

FYS4150 Project 5:

N -body simulation of an open cluster

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

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur 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:

$$\frac{d^2x}{dt^2} = \frac{F_{G,x}}{M_1} \quad (1)$$

where M_1 is the mass of the object we are considering and $F_{G,x}$ is the **gravitational force** in the x direction, given by

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

where M_1 and M_2 are the masses of the two objects exerting a gravitational force on each other 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:

$$\frac{dx}{dt} = v_x \quad \text{and} \quad \frac{dv_x}{dt} = \frac{F_{G,x}}{M_1}. \quad (3)$$

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

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 $(v_{i,x}, v_{i,y}, v_{i,z})$. Both methods are self-starting.

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

$$x_{i+1} \approx x_i + \frac{h}{6}(v_{i,x} + 2v_{i+1/2,x} + 2v_{i+1/2,x} + v_{i+1,x}), \quad (4)$$

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:

$$x_{i+1} = x_i + \frac{1}{6}(K_1 + 2K_2 + 2K_3 + K_4).$$

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

This method is derived by performing a Taylor expansion of $x(t + h)$, which in discretized notation can be written as

$$x_{i+1} = x_i + v_i \Delta t + \frac{\Delta t^2}{2} a_i, \quad (5)$$

where a_i is the acceleration and Δt is the time step. For this method the next step in the velocity is calculated by

$$v_{i+1} = v_i + \frac{1}{2} \Delta t (a_i + a_{i+1}), \quad (6)$$

where a_{i+1} is the acceleration calculated using x_{i+1} . Thus the algorithm can be summed up as:

- Given the initial position x_0 and velocity v_0 at time $t_0 = 0$, calculate the next position x_1 using Eq. (5).
- Calculate a_1 from the interacting forces using the new position x_1 .
- Calculate v_1 as described in Eq. (6) given a_1 .
- Increase the time step, so that $t = t_0 + \Delta t$.
- Keep calculating x_{i+1} and v_{i+1} , and increasing the time step, until the pre-determined final time is reached.

Using this method gives a global truncating error which goes like $O(h^3)$ 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

$$\begin{aligned} x &= r \sin\theta \cos\phi, \\ y &= r \sin\theta \sin\phi, \\ z &= r \cos\theta, \end{aligned} \quad (7)$$

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

$$\begin{aligned} \theta &= \arccos(1 - 2v), \\ \phi &= 2\pi w, \\ r &= R_0 u^{1/3}, \end{aligned} \quad (8)$$

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. (8) into Eq. (7) 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. We also use as the **collapse time** of the system τ_{crunch} as the unit of time,

$$\tau_{\text{crunch}} = \sqrt{\frac{3\pi}{32G\rho_0}}, \quad (9)$$

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 . First we rewrite Eq. (9) with respect to G , which gives

$$G = \frac{3\pi}{32\rho_0 \tau_{\text{crunch}}^2}. \quad (10)$$

where we can replace ρ_0 by using that

$$n = \frac{\rho N}{\mu} \Rightarrow \rho = n\mu N$$

where N is the total number of particles and μ is the average mass of the system,

$$\mu = \frac{M_{\text{tot}}}{N} = \frac{\rho V}{N} = \frac{\frac{4}{3}\pi R_0^3 \rho}{N}$$

Combining these relations we can finally rewrite Eq. (10) as

$$G = \frac{4\pi^2 R_0^3}{32\mu N \tau_{\text{crunch}}^2}. \quad (11)$$

giving G in units of $[\text{ly} M_\odot^{-1} \tau_{\text{crunch}}^{-2}]$.

Now let us look at the differential equation itself, described in Eq. (1), and see how we make this dimensionless. We introduce dimensionless quantities

$$\hat{x} = \frac{x}{l_c}, \quad \hat{t} = \frac{t}{\tau_{\text{crunch}}}, \quad \hat{M}_2 = \frac{M_i}{M_\odot}, \quad \hat{r} = \frac{r}{l_c}$$

where $l_c = 1$ ly is the characteristic length scale of the system. Then we can write Eq. (1) as

$$\frac{l_c}{\tau_{\text{crunch}}^2} \frac{d^2 \hat{x}}{d \hat{t}^2} = - \frac{G \hat{M}_2 M_\odot \hat{x} l_c}{\hat{r}^3},$$

which can be simplified to

$$\frac{d^2 \hat{x}}{d \hat{t}^2} = -G \frac{\tau_{\text{crunch}}^2 M_\odot}{l_c^3} \frac{\hat{M}_2 \hat{x}}{\hat{r}^3}$$

