THE UNIVERSITY *of* EDINBURGH
School of Physics and Astronomy

Project A: *N*-body simulation

Aims

- Write code to describe N-body systems interacting through Newtonian gravity
- Introduce realistic conditions and simulate the Solar System
- Compare simulation results to known astrophysical data

N-body Systems

In this project, you will generalise your simulation program from Exercise 3 so that you can simulate an arbitrary number of particles. The reference test case should be the two-particle simulation you performed in Exercise 3 – you should check that your modified program can replicate the results for this simple system before moving on to the more complex simulations. Next, you should set up the simulation with Mercury and Venus orbiting the Sun before completing the inner Solar System with the addition of the Earth, Moon and Mars.

The largest cost in a simulation of this type is in computing all the pairwise interactions between particles. Typing all the $N(N-1)$ ordered particle pairings to compute the force and energy is clearly going to get tedious even for just three particles (here there would be 6 interactions). Ponder the cost of this operation for an entire protein or for a galaxy of objects where the number of particles required could be on the order of millions. We will use *arrays* and *loops* to repeat the force calculation for all pairs of particles rather than try to write each interaction individually.

Trajectory Files and VMD

Trajectory files 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 or by plotting all the points simultaneously.

VMD (http://www.ks.uiuc.edu/Research/vmd/) is a standard tool that is used for visualising trajectories in a variety of scientific fields ranging from biology to astrophysics.

The format of the trajectory file for VMD (for a system containing two points) looks like this:

```
2
Point = 1
s1₁ x1₁ y1₁ z1₁
```

```
s2₁ x2₁ y2₁ z2₁
2
Point = 2
s1₂ x1₂ y1₂ z1₂
s2₂ x2₂ y2₂ z2₂
⋮
2
Point = m
s1ₘ x1ₘ y1ₘ z1ₘ
s2ₘ x2ₘ y2ₘ z2ₘ
```

You can see that the file consists of "m" repeating units (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 a title line (in the example above it specifies the point number). Following the header lines are the lines specifying the Cartesian coordinates for the points (one for each point), which consist of: the point label (some text) and the x, y, and z coordinates for that point at this trajectory entry.

You should already have a `toString()` method in your Particle3D class that produces a String in the correct format for a VMD trajectory file.

Task: write a method to write out an entry for a single timestep to a VMD trajectory file.

Visualising simulations in VMD

You should use VMD to visualise the trajectories for the simulations you run below. Do they do what you expect? You can view a trajectory in VMD using something like:

```
vmd myTrajFile.xyz
```

You should experiment with visual representations in VMD (Graphics → Representations menu item) to find different ways of representing the time series. In particular:

- Use the 'Points' drawing method to visualise the trajectories as particles moving in time. Can you speed up and slow down the animation?
- Use the 'Points' drawing method to create a static representation of the entire trajectory. You will need to use the "Trajectory" tab of the representations window to set the trajectory *range* (lower and upper limit of steps to display) and *stride* (increment between displayed steps) to generate a meaningful representation.

Task: Gravitational force and energy between two particles

In Exercise 3 we considered a single particle orbiting a massive point located at the origin. Now you should generalise the force vector and energy calculation routines so they can calculate the force vector arising from the interaction between two Particle3D objects and the energy from the interaction between two Particle3D objects.

Create two **methods** that take two Particle3D objects as their arguments and return the force vector (for the force method) or the energy (from the energy method). Assume for simplicity that the gravitational constant is 1.

Task: Particle Array Methods

You should now add a set of methods that can operate on *arrays* of particles. Using these you can easily extend simulations to an arbitrary number of particles. You should add functionality that:

- Updates the velocity for all particles in an array. Remember the total force vector for a particle is the sum of the forces due to all the other particles. You should formulate an algorithm to compute this sum for a single particle and then consider how you might modify this so it operates on each particle in turn.
- Updates the position for all the particles in an array.
- Calculates the total energy due to atom pair interactions and particle kinetic energies. Take care here to avoid double-counting interactions.

