Requirements of an N-body integrator for astronomy

P.W. Sharp
Department of Mathematics, University of Auckland
Private Bag 92019, Auckland, NEW ZEALAND.
sharp@scitec.auckland.ac.nz

May 4, 2000

Abstract

N-body simulations in astronomy form a challenging set of initial value problems for numerical integrators. The challenge comes from the variety of problems and their size - one recent simulation required 300 million second order equations, another 9×10^{10} integration steps.

A number of integrators for specific types of simulations are available. We investigate what is required of an integrator intended to efficiently perform a wide range of N-body simulations.

Keywords: N-body integrator, efficiency, requirements

2000 MSC: Primary - 70F10, 85-04, secondary - 65L05

1 Introduction

Simulations of N bodies interacting are used extensively to model the dynamics of astronomical systems. They are used to generate ephemerides, to study the evolution of the solar system, the asteroid belt, Kuiper's belt and the Oort's cloud, and to study the dynamics of star groups, such as the evolution of globular clusters, the formation of spiral arms in galaxies and the interaction between galaxies.

Most N-body simulations require the numerical solution of an initial value problem (IVP) of second order ordinary differential equations. When a very accurate model is needed, the IVP is often of the form

$$\ddot{y}(t) = f(t, y(t), \dot{y}(t)), \quad y(t_0) = y_0, \quad \dot{y}(t_0) = \dot{y}_0, \tag{1}$$

where $\dot{}$ denotes differentiation with respect to t, $f : \mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}^n$, and $n = 2N$ and $3N$ for simulations in two and three space dimensions respectively. The function f can contain terms modelling Newtonian and post-Newtonian gravitational forces, and non-gravitational forces such as radiation pressure, cometary jet forces and Poynting-Robertson drag.

In simulations with just Newtonian gravitational forces, the IVP is of the form

$$\ddot{y}(t) = f(y(t)), \quad y(t_0) = y_0, \quad \dot{y}(t_0) = \dot{y}_0, \tag{2}$$

where $f : \mathbb{R}^n \mapsto \mathbb{R}^n$.

We refer to (1) as GP (General Problem) and to (2) as SP (Special Problem). We refer to the IVP obtained by transforming either GP or SP to a first order system as FP (First order Problem).

The number of particles for SP can be very large. In a recent simulation, Dubinski [7] used 10^8 particles to study the interaction between the Andromeda Galaxy and the Milky Way. The number of integrations step can be large. For example, Grazier, Newman, Kaula and Hyman [10] used 9×10^{10} steps in their simulation of the outer solar system.

When evaluating f for SP, the acceleration of each particle can be calculated by summing the contribution to the acceleration from every other particle. This requires $O(N^2)$ operations for each evaluation of f and is known as direct summation. The cost of the evaluation can be reduced if the exact value of f is not required. The field approach assumes the force on each particle is due to a gravitational field; this approach requires $O(N)$ operations for each evaluation. The multipole approach, which is intermediate to direct sum and the field approach in cost and accuracy, requires $O(N \log N)$ operations for each evaluation. The gravitational force on each particle is approximated by a multipole expansion and distant particles are lumped together when calculating the force on a particle.

Particles in a simulation can have different interaction properties. One common choice is to have massive particles that interact with all other particles and massless particles that interact with the massive particles only. The cost of evaluating f is reduced because not all possible interactions are calculated.

In some simulations, particles are eliminated as the integration proceeds. Small particles may be eliminated because they cross the orbit of a large particle or have a close encounter with it. Any particle, big or small, may be eliminated because it is too far from the centre of mass to have a significant effect on the solution, or because it collides and merges with another particle. It is also possible for the number of particles to increase if fragmentation upon collision is permitted.

A number of integrators for N-body simulations are available. These include `treecode` [1], `NBI` [23], `SWIFT` [6], `DE118i` and `mercury` [3]. The first three integrators solve SP, the last two GP.

`Treecode` is a fixed-stepsize integrator that uses the multipole approach up to the quadpole. The integration is performed using a leapfrog method. `NBI` has four integration methods, all fixed-stepsize: a fixed-order Cowell-Störmer method, a Gragg-Bulirsch-Stoer method, a symplectic method and a method that uses Wisdom-Holman mapping. The first three integration methods use direct summation, the fourth a hierarchical scheme. `SWIFT` uses direct summation and has three fixed-stepsize symplectic methods and one variable-step Bulirsch-Stoer method.

