Astrophysics: N-Body Simulation

3rd year Physics Laboratories

Last compiled January 15, 2018

Contents

1 Background

It is a remarkable and sobering thought that it is impossible to solve the equations for three or more bodies flying around in space under the influence of their mutual gravity. In special cases you can make useful approximations, but in general, the problem is insoluble. This is a major menace in astronomy, which after all is the study of large numbers of objects flying around in space under the influence of gravity.

In the late 18th century, a partial solution was found. While it is impossible to calculate the orbits exactly, you can calculate a good approximation valid for a short period of time. You start by putting all your bodies in their starting positions. You then work out the gravitational force on each object at this time. Using this force, you work out the acceleration on each body. Using this acceleration, you work out the velocity of each body. Using this velocity, you work out where each body will be a short time in the future. You move each body to its new position. You then go through the whole process again, starting at the new position.

In this slow, tedious way you can calculate trajectories for even the most complex situations, though thousands of calculations are required. In the past, teams of students would sit at rows of desks, each calculating one tiny stage over and over again, these were the original computers. Though it

sounds like a monstrous waste of time, the rewards were great – it was from calculations such as these that the planet Neptune was discovered.

Luckily for you, computers have evolved from rooms full of hapless students into the silicon chip-based devices we know and love. They are perfectly suited for this job; computers love nothing more than doing simple calculations over and over again. In this exercise, you will use a computer program that will do in seconds the same calculations that took some of the greatest astronomers of the 19th and 20th centuries most of their lives.

2 N-body Codes

The program you will be working with is one of a class of programs called N-body codes. These codes simulate the motion of compact, massive particles, moving under the influence of their mutual gravitational attraction. N-body codes are big business in astrophysics, where they are used for diverse purposes, perhaps most notably cosmological simulations of galaxy formation. The Illustris simulation project (http://www.illustris-project.org/) is a recent suite of simulations that modelled the motion of billions of particles from the beginning of the Universe until close to the present day. A similar project currently ongoing here at the University of Melbourne is the DRAGONS project, led by

Professor Stuart Wyithe (http://dragons.ph.unimelb.edu.au/index.html).

The program you will be using is somewhat cruder than the state of the art, but the principle is the same as for all N-body codes.

Warning! Do not attempt to solve the equations on paper – there is no general solution to the three-body problem, and even limited, special-case solutions are hair-raisingly difficult. If you find yourself writing down pages of equations, you're doing this the wrong way. All the maths you need can be written in about three lines!

2.1 Algorithm

How can we simulate particles moving in orbit around each other? The physics is simple; we use Newton's inverse square law of Gravity,

$$F = \frac{GM_1M_2}{r^2} \tag{1}$$

where F is the force, $G = 6.67 \times 10^{-11}$ $\mathrm{Nm^2kg^{-2}}$, and $\mathrm{r} = r \times \hat{\mathrm{r}}$ is the vectorial separation of the two objects. Assume throughout that all the objects can be treated as points; for example if the Earth is in orbit around the Sun, assume that all the mass of the Sun is at the centre of the Sun for calculation purposes. We proceed as follows:

1. We choose initial positions and velocities for all our objects.

2. Using Newton's law (above) we work out the force on each object due to the gravitational attraction of the others. The force is a vector quantity, and as gravity is linear, the total force on each object is the vector sum of the forces on it due to each of the other bodies.

3. Using the force we just calculated, we work out the acceleration of the body.

4. We now take a small step forward in time, and compute the new position and velocity of all the objects. The new position is simply the old position plus the product of the velocity and the time step. The new velocity is the old velocity plus the product of the acceleration and the time step.

5. Repeat steps 2—4.

Simple! The key to computer programming is to remember that computers are pretty stupid. They can only do simple things, but they do them very fast. Break your problem into tiny little pieces, and get the computer to do them one after another very fast.

2.2 Accuracy

"How big should our time step be?". "How long should we run our code for?". These are the most common questions demonstrators get asked, and there is no easy answer. In the algorithm above, you've approximated the true, curved trajectories of the bodies with a series of straight lines. Every time you take a step, your final position will be slightly in error, and if these errors are too large they can compound one another until your program gives you complete garbage.

