N-body simulation of an open galactic cluster

Andreas Ellewsen(52), Peder Forfang(55)

Abstract. An open cluster is a typical phenomenon in our galaxies. It consists of a few thousand stars of roughly the same age. They are loosely bound by mutual gravitational attraction and can migrate, through internal close encounters, to the main body of the host galaxy. In this report we have created a simulation of an open cluster based on Newtonian gravity. We study how it evolves through time.

1. Introduction

In this project we develop a code that performs simulations of an open cluster using Newtonian gravity. Since stars in these clusters are about the same age and material, many of the variable parameters are kept constant when compared to each other. Thus they make an interesting study of stellar evolution. The programming languages used are C++ and Python. We use C++ for the calculations, and Python for generating plots and animations. See section 6 for a list of programs used.

First of we will consider the simple Newtonian two-body system. We would like to compare two different numerical approaches to this problem, namely the fourth order Runge Kutta method and the Velocity-Verlet method, to verify that our code works as intended, and to get an idea of which method we should continue using in larger systems. We find that the Velocity-Verlet make a good candidate for larger simulations.

We continue to simulate an open cluster and extend our code to an arbitrary number of stars. We will consider a closed system, meaning it will not be affected by the environment outside of the cluster such as dust and other clouds. In such an over dense system it is natural to predict that it will eventually collapse, virialize and reach an equilibrium state. We will study the evolution of the cluster.

2. The two-body system

The first step is to make a code that can simulate gravity between two bodies. We choose to use the earth-sun system as a start since that gives a nice benchmark for our program.

The gravitational force on earth from the sun is

$$F_G = -G\frac{M_1 M_2}{R^2} \tag{1}$$

which by using Newton's second law yields

$$\overrightarrow{a_2} = -G\frac{M_1}{R^2}\frac{\overrightarrow{R_1}}{R} \tag{2}$$

similarly the acceleration on the sun from earth is

$$\overrightarrow{a_1} = -G\frac{M_2}{R^2}\frac{\overrightarrow{R_2}}{R} \tag{3}$$

where $\overrightarrow{R_1}$ is the vector pointing from the sun towards the earth, $\overrightarrow{R_2}$ is the other way, and $R = |R_1| = |R_2|$.

Using both methods to calculate the velocity and position from this results in figures 1-7. It is clear from the figures that Runge Kutta 4 is more stable than Velocity-Verlet for this system. While Velocity-Verlet starts to migrate outward with $\Delta t = 12$hours, Runge Kutta 4 stays in orbit with steps up to 2 weeks. Increasing the time step even further up to 1 month reveals that Runge Kutta stays in orbit (even if it is 12 sided polygon instead of an ellipse) for a very long time before it migrates inward, and gets thrown out when it gets so close that it gets an acceleration which it can not recover from in the next step.

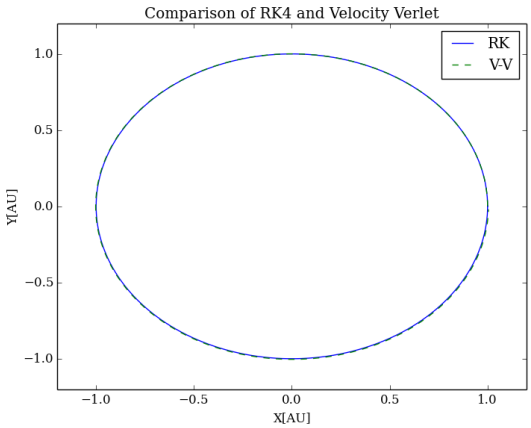

Fig. 1. Simulation of the earth-sun system for 2 years with a time step of 1 hour.