DE118i is a fixed-stepsize integrator for generating ephemerides of the planets and the Moon. It uses the equation of motion described in [16] for the acceleration. Each integration is performed using a fixed-order Adams method. Mercury performs simulations in which there is a large central body. Direct summation is used and there are five integration methods: two fixed-stepsize symplectic methods, two variable-stepsize Bulirsch-Stoer methods, one for conservative systems and one for general systems, and a variable-stepsize order 15 Gauss-Radau method.

Each integrator described above is intended to efficiently perform specific types of simulations. We investigate what is required of an integrator intended to efficiently perform a wide range of N-body simulations. We assume the IVP for the simulation is either GP or SP.

We begin in §2 with the implications of n (the number of equations) changing during the integration. In §3 we discuss what integration methods could be included in the integrator. Simulations can require the user to specify a lot of input, and generate copious output. This is discussed in §4. In §5, 6 and 7 we discuss the data structures, the influence of round-off error, and the advantages and disadvantages of writing the integrator in C, Fortran 77 or Fortran 90. We end in §8 with a summary of the possible routines for the integrator.

Throughout the discussions, we measure efficiency by the amount of CPU time required to find a solution with a prescribed accuracy. We restrict the term linear multistep method to multistep methods that use just y, $\dot{y}$ and f; those that use derivatives of f are called multistep multiderivative methods.

2 Number of equations

As noted in §1, n may change during the integration. For a fixed-stepsize integration method, fewer particles means fewer arithmetic operations; conversely, more particles mean more operations. The change in the number of operations for variable-stepsize methods is more difficult to predict because changing N changes f and hence the sequence of stepsizes.

A reduction in n may decrease the reductions in CPU time obtained from compiler optimisation, hand optimisation and special hardware. For example, the simulation in [10] began with 10^5 massless particles, ended with 36, and after just one percent of the integration steps had been taken, the number of particles had dropped to approximately 100. The benefits from loop optimisation or vector hardware could change significantly during such a simulation.

The above observations have several implications. The data structures (see §5) must be designed so that the overhead in deleting or adding a particle is minimised or nearly so. The information recorded in the output files about the changes to n must be sufficiently detailed that post-processing is not affected. More than one optimised version of inner loops could be provided.

3 Integration methods

The choice of which integration method to use depends on a number of factors. These include whether the stepsize is fixed or variable, whether the solution is found by solving FP or by solving SP or GP, stability and accuracy requirements, and considerations such as the cost of forming a continuous approximation of sufficient accuracy, the propagation of round-off error and storage requirements.

A survey published in 1989 by Kinoshita and Nakai [13] concluded that people performing N-body simulations rated Cowell-Störmer and associated linear multistep methods as their preferred integration method, followed closely by the classical order four explicit Runge-Kutta method. The choice is no longer as obvious because, as discussed below, in the last decade or so symplectic methods and their modifications have proven competitive, and new explicit Runge-Kutta Nyström (ERKN) methods are available that are considerably more efficient than older ERKN methods. In addition, extrapolation integrators have a greater prominence and numerical testing has shown multistep multiderivative methods can be more efficient than Cowell-Störmer methods.

3.1 Fixed or variable stepsize

Integration methods for N-body simulations in astronomy are usually fixed-stepsize. This contrasts with general initial value problems for which state-of-the-art integrators are invariably variable-stepsize.

The Newtonian attraction between particles is often modelled using an inverse square law. In this case, if the particles in a simulation do not make close encounters, the acceleration will not become large and a fixed-stepsize method may be as efficient as a variable-stepsize one, especially since stepsize changing incurs an overhead.

The Newtonian attraction is also modelled by a law of the form $(r^2 + \epsilon^2)^{-1}$, where ϵ is a small parameter called the softening parameter. It prevents the acceleration from growing unbounded during close encounters (at the cost of using an approximation to the acceleration). This potentially increases the efficiency of fixed-stepsize methods relative to variable-stepsize ones.