As we will use τ_{crunch} as the time unit, M_\odot as the mass unit and l_c as the length unit, these can be set to 1. Also, if we plug in values for R_0 and μ in Eq. (11) that are dimensionless, G will be dimensionless, denoted with \hat{G} . Then the differential equation in Eq. (1) can finally be written as the dimensionless equation

$$\frac{d^2 \hat{x}}{d \hat{t}^2} = -\hat{G} \frac{\hat{M}_2 \hat{x}}{\hat{r}^3}. \quad (12)$$

Again we rewrite this second-order differential equation as two coupled first-order differential equations:

!!
MORE!

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:

$$F = -kx \Rightarrow \frac{d^2 x}{dt^2} = -\frac{k}{m}x = -\omega_0^2 x, \quad (13)$$

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

$$\frac{dx}{dt} = v_x \quad \text{and} \quad \frac{dv_x}{dt} = -\frac{k}{m}x. \quad (14)$$

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-d, here for time steps of $\Delta t = 0.10$ and $\Delta t = 1.0$. We see that the numerical algorithms reproduce the result of the analytical expression extremely well for small time steps – it's impossible to tell the methods apart. The result is less impressive for larger time step.

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

$$\begin{aligned} E(t=0) &= E_0 = \frac{1}{2}mv_0^2 + \frac{1}{2}kx_0^2, \\ E(t=t_i) &= E_i = \frac{1}{2}mv_i^2 + \frac{1}{2}kx_i^2, \end{aligned} \quad (15)$$

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. 1e-f 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 fourth decimal place. For RK4 with large time steps, the energy is not conserved at all, but declines rapidly.

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

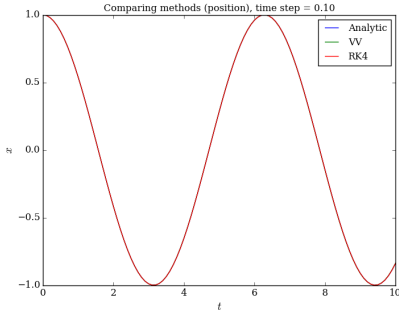
3.2 Stability of methods

Let us first comment on the stability of the analytical case before moving on to the gravitational system. Studying the analytical case with spring force, the two methods described in Sec. 2 are good estimates for time step $\Delta t < 1$. RK4 is slightly closer to the analytical solution. For $\Delta t \geq 1$, the oscillation in position and velocity is damped when using RK4. This is more visible for longer times. With the VV method, the position amplitude is 1, whereas the velocity amplitude is lowered to 0.75. Thus, for larger time steps, VV is the most stable method.

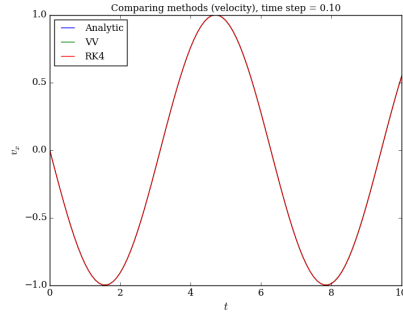
Now let us consider the Newtonian two-body problem in three dimensions. Here we have chosen a configuration similar to that of the Sun-Earth system: One mass is heavy ($M = 1M_\odot$) and has no velocity of its own (Sun), while the other mass is much lighter ($M = 1M_\oplus = 0.000003M_\odot$) with some orbital velocity (Earth). In this scenario we can use that $GM_\odot = 4\pi^2 \text{ AU}^3/\text{years}^2$. By letting Earth sit at a distance 1 AU from the Sun and giving the planet a velocity close to that of the Earth which is 29.78 km/s, we get a system where our Earth-like object follows a nice orbit around the massive object in the center.

