FYS4150 Project 5: N-body simulation of an open cluster

Marie Foss (# 56), Maria Hammerstrøm (# 59)

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Github: *https://github.com/mariahammerstrom/Project5*

1 Introduction

An open cluster is a group of up to a few thousand stars that are gravitationally bound to each other. These stars are created from the same giant molecular cloud, making the age and composition of the stars similar, which in turn makes the properties of the stars, such as distance, age, metallicity and extinction, more easy to determine.

Open clusters are usually found in the arms of spiral galaxies or in irregular galaxies. Once an open cluster has formed, it will gradually dissipate as its members gets ejected from the cluster due to random collisions, lasting only a few hundred million years.

In this project we make a simple model for how an open cluster is formed from a gravitational collapse and model the interactions among a large number of stars. We have simulated a "cold collapse", meaning the stars will start at rest. The stars have masses randomly distributed by a Gaussian distribution around $10M_\odot$ with a standard deviation of $1M_\odot$, and uniform randomly distributed positions within a sphere of a given radius R_0.

We first study the Newtonian two-body problem in three dimensions, then extend this to an N-body problem in three dimensions. The time evolution of the system is described through **Newton's law**, given by:

$$\frac{\mathrm{d}^2x}{\mathrm{d}t^2} = \frac{F_{G,x}}{M} \tag{1}$$

where $F_{G,x}$ is the **gravitational force** in the x direction, given by

$$F_{G,x} = -\frac{GM_1M_2}{r^3}x, \tag{2}$$

where M_1 and M_2 are the masses of the two objects and r is the distance between them. In three dimensions, $r = \sqrt{x^2 + y^2 + z^2}$. We have similar expressions for the y and z direction.

Eq. (1) is a second-order differential equation, which we can rewrite as a set of coupled first-order differential equations:

$$\frac{\mathrm{d}x}{\mathrm{d}t} = v_x \quad \text{and} \quad \frac{\mathrm{d}v_x}{\mathrm{d}t} = F_{G,x}. \tag{3}$$

These equations can be used to determine the position and velocity of an object.

When solving this problem we will convert the units of mass and length to dimensionless variables using solar masses $M_\odot$ and light years, which is described in Sec. 2.4.

In the case where $N \to \infty$ and keeping the mean density ρ_0 constant, the system will collapse into a singularity at a finite **collapse time**

$$\tau_{\text{crunch}} = \sqrt{\frac{3\pi}{32G\rho_0}}, \tag{4}$$

derived in Appendix A.1. We can write G in units of τ_{crunch} so that G becomes a function of the number of particles N and the average mass of the particles μ, which gives

$$G = ?, \tag{5}$$

which is derived in the Appendix A.2.

After running calculations for the gravitational force as expressed in Eq. (2), we introduce a smoothing function to take care of any numerical instability that may arise when two stars come very close. We modify the gravitational force as

$$F_{G,\text{mod}} = -\frac{GM_1M_2}{r^2 + \epsilon^2}, \tag{6}$$

where ϵ is a small real constant. We will try out different values for ϵ.

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

$$2\langle K\rangle = -\langle V\rangle, \tag{7}$$

where $\langle K\rangle$ is the time-averaged kinetic energy of the system and $\langle V\rangle$ is the time-averaged potential energy of the system. The virial theorem deploys the ergodic hypothesis, which states that we can take an ensemble average instead of a time average. By calculating the kinetic and potential energy of our system, we can check if our results are consistent with Eq. (7).

Finally we wish to study the **radial density** of the particles in the equilibrium state. This can be calculated from our computations and be fitted with the simple expression

$$n(r) = \frac{n_0}{1 + (\frac{r}{r_0})^4} \tag{8}$$

where n_0 and r_0 are scaling constants.

2 Methods

The numerical algorithms we wish to implement and compare are the fourth-order Runge-Kutta method and the Velocity-Verlet method. Both methods assume a known initial value for the position (x_i, y_i, z_i) and the velocity $(v_{i,x}, v_{i,y}, v_{i,z})$.

