Chapter 10

Numerical Procedures

10.1 Differences and Sums

In this chapter we shall describe the notation and procedures involved in some of the numerical work that occurs in celestial mechanics. No attempt has been made to make this account complete; various formulas are quoted, but no proofs are offered since the chapter is intended partly for reference. For an account of the theory the reader should consult a work on numerical analysis. Difference notation is not universal; here we shall use that given in *Interpolation and Allied Tables*, published by H. M. Nautical Almanac Office (Ref. 47). This is a small but invaluable volume containing details of many numerical procedures and hints on using them, as well as suitable tables.

Suppose some function, $f(x)$, to be tabulated at equal intervals, h, of the independent variable x. If x_0 is some suitable tabular point, then we have a table of values $f(x_0+ph)$ for integral values of p; this may be written $f(x_p)$, or simply f_p. The argument is (x_0+ph), but this can be written without ambiguity as p.

If we subtract f_p from f_{p+1}, we shall have formed the *first difference*, $\delta_{p+\frac{1}{2}}$. It is written to the right of the column of values of f_p and between the value of f_p and f_{p+1} (see Table 10.1). If we form a column of first differences in this way, and difference this column, we form the second differences. Thus

$$\delta_{p+\frac{1}{2}} - \delta_{p-\frac{1}{2}} = \delta_p^2.$$

Here the superscript denotes the number of the difference, and the subscript shows the position with respect to the argument. This differencing can be carried out indefinitely, in principle. A formal scheme is shown in Table 10.1.

Table 10.1 can be modified by inserting arithmetical means of the entries standing immediately above and below a space. These *half-differences* will always be preceded by "μ." Thus

$$\mu\delta f_p = \tfrac{1}{2}(\delta_{p+\frac{1}{2}} + \delta_{p-\frac{1}{2}}).$$

This is shown in Table 10.2.

Table 10.1

		Differences				
Argument	*Function*	*1st*	*2nd*	*3rd*	*4th*	⋯
-2	f_{-2}		δ^2_{-2}		δ^4_{-2}	
		$\delta_{-1\frac{1}{2}}$		$\delta^3_{-1\frac{1}{2}}$		
-1	f_{-1}		δ^2_{-1}		δ^4_{-1}	
		$\delta_{-\frac{1}{2}}$		$\delta^3_{-\frac{1}{2}}$		
0	f_0		δ^2_0		δ^4_0	⋯
		$\delta_{\frac{1}{2}}$		$\delta^3_{\frac{1}{2}}$		
$+1$	f_1		δ^2_1		δ^4_1	
		$\delta_{1\frac{1}{2}}$		$\delta^3_{1\frac{1}{2}}$		
$+2$	f_2		δ^2_2		δ^4_2	

This table is assumed to be part of a more extensive one. If we start with only the five values tabulated, then only one fourth difference, δ^4_0, can be found, and the difference table ends there.

Table 10.2

		Differences				
Argument	*Function*	*1st*	*2nd*	*3rd*	*4th*	⋯
-1	f_{-1}	$[\mu\delta_{-1}]$	δ^2_{-1}	$[\mu\delta^3_{-1}]$	δ^4_{-1}	
	$[\mu f_{-\frac{1}{2}}]$	$\delta_{-\frac{1}{2}}$	$[\mu\delta^2_{-\frac{1}{2}}]$	$\delta^3_{-\frac{1}{2}}$	$[\mu\delta^4_{-\frac{1}{2}}]$	
0	f_0	$[\mu\delta_0]$	δ^2_0	$[\mu\delta^3_0]$	δ^4_0	⋯
	$[\mu f_{\frac{1}{2}}]$	$\delta_{\frac{1}{2}}$	$[\mu\delta^2_{\frac{1}{2}}]$	$\delta^3_{\frac{1}{2}}$	$[\mu\delta^4_{\frac{1}{2}}]$	
$+1$	f_1	$[\mu\delta_1]$	δ^2_1	$[\mu\delta^3_1]$	δ^4_1	

We can regard the tabulated values, f_p, as being the first differences of another function, which we shall call the *first sum* of f_p. The first sums can be regarded in their turn as the differences of the *second sums*, and so on. We then have the scheme shown in Table 10.3.

In forming a sum there is uncertainty to the extent of an additive constant. There is an analogy here between summing and integrating, and also between differencing and differentiating. In fact the δ's can be regarded as operators

that obey the ordinary laws of algebra.

Table 10.3

	Sums			*Differences*				
Argument	*2nd*	*1st*	*Function*	*1st*	*2nd*	*3rd*	*4th*	⋯
-2	δ^{-2}_{-2}		f_{-2}		δ^2_{-2}		δ^4_{-2}	
		$\delta^{-1}_{-1\frac{1}{2}}$		$\delta_{-1\frac{1}{2}}$		$\delta^3_{-1\frac{1}{2}}$		
-1	δ^{-2}_{-1}		f_{-1}		δ^2_{-1}		δ^4_{-1}	
		$\delta^{-1}_{-\frac{1}{2}}$		$\delta_{-\frac{1}{2}}$		$\delta^3_{-\frac{1}{2}}$		
0	δ^{-2}_0		f_0		δ^2_0		δ^4_0	⋯
		$\delta^{-1}_{\frac{1}{2}}$		$\delta_{\frac{1}{2}}$		$\delta^3_{\frac{1}{2}}$		
$+1$	δ^{-2}_1		f_1		δ^2_1		δ^4_1	
		$\delta^{-1}_{1\frac{1}{2}}$		$\delta_{1\frac{1}{2}}$		$\delta^3_{1\frac{1}{2}}$		
$+2$	δ^{-2}_2		f_2		δ^2_2		δ^4_2	

The notation just used is that of *central differences*. Another notation, to be used later in this chapter, is that of *backward differences*. The numbers and their location are precisely the same as they would be in Table 10.3, but the symbols are changed as follows:

Table 10.4

	Sums			*Differences*				
Argument	*2nd*	*1st*	*Function*	*1st*	*2nd*	*3rd*	*4th*	⋯
-2	∇^{-2}_{-3}		f_{-2}		∇^2_{-1}		∇^4_0	
		∇^{-1}_{-2}		∇_{-1}		∇^3_0		
-1	∇^{-2}_{-2}		f_{-1}		∇^2_0		∇^4_1	
		∇^{-1}_{-1}		∇_0		∇^3_1		
0	∇^{-2}_{-1}		f_0		∇^2_1		∇^4_2	⋯
		∇^{-1}_0		∇_1		∇^3_2		
$+1$	∇^{-2}_0		f_1		∇^2_2		∇^4_3	
		∇^{-1}_1		∇_2		∇^3_3		
$+2$	∇^{-2}_1		f_2		∇^2_3		∇^4_4	

10.2 Interpolation

The tables of the preceding section refer to values of $f(x_0 + ph)$ for integral values of p; but once an adequate table has been constructed, it can be used for finding the value of f for any value of p that is included comfortably within the range of the table. This is known as *interpolation*. In the following work we shall assume $0 \le p < 1$.

We notice that if the second differences are negligible or zero (and there need be no distinction in numerical work), then

$$f_p = f_0 + p\delta_{\frac{1}{2}}. \tag{10.2.1}$$

In most books of tables the interval of tabulation, h, is chosen to be so small that second differences are usually negligible; but this cannot normally be done in a practical calculation. We might suspect that f_p would in general be given by a formula involving successive differences in a way that depends on p. This is the case; two such relations, out of many, will be described here. The first is Bessel's formula:

$$f_p = f_0 + p\delta_{\frac{1}{2}} + B_2(\delta_0^2 + \delta_1^2) + B_3\delta_{\frac{1}{2}}^3 + B_4(\delta_0^4 + \delta_1^4) + \cdots \tag{10.2.2}$$

where the B's, or *Bessel interpolation coefficients*, are functions of p. For instance

$$B_2 = \tfrac{1}{4}p(p-1), \quad B_3 = \tfrac{1}{12}p(p-1)(2p-1), \quad B_4 = \tfrac{1}{48}p(p^2-1)(p-2), \ldots$$

These functions are tabulated in standard references.

The second relation is Everett's formula:

$$f_p = (1-p)f_0 + pf_1 + E_2\delta_0^2 + F_2\delta_1^2 + E_4\delta_0^4 + F_4\delta_1^4 + \cdots \tag{10.2.3}$$

The Everett coefficients are related to the Bessel coefficients by

$$E + F = 2B,$$

and we also have $E(p) = F(1-p)$ and $E(1-p) = F(p)$.

We shall illustrate the use of these formulas by taking a table of $\sin x$ to six places of decimals where $h = 5°$. This is given in Table 10.4, with differences up to the sixth. When differencing a table, the differences should be taken up to the stage where they become small and irregular; this is the case for the fifth and sixth differences. Since the interpolation coefficients become very small anyway for these differences, they will not contribute anything in the interpolation.

Table 10.5

$x°$	$\sin x$	δ	δ^2	δ^3	δ^4	δ^5	δ^6
0	0.000 000						
		+87 156					
5	+0.087 156		− 664				
		86 492		−657			
10	173 648		1321		+ 8		
		85 171		649		+ 8	
15	258 819		1970		16		− 4
		83 201		633		4	
20	342 020		2603		20		− 1
		80 598		613		3	
25	422 618		3216		23		+ 6
		77 382		590		+ 9	
30	500 000		3806		32		−12
		73 572		558		− 3	
35	573 576		4364		29		+14
		69 212		529		+11	
40	642 788		4893		40		− 9
		64 319		489		+ 2	
45	707 107		5382		42		− 3
		58 937		447		− 1	
50	766 044		5829		41		+11
		53 108		406		+10	
55	819 152		6235		51		−14
		46 873		355		− 4	
60	866 025		6590		47		+11
		40 283		308		+ 7	
65	906 308		6898		54		− 6
		33 385		254		+ 1	
70	939 693		7152		55		− 1
		26 233		199		0	
75	965 926		7351		55		+ 2
		18 882		144		+ 2	
80	984 808		7495		+57		
		11 387		− 87			
85	996 195		−7582				
		+ 3 805					
90	+1.000 000						

Decimal points can be omitted without ambiguity in this case, with the understanding that quantities are in units of the sixth place.

