Chapter 11

Gravitational Collapse: simulating interactions between many bodies

11.1 Background physics

Gravity is an interesting force. Because the force is always attractive, systems of particles that are brought together by gravity have unusual properties. In particular, the system may be dominated by collective effects where the motion of a particle depends on the locations of all other particles in the system. Such systems have a negative specific heat - when energy is lost from the system, its temperature *increases*. The time evolution and properties of such systems are too complex to predict analytically.

Computer models of physical systems, in which the interactions of many particles are followed in time have made significant contributions to research in many areas such as plasma physics, semiconductors, fluid dynamics and astrophysics. This is due to a combination of advances in both the speed of computers and in numerical techniques used to rapidly calculate the inter-particle forces. In this project we consider the gravitational interaction between N massive particles.

For a system containing N particles of mass m_i, Newton's law of gravity gives the gravitational force on the i^{th} particle due to the other $N-1$ particles as

$$\mathbf{F}_i = G \sum_{j \neq i} -m_i \; m_j \; \frac{\mathbf{r_i} - \mathbf{r_j}}{|\mathbf{r_i} - \mathbf{r_j}|^3} = \sum_{j \neq i} \mathbf{F}_{ij}, \tag{11.1}$$

where $\mathbf{r_i}$ is the position vector of the i^{th} and G is Newton's gravitational constant. Once we know the force on a particle, its acceleration is given by Newton's second law:

$$\mathbf{F_i} = m_i \frac{\mathrm{d}\mathbf{v}_i}{\mathrm{d}t}. \tag{11.2}$$

The goal of this project is to write a computer program which can follow the motion of N particles over time, taking into account the gravitational forces between them.

11.2 Numerical solution: the leapfrog approximation

Since the force on a particle in a gravitational system depends only upon the positions of the other particles and not on the velocity (as would be the case in for a particle moving in a magnetic field, for example), we can use a particularly simple scheme to solve or numerically integrate equations (1) and (2), and thus advance the motion of the particles with time.

The method is called the leapfrog finite difference approximation. It is the basis for the integrator used in most astrophysical simulations of gravitational systems. In particular, the leap-frog scheme is used in cosmological simulations of the formation of dark matter structures in the Universe, which can use billions of particles. There are higher order, more accurate integration schemes, such as the Runge-Kutta method for solving differential equations, and these are used in models of the evolution of the solar system. However, these require the storage of more variables, and hence take up more computer memory, which is a drawback when designing simulations that will run with large numbers of particles.

In the leapfrog scheme, we increment the time variable t in steps of dt; we refer to the n^{th} timestep as $t_0 + n\,dt$, where t_0 is the initial time. We calculate the positions and velocities half a timestep out of phase, hence the name *leapfrog*. (Note this scheme is similar to the mid-point version of Euler's method, in which variables are evolved using the value of a gradient estimated half way across the time step.) We use the positions of the particles at timestep n to compute the forces F_{ij}. These forces are then used to update the particle velocities from their values at the timestep labelled $n-1/2$ to the new values at timestep $n+1/2$:

$$\mathbf{v}_i^{n+1/2} = \mathbf{v}_i^{n-1/2} + \mathbf{F}_i\,dt/m_i. \tag{11.3}$$

We then use these new particle velocities to update the particle positions to timestep n

$$\mathbf{x}_i^{n+1} = \mathbf{x}_i^{n} + \mathbf{v}_i^{n+1/2}\,dt. \tag{11.4}$$

For this scheme to work, it is important to note that we need to specify the initial velocities at time $t = -\frac{1}{2}dt$ even though we specify the initial positions at time $t = 0$. While this makes little difference if we start from a random set of particle positions and velocities, it is important in the circular orbit test problem that we consider below.

By applying this scheme, we do not know the particle positions and velocities explicitly at the same time. However, if we wish to compute the energy of a system of particles, we need to know their potential energy and kinetic energy at a given epoch, and so in this case we would need to estimate the velocities and positions at the *same* timestep.

