Reduction of Round-off Errors in Symplectic Integrators

Article *in* The Astronomical Journal · January 2001

CITATIONS
2

READS
256

1 author:

Toshio Fukushima
National Astronomical Observatory of Japan
776 PUBLICATIONS **3,863** CITATIONS

SEE PROFILE

THE ASTRONOMICAL JOURNAL, 121:1768–1775, 2001 March

REDUCTION OF ROUND-OFF ERROR IN SYMPLECTIC INTEGRATORS

TOSHIO FUKUSHIMA
National Astronomical Observatory, 2-21-1 Osawa, Mitaka, Tokyo 181-8588, Japan; Toshio.Fukushima@nao.ac.jp
Received 2000 May 16; accepted 2000 November 15

ABSTRACT

Three approaches to reduce the accumulation of round-off errors in symplectic integrators are examined. The first is to reduce the number of summations in the implementation of symplectic algorithms. Although the effect is small, around a 30% reduction of accumulated round-off error, its use is always recommended because it is achieved with no extra computational time. The second approach is the use of Dekker's 1971 double-length addition routine in the main summation procedure. It provides a result that is a few digits more precise for less than a 25% increase in computational time. The third is the full application of Dekker's double-length routine library to the whole procedure, including the force evaluation. This realizes a quasi–quadruple-precision integration at a cost of a 6–13 times increase in CPU time, which is still 4–23 times faster than full quadruple-precision computation. All these reductions of round-off error depend little on the order of the symplectic integrator. Unfortunately, the last two approaches will not be applicable to mixed-variable symplectic integrators unless double-length routines to evaluate trigonometric functions are developed.

Key words: celestial mechanics — methods: numerical — solar system: general

1. INTRODUCTION

In the days of powerful computers, the errors of numerical integration are the main limitation in the research of complex dynamical systems, such as the long-term stability of our solar system and of some exoplanets (Milani & Nobili 1988). As for the long-term behavior, it is well known that orbital integration errors along track grow quadratically with respect to time when using traditional integrators such as the Runge-Kutta methods, the Adams/Cowell multistep methods, and the extrapolation methods (Hairer, Nørsett, & Wanner 1993). On the other hand, many papers have claimed that the growth rate becomes linear if symplectic integrators are used (see Sanz-Serna 1992 for a review). Similar reports were given for the symmetric linear multistep methods (Lambert & Watson 1976; Quinlan & Tremaine 1990; Fukushima 1999; Evans & Tremaine 1999).

Unfortunately, this simply is not true, or more exactly speaking, it is true only in exact arithmetic, which is not available on any real computer. Actual integration errors are a composite of the truncation and the round-off errors. Once the truncation error has been made tiny by means of symplectic integrators or symmetric linear multistep methods, then the round-off error will emerge and start to play a key role (Petit 1998). This is because of its larger growth rate, to the 3/2 power in time (Brouwer 1937; see Fig. 1). Note that this growth rate is independent of the order and type of integrator used.

Thus, it is quite important to reduce round-off error as much as possible in conducting long-term numerical integrations. For example, consider integrating numerically our solar system, consisting of the Sun and nine major planets, for 5×10^9 yr, a period comparable to its entire history. If the step size is set to 1% of the least orbital period,[1] say, 0.0025 yr, then the total number of steps becomes 2×10^{12}. Even if we limit ourselves to the stability of the system of the Sun and the four outer planets, the necessary number of steps amounts to around 5×10^{10}. For these huge numbers of steps, integration in standard double-precision arithmetic with 53-bit mantissas will lead to accumulated round-off error of more than 1 radian. It will cause the loss of any physically meaningful information.

Of course, the ultimate approach to decreasing round-off error is the introduction of higher precision computations. However, such a remedy is not always possible. In fact, quadruple-precision computation is not available on ordinary personal computers. Although it is available on some workstations, it costs quite a bit (see Tables 1 and 2). The CPU time increases by factors of around 30–300 when the arithmetic changes from double to quadruple precision. This is because quadruple-precision computations are usually realized not by hardware but by software. Therefore, we must seriously consider how to decrease round-off error in a more active manner.

To accomplish this, many techniques have been proposed (Milani & Nobili 1988; Quinn & Tremaine 1990; Quinlan 1994). A famous example is the Gauss-Jackson method (Jackson 1924), rewriting the integration of special second-order ordinary differential equations in summed forms. The use of backward-difference forms is another example (Quinlan 1994). For the multistep methods, the introduction of a special `ADD` routine in evaluating the linear forms was proposed to reduce local round-off errors by Quinn & Tremaine (1990), who used it in computing the linear forms of integrals in the symmetric multistep methods. For the extrapolation methods, we learned that changing from variables to their increments reduces global round-off errors significantly with almost no extra increase of computational time (Fukushima 1996b). For the symplectic integrators, use of integer arithmetic was proposed by Earn & Tremaine (1992). They reported a gain of a few digits' precision.