If close encounters or collisions are permitted and there is no softening parameter, variable-stepsize methods can offer marked gains in efficiency. In addition, variable-stepsize methods give greater confidence that important changes in the solution have not been missed.

3.2 Direct or indirect

ERKN methods (see for example [5]) use fewer f evaluations per step and have smaller error coefficients than explicit Runge-Kutta (ERK) methods of the same order. Hence, when one-step methods are used to solve SP, it is decidedly more efficient to solve SP directly than to solve FP. For GP, the tests in [8] and [22] show explicit Generalised

Runge-Kutta Nyström (ERKNG) methods of orders four, five and six are often more efficient than ERK methods of the same order.

For linear multistep methods, Krogh [private communication] presented a simple argument using the test equation

$$a\ddot{y} + b\dot{y} + cy = g(t),$$

where a, b and c are real. According to Krogh, the method of choice depends on the roots of $az^2 + bz + c = 0$; if these roots are close to equal, stability restrictions suggest it is more efficient to solve FP; if $b = 0$, the roots are as far apart as possible and solving SP directly will be more efficient.

The numerical comparisons on page 274 of [11] suggests extrapolation methods which solve SP directly require half the CPU time of extrapolation methods for FP.

3.3 Stability requirements

The stability properties of the integration methods used in N-body simulations do not receive a lot of attention, possibly because GP and SP are usually regarded as being non-stiff. However, for simulations which require a lot of CPU, any restriction of the stepsize by stability requirements could lead to a significant increase in CPU time. Hence we believe the size of the stability region must be considered, particularly for low accuracy requirements.

3.4 High accuracy

High accuracy is required to generate ephemerides, to check the answers produced by low order integration methods, and for some long integrations.

If f is expensive to evaluate, linear multistep methods have much to recommend them when solving SP, with a high order Cowell-Störmer method taking pride of place. If f is inexpensive to evaluate, a high order ERKN method may be competitive.

There are fewer choices for GP. ERKNG methods require more derivative evaluations per step than ERKN methods of the same order. Hence, any gains in efficiency ERKNG methods have over linear multistep methods for GP will in general be smaller than the gain ERKN methods have over linear multistep methods for SP.

High order multistep multiderivative methods can also be considered. At first glance these methods may appear impractical because the derivatives of f contain more terms than f. However, the crucial measure of the performance of any N-body integrator is the amount of CPU time required to achieve a solution with a prescribed accuracy. Hence a fairer test of whether to use a multiderivative method is the CPU time required to evaluate f and its derivatives.

Sharp [21] conducted some timing tests in Fortran 77 for a system of N planets orbiting in a plane under Newtonian attraction. He found that when $N \geq 5$ the time to evaluate the acceleration and its first derivative was less than twice that to evaluate the acceleration, if a high level of compiler optimisation was used. The optimising compiler was able to use values from the acceleration for the derivative (of the acceleration).

High order multistep multiderivative methods have other possible advantages. Huang and Innanen [12] investigated the stability of these methods and found that some had larger stability regions than linear multistep methods. They also cited earlier work of Huang and Valtonen who showed the methods had smaller local truncation errors. Huang and Innanen found for the circular restricted three body problem that high order multiderivative methods often used less CPU time than linear multistep methods.

3.5 Low accuracy

Symplectic methods come into their own for low accuracy requirements. Simple methods include leapfrog (an order two method) and the order four methods of Calvo and Sanz-Serna [2], Rowlands [18], and López-Marcos, Sanz-Serna and Skeel [14].

The above symplectic methods use the same stepsize for every particle. Saha and Tremaine [19] showed it was possible to use a different stepsize for each particle and retain symplecticness. This increases the efficiency when the orbital periods of the particles cover a wide range. Chambers [3] presented a second order symplectic method which, instead of varying the stepsize, varies the splitting of the Hamiltonian. This scheme permits efficient integration of close encounters.

If one is prepared to forego symplecticness, the efficiency can potentially be improved in a number of ways. For example, Quinn, Katz, Stadel and Lake [17] developed a non-symplectic leapfrog method which allows variable stepsize for each particle while retaining the low overhead of the standard leapfrog. Another possibility is to use pseudo-symplectic methods (see for example, Chambers and Murison [4]). Low order one-step multiderivative methods can be used, permitting different stepsizes for different particles.