As seen in Fig. 2a-d, the orbit – or lack thereof – very much depends on the time step. If the time step is too small, $\Delta t = 0.5$ years in this case, no stable orbit is achieved, though RK4 is hinting at an orbit in Fig. 2a. For a smaller time step of $\Delta t = 0.05$ years, what originally looks like nice Earth orbits are obtained, though the VV orbit is slowly increasing drifts with time, as seen in Fig. 2e. However, for a time step of $\Delta t = 0.005$ years, both methods give stable orbits, both for short and long time scales, as seen in Fig. 2c and Fig. 2f. $\Delta t = 0.005$ years is thus the best time step to use in this case, and based on Fig. 2b and Fig. 2e, RK4 appears to be the more stable of the two methods. $\Delta t = 0.005$ years translates into 1.8 days.

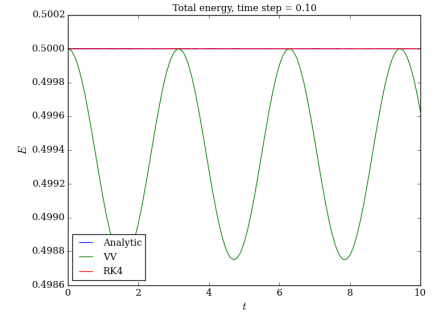
Also of interest is to compare the time used by the two numerical methods to see which one is the most efficient. Instead of quoting the time used to compute *one* time step, we here quote the time taken by the calculation as a whole. See Table 1. Here we have used $t_{\text{final}} = 1.0$. In the 2-body case (we will get back to



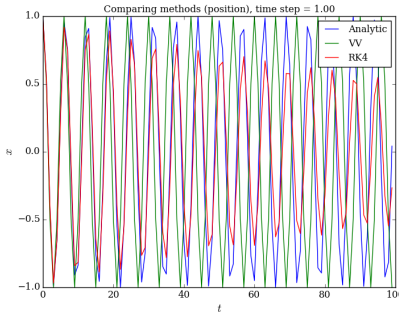
(a) Position, $\Delta t = 0.10$, $t_{\text{final}} = 10$



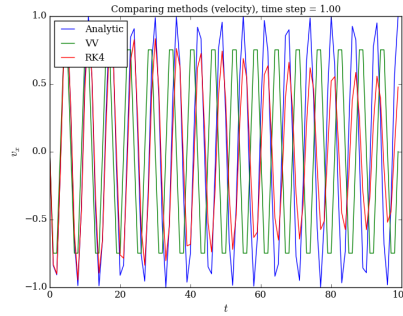
(b) Velocity, $\Delta t = 0.10$, $t_{\text{final}} = 10$



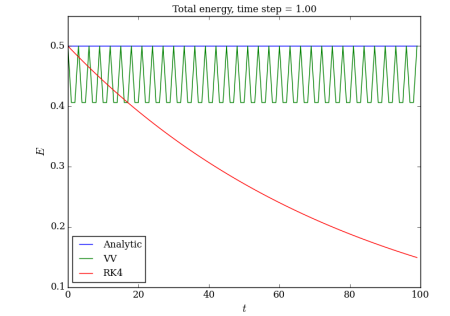
(c) Energy, $\Delta t = 0.10$, $t_{\text{final}} = 100$



(d) Position, $\Delta t = 1.00$, $t_{\text{final}} = 100$



(e) Velocity, $\Delta t = 1.00$, $t_{\text{final}} = 100$



(f) Energy, $\Delta t = 1.00$, $t_{\text{final}} = 100$

Figure 1: Comparison of our numerical algorithms in the analytical case of the box on a spring: Velocity-Verlet (green) and 4th-order Runge-Kutta (red), with the analytical solution (blue) are shown for two different time steps $\Delta t = 0.10$ and $\Delta t = 1.00$, and for two different durations $t_{\text{final}} = 10$ and $t_{\text{final}} = 100$.