11.3 Some numerical points and tips

Equation (1) needs to be modified when particles come close together. If we did not modify the equation, very short timesteps would have to be used to follow close encounters between particles due to the huge accelerations which the particles would experience. The gravitational force is modified or, technically speaking, *softened* by including a constant ϵ^2 in the denominator of equation (1)

$$\mathbf{F}_i = G\sum_{j\neq i} -m_i\ m_j\ \frac{\mathbf{r_i} - \mathbf{r_j}}{\left[|\mathbf{r_i} - \mathbf{r_j}|^2 + \epsilon^2\right]^{3/2}}. \tag{11.5}$$

This effectively makes the particles behave like hard spheres with radii $\epsilon/2$. Note that the expression for the potential energy of two particles needs also to be adapted if we wish to check for energy conservation.

The gravitational force given by Equation (1) is antisymmetric. The force on particle i due to particle j is the negative of the force on particle j due to particle i: $F_{ij} = -F_{ji}$. Hence, when you have computed F_{ij}, you have F_{ji} for free! This trick can cut the number of operations by a factor of two. Be careful to write the force calculation so that you do not re-calculate the particle separation unnecessarily.

A system of particles interacting through a conservative force should conserve energy over time. In a numerical calculation, energy will not be conserved exactly. In practice, the energy will be conserved to within some tolerance e.g. a variation of $x\%$ of the initial energy, where x depends on the timestep chosen. The size of x should reduce as more steps are used to evolve system over a fixed time interval; if this doesn't happen, then there is probably a bug in your code!

To compute the energy of a system of particles, as commented upon in the previous section, the positions and velocities of the particles need to be calculated at the same time. This can be done by modifying Equation 4 to define a new position vector which has half of the increment.

Finally, when calculating forces, note that a particle cannot exert a gravitational force on itself. When calculating the gravitational potential energy, only count the contribution from each *pair* of particles once.

11.4 Your Work Plan

1. **Design your program** Before writing any Python code, design a code to follow the gravitational interactions and motions of N particles. Write a pseudocode, a list of the instructions or steps to be followed to solve this problem.

   Things to bear in mind: (1) The initial conditions (starting positions and velocities, particles masses) will be different for each problem: the pseudo code should indicate the different ways in which these may be set, but does not need to go into detail. (2) The guts of the program, i.e. the calculation of the force between particles, the update of the positions and velocities, and the calculation of the system energy, should be general for the case of N particles. (3) The output will be different for each problem. Think about which quantities you need to store as arrays. This is determined in part by the algorithms used but also by the output that you need to plot for each task e.g. for the N particle case, we tend to plot the particle positions at a given time, rather than plotting their orbits, which we would do in the case of $N = 2$.

   In order to simplify running your program, it is helpful for you to arrange to read the input paramaters (such as the intitial particle positions and velocities) from a file, rather than having to type them in each time.

2. **Solve the Milestone Problem** Write a Python code to model the motion of a light particle around a heavier particle. Following your pseudocode design, as much as your code as possible should be written for N particles - only the initial conditions and the output should be specific for 2 or 3 particles. In this way, we can test the main part of the code and use it without modification in the next problem.

   Solve the motion of the Earth about the Sun. Assume that the Earth follows a circular orbit. (Mass of the Earth $= 5.9742 \times 10^{24}$kg, mass of the Sun $= 1.9889 \times 10^{30}$kg, Earth-Sun distance: 1.4960×10^{11}m). You can determine the velocity the Earth needs to follow a circular orbit analytically. What is the period of the orbit? For this problem, if your code works and your analytic calculation of the velocity is correct, the two particles should never come together, so the softening can be set to zero. Show that your simulation produces a circular orbit with the expected period.

It is important to note, that the initial velocities need to be specified half a timestep before the initial position. Thus if the initial position of the Earth (at $t = 0$) is on the x-axis, there will still be significant initial velocity components (at $t = -\frac{1}{2}dt$) in both y and x directions.

You should follow the system for 100 orbits to show that the numerical calculation is stable and that the errors do not grow over time. A useful diagnostic is the energy of the system, so you should also plot the energy of the system as a function of time (KE, PE, total energy, and on a separate plot, the percentage change in total energy, compared to the initial energy, as a function of time). You should also plot the radius of the particle as a function of time. Examine your results for different sizes of time step. It should easily be possible to conserve energy to better than a percent.

Once you have a program that works well, you can introduce a third particle representing Jupiter. Jupiter has a mass of 1.8986×10^{27}kg and orbits at a radius of approximately 7.7854×10^{11}m. Initialise your calculation assuming that the Earth and Jupiter are co-linear but located on opposite sides of the sun. The presence of Jupiter will perturb the Earth's orbit slightly and the system's potential and kinetic energy will fluctuate; however, the total energy should be constant to better than a percent.

