitor is given as a function of time by

$$LC\frac{d^2V}{dt^2} + RC\frac{dV}{dt} + V = 0, \tag{6.3.2}$$

subject to the initial conditions

$$\begin{aligned}V(0) &= V_0,\\ \frac{dV}{dt}(0) &= 0.\end{aligned} \tag{6.3.3}$$

Let

$$\begin{aligned}\bar{\alpha}^2 &= \frac{1}{LC} - \frac{R^2}{4L^2},\\ \alpha &= \sqrt{|\bar{\alpha}^2|},\end{aligned} \tag{6.3.4}$$

and show that the following analytical solutions satisfy (6.3.2) with conditions (6.3.3):

For $\bar{\alpha}^2 > 0$ (the under-damped or oscillatory case):

$$V = \frac{V_0 \exp\left(-\frac{Rt}{2L}\right)\cos\left(\alpha t - \tan^{-1}\left(\frac{R}{2L\alpha}\right)\right)}{\alpha\sqrt{CL}} \tag{6.3.5}$$

For $\bar{\alpha}^2 = 0$ (the critically-damped case):

$$V = V_0 \exp\left(-\frac{Rt}{2L}\right)\left(1 + \frac{Rt}{2L}\right) \tag{6.3.6}$$

For $\bar{\alpha}^2 < 0$ (the over-damped case):

$$V = V_0 \exp\left(-\frac{Rt}{2L}\right)\left[\left(\frac{1}{2} + \frac{R}{4L\alpha}\right)\exp(\alpha t) + \left(\frac{1}{2} - \frac{R}{4L\alpha}\right)\exp(-\alpha t)\right] \tag{6.3.7}$$

Write a test program that calls on RUNGE to solve the differential equation (6.3.2) with initial conditions (6.3.3), and then compares the numerical solution with the value of the appropriate analytical solution. For test purposes, consider the following cases:

$V_0 = 100$ volts
$C = 2 \times 10^{-6}$ farads
$L = 0.5$ henries
$R = 0$, 100, 1000, and 1500 ohms.

Use step-size $h = 0.0001$ sec and tabulate the solutions at reasonable intervals in time during the first t_{max}=0.02 sec. Observe the effects of step-size variation by finding numerical solutions for the cases:

$V_0 = 100$ volts
$C = 2 \times 10^{-6}$ farads
$L = 0.5$ henries
$R = 100$ ohms
$h = 0.00001$, 0.0001, 0.001, 0.002, 0.005, 0.01.

Method of Solution

The fourth-order Runge-Kutta algorithm of (6.66) for the one-step integration of a single first-order equation (6.4) with one appropriate initial condition, $y_i \doteq y(x_i)$, may be written for a system of n first-order equations (6.3.1) with n initial conditions,

$$y_{ji} \doteq y_j(x_i), \qquad j = 1, 2, \ldots, n.$$

Here, y_{ji} is the solution of the jth equation in (6.3.1) at x_i. The initial conditions for the zeroth step, $y_{j0}, j = 1, 2, \ldots, n$, will usually be known exactly. Thereafter, the initial conditions for the ith step will be approximations to the true initial conditions, $y_j(x_i), j = 1, 2, \ldots, n$, since

they will result from applications of the Runge-Kutta method on the $(i-1)$th interval. For a system of n equations, the one-step integration across the ith interval may be described by:

$$y_{j\,i+1} = y_{ji} + h\phi_j = y_{ji} + h(k_{j1} + 2k_{j2} + 2k_{j3} + k_{j4})/6 \quad (6.3.8a)$$

$$k_{j1} = f_j(x_i, y_{1i}, y_{2i}, \ldots, y_{ni}), \quad (6.3.8b)$$

$$y^*_{ji} = y_{ji} + \tfrac{1}{2}hk_{j1}, \quad (6.3.8c)$$

$$k_{j2} = f_j(x_i + \tfrac{1}{2}h, y^*_{1i}, y^*_{2i}, \ldots, y^*_{ni}), \quad (6.3.8d)$$

$$\bar{y}_{ji} = y_{ji} + \tfrac{1}{2}hk_{j2}, \quad (6.3.8e)$$

$$k_{j3} = f_j(x_i + \tfrac{1}{2}h, \bar{y}_{1i}, \bar{y}_{2i}, \ldots, \bar{y}_{ni}), \quad (6.3.8f)$$

$$\bar{y}^*_{ji} = y_{ji} + hk_{j3}, \quad (6.3.8g)$$

$$k_{j4} = f_j(x_i + h, \bar{y}^*_{1i}, \bar{y}^*_{2i}, \ldots, \bar{y}^*_{ni}) \quad (6.3.8h)$$

The relationships in (6.3.8) are applied in parallel at each point in the algorithm for all n equations, that is, for $j = 1, 2, \ldots, n$.

Because the Runge-Kutta method is a one-step method, the subscript i throughout (6.3.8) is, in a sense, superfluous. In addition, because the k_{j1} are not needed after the y^*_{ji} are computed, and the y^*_{ji} in turn are not needed after the k_{j2} are computed, etc., it is possible to write a general-purpose Runge-Kutta function and calling program which require relatively few memory locations for retention of the many variables appearing in (6.3.8). The entire algorithm of (6.3.8) will be implemented in program form as outlined below.

Let four vectors of length at least n be denoted by the names Y, $\bar{Y}$, F, and ϕ. Before carrying out the Runge-Kutta integration for the ith step, the following variables must be initialized:

$x \leftarrow x_i$, value of the independent variable.

$h \leftarrow h$, step size for integration across the ith step, $x_{i+1} - x_i$.

$n \leftarrow n$, number of first-order differential equations.

$Y_j \leftarrow y_{ji}$, $j = 1, 2, \ldots, n$, solution values for the n equations at x_i

Then (6.3.8) may be described by a five-pass procedure.

Pass 1

1. Calculate the values $f_j, j = 1, 2, \ldots, n$, using the current x and Y_j values. These are equivalent to the values $k_{j1}, j = 1, 2, \ldots, n$, of (6.3.8b).

$$F_j \leftarrow f_j(x, Y_1, Y_2, \ldots, Y_n) = k_{j1} = f_j(x_i, y_{1i}, y_{2i}, \ldots, y_{ni}), \quad j = 1, 2, \ldots, n \quad (6.3.9)$$

Pass 2

2. Save the current values $Y_j, j = 1, 2, \ldots, n$, in another vector of equal length, $\bar{Y}$. This assigns the solution values at the beginning of the ith step to the vector $\bar{Y}$.

$$\bar{Y}_j \leftarrow Y_j = y_{ji}, \quad j = 1, 2, \ldots, n \quad (6.3.10)$$

3. Begin accumulation of the values $\phi_j, j = 1, 2, \ldots, n$, in (6.3.8a).

$$\phi_j \leftarrow F_j = k_{j1}, \quad j = 1, 2, \ldots, n \quad (6.3.11)$$

4. Compute the values $y^*_{ji}, j = 1, 2, \ldots, n$, in (6.3.8c) and assign them to elements of the vector Y.

$$Y_j \leftarrow \bar{Y}_j + \frac{h}{2}F_j = y^*_{ji} = y_{ji} + \frac{h}{2}k_{j1}, \quad j = 1, 2, \ldots, n \quad (6.3.12)$$

