Exercise: Planetary Orbits

Toni Collis and David Henty

October 28, 2015

1 Introduction

The problem here is to examine the evolution of a two body gravitational system. An analytic solution can be derived so that, for this special case, you can compare the numerical solution with a known exact answer. However, for most systems that might be considered, the analytic result is not known. Instead, diagnostics like conserved quantities must be relied on (for example, the value of the total energy should remain a constant).

The aim of this exercise is to understand what the errors are in a simple numerical molecular dynamics (in this case orbits) problem. A template is provided in either C or Fortran. The template sets up the simulation and has a simple first order Euler integration scheme implemented. You have to:

- compare the orbits of the simulations
- examine the effect of changing the integration step size
- implement the leapfrog integration scheme

The code produces plots which can be viewed using the application `xmgrace` - you need to type `module` load xmgrace on ARCHER.

The exercise illustrates a number of key concepts:

- The major error in simulations like these is the truncation error, i.e. how accurately we can model the real-world equations.
- The simple Euler algorithm has a truncation error proportional to the time step.
- The Leapfrog algorithm has a truncation error proportional to the square of the time step.
- This means that the Leapfrog algorithm is vastly superior, i.e. for a given error in the change in the total energy (which should be zero), this can be achieved using a much large time step in the Leapfrog algorithm and hence a much shorter computation time.

2 Planetary orbits

If the potential for a system $U(r) \sim 1/r$, the resulting two-body orbits fall into one of four possible classes of solution.

The resultant orbit is independent of the interaction being considered, be it gravitational, electrostatic or something else. The orbit is defined by its *eccentricity*, e.

Circle $e = 0$
Ellipse $0 < e < 1$
Parabola $e = 1$
Hyperbola $e > 1$

3 Code

The template files are in both C and Fortran. In addition, the setup of the simulation can changed with the input file params.dat. This is a text file which allows you to set parameters as shown:

```
1.0       # Eccentricity of orbit
1.0       # Pericentric distance
1000.0    # Mass of first object
0.1       # Mass of second object (less than first)
75.0      # Initial separation
20.0      # End time for simulation
0.0001    # timestep size
```

Changing the eccentricity changes the type of orbit. The pericentric distance is the distance of closest approach. These can be varied to explore different orbits. The masses of the two objects are chosen so that simulation is of a small, light object orbiting around a static, massive object. The initial separation is the initial condition. In principle any of the parameters can be varied, but for the purposes of this exercise, only the eccentricity, pericentric distance and timestep size should be changed.

4 Exercise

4.1 Changing the orbit

The default params.dat simulates a comet orbiting the sun. Familiarise yourself with the code. What happens to the trajectory when the step size is changed? How would you go about verifying the trajectory if you didn't know the analytic solution? You can view the trajectory using xmgrace trajectory.agr, or the energy change with xmgrace energy.agr

4.2 Implement leapfrog integration

Implement a leapfrog integration scheme in the code. This requires three changes: the changes required are indicated in the code as comments – search for comments containing “Leapfrog”.

Check that the solution is correct. Compare the graphs of the energy with those from Euler integration as a function of the timestep size. Can you see any qualitative difference? Recall, the leapfrog integration scheme integrates the position r and velocity v as a function of time, where $a(t)$ is the acceleration.

$$\begin{aligned} r(t+\Delta t) &= r(t) + v(t + {}^{1}\!/_{2}\Delta t) \times \Delta t \qquad (1) \\ v(t + {}^{1}\!/_{2}\Delta t) &= v(t - {}^{1}\!/_{2}\Delta t) + a(t) \times \Delta t \qquad (2) \end{aligned}$$

Note that this algorithm is not self-starting, you need one step of Euler integration to step either the position or the velocity by a half timestep. Also you need to calculate the velocity before the position due to the half time step.

4.3 Plotting the error

To quantify the difference in solution accuracy using these different integration schemes, we need to consider the energy during the simulation. Calculate the error in the total energy:

$$\Delta E = E_{initial} - E_{final} \qquad (3)$$

and plot ΔE against Δt (the timestep size) on a log-log plot.

Suppose the relationship between the two variables is of the form:

$$\Delta E \propto (\Delta t)^m$$

where m depends on the integration scheme. By taking the logarithm of both sides of the equation we obtain

$$\log \Delta E \propto m \log \Delta t$$

and so the power m becomes the gradient on the log-log plot. Measure the gradient, and thus the power of the error in the step size for both schemes. This behaviour is best seen with closed orbits - see params-closed.dat.

4.4 Summing up

This exercise has looked at Euler and leapfrog integration schemes for a simple two-body problem. The simulations already display complex behaviour, so it is useful to study this simple scheme where we can derive an analytic result to study the error our discretisation introduces. For real calculations with many bodies, more advanced integration schemes can be used. Adaptive step sizes and higher-order integration schemes are beyond the scope of this exercise. However, the lesson here is (hopefully) that you should always study the error in a simulation as much as the final result itself.