In your milestone interview, your program should evolve the Sun, Earth, Jupiter system for 100 orbits using a time step of 10 days. Use your program to plot the Sun-Earth separation as a function of time, and to plot the KE, PE and total energy as a function of time. Compute the change in the total energy after 100 orbits as a percentage of the initial energy of the system.

3. **Extend your program** Now that you have a working code, you can apply it to a system that cannot be followed analytically, the gravitational collapse of N particles. The only parts of your code which will need development are the initial conditions and the output.

   The starting point is to put N particles within a sphere of radius R with random x, y and z co-ordinates. Give the particles zero velocity to start with and equal mass. You need to devise an algorithm to assign an x, y and z position at random, where each co-ordinate lies within the range $-R$ to R. Then check to see if a particle is within a sphere of radius R from the origin; retain co-ordinates which satisfy this condition. (**Tip:** set the softening to $0.1R$. Start off with $N = 100$.)

   Plot the $x - y$ positions of the particles after selected numbers of steps (e.g. if you use 1000 steps, make a plot every 200 steps). Plot the KE, PE, total energy of the particles as a function of time. Experiment with the size of the timestep. You should aim to conserve energy to better than 10%.

4. **Research with your program.** Now you can apply your code to model the collapse of a gravitating system. The points below are suggestions for some of the things you could look at. You should not need to consider all of them. It is better to focus on one topic in more depth.

   - A good example of a system of N particles is a globular cluster, a system of stars born in a compact configuration (see Vesperini 2010 for a recent review). Assuming the stars are born with zero velocity, how long does it take the cluster to collapse to an equilibrium state in which the rms (root-mean-square) radius is no longer contracting? What happens if you continue to evolve the system for longer? What happens if the stars have different masses?
   - Follow the collapse of a system with a large initial radius. Explore whether the final system satisfies the virial theorem ($2T + W = 0$, where T is the total kinetic energy of the particles and W is their potential energy) and investigate the thermodynamics of the system when particles lose energy or interact with each other (eg., Hut 1997)

- A topical question is whether globular clusters contain black holes. Explore what happens if you add a few very massive particles to your globular clusters with mass 100 times greater than that of a typical star. Explore what happens if you allow these particles to merge with each other if they come sufficiently close (eg., Gultekin et al., 2004).
- By colliding two systems of particles you can simulate the effects of collisions between globular clusters or galaxies (eg., Barnes et al. 1991). What happens? Does the final system differ from that formed in the collapse of a uniform sphere? An elegant analytic theory is presented in Lynden-Bell 1967.
- By starting your simulation from almost uniform, periodic initial conditions, you can simulate the formation of structure in the Universe (eg., Aarseth et al 1979, Frenk & White 2012). You will need to modify your equations to take the periodicity and expansion of the universe into account. How does the collapse of structure in the Universe differ from the collapse of a spherical distribution?

References

Landau et al. Computational physics.

Binney J., Tremaine S., 1987, Galactic Dynamics

Barger V.D. and Olsson, M.G.: Classical Mechanics, A Modern Perspective (McGraw-Hill)

A nice explanation of the Lagrangian points can be found at:
http://www.esa.int/esaSC/SEMM17XJD1E_index_0.html

Vesperini 2010, Phil. Trans. Royal Soc. A, 368, 829.
http://rsta.royalsocietypublishing.org/content/368/1913/829.full.pdf

Hut 1997, Complexity, 3, 38,
http://arxiv.org/pdf/astro-ph/9704286v1.pdf

Gultekin K., Miller M. C., Hamilton D. P., 2004, 616, 221,
http://adsabs.harvard.edu/abs/2004ApJ...616..221G

Barnes J., Hernquist L., Schweizer F., 1991, Sci. Am., 265, 40,
http://adsabs.harvard.edu/abs/1991SciAm.265...40B

Lynden-Bell D., 1967, MNRAS, 136, 101
http://adsabs.harvard.edu/abs/1967MNRAS.136..101L

Aarseth S. J., Turner E. L., Gott, J. R., III, 1979, ApJ, 228, 664,
http://adsabs.harvard.edu/abs/1979ApJ...228..664A

Frenk C. S., White S. D. M., 2012, Ann. Phys., 524, 507,
http://adsabs.harvard.edu/abs/2012AnP...524..507F