Solar System: Background

Computer Modelling

2023/2024

Aims

In Semester 2 you will write code to describe an N-body system interacting through Newtonian gravity. The code will, eventually, simulate the motion of the main bodies of our solar system, starting from realistic initial conditions. Your code will enable you to compare simulation results to astrophysical data.

After finishing your source code you will use it for the final piece of coursework, and write a report on a computational experiment using it.

This document gives general background information for the tasks below. You should read it once before starting, and then refer back to it throughout the parts of the project.

1 Gravity

Simulations of gravitational systems are a key tool in modern astrophysics. Within our solar system, Newtonian Gravity is sufficient for almost every observation we can make.

The gravitational force on a body of mass m_1 at position $\boldsymbol{r_1}$ from a body with mass m_2 at position $\boldsymbol{r_2}$ is given by

$$\boldsymbol{F}_{12} = -Gm_1m_2\frac{\boldsymbol{r_1} - \boldsymbol{r_2}}{|\boldsymbol{r_1} - \boldsymbol{r_2}|^3} \tag{1}$$

and the gravitational potential of the two by:

$$U_{12} = -\frac{Gm_1m_2}{|\boldsymbol{r_1} - \boldsymbol{r_2}|} \tag{2}$$

2 N-Body simulations

When we upgrade simulations from two bodies (like the ones we simulated in semester 1) to arbitrary numbers of particles, there are several important changes we have to understand.

First the force on any one particle is now the sum of the forces on it from all the other particles:

$$\boldsymbol{F}_i = \sum_j \boldsymbol{F}_{ij} = -Gm_i\sum_j m_j\frac{\boldsymbol{r_i} - \boldsymbol{r_j}}{|\boldsymbol{r_i} - \boldsymbol{r_j}|^3} \tag{3}$$

We have to be more careful with the potential energy, and avoid double counting, as the potential on A due to B should not be added to the potential of B due to A - this is the same energy. There are a few ways of implementing this. You can either do the complete sum and then divide by two, or avoid the double count in the first place:

$$\begin{align} U &= \frac{1}{2}\sum_i\sum_{j\neq i} U_{ij} \tag{4}\\ &= \sum_i\sum_{j>i} U_{ij} \tag{5}\end{align}$$

Note that in each case we must avoid the potential of a particle with itself.

3 Units

The choice of units to use in a simulation is important - the simulation can be more accurate and easier to understand if the units give values which within a few orders of magnitude of one. In solar system simulations, using metres or seconds would lead to very awkward numbers.

In this simulation you shall adopt the units:

- astronomical units (AU) for length,
- Earth days for time
- Earth mass for mass

To do this we must make sure the constant G is in these units[1]. G in SI units is $6.67430(15)\times10^{11}\,\mathrm{m}^3\,\mathrm{kg}^{-1}\,\mathrm{s}^{-2}$. This corresponds to $G = 8.887692593\times10^{-10}\,\mathrm{AU}^3\,{\mathrm{M}_{\mathrm{earth}}}^{-1}\,\mathrm{day}^{-2}$.

4 N-Body integration

The Verlet integration scheme applies to N-body systems just like it applied to the two-body problem you ran last semester. An important thing to keep in mind, though, is the order of the updates you do to the different bodies.

The Verlet update steps are:

1. Update positions using the current forces f_1:
   $x_{i+1} = x_i + v_i\delta t + f_1\delta t^2/2m$
2. Calculate forces at the new positions f_2:
3. Update the velocities using forces $\frac{1}{2}(f_1 + f_2)$
   $v_{i+1} = v_i + (f_1 + f_2)\delta t/2m$

In N-body systems it is crucial to update **all** the particle positions at once in step 1, then update all the forces in step 2, then update all the velocities in step 3.

If you try to loop through the particles running steps 1-3 on each in turn then the algorithm is no longer correct and fails badly at reasonable time steps.

[1] One advantage of our unit choice is that, while it is hard to measure precisely both G and $\mathrm{M}_{\mathrm{earth}}$ separately, their product is known to 10 significant figures.

5 Centre-of-mass corrections

The initial conditions of an N-body system are unlikely to have zero total linear momentum. This leads to an undesirable drift during long simulations. To remove this, we subtract the centre-of-mass velocity from each particle's initial velocity. If the bodies have initial velocities $\boldsymbol{v}_i$, then the simulation has a total momentum of:

$$\boldsymbol{P} = \sum_i m_i \boldsymbol{v}_i \tag{6}$$

If we subtract the centre-of-mass velocity $\boldsymbol{v}_{\text{com}}$, which is

$$\boldsymbol{v}_{\text{com}} = \frac{\boldsymbol{P}}{\sum_i m_i} \tag{7}$$

from each body's motion at the start of the simulation, then there will be no drift to the overall system.