Let us use this table to find the value of $\sin 41°$; then $x_0 = 40°$ and $p = 0.200\,000$. First we shall use the Everett formula. From tables we find

$$E_2 = -0.048, \qquad F_2 = -0.032,$$
$$E_4 = +0.008, \qquad F_4 = +0.006;$$

further coefficients are not needed. We then find

$$\begin{aligned}(1-p)f_0 &= +0.514\,230\\ pf_1 &= +\ 141\,421\\ E_2\delta_0^2 &= +\ 235\\ F_2\delta_1^2 &= +\ 172\\ E_4\delta_0^4 &= \ 0\\ F_4\delta_1^4 &= \ \underline{\quad 0}\\ f_p &= +0.656\,058.\end{aligned}$$

The appropriate Bessel coefficients are

$$B_2 = -0.040, \quad B_3 = +0.008, \quad B_4 = +0.007.$$

We find

$$\begin{aligned}f_0 &= +0.642\,788\\ p\delta_{\frac{1}{2}} &= +\ 12\,864\\ B_2(\delta_0^2+\delta_1^2) &= +\ 411\\ B_3\delta_{\frac{1}{2}}^3 &= -\ 4\\ B_4(\delta_0^4+\delta_1^4) &= \underline{+\quad 1}\\ f_p &= +0.656\,060.\end{aligned}$$

The actual answer, correct to six places, is +0.656 059, so that the agreement is good, considering the number of calculations performed.

If the fourth differences are small (less than 1000), the term in B_4 may be neglected if the second differences are modified so that

$$\delta_{mod}^2 = \delta^2 - 0.184\,\delta^4. \tag{10.2.4}$$

The fourth differences are said to have been *thrown back*. A similar device can be used in Everett's formula. There are more complicated throwbacks involving higher differences; the reader will find these described in the standard references.

If in our example we had required the value of $\sin 86°$, on the basis of Table 10.5 alone, it would have been necessary to guess some of the differences. This can be done with reasonable confidence, since the run of differences is smooth; we can reasonably assume that the fourth differences will remain at about +57, and so build up the lower differences. If our guess of the fourth differences was wrong by 10, the error in the product $B_4\delta^4$ would be less than one, and so would not affect the accuracy of the answer.

If we build up extra differences in this way, it is possible to add extra values to the table. Let us assume that the next fourth difference is +57. Then, extra third, second, and first differences of −30, −7612, and −3807, respectively, can be added to the table of differences, so that we can deduce

that $\sin 95° = +0.996\,193$, giving an error of two only. This process is known as *extrapolation*.

We can also use Table 10.5 to find the argument for a given value of $\sin x$. This is known as *inverse interpolation*. It requires successive approximations, as, unfortunately, there is no direct formula that can be used.

Backward differences also provide convenient notation for interpolation. Suppose that a function is tabulated as in Table 10.6, where $f_{-i} = f(t_0 - ih)$. The important differences are those in the final diagonal. They can easily be computed from the initial table (see below); also, if we know only f_0 and the lowest diagonal of differences, the values of the function, and all other differences can be recovered. It is the terms in the lowest diagonal that are used in the formula.

Table 10.6

f_{-m}					
⋮	⋮				
⋮	⋮	⋮			
⋮	⋮	⋮	⋮		
⋮	⋮	⋮	⋮	⋮	∇_0^m
⋮	⋮	⋮	⋮	⋮	
f_{-2}	⋮	⋮	⋮		
	⋮	∇_0^3			
f_{-1}	∇_0	∇_0^2			
f_0					

Let $t = t_0 - ph$ lie in the range of the table, so that p is not restricted to the interval $(0, 1)$. The formula for interpolating $f(t_0 - ph)$ is

$$\left.\begin{aligned} &f(t_0 - ph) \simeq f(t_0) + \sum_{n=1}^{m} c_n(p)\nabla_0^n,\\ &\text{where}\\ &c_n(p) = (-1)^n\binom{p}{n} = (-1)^n\frac{p(p-1)\cdots(p-n+1)}{1\cdot 2\cdot\cdots\cdot n}. \end{aligned}\right] \tag{10.2.5}$$

The coefficients c_n can be found recursively from

$$\begin{aligned} &c_0(p) = 1,\\ &c_n(p) = \frac{n-p-1}{n}c_{n-1}(p), \quad n = 1, 2, \ldots, m. \end{aligned} \tag{10.2.6}$$

Consider a program for carrying this out. Suppose that we have a table of values of a function and that `F(I)` stands for f_{-i}. Then if `D(I)` stands for ∇_0^i, they can be found from

```
100 FOR I = 1 TO M
110  D(I) = F(I-1) - F(I)
120 NEXT I
130 FOR J = 2 TO M
140  FOR I = J TO M
150   D(M+J-I) = D(M-1+J-I) - D(M+J-I)
160  NEXT I
170 NEXT J
```

If we want to approximate the value of $f(t)$ by interpolation, then

```
200 P = (T0 - T)/H
210 FINT = F
220 C = 1
230 FOR N = 1 TO M
240  C = C*(N - P - 1)/N
250  FINT = FINT + C*D(N)
260 NEXT N
```

Write a program for carrying out these computations (preferably in double precision) using a function that can be accurately computed, so that the error of the interpolation will be printed out. Experiment with various h and M, playing with some ridiculous extremes. Also notice how the error mounts steeply if the value of t wanders outside the interval covered by the table.

A *Lagrangian* formula is one in which the tabular values of f appear, but no differences. For instance,

$$f(t_0 - x) \simeq \sum_{n=0}^{m} p_n(x) f(t_0 - nh),$$

where the $p_n(x)$ are polynomials in x. Such formulas will be applied to the solution of differential equations in section 10.7, but are not recommended here for interpolation.

One other useful technique for interpolation involves *divided differences.* An account of this, with programs, can be found in Refs. 49 and 52. An advantage of the method is that the values of f do not have to be tabulated at regular intervals, nor, indeed, in any special order.

10.3 Differentiation

If all the second differences in a table were zero, the function would be an arithmetical series, and its differential coefficient would be given exactly by

$$h\frac{df}{dx} = hf' = \frac{df}{dp} = \delta_{\frac{1}{2}}. \tag{10.3.1}$$

This corresponds to the case (10.2.1) considered in the preceding section, and so we would expect in general to find f' given by a series involving higher differences. The reader should note the appearance of h on the left-hand side of (10.3.1); it must never be forgotten.

If the Bessel interpolation coefficients are expressed in terms of p, then (10.2.2) can be differentiated any number of times to give appropriate formulas

for successive differential coefficients. We have, for instance,

$$hf'_p = \delta_{\frac{1}{2}} + \tfrac{1}{4}(2p-1)(\delta_0^2 + \delta_1^2) + B'_3\delta_{\frac{1}{2}}^3 + B'_4(\delta_0^4 + \delta_1^4) + \cdots \tag{10.3.2}$$

The B'_n are tabulated in the standard references. As in the case of interpolation, there are alternative formulas.

Often only the derivatives at tabular or half-tabular points are required. The coefficients can then be given definite numerical values. For tabular points we have

$$hf'_0 = \mu\delta_0 - \tfrac{1}{6}\mu\delta_0^3 + \tfrac{1}{30}\mu\delta_0^5 - \tfrac{1}{140}\mu\delta_0^7 + \cdots \tag{10.3.3}$$

and

$$h^2 f''_0 = \delta_0^2 - \tfrac{1}{12}\delta_0^4 + \tfrac{1}{90}\delta_0^6 + \cdots \tag{10.3.4}$$

As an illustration let us find the differential coefficient of $\sin 40°$ from Table 10.4. Expressed in radians, h is 0.087 2665. We find

$$\mu\delta_0 = +66\,766,$$

and

$$\tfrac{1}{6}\mu\delta_0^3 = +85,$$

from which we find

$$\begin{aligned}\sin' 40° &= +0.066\,681 \text{ divided by } 0.087\,266\\ &= +0.766\,06.\end{aligned}$$

This compares with the correct figure, 0.766 04.

The formulas for the derivatives at half-tabular points are

$$hf'_{\frac{1}{2}} = \delta_{\frac{1}{2}} - \tfrac{1}{24}\delta_{\frac{1}{2}}^3 + \tfrac{3}{640}\delta_{\frac{1}{2}}^5 - \cdots \tag{10.3.5}$$

and

$$h^2 f''_{\frac{1}{2}} = \mu\delta_{\frac{1}{2}}^2 - \tfrac{5}{24}\mu\delta_{\frac{1}{2}}^4 + \tfrac{259}{5760}\mu\delta_{\frac{1}{2}}^6 - \cdots \tag{10.3.6}$$

10.4 Integration

The analogy between sums and integrals (or the direct integration of Bessel's interpolation formula) leads us to expect a formula of the type

$$\int^p f_p dp = \delta_{\frac{1}{2}}^{-1} + A_0^1(f_0 + f_1) + A_1^1\delta_{\frac{1}{2}} + A_2^1(\delta_0^2 + \delta_1^2) + \cdots \tag{10.4.1}$$

for the integral of f_p. This is the case, and the A's are tabulated in standard references. A similar formula holds for double integrals; these involve the second sums. We note that

$$\int f(x)dx = h\int f_p dp.$$

For our purposes we shall need only to evaluate integrals at tabular points, for which $p = 0$. Then we have

$$h^{-1}\int^0 f(x)dx = \int^0 f_p dp$$
$$= \mu\delta_0^{-1} - \tfrac{1}{12}\mu\delta_0 + \tfrac{11}{720}\mu\delta_0^3 - \tfrac{191}{60480}\mu\delta_0^5 + \cdots \quad (10.4.2)$$

and

$$h^{-2}\iint^0 f(x)dx^2 = \iint^0 f_p dp^2$$
$$= \mu\delta_0^{-2} + \tfrac{1}{12}f_0 - \tfrac{1}{240}\delta_0^2 + \tfrac{31}{60480}\mu\delta_0^4 - \cdots \quad (10.4.3)$$

These expressions will be needed later when we consider the numerical solution of differential equations.

