Astrophysical Simulations – Project Assignment
N-body problem with various integrators

October 2018

Peter Camps
peter.camps@ugent.be
S9, 1st floor, office 110.014

1 Assignment

These are the project's key objectives:

1. Write a C++ program to simulate the time evolution of N-body systems in three dimensions, for arbitrary but small N.
2. Implement various integration schemes including Runge-Kutta-4, embedded Runge-Kutta, Verlet and Forest-Ruth integrators; provide a variable or adaptive time step alternative for one of these integrators.
3. Construct a number of relevant initial conditions with up to seven bodies, and study energy conservation and accuracy of the calculated orbits for these initial conditions with the various integration schemes.
4. Compare the accuracy and cost characteristics of the integration schemes.
5. Report on your results and experiences in a formal slide show and live presentation.

The following sections provide some more information on the tasks to be accomplished.

2 Mathematics

Basic equations

Consider N point-like bodies $i = 1 \dots N$ with masses m_i gravitating about each other (the system's center of mass is stationary). Denoting the respective position vectors as $\boldsymbol{r}_i(t)$ the equations of motion can be written as

$$\ddot{\boldsymbol{r}}_i = -\sum_{j=1,j\neq i}^{N} \frac{Gm_j(\boldsymbol{r}_i - \boldsymbol{r}_j)}{|\boldsymbol{r}_i - \boldsymbol{r}_j|^3}$$

where G is the gravitational constant. The total energy of the system is given by

$$\mathcal{E} = \sum_{i=1}^{N} \frac{1}{2} m_i \left|\dot{\boldsymbol{r}}\right|^2 - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1,j\neq i}^{N} \frac{Gm_im_j}{|\boldsymbol{r}_i - \boldsymbol{r}_j|}$$

Discretisation

To apply our integration methods, we rewrite the equations of motion as a set of first-order ordinary differential equations of the general form $\dot{\boldsymbol{y}} = \boldsymbol{g}(\boldsymbol{y})$, where $\boldsymbol{g}(\boldsymbol{y})$ is called the driver function:

$$\begin{aligned} \dot{\boldsymbol{r}}_i &= \boldsymbol{v}_i \\ \dot{\boldsymbol{v}}_i &= \boldsymbol{a}_i \equiv -\sum_{j=1, j\neq i}^{N} \frac{G m_j (\boldsymbol{r}_i - \boldsymbol{r}_j)}{|\boldsymbol{r}_i - \boldsymbol{r}_j|^3} \end{aligned}$$

We use this general form $\dot{\boldsymbol{y}} = \boldsymbol{g}(\boldsymbol{y})$ to implement the various integration methods.

Integration schemes

The various integration schemes are described in the course syllabus. Note that the indices used in these schemes indicate discretised time, while the indices in the equations of motion above indicate the participating bodies.

Variable time step

Aside from the adaptive time step methods discussed in the course syllabus, one can implement a simple scheme to vary the time step in the special case of N-body motion. We know that the accelerations and velocities increase when bodies closely approach each other, and thus it is meaningful to reduce the time step in that situation. For example, one could determine the time step h (for all bodies) as a function of the smallest separation d_{min} between the bodies. In units where the gravitational constant $G = 1$, a workable scheme is

$$h = h_{\text{base}} \times \min(1, d_{\text{min}})$$

where h_{base} is a constant set by the user. We call this a "variable time step" scheme in contrast with the more general "adaptive time step" methods.

3 Implementation

Completing the steps described in this section should lead to a successful project implementation. There is no firm prescribed ordering, but there are important dependencies; for example, you need to implement an integrator before you can study its behavior. Other than these dependencies, you may choose to complete steps in a different order, and members in your group may work on different steps in parallel.

Regardless of how you plan and distribute the work, define milestones and complete them one by one. Compile and test your source code often, and perform more thorough tests after each milestone. Make a backup after each milestone so you can revert to a working version if things go wrong. *If you try to complete the assignment in one go, you will surely fail, and it will take a miracle to locate the problems in your code.*

Program

1. Develop and test a data type (e.g. a "Vec" class) to represent vectors in three-dimensional space. This will make the rest of your source code much easier to develop and maintain.