5. Increment x to the value needed in (6.3.8d).

$$x \leftarrow x + \frac{h}{2} = x_i + \frac{h}{2} \quad (6.3.13)$$

6. Calculate the values $f_j, j = 1, 2, \ldots, n$, using the current x and Y_j values. These are equivalent to the values $k_{j2}, j = 1, 2, \ldots, n$, of (6.3.8d).

$$F_j \leftarrow f_j(x, Y_1, Y_2, \ldots, Y_n) = k_{j2} = f_j\left(x_i + \frac{h}{2}, y^*_{1i}, y^*_{2i}, \ldots, y^*_{ni}\right), \quad j = 1, 2, \ldots, n \quad (6.3.14)$$

Pass 3

7. Add the contribution of k_{j2} to $\phi_j, j = 1, 2, \ldots, n$, in (6.3.8a).

$$\phi_j \leftarrow \phi_j + 2F_j = k_{j1} + 2k_{j2} \quad (6.3.15)$$

8. Compute the values $\bar{y}_{ji}, j = 1, 2, \ldots, n$, in (6.3.8e) and assign them to elements of the vector Y.

$$Y_j \leftarrow \bar{Y}_j + \frac{h}{2}F_j = \bar{y}_{ji} = y_{ji} + \frac{h}{2}k_{j2}, \quad j = 1, 2, \ldots, n \quad (6.3.16)$$

9. Calculate the values $f_j, j = 1, 2, \ldots, n$, using the current x and Y_j values. These are equivalent to the values $k_{j3}, j = 1, 2, \ldots, n$, of (6.3.8f).

$$F_j \leftarrow f_j(x, Y_1, Y_2, \ldots, Y_n) = k_{j3} = f_j\left(x_i + \frac{h}{2}, \bar{y}_{1i}, \bar{y}_{2i}, \ldots, \bar{y}_{ni}\right), \quad j = 1, 2, \ldots, n \quad (6.3.17)$$

Pass 4

10. Add the contribution of k_{j3} to $\phi_j, j = 1, 2, \ldots, n$, in (6.3.8a).

$$\phi_j \leftarrow \phi_j + 2F_j = k_{j1} + 2k_{j2} + 2k_{j3}, \quad j = 1, 2, \ldots, n \quad (6.3.18)$$

11. Compute the values $\bar{y}^*_{ji}, j = 1, 2, \ldots, n$, in (6.3.8g) and assign them to elements of the vector Y.

$$Y_j \leftarrow \bar{Y}_j + hF_j = \bar{y}^*_{ji} = y_{ji} + hk_{j3}, \quad j = 1, 2, \ldots, n \quad (6.3.19)$$

12. Increment x to the value needed in (6.3.8h).

$$x \leftarrow x + \frac{h}{2} = x_i + h = x_{i+1}. \qquad (6.3.20)$$

13. Calculate the values $f_j, j = 1, 2, \ldots, n$, using the current x and Y_j values. These are equivalent to the values $k_{j4}, j = 1, 2, \ldots, n$, of (6.3.8h)

$$F_j \leftarrow f_j(x, Y_1, Y_2, \ldots, Y_n) = k_{j4}$$
$$= f_j(x_i + h, \bar{y}^*_{1i}, \bar{y}^*_{2i}, \ldots, \bar{y}^*_{ni}), \qquad j = 1, 2, \ldots, n. \qquad (6.3.21)$$

Pass 5

14. Complete the evaluation of $\phi_j, j = 1, 2, \ldots, n$, in (6.3.8a).

$$\phi_j \leftarrow (\phi_j + F_j)/6 = (k_{j1} + 2k_{j2} + 2k_{j3} + k_{j4})/6. \qquad (6.3.22)$$

15. Compute the values $y_{j,i+1}, j = 1, 2, \ldots, n$, in (6.3.8a) and assign them to elements of the Y vector.

$$Y_j \leftarrow \bar{Y}_j + h\phi_j = y_{ji} + h\phi_j, \qquad j = 1, 2, \ldots, n. \qquad (6.3.23)$$

As a consequence of the procedure described by (6.3.9) to (6.3.23), h and n remain unchanged, and x and the vector Y contain:

$x = x_{i+1}$, value of the independent variable.

$Y_j = y_{j,i+1}$, $j = 1, 2, \ldots, n$, solution values for the n equations at x_{i+1}.

All initial values have been assigned for the next integration step, from x_{i+1} to x_{i+2}. The step size, h, may be changed if desired (see Section 6.6). The five-pass procedure of (6.3.9) to (6.3.23) may be repeated for the next step.

Note that the parts of the procedure given by (6.3.9), (6.3.14), (6.3.17), and (6.3.21) are identical,

$$F_j \leftarrow f_j(x, Y_1, Y_2, \ldots, Y_n), \qquad j = 1, 2, \ldots, n, \qquad (6.3.24)$$

and are the only ones that refer directly to the n differential equations, $f_j, j = 1, 2, \ldots, n$, of (6.3.1). Therefore, it is possible to write a general-purpose function called RUNGE that implements all parts of the procedure outlined, *except* for initialization of n, x, h, and $Y_j, j = 1, 2, \ldots, n$, and steps (6.3.9), (6.3.14), (6.3.17), and (6.3.21). The calling program will then contain all information about the specific system of differential equations, and be responsible for printing results, changing the step size, terminating the integration process, and evaluating the F_j when needed.

The five passes of the algorithm can be handled by five different calls upon RUNGE, as shown schematically in Fig. 6.3.2. Let a step counter, m, preset to 0, be maintained by the function RUNGE, and let the value returned by RUNGE signal the calling program to indicate whether the $F_j, j = 1, 2, \ldots, n$, of (6.3.24) are to be computed (following the first four passes) or not (when integration across one step is completed following the fifth pass). Let the value returned be 1 when the F_j are to be evaluated and 0 when one complete integration step is completed.

The main program used to test the function RUNGE solves the second-order ordinary differential equation (6.3.2) subject to initial conditions (6.3.3). Since the charge, q, on a capacitor is related to the capacitance, C, and voltage, V, by

$$q = CV, \qquad (6.3.25)$$

the current, i, into the capacitor is given by

$$i = -\frac{dq}{dt} = -C\frac{dV}{dt}. \qquad (6.3.26)$$

The voltage, V, across a resistor and inductance in series is given by

$$V = L\frac{di}{dt} + Ri. \qquad (6.3.27)$$

Then, from (6.3.26) and (6.3.27), the voltage across the capacitor as a function of time is given by (6.3.2),

$$\frac{d^2V}{dt^2} = -\frac{R}{L}\frac{dV}{dt} - \frac{1}{LC}V. \qquad (6.3.28)$$

The initial conditions are

$$V(0) = V_0,$$
$$\frac{dV}{dt}(0) = 0. \qquad (6.3.29)$$

Differentiation of the three proposed solutions, (6.3.5), (6.3.6), and (6.3.7), and substitution into (6.3.28), shows that, for the given value of $\bar{\alpha}^2$, each satisfies (6.3.28) and initial conditions (6.3.29).