It is difficult to know in advance what time step to use for a numerical problem. The only way to know is to see how changing your time step affects your results. For example, in section 4 below you will simulate the Earth going round the Sun. If your program doesn't start with the Earth in orbit with a period of about a year then you may have set the code up incorrectly, but if your orbit starts out fine and then starts to go haywire you most likely need to reduce your time step. If you decrease the time step too far, however, you will never be able to run enough orbits to see if your simulation is working properly. By adjusting the time step in a careful and systematic fashion for the Earth-Sun system, you should be able to find a happy medium between accuracy and computation time. Unfortunately, a time step that works for the Earth and the Sun won't necessarily work for all the other scenarios in this exercise,

you will need to perform the same analysis in each case and include them in your lab report.

What will happen if your objects get close to each other? If the distance they travel in a time step is comparable to the distance between two objects, you've got trouble (why?). There are ways to fix this in the code, but the simplest way out is not to let the objects get this close; if you have to have close encounters in your simulation, use very small time steps.

Step 4 above is the most obvious way to step positions/velocities forward in time, and is known as Euler's method. Leonhard Euler was a famous Swiss mathematician who died in 1783; in the subsequent 250 years more sophisticated techniques have been developed. One such technique which is very popular in simple applications is the fourth-order Runge-Kutta method (RK4) – if you have taken the Computational Physics course you should be familiar with Runge-Kutta methods. If you have some experience with programming, try to adapt the nbody.py code described below to use RK4.

2.3 Exploring Parameter Space

In all the projects below, you have a wide choice of free parameters. What are the starting masses? What are the starting positions and speeds? You may feel that you have

been given far too little guidance about these things. You are right, but this is a problem professional astronomers face all the time. There are no definite answers, what you have to do is explore parameter space. There will be some plausible range of input parameters. Explore it! Try out different values; home in on the most interesting ones. This is an important and difficult skill, and a lot of marks will depend on how well you do (and document) it.

2.4 Units

You are probably used to working in S.I. units. These are fine for things of order 1m long (like you and me), but for your average star (mass $\sim 10^{30}$ kg), you've got problems. Most computers can handle numbers as large as $10^{\pm 34}$, but all you would need to do is multiply the mass of the Earth and the Sun together, and the program would crash. The simplest solution is to invent new units for mass, length and time. If you choose these units correctly, your program will be dealing with numbers like 1, rather than 10^{30}.

3 Programming

3.1 The program

The computer for this lab uses Windows and a Python interpreter, iPython. Create a folder for yourself in the astro_comp folder on the Desktop (for example, bob). Double-click the iPython icon on the Desktop – this will open a terminal that you can type commands into. Type the following commands:

> cd
> cd Desktop/astro_comp/bob

The code for this lab is written in Python. Python is a programming language that is widely used in astronomy. The sample code is located in astro_comp, and is called nbody.py. Copy this file into your folder in astro_comp.

nbody.py has the basic N-body code that will calculate the positions and velocities of the bodies for a given number of time-steps. This has been provided so that you can concentrate more on the actual science involved, rather than spending most of your time writing programs. You will, however, need to do a little bit of programming, to calculate things like the total energy of the system, and the planetary temperature and so forth. In this case, the following section may be of some use.

3.2 Programming Hints

Talk to your demonstrators for lots of hints on good programming style. But here are a few key hints:

1. Write out your code on paper before you type it in. The most common error novice (and experienced) programmers make is to write half the code before realising that they don't really understand in detail the algorithm they are trying to code. Make sure you are absolutely clear in your mind about what you want to do before touching finger to keyboard.

2. Write clear, well commented code. Far more time is spent debugging programs than writing them in the first place. If your code is clearly and logically laid out, with lots of comments, it will save you hours in the long term.

3. Give your variables sensible names. If all your variables are called things like XAA you will have to look up what they mean every time you have to alter your code, but something called XVELOCITY is a little more obvious.

4. Test your code. No program longer than about ten lines ever works first time. If you test each little part of your program independently, it will be much easier to debug. Also, never believe what your program is

telling you until you've tried it on a problem where you know the answer.

4 Project: N-body dynamics in the Solar System

This exercise is set up like a mini research project. You should work through the theory, and make sure you understand what you will be modelling with the program. Next, familiarise yourself with the N-body code provided, making sure you can get it to work. Then, do the exercises examining the motion of massive objects interacting via gravity.

