Format for delivery of report and programs

The format of the project is that of a printed file or hand-written report. The programs should also be included with the report. Write **only your candidate number** on the first page of the report and state clearly that this is your report for project 5 of FYS3150/FYS4150, fall 2013. There will be a box marked 'FYS3150/FYS4150' at the reception of the Department of Physics (room FV128).

Project 5, N-body simulation of an open galactic cluster, deadline Monday December 9, 12pm (noon)

The goal in this project is to develop a code that can perform simulations of an open cluster using Newtonian gravity. First, however we will compare the stability of two different methods. This is because when we are looking at a system with a large number of particles, we are more interested in the statistical properties of the system than in the individual motion of each of the particles. This means that the stability of the solution method is more important than its short term accuracy. This project is inspired by the preprint by Joyce *et al.*, see Ref. [1] below. When solving this project, we recommend downloading this article. It is a good companion to understand the physics discussed here.

In the first part of this project we will explore the stability of two well-tested numerical methods for solving differential equations. Here you can use the algorithms and program you developed for the solar system in project 3. The algorithms to test and implement are the fourth-order Runge-Kutta method and the so-called Leap-Frog method. The latter is derived here.

Consider a second-order differential equation like Newton's second law, whose one-dimensional version reads

$$m\frac{d^2x}{dt^2} = F(x,t),$$

which we rewrite in terms of two coupled differential equations

$$\frac{dx}{dt} = v(x,t) \qquad \text{and} \qquad \frac{dv}{dt} = F(x,t)/m = a(x,t).$$

If we now perform a Taylor expansion

$$x(t+h) = x(t) + hx^{(1)}(t) + \frac{h^2}{2}x^{(2)}(t) + O(h^3).$$

In our case the second derivative is know via Newton's second law, namely $x^{(2)}(t) = a(x,t)$. If we add to the above equation the corresponding Taylor expansion for $x(t-h)$, we obtain, using the discretized expressions

$$x(t_i \pm h) = x_{i\pm 1} \qquad \text{and} \qquad x_i = x(t_i),$$

$$x_{i+1} = 2x_i - x_{i-1} + h^2x_i^{(2)} + O(h^4).$$

We note that the truncation error goes like $O(h^4)$ since all the odd terms cancel when we add the two Taylor expansions. We see also that the velocity is not directly included in the equation since the function $x^{(2)} = a(x,t)$ is supposed to be known. If we need the velocity however, we can compute it using the well-known formula

$$x_i^{(1)} = \frac{x_{i+1} - x_{i-1}}{2h} + O(h^2).$$

We note that the velocity has a truncation error which goes like $O(h^2)$. In for example so-called Molecular dynamics calculations, since the acceleration is normally known via Newton's second law, there is seldomly a need for computing the velocity. The above sets of equations for the position $x(t)$ and the velocity defines the Verlet formula. The Leapfrog algorithm is also easily derived.

We can rewrite the above Taylor expansion for $x(t+h)$ as (skipping higher terms in h)

$$x(t+h) = x(t) + h\left(x^{(1)}(t) + \frac{h}{2}x^{(2)}(t)\right).$$

Noting that

$$x^{(1)}(t+h/2) = \left(x^{(1)}(t) + \frac{h}{2}x^{(2)}(t)\right),$$

we obtain

$$x(t+h) = x(t) + hx^{(1)}(t+h/2),$$

which needs to be combined with

$$x^{(1)}(t+h/2) = x^{(1)}(t-h/2) + hx^{(2)}(t).$$

Again, there is a lower truncation error in h for the velocity. Furthermore, the positions and the velocities are evaluated at different time steps. If one needs $x^{(1)}(t)$, this can be computed using

$$x^{(1)}(t) = \left(x^{(1)}(t \mp h/2) \pm \frac{h}{2}x^{(2)}(t)\right).$$

The initial conditions can be handled in similar ways and the inaccuracy which arises between $x^{(1)}(0)$ and $x^{(1)}(h/2)$ is normally ignored. Summarizing, the popular Leapfrog algorithm implies the evaluation of position and velocity at different time steps. The final algorithm is given by the following steps

$$x^{(1)}(t+h/2) = x^{(1)}(t) + \frac{h}{2}x^{(2)}(t),$$

which is used in

$$x(t+h) = x(t) + hx^{(1)}(t+h/2),$$

and finally

$$x^{(1)}(t+h) = x^{(1)}(t+h/2) + \frac{h}{2}x^{(2)}(t+h),$$

The last three steps constitute the Leap-Frog method. Convince yourself that the above steps are correct. Can you find the approximation error, that is the factor n in $O(h^n)$?