3.6 Other considerations

If a continuous approximation to the solution is available and it is sufficiently accurate, it can be used to produce dense output and help decide if a particle has made a close approach to another particle or crossed its orbit. Linear multistep methods have an appropriate continuous approximation for free, as do low order one-step methods. For high order one-step methods, it is necessary to perform additional evaluations of f.

As discussed later in §6, round-off error can be significant in integrations of many steps. Because the error will in general depend on the integration method, round-off error characteristics could be used as a criterion when selecting a method.

When N is large, the storage requirements of an integration method may exceed what is available. If the integrator is to solve a wide range of simulations, an integration method with lower than normal storage requirements is needed.

3.7 Summary

The above discussion strongly suggest it is impossible for one integration method to be efficient for a wide range of simulations. We believe that at least three integrations methods are required. We would use a fixed-order, fixed-stepsize Cowell-Störmer method (for SP), a symplectic method that handled close encounters efficiently (also for SP), and a variable-order, variable-stepsize linear multistep method (for both GP and SP). The Cowell-Störmer method would have a maximum order of at least 13. A continuous approximation to the solution for the symplectic method would be calculated using cubic or quintic Hermite interpolation. The linear multistep method would be implemented so the user could specify the maximum order, thus giving control over the amount of storage used.

4 Run-time input and output

4.1 Run-time input

The main items of run-time input to an integrator are the initial state of the N-body system, a description of the model including how the accelerations are calculated, the specification of the integration method, and flags to control the extent and format of the output. It is also useful to permit some control of overhead, such as specifying how often checks are made for collisions. Providing this information can be a lot of work for the user. This burden can be reduced in at least two ways.

The integrator `treecode` has default values for all of the run-time input. These defaults can be changed on the command line when executing `treecode`. For example, the command

```
treecode in=in_run freq=64 freqout=1 tstop=100 nbody=1024 out=out_%05d
```

will read the initial state from the file `in_run` instead of generating the state internally, use a stepsize of 1/64 instead of the default of 1/32, print the solution once per unit of time (the default is never to print), use 1024 bodies instead of the default of 4096, and print the solution for $t = 1, 2, \ldots, 100$ to the files `out_00064`, `out_00128`, `out_00192`, ..., `out_03200` (the number at the end of the file names is the step number).

Another way to reduce the burden is to provide a file containing the default values, and have the integrator read this file at the start of the integration. The user changes the default values by editing the file.

4.2 Output

The output files produced by existing N-body integrators can be divided into four types: log, dump, solution and event. Log files are typically small. Dump files contain enough information and in the right format that the integration can be continued after the computer crashes or the program is interrupted. Solution files contain the position and velocity of the particles at times specified by the user. These files may also contain addition information such as the osculatory orbital elements, energy and momentum of the particles, and the energy of the system. Event files give information about special situations which occurred during the integration, such as collisions or close encounters.

The solution and event files can be large and may be too large for the user's computer. In addition, if a remote computer such as a supercomputer is being used, large files can take a long time to transmit to the local computer.

One technique to reduce the size of files is to use data compression, such as by representing floating point numbers as character strings (see for example [3]). This necessitates having routines which do the transformations between the two data types. Another technique permits the user to specify the number of significant figures for floating point numbers printed to output files.

Large files take a long time to read and write. It is convenient to use ASCII files, but binary files require less time.

The data in the solution files will often be used to produce graphs, coloured pictures or movies. Hence it is important the output be in the appropriate format. One possibility is to write routines which produce output correctly formatted for commonly used graphics packages.

5 Data structures

As well as recording the position and velocity of a particle, it is often useful to record other information such as the mass, energy and momentum. If eliminations can occur, the index of the particle at the start of the simulation should be stored.

Consider a simulation in which the position, velocity, type of particle, mass and energy is stored for each particle, a total of nine numbers for simulations in three space dimensions. To store the information for all of the particles, we could use a) nine one-dimensional arrays of length N, or b) one $N \times 9$ array, or c) one $N \times 3$ array for each vector quantity and a one-dimensional array of length N for each scalar quantity, or d) a one-dimensional array of length N in which an element is a record containing the position, velocity, type, mass and energy.