In this paper, we report that the introduction of double-length routines (Dekker 1971) works quite well in reducing round-off error in symplectic integrators.

2. METHOD

Symplectic integrators can be regarded as instances of fixed-step integrators to solve the initial-value problem of

[1] This is a typical choice to make the truncation error comparable to the machine epsilon.

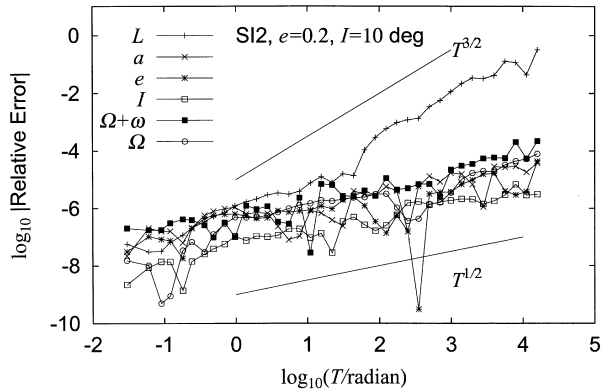

FIG. 1.—Growth of round-off error in elements. Shown are the round-off errors of an integrated Keplerian orbit. The integrator used was the second-order symplectic integrator. The round-off errors were measured by comparison of the integrations in the single- and double-precision environments. The measured environment was a personal computer with an Intel Pentium processor under Windows 95. The adopted compiler was Microsoft Fortran PowerStation version 4.

an autonomous special second-order ordinary differential equation,

$$\frac{d^2x}{dt^2} = f(x), \quad x(t_0) = x_0\ , \quad v(t_0) \equiv \left(\frac{dx}{dt}\right)(t_0) = v_0\ . \tag{1}$$

Let us introduce two basic operators,

$$\begin{aligned} S_X(h)&: \quad x_{n+1} = x_n + hv_n\ , \quad v_{n+1} = v_n\ , \\ S_V(h)&: \quad x_{n+1} = x_n\ , \quad v_{n+1} = v_n + hf(x_n)\ . \end{aligned} \tag{2}$$

Then the single-step formulae for the second and higher (even) order symplectic integrators are expressed as symmetric products of these two basic operators (Yoshida 1990; see also Appendix A). In the following, we will examine the possible sources of round-off error in the above basic operations. Similar discussions are given by Quinlan & Tremaine (1990) for the linear multistep methods and by Fukushima (1996b) for the midpoint rule as the best test integrator in the extrapolation methods.

In floating-point arithmetic, underflow is one of the major sources of round-off (Wilkinson 1960). Underflow appears in the sum of two quantities that have a large difference in magnitude. Assume the use of four-digit mantissa operations. Then a summation such as $1.000 + 2.102 \times 10^{-3} \approx 1.002$ will lose the information from the lower digits, 0.102×10^{-3}. Apart from the internal procedure to evaluate f, which we will discuss separately, the only candidate for underflow in the formulae (eq. [2]) is the addition of the form $y \leftarrow y + \Delta y$, where $y = x$ or $y = v$. If $|\Delta y|$ is considerably smaller than $|y|$, and this is almost always true when h is sufficiently small, an underflow will occur in this process. Therefore, a reduction in the number of additions leads to the reduction of round-off error. A few primitive but effective techniques for symplectic integrators are given in Appendix A, although the expected gain is around a 30% decrease at most.

Another source of information loss is the multiplications. To express the full information in the multiplication of two quantities with m-digit mantissas, we need a mantissa of around $2m$ digits, as $1.343 \times 2.103 = 2.824329$. Thus, if we store the product in a variable with the same m-digit mantissa, as $1.343 \times 2.103 \approx 2.824$, we will lose the information from around the lower m digits, 0.000329. Note that this information loss bears no relation to the magnitudes, or the

TABLE 1

MEASURED CPU TIMES OF BASIC MATHEMATICAL OPERATIONS

	DOUBLE PRECISION					QUADRUPLE PRECISION	
OPERATION	i486	Pentium	PA-7000	μSPARC-II	Alpha	PA-7000	μSPARC-II
$x = y$	0.46	0.24	1.00	1.00	0.41	5.1	2.8
$x = y + z$	0.91	1.00	1.00	1.00	1.00	131	11.3
$x = yz$	1.09	1.00	1.00	1.00	1.00	457	99.5
$x = y/z$	2.81	4.49	4.92	3.32	6.59	1100	168
$x = y^{1/2}$	12.0	19.6	4.95	24.4	38.5	1100	149
$x = \sin y$	19.8	23.3	49.8	22.0	18.5	9510	1740
$x = \log y$	20.2	23.9	47.8	31.9	22.0	5420	1240
$x = \tan^{-1} y$	19.8	27.3	46.3	37.3	23.5	5930	1270
$x = \tan y$	18.0	28.3	63.2	30.0	35.1	8450	2040
$x = \exp y$	24.9	32.7	42.0	19.0	23.2	6560	1370

NOTES.—Shown are the relative CPU times for basic arithmetic operations and some mathematical functions for typical processors. The units are DFLOPs, defined as a simple mean of additions and multiplications in double-precision arithmetic.