a) Implement the Newtonian two-body (you can choose masses and dimensionalities as you wish) problem in three dimensions using the fourth order Runge-Kutta method and the so-called Leap-Frog method discussed in the lecture notes and restated above here.

   You can build on the code you developed for project 3. Compare the stability of the two different methods. How do they work for large time steps? How do they work for very long times? Compare also the time used to advance one timestep for the two different methods. Comment your results. Which algorithm would you use for simulating systems that require long times?

We will now try to build a simple model of an open cluster, see Ref. [2]. An open cluster is a group of up to a few thousand gravitationally bound stars created from the collapse of a molecular cloud. This collapse leads to a flurry of star formation. Open clusters are usually found in the arms of spiral galaxies, or in irregular galaxies. Since stars in an open cluster have roughly the same age, and are made from the same material, they are interesting in the study of stellar evolution, since many of the variable parameters we have when comparing two stars are kept constant.

Once open clusters are formed they gradually dissipate as members get ejected from the cluster due to random collisions, this means that open clusters generally last only a few hundred million years. In figure 1, we see the Hertzsprung-Russell diagrams for two open clusters.

We will look at a simple model for how an open cluster is made from the gravitational collapse and interaction among a large number of stars. We want to study this collapse, and the statistical properties of the collapsed system.

One particle in our model represents one or a few stars, and we will work with a few hundred particles. We will simulate what is called a "cold collapse", this means that we start the particles with little or no initial velocity.

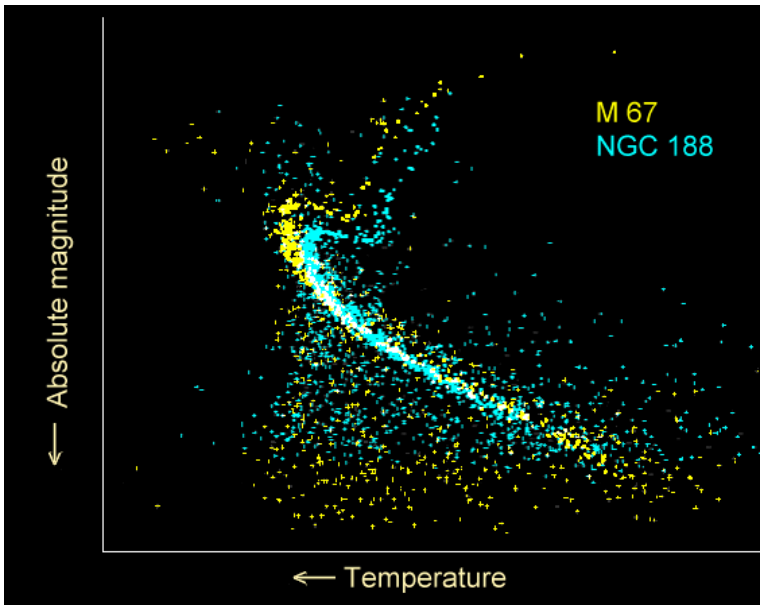

Figure 1: Hertzsprung-Russell diagrams for two open clusters, M67 and NGC 188. We see that most of the stars are on the main sequence. In the older cluster, NGC 188, we see that the heaviest stars are just now leaving the main sequence, while the younger cluster, M67, is following closely after.

b) Extend your code to an arbitrary number of particles, N, starting with a uniform (random) distribution within a sphere of a given radius R_0. Start the particles at rest, with masses randomly distributed by a Gaussian distribution around ten solar masses with a standard deviation of one solar mass. Use solar masses and light years as units of mass and length and make your equations dimensionless. The function *GaussPDF* included with this project can be used to generate random numbers which follow a Gaussian (or normal) distribution.