On some computers, accessing elements in a two-dimensional array can take significantly more CPU than for a one-dimensional array. In this case, a) and d) may be the most efficient data structures since they use just one subscript. However, a) may require indirect addressing which would increase the CPU time. Option c) uses smaller arrays than a), b) and d) and hence it is more likely the arrays will fit into main memory. Options c) and d) mimic the vector nature of the mathematical formulation more than a) or b), reducing the likelihood of an error when writing the integrator. Option d) is the most elegant and has the simplest syntax for the deletion or addition of a particle, but not all languages used in numerical computation, notably Fortran 77, support records.

Two or more of a) - d) could be implemented in the integrator. The user would select one at compile-time or run-time.

6 Round-off error

In integrations of many steps, the round-off error can become significant, making it obligatory to use techniques to limit the growth of round-off error.

A number of techniques have proven successful. For Cowell-Störmer methods, backward differences in summed form can be used and the differences added from highest to lowest order (see for example [9] and [10]).

Another technique assumes the biggest contribution to the round-off error is from the addition when the solution is updated at the end of a step. This error is estimated and subtracted on the next step. A very good introduction to this technique is given on pages 153 to 155 of [20].

In some simulations such as for the solar system, the masses and the typical distance between bodies can be used to sum the terms in the acceleration from approximately the smallest to largest in magnitude.

Some computers have extended precision of 80 bits, giving four to five decimal digit more significant figures than when using double precision (64 bits). The integrator `DE118i` permits the user to take advantage of this extended precision. Some computers have quadruple precision, giving about 34 decimal digits. Quadruple precision is significantly slower than double precision (for example, on SGI computers it is about 20 times slower), but it can be feasible to use quadruple precision for smaller simulations. Alternatively, it may be sufficient to use quadruple precision for some particles, such as the massive ones, saving CPU time compared with when all the calculations are done in quadruple precision.

For simulations of conservative systems in which direct summation is used and the round-off error dominates the truncation error, the size of the round-off error can be assessed by seeing how well energy is conserved. A similar assessment can be used with momentum.

7 Programming languages

Although an N-body integrator could be written in languages such as Pascal or Basic, we believe C, Fortran 77 and Fortran 90 are the serious contenders. Two questions immediately arise, should the language be C or Fortran, and if Fortran, Fortran 77 or Fortran 90?

The first question is much debated and we will not add to this debate. As for the second question, we believe Fortran 90 should be used except in special circumstances. The vector operations of Fortran 90 permit a more transparent translation from the mathematical formulation, reducing the likelihood of an error. Derived data types make manipulation of the information about each particle easier and dynamical arrays make the management of temporary workspace easier. For computers with quadruple precision, invoking it is achieved by changing one statement. Fortran 77 could be used when many of the routines for the integrator are already available in Fortran 77, or when the target computers do not have Fortran 90.

8 Routines

Table 1 contains a list of possible routines for an N-body integrator. We have divided the routines into eight groups: (integration) methods, input, output, acceleration, events, initial, transformation and the ubiquitous other. The purpose of most routines have been described in previous sections. For those that have not, we have included a short explanation in the table.

References

[1] J.E. Barnes, *Treecode guide*, June 1999, `http://www.ifa.hawaii.edu/faculty/barnes/treecode/treeguide.html`

[2] M.P. Calvo, J.M. Sanz-Serna, *The development of variable-step symplectic integrators, with applications to the two-body problem*, SIAM J. Sci. Stat. Comp. **14** (1993), 936-952.

[3] J.E. Chambers, *A hybrid symplectic integrator that permits close encounters between massive bodies*, Monthly Notices of the Royal Astronomical Society **304** (1999), 793 - 799.

[4] J.E. Chambers, M.A. Murison, *Pseudo-high-order symplectic integrators*, Astron. J. **199** (2000), 425-433.

[5] J.R. Dormand, M.E.A. El-Mikkawy, P.J. Prince, *High-order embedded RKN formulae*, IMA J. Num. Analysis **7** (1987), 423-430.

[6] M. Duncan, H. Levison, *SWIFT*, `http://k2.boulder.swri.edu/ ~hal/swift.html`

[7] J. Dubinski, University of Toronto news report, April 2000, `http://www.newsandevents.utoronto.ca/bin/000414b.asp`

[8] J.M. Fine, *Low order practical Runge-Kutta-Nyström methods*, Computing **38** (1987) 281-297.