TABLE 2

ESTIMATED CPU TIMES OF N-BODY ROUTINE

Precision	i486	Pentium	PA-7000	μSPARC-II	Alpha
Double	27	36	22	40	57
Double-length double.....	360	282	268	284	311
Quadruple	...	...	6054	1070	...

NOTES.—Shown are the relative CPU times for the core part of a routine to evaluate N-body interactions, $\sum_{J \neq K} = (\mu_J / r_{JK}^3) \mathbf{r}_{JK}$. The units are DFLOPs per perturber. The numbers were estimated from the data in Tables 1 and 4 and the assumption that the ratios of basic operations used in the routine are five additions/subtractions, three squarings, four multiplications, one division, and one square root.

exponents if rigorously speaking, of the multiplicands. The exception occurs in the case the lower information is equal to zero. This happens when the sum of the effective digits (or bits) is not greater than the size of the mantissa, as $1.343 \times 2.000 = 2.686$ or $1.300 \times 2.100 = 2.730$, for example. In particular, if one of the multiplicands is an integral power of 2, then no information loss occurs in binary arithmetic. In the above procedures (eq. [2]), the source of this type of information loss is the multiplications by h. Thus, if one can choose h as a power of 2, as $h = 2^{-k}$, this type of error is greatly reduced. This is feasible in the second-order symplectic integrator, S_2. However, the step sizes in the higher order methods, say, S_4 and S_6, are not commensurable with each other (see Appendix A). Therefore, the information loss in multiplications is unavoidable in general.

Now consider ways to reduce these round-off errors. As stated earlier, the orthodox approach is to increase the mantissa of the variables under consideration. To do this economically, we borrow the concept of a *double length* computation from Dekker (1971). The idea is to assign two double-precision floating-point words, which we call a double-length variable, to each variable and to realize their arithmetic and some basic mathematical operations such as `sqrt` in a quasi–quadruple-precision computation. The actual implementation consists of exact addition and multiplication, Newton's method, and other numerical techniques (Dekker 1971; Appendix B). Given such a double-length variable x, we label its components with indexes 1 and 2. Namely, we denote the single-length variable, which is a double-precision word, for the larger part by x_1, and the other for the smaller part by x_2. The sum of these two parts must represent the original double-length variable exactly, $x = x_1 + x_2$. We assume that the smaller part is not greater than the larger one by a factor of the single-length machine epsilon ϵ, as $|x_2| \leq \epsilon |x_1|$. When this condition is not satisfied, it is enforced *without loss of precision* by the FORTRAN-77 routine `norm2` given in Appendix B. The basic arithmetic operations for double-length variables are realized by the FORTRAN-77 routines in Appendix B, where the double-precision 53-bit mantissa is taken as the single length: addition by `add2`, subtraction by `sub2`, multiplication by `mul2`, division by `div2`, and the square root by `sqrt2`. By using these routines, we may replace the corresponding operations in the symplectic integrators, namely, those given in Appendix A as equations (A1), (A5), (A7), (A11), and (A13). The resulting routines actually realize around 105-bit integrations, which we have confirmed by numerical experiments. Table 2 lists estimated CPU times for double-length routines to evaluate an N-body interaction for various processors. The ratio of the increase in CPU time from the double-precision computation to the double-length double-precision one ranges from 5.5 for a DEC Alpha chip to 13.3 for an Intel 486. On the other hand, the ratio when moving from the double-length double to the quadruple environment spans the range 3.8–22.6. This is a significant speedup when considering the fact that a quasi–quadruple-precision ($\sim$105-bit mantissa) computation is achieved.

3. NUMERICAL EXPERIMENTS

In order to measure the efficiency of the introduction of double-length arithmetic, we conducted a series of numerical integrations of a Keplerian motion with $e = 0.2$ and $I = 10°$, representing Mercury's orbit. To emphasize the effects of round-off error clearly, we performed the test integrations utilizing the double-length routines in single-precision computations containing 24-bit mantissas and compared them with the results of the ordinary integrations in the double-precision computations. The routines used for single-precision computations are almost the same as those given in Appendix B. The only difference except the change in type declarations from `REAL*8` to `REAL*4` is the replacement of a numerical constant in `mul2` and others to split the variables into quasi–half-length ones, namely, $4097.0 \equiv 2^{12} + 1$ instead of $134{,}217{,}729.\text{d}0 \equiv 2^{27} + 1$.