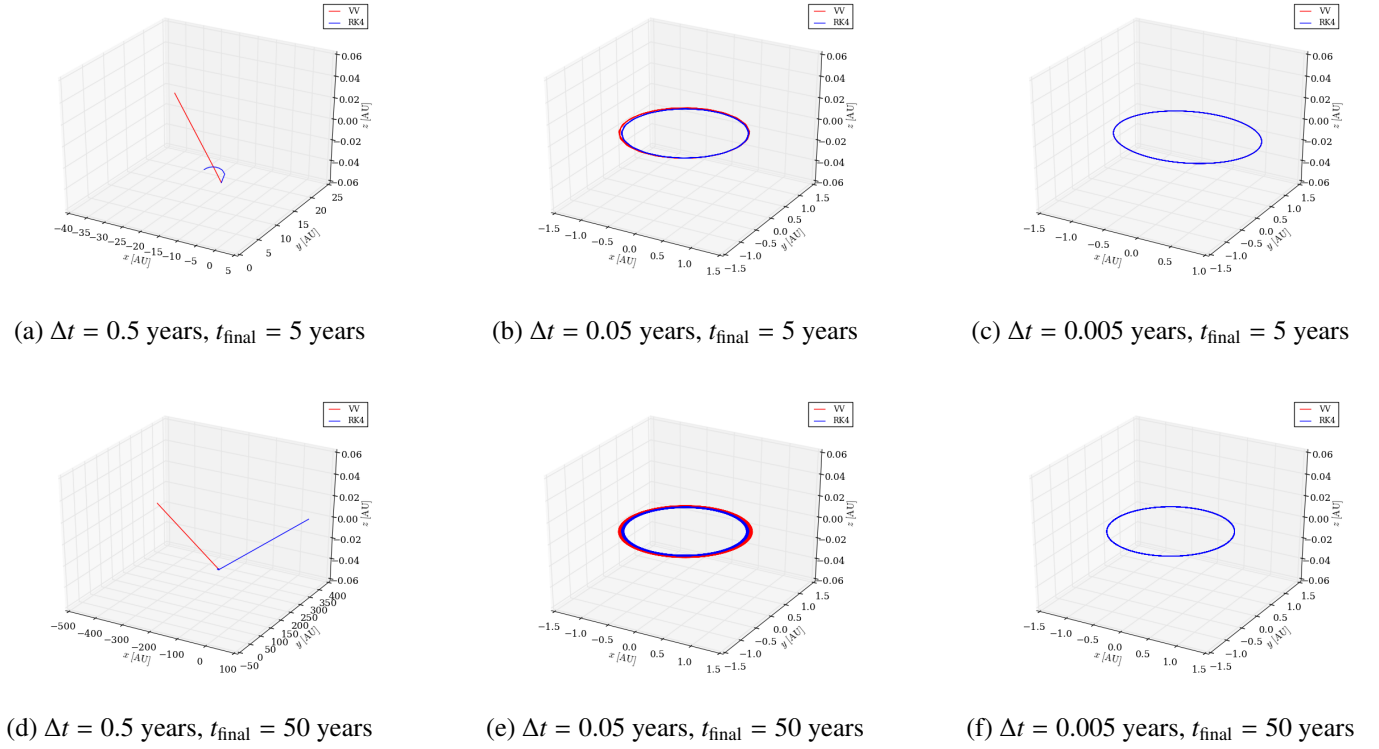


Figure 2: Orbits for our two numerical algorithms in the 2-body case where one body is massive and sitting in the center of the orbit (not plotted): Velocity-Verlet (red) and 4th-order Runge-Kutta (blue) for different time steps Δt , and two different final times $t_{\text{final}} = 5$ and $t_{\text{final}} = 50$. Axes are in AUs.

the 100-body case later) we see that the time usage is very similar for the two methods, increasing by a factor of 10 when the number of integration points is increased by a factor of 10. The times quoted here does however differ from run to run, where the first digits – if rewriting to scientific notation – typically range from 1.5 to 2.6. It also happens that which of the methods is the fastest one changes. In conclusion, the time differences for the two methods are too small for timing to be an issue for choosing one of the methods over the other considering the 2-body case.

In conclusion, based on our study of the stability and time usage of the two methods, we see that RK4 would be the better choice of method when simulating systems that require long times, though it does not matter much which method we choose when the time step is sufficiently small, according to Fig. 2c and Fig. 2f, as well as the results in Table 1.

3.3 Evolution of the N -body system

We go on to study the N -body system where we use a dimensionless G as described in Sec. 2.4 and treat the system in units of light years and τ_{crunch} .