The second-order equation (6.3.28) must be rewritten as a system of two first-order equations. Let

$$Y_1 = V,$$
$$Y_2 = \frac{dV}{dt}, \qquad (6.3.30)$$
$$x = t.$$

Then

$$F_1 = \frac{dY_1}{dx} = \frac{dV}{dt} = Y_2,$$
$$F_2 = \frac{dY_2}{dx} = \frac{d^2V}{dt^2} = -\frac{R}{L}\frac{dV}{dt} - \frac{1}{LC}V = -\frac{R}{L}Y_2 - \frac{1}{LC}Y_1. \qquad (6.3.31)$$

The initial conditions of (6.3.29) are

$$Y_{1,0} = V_0,$$
$$Y_{2,0} = 0. \qquad (6.3.32)$$

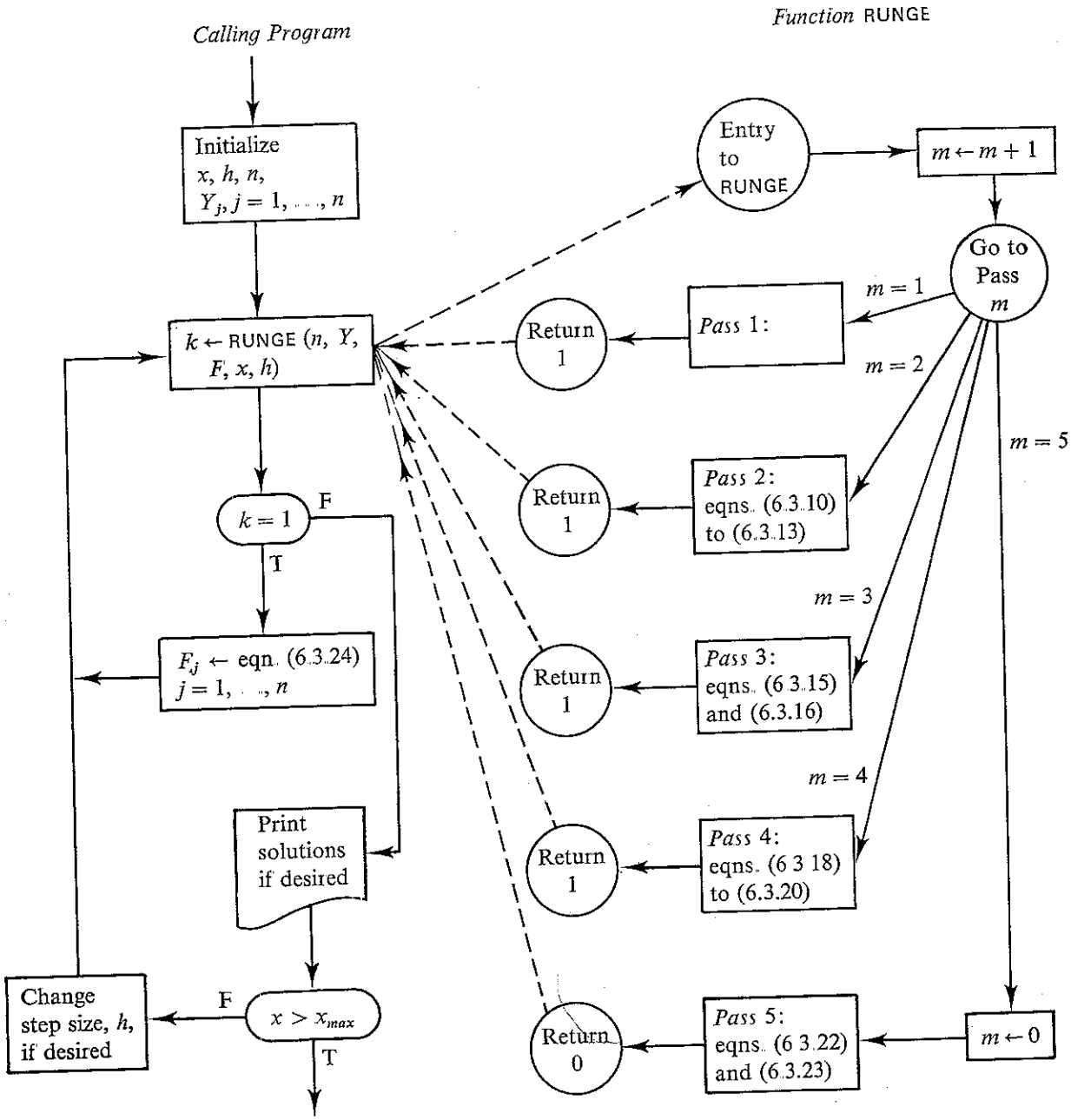
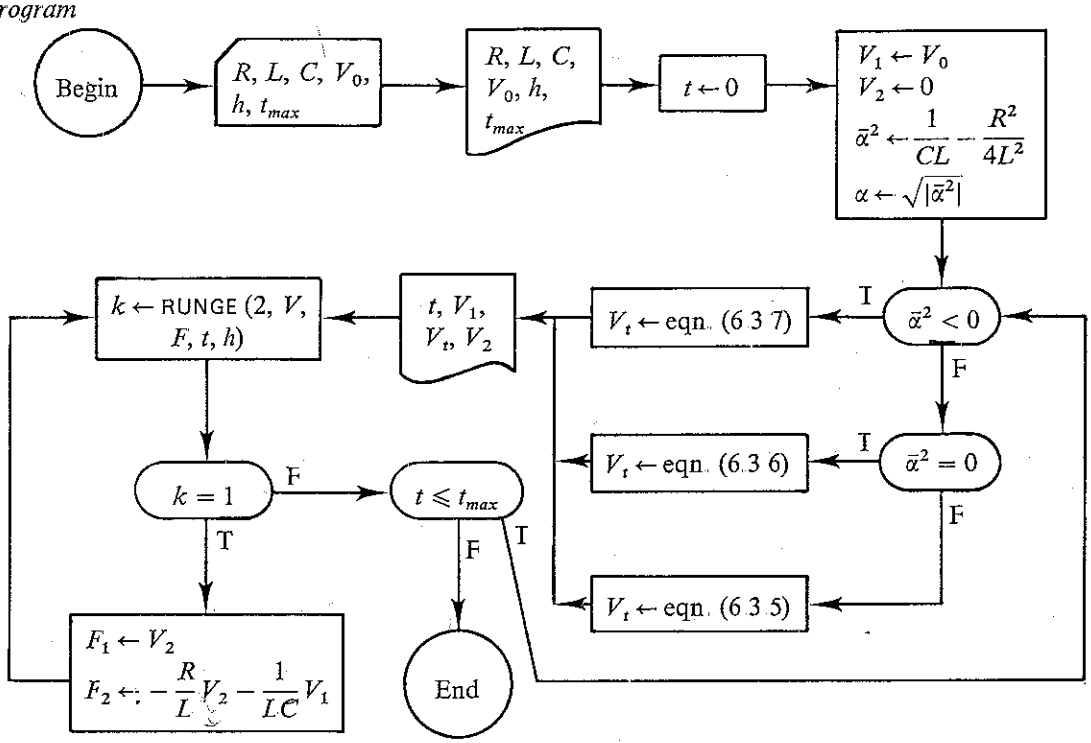