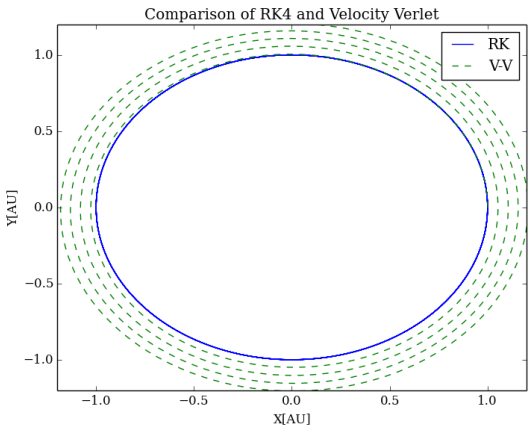

Fig. 2. Simulation of the earth-sun system for 5 years with a time step of 12 hours.

From this we can conclude that Runge Kutta 4 is more stable for large time steps than Velocity-Verlet. However, if Velocity-Verlet is much faster to calculate it may be better in the end for simulating many objects.

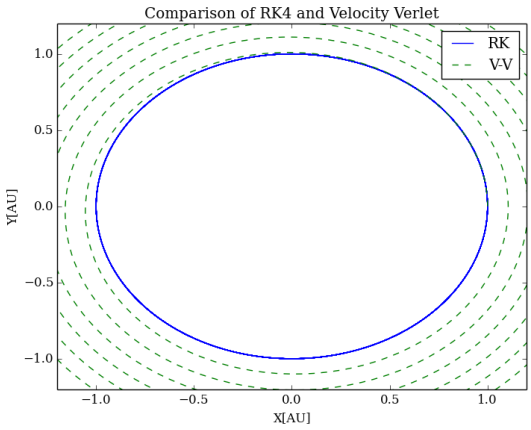

Fig. 3. Simulation of the earth-sun system for 10 years with a time step of 1 day.

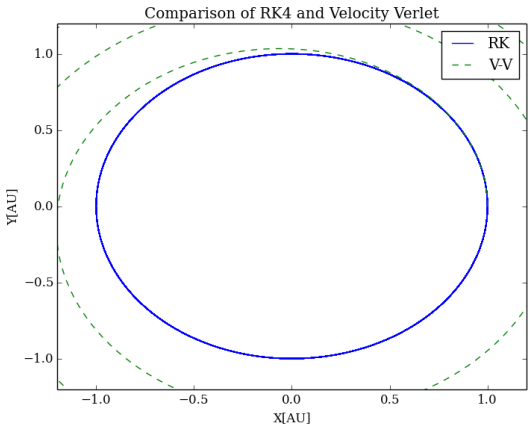

Fig. 4. Simulation of the earth-sun system for 50 years with a time step of 3.5 days.

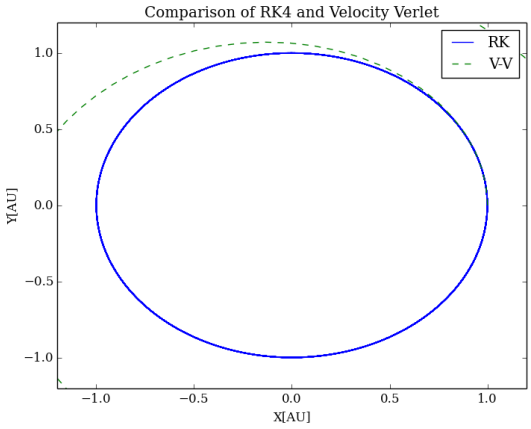

Fig. 5. Simulation of the earth-sun system for 50 years with a time step of 1 week.

Setting the time step to be 1 day, and simulating for 20 years and measuring the average time one step takes shows that Runge Kutta 4 uses 0.0016seconds per step, while Velocity-Verlet uses 0.0008seconds per step. This indicates that Velocity-Verlet is twice as fast that compute. Doing the same with time step 2 days, and simulating for 20 years shows that Runge Kutta 4 uses 0.0009 seconds, while Velocity-Verlet uses 0.00045 seconds per step. This shows that Velocity-Verlet is still twice as fast as Runge Kutta 4.