2.1 4th order Runge-Kutta method

Using the 4th order Runge-Kutta method (RK4), we can estimate the next value x_{i+1} (and similarly y_{i+1} and z_{i+1}) through the formula

$$x_{i+1} \approx x_i + \frac{h}{6} + 2v_{i+1/2,x} + 2v_{i+1/2,x} + v_{i+1,x} \tag{9}$$

where $v_{i+1/2,x}$ is the velocity in the x direction at time $t_{i+1/2}$ and position $x_{i+1/2}$ after taking a step $h/2$, which marks the midpoint between $v_{i,x}$ and $v_{i+1,x}$. The midpoint is evaluated two times to get a better estimate of the function value at this point, which in turn will give a better estimate for the final point $v_{i+1,x}$. h denotes the time step, defined as $h = (t_f - t_0)/N$ where t_f is the end-time of our simulation, t_0 is the start-time (usually chosen as zero), and N is the number of integration points (not to be confused with the number of particles in the simulation!).

The algorithm goes as follows:

- Compute $K_1 = h\, v_x(t_i, x_i)$, which gives the slope at t_i.
- Compute the slope at the midpoint using Euler's method, which gives $K_2 = h\, v_x(t_i + h/2, x_i + K_1/2)$.
- The estimate of the slope at the midpoint is then further improved by $K_3 = h\, v_x(t_i + h/2, x_i + K_2/2)$.
- Finally compute the end-point through: $x_{i+1} = x_i + \frac{1}{6}(K_1 + 2K_2 + 2K_3 + K4)$.

These equations must be deployed for all three dimensions x, y and z.

Using this method gives a global truncating error which goes like $O(h^4)$.

2.2 Velocity-Verlet method

This method is derived by performing a Taylor expansion of $x(t + h)$ and $x(t - h)$, and add the two equations. This results in the formula

$$x_{i+1} = 2x_i - x_{i-1} + a_x(t_i, x_i)h^2 + O(h^4) \quad i = 1, 2, \ldots \tag{10}$$

where a_x is the acceleration, which is the second derivative of x, or the first derivative of v_x. Velocity is not included in the equation, but can be computed using the well-known formula

$$v_{i,x} = \frac{x_{i+1} - x_{i-1}}{2h} + O(h^2) \tag{11}$$

where x_{i+1} has been estimated by Eq. (10).

We see from both Eq. (10) and Eq. (11) that this algorithm is not self-starting as it depends on x_{i-1}. The first time we calculate Eq. (10), meaning x_2, we need to know both x_0 and x_1. The known initial position gives us x_0 and we have to use a different algorithm to make an estimate for x_1. The simplest solution is to use Euler's method, which approximates x_1 by:

$$x_1 = x_0 + v_{0,x}h^2 \tag{12}$$

The algorithm can be summed up as:

- Given the initial position x_0 and velocity v_0, calculate the next position x_1 using Eq. (12).
- Calculating all subsequent positions x_{i+1} using Eq. (10).
- Calculate all velocities $v_{i,x}$ using Eq. (11).

The same algorithm follows for all dimensions (x, y, z).

Using this method gives a global truncating error which goes like $O(h^4)$ for the positions (x_i, y_i, z_i), and $O(h^2)$ for the velocities $(v_{i,x}, v_{i,y}, v_{i,z})$, thus the velocities are not approximated to the same accuracy as the positions are.

2.3 Random distributions

The particles in the N-body case will start with positions (x, y, z) drawn from a **random uniform distribution** within a sphere of radius R_0. Achieving this numerically is not entirely trivial if we want to avoid the particle density being much higher in the centre. First we consider the spherical representation of the coordinates

$$\begin{aligned} x &= r\sin\theta\cos\phi \\ y &= r\sin\theta\sin\phi \\ z &= r\cos\theta \end{aligned} \tag{13}$$

where $\theta \in [0, \pi]$, $\phi \in [0, 2\pi]$ and $r \in [0, R_0]$. The trick is to rewrite the volume element as

$$r^2 \sin\theta\, dr d\theta d\phi = A\, du dv dw,$$

where $u, v, w \in [0, 1]$ and A is a constant. Following the approach of H.T. Ihle (2015), we find

$$\begin{aligned}\theta &= \arccos(1-2v)\\ \phi &= 2\pi w\\ r &= R_0 u^{1/3}\end{aligned} \tag{14}$$

where u, v, w are computed using the built-in C++ function `uniform_real_distribution`, which will give randomly distributed numbers in the interval $[0,1]$. Then, plugging the computed values of Eq. (14) into Eq. (13) gives the desired result.