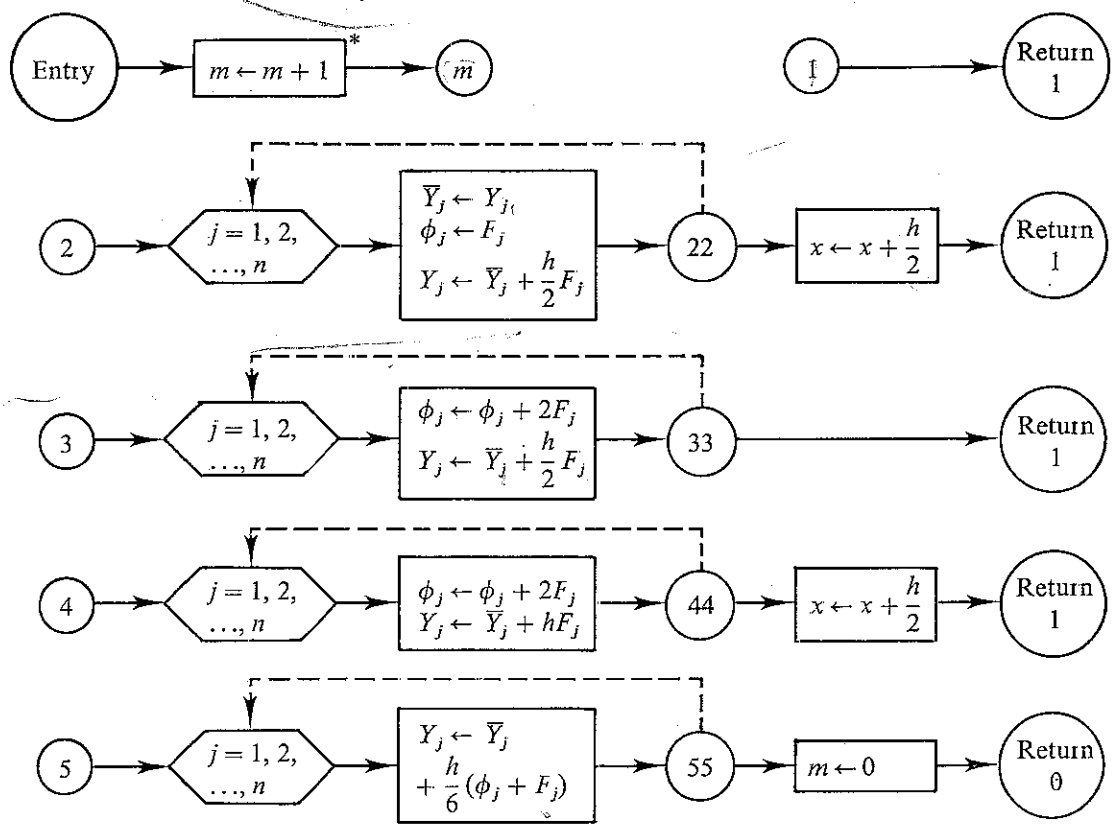
Figure 6.3.2 Communication between calling program and function RUNGE.

Flow Diagram

Main Program



Function RUNGE (Dummy arguments: n, Y, F, x, h; calling arguments: 2, V, F, t, h)



* m is assumed to be preset to zero, before RUNGE is called for the first time

FORTRAN *Implementation*

List of Principal Variables

Program Symbol	*Definition*
(Main)	
ALPHA, ALPHSQ	α and $\bar{\alpha}^2$, respectively
C	Capacitance, C, farads.
F	Vector of derivative approximations, $\bar{F}_j$ (see (6.3.24)).
H	Step size, h.
HL	Inductance, L, henries.
ICOUNT	Step counter.
IFREQ	Frequency of intermediate printout. Solutions are printed at $t = 0$ and after every IFREQ steps thereafter.
IFPLOT	Variable to control plotting of solutions on printer. If IFPLOT = 1, a plot is prepared by calling on PLOT1, PLOT2, etc.; if IFPLOT $\neq$ 1, no plot is prepared.
IMAGE	Large vector used to store image of plot before printing.
K	Value returned by RUNGE. If K = 1, the elements of F are to be calculated, if K $\neq$ 1, one step of the integration is completed.
PLOT1, PLOT2, PLOT3, PLOT4	Subroutines used for preparing on-line graph of V against t. See Examples 7.1 and 8.5 for further details.
R	Resistance, R, ohms.
RO2L	$R/(2L)$.
T	Time, t, sec.
TMAX	Upper limit of integration, t_{max}, sec.
TRUV	True voltage at time t, V_t, calculated from analytical solutions of (6.3.2) subject to initial conditions (6.3.3).
V	Vector of solutions of (6.3.2) computed by the function RUNGE. V(1) = V, volts. V(2) = dV/dt, volts/sec.
VZERO	Initial voltage across capacitor, V_0, volts.
(Function RUNGE)	
J	Subscript, j
M	Pass counter for the Runge-Kutta algorithm, m. Must be preset to 0 before first call upon RUNGE.
N	Number of equations, n.
PHI	Vector of values, ϕ_j.
SAVEY	Vector of initial conditions, $\bar{Y}_j$ (see (6.3.10)).
X	Independent variable, x.
Y	Vector of dependent variable (solution) values, Y_j.