4.1 Theory

Write down the basic equations governing motion – those for force, acceleration and so on. Write these in terms of vectors (using $\vec{r_1}$ and $\vec{r_2}$ as the position vectors of two particles), and separate out into x, y, and z components. Now rewrite them using discrete time steps Δt. These are the equations that will be used in a computer program.

Draw up a flow chart of how an N-body algorithm will work. This does not mean write a program! Rather, write the steps required to calculate the necessary forces, positions

etc. at each time-step. (This should be more detailed than the basic algorithm given in Section 2.1) Also, think about what units to use – for instance, what would you measure the distance in? Why? You will need to calculate the gravitational constant G in your new units. Wait for a demonstrator to check what you have done before continuing.

4.2 Code testing

Now look at the template code that is provided on the computers (this code is written in the Python programming language). Match up what the code does with your flow chart. How well do they agree? Make your own copy of the template, so that you can alter it as you go along. Please don't edit the template file!

Run the code using the iPython terminal:

> run nbody.py

Take a look at the outputs of your code. What is the motion of the Earth and Sun? Is this what you expect, given the initial conditions?

For your first exercise, find the speed at which the Earth rotates around the Sun, and set its initial velocity to this speed. Is the orbit stable? If not, what change to the initial conditions should you make?

Try to determine how accurate your code is. Give some thought as to how you can work out the errors in your results, and how you can make your program more accurate. In doing this, explore the parameter space a little and get a feel for the range of parameters for which things still work well. Things to try would be :

- Change Δt. (What is the physical interpretation of a large or small Δt?)
- Try increasing/decreasing the initial velocity by a small amount and/or a significant amount (order of magnitude perhaps?) – what happens? Calculate the escape velocity of the Earth, what happens if your initial velocity is near (above and below) this value?
- Find a way to work out how many orbits the planet is doing. – how many orbits can you get in a sensible time? (Make this independent of the initial parameters – it needs to be adaptable in case the orbit changes e.g. due to the influence of a third body.)
- Change the number of orbits your Earth makes. Does it remain stable?
- Check whether energy is being conserved by your program – what is the expression for the total energy in the system? If conservation is not taking place, explain why.

Finally, convert your program to a 3-body problem by adding Jupiter to the simulation. Repeat your investigation of 2-body dynamics for the Sun-Earth-Jupiter system.

See appendix for information on calculating the temperature at Earth (as this is required for both projects).

5 Project: Black Holes and the End of the Earth

5.1 Background

What would happen if a black hole wandered past our Solar system? Would it suck the Earth into its orbit and carry us away? Would it disturb our orbit and make us fall into the Sun, or fling us out into interstellar space to slowly freeze? This project aims to find out.

5.2 Program

For this exercise you will use the 3-body code you have tested in section 3.

5.2.1 Initial Conditions

Start with the Earth in its usual nice steady orbit around the Sun. Fling in the black hole from a range of directions and orientations. See what happens if it gives the Sun a near miss, or passes by much further out.

How massive should the black hole be? Dying stars end up as either white dwarfs, neutron stars or black holes, depending on their initial mass and the conditions of their death. Both neutron stars and black holes are thought to form in Supernova explosions, with the final product determined by the initial mass of the dying star. Neutron stars have a maximum mass of $\sim 2-3$ solar masses, after which they collapse to form a black hole. This gives a lower limit on the range of masses you should play with. There is no upper limit to the size of a black hole, but let's assume that our wandering black hole was formed from the collapse of a star and has not grown by merging with other black holes since then. Such a black hole could conceivably weigh thousands of times more than the Sun, but since stars massive enough to form such heavy black holes are rare, moderate masses are much more likely. Concentrate on the effects of smaller black holes, but run a few simulations with the really big ones just to find out what happens.

How fast will the Black Hole approach? Stars in our part of the galaxy are typically moving at speeds of about 200 km/s, in their orbits around the centre of our galaxy. If

the black hole were in a similar orbit to our Sun, it might sidle up relatively slowly, say 10 km/s. If however it was rotating the other way around the galactic centre, it might be travelling at 400 km/s, and if it was an interloper from intergalactic space, the speed might be higher still.