The particles will be assigned a randomly distributed mass from a **Gaussian distribution** for a given mean value $10M_\odot$ and with a standard deviation $M_\odot$. This is simply done using the built-in C++ function `normal_distribution`.

2.4 Nondimensionalization

We use solar masses $M_\odot$ and light years as units of mass and length to make our equations dimensionless.

...

3 Results

3.1 Testing the numerical methods

We test our implementation of the numerical methods described in Sec. 2 by running the code for a case where the analytical solution is known. In the case of a second-order differential equation like the one we have in Eq. (1), we can test our algorithm by considering a box on a spring. In this case the equation to be solved is:

$$F = -kx \quad \Rightarrow \quad \frac{\mathrm{d}^2x}{\mathrm{d}t^2} = -\frac{k}{m}x = -\omega_0^2 x, \tag{15}$$

which has the known analytical solution $x(t) = A\cos(\omega_0 t + \nu)$, where ν is the phase constant. The coupled first-order differential equations can in this case be written as

$$\frac{\mathrm{d}x}{\mathrm{d}t} = v_x \quad \text{and} \quad \frac{\mathrm{d}v_x}{\mathrm{d}t} = -\frac{k}{m}x. \tag{16}$$

where we can set $k = m = 1$ for simplicity when checking the validity of our numerical methods. Then the analytical solution is simply $x(t) = \cos(t)$, with $v(t) = -\sin(t)$.

A visualization of our test is shown in Fig. 1a-b, here for a time step of $\Delta t = 0.2$. We see that the numerical algorithms reproduce the result of the analytical expression very well. Here we have chosen a time step large enough for it to be possible to distinguish the different methods. At a time step of a factor of 10 smaller, $\Delta t = 0.02$, it is impossible to tell the methods apart.

Another test we can make is to check for **energy conservation**. With initial conditions x_0 and v_0, we can calculate

$$\begin{aligned}E(t=0) = E_0 &= \frac{1}{2}mv_0^2 + \frac{1}{2}kx_0^2,\\ E(t=t_i) = E_i &= \frac{1}{2}mv_i^2 + \frac{1}{2}kx_i^2,\end{aligned} \tag{17}$$

and check that $E_0 = E_i$ for every period T. It is however easier to just plot E_i as a function of time for the different methods and inspect the result. If the energy is conserved, the plot should be a straight line, as the energy should not change over time. As shown in Fig. 2 this is indeed the case in the analytical case. If we look closely at the y-axis we see that this really is the case for all the methods, with the numerical methods deviating from the expected result in the fifth decimal place.

Now that we have verified our numerical methods, we can go on to solve the problem at hand.

3.2 Stability of methods

The stability of the two methods described in Sec. 2 can be compared by running the two algorithms over a very long time period.

...

We also check how the two methods work for large time steps.

...

What is also interesting is to compare the time used to advance one time step for the two different methods, to see which one is the most efficient.

...

Based on our study of the stability and time usage of the two methods, we can say that ... COMMENT.