Program Listing

Main Program

```
C       APPLIED NUMERICAL METHODS, EXAMPLE 6.3
C       ELECTRICAL TRANSIENTS USING THE RUNGE-KUTTA METHOD
C
C       THIS TEST PROGRAM CALLS ON THE FUNCTION RUNGE TO SOLVE
C       THE DIFFERENTIAL EQUATION V'' = -R*V'/HL - V/(HL*C) SUBJECT
C       TO THE INITIAL CONDITIONS V(0.0) = VZERO AND V'(0.0) = 0.0.
C       V(1) AND V(2) ARE THE VALUES OF V AND V' RESPECTIVELY.  F(1)
C       AND F(2) ARE THE DERIVATIVES OF V(1) AND V(2) RESPECTIVELY.
C       R, HL, AND C ARE THE RESISTANCE, INDUCTANCE, AND CAPACITANCE
C       (IN OHMS, HENRIES, AND FARADS) OF A CIRCUIT (SEE FIGURE)
C       IN WHICH THE CAPACITOR IS CHARGED TO A VOLTAGE VZERO AT TIME
C       ZERO.  T (TIME IN SECONDS) IS THE INDEPENDENT VARIABLE, H IS
C       THE STEPSIZE, AND TMAX THE UPPER INTEGRATION LIMIT USED IN
C       THE RUNGE-KUTTA INTEGRATION.  THE EQUATION WITH THE GIVEN
C       INITIAL CONDITIONS DESCRIBES THE DAMPED OSCILLATION OF VOLTAGE
C       ACROSS THE CAPACITOR UPON CLOSURE OF THE CIRCUIT.  TRUV IS
C       THE ANALYTICAL SOLUTION WHICH APPLIES FOR THE UNDER-DAMPED,
C       CRITICALLY DAMPED OR OVER-DAMPED CASE DEPENDING ON THE VALUE
C       OF ALPHSQ=1/(C*HL)-R*R/(4*HL*HL) (POSITIVE, ZERO, AND NEGATIVE
C       VALUE RESPECTIVELY).  K IS THE VALUE RETURNED BY THE FUNCTION
C       RUNGE.  IT EQUALS  1  WHEN THE DERIVATIVES ARE TO BE
C       CALCULATED AND  0  WHEN INTEGRATION ACROSS ONE STEP IS
C       COMPLETE.  ICOUNT IS A STEP COUNTER.  RESULTS ARE PRINTED
C       AFTER EVERY IFREQ INTEGRATION STEPS.  IF IFPLOT = 1, THE
C       RESULTS ARE ALSO PLOTTED USING THE LIBRARY SUBROUTINES
C       PLOT1, PLOT2, PLOT3, AND PLOT4.
C
      IMPLICIT  REAL*8(A-H, O-Z)
      INTEGER  RUNGE
      DIMENSION F(2), V(2), IMAGE(1500)
C
C     ..... READ AND PRINT DATA .....
    1 READ (5,100) R,HL,C,VZERO,H,TMAX,IFREQ,IFPLOT
      WRITE (6,200) R,HL,C,VZERO,H,TMAX,IFREQ,IFPLOT
C
C     ..... INITIALIZE T, V(1), V(2), AND ICOUNT .....
      T = 0.0
      V(1) = VZERO
      V(2) = 0.0
      ICOUNT = 0
C
C     ..... COMPUTE ALPHSQ AND ALPHA, PRINT HEADINGS .....
      ALPHSQ = 1./(C*HL) - R*R/(4.0*HL*HL)
      ALPHA = DSQRT(DABS(ALPHSQ))
      RO2L = R/(2.*HL)
      IF ( IFPLOT.NE.1 )  GO TO 3
         CALL PLOT1( 0, 5, 11, 6, 19 )
         CALL PLOT2( IMAGE,TMAX,0.,DABS(VZERO),-DABS(VZERO) )
    3 WRITE (6,201)
C
C     ..... IS CIRCUIT OVER-, CRITICALLY-, OR UNDER-DAMPED .....
C     ..... COMPUTE ANALYTICAL SOLUTION, PRINT AND PLOT .....
    4 IF (ALPHSQ) 5, 6, 7
    5    TRUV = VZERO*DEXP(-RO2L*T)*((1.+RO2L/ALPHA)*DEXP(ALPHA*T)
     1          +(1.-RO2L/ALPHA)*DEXP(-ALPHA*T))/2.0
         GO TO 8
    6    TRUV = VZERO*DEXP(-RO2L*T)*(1. + RO2L*T)
         GO TO 8
    7    TRUV = VZERO*DEXP(-RO2L*T)*DCOS(ALPHA*T-DATAN(RO2L/ALPHA))
     1          /(ALPHA*DSQRT(C*HL))
    8 IF ( IFPLOT.NE.1 )  GO TO 10
         CALL PLOT3( 1H*, T, V(1), 1, 8 )
   10 WRITE (6,202) T,V(1),TRUV,V(2)
C
C     ..... CALL ON THE FOURTH-ORDER RUNGE-KUTTA FUNCTION .....
   11 K = RUNGE( 2, V, F, T, H )
```

Program Listing (*Continued*)

```
C
C       ..... WHENEVER K=1, COMPUTE DERIVATIVE VALUES .....
        IF ( K.NE.1 )   GO TO 13
           F(1) = V(2)
           F(2) = -R*V(2)/HL -V(1)/(HL*C)
           GO TO 11
C
C       ..... IF T EXCEEDS TMAX, TERMINATE INTEGRATION .....
   13   IF ( T.LE.TMAX )   GO TO 16
           IF ( IFPLOT.NE.1 )   GO TO 1
              WRITE (6,203)
              CALL PLOT4( 7, 7HVOLTAGE )
              WRITE (6,204)
              GO TO 1
   16   ICOUNT = ICOUNT + 1
C
C       ..... PRINT RESULTS OR CALL DIRECTLY ON RUNGE .....
        IF ( ICOUNT.NE.IFREQ )   GO TO 11
        ICOUNT = 0
        GO TO 4
C
C       ..... FORMATS FOR INPUT AND OUTPUT STATEMENTS .....
  100   FORMAT ( 9X, F7.2,14X,F6.2,14X,E6.1/ 10X,F6.2,14X,F7.5,13X,F6.3/
     1    10X,I4,16X,I1 )
  200   FORMAT ( 10H1R      = ,F14.5/ 10H HL     = ,F14.5/10H C      = ,
     1    E14.1  /10H VZERO  = ,F14.5/ 10H H      = ,F14.5/10H TMAX   = ,
     2    F14.5/ 10H IFREQ  = ,I8  / 10H IFPLOT = , I8 )
  201   FORMAT ( 64H0       T              CALC. V          TRUE  V
     1  CALC. V' / 1H  )
  202   FORMAT ( 1H , F10.5, 2F18.8, F18.5 )
  203   FORMAT ( 1H1 )
  204   FORMAT ( 1H0, 54X, 16HTIME - (SECONDS) )
C
        END
```

Function RUNGE

```
        FUNCTION RUNGE( N, Y, F, X, H )
C
C          THE FUNCTION RUNGE EMPLOYS THE FOURTH-ORDER RUNGE-KUTTA METHOD
C          WITH KUTTA'S COEFFICIENTS TO INTEGRATE A SYSTEM OF N SIMULTAN-
C          EOUS FIRST ORDER ORDINARY DIFFERENTIAL EQUATIONS F(J)=DY(J)/DX,
C          (J=1,2,...,N), ACROSS ONE STEP OF LENGTH H IN THE INDEPENDENT
C          VARIABLE X, SUBJECT TO INITIAL CONDITIONS Y(J), (J=1,2,...,N).
C          EACH F(J), THE DERIVATIVE OF Y(J), MUST BE COMPUTED FOUR TIMES
C          PER INTEGRATION STEP BY THE CALLING PROGRAM.  THE FUNCTION MUST
C          BE CALLED FIVE TIMES PER STEP (PASS(1)...PASS(5)) SO THAT THE
C          INDEPENDENT VARIABLE VALUE (X) AND THE SOLUTION VALUES
C          (Y(1)...Y(N)) CAN BE UPDATED USING THE RUNGE-KUTTA ALGORITHM.
C          M IS THE PASS COUNTER.  RUNGE RETURNS AS ITS VALUE 1 TO
C          SIGNAL THAT ALL DERIVATIVES (THE F(J)) BE EVALUATED OR 0 TO
C          SIGNAL THAT THE INTEGRATION PROCESS FOR THE CURRENT STEP IS
C          FINISHED.  SAVEY(J) IS USED TO SAVE THE INITIAL VALUE OF Y(J)
C          AND PHI(J) IS THE INCREMENT FUNCTION FOR THE J(TH) EQUATION.
C          AS WRITTEN, N MAY BE NO LARGER THAN 50.
C
        IMPLICIT  REAL*8(A-H, O-Z)
        REAL*8  Y, F, X, H
        INTEGER  RUNGE
        DIMENSION PHI(50), SAVEY(50), Y(N), F(N)
        DATA  M/0/
C
        M = M + 1
        GO TO (1,2,3,4,5), M
C
C       ..... PASS 1 .....
    1   RUNGE = 1
        RETURN
```