5.3 Exercise

See where the Earth ends up after each Black hole encounter. If it ends up in orbit around the Black hole, or flung into interstellar space, it is obviously curtains for life on Earth. But what if it ends up in a different and/or distorted orbit around the Sun? Get your program to work out the temperature of the Earth at each point. Assume it is a black body whose only heat source is the Sun, and that it is in thermodynamic equilibrium. Is this assumption reasonable? If we were flung out into interstellar space, how long would we take to cool down?

5.4 Discussion

1. Write a brief summary of the results of your investigation. Make sure you answer the aims you had at the start.

2. What happens when two bodies got too close to each other? What does Newton's equation predict? Is

this scenario physical? How could you make your calculations more realistic?

3. Did you notice any change in speed while running the program, particularly when going from two bodies to three? Do you think this is a realistic method for N-body simulations where $N \sim 100$?

6 Numbers

- An astronomical unit (AU, the distance from the Earth to the Sun) is 1.496×10^{11} m.
- The radius of the Earth ($r_\oplus$) is 6380 km
- One parsec (pc) is 3.086×10^{16}m (3.26 light years)
- The distance between the Sun and the Kuiper belt is between 40 and 100 AU.
- The mass of the Earth ($m_\oplus$) is 5.97×10^{24} kg.
- The mass of the Sun ($M_\odot$) is 2.00×10^{30} kg.
- The luminosity of the Sun ($L_\odot$) is 3.8×10^{26} W.
- The temperature of the Sun ($T_\odot$) is 5800 K.
- The radiation temperature of deep space is 2.7 K.

Appendix – planetary temperatures

To calculate the temperature of a planet, we assume that both the planet and the Sun are blackbodies, and we find the temperature at which the little blackbody must radiate to balance the energy input from the big blackbody.

Assuming the Sun can be approximated by a blackbody, the energy flux is given by the Stefan-Boltzmann law

$$F = \sigma T^4 \text{ W m}^{-2} \tag{2}$$

where $\sigma = 5.67 \times 10^{-8}$ W.m^{-2}.K $^{-4}$.

This can be converted into the luminosity – the total power output – by multiplying by the surface area of the Sun

$$L_\odot = 4\pi R_\odot^2 F_\odot = 4\pi R_\odot^2 \sigma T^4 \text{ W} \tag{3}$$

Since this energy all flows through a sphere of radius r_p, the planet-Sun distance, the flux at r_p must be given by

$$F_p = \frac{4\pi R_\odot^2 F_\odot}{4\pi r_p^2} = (R_\odot / r_p)^2 F_\odot$$

Now, it would be possible to calculate the temperature straight from F_p, which is called the subsolar temperature. But, this is not the appropriate value for most planets, which have atmospheres, or reasonably rapid rotation. For

these, we need to calculate the equilibrium temperature, which takes into account the energy radiated by the planet, as well as the albedo, which is the fraction of solar radiation incident on the planet that is reflected.

The albedo is denoted by A, and if A is the fraction reflected, then $1 - A$ is absorbed. This means that power absorbed by the planet is

$$(1 - A)\pi R_p^2 F_p = (1 - A)\pi R_p^2 (R_\odot / r_p)^2 F_\odot$$

since the cross section of the planet is πR_p^2, where R_p is the planet's radius. For an equilibrium temperature, the absorbed power must equal the radiated power, which is given by $4\pi R_p^2 \sigma T_p^4$.

Thus, equating these two expressions, we get the temperature to be

$$T_p = (1 - A)^{1/4} (R_\odot / 2r_p)^{1/2} T_\odot$$

or, expressing r_p in astronomical units (AU),

$$T_p \approx 279(1 - A)^{1/4} (r_p)^{-1/2}$$

Note that this temperature does not take into account effects such as the heat retention of atmospheres (e.g. greenhouse effects), internal heating of planets, or variations of A with wavelength. Typical values of A for planets are 0.4 (Earth), 0.76 (Venus), 0.16 (Mars), and 0.51 (Jupiter).

[Adapted from Zeilik & Gregory, "Introductory Astronomy and Astrophysics", 4th Edition, Saunders College Publishing.]

7 Acknowledgements

Original document: Paul Francis, Chris Fluke and Matt Whiting.
Updates 2005: Randall Wayth.
Updates 2016: George Howitt, Stephanie Bernard, Suk Yee Yong, Daniela Carrasco