For a system with $N = 100$ objects and a radius $R_0 = 20$ ly, we would first like to know how large the time steps are required to be for the methods to produce good results. We run the calculations for both methods in Sec. 2 for different time steps to see how a change in time step will change the results. This is visualized in Fig. 3a-b where we show the effect of varying the time

step on the position for both methods on one star in the system. Comparing the positions in Fig. 3a and Fig. 3b we see that RK4 suffers more dramatically from the change to a larger time step in this case, as apposed to the 2-body case, while VV keeps its shape impressively well.

Time spent running the two methods is also something to consider. Let us revisit Table 1. In the 100-body case it is now more clear, which it was not in the 2-body case, that VV is the fastest of the two methods, however only by a few seconds, making time usage a non-critical issue.

COMMENTS!

We also want to study the system as a whole by plotting the orbits of all the stars. MORE!

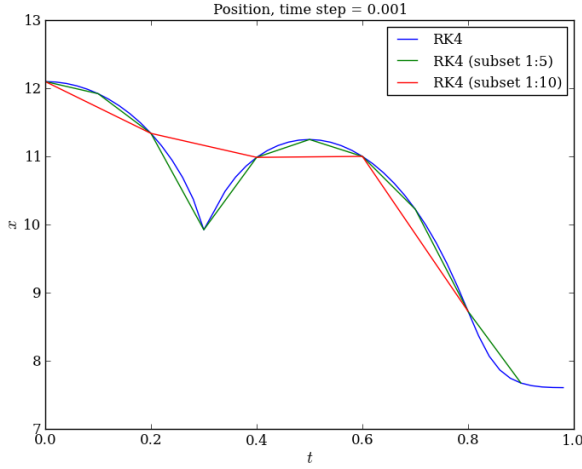
In the case where the number of objects $N \rightarrow \infty$ and the mean density ρ_0 of the system is kept constant, the system would collapse into a singularity at a finite **collapse time** because it can be considered a continuous fluid. Then each imaginary shell for different radii from the center will be fully filled with stars and feel the same gravitational force towards the center of the system, thus reaching the center at the same time.

This will not be the case in our model where N is finite. Then the gravitational field will be largely non-uniform, which means the objects will be pulled towards the center of the system and reach it at different times, and keep moving to the other side of the center, instead of all particles gathering in the center at the same time, creating a collapse. This we can see in Fig. 4.

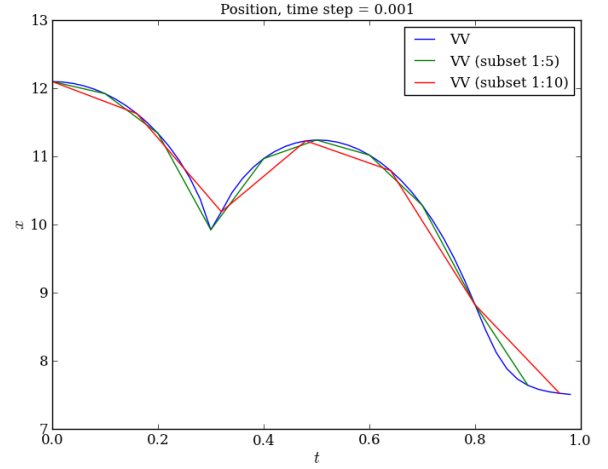
COMMENTS.

Δt	Integration points	Time: 2-body RK4	Time: 2-body VV	Time: 100-body RK4	Time: 100-body VV
0.0001	10 000	0.168853	0.157352	22.0822	18.539
0.001	1 000	0.020674	0.021062	2.10406	1.78809
0.01	100	0.002490	0.002334	0.220827	0.171404
0.1	10	0.000165	0.000147	0.02771	0.018001

Table 1: The time (in seconds) spent running the calculations for the two different numerical methods in the 2-body and 100-body cases.