The notation $\int^p$ is confusing because an integral must have two limits for it to have a definite value. Specifying only the upper limit results in uncertainty to the extent of an added constant; but this is related in (10.4.1) to the uncertainty in forming the first sums. We can express a definite integral as

$$\int_q^p f\,dp = \int_C^p f\,dp - \int_C^q f\,dp,$$

where C is arbitrary. Then let us omit the C altogether and apply (10.4.1) to each integral. The final answer for the definite integral will not be influenced by the choice of arbitrary constant when forming the first sums, as this will disappear in the subtraction.

When solving a differential equation, the sums must be known definitely; in this case they are found by using the initial conditions. Suppose we are required to calculate y from

$$\frac{dy}{dx} = \sin x,$$

subject to the initial condition that when $x = 45°$, $y = -0.707\,107$. Using Table 10.4, let $x_0 = 45°$; then from (10.4.2) we find

$$-0.707\,107 = h(\mu\delta_0^{-1} - 0.005\,143),$$

from which

$$\mu\delta_0^{-1} = -8.097\,707,$$

where the final digit is unreliable. Then since

$$\delta_{\frac{1}{2}}^{-1} - \delta_{-\frac{1}{2}}^{-1} = +0.707\,107,$$

we find

$$\delta_{\frac{1}{2}}^{-1} = -7.744\,154.$$

Using this value, the complete column of first sums can be entered into Table 10.4, and the application of (10.4.2) will yield values of $-\cos x$. The reader should verify that this is, in fact, the case.

Definite integrals are often evaluated numerically by using values of the function to be integrated at tabular points, instead of its sums and differences. There are many suitable formulas; here we shall quote only one, known as the *repeated Simpson rule*. Let the function $y = f(x)$ be integrated from $x_0 = a$ to $x_n = b$, and suppose y to be tabulated for all the x_i, such that $y_i = f(x_i)$, $i = 1, 2, 3, \ldots, n$. We assume that n is even. Then

$$\int_a^b y dx = \frac{b-a}{3n}(y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_4 + \cdots + 2y_{n-2} + 4y_{n-1} + y_n). \quad (10.4.4)$$

The error of this formula is proportional to the fourth power of h, the interval of tabulation, making the approximation adequate for many purposes, as well as being simple to use.

If you must evaluate a lot of integrals numerically, then you should acquire the technique of Gaussian integration.

10.5 Errors

In a world in which the stored precision of a number must be limited, and we are to use finite numerical procedures to approximate the results of calculus, numerical errors are unavoidable and natural. In this section we shall look at some aspects of their nature and control. "Error" in this context should be distinguished from "blunder". Blunders can happen because a program is not completely debugged, or because the `INPUT` quantities have errors in them, or because the operator does not understand the natural limitations of a program, or the `OUTPUT` quantities are misinterpreted. Remember that the hardest part of debugging starts *after* the program has run all the way for the first time. This is why it is important, whenever possible, to apply a program first to problems for which the solution is known. Also debugging should be comprehensive; for instance, if an angle is to be computed from inverse trigonometric functions, then the program should be run for enough cases to ensure the correctness of the quadrant. Before starting a calculation, you should have some idea as to what the `OUTPUT` should look like, so that you can spot absurdities. This requires experience. There may be internal checks that can be used. For instance, a system of differential equations may have an *integral*, i.e., a function of the variables that is constant along any solution. (Conservation of energy, for instance, or Jacobi's integral in the restricted problem.) Monitoring this function at each state of the integration can make possible an early warning if something is going wrong.

The *respectable* errors to be discussed arise from two causes. Round-off error is due to finite word length. Truncation error is due basically to the fact that

while a mathematical result may depend on a limiting process (the convergence of a series, for instance), we are limited to a finite number of steps.

In the past, a potent reason for the seriousness of round-off error was the fact that in most calculations every digit was precious. This need not be the case today. In simple calculations, round-off errors will be mostly (but not invariably) confined to the final two or three decimal places; so if we require, ten-figure accuracy, and use double precision, the round-off errors may be safely out of sight. They should never be out of mind; ultimately, if a program is large enough, or runs for long enough, round-off errors can kill it. Certain bad programming practices can expedite the process; subtracting nearly equal quantities, for instance, or using units unwisely so that quantities in the same formula (position and velocity, say) can have very different orders of magnitude. When debugging a program, using a model with known results, be satisfied only with agreement consistent with the precision that you are using.

A good way to get some experience with truncation error is to experiment with the program of section 10.2, varying M. A mathematical formula of interpolation using differences would involve an infinite series; in theory, it might seem that the more differences we use, the better will be the results. This is only true up to a point. We try to illustrate this using Table 10.7. If the entries in a table are rounded off to k decimal places, then the neglected parts will lie between $\pm\frac{1}{2}$; these errors will affect the differences, and the maximum possible effects are shown by assuming that the errors are alternatively $+\frac{1}{2}$ and $-\frac{1}{2}$, in units of the final decimal used. Suppose, for example, that we are dealing with a tabulated function where (were there no round-off error) the sixth differences would be near zero; it is easy to see that any numbers that *actually* appear in the sixth differences (and they could be as great as 32) are garbage, and that any calculation involving them is contributing nonsense. The higher the differences that are involved, the greater is the danger.

Table 10.7

	δ	δ^2	δ^3	δ^4	δ^5	δ^6	
$+\frac{1}{2}$		-2		$+8$		-32	
	-1		$+4$		-16		
$-\frac{1}{2}$		$+2$		-8		$+32$	
	$+1$		-4		$+16$		
$+\frac{1}{2}$		-2		$+8$		-32	$\cdots$
	-1		$+4$		-16		
$-\frac{1}{2}$		$+2$		-8			
	$+1$		-4				
$+\frac{1}{2}$		-2					
	-1						
$-\frac{1}{2}$							

The rest of this chapter is concerned with the numerical solution of a system of differential equations. Before looking at some specific methods, we need to make a few general points. Any method proceeds in *steps*; the independent variable is increased by a quantity h, called the *stepsize*. Associated with a step

there will be *local truncation error* that will depend on the method used and on the stepsize. This error can usually be approximated by an expression of the form Ah^k. Mathematically, we could say that the limit, as h approaches zero, of (local truncation error)$/h^k$ exists, and is equal to A. The number $(k-1)$ is called the *order* of the method. The step will also suffer from round-off error. So even if conditions were perfect at the start of a step, there will be error at the end. We have consequently been set onto a different solution path of the differential equations. The error will propagate into future steps, giving *accumulated* error, which depends not only on the numerical method used, but also on the nature of the differential equations. This can be serious. Consider

$$\frac{d^2x}{dt^2} = x; \qquad x(0) = 1, \quad \dot{x}(0) = -1.$$

The general solution is $x = c_1e^t + c_2e^{-t}$; applying the initial conditions, and assuming perfect precision, we find $c_1 = 0$ and $c_2 = 1$, so the solution decays exponentially. But at the end of the first step of a numerical integration we shall be on a different solution path — one for which c_1 is not zero; however small this false c_1 may be, we shall now be on a solution path that has exponential growth.

In orbital work the principal effect of errors can be seen by considering Keplerian motion. A small error in position and velocity will lead to small errors in the elements, so that the size, shape and orientation of the orbit will be changed, but only by a correspondingly small amount. But since the mean motion will be changed, the mean anomaly will have an error that increases steadily with the time. So the principal error in most orbital integrations is in the position along the orbital path, rather than in the path itself. Physically, this corresponds to fragments of a comet spreading out along the orbit of the comet.

Numerical methods for the solution of differential equations can be classified under *single-step* and *multistep* methods. In the first, every step taken is like an initial value problem; no account is taken of the past history of the solution. Such methods include the application of power series and Runge-Kutta methods. Power series can be powerful and are especially useful if a set of equations is to be solved many times; but each problem must be programmed individually. They will not (with regret) be discussed here; an introduction to their use, with applications, can be found in Ref. 52. In a single-step method the stepsize can be changed at each step; if a solution is uneventful, h can be increased, but it must be decreased as the action becomes more exciting. In Section 10.6 we describe a Runge-Kutta method that has automatic stepsize control.

Runge-Kutta methods are usually of low order and are slow because for each step the right-hand sides of the equations must be calculated many times. Multistep methods use results from preceding steps; they are faster in operation and can easily be designed to higher order. But variation of the stepsize is clumsy. Usually it is either halved or doubled. Doubling requires storage of a lot of previous calculations; halving requires interpolation. The operations are

not difficult, and involve only programming considerations. More troublesome is the logic for deciding whether or not the stepsize should be changed; here opinions may differ. For reasons of space, stepsize control for multistep methods will not be included in the programs to be described.

Multistep methods can be set out to use tabular values (Lagrangian formulas) or differences. Tabular values are more commonly used today, although I prefer differences. In section 10.7 a method is given for first-order equations using Lagrangian formulas. In section 10.8 you will find the traditional method of celestial mechanics for second-order equations.

10.6 The Numerical Integration of Differential Equations—Runge-Kutta Methods

The subject of the numerical solution of differential equations is vast and wonderful; here I offer only the briefest introduction, with the outlines of some basic methods, and, I hope, some useful ideas for programs. On no account should you allow your knowledge to be restricted by what is contained in this book. There is an element of art to the subject. A lot of judgment is needed; so experience, gained through experiment, is important.

Methods will be described here in the first place for single equations, but the programs will apply to systems. Runge-Kutta methods are *single-step* methods, which means that each step is like starting an initial value problem, being independent of whatever went before. Consider

$$\frac{dx}{dt} = f(t,x); \quad x(t_0) = x_0. \tag{10.6.1}$$

The best known Runge-Kutta method has for a step, with stepsize h, the following operations:

$$\left.\begin{aligned} f_0 &= f(t_0, x_0), \\ f_1 &= f(t_0 + \tfrac{1}{2}h, x_0 + \tfrac{1}{2}hf_0), \\ f_2 &= f(t_0 + \tfrac{1}{2}h, x_0 + \tfrac{1}{2}hf_1), \\ f_3 &= f(t_0 + h, x_0 + hf_2), \\ x(t_0 + h) \simeq x_1 &= x_0 + \tfrac{1}{6}(f_0 + 2f_1 + 2f_2 + f_3). \end{aligned}\right] \tag{10.6.2}$$

This method is of the *fourth order*. This means that for small enough h, the error of the approximation can itself be approximated by Ah^5, for A independent of h; or

$$x(t_0 + h) - x_1 \simeq Ah^5. \tag{10.6.3}$$

A great advantage of this algorithm is that it is quick and easy to program; but for most orbital calculations it is neither accurate nor fast enough to run. It will be used in the following section to set up a table of values of

$x(t_0 + nh), \quad n = 0, 1, \ldots$ so that a more powerful method of solution can then be started.