Program Listing (*Continued*)

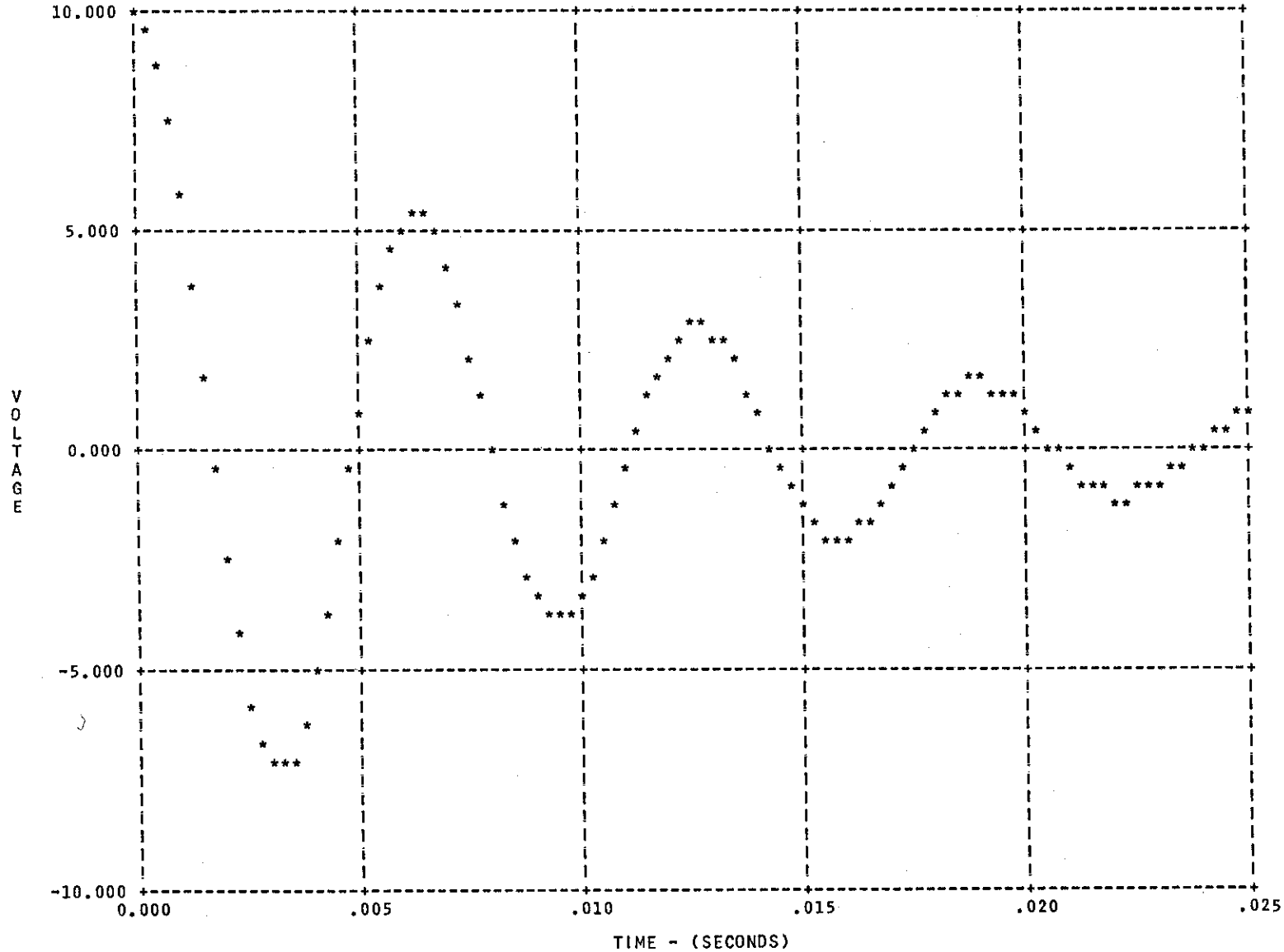
```
C
C       ..... PASS 2 .....
    2   DO 22  J = 1, N
        SAVEY(J) = Y(J)
        PHI(J) = F(J)
   22   Y(J) = SAVEY(J) + 0.5*H*F(J)
        X = X + 0.5*H
        RUNGE = 1
        RETURN
C
C       ..... PASS 3 .....
    3   DO 33  J = 1, N
        PHI(J) = PHI(J) + 2.0*F(J)
   33   Y(J) = SAVEY(J) + 0.5*H*F(J)
        RUNGE = 1
        RETURN
C
C       ..... PASS 4 .....
    4   DO 44  J = 1, N
        PHI(J) = PHI(J) + 2.0*F(J)
   44   Y(J) = SAVEY(J) + H*F(J)
        X = X + 0.5*H
        RUNGE = 1
        RETURN
C
C       ..... PASS 5 .....
    5   DO 55  J = 1, N
   55   Y(J) = SAVEY(J) + (PHI(J) + F(J))*H/6.0
        M = 0
        RUNGE = 0
        RETURN
C
        END
```

Data

```
R       =  100.00     HL      =    0.50     C       =  2.0E-6
VZERO   =   10.00     H       =  0.00001    TMAX    =   0.025
IFREQ   =      25     IFPLOT  =  1
R       =    0.00     HL      =    0.50     C       =  2.0E-6
VZERO   =   10.00     H       =  0.00010    TMAX    =   0.020
IFREQ   =      10     IFPLOT  =  0
R       = 1000.00     HL      =    0.50     C       =  2.0E-6
VZERO   =   10.00     H       =  0.00010    TMAX    =   0.020
IFREQ   =      10     IFPLOT  =  0
R       = 1500.00     HL      =    0.50     C       =  2.0E-6
VZERO   =   10.00     H       =  0.00010    TMAX    =   0.020
IFREQ   =      10     IFPLOT  =  0
R       =  100.00     HL      =    0.50     C       =  2.0E-6
VZERO   =   10.00     H       =  0.00001    TMAX    =   0.020
IFREQ   =     100     IFPLOT  =  0
R       =  100.00     HL      =    0.50     C       =  2.0E-6
VZERO   =   10.00     H       =  0.00010    TMAX    =   0.020
IFREQ   =      10     IFPLOT  =  0
R       =  100.00     HL      =    0.50     C       =  2.0E-6
VZERO   =   10.00     H       =  0.00100    TMAX    =   0.020
IFREQ   =       1     IFPLOT  =  0
R       =  100.00     HL      =    0.50     C       =  2.0E-6
VZERO   =   10.00     H       =  0.00200    TMAX    =   0.020
IFREQ   =       1     IFPLOT  =  0
R       =  100.00     HL      =    0.50     C       =  2.0E-6
VZERO   =   10.00     H       =  0.00500    TMAX    =   0.020
IFREQ   =       1     IFPLOT  =  0
R       =  100.00     HL      =    0.50     C       =  2.0E-6
VZERO   =   10.00     H       =  0.01000    TMAX    =   0.020
IFREQ   =       1     IFPLOT  =  0
```