Task: N-Body Simulation Code

Write a program to simulate many-body systems. It should:

- Read in the number of particles in the system,
- Set up an array of Particle3D objects to hold the particles,
- Read the particles into this array,
- Simulate the evolution of the system using the velocity Verlet time integration algorithm,
- Estimate the error in the simulation by monitoring the fluctuations of the energies,
- Write a trajectory file for the simulation that can be visualised using VMD.

As we may now also have many particles in a simulation it makes sense to read in the simulation parameters (number of steps, time step, constants) from one file and the details of the particles (number of particles, label, mass, starting position, starting velocity) from another file. This means you program should take three command line arguments, *i.e.*:

```
java ParticleManyBody particle.input param.input traj.xyz
```

where `particle.input` is the file containing the particle details; `param.input` is the file containing the simulation parameters; and `traj.xyz` is the name of the output file containing the trajectory (this must have a `.xyz` extension to work correctly with VMD, see above).

Check that your code works for the two-body simulation case that was used in Exercise 3 and produces the same results as the code you wrote in Exercise 3.

This completed program now looks similar (although much simpler) than many of the large software packages that exist for performing this type of simulation.

Task: 3-body planetary system

Now you have a working code we will start to simulate more complex systems and add functionality to your program for correctness checking.

Set up a three-body system that includes the Sun, Mercury and Venus. You should use realistic masses and orbital radii. Adjust the gravitational constant in the force and energy calculations to

correspond to your choice of units. Initial velocities (for a circular orbit) can be computed by using the formula for the tangential velocity of circular motion:

$$v_c = \sqrt{\frac{\mu}{r}}$$

$$\mu = (m_1 + m_2)$$

Here r is the separation of the two bodies, m_1 is the mass of the first body and m_2 is the mass of the second body. Note that the total linear momentum of your simulation must vanish, otherwise the centre of mass will drift. If your bodies have initial velocities $\boldsymbol{v}_i$, then the simulation has a total momentum of

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

Adjust the initial velocity of each body in the system by the negative of the centre-of-mass velocity, -$\boldsymbol{v}_{com}$, given as

$$\boldsymbol{v}_{com} = \frac{1}{\sum_i m_i} \boldsymbol{P}$$

Hence, the Sun will not remain at the origin and also has an initial velocity. **Your written report should outline how you obtained the values for the masses, initial positions and initial velocities for the planets and for the Sun, and your choice of units.**

Use VMD to visualise the simulations and make sure they are working correctly.

Add functionality to your code to count how many times each of the planets orbits the Sun during the simulation. You will first need to formulate an algorithm and then convert this to actual Java code. There are various ways to do this, and it is preferred to be able to track partial orbit completions instead of only complete orbits. Do the two planets' years have the correct ratio? **Your written report should contain a description of the algorithm you have formulated for computing the number of complete orbits and a discussion of the results.**

Update the initial positions and velocities of the planets so that they now move in the correct elliptical orbits. Does this change ratio of year lengths compared to the circular orbits? **Discuss any changes in your written report.** Add functionality to your code to compute the aphelion and perihelion for each planet in the simulation. Use these values to check that your model is functioning correctly. **You should include these results in your written report. The input files for the 3-body simulation with elliptical orbits should be included in your submitted files.**

Task: Solar system

Add correct initial parameters (masses, positions, velocities) for the remaining planets in the Solar System, Earth's Moon, Pluto and Halley's Comet. You may want to use NASA/JPL's HORIZONS system to obtain those values at a given moment in time: http://ssd.jpl.nasa.gov/horizons.cgi.