But one modification of the Runge-Kutta method will be described here in detail. It allows for full automatic control of the stepsize from one step to the next, in conformity with an error bound that can be set by the operator. It is one of a class of methods due to Fehlberg, E. (Ref. 51). He has published algorithms for many different orders; the one chosen here can give fifth-order accuracy. In essence, the method runs, in tandem, two algorithms; at the end of a step there are two separate approximations for x: x_1 has fourth-order, and $\hat{x}_1$ has fifth-order accuracy. So

$$x(t_0 + h) \simeq x_1 + Ah^5, \quad x(t_0 + h) \simeq \hat{x}_1 + Bh^6. \tag{10.6.4}$$

The difference $(\hat{x}_1 - x_1)$ is therefore an approximation to the *local truncation error* of the approximation x_1. The stepsize is calculated, or adjusted, so that this difference will always be less than some prescribed quantity. If the error of x_1 is less than this quantity, so, a fortiori, will be the error of $\hat{x}_1$; $\hat{x}_1$ is used as the starting value for the next step.

Let

$$TE = |\hat{x}_1 - x_1|. \tag{10.6.5}$$

Suppose that we have just calculated a step in the integration. TE is calculated; if it is too large, or, say, if

$$TE > TL, \tag{10.6.6}$$

where TL is the greatest tolerable local truncation error, then the step cannot be used, and must be repeated with a smaller stepsize. But if $TE < TL$, then the step was good, and we need to find a stepsize for the next step. Either way, when a step has been calculated, a new value of h must be found.

Now, approximately,

$$TE \simeq Ah^5, \quad \text{so} \quad A \simeq \frac{TE}{h^5}. \tag{10.6.7}$$

If the local truncation error resulting from a new stepsize h_1 were to equal TL, then

$$Ah_1^5 \simeq TL,$$

so, from (10.6.7),

$$h_1 \simeq \left(\frac{TL}{A}\right)^{1/5} \simeq h\left(\frac{TL}{TE}\right)^{1/5}.$$

It would be dangerous to replace "$\simeq$" by "$=$" to give a formula for h_1, because of all the approximations made. Accordingly, set

$$h_1 = CH \cdot h\left(\frac{TL}{TE}\right)^{1/5}, \tag{10.6.8}$$

where *CH*, the "chicken factor", is a number less than one. I have found that *CH*= .9 is usually satisfactory, ensuring that *TE* will actually be less than *TL*. Its value can change in different circumstances, depending on the system to be solved, and the courage of the operator.

The basic formulas follow. The notation has been changed from that originally used by Fehlberg in order to eliminate zero subscripts, making programing in FORTRAN easier.

For

$$\frac{dx}{dt} = f(t,x); \quad x(t_0) = x_0,$$

let

$$\left.\begin{aligned}
f_1 &= f(t_0, x_0),\\
f_2 &= f[t_0 + a_2 h, x_0 + h(b_{21} f_1)],\\
f_3 &= f[t_0 + a_3 h, x_0 + h(b_{31} f_1 + b_{32} f_2)],\\
f_4 &= f[t_0 + a_4 h, x_0 + h(b_{41} f_1 + b_{42} f_2 + b_{43} f_3)],\\
f_5 &= f[t_0 + a_5 h, x_0 + h(b_{51} f_1 + b_{52} f_2 + b_{53} f_3 + b_{54} f_4)],\\
f_6 &= f[t_0 + a_6 h, x_0 + h(b_{61} f_1 + b_{62} f_2 + b_{63} f_3 + b_{64} f_4 + b_{65} f_5)],\\
x_1 &= x_0 + h(c_1 f_1 + c_2 f_2 + c_3 f_3 + c_4 f_4 + c_5 f_5),\\
\hat{x}_1 &= x_0 + h(\hat{c}_1 f_1 + \hat{c}_2 f_2 + \hat{c}_3 f_3 + \hat{c}_4 f_4 + \hat{c}_5 f_5 + \hat{c}_6 f_6).
\end{aligned}\right\} \tag{10.6.9}$$

The values of the constants are given in Table 10.8.

Table 10.8

	a_K	b_{KL} = B(K,L)					c_K	$\hat{c}_K$
	A(K)	$L=1$	2	3	4	5	C(K)	CH(K)
$K=1$	0						$\frac{1}{9}$	$\frac{47}{450}$
2	$\frac{2}{9}$	$\frac{2}{9}$					0	0
3	$\frac{1}{3}$	$\frac{1}{12}$	$\frac{1}{4}$				$\frac{9}{20}$	$\frac{12}{25}$
4	$\frac{3}{4}$	$\frac{69}{128}$	$-\frac{243}{128}$	$\frac{135}{64}$			$\frac{16}{45}$	$\frac{32}{225}$
5	1	$-\frac{17}{12}$	$\frac{27}{4}$	$-\frac{27}{5}$	$\frac{16}{15}$		$\frac{1}{12}$	$\frac{1}{30}$
6	$\frac{5}{6}$	$\frac{65}{432}$	$-\frac{5}{16}$	$\frac{13}{16}$	$\frac{4}{27}$	$\frac{5}{144}$		$\frac{6}{25}$

From this it follows that

$$\begin{aligned} TE &= \hat{x}_1 - x_1 \\ &= h\left(-\tfrac{1}{150} f_1 + \tfrac{3}{100} f_3 - \tfrac{16}{75} f_4 - \tfrac{1}{20} f_5 + \tfrac{6}{25} f_6\right). \end{aligned} \tag{10.6.10}$$

In the program *TE* and $\hat{x}_1$ are calculated, but not x_1; the coefficients in (10.6.10) are stored as CT(I).

To be useful, these formulas must be generalized to apply to a system of first-order equations. This generalization can be considered in terms of programing.

We shall consider a system of *NQ* equations:

$$\frac{dX(I)}{dT} = F_I[T, X(1), X(2), \ldots, X(NQ)], \quad I = 1, 2, \ldots, NQ. \tag{10.6.11}$$

The f's computed in (10.6.9) must be taken for each equation, so that a double array, f_{ij} or F(I,J), will be generated, with i (I) going from 1 to 6 and j (J) going from 1 to *NQ*. Then, generalizing (10.6.10), there will be *NQ* separate values of *TE*; hence the appearance of the array TE(I). The greatest of these, in absolute value, is what Fehlberg calls *TEMAX*; it is TM in the program. *TEMAX* is used in the calculation of the next stepsize. To calculate the right-hand sides of the equations (10.6.11) it is necessary to write a subroutine which, when given values for $t, x_1, x_2, \ldots, x_{NQ}$, will have as output $Z_i = f_i(t, x_1, x_2, \ldots, x_{NQ})$ (or the equivalent, with changed notation). This starts at line 1000 in the program.

But what do we do if we do *not* start with a system of the form (10.6.11)? Suppose that we want to integrate numerically the system:

$$\left.\begin{aligned} \frac{d^2x}{dt^2} &= FX(t,x,y,z,\dot{x},\dot{y},\dot{z}),\\ \frac{d^2y}{dt^2} &= FY(t,x,y,z,\dot{x},\dot{y},\dot{z}),\\ \frac{d^2z}{dt^2} &= FZ(t,x,y,z,\dot{x},\dot{y},\dot{z}). \end{aligned}\right\} \tag{10.6.12}$$

Define

$$X_1 = x, \quad X_2 = y, \quad X_3 = z, \quad X_4 = \dot{x}, \quad X_5 = \dot{y}, \quad X_6 = \dot{z}. \tag{10.6.13}$$

Then the original system can be written in the form:

$$\left.\begin{aligned} \frac{dX_1}{dt} &= X_4,\\ \frac{dX_2}{dt} &= X_5,\\ \frac{dX_3}{dt} &= X_6,\\ \frac{dX_4}{dt} &= FX(t,X_1,X_2,X_3,X_4,X_5,X_6),\\ \frac{dX_5}{dt} &= FY(t,X_1,X_2,X_3,X_4,X_5,X_6),\\ \frac{dX_6}{dt} &= FZ(t,X_1,X_2,X_3,X_4,X_5,X_6). \end{aligned}\right\} \tag{10.6.14}$$

Any system of higher-order equations can be reduced to a system of first-order equations by a device of this kind.

In the following listing the functions subroutine is shown for pure Keplerian motion, and the initial conditions are the same as those used in section 6.8. The program should, anyway, be debugged using a problem for which a check is available. In starting up the program, there must be an initial suggestion for the stepsize; if it is too large, the program will automatically reduce it; if it is too small, the program will use it for one (good) step, and then increase it. In the program the greatest tolerable local truncation error was TL $= 10^{-7}$. It was arranged that the program would finish with the value $t = 1$ (MT). This is accomplished by working until MT is exceeded, and then taking a (negative) step to land precisely on MT; in this way the very last step is guaranteed to be good.

The program took steps giving tabulated values at the times: .1, .2191.., .3387.., .4593.., .5817.., .7069.., .8337.., .9583.., 1.0816.., and 1. The final values (with those from section 6.8 in brackets for comparison) were:

$$
\begin{aligned}
&t = 1.\\
&x = \ \ .466\,080\,761\,934\,931\,5 \quad (\ \ .466\,080\,784\,653\,923\,4),\\
&y = \ \ .900\,611\,213\,341\,938\,9 \quad (\ \ .900\,611\,234\,907\,723\,6),\\
&z = \ \ .114\,045\,979\,309\,679\,8 \quad (\ \ .114\,045\,980\,667\,323\,4),\\
&x = -.876\,594\,442\,579\,689\,2 \quad (-.876\,594\,436\,852\,508\,4),\\
&y = \ \ .473\,156\,593\,619\,036\,7 \quad (\ \ .473\,156\,604\,979\,478\,6),\\
&z = \ \ .193\,159\,490\,987\,306\,6 \quad (\ \ .193\,159\,492\,443\,962\,3).
\end{aligned}
$$

The results conform with the accuracy requirement. But remember that after many steps the accumulated truncation error will increase without bound.