Computer Output

Plotted Results for 1st Data Set



Computer Output (*Continued*)

Results for the 2nd Data Set

```
R       =       0.0
HL      =       0.50000
C       =       0.2D-05
VZERO   =      10.00000
H       =       0.00010
TMAX    =       0.02000
IFREQ   =      10
IFPLOT  =       0
```

T	CALC. V	TRUE V	CALC. V'
0.0	10.00000000	10.00000000	0.0
0.00100	5.40302967	5.40302306	-8414.70478
0.00200	-4.16145269	-4.16146837	-9092.97992
0.00300	-9.89991939	-9.89992497	-1411.22445
0.00400	-6.53645953	-6.53643621	7568.00114
0.00500	2.83658106	2.83662185	9589.25120
0.00600	9.60168495	9.60170287	2794.20166
0.00700	7.53905707	7.53902254	-6569.81898
0.00800	-1.45493381	-1.45500034	-9893.58664
0.00900	-9.11126613	-9.11130262	-4121.25037
0.01000	-8.39075464	-8.39071529	5440.13766
0.01100	0.04416561	0.04425698	9999.89484
0.01200	8.43847910	8.43853959	5365.80880
0.01300	9.07450499	9.07446781	-4201.56862
0.01400	1.36748601	1.36737218	-9906.04804
0.01500	-7.59679023	-7.59687913	-6502.96626
0.01600	-9.57662243	-9.57659480	2878.90274
0.01700	-2.75176585	-2.75163338	9613.92474
0.01800	6.60304659	6.60316708	7509.96179
0.01900	9.88705680	9.88704618	-1498.61414
0.02000	4.08096657	4.08082062	-9129.37207

Results for the 3rd Data Set

```
R       =    1000.00000
HL      =       0.50000
C       =       0.2D-05
VZERO   =      10.00000
H       =       0.00010
TMAX    =       0.02000
IFREQ   =      10
IFPLOT  =       0
```

T	CALC. V	TRUE V	CALC. V'
0.0	10.00000000	10.00000021	0.0
0.00100	7.35757855	7.35758878	-3678.78080
0.00200	4.06005339	4.06005848	-2706.69810
0.00300	1.99148127	1.99148272	-1493.60923
0.00400	0.91578189	0.91578194	-732.62484
0.00500	0.40427710	0.40427682	-336.89732
0.00600	0.17351291	0.17351265	-148.72525
0.00700	0.07295073	0.07295056	-63.83185
0.00800	0.03019173	0.03019164	-26.83708
0.00900	0.01234103	0.01234098	-11.10692
0.01000	0.00499402	0.00499399	-4.54001
0.01100	0.00200422	0.00200420	-1.83720
0.01200	0.00079875	0.00079875	-0.73731
0.01300	0.00031645	0.00031645	-0.29384
0.01400	0.00012473	0.00012473	-0.11641
0.01500	0.00004894	0.00004894	-0.04589
0.01600	0.00001913	0.00001913	-0.01801
0.01700	0.00000745	0.00000745	-0.00704
0.01800	0.00000289	0.00000289	-0.00274
0.01900	0.00000112	0.00000112	-0.00106
0.02000	0.00000043	0.00000043	-0.00041

Results for the 4th Data Set

```
R       =    1500.00000
HL      =       0.50000
C       =       0.2D-05
VZERO   =      10.00000
H       =       0.00010
TMAX    =       0.02000
IFREQ   =      10
IFPLOT  =       0
```

T	CALC. V	TRUE V	CALC. V'
0.0	10.00000000	10.00000000	0.0
0.00100	7.86644015	7.86645599	-2726.04778
0.00200	5.44495442	5.44495666	-2059.45739
0.00300	3.72182288	3.72182306	-1420.12665
0.00400	2.54061638	2.54061634	-970.32090
0.00500	1.73404656	1.73404650	-662.33895
0.00600	1.18352059	1.18352054	-452.06406
0.00700	0.80777456	0.80777452	-308.54239
0.00800	0.55132090	0.55132087	-210.58584
0.00900	0.37628658	0.37628655	-143.72868
0.01000	0.25682246	0.25682244	-98.09745
0.01100	0.17528602	0.17528600	-66.95330
0.01200	0.11963591	0.11963590	-45.69685
0.01300	0.08165369	0.08165368	-31.18893
0.01400	0.05573013	0.05573013	-21.28702
0.01500	0.03803683	0.03803683	-14.52878
0.01600	0.02596083	0.02596083	-9.91616
0.01700	0.01771874	0.01771874	-6.76796
0.01800	0.01209337	0.01209336	-4.61925
0.01900	0.00825394	0.00825394	-3.15273
0.02000	0.00563347	0.00563347	-2.15179

Results for the 5th Data Set

```
R       =     100.00000
HL      =       0.50000
C       =       0.2D-05
VZERO   =      10.00000
H       =       0.00001
TMAX    =       0.02000
IFREQ   =     100
IFPLOT  =       0
```

T	CALC. V	TRUE V	CALC. V'
0.0	10.00000000	10.00000000	0.0
0.00100	5.68971891	5.68971891	-7627.57679
0.00200	-2.58070263	-2.58070263	-7516.15502
0.00300	-7.20135221	-7.20135221	-1161.42920
0.00400	-4.98325602	-4.98325602	5009.24394
0.00500	0.98550667	0.98550668	5886.96794
0.00600	5.05105559	5.05105559	1699.75050
0.00700	4.17040640	4.17040639	-3144.92073
0.00800	-0.02596842	-0.02596843	-4490.61851
0.00900	-3.44002905	-3.44002905	-1850.17734
0.01000	-3.36851681	-3.36851681	1853.45707
0.01100	-0.50285277	-0.50285276	3341.17931
0.01200	2.26240109	2.26240109	1774.88989
0.01300	2.64105351	2.64105351	-986.56352
0.01400	0.75017631	0.75017631	-2425.30898
0.01500	-1.42309381	-1.42309381	-1582.15077
0.01600	-2.01649802	-2.01649802	426.63595
0.01700	-0.82191085	-0.82191085	1715.75924
0.01800	0.84106437	0.84106437	1341.39588
0.01900	1.50170200	1.50170199	-82.94375
0.02000	0.79116024	0.79116024	-1179.97420

Computer Output (*Continued*)