   How large timesteps are required given $R_0 = 20ly$ (light years), and a $N = 100$? Do we have any units of time that fits this timescale? In the limit where $N \to \infty$, keeping ρ_0 constant, we get a continuous fluid. In this case the system collapses into a singularity at a finite time $\tau_{crunch} = \sqrt{\frac{3\pi}{32G\rho_0}}$. (For the especially interested (Not required!): Can you derive this result? Hint: recall the Friedman equations [3].

   Why do we not observe this singularity in our model? Use τ_{crunch} as the unit of time, and find G in these units (G will become a function of the number of particles N, and the average mass of the particles, μ).

   You should run these calculations with both the fourth-order Runge-Kutta algorithm and the Leap-Frog method. Which method would you prefer? Give a critical discussion.

   For the remaining exercises, you should use only one of the above methods.

c) Run the system for a few τ_{crunch}. Save the positions of the particles at different times to file, and make an animation of the time evolution using your preferred software? Does the system reach an equilibrium? How long time does this take ?

d) Make a function that calculates the kinetic and potential energy of each particle. Is the energy conserved? Some of the particles are ejected from the system, how can we identify these particles from the energies we have calculated? How much of the energy of the system is taken away by particle ejection? How does this change with different values of N? Are there still particles being ejected after the system reaches equilibrium?

e) We will now introduce a smoothing function to take care of the numerical instability that arises when two of the particles come very close. There is a lot of ways to insert such a smoothing, but we will just look at a very simple one. We will modify the Newtonian force law to make it finite at short ranges

$$F_{mod} = -\frac{GM_1M_2}{r^2+\epsilon^2}.$$

The parameter ϵ is a small real constant that we can set to $\epsilon = 0.15ly$. We can justify this correction to the pure Newtonian force by noting that our particles do not represent actual point particles but rather mass distributions of some finite extent. Does the addition of this correction change any of the results from part d) ?

f) Now we will look at the particles that are bound (not ejected). What is the distribution of potential and kinetic energy?

The virial theorem says that for a bound gravitational system in equilibrium we have

$$2\langle K\rangle = -\langle V\rangle,$$

where $\langle K\rangle$ is the average kinetic energy of the particles and $\langle V\rangle$ is the average potential energy.

Are your results consistent with the virial theorem?

g) Try to plot the radial density of the particles (the particle density as a function of radius) in the equilibrium state. How would you extract such an information from your calculations? (Hint: make a histogram for the radial particle density) What is the average distance? What is the standard deviation? Plot the radial distribution of particles.

Run the code for different number of initial particles, keeping the total mass constant. What is the average distance as a function of N?

The radial distribution of particles in this kind of cold collapse can often be fit very well with the simple expression

$$n(r) = \frac{n_0}{\left(1+\left(\frac{r}{r_0}\right)^4\right)}.$$

Try to fit your data to this curve, what is the value n_0 and r_0? Can you find how these values depend on N?

How many particles can you simulate?

Compare your results with those found in Ref. [1].

h) This part is optional but gives you an additional 30% on the final score! Your task here is to parallelize using either MPI or OpenMP the differential solver you have developed for this project. The parallelization of ordinary differential equations will be discussed during the remaining lectures. What kind of speed-up do you get, that is, how does the speed-up of your code scale with the number of processes? Make sure that your previous results reproduced. The parallelization of ordinary differential equations will be discussed during week 47.

References

[1] M. Joyce, B. Marcos, and F. Sylos Labini, Cold uniform spherical collapse revisited, arXiv1011.0614 (2011), http://arxiv.org/abs/1011.0614.

[2] P. J. E. Peebles, *The Large-Scale Structure of the Universe*, Princeton University Press, 1980. See also C. Payne-Gaposchkin, *Stars and clusters*, (Cambridge, Harvard University Press, 1979).

[3] A. Friedman, *On the Curvature of Space*, General Relativity and Gravitation **31**, 1991 (1999).