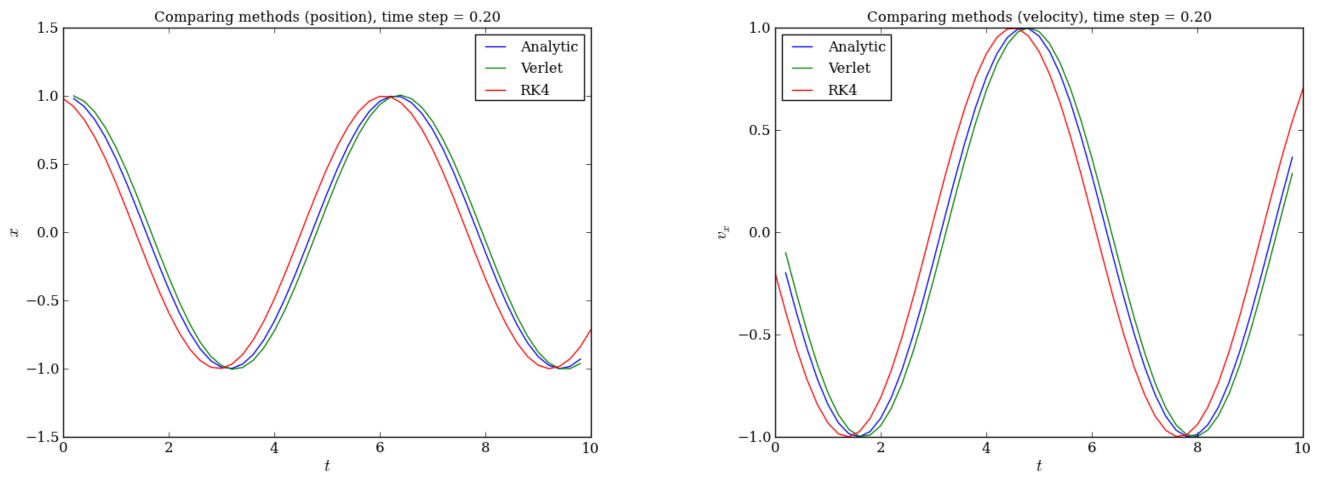
4 Conclusions

...

5 List of codes

The codes developed and used for this project are:

`main.cpp` – main program (C++).
`plotting_analytical.py` – plotting program to compare the one-dimensional 2-body problem with an analytical solution (Python).



(a) Position x (b) Velocity v_x

Figure 1: Comparison of our numerical algorithms, Velocity-Verlet (green) and 4th-order Runge-Kutta (red), with the analytical solution (blue) in the case of the box on a spring. Here for a time step of $\Delta t = 0.2$.

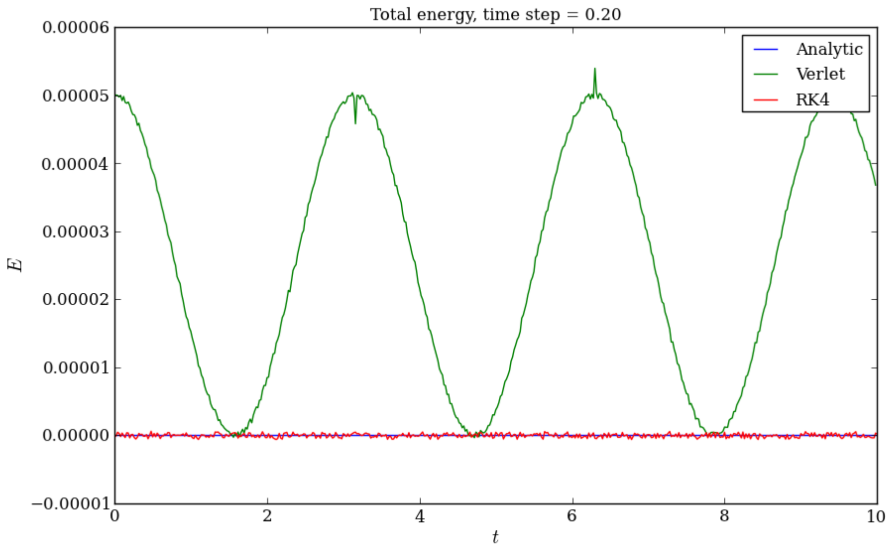


Figure 2: The total energy E as a function of time t of our numerical algorithms, Velocity-Verlet (green) and 4th-order Runge-Kutta (red), with the analytical solution (blue) in the case of the box on a spring. Here for a time step of $\Delta t = 0.2$.

A Appendix

A.1 Derivation of t_{crunch}

...

A.2 Rewriting of t_{crunch}

We want to write G in units of τ_{crunch} so that G becomes a function of the number of particles N and the average mass of the particles μ. DERIVATION.