Program the method in the way that you think is best. Debug it carefully. Run some cases using elliptic orbits with high eccentricity in order to appreciate the luxury of the stepsize control.

A more detailed account of this method, together with flowcharts and exercises for debugging, can be found in the author's "Computing Applications to Differential Equations", Ref. 52. This reference also includes listings in FORTRAN and PASCAL. I recommend that you read Fehlberg's papers, if only to appreciate the genius that was responsible for this very elegant algorithm.

```
10       REM RUNGE-KUTTA-FEHLBERG 4(5).
20       REM THE PROGRAM SOLVES A SYSTEM OF NQ FIRST-ORDER
30       REM DIFFERENTIAL EQUATIONS.  IT GIVES RESULTS OF
40       REM FIFTH ORDER ACCURACY. SEE NASA TR R315 (1969).
50 DEFDBL A-H,O-Z : DEFINT I-N
60 DIM A(6), B(6,7), CH(6), X(6), Z(6), XT(6), TE(6), F(6,6)
70       REM SUITABLE FOR SIX SIMULTANEOUS FIRST-ORDER EQUATIONS
80 A(2) = 2#/9# : A(3) = 1#/3# : A#(4) = 3#/4#
90 A(5) = 1# : A(6) = 5#/6#
100 B(2,1) = 2#/9# : B(3,1) = 1#/12# : B(3,2) = 1#/4#
110 B(4,1) = 69#/128# : B(4,2) =  - 243#/128# : B(4,3) = 135#/64#
120 B(5,1) =  - 17#/12# : B(5,2) = 27#/4# : B(5,3) =  - 27#/5#
130 B(5,4) = 16#/15# : B(6,1) = 65#/432# : B(6,2) =  - 5#/16#
140 B(6,3) = 13#/16# : B(6,4) = 4#/27# : B(6,5) = 5#/144#
150 CH(1) = 47#/450# : CH(2) = 0# : CH(3) = 12#/25#
160 CH(4) = 32#/225# : CH(5) = 1#/30# : CH(6) = 6#/25#
170 CT(1) =  - 1#/150# : CT(2) = 0# : CT(3) = 3#/100#
180 CT(4) =  - 16#/75# : CT(5) =  - 1#/20# : CT(6) = 6#/25#
190      REM THIS COMPLETES THE LISTING OF THE COEFFICIENTS.
200 NQ = 6 : REM NUMBER OF EQUATIONS IN THE SYSTEM.
```



```
210 TL = .0000001#
220      REM GREATEST TOLERABLE LOCAL TRUNCATION ERROR.
230 H = .1# : REM SUGGESTED STEPSIZE FOR THE FIRST STEP.
240 T = 0# : X(1) = 1# : X(2) = .1# : X(3) = - .1#
250 X(4) = - .1# : X(5) = 1# : X(6) = .2#
260      REM INITIAL CONDITIONS. HERE X(1), X(2) AND X(3) ARE
270      REM COMPONENTS OF POSITION; X(4), X(5) AND X(6) ARE
280      REM COMPONENTS OF VELOCITY.
290 TF = 1# : REM FINAL TIME FOR THE INTEGRATION.
300      REM THE PROGRAM IS SET UP SO THAT THE INTEGRATION
310      REM ENDS PRECISELY AT TIME TF.
320      REM
400      REM NOW AN INTEGRATION STEP IS STARTED.
410 TT = T : REM SAVE THE TIME FOR THE START OF THE STEP.
420 GOSUB 1000
430 FOR N = 1 TO NQ
440    F(1,N) = Z(N)
450    XT(N) = X(N)
460 NEXT N
470 REM RETURN HERE WITH A SMALLER STEPSIZE IF NECESSARY.
480 FOR K = 2 TO 6
490   T = TT + A(K)*H
500   FOR N = 1 TO NQ
510      X(N) = XT(N) : K1 = K - 1
520      FOR L = 1 TO K1
530         X(N) = X(N) + H*B(K,L)*F(L,N)
540         NEXT L
550      NEXT N
560   GOSUB 1000
570   FOR N = 1 TO NQ
580      F(K,N) = Z(N)
590      NEXT N
600   NEXT K
610 FOR N = 1 TO NQ
620   TE(N) = 0 : X(N) = XT(N)
630   FOR K = 1 TO 6
640      X(N) = X(N) + H*CH(K)*F(K,N)
650      TE(N) = TE(N) + H*CT(K)*F(K,N)
660      NEXT K
670   TE(N) = ABS(TE(N))
680 NEXT N
690 TM = 0#
700 FOR N = 1 TO NQ
710   IF TM > TE(N) THEN 730
720   TM = TE(N)
730   NEXT N
740      REM TM IS THE TEMAX OF THE TEXT.
750 HT = H
760 H = .9#*H*(TL/TM)^.2# : REM NEW STEPSIZE.
770 IF TM > TL THEN 470
780      REM THE STEP MUST BE REPEATED WITH THE NEW, REDUCED, H.
790 T = TT + HT
800      REM A SUCCESSFUL STEP HAS BEEN COMPLETED.
810 PRINT "TIME = " ; T
820 PRINT "COMPONENTS OF POSITION ARE " ; X(1), X(2), X(3)
830 PRINT "COMPONENTS OF VELOCITY ARE " ; X(4), X(5), X(6)
840 IF H < 0 THEN 880 : REM THE FINAL STEP HAS BEEN TAKEN.
850 IF T < TF THEN 410
860 H = TF - T : GOTO 410
870      REM READY FOR THE FINAL STEP. H WILL BE NEGATIVE.
880 END
1000     REM THIS SUBROUTINE CALCULATES THE RIGHT-HAND SIDES
1010     REM OF THE DIFFERENTIAL EQUATIONS AND PUTS THEM
1020     REM INTO THE ARRAY Z.
1030 Z(1) = X(4) : Z(2) = X(5) : Z(3) = X(6)
1040 R = SQR(X(1)*X(1) + X(2)*X(2) + X(3)*X(3)) : R3 = R*R*R
1050 Z(4) = - X(1)/R3 : Z(5) = - X(2)/R3 : Z(6) = - X(3)/R3
1060 RETURN
```

10.7 The Numerical Integration of Differential Equations—A Multistep Method for First-Order Systems

We start once more with the differential equation

$$\frac{dx}{dt} = f(t,x); \quad x(t_0) = x_0. \tag{10.7.1}$$

The solution (and we assume, of course, existence and uniqueness in any domain in which we are working) can be formally written as

$$x = x(t), \tag{10.7.2}$$

and for any time t_i, we have the formal relation

$$x(t_i) - x(t_0) = \int_{t_0}^{t_i} f[t, x(t)]dt. \tag{10.7.3}$$

This will be applied by replacing the integrand by a polynomial that interpolates a set of values of f. In order to do this, we must know some of the past history of f, so the method cannot start until a set of values is available:

$$t_i = t_0 + ih, \quad x_i, \quad f_i = f(t_i, x_i), \quad i = 1, 2, \ldots, k.$$

h is the stepsize to be used, and k is related to the order of the method: the more values used, the higher the degree of the interpolating polynomial, and, up to a point, the greater the accuracy of the method. In the program that follows, these initial values are found through the use of the Runge-Kutta algorithm (10.6.2): slow, but safe, provided that the stepsize for the Runge-Kutta start is small enough compared with the stepsize for the regular integration. Fehlberg's method could also be used. In any event, care should be taken over these starting values. We mention also that k is an `INPUT` parameter in the program that follows.

A regular step in the program has two components, *prediction* and *correction*. Let us start with a set of values for $f_1, f_2, \ldots, f_k$. These can be fitted to an interpolating polynomial of degree $(k-1)$. Call this $P_p(t)$, where

$$P_p(t) = p_1(t)f_1 + p_2(t)f_2 + \cdots + p_k(t)f_k.$$

The value of x_k at time t_k is, of course, known; we need to approximate x_{k+1} at time $t_{k+1} = t_k + h$. This is done by using (10.7.3) in the form

$$\begin{aligned} x^a_{k+1} &= x_k + \int_{t_k}^{t_{k+1}} P_p(t)dt \\ &= x_k + \int_{t_k}^{t_k+h} [p_1(t)f_1 + \cdots + p_k(t)f_k]dt \\ &= x_k + h[b_k f_1 + b_{k-1}f_2 + \cdots + b_1 f_k]. \end{aligned} \tag{10.7.4}$$

This completes the *prediction*.

After x^a_{k+1} is found, $f^a_{k+1} = f(t_{k+1}, x^a_{k+1})$ is computed, and the set: $f_2, f_3, \ldots, f_k, f^a_{k+1}$ is the basis for an interpolating polynomial $P_c(t)$. A *corrected* value, x^c_{k+1}, is then found from

$$x^c_{k+1} = x_k + \int_{t_k}^{t_{k+1}} P_c(t)dt. \tag{10.7.5}$$

This leads to a corrected value for f_{k+1}, and a corrected interpolating polynomial, and (10.7.5) can be used again. Thus correction can be viewed as a recursive operation; if all goes well, the recursion will lead to an acceptable value for x_{k+1}, and the step is complete.

The preceding paragraph describes the principle of the correction. To describe the detail that will be used in the program, the notation will be changed. After prediction, the subscripts are shifted, with the old f_1 abandoned, and $i \leftarrow i+1$. So

$$\begin{gathered} f_1 \leftarrow f_2, \quad f_2 \leftarrow f_3, \quad \ldots, \quad f_{k-1} \leftarrow f_k, \quad f_k \leftarrow f^a_{k+1}, \\ t_{k-1} \leftarrow t_k, \quad t_k \leftarrow t_{k+1}. \end{gathered} \tag{10.7.6}$$

(In the program, the final f-switch is delayed.) Then (10.7.5) is modified to

$$\begin{aligned} x^c_k &= x_{k-1} + \int_{t_{k-1}}^{t_k} P_c(t)dt \\ &= x_{k-1} + h[b^*_k f_1 + b^*_{k-1}f_2 + \cdots + b^*_2 f_{k-1} + b^*_1 f_k]. \end{aligned} \tag{10.7.7}$$

The coefficients b_i and b^*_i are computed in the program; they depend on k as well as i, but are, of course, independent of h. When (10.7.7) is used recursively, only the final term changes from one iteration to the next, simplifying the programing.

