The Art of Computational Science

The Kali Code

vol. 10

Integration Algorithms: Symplectic Schemes

Piet Hut, Jun Makino and Walter Dehnen

July 11, 2005

Contents

Preface

0.1 xxx

We thank Walter Dehnen for bringing the Chen and Chen algorithm to our attention, and for his help in applying it to the N-body problem. We thank xxx, xxx, and xxx for their comments on the manuscript.

Piet Hut and Jun Makino

Chapter 1

A Surprising Algorithm

1.1 Picking up the Pieces

Alice: We have been quite busy with our project to lay the foundations for a N-body simulation environment.

Bob: I'd say! It seems like ages since we did some actual N-body calculations. We started with the two-body problem . . .

Alice: . . . and then you got carried away, adding one integrator after another, before we finally moved on to the general N-body problem . . .

Bob: . . . at which point you told us to stop moving, and to lay foundations instead! I feel like we turned into computer scientists instead of astrophysicists.

Alice: I'm afraid we had no choice. The alternative would have been to come up with stopgap solutions at every turn in the road. Now at least we have a reliable and flexible data format and corresponding I/O routines, and we have a library structure that allows us to organize our codes. And even before we built that, we introduced extendable command line options that maked our codes self-describing through a detailed help facility.

Bob: I must admit, all those features do make life easier. I remember getting rather tired, editing a file each time I wanted to perform a different run, before we had command line options. That seems like a long time ago! Okay, where were we?

Alice: In volume ???acsio??? we had collected the various integrators in a single file `nbody_cst1.rb`, while we were getting the ACS data format straightened out. Let us start from the same file, in this new directory, corresponding to the current volume.

Bob: And let us start by calling it `nbody_cst1a.rb`, so that we can experiment with a few versions 1a, 1b, etc, until we are happy with a more stable version, which we can then call `nbody_cst1.rb` again, in this directory. At that point, we can export that version once more to our "bin/kali" directory.

Alice: Do you remember how to run `nbody_cst1a.rb`?

Bob: Don't have to! Remember, we had a `---help` option, which should give not only a detailed description of what the codes does, but in addition it should give a simple example invocation.

Alice: Ah, yes, that's one of the nifty features we added. We've sure done a lot! Let's try:

```
|gravity> kali nbody_cst1a.rb ---help

    This program evolves an N-body code with a fourth-order Hermite Scheme,
    or various other schemes such as forward Euler, leapfrog, or Runge-Kutta,
    using constant time steps, shared by all particles, where the size of
    of the time step is prescribed beforehand.  The program includes the
    option to provide softening for the potential.  This is essential for
    a constant time step code; the alternative, instead of softening, would
    be to use a variable time step algorithm.

    (c) 2005, Piet Hut and Jun Makino; see ACS at www.artcompsi.org

    example:
    kali mkplummer.rb -n 4 -s 1 | kali nbody_cst1a.rb -t 1 > /dev/null
```

Well, let's follow the advice:

```
|gravity> kali mkplummer.rb -n 4 -s 1 | kali nbody_cst1a.rb -t 1 > /dev/null
==> Plummer's Model Builder <==
Number of particles: N = 4
pseudorandom number seed given: 1
Screen Output Verbosity Level: verbosity = 1
ACS Output Verbosity Level: acs_verbosity = 1
Floating point precision: precision = 16
Incremental indentation: add_indent = 2
             actual seed used: 1
==> Constant Time Step Code <==
Integration method: method = hermite
Softening length: eps = 0.0
Time step size: dt = 0.001
Interval between diagnostics output: dt_dia = 1.0
Time interval between snapshot output: dt_out = 1.0
```

```
Duration of the integration: t = 1.0
Screen Output Verbosity Level: verbosity = 1
ACS Output Verbosity Level: acs_verbosity = 1
Floating point precision: precision = 16
Incremental indentation: add_indent = 2
at time t = 0, after 0 steps :
  E_kin = 0.25 , E_pot =  -0.5 , E_tot = -0.25
             E_tot - E_init = 0
  (E_tot - E_init) / E_init = -0
at time t = 1, after 1000 steps :
  E_kin = 0.0671 , E_pot =  -0.317 , E_tot = -0.25
             E_tot - E_init = -2.74e-08
  (E_tot - E_init) / E_init = 1.1e-07
```

1.2 The Chin and Chen paper

Alice: That all seems quite reasonable. And we do have quite a number of different algorithms implemented, by now. Shall we move on, from constant time steps to adaptive time steps, and after that, to individual time steps?

Bob: Yes, we should do that soon. However, before moving on, let me show you a paper that I stumbled upon, by Sia. A. Chin and C. R. Chen. I found it on astro-ph, under *astro-ph/0304223(*`http://arxiv.org/abs/astro-ph/0304223`*)*. This paper presents an algorithm that is totally different from what I've seen so far.

1.3 xxx

$$\frac{d^2}{dt^2}\mathbf{r}_i = G\sum_{j\neq i} M_j \frac{\mathbf{r}_j - \mathbf{r}_i}{|\mathbf{r}_j - \mathbf{r}_i|^3} \tag{1.1}$$

We will omit the term G by putting it equal to unity; we can always restore it, whenever we want, by applying dimensional analysis.