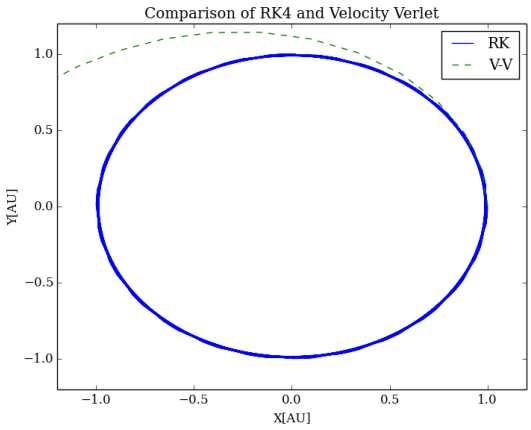

Fig. 6. Simulation of the earth-sun system for 50 years with a time step of 2 weeks.

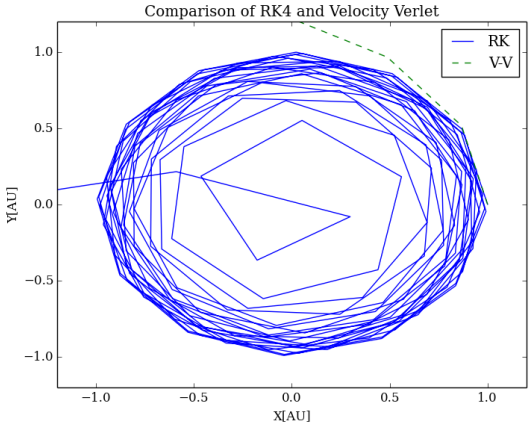

Fig. 7. Simulation of the earth-sun system for 50 years with a time step of 1 month.

However, it also shows that reducing the number of steps to half also reduces the time used for each step by approximately half as well. This indicates that there may be something slowing down the code in a part of the code not related to the methods. This is probably caused by the function that calculates the acceleration taking in the entire armadillo cube containing positions and velocities at all times. If this code is to be developed further after this project this is something that should be investigated to reduce computation times.

From this we can conclude that even though Velocity-Verlet is twice as fast, Runge Kutta 4 can use more than twice as large time steps, and thus simulate the same total time in less than half the time steps resulting in a faster total computation time.

In this case one should use Runge Kutta 4. Note however that this does not say anything about conservation of energy in the system. If one wants to simulate the system for hundreds or thousands of years Velocity-Verlet is the way to go since that method is symplectic, which is not the case for Runge Kutta 4.

The Verlet method is actually a second-order symplectic integrator, and there are higher order ones with better precision, but they need more computation time so we will stick with the Velocity-Verlet method.

3. N-body problem

Since we want to simulate an open cluster we need to extend our code to work on an arbitrary number of objects. An open cluster is a group of up to a few thousand gravitationally bound stars created from the collapse of a molecular cloud. To simulate this we should make our objects uniformly distributed inside a sphere of some radius R_0. Since the size of these clouds are on the light year scales we set $R_0 = 20$ ly. We choose to makes the masses of our objects follow a Gaussian distribution around 10 solar masses with a standard deviation of 1 solar mass.

To make this dimensionless we need a time scale. A reasonable time scale to use would be the time it takes for a gravitationally bound structure of this size to collapse. The answer to this can be found in the Friedmann equations. It turns out that the time it takes can be written

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

Note however that this assumes that our cluster is a continuous fluid, which would mean that $N \to \infty$. If we actually calculate this time for 100 objects inside a sphere of radius R_0, with masses distributed around 10 solar masses we find that $\tau_{crunch} \approx 8$ million years.

Redefining our variables such that $\hat{x} = x/1ly$, $\hat{r} = r/1ly$, $\hat{m} = m/M_\odot$, and $\hat{t} = t/\tau_{crunch}$ means we can rewrite our equation for the acceleration on an object from another in the x dimension as

$$\frac{d^2\hat{x}}{d\hat{t}^2} = -\hat{G}\hat{m}\frac{\hat{x}}{\hat{r}^3}. \tag{5}$$

The equations for the y and z direction look the same only changing $\hat{x}$ to $\hat{y}$ or $\hat{z}$.

Since we want to use τ_{crunch} as our time scale we set $\tau_{crunch} = 1$. This allows us to solve for G resulting in $\hat{G} = \pi^2 R_0^3/8N\mu$, where N is the number of objects, and μ the average mass of the objects.