The prediction and correction procedures are respectively known as the *Adams-Bashforth* and *Adams-Moulton* methods. Technical details can be found in references such as 50. The formulas for the coefficients may be summarized as follows. Introduce g_i and g^*_i, $i = 1, 2, \ldots, k$, where $g_1 = g^*_1 = 1$, and

$$\left.\begin{aligned} g_i + \frac{1}{2}g_{i-1} + \frac{1}{3}g_{i-2} + \cdots + \frac{1}{i+1}g_1 = 1, \\ g^*_i + \frac{1}{2}g^*_{i-1} + \frac{1}{3}g^*_{i-2} + \cdots + \frac{1}{i+1}g^*_1 = 0. \end{aligned}\right\} \quad i = 2, 3, \ldots, k. \tag{10.7.8}$$

(These would be actual coefficients if backward differences were to be used.)

Then

$$b_m = \sum_{j=m}^{k} p_{mj} g_j \quad \text{and} \quad b_m^* \sum_{j=m}^{k} p_{mj} g_j^*,$$

where

$$\left.\begin{aligned} p_{mj} &= (-1)^{m-1}\binom{j-1}{m-1} \\ &= (-1)^{m-1}\frac{(j-1)(j-2)\cdots(j-m+1)}{1\cdot 2\cdot\cdots\cdot(m-1)}. \end{aligned}\right] \tag{10.7.9}$$

The stepsize in the Runge-Kutta starting procedure should be smaller than the regular stepsize; if this is H, then a rule of thumb for that of the Runge-Kutta section is

$$HFK = \frac{H}{2^{(k-4)}}.$$

You should experiment with this. The ratio of the two stepsizes, RK, is an INPUT parameter of the program. In the program the corrector is iterated for up to three times. As it stands, the quantity used in line 2320 to test whether another iteration should be made is too small for the stepsize used. Here again, you should run experiments. When experimenting, be generous with your printout.

The program was tested with the same orbit that was used in the preceding section. Results for time $t = 1$ were:

$$\begin{aligned} x &= .\underline{466\,080\,784}\,635\,792\,5, \\ y &= .\underline{900\,611\,240}\,900\,481\,7, \\ z &= .\underline{114\,045\,981}\,796\,828\,2, \\ x &= -.\underline{876\,594\,448}\,229\,200\,6, \\ y &= .\underline{473\,156\,610}\,981\,887\,4, \\ z &= .\underline{193\,159\,494}\,924\,814\,5. \end{aligned}$$

The good figures are underlined. The accuracy is not spectacular; you may find it worth the programing effort to rewrite the method using backward differences. Here is the program that generated these figures:

```
10       REM PREDICTOR-CORRECTOR METHOD WITH RUNGE-KUTTA START.
20 DEFDBL A-H, O-Z : DEFINT I-N
30 DIM X(6), Z(6), F(6,10), XT(6), R1(6), R2(6), R3(6), SUM(6)
40 K = 7
50       REM K IS THE PARAMETER FOR THE ORDER OF THE METHOD.
60 RK = 8#
70       REM RK IS THE NUMBER OF RUNGE-KUTTA STEPS PER REGULAR
80       REM STEPSIZE TO BE USED IN THE STARTING PROCEDURE.
90 H = .1# : REM STEPSIZE.
100 TMAX = 1# : REM FINAL TIME FOR THE INTEGRATION.
110 NQ = 6 : REM NUMBER OF EQUATIONS IN THE SYSTEM.
120 T = 0# : X(1) = 1# : X(2) = .1# : X(3) = - .1#
130 X(4) = - .1# : X(5) = 1# : X(6) = .2#
140      REM INITIAL CONDITIONS.
150 PRINT "TIME " ; T
160 FOR N = 1 TO NQ
170    PRINT "X("; N ;") = " ; X(N)
180 NEXT N : PRINT
190 GOSUB 500 : REM FIND COEFFICIENTS.
200 GOSUB 1500 : REM RUNGE-KUTTA START.
```



```
210 GOSUB 2000 : REM PREDICTOR-CORRECTOR STEP.
220 PRINT : PRINT "TIME " ; T
230 FOR N = 1 TO NQ
240    PRINT "X("; N ;") = " ; X(N)
250 NEXT N
260 IF T >= TMAX THEN 280
270 GOTO 210
280 END
500 REM CALCULATE COEFFICIENTS FOR THE INTEGRATION.
510 FOR J = 1 TO K
520    B(J) = 0#
530    BS(J) = 0#
540    P(1,J) = 1#
550 NEXT J
560 FOR I = 2 TO K
570    FOR J = I TO K
580       FI = I
590       FJ = J
600       P(I,J) = - P(I - 1,J)*(FJ - FI + 1#)/(FI - 1#)
610    NEXT J
620 NEXT I
630 G(1) = 1# : GG(1) = 1#
640 FOR I = 2 TO K
650    G(I) = 1# : GG(I) = 0#
660    FOR J = 2 TO I
670       FJ = J
680       G(I) = G(I) - G(I - J + 1)/FJ
690       GG(I) = GG(I) - GG(I - J + 1)/FJ
700    NEXT J
710 NEXT I
720 FOR M = 1 TO K
730    FOR J = M TO K
740       B(M) = B(M) + P(M,J)*G(J)
750       BS(M) = BS(M) + P(M,J)*GG(J)
760    NEXT J
770 NEXT M
780 RETURN
1000 REM RIGHT-HAND SIDES OF THE EQUATIONS
1010 Z(1) = X(4) : Z(2) = X(5) : Z(3) = X(6)
1020 R = SQR(X(1)*X(1) + X(2)*X(2) + X(3)*X(3)) : R3 = R*R*R
1030 Z(4) = - X(1)/R3 : Z(5) = - X(2)/R3 : Z(6) = - X(3)/R3
1040 RETURN
1500 REM RUNGE-KUTTA START
1510 HRK = H/RK : REM REDUCED STEPSIZE FOR THIS SUBROUTINE.
1520 GOSUB 1000
1530 FOR N = 1 TO NQ
1540    F(N,1) = Z(N)
1550 NEXT N
1560 FOR L = 2 TO K
1570    FOR M = 1 TO RK
1580       FOR N = 1 TO NQ
1590          XT(N) = X(N)
1600          R1(N) = HRK*Z(N)
1610          X(N) = XT(N) + R1(N)/2#
1620       NEXT N
1630       T = T + HRK/2# : GOSUB 1000
1640       FOR N = 1 TO NQ
1650          R2(N) = HRK*Z(N)
1660          X(N) = XT(N) + R2(N)/2#
1670       NEXT N
1680       GOSUB 1000
1690       FOR N = 1 TO NQ
1700          R3(N) = HRK*Z(N)
1710          X(N) = XT(N) + R3(N)
1720       NEXT N
1730       T = T + HRK/2# : GOSUB 1000
1740       FOR N = 1 TO NQ
1750          X(N) = XT(N) + (R1(N) + 2#*(R2(N) + R3(N))
                    + HRK*Z(N))/6#
1760       NEXT N
1770       GOSUB 1000
1780    NEXT M
1790    FOR N = 1 TO NQ
1800       F(N,L) = Z(N)
1810    NEXT N
```

```
1820    PRINT : PRINT "TIME " ; T
1830    FOR N = 1 TO NQ
1840       PRINT "X("; N ;") = " ; X(N)
1850    NEXT N
1860 NEXT L
1870 RETURN
2000    REM PREDICTOR-CORRECTOR.
2010    REM PREDICTION.
2020 FOR N = 1 TO NQ
2030    Z = 0#
2040    FOR J = 1 TO K
2050       Z = Z + B(J)*F(N,K - J + 1)
2060    NEXT J
2070    XT(N) = X(N)
2080    X(N) = XT(N) + H*Z
2090 NEXT N
2100    REM SHIFT.
2110 FOR N = 1 TO NQ
2120    FOR J = 1 TO K - 1
2130       F(N,J) = F(N,J + 1)
2140    NEXT J
2150 NEXT N
2160 REM CORRECTION.
2170 T = T + H
2180 FOR N = 1 TO NQ
2190    SUM(N) = 0#
2200    FOR J = 2 TO K
2210       SUM(N) = SUM(N) + BS(J)*F(N,K - J + 1)
2220    NEXT J
2230    SUM(N) = XT(N) + H*SUM(N)
2240 NEXT N
2250 FOR NC = 1 TO 3
2260    P = 0# : GOSUB 1000
2270    FOR N = 1 TO NQ
2280       CORR = SUM(N) + H*BS(1)*Z(N)
2290       P = P + ABS(X(N) - CORR)
2300       X(N) = CORR
2310    NEXT N
2320    IF P < 1E-11 THEN 2350
2330 NEXT NC
2340 PRINT "CONVERGENCE PROBLEM. P = " ; P
2350 FOR N = 1 TO NQ
2360    F(N,K) = Z(N)
2370 NEXT N
2380 RETURN
```

10.8 The Numerical Integration of Differential Equations—Systems of Second-Order Equations

This section is based on the method traditionally used in celestial mechanics. It is credited to Gauss, although often referred to as the "Gauss-Jackson" method. For much orbital work it is arguably without rival. As an introduction, consider the single equation

$$\frac{d^2x}{dt^2} = F(x, \frac{dx}{dt}, t), \tag{10.8.1}$$

or $\ddot{x} = F(x, \dot{x}, t)$. At some time, t_i, there will be values $x_i, \dot{x}_i$, and

$$F_i = h^2 F(x_i, \dot{x}_i, t_i), \tag{10.8.2}$$

where h is the stepsize. So $t_i = t_0 + ih$, t_0 being the starting time.