We compared four options: (1) no reduction of round-off error; (2) the use of `sum2`, a variation of `add2`, in the summation procedures $x \leftarrow x + (h/2)v$ and $v \leftarrow v + hf$; (3) the use of `mul2` and `sum2` in the above summation procedures; and (4) full use of double-length operations, both in the summation procedures and in the evaluation of two-body accelerations, $f(x)$. Table 3 shows the relative CPU times of these four options actually measured using a personal computer with an Intel Pentium processor. The measured value for the last option, 5.6–6.6, roughly confirms the previous estimate, 7.8, which can be deduced from Table 2. The difference is due to the effect of other minor operations such as copying values and the overhead of calling subroutines.

Figure 2 shows the growth of accumulated round-off error in the mean longitude, L, for these four options using the SI2 integrator. Here the step size was fixed at $1/(66\pi)$ of the orbital period. It is clear that the second and third options gave almost identical results, which are around a few digits better than the case of no reduction. Since the third option costs more than the second one, we conclude that the second is better than the third. Of course, the last option resembles the double-precision computations and, therefore, achieved a precision gain of around eight digits.

TABLE 3

MEASURED CPU TIMES OF OPTIONS TO REDUCE ROUND-OFF ERROR

Option	Case A	Case B
No reduction	1.00	1.00
`sum2`	1.22	1.00
`sum2` and `mul2`	3.48	1.00
Full	5.64	6.56

NOTES.—Shown are the relative CPU times of the options to reduce round-off error in orbital integrations using symplectic integrators. The measured environment was a personal computer with an Intel Pentium processor under Windows 95. The adopted compiler was Microsoft Fortran PowerStation version 4. The options compared are (1) no extra care for the reduction, (2) use of the double-length summation routine `sum2` in summations such as $x_{n+1} = x_n + \Delta x_n$, (3) option 2 plus the use of the double-length multiplication routine, `mul2`, in operations such $\Delta x_n = h v_{n+1/2}$, and (4) the full use of double-length routines. The cases compared are the pure two-body problem, for which the relative weight of options 2 and 3 is a maximum (case A), and sufficiently large N-body problems, in which most of the CPU time is consumed by evaluation of mutual gravitational interactions containing `sqrt` (case B).

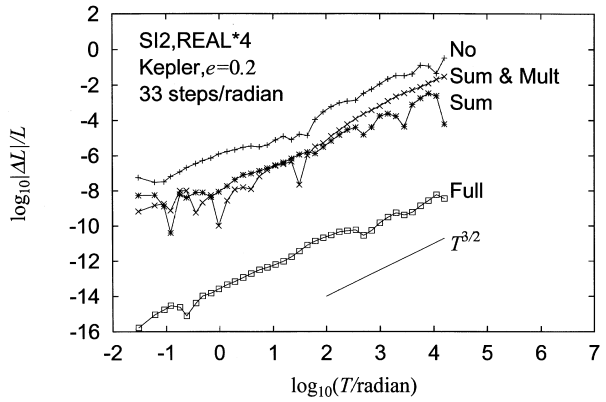

FIG. 2.—Reduction of round-off error in the mean longitude, L. Compared are the four options listed in Table 3.

However, this is at the cost of a great increase in CPU time, as listed in Table 2.

We obtained almost the same graphs for the SI4 and SI6 integrators. It is noteworthy that the magnitudes of the errors are rather larger in the case of the higher order formulae. This is because the number of operations is greater than in SI2, even after utilizing the technique to reduce the number of operations described in Appendix A. On the other hand, Figure 3 shows a similar error growth in the semimajor axis, a. As is easily conjectured from Figure 1, the effects of round-off error reductions for other Keplerian elements are almost the same as those for a.

4. CONCLUSION

The possible sources of round-off error in symplectic integrators have been examined. As a remedy, not so effective was the reduction of the number of operations described in Appendix A.[2] On the other hand, the introduction of the double-length addition routine of Dekker (1971) in the summations turns out to reduce accumulated round-off error by a few digits. The additional introduction of Dekker's double-length multiplication routine yields practically no additional gain in precision, while costing significantly more. Full use of Dekker's double-length routines, not only in the integration formula but in the force evaluation as well, realizes a quasi–quadruple-precision integration and achieves around 16 digits' gain in precision. Unfortunately, this is comes at the cost of a 5.5–13.3 times increase in CPU time. The ratio depends on the processor used and the problems applied. Note, however, that the increased CPU time is still significantly—say, 3.8–22.6 times—smaller than that required in a fully quadruple-precision computation.