6 Energy

The total energy of the system is the sum of the potential and kinetic energies:

$$E = U + \sum_i \frac{1}{2} m_i |v_i|^2 \tag{8}$$

Energy is conserved in a perfect simulation; in an approximate one like this it will vary slightly throughout the integration. Ideally the level of that variation will be small. We quantify this with the fractional deviation:

$$\frac{\Delta E}{E_0} = \left| \frac{E_{\text{max}} - E_{\text{min}}}{E_0} \right| \tag{9}$$

where E_0 is the energy at the start of the simulation. If

$$\frac{\Delta E}{E_0}$$

is much less than 1 then the energy is conserved to a high level of accuracy.

7 Astrophysical Observables

There are various measured orbital properties that we can compare to our simulation:

- Perihelion and aphelion, the smallest and largest distance of a planet from the sun,
- Orbital periods, the time for a complete orbit,
- For the moon, perigee and apogee, the min and max distances from the Earth.

The word "periapsis" means either "perigee" or "perihelion" depending which body is being orbited.

The semi-major axis of an orbit around the sun is half of the sum of the perihelion and aphelion.

8 Trajectory Files

Trajectory files in the standard "XYZ" format are often used within scientific modelling to represent a set of points (in three-dimensional Cartesian space) at a number of different steps within an ordered time series. Once the data has been stored in a trajectory file we can then visualise it in a number of ways, for example by animating it. The trajectory file can be used to perform further analysis, or to compare different runs, as well as for artistic purposes.

The XYZ format (for e.g. two particles) looks like this:

```
2
Point = 1
s1 x11 y11 z11
s2 x21 y21 z21
2
Point = 2
s1 x12 y12 z12
s2 x22 y22 z22
⋮
2
Point = m
s1 x1m y1m z1m
s2 x2m y2m z2m
```

You can see that the file consists of m repeating time steps (where m is the number of steps in the trajectory) and that each unit consists of two header lines: the first specifies the number of points to plot (this should be the same for each unit, and is "2" above) and a comment line (in the example above it specifies the point number). Following that, the file specifies the label and the Cartesian coordinates of each particle at this time step.

9 Command Line Interfaces

It's annoying to have to modify the code whenever you want to change a parameter in the code, such as a time-step size, or a file name to use. It also makes it more difficult to keep track of things. So it is usual to provide a command-line interface to your code to let users specify the inputs to use.

In spyder, you specify command line inputs by typing in the console window. If you do this:

```
%run my_code.py abc 1.0
```

Then in your python code if you import the built-in module called `sys`, then `sys.argv` will be a list of strings of what you typed:

```
import sys
print(sys.argv)
```

This would print out

```
["my_code.py", "abc", "1.0"]
```

The first entry is just the name of the code, which is not normally useful, but you can use the others to control your code. You often need to convert parameters from strings to ints or floats.

Note: This is not the same as using the python `input()` function to ask the user to type the inputs each time. That gets increasingly annoying, because you can't press the UP arrow to re-run the last thing again, which is crucial for debugging.

10 Convergence

A *convergence test* is a check that a particular approximation that is made in a code is not significantly affecting the results of the code.

In our case, the key convergence test is the size of the δt time-step size. We want to know if the value of δt is small enough for this not to be causing a significant error in the simulation.

We do this by examining observables, like periods and apsides, and theoretical quantities, like energy, derived from the code. Because we've made lots of other approximations, though, like neglecting many moons and minor bodies, we cannot check this just by comparing to real observational data. Instead, we compare to a high-precision version of the approximation.

If your code is working then as δt decreases the values of observables should converge towards the same value they have with an extremely small δt value, like $\delta t = 10^{-5}$. Convergence tests demonstrate that this is true for a range of different observables, and find a larger δt value where observables are close enough to their converged value for the specific analysis being done.

11 Submissions

Your submitted code will be anonymised and then uploaded for a plagiarism check using MOSS. Ensure none of your files include your exam number (e.g. B0000). Also ensure your names and matriculation numbers appear only on the headers of the python files, and not elsewhere.

Unit 1's submission is a single file. From unit 2 onward you will submit multiple files in a directory. In each case you must zip them like this:

Windows Right-click on the folder, and under the "Send-to" menu, select "Compressed (zipped) folder". You can then rename the zip file that is created.

Mac Ctrl-click on the folder and click "Compress name_of_folder". You can then rename the zip file that is created.

Linux Run this in a terminal, replacing the project directory name with whatever you are using: `zip -r submission.zip project_directory_name`

If you don't package your files correctly you may lose marks.