[9] K.R. Grazier, *The stability of planetesimal niches in the outer solar system: a numerical study*, Ph.D. thesis, University of California, Los Angeles, 1997.

[10] K. Grazier, W.I. Newman, W.M. Kaula, J.M. Hyman, *Dynamical evolution of planetesimals in the outer solar system: I. the Jupiter/Saturn zone*, Icarus **140** (1999), 341-352.

[11] E. Hairier, S.P. Nørsett, G. Wanner, *Solving ordinary differential equations I: nonstiff problems*, Springer-Verlag, 1ed, 1987.

[12] Tian-Yi Huang, K. Innanen, *A survey of multiderivative multistep integrators*, Astron. J. **112**, 1 (1996), 1254 - 1262.

Group	Routines
Methods	A set of routines for each integration method, with each set containing a routine to evaluate a continuous approximation to the solution. Routines to initialise the integration.
Input	Opening input files, handing any errors on opening, checking the validity of the input, reading the input, handling a restart from a dump file, data uncompression.
Output	Opening output files, handling any errors on opening, writing to the files, data compression, printing error messages, formatting the output for commonly used graphics packages.
Acceleration	For GP: a set of routines that use direct summation to calculate the accelerations, with different routines for different types of forces. For SP: one set of routines for direct summation and one set for the multipole approach. A set could also be provided for standard gravitational fields. The sets using direct summation and the multipole approach would contain routines for simulations of solely massive particles, of massive and massless particles, of systems with a central body, and for interactions specified by the user. A routine to calculate the acceleration due to an oblate body would be useful for some simulations.
Events	Routines to establish if special events such as collisions, close encounters and orbit crossings have occurred, and for updating the data structures when particles are deleted or added.
Initial	In many simulations, the initial position, velocity and mass are generated by the integrator using pseudo random number generators. Routines are needed to generate uniform, normal and negative exponential variates.
Transformation	Routines to transform between coordinate systems, both space and time, and for applying precession and nutation.
Other	Routines for the norms of vectors and for solving Kepler's equation and polynomials (the latter arise when checking for close encounters), routines for the calculation of energy and momentum and for controlling the integration of particles to a common epoch at the start of an integration (the latter is useful because the initial conditions need not all be at the same epoch).

Table 1: List of possible routines for an N-body integrator.

[13] H. Kinoshita, H. Nakai, *Numerical integration methods in dynamical astronomy*, Celest. Mech. Dyn. Astron. **45** (1989), 231-244.

[14] M.A. López-Marcos, J.M. Sanz-Serna, Robert D. Skeel, *Explicit symplectic integrators using Hessian-vector products*, SIAM J. Sci. Comput. **18** (1997) 1, 233-238.

[15] S. Moshier, *DE118i*, `http://people.ne.mediaone.net/moshier/index.html`

[16] X.X. Newhall, E.M. Standish, Jr., J.G. Williams, *DE102: a numerically integrated ephemeris of the Moon and the planets spanning forty-four centuries*, Astron. Astrophys. **125** (1983), 150-167.

[17] T. Quinn, N. Katz, J. Stadel, G. Lake, *Timing stepping N-body simulations*, submitted to Astron. J.

[18] G. Rowlands, *A numerical algorithm for Hamiltonian systems*, J. Comput. Phys. **97** (1991), 235-239.

[19] P. Saha, S. Tremaine, *Long-term planetary integration with individual time steps*, Astron. J. **108** (1994), 1962 - 1969.

[20] L.F. Shampine, M.K. Gordon, *Computer solution of ordinary differential equations; the initial value problem*, W.H. Freeman & Co., San Francisco, 1975.

[21] P.W. Sharp, *Order five explicit second-derivative Runge-Kutta pairs with interpolants*, Report Series No. 312, Department of Mathematics, University of Auckland, 1994.

[22] P.W. Sharp, J.M. Fine, *Some Nyström pairs for the general second order initial value problem*, J. Comp. and Appl. Math. **42** (1992), 279-291.

[23] F. Varadi, *NBI*, `http://www.astrobiology.ucla.edu/ ~varadi/NBI/NBI.html`