Let us examine whether the application of the second technique in the foregoing enables us to conduct a physically meaningful integration for the long-term problems discussed in § 1. A numerical integration requiring of order 5×10^{10} steps leads to a precision loss of around 16 digits due to the accumulation of round-off errors. Thus, the increase of a few effective digits achieved here makes such an integration meaningful in ordinary double-precision (around 16 digits' accuracy) environments. In order to conduct physically meaningful integrations requiring larger numbers of steps, such as 2×10^{12}, however, the gain of a few more digits by additional care will be necessary.

One way to obtain such additional bits is the use of *extended*-precision arithmetic, the full 65-bit mantissa computation realized within the IEEE scheme (Vitagliano 1997). In fact, such a C compiler[3] existed once. Currently, at least a Fortran compiler[4] and a BASIC interpreter[5] are available for this purpose. We have confirmed that the upgrade from a 53- to a 65-bit environment requires no practical increase in CPU time.

Another possibility is the use of mixed-variable symplectic integrators (Kinoshita, Yoshida, & Nakai 1991; Wisdom & Holman 1991). Also effective may be the introduction of Encke's method and its generalization (Fukushima 1996a), although the symplectic methods are not applicable to this type of formulation. Of course, the integration of Lagrangian/Gaussian planetary equations is worth considering (Fukushima 1996a). Recently we found that the Kustaanheimo-Stiefel transformation also contributes to the reduction of error accumulation (Arakida & Fukushima 2000). Unfortunately, the use of Dekker's double-length addition routine by itself is not directly applicable to these formulations, except the introduction of extended-precision computation. The reason is that such application still requires the use of trigonometric, exponential, and/or logarithmic functions in double-length environments. The issue will be resolved when these routines are developed.

For the reader's convenience, we present some basic FORTRAN-77 routines for double-length double-precision mathematical operations in Appendix B. The full package of these routines, as well as the enhanced versions of symplectic integrators of orders of 2, 4, and 6, are available from the author upon request.

The author thanks the referee for suggestions to improve the quality of the presentation.

[2] Yet we recommend its use at any time, since it is implemented with no extra computational time.

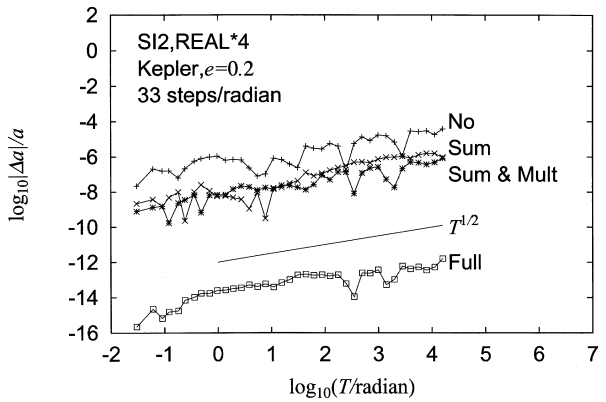

FIG. 3.—Same as Fig. 2, but for the semimajor axis, a

[3] Some old versions of the Microsoft C compiler, including MS C 7.0.

[4] Salford Fortran 95 can deal with `REAL*10` variables with almost no extra CPU overhead; see http://www.salfordsoftware.co.uk/compilers/ftn95/index.shtml.

[5] PowerBASIC, detailed information about which is available at http://www.powerbasic.com/.

APPENDIX A

EXPRESSIONS FOR SYMPLECTIC INTEGRATORS PRODUCING LESS ROUND-OFF ERROR

As will be shown below, the round-off error of the symplectic integrators can be reduced roughly 30% by rewriting the internal operations appropriately. Although this is a significant achievement and we encourage strongly its full usage, the practical effect is far less than that obtained by the use of double-length arithmetic, as explained in the main text.

A1. SI2

The "position first" symplectic integrator of the second order (SI2) is defined in terms of the first-order symplectic operators (eq. [2]) as

$$S_2(h) \equiv S_X(h/2) \circ S_V(h) \circ S_X(h/2) , \tag{A1}$$

where h is the (fixed) step size. The actual procedure is explicitly written as the mapping

$$S_2(h): \qquad (x_n, v_n) \to (x_{n+1}, v_{n+1}) , \tag{A2}$$

where

$$x_{n+1/2} = x_n + \tfrac{1}{2} h v_n , \qquad v_{n+1} = v_n + h f(x_{n+1/2}) , \qquad x_{n+1} = x_{n+1/2} + \tfrac{1}{2} h v_{n+1} . \tag{A3}$$