Results for the 6th Data Set

```
R       =      100.00000
HL      =        0.50000
C       =        0.2D-05
VZERO   =       10.00000
H       =        0.00010
TMAX    =        0.02000
IFREQ   =       10
IFPLOT  =        0
```

T	CALC. V	TRUE V	CALC. V'
0.0	10.00000000	10.00000000	0.0
0.00100	5.68972630	5.68971891	-7627.57558
0.00200	-2.58069239	-2.58070263	-7516.16529
0.00300	-7.20135522	-7.20135221	-1161.44732
0.00400	-4.98327674	-4.98325602	5009.23693
0.00500	0.98548526	0.98550668	5886.98405
0.00600	5.05105572	5.05105559	1699.77815
0.00700	4.17043109	4.17040639	-3144.90741
0.00800	-0.02594075	-0.02596843	-4490.63369
0.00900	-3.44002437	-3.44002905	-1850.20820
0.01000	-3.36854000	-3.36851681	1853.43882
0.01100	-0.50288259	-0.50285276	3341.19041
0.01200	2.26239181	2.26240109	1774.91974
0.01300	2.64107247	2.64105351	-986.54239
0.01400	0.75020528	0.75017631	-2425.31507
0.01500	-1.42308113	-1.42309381	-1582.17716
0.01600	-2.01651180	-2.01649802	426.61390
0.01700	-0.82193704	-0.82191085	1715.76066
0.01800	0.84104973	0.84106437	1341.41766
0.01900	1.50171074	1.50170199	-82.92239
0.02000	0.79118262	0.79116024	-1179.97185

Results for the 7th Data Set

```
R       =      100.00000
HL      =        0.50000
C       =        0.2D-05
VZERO   =       10.00000
H       =        0.00100
TMAX    =        0.02000
IFREQ   =        1
IFPLOT  =        0
```

T	CALC. V	TRUE V	CALC. V'
0.0	10.00000000	10.00000000	0.0
0.00100	5.73333333	5.68971891	-7563.33333
0.00200	-2.43329000	-2.58070263	-7528.54200
0.00300	-7.08917353	-7.20135221	-1337.16829
0.00400	-5.07580444	-4.98325602	4797.40409
0.00500	0.71830874	0.98550668	5863.82444
0.00600	4.84683623	5.05105559	1931.64399
0.00700	4.23981951	4.17040639	-2850.54126
0.00800	0.27487048	-0.02596843	-4409.83527
0.00900	-3.17771301	-3.44002905	-2069.13818
0.01000	-3.38684697	-3.36851681	1530.09602
0.01100	-0.78452964	-0.50285276	3207.38778
0.01200	1.97605730	2.26240109	1947.09739
0.01300	2.60559417	2.64105351	-672.75310
0.01400	0.98504840	0.75017631	-2254.64438
0.01500	-1.14050162	-1.42309381	-1696.63518
0.01600	-1.93710934	-2.01649802	146.50624
0.01700	-0.99980180	-0.82191085	1526.93576
0.01800	0.58165271	0.84106437	1400.65212
0.01900	1.39284077	1.50170199	151.24523
0.02000	0.91295386	0.79116024	-989.61633

Results for the 8th Data Set

```
R       =      100.00000
HL      =        0.50000
C       =        0.2D-05
VZERO   =       10.00000
H       =        0.00200
TMAX    =        0.02000
IFREQ   =        1
IFPLOT  =        0
```

T	CALC. V	TRUE V	CALC. V'
0.0	10.00000000	10.00000000	0.0
0.00200	-0.93333333	-2.58070263	-5813.33333
0.00400	-3.29237333	-4.98325602	1761.05244
0.00600	1.33104667	5.05105559	1544.84977
0.00800	0.77384165	-0.02596843	-1097.58231
0.01000	-0.71028640	-3.36851681	-219.80669
0.01200	-0.06148756	2.26240109	458.98464
0.01400	0.27256191	0.75017631	-60.45841
0.01600	-0.06058560	-2.01649802	-145.77724
0.01800	-0.07909051	0.84106437	65.77534
0.02000	0.04561918	0.79116024	32.19144

Results for the 9th Data Set

```
R       =      100.00000
HL      =        0.50000
C       =        0.2D-05
VZERO   =       10.00000
H       =        0.00500
TMAX    =        0.02000
IFREQ   =        1
IFPLOT  =        0
```

T	CALC. V	TRUE V	CALC. V'
0.0	10.00000000	10.00000000	0.0
0.00500	176.66666667	0.98550668	72916.66667
0.01000	2589.42708333	-3.36851681	2682725.69444
0.01500	26185.00361690	-1.42309381	70188368.05556
0.02000	-49188.45317322	0.79116024	1533284857.10237

Computer Output (*Continued*)

Results for the 10th Data Set

```
R       =       100.00000
HL      =         0.50000
C       =         0.2D-05
VZERO   =        10.00000
H       =         0.01000
TMAX    =         0.02000
IFREQ   =         1
IFPLOT  =         0

        T            CALC. V          TRUE  V          CALC. V'

   0.0            10.00000000      10.00000000          0.0
   0.01000      3843.33333333      -3.36851681    -33333.33333
   0.02000   1477009.99999999       0.79116024  -25600000.00000
```

Discussion of Results

All calculations have been made using double-precision arithmetic. Only the plotted output is shown for the under-damped solution resulting from the input parameters of the 1st data set. Results for the 2nd, 3rd, and 4th data sets correspond to an undamped ($R = 0$), critically-damped ($R = 1000$), and over-damped solution, respectively. For the step size used, $h = 0.0001$, the numerical solutions agree with the appropriate analytical solution of (6.3.5), (6.3.6), and (6.3.7) to five or six significant figures.

Results for data sets 5 through 10 show the influence of the step-size h, on the numerical solution for an under-damped case (see plotted output for data set 1). Table 6.3.1 shows the calculated solution, V, the true solution, V_t, and the error, $V - V_t$, at time $t = 0.02$ sec. Because double-precision arithmetic has been used throughout, and the total number of integration steps is no larger than 2000 for any case, the error may be attributed solely to truncation effects; any round-off error is negligible in comparison.

Table 6.3.1 Comparison of Numerical and Analytical Solutions at Time $t = 0.02$ sec.

Step size, h (Sec)	Numerical Solution, V (Volts)	Analytical Solution, V_t (Volts)	Error $V - V_t$ (Volts)
0.00001	0.79116024	0.79116024	0.00000000
0.0001	0.79118262	0.79116024	0.00002238
0.001	0.91295386	0.79116024	0.12179362
0.002	0.04561918	0.79116024	−0.74554106
0.005	−49188.45317322	0.79116024	−49189.24433346
0.01	1477009.99999999	0.79116024	1477009.20883975

For small step sizes, the solutions computed by the Runge-Kutta fourth-order method are extremely accurate. The truncation error increases with increasing step size, until the solution eventually "blows up" and bears no resemblance to the true solution. This is not surprising, in view of the fact that the period of oscillation for the solution is $2\pi/\alpha \doteq 0.00628$ sec. As the step length approaches the period of oscillation, virtually all local information about the curvature of the solution function, V_t, is lost.