In terms of central differences, the basic formulas are

$$x_i = \delta^{-2}F_i + \tfrac{1}{12}F_i - \tfrac{1}{240}\delta^2 F_i + \tfrac{31}{60480}\delta^4 F_i - \tfrac{289}{3628800}\delta^6 F_i + \cdots \tag{10.8.3}$$

and

$$h\dot{x}_i = \delta^{-1}F_{i+\frac{1}{2}} - \tfrac{1}{2}F_i - \tfrac{1}{12}\mu\delta F_i + \tfrac{11}{720}\mu\delta^3 F_i - \tfrac{191}{60480}\mu\delta^5 F_i + \cdots \tag{10.8.4}$$

For the sake of the program, these formulas will be rewritten to use backward differences. We start by considering a typical "step" once the method is established. As usual, we have *prediction* followed by *correction*. Assume that a step has just been satisfactorily completed, giving "good" values for the time t_i. The most up-to-date information will be stored in the differences and sums:

Table 10.9

						$\nabla^6 F_i$
					$\nabla^5 F_i$	
				$\nabla^4 F_i$		
			$\nabla^3 F_i$			
		$\nabla^2 F_i$				
	∇F_i					
F_i						
	$\nabla^{-1} F_i$					
$\nabla^{-2} F_i$						

This could be taken further (and the program will be capable of using more terms) but this table will be good enough for discussion. x_i and $\dot{x}_i$ are known for time t_i. We *predict*

$$\begin{aligned} x^a_{i+1} = {} & \nabla^{-2}F_i + \tfrac{1}{12}F_i + \tfrac{1}{12}\nabla F_i + \tfrac{19}{240}\nabla^2 F_i + \tfrac{18}{240}\nabla^3 F_i \\ & + \tfrac{863}{12096}\nabla^4 F_i + \tfrac{825}{12096}\nabla^5 F_i + \tfrac{237671}{3628800}\nabla^6 F_i, \end{aligned} \tag{10.8.5}$$

and

$$\begin{aligned} h\dot{x}^a_{i+1} = {} & \nabla^{-1}F_i + \tfrac{1}{2}F_i + \tfrac{5}{12}\nabla F_i + \tfrac{3}{8}\nabla^2 F_i + \tfrac{251}{720}\nabla^3 F_i \\ & + \tfrac{95}{288}\nabla^4 F_i + \tfrac{19087}{60480}\nabla^5 + \tfrac{36799}{120960}\nabla^6 F_i. \end{aligned} \tag{10.8.6}$$

With these, we can calculate

$$F^a_{i+1} = h^2 F(x^a_{i+1}, \dot{x}^a_{i+1}, t_{i+1}) \tag{10.8.7}$$

and move down the table, finding the next diagonal of differences and sums:

Table 10.10

						$\nabla^6 F^a_{i+1}$
					$\nabla^5 F^a_{i+1}$	
				$\nabla^4 F^a_{i+1}$		
			$\nabla^3 F^a_{i+1}$			
		$\nabla^2 F^a_{i+1}$				
	$\nabla^1 F^a_{i+1}$					
$\nabla^{-2} F_i$		F^a_{i+1}				
	$\nabla^{-1} F_{i+1}$					

Next we find corrected values, x^b_{i+1} and $\dot{x}^b_{i+1}$, from

$$x^b_{i+1} = \nabla^{-2}F_i + \tfrac{1}{12}F^a_{i+1} - \tfrac{1}{240}\nabla^2 F^a_{i+1} - \tfrac{1}{240}\nabla^3 F^a_{i+1} - \tfrac{221}{60480}\nabla^4 F^a_{i+1} - \tfrac{190}{60480}\nabla^5 F^a_{i+1} - \tfrac{9829}{3628800}\nabla^6 F^a_{i+1}, \quad (10.8.8)$$

and

$$h\dot{x}^b_{i+1} = \nabla^{-1}F^a_{i+1} - \tfrac{1}{2}F^a_{i+1} - \tfrac{1}{12}\nabla F^a_{i+1} - \tfrac{1}{24}\nabla^2 F^a_{i+1} - \tfrac{19}{720}\nabla^3 F^a_{i+1} - \tfrac{3}{160}\nabla^4 F^a_{i+1} - \tfrac{863}{60480}\nabla^5 F^a_{i+1} - \tfrac{1375}{120960}\nabla^6 F^a_{i+1}. \quad (10.8.9)$$

A new F can now be computed, using an obvious modification of (10.8.7), and the differences adjusted. Traditionally, the stepsize has been chosen so that correction was not needed; but provided that more decimal places are carried in the computation than are needed in the solution, the difference $(x^b - x^a)$ will be informative. For now, provided that the difference is smaller than some preset tolerance, the step will have been completed.

We now change the notation in order to generate a program. Table 10.9 becomes:

Table 10.11

								$D(M)$
							$\cdots$	
						$D(2)$		
					$D(1)$			
				F				
			$S1$					
		$S2$						

The predicted values for x and $\dot{x}$ are found from

$$\left.\begin{aligned} XA &= S2 + A(0,2)*F + \sum_{N=1}^{M} A(0,N+2)*D(N), \\ H*XVA &= S1 + B(0,1)*F + \sum_{N=1}^{M} B(0,N+1)*D(N). \end{aligned}\right] \quad (10.8.10)$$

The time is increased, so that

$$T = T + H,$$

and then

$$FA = H^2 * F(XA, XVA, T) \quad (10.8.11)$$

and the new differences, and a new first sum are found. We then have the situation of Table 10.9, which, in the new notation, is

Table 10.12

								$D(M)$
							$\cdots$	
						$D(2)$		
					$D(1)$			
$S2$				F				
	$S1$							

The corrections follow:

$$\left.\begin{aligned} XB &= S2 + A(1,2)*FA + \sum_{N=1}^{M} A(1,N+2)*D(N), \\ H*XVB &= S1 + B(1,1)*FA + \sum_{N=1}^{M} B(1,N+1)*D(N). \end{aligned}\right] \quad (10.8.12)$$

If the difference $(XB - XA)$ is small enough, then we find

$$FB = H^2 * F(XB, XVB, T). \quad (10.8.13)$$

Increase each of $S1, D(1), \ldots, D(M)$ by the amount $(FB - FA)$, and find the new second sum from $S2 = S2 + S1$. Set $F = FB$, and the step is complete.

If XB is not satisfactory, further adjustments, DX and DXV, can be made, where

$$\left.\begin{aligned} DX &= (FB - FA)*[A(1,2) + A(1,3) + \cdots + A(1,M+2)], \\ H*DXV &= (FB - FA)*[B(1,0) + B(1,1) + \cdots + B(1,M+1)]. \end{aligned}\right] \quad (10.8.14)$$

So continued correction is no great burden.

There are several schools of thought about the best way to set up the initial table, before the first prediction can be made. The most obvious way is to use a single-step method, as in the preceding section. Alternatives involve iteration procedures. We shall look at just one. We assume that the equations to be solved can be written as

$$\ddot{\mathbf{r}} = \mathbf{F}_0 + \mathbf{F}_1, \quad (10.8.15)$$

where $\ddot{\mathbf{r}} = \mathbf{F}_0 = -\mu\mathbf{r}/r^3$ would yield a Keplerian orbit and $|\mathbf{F}_1| \ll |\mathbf{F}_0|$. In orbital work this is, of course, usually the case. In setting out the formulas, I shall continue to use the notation for just one equation, (10.8.1); the generalization to three should be obvious.

The initial table is set out using the approximate known solution, corresponding to the given initial conditions, $X0$ and $XV0$ at time $T0$. Let

$$\left.\begin{aligned} &F(0) = H^2 * F(X0, XV0, T0). \\ &\quad \text{For each time } TN = T0 - N*H, \text{ calculate } XN, XVN \text{ from the approximate solution and let} \\ &F(N) = H^2 * F(XN, XVN, TN), \quad N = 1, 2, \ldots, M. \end{aligned}\right] \quad (10.8.16)$$

Next, find the backward differences, $D(1), D(2), \ldots, D(M)$, and use the formulas (10.8.12) to find the sums:

$$\left.\begin{aligned} S1 &= H*XV0 - B(1,1)*F(0) - \sum_{N=1}^{M} B(1,N+1)*D(N), \\ S2 &= S1 + X0 - A(1,2)*F(0) - \sum_{N=1}^{M} A(1,N+2)*D(N). \end{aligned}\right] \quad (10.8.17)$$

This table is now used to calculate new values for X, XV for each time, and these generate corrected values for $F(N)$. Specifically, for $N = 1, 2, \ldots, M$, find $TN = T0 - N * H$,

$$\left.\begin{aligned} XN &= S2 + A(N+1,1) * S1 + A(N+I,2) * F(0) \\ &\quad + \sum_{I=1}^{M} A(N+1, I+2) * D(N), \\ H * XVN &= S1 + B(N+1,1) * F(0) \\ &\quad + \sum_{I=1}^{M} B(N+1, I+1) * D(N), \\ F(N) &= H^2 * F(XN, XVN, TN). \end{aligned}\right\} \tag{10.8.18}$$

As each $F(N)$ is found the difference between it and its predecessor should be calculated, and the sum of the absolute values of all the differences computed. With the completion of a new set of $F(N)$, fresh differences and sums are found, and another iteration can be made. This continues until the changes in the $F(N)$ become acceptably small.

The coefficients are found from the following formulas:

$$\left.\begin{aligned} &A(J,0) = 1, \\ &A(J,N) = (-1)^N \binom{J-1}{N} - \frac{2}{3}(1+\tfrac{1}{2})A(J,N-1) \\ &\qquad - \frac{2}{4}(1+\tfrac{1}{2}+\tfrac{1}{3})A(J,N-2) - \cdots \\ &\qquad \cdots - \frac{2}{N+2}(1+\tfrac{1}{2}+\cdots+\tfrac{1}{N+1})A(J,0), \\ &B(J,0) = 1, \\ &B(J,N) = (-1)^N \binom{J-1}{N} - \frac{1}{2}B(J,N-1) - \frac{1}{3}B(J,N-2) - \cdots \\ &\qquad \cdots - \frac{1}{N+1}B(J,0). \end{aligned}\right\} \tag{10.8.19}$$

A derivation of these can be found in the NASA Technical Memorandum 33-451, which contains also a wealth of other useful information.