When this formula is repeatedly used with a fixed step size, as is usually done, the number of additions, the major sources of round-off, required to transit from x_0 to x_n is $2n$. (Here we have ignored those in the velocity computation since the round-off produced there is decreased by the factor h and therefore negligible.) Now, this number can be roughly halved by rewriting the above formulae as

$$x_{n+1/2} = x_{n-1/2} + h v_n , \qquad v_{n+1} = v_n + h f(x_{n+1/2}) , \qquad x_{n+1} = x_{n+1/2} + \tfrac{1}{2} h v_{n+1} . \tag{A4}$$

This time the number of additions needed in order to reach x_n is only $n + 1$. Thus, the accumulated round-off error is expected to be reduced by $1 - (1/2)^{1/2} \approx 29\%$ in the long run. Note that this rewriting requires no increase in computational time.

A2. SI4

The symplectic integrators of fourth and higher (even) orders are constructed from that of the second order (Yoshida 1990). In fact, Ruth's fourth-order formula (SI4) is expressed as

$$S_4(h) \equiv S_2(a_1 h) \circ S_2(a_2 h) \circ S_2(a_1 h) , \tag{A5}$$

where

$$a_1 \equiv \frac{1}{2 - \sqrt[3]{2}} \approx 1.3512071919596576 , \qquad a_2 \equiv 1 - 2a_1 \approx -1.7024143839193153 . \tag{A6}$$

In order to reduce the round-off error locally, it is better to rewrite these formulae in terms of basic operators directly, as

$$S_4(h) = S_X(c_1 h) \circ S_V(a_1 h) \circ S_X(c_2 h) \circ S_V(a_2 h) \circ S_X(c_2 h) \circ S_V(a_1 h) \circ S_X(c_1 h) , \tag{A7}$$

where

$$c_1 \equiv \tfrac{1}{2} a_1 \approx 0.67560359597982882 , \qquad c_2 \equiv \tfrac{1}{2}(a_1 + a_2) = \tfrac{1}{2} - c_1 \approx -0.17560359597982882 . \tag{A8}$$

The number of additions with respect to positions x, which are the main source of round-off error, is reduced from six to four when compared with the naive combination of equations (A1) and (A5). Therefore, the effect of round-off error is expected to reduce locally by the factor $1 - (2/3)^{1/2} \sim 18\%$.

When this integrator is repeatedly used, we can reduce the number of additions with respect to x by one. This is done by combining the first S_X operation and the last S_X in the previous step as we did in the case of SI2. To illustrate this explicitly, we write the above rewritten expression for SI4 (eq. [A7]) explicitly as

$$\begin{aligned} x_{n+c_1} &= x_n + c_1 h v_n , \qquad v_{n+a_1} = v_n + a_1 h f_{n+c_1} , \qquad x_{n+1/2} = x_{n+c_1} + c_2 h v_{n+a_1} , \\ v_{n+1-a_1} &= v_n + a_2 h f_{n+1/2} , \qquad x_{n+1-c_1} = x_{n+1/2} + c_2 h v_{n+1-a_1} , \\ v_{n+1} &= v_{n+1-a_1} + a_1 h f_{n+1-c_1} , \qquad x_{n+1} = x_{n+1-c_1} + c_1 h v_{n+1} . \end{aligned} \tag{A9}$$

where we have abbreviated $f(x_v)$ as f_v. In further rewriting, the first operation is replaced as

$$x_{n+c_1} = x_n + c_1 h v_n \Rightarrow x_{n+c_1} = x_{n-1-c_1} + 2c_1 h v_n . \tag{A10}$$

Therefore the number of additions finally reduces to three, exactly half of the naive combination. That is, we can expect the accumulated round-off error will be reduced by the same factor as in SI2, $\sim 29\%$. Again there is no increase of CPU time by virtue of the introduction of this rewriting.

A3. SI6

The sixth-order formula is written in terms of S_2 as

$$S_6(h) \equiv S_2(b_1 h) \circ S_2(b_2 h) \circ S_2(b_3 h) \circ S_2(b_4 h) \circ S_2(b_3 h) \circ S_2(b_2 h) \circ S_2(b_1 h) , \tag{A11}$$

where one of Yoshida's three sets of coefficients (SI6A) is

$$b_1 \approx 0.784513610477560 , \quad b_2 \approx 0.235573213359357 , \quad b_3 \approx -1.177679984178870 ,$$
$$b_4 \equiv 1 - 2(b_1 + b_2 + b_3) \approx 1.315186320683906 . \tag{A12}$$