- Compute the length of your timestep in Earth days. Do you get accurate year lengths for Earth and the other planets in your model?
- Show that the year ratios for the planets are correct.
- Show that the Moon orbits the Earth the correct number of times in a year.
- Verify Kepler's third law for all planets from your acquired data on orbit periods and eccentricities.
- Estimate the largest timestep you can use to accurately simulate the solar system.
- Does the comet's orbit show any perturbations?
- Is the orbital period of the comet in your model consistent with that of the real comet?

The input files and trajectory for this simulation should be included with your report submission. Your write-up should include the length of the years for each of the planets and, if they differ from the real values, a discussion of why this could be. You should also include the maximum timestep you can use for your simulation and discuss what the limiting factors are that influence this value.

Source Code Submission

Package the subdirectory that contains `Vector3D.java, Particle3D.java`, your simulation program, input files, and trajectory files, as requested in the tasks above. For instance, if your subdirectory is called "project", you can use the `tar` and `gzip` commands by running:

```
$ tar cvf project.tar project
$ gzip project.tar
```

In the documentation of your simulation program, describe how to run it and what the format and units of the input and output files are.

We will not accept submissions that are larger than 50MB total. Hence, make informed choices on how often to print planetary positions to file (is every single time step necessary?), how many digits are relevant (use formatted outputs), and restrict the overall simulation time to reasonable values (what is the longest period in the system, and how many cycles do you need?). Submit the compressed package (e.g., `project.tar.gz`) through the course LEARN page, by **5pm on 9 March 2016** or **5pm on 16 March 2016**, depending on which group you are in. Only one submission per group is necessary.

Task: Test and evaluate another code

Shortly after the code submission deadline, you will be sent the source code and input files of another group. Study the code – is all required functionality implemented? Does the implementation differ from your own? Then compile and run the code. Can you adopt your own input data to be used with this code? Do the results agree with your own implementation? Which code runs faster? **In your written report, include a section (about half a page) on those tests and the comparisons with your own implementation.** ***Note that the other code might be buggy, incomplete, or inaccurate. You are not asked to fix the code, and will not be penalised if the other code does not work correctly. You should, however, be able to establish and comment on whether it works satisfactorily.***

Project Report Submission

Prepare your report according to the following guidelines:

- Indicate at the top which project you have attempted
- Your report should contain the following sections:
    - Tasks and Objectives of the project.
    - Descriptions of the program, including the specific algorithms you implemented. Comment on any changes to the code structure and algorithms from the outline in the Design Document, and why those changes were made.
    - Results and discussion – includes the results from your simulations (as requested above) and your discussion of their meaning.
    - Comparison – discuss the test results and comparisons with the other code you were given.
    - Conclusions and post-mortem – Discuss any problems associated with the project and the team, and reflect on whether (and why) you would do anything differently if you were to complete the task again.
- Include screenshot(s) of your simulations and tables and graphs of relevant quantities wherever they are helpful to underpin your discussion.
- Maximum report length is 10 pages, font size not smaller than 11pt, including all diagrams and tables.

The report should read similar to a lab report, and be less technical than the Design Document. Submit your report through the Turnitin Link on the course Learn page, by **5pm on 23 March 2016** or **5pm on 30 March 2016**, depending on which group you are in. This is an individual task to be completed by every student.

Marking Scheme – Source Code (40 Marks)

- Code compiles and works with inputs provided [4]
- Input and output formats sensible and appropriate [4]
- Code has all required functionality: array methods, file I/O, velocity Verlet integrator, total energy, centre-of-mass correction, orbit period counters [20]
- Code layout, naming conventions, and comments are clear and logical [12]

Marking Scheme – Project Report (80 Marks)

- Descriptions of tasks, objectives, implementation, and any changes [15]
- Results and discussion: choice of initial parameters and units; 3-body simulation; full solar system simulation; comparisons to experimental data; verification of Kepler's third law; estimate of maximum time step; Halley's comet. [45]
- Software testing and comparisons [5]
- Conclusions and post-mortem [5]
- Report layout, language, graphics quality and usefulness [10]