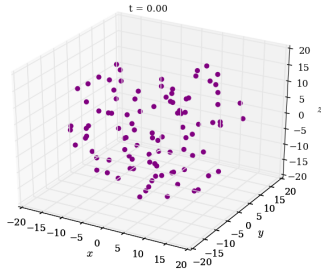


(a) Position (RK4)

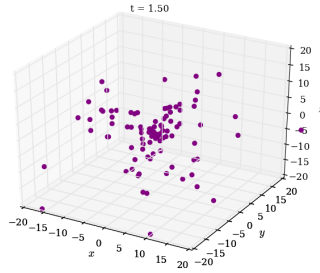


(b) Position (VV)

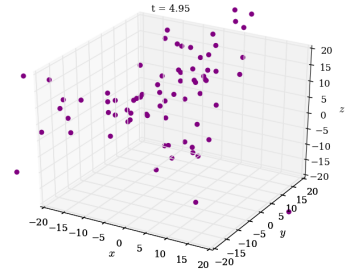
Figure 3: The decay of precision of RK4 (top) and VV (bottom) for three different time steps $\Delta t \in [0.001, 0.005, 0.025]$ for one of the stars in the N -body system.



(a) Initial configuration



(b) Closest to collapse at $t = 1.50$



(c) Final configuration

Figure 4: Evolution of the N -body system as a function of time for a total of $5\tau_{\text{crunch}}$.

3.3.1 The energy of the system

We calculate the kinetic and potential energy of our system to study whether the energy is conserved, and see if this is dependent on the number of objects N , using

$$E = \frac{1}{2}Mv^2 - \frac{GMm}{r} \quad (16)$$

where M is the mass of the star we are considering and m is the mass of another star exerting a gravitational force on our stars. The first term represents the kinetic energy and the second term represents the potential energy due to gravity.

MORE!!!

3.3.2 Introducing the smoothing function

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. Because the gravitational force $F \sim 1/r^2$, we see that the force will become very large as $r \rightarrow 0$, ultimately throwing objects out of the system.

We modify the gravitational force as

$$F_{G,\text{mod}} = -\frac{GM_1M_2}{r^2 + \epsilon^2}, \quad (17)$$

where ϵ is a small real constant. We will try out different values for ϵ to see which value gives the best energy conservation.

Adding the value of ϵ can be understood by noting that we are in reality considering extended objects, rather than point sources. These objects will have a radius R , thus the centers of two objects can only get as close as $2R$ to each other. (An alternative approach could be to let objects merge when getting too close.)

MORE!!!

3.3.3 Testing the virial theorem

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

$$2\langle K \rangle = -\langle V \rangle, \quad (18)$$

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 the stars in our system that are bound (not ejected), we can check if our results are consistent with Eq. (18).

MORE!!

3.3.4 Radial profile

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

$$n(r) = \frac{n_0}{1 + (\frac{r}{r_0})^4} \quad (19)$$

where n_0 and r_0 are scaling constants. The result can also be compared with the well-known Navarro-Frenk-White profile

$$\rho(r) = \frac{\rho_0}{\frac{r}{r_0}(1 + \frac{r}{r_0})^2}, \quad (20)$$

where ρ_0 and r_0 are scaling constants, to see if this profile gives a better fit to our data set.

MORE!!!

4 Conclusions

...

5 References

H. T. Ihle. (2015) *On the, not entirely trivial, issue of getting uniformly distributed random coordinates within a sphere.*

M. Joyce, B. Marcos, and F. Sylos Labini. (2010) *Cold uniform spherical collapse revisited.* AIP Conf. Proc. 1241, 955.

6 List of codes

The codes developed and used in this project are:

6.1 Running calculations (C++)

- `main.cpp` – main program where the calculations are run, calling on the classes listed below.
- `star.cpp` – class collecting various properties of the stars.
- `galaxy.cpp` – class containing all the stars in the galaxy as well as the numerical methods described in Sec. 2.

6.2 Plotting (Python)

- `plotting_analytical.py` – program to graphically compare the one-dimensional 2-body problem with an analytical solution, described in Sec. 3.1.
- `plot_method.py` – program to graphically compare the two numerical methods described in Sec. 2.1 and Sec. 2.2 for the 2-body problem.
- `plot_method_Nbody.py` – program to graphically compare the two numerical methods for the N -body problem.
- `plotting_scatter.py` – plotting program to visualize the 3D distribution of particles in the N -body system as a scatter plot, as shown in Fig. 4.
- `plotting_radialprofile.py` – plotting program that calculates the radial profile of the system, fitted with the profiles in Eq. (19) and Eq. (20).