2. Write a program that solves a two-body system using, for example, the fourth-order Runge-Kutta (RK4) integrator and a fixed time step. Provide proper output and visualization tools to show orbits and track the evolution of the relative energy error. Verify the results for various initial conditions.

3. Define a simple text file format to describe the initial conditions for an N-body system. Find or invent initial conditions of some interesting few-body systems with $N \lesssim 7$.

4. Extend your program and visualization tools to handle N bodies, still using RK4 with fixed time steps. Verify the results for various initial conditions. Identify ways to convince yourself that your solution for these systems is correct, other than comparing to a reference solution available from other sources.

5. Measure execution time per integration time step for increasing N and plot this against the expected scaling behaviour (while your code should work with any number of bodies, in the context of this project you do not need to implement any optimizations for large N).

6. Extend your program to also implement other (mandatory or optional) integration schemes as an alternative to the RK4 integrator, to be selected by the user at run time. Do this in steps. For example, implement one integrator at a time and verify the results for various initial conditions before moving on.

7. Implement and test a variable or adaptive time step scheme for at least one of the integrators (varying the time step h for all bodies simultaneously).

8. For all integrators, add a mechanism to your code to output the number of driver function evaluations per unit of simulated time, averaged over the complete integration. This number serves as a measure for the cost of the calculation.

9. For some relevant initial conditions, study the evolution of the relative energy error (accuracy) and the number of driver function evaluations (cost). Compare the accuracy and cost of the various integration schemes, and for each scheme study the effects of changing the time step (for fixed time step schemes) or the parameters for varying the time step (for variable or adaptive schemes).

10. For a plain semi-eccentric two-body system, compare the long-term behavior of the integrators over many thousands of orbits. Which integrator would you recommend?

11. Create one or more movies animating your solutions. After all, this is why you were going through so much trouble!

Presentation

1. Prepare to discuss your results and experiences in a live presentation of 20-25 minutes per group (and *not* longer!). Focus on the important issues. Show your results, your experiments and tests. Explain the motivations for the choices you made.

2. Create a formal slide show to serve as the backbone for your live presentation. Include a lot of visual material, such as plots or diagrams. Movies are always very much appreciated! Don't show source code in the presentation.

3. Be prepared to answer questions after your presentation. It is very important that you understand why your code operates the way it does. A nice result is not always a physically correct result.

Project groups

The composition of the project groups has been announced via Minerva. Most groups have four members. Some groups have three or five members because the number of students is not necessarily a multiple of four. It is not allowed to switch between groups.

Evaluation

The project is graded on 10 points (the theory exam accounts for the other 10 points). The evaluation takes into account the C++ source code (structure, quality, comments), the project presentation (presentation skills, quality of slides, plots), the presented results (correctness, parameter study, extra's), and insight in the physical processes and numerical methods.

In addition, each group member is expected to peer-review the other members in her/his group through an electronic form on Minerva (link will be announced via Minerva). The peer review is anonymous to the other group members. The results of the peer review and the individual performance during the presentation will be weighted into the individual grades. However, the difference between the lowest and highest individual grade within the same group will not exceed 3 points (except for manifestly exceptional cases, e.g. when a group member has not participated in the project at all).

Deliverables and deadlines

1. We expect you to hand in your C++ source code for the project by a very specific deadline (usually three days in advance of the presentations). Provide any special instructions needed for using your code. Visualization code does not need to be handed in (and will be ignored anyway).

2. We expect you to perform the peer review procedure by a very specific deadline (usually one day in advance of the presentation).

3. During the live presentation bring your slide show and any related visual material. Preferably use your own laptop. You're free to use any presentation software running on any operating system. Bring a copy of your presentation on a USB stick (in original format and in PDF) as a backup in case we can't get the projection to work from your laptop. You do not need to hand in the presentation files.

The precise deadlines and mechanisms for handing in the C++ code and for performing the peer review, and the time schedule for the live presentation will be announced via Minerva.