Let me remind you how this equation can be derived, starting with the gravitational potential

$$V(\mathbf{r}_i) = -\sum_{j\neq i} \frac{M_j}{r_{ij}} \tag{1.2}$$

with

$$r_{ij} = |\mathbf{r}_{ij}| = |\mathbf{r}_j - \mathbf{r}_i| \tag{1.3}$$

The equations of motion are defined as:

$$\frac{d^2}{dt^2}\mathbf{r}_i = -\nabla_{\mathbf{r}_i} V(\mathbf{r}_i) \tag{1.4}$$

Let us write these last two equation in terms of vector components, in order to make it absolutely clear what we are doing. If we are working in D dimensions, we can write $\mathbf{r}_i = \{r_{i,1}, r_{i,2}, \ldots, r_{i,D}\}$, and similarly for the other vectors. The last equations then become:

$$r_{i,j} = \sqrt{\sum_k \left(r_{i,k} - r_{j,k}\right)^2} \tag{1.5}$$

and

$$\frac{d^2}{dt^2} r_{i,k} = -\frac{\partial}{\partial_{r_{i,k}}} V \tag{1.6}$$

In order to evaluate the right-hand side of this last equation, we will need to compute

$$\begin{aligned}
\frac{\partial}{\partial_{r_{i,k}}} r_{ij} &= \frac{\partial}{\partial_{r_{i,k}}} \left[\sum_{k'} \left(r_{i,k'} - r_{j,k'}\right)^2\right]^{1/2} \\
&= \tfrac{1}{2} \left[\sum_{k'} \left(r_{i,k'} - r_{j,k'}\right)^2\right]^{-1/2} \frac{\partial}{\partial_{r_{i,k}}} \left[\sum_{k'} \left(r_{i,k'} - r_{j,k'}\right)^2\right] \\
&= \tfrac{1}{2} \frac{1}{r_{ij}} \sum_{k'} 2\left(r_{i,k'} - r_{j,k'}\right) \frac{\partial}{\partial_{r_{i,k}}} \left(r_{i,k'} - r_{j,k'}\right) \\
&= \tfrac{1}{2} \frac{1}{r_{ij}} \sum_{k'} 2\left(r_{i,k'} - r_{j,k'}\right) \delta_{k,k'} \\
&= \frac{1}{r_{ij}} \left(r_{i,k} - r_{j,k}\right)
\end{aligned} \tag{1.7}$$

where we have used the fact that $i \neq j$ and therefore

$$\frac{\partial}{\partial_{r_{i,k}}} r_{j,k'} = 0 \quad \text{and} \quad \frac{\partial}{\partial_{r_{i,k}}} r_{i,k'} = \delta_{k,k'} \tag{1.8}$$

When we write this again in vector notation, the expression becomes more compact:

$$\nabla_{\mathbf{r}_i} r_{ij} = \frac{1}{r_{ij}} \mathbf{r}_{ij} \tag{1.9}$$

We can now complete our derivation of the equations of motion for the N-body problem:

$$\begin{aligned}
\frac{d^2}{dt^2}\mathbf{r}_i &= -\nabla_{\mathbf{r}_i} V(\mathbf{r}_i) \\
&= \nabla_{\mathbf{r}_i} \sum_{j\neq i} \frac{M_j}{r_{ij}} \\
&= \sum_{j\neq i} M_j \nabla_{\mathbf{r}_i} r_{ij}^{-1} \\
&= -\sum_{j\neq i} M_j r_{ij}^{-2} \nabla_{\mathbf{r}_i} r_{ij} \\
&= -\sum_{j\neq i} M_j r_{ij}^{-2} \left\{ \frac{1}{r_{ij}} \mathbf{r}_{ij} \right\} \\
&= -\sum_{j\neq i} M_j r_{ij}^{-3} \mathbf{r}_{ij} \\
&= \sum_{j\neq i} M_j \frac{\mathbf{r}_j - \mathbf{r}_i}{|\mathbf{r}_j - \mathbf{r}_i|^3} \qquad (1.10)
\end{aligned}$$

1.4 xxx

$$\tilde{V} = V + \frac{1}{48}\tau^2 U \qquad (1.11)$$

$$\tilde{\mathbf{a}} = \mathbf{a} + \frac{1}{48}\tau^2 \mathbf{b} \qquad (1.12)$$

$$U(\mathbf{r}_i) = \frac{1}{M_i} \sum_j M_j^2 a_j^2 \qquad (1.13)$$

where, as usual, $a_j = |\mathbf{a}_j|$, which implies

$$a_j^2 = (\mathbf{a}_j \cdot \mathbf{a}_j)^2 \qquad (1.14)$$

1.5 xxx

Chapter 2

XXX

2.1 xxx

Bob: At first the Chin/Chen derivation seemed to work only if I would arbitrarily change forces into accelerations. I thought that I might have made a mistake in my implementation, since I had only tested it for equal masses, but when I tested it for the Pythagorean problem, my old implementation was fourth-order, and the newly derived implementation was only second order. I was baffled.

Chapter 3

Literature References

Forward Symplectic Integrators for Solving Gravitational Few-Body Problems,
by Sia. A. Chin and C. R. Chen, 2003, preprint, submitted to Astronomical Journal; see also the *astro-ph/0304223(*`http://arxiv.org/abs/astro-ph/0304223`*)* preprint.