We simulate this system with both methods. See figures 8 and 9. These figures are quite crowded so it will probably be more interesting to watch the simulations of the system. See movies named RK4N100dt0_001.avi and verletN100dt0_001.avi in the project folder.

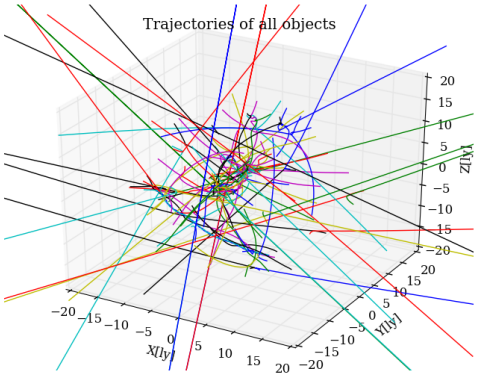

Fig. 8. Trajectories for 100 objects using time step $0.001\tau_{crunch}$, simulating until $1.2\tau_{crunch}$. This simulation was done using Runge Kutta 4.

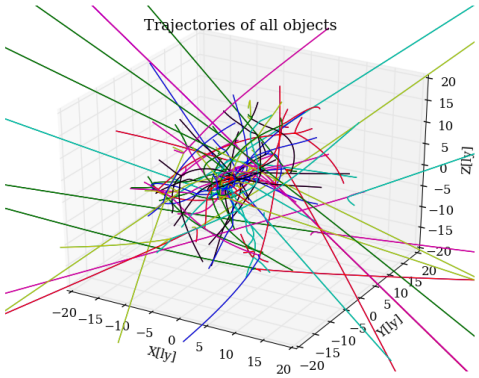

Fig. 9. Trajectories for 100 objects using time step $0.001\tau_{crunch}$, simulating until $1.2\tau_{crunch}$. This simulation was done using Velocity-Verlet.

Now one may expect to see all the objects fall into a singularity at $t = \tau_{crunch}$. This does not happen, and it looks like they collapse later than that as well. Why is this the case? While using τ_{crunch} as a measure of the collapse time of the system works to a degree, our system does not have an infinite number of objects, and we don't have an infinite number of integration points in our simulation. Both of these facts contribute to us not observing a singularity in the model.

For this system watching the simulations or looking at the trajectories does not indicate that any of the methods is better than the other. Using Runge Kutta 4 we observe collapse around $t = 1.2\tau_{crunch}$. A lot of objects are ejected from the system. And it takes on average 0.009 seconds to advance one step. Using Velocity-Verlet we observe the same. There is however the point that it takes half the time to advance one step with Velocity-Verlet. We also know that Runge Kutta does not conserve energy while Velocity-Verlet does.

Since this will both save computation time and conserve the energy of the system we choose to continue with the Velocity-Verlet method.

It would be interesting to see if the system reaches an equilibrium after some time. To study this we run the simulation until $t = 5\tau_{crunch}$. You can see the trajectories of the objects in figure 10. As expected this figure is quite crowded. It is easier to see what is happening by watching the video created from the simulation. This can be seen in the movie named verletN100dt0_001_Tcrunch5.avi in the project folder.

There is no clear equilibrium to be seen directly from the simulation. Since this is the case we choose to make a function that calculates the kinetic and potential energy of the system for each time step. Observing this through time should be a better indicator of whether the energy is conserved or not.

Calculating the kinetic, potential and total energy of the system for each time step returns figure 11. We see that the kinetic energy does some enormous jumps. This should coincide with the times objects get thrown out of the system in the simulation. Clearly the energy is not conserved because of these huge spikes in kinetic energy. Since the kinetic energy of the objects that get thrown out distorts the energy of the system so much it would be interesting to see the behavior of the system if these were removed from the calculation. Figure 12 shows exactly this. In this figure we have increased the simulation time again to see what happens for a few τ_{crunch}. We see that if we exclude the objects which are eventually lost in the calculation of the energy, we get