Similarly, the sixth-order formula (eq. [A11]) is rewritten in terms of basic operations as

$$S_6(h) = S_X(d_1 h) \circ {}_V(b_1 h) \circ S_X(d_2 h) \circ S_V(b_2 h) \circ S_X(d_3 h) \circ S_V(b_3 h) \circ S_X(d_4 h) \circ S_V(b_4 h)$$
$$\circ S_X(d_4 h) \circ S_V(b_3 h) \circ S_X(d_3 h) \circ S_V(b_2 h) \circ S_X(d_2 h) \circ S_V(b_1 h) \circ S_X(d_1 h) , \tag{A13}$$

where

$$d_1 \equiv \tfrac{1}{2} b_1 \approx 0.3922568052387800 , \quad d_2 \equiv \tfrac{1}{2}(b_1 + b_2) \approx 0.5100434119184585 ,$$
$$d_3 \equiv \tfrac{1}{2}(b_2 + b_3) \approx -0.4710533854097565 , \quad d_4 \equiv \tfrac{1}{2}(b_3 + b_4) \approx 0.0687531682525180 . \tag{A14}$$

This time the number of additions with respect to x decreases from 14 to eight when compared with the combination of equations (A1) and (A11). Therefore, the effect of local round-off errors is expected to be reduced by the factor $1 - (8/14)^{1/2} \sim 24\%$ in the long run. Again, in the repeated use of this integrator, the factor can be made the same as in SI2 by combining the first S_X operation with the last S_X in the previous step.

APPENDIX B

DOUBLE-LENGTH ROUTINES

Below are FORTRAN-77 adaptations of some basic routines to realize double-length arithmetic in the standard double-precision environment (Dekker 1971). In the following, a magic number, $134{,}217{,}729 \equiv 2^{26} + 1$, appears. This is used to pick up the upper 26 bits of double-precision variables, whose mantissas are 53 bits in length.

B1. NORMALIZATION

```
SUBROUTINE norm2(x1,x2,z1,z2)
REAL*8 x1,x2,z1,z2
z1=x1+x2
z2=(x1-z1)+x2
return
END
```

B2. ADDITION

```
SUBROUTINE add2(x1,x2,y1,y2,z1,z2)
REAL*8 x1,x2,y1,y2,z1,z2,v1,v2,w1,w2
w1=x1+y1
if (abs(x1).gt.abs(y1)) then
   w2=(((x1-w1)+y1)+y2)+x2
else
   w2=(((y1-w1)+x1)+x2)+y2
endif
z1=w1+w2
z2=(w1-z1)+w2
return
END
```

B3. SUBTRACTION

```
SUBROUTINE sub2(x1,x2,y1,y2,z1,z2)
REAL*8 x1,x2,y1,y2,z1,z2
call add2 (x1,x2,-y1,-y2,z1,z2)
return
END
```

B4. MULTIPLICATION

```
SUBROUTINE mul2(x1,x2,y1,y2,z1,z2)
REAL*8 x1,x2,y1,y2,z1,z2,hx,tx,hy,ty,p,q,w1,w2
p=x1*134217729.d0
hx=(x1-p)+p
tx=x1-hx
p=y1*134217729.d0
hy=(y1-p)+p
ty=y1-hy
p=hx*hy
q=hx*ty+tx*hy
w1=p+q
w2=(((p-w1)+q)+tx*ty)+(x1*y2+x2*y1)
z1=w1+w2
z2=(w1-z1)+w2
return
END
```

B5. DIVISION

```
SUBROUTINE div2(x1,x2,y1,y2,z1,z2)
REAL*8 x1,x2,y1,y2,z1,z2,v1,v2,w1,w2,hw,tw,hy,ty,p,q
w1=x1/y1
p=w1*134217729.d0
hw=(w1-p)+p
tw=w1-hw
p=y1*134217729.d0
hy=(y1-p)+p
ty=y1-hy
p=hw*hy
q=hw*ty+tw*hy
v1=p+q
v2=((p-v1)+q)+tw*ty
w2=((((x1-v1)-v2)+x2)-w1*y2)/y1
z1=w1+w2
z2=(w1-z1)+w2
return
END
```

B6. SQUARE ROOT

```
SUBROUTINE sqrt2(x1,x2,z1,z2)
REAL*8 x1,x2,z1,z2,v1,v2,w1,w2,hw,tw,p,q
if (x1.le.0.d0) then
   w1=0.d0
   w2=0.d0
   return
endif
w1=sqrt(x1)
p=w1*134217729.d0
hw=(w1-p)+p
tw=w1-hw
p=hw*hw
q=2.d0*hw*tw
v1=p+q
v2=((p-v1)+q)+tw*tw
w2=(((x1-v1)-v2)+x2)/(2.d0*w1)
z1=w1+w2
z2=(w1-z1)+w2
return
END
```