The starting procedure can be improved, with a little more programing effort, if the initial time is in the *middle* of the starting values. Suppose, for instance, that $M = 6$; then $F(3)$ would correspond to $H^2 * F(X0, XV0, T0)$, and the initial sums found would be $\nabla^{-1}F_3$ and $\nabla^{-2}F_2$, using implicitly or explicitly the central difference formulas (10.8.3) and (10.8.4).

Another starting procedure begins with $F = H^2 * F(X0, XV0, T0)$, and all of the differences equal to zero. $S1$ and $S2$ are found using the initial conditions, and M steps are taken. There will then be a non-zero value for $D(M)$. The preceding Mth differences are all put equal to this value, and the differences

extrapolated back to the initial diagonal. The process starts again with the numbers in this diagonal and F being the starting value as before. $S1$ and $S2$ are calculated again, and M steps taken. This continues until the magnitudes of the values of X all change by amounts less than some given tolerance between two iterations.

The listing that follows is not general, but was designed to test the basic formulas using the known results for Keplerian motion. The complete program included a subroutine (starting on line 2500) that solves the initial value problem of Keplerian motion using universal variables; this is similar to programs listed earlier, and is not included here. The starting values were "exact" so no iteration was required. Consequently, $A(J,N)$, $B(J,N)$ are only needed when $J = 0$, 1. $F(0)$ corresponds to the starting time, t_0; as remarked above, the accuracy is increased if the F for the starting time is set in the middle of the table. Velocities are not calculated at each step since the right-hand sides of the equations depend on position only.

Program the method in your own way; but I recommend that you delay experiments with starting procedures until you are sure of the regular steps. Experiment with the order of the method, and the stepsize; also decide whether or not you think it is worthwhile to include the correction cycle.

```
10     REM PREDICTOR-CORRECTOR METHOD FOR A SYSTEM
20     REM OF SECOND-ORDER EQUATIONS. FOR THIS
30     REM LISTING THE DIFFERENTIAL EQUATIONS ARE
40     REM FOR KEPLERIAN MOTION, AND THE STARTING
50     REM VALUES ARE FOUND BY THE FORMULAS OF
60     REM KEPLERIAN MOTION. THESE ARE NOT LISTED.
70     REM
80 DEFDBL A-H, O-Z : DEFINT I-N
90 DIM A(12,12), B(12,12), D(10), FX(10), FY(10)
100 DIM FZ(10), DX(10), DY(10), DZ(10)
110 GMU = 1#
120    REM "MU" FOR THE EQUATIONS OF MOTION. SEE LINE 1510.
130 H = .1# : REM STEPSIZE.
140 M = 8 : REM ORDER OF THE INTEGRATION.
150 T0 = 0# : X0 = 1# : Y0 = .1# : Z0 =  - .1#
160 XV0 =  - .1# : YV0 = 1# : ZV0 = .2#
170    REM INITIAL CONDITIONS.
180    REM INITIALIZE THE INTEGRATION TABLE.  SINCE THIS
190    REM IS A TEST RUN WITH PURE KEPLERIAN MOTION, THE
200    REM INITIAL VALUES ARE FOUND FROM EXACT FORMULAS.
210    REM THE FORMULAS ARE CONTAINED IN A SUBROUTINE
220    REM STARTING AT LINE 2500, AND ARE NOT LISTED.
230 X = X0 : Y = Y0 : Z = Z0
240 XV = XV0 : YV = YV0 : ZV = ZV0 : GOSUB 1500
250    REM FIND TABULAR VALUES.
260 FX(0) = H*H*XF
270 FY(0) = H*H*YF
280 FZ(0) = H*H*ZF
290 FOR I = 1 TO M : FI = I
300    T = T0 - FI*H
310    GOSUB 2500 : GOSUB 1500
320    PRINT : PRINT T : PRINT X, Y, Z
330    FX(I) = H*H*XF
340    FY(I) = H*H*YF
350    FZ(I) = H*H*ZF
360 NEXT I
370    REM FIND DIFFERENCES.
380 FOR I = 1 TO M
390    DX(M + 1 - I) = FX(M - I) - FX(M + 1 - I)
400    DY(M + 1 - I) = FY(M - I) - FY(M + 1 - I)
410    DZ(M + 1 - I) = FZ(M - I) - FZ(M + 1 - I)
420 NEXT I
430 FOR J = 2 TO M
```

```
440     FOR I = J TO M
450        DX(M + J - I) = DX(M - 1 + J - I) - DX(M + J - I)
460        DY(M + J - I) = DY(M - 1 + J - I) - DY(M + J - I)
470        DZ(M + J - I) = DZ(M - 1 + J - I) - DZ(M + J - I)
480     NEXT I
490 NEXT J
500 GOSUB 2000 : REM CALCULATE COEFFICIENTS.
510     REM FIND SUMS.
520 FX = FX(0) : FY = FY(0) : FZ = FZ(0)
530 S1X = H*XV0 - B(1,1)*FX
540 S1Y = H*YV0 - B(1,1)*FY
550 S1Z = H*ZV0 - B(1,1)*FZ
560 S2X = X0 - A(1,2)*FX
570 S2Y = Y0 - A(1,2)*FY
580 S2Z = Z0 - A(1,2)*FZ
590 FOR N = 1 TO M
600     S1X = S1X - B(1,N + 1)*DX(N)
610     S1Y = S1Y - B(1,N + 1)*DY(N)
620     S1Z = S1Z - B(1,N + 1)*DZ(N)
630     S2X = S2X - A(1,N + 2)*DX(N)
640     S2Y = S2Y - A(1,N + 2)*DY(N)
650     S2Z = S2Z - A(1,N + 2)*DZ(N)
660 NEXT N
670 S2X = S1X + S2X
680 S2Y = S1Y + S2Y
690 S2Z = S1Z + S2Z
700 T = T0
710     REM PREDICTION.
720 T = T + H
730 XA = S2X + A(0,2)*FX
740 YA = S2Y + A(0,2)*FY
750 ZA = S2Z + A(0,2)*FZ
760 FOR N = 1 TO M
770     XA = XA + A(0,N + 2)*DX(N)
780     YA = YA + A(0,N + 2)*DY(N)
790     ZA = ZA + A(0,N + 2)*DZ(N)
800 NEXT N
810 PRINT : PRINT "PREDICTION. TIME " ; T
820 PRINT "COORDINATES " ; XA, YA, ZA
830     REM FIND NEW DIFFERENCES.
840 X = XA : Y = YA : Z = ZA : GOSUB 1500
850 FXA = H*H*XF
860 FYA = H*H*YF
870 FZA = H*H*ZF
880 S1X = S1X + FXA
890 S1Y = S1Y + FYA
900 S1Z = S1Z + FZA
910 DX(0) = FXA - FX
920 DY(0) = FYA - FY
930 DZ(0) = FZA - FZ
940 FOR I = 1 TO M - 1
950     DX(I) = DX(I - 1) - DX(I)
960     DY(I) = DY(I - 1) - DY(I)
970     DZ(I) = DZ(I - 1) - DZ(I)
980 NEXT I
990 FOR I = 1 TO M
1000    DX(M + 1 - I) = DX(M - I)
1010    DY(M + 1 - I) = DY(M - I)
1020    DZ(M + 1 - I) = DZ(M - I)
1030 NEXT I
1040     REM CORRECTION.
1050 XB = S2X + A(1,2)*FXA
1060 YB = S2Y + A(1,2)*FYA
1070 ZB = S2Z + A(1,2)*FZA
1080 FOR N = 1 TO M
1090    XB = XB + A(1,N + 2)*DX(N)
1100    YB = YB + A(1,N + 2)*DY(N)
1110    ZB = ZB + A(1,N + 2)*DZ(N)
1120 NEXT N
1130 X = XB : Y = YB : Z = ZB : GOSUB 1500
1140 FX = H*H*XF
1150 FY = H*H*YF
1160 FZ = H*H*ZF
1170 DFX = FX - FXA
1180 DFY = FY - FYA
```



```
1190 DFZ = FZ - FZA
1200 S1X = S1X + DFX
1210 S1Y = S1Y + DFY
1220 S1Z = S1Z + DFZ
1230 S2X = S2X + S1X
1240 S2Y = S2Y + S1Y
1250 S2Z = S2Z + S1Z
1260 FOR I = 1 TO M
1270    DX(I)=DX(I) + DFX
1280    DY(I)=DY(I) + DFY
1290    DZ(I)=DZ(I) + DFZ
1300 NEXT I
1310    REM CORRECTION COMPLETED.
1320 PRINT : PRINT "CORRECTION. TIME " ; T
1330 PRINT "COORDINATES " ; X, Y, Z
1340 IF T < 1# THEN 710
1350 END
1500   REM RIGHT-HAND SIDES OF THE EQUATIONS.
1510 R = SQR(X*X + Y*Y + Z*Z) : R3 = R*R*R
1520 XF = - GMU*X/R3 : YF = - GMU*Y/R3 : ZF = - GMU*Z/R3
1530 RETURN
2000 FOR J = 0 TO 1
2010    FJ = J
2020    REM THIS LOOP MUST BE EXTENDED IF THE STARTING
2030    REM VALUES ARE TO BE FOUND THROUGH ITERATION.
2040    A(J,0) = 1# : B(J,0) = 1# : SN = 1#
2050    FOR N = 1 TO M + 2
2060       FLN = N
2070       SI = 1#
2080       SN =  - SN*(FJ - FLN)/FLN
2090       A(J,N) = SN
2100       B(J,N) = SN
2110       FOR K = 1 TO N
2120          FK=K
2130          Z = 1#/(FK + 1#)
2140          SI = SI + Z
2150          B(J,N) = B(J,N) - Z*B(J,N - K)
2160          A(J,N) = A(J,N) - SI*A(J,N - K)*2#/(FK + 2#)
2170       NEXT K
2180    NEXT N
2190 NEXT J
2200 RETURN
2500    REM SUBROUTINE FOR FINDING COMPONENTS OF POSITION
2510    REM AND VELOCITY IN KEPLERIAN MOTION.
```

The coordinates for $t = 1$ for this listing were

$$x = .\underline{466\,080\,784}\,812\,956\,4,$$
$$y = .\underline{900\,611\,234}\,846\,438\,4,$$
$$z = .\underline{114\,045\,980\,6}36\,899\,5.$$