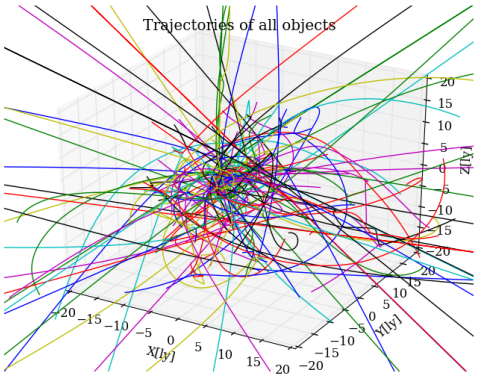

Fig. 10. Trajectories for 100 objects using time step $0.001\tau_{crunch}$, simulating until $5\tau_{crunch}$. This simulation was done using Velocity-Verlet.

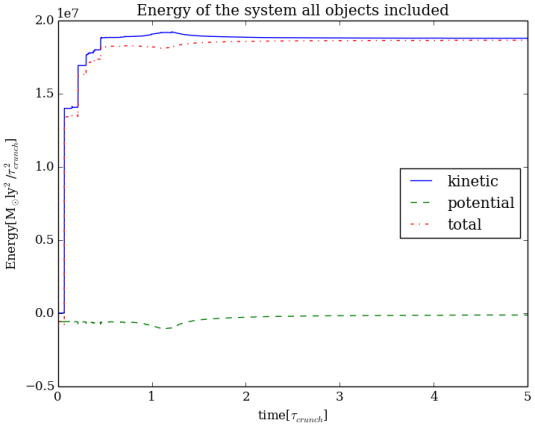

Fig. 11. The figure shows the energies of the system as a function of time. It is clear that energy is not conserved, and it is obvious that the objects that get thrown out of the system contribute very much to the kinetic energy.

a peak in kinetic and potential energy around $1.1\tau_{crunch}$. This is a good sign that the energy function we have created functions properly. Note also that we do not end up with the same total energy at the end as we had at the beginning. This energy difference is caused by the objects that were thrown out of the system. These objects "stole" some of the energy from the bound objects before they left the system. For this simulation it amounts to approximately $E_{lost} = 2.16$ in units $10^5 M_\odot (ly)^2/\tau^2_{crunch}$. This amounts to approximately 70% of the total energy. Note also that the total energy stabilizes shortly after τ_{crunch}. This is easily confirmed by looking at figure 11 where we included all objects. This indicates that the systems stops throwing out objects after it reaches equilibrium. Decreasing the number of objects to $N = 50$ returns figures 13 with all objects included, and figure 14 where only the bound objects are included. For this simulation the energy lost is approximately $E_{lost} = 1.54$ in units $10^5 M_\odot (ly)^2/\tau^2_{crunch}$. This is an order of magnitude lower than for the system with $N = 100$ but in contrast to the earlier system this amounts to about 85% of the total energy the system started with. Doing the same experiment for a system with $N = 200$ shows the same spike in kinetic energy early on, stabilizing at an equilibrium. See figure 15. The interesting thing here is as before the figure with only gravitationally bound objects counted. This

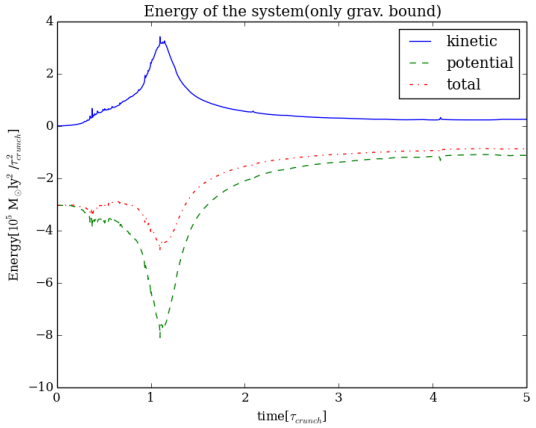

Fig. 12. The figure shows the energies for the system if one only includes the objects that were still gravitationally bound at the end of the simulation. We see a spike in the kinetic energy around $1.1\tau_{crunch}$. This is a bit lower than what we estimated by eye form the simulation. But this is of course more accurate.

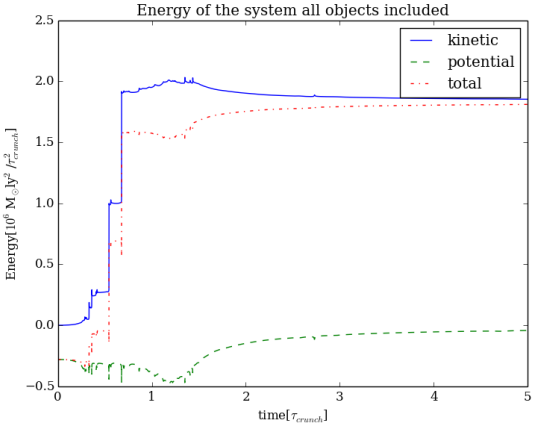

Fig. 13. In this figure we see the same behavior as we saw in the system with $N = 100$. The kinetic energy of the ejected objects dominates.

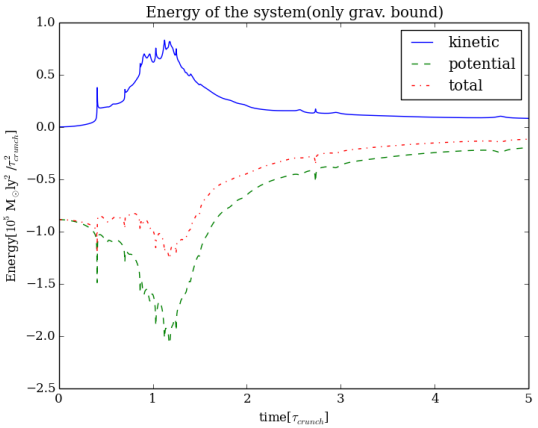

Fig. 14. The figure shows the energies for the system if one only includes the objects that were still gravitationally bound at the end of the simulation. We see a spike in the kinetic energy around $1.1\tau_{crunch}$. We see a slightly different form of the peak but it is still placed around the same time even if it is no.

is plotted in figure 16. We note that the curve is much smoother this time and we get a nice distinct peak at the same time as we have estimated earlier. There is still a substantial energy lost to the ejected objects. In this simulation the loss was $E_{lost} = 4.41$ in units $10^5 M_\odot (ly)^2/\tau^2_{crunch}$. This amounts to about 60% of the total energy.

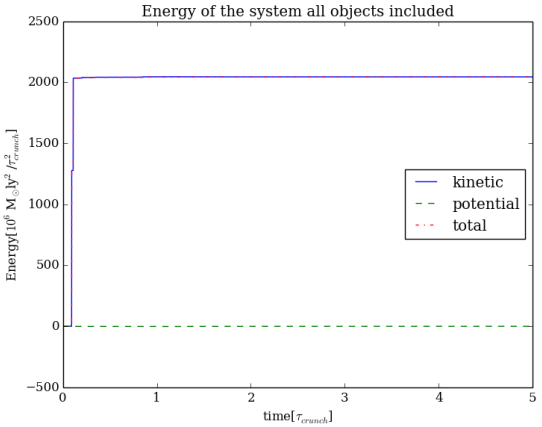

Fig. 15. In this figure we see the same behavior as we saw in the system with $N = 50$ and $N = 100$. The kinetic energy of the ejected objects dominates very early and because of this there is very little information to be extracted from the figure.

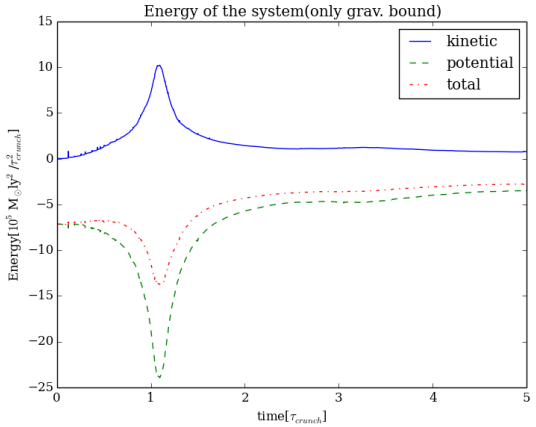

Fig. 16. The figure shows the energies for the system if one only includes the objects that were still gravitationally bound at the end of the simulation. We see a spike in the kinetic energy around $1.1\tau_{crunch}$. The peak in kinetic energy is still placed at the same place, but the graph as a whole is much smoother.

From this it would seem that increasing the number of objects decreases the fraction of energy lost to objects ejected from the system. However there is still an enormous loss of objects.

This loss of objects is caused by objects experiencing a large acceleration when they get too close to another object. Since we do not have infinitely small steps in time, this acceleration which should have lasted for a short time gets applied to the object for too long. To combat this numerical instability we introduce a smoothing function. There are many different methods to introduce such a smoothing. In this project we choose to use a very simply one. Instead of using the regular Newtonian force law we will now use

$$F_{mod} = -G\frac{M_1 M_2}{r^2 + \epsilon^2}. \quad (6)$$

Where ϵ is a "small" constant. The question now becomes, what ϵ value leads to the best energy conservation? And can we justify this value physically?

Trial and error leads us to conclude that we need $\epsilon \geq 0.01$ ly. This becomes clear by looking at the energy plots for the system when using different values. Figures 17, 18, and 19 show the energy for $\epsilon = 1, 0.1, 0.001$ respectively.

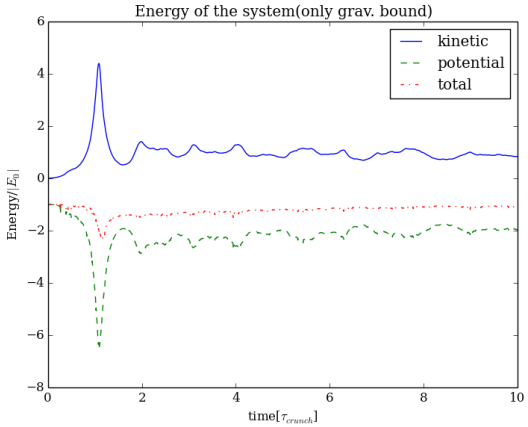

Fig. 17. The figure shows the energies for the system when we only include objects that were still bound at the end of the simulation. This is after we introduced the smoothing function. In this case we used $\epsilon = 1$.

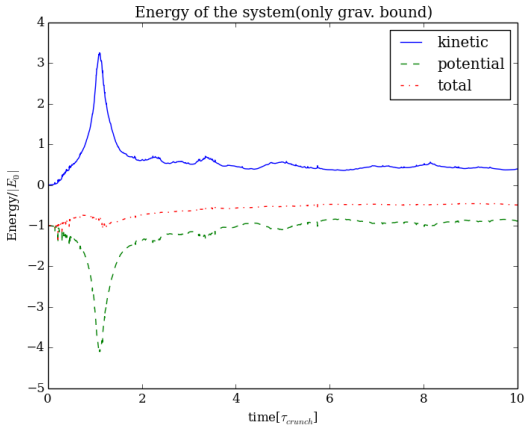

Fig. 18. The figures shows the same as the previous, but this is for $\epsilon = 0.1$.

By introducing the ϵ parameter we have made the system conserve energy better. Note however that the value we need is not easily justified physically. If one tries to argument that ϵ is a kind of minimum distance between the centers of two stars passing by each other, it breaks down since the distances we set it to are enormous. A quick calculation shows that $0.1ly \approx 6300AU$. If one looks at Betelgeuse, a red giant with a mass $M = 7.7 - 20M_\odot$, it has a radius $R = 950 - 1200R_\odot$. It is clear from this that the ϵ parameter can not be thought of as a kind of minimum distance between the objects. Instead we must conclude that it merely functions as an error correcting parameter to the error caused by finite time resolution.

After our cluster has collapsed it will virialize around mass center. The next step in this project is to look at the so called virial theorem. The virial theorem simply states that for a bound