TABLE 4

NUMBER OF BASIC MATHEMATICAL OPERATIONS FOR DOUBLE-LENGTH ROUTINES

ROUTINE	NUMBER OF OPERATIONS						CPU TIME (DFLOPs)				
	±	*	if	abs	/	sqrt	i486	Pentium	PA-7000	μSPARC-II	Alpha
norm2	3	0	0	0	0	0	2.7	3.0	3.0	3.0	3.0
add12	6	0	1	2	0	0	8.5	9.0	9.0	9.0	9.0
add21, sum21	7	0	1	2	0	0	9.4	10.0	10.0	10.0	10.0
add2, sum2	8	0	1	2	0	0	10.3	11.0	11.0	11.0	11.0
sq12	10	5	0	0	0	0	14.6	15.0	15.0	15.0	15.0
sq2	12	7	0	0	0	0	18.6	19.0	19.0	19.0	19.0
mul12	14	6	0	0	0	0	19.3	20.0	20.0	20.0	20.0
mul2	16	8	0	0	0	0	23.3	24.0	24.0	24.0	24.0
inv2	17	7	0	0	2	0	28.7	28.5	28.9	27.3	37.2
div2	18	7	0	0	2	0	29.6	29.5	29.9	28.3	38.2
sqrt2	13	6	1	0	1	1	34.2	44.1	29.9	47.7	65.1

NOTES.—The estimated CPU times are computed from the data in Table 1 and the assumption that the average CPU times for the `if` statement and the `abs` function are 1.0 DFLOP each.

B7. OTHERS

We have developed some other routines that in theory can be derived by calling these basic routines but which run a little faster (see Table 4). They are

`add12(x1,y1,z)`: Evaluate the sum of two single-length variables, `x1` and `y1`, and return the result in a double-length one, `z`;

`add21(x,y1,z)`: Evaluate the sum of a single-length variable, `y1`, and a double-length one, `x`, and return the result in another double-length one, `z`;

`sum21(x,y1)`: Add a single-length variable, `y1`, to a double-length one, `x`;

`sum2(x,y)`: Add a double-length variable, `y`, to another double-length one, `x`;

`mul12(x1,y1,z)`: Evaluate the product of two single-length variables, `x1` and `y1`, and return the result in a double-length one, `z`;

`sq12(x1,z)`: Square a single-length variable, `x1`, and return the result in a double-length one, `z`;

`sq2(x,z)`: Square a double-length variable, `x`, and return the result in another double-length one, `z`;

`inv2(x,z)`: Evaluate the inverse of a double-length variable, `x`, and return the result in another double-length one, `z`.

B8. CPU TIME

The CPU times of the above routines can be roughly estimated by counting the basic operations used. The results are shown in Table 4. Comparison with Table 1 reveals that the CPU times of the double-length routines are of the same order as those for the typical mathematical libraries.

REFERENCES

Arakida, H., & Fukushima, T. 2000, AJ, 120, 3333
Brouwer, D. 1937, AJ, 46, 149
Dekker, T. J. 1971, Numer. Math., 18, 224
Earn, D. J. D., & Tremaine, S. 1992, Physica D, 56, 1
Evans, N. W., & Tremaine, S. 1999, AJ, 118, 1888
Fukushima, T. 1996a, AJ, 112, 1263
———. 1996b, AJ, 112, 1298
———. 1999, in IAU Colloq. 173, Evolution and Source Regions of Asteroids and Comets, ed. J. Svoren, E. M. Pittich, & H. Rickman (Tatranská Lomnica: Astron. Inst. Slovak Acad. Sci.), 309
Hairer, E., Nørsett, S. P., & Wanner, G. 1993, Solving Ordinary Differential Equations I (2d ed.; Berlin: Springer)
Jackson, J. 1924, MNRAS, 84, 602
Kinoshita, H., Yoshida, H., & Nakai, H. 1991, Celest. Mech. Dyn. Astron., 50, 59
Lambert, J. D., & Watson, I. A. 1976, J. Inst. Math. Applications, 18, 189
Milani, A., & Nobili, A. M. 1988, Celest. Mech., 43, 1
Petit, J.-M. 1998, Celest. Mech. Dyn. Astron., 70, 1
Quinlan, G. D. 1994, Celest. Mech. Dyn. Astron., 58, 339
Quinlan, G. D., & Tremaine, S. 1990, AJ, 100, 1694
Quinn, T., & Tremaine, S. 1990, AJ, 99, 1016
Sanz-Serna, J. M. 1992, Acta Numer., 1, 243
Vitagliano, A. 1997, Celest. Mech. Dyn. Astron., 66, 293
Wilkinson, J. H. 1960, Numer. Math., 2, 319
Wisdom, J., & Holman, M. 1991, AJ, 102, 1528
Yoshida, H. 1990, Phys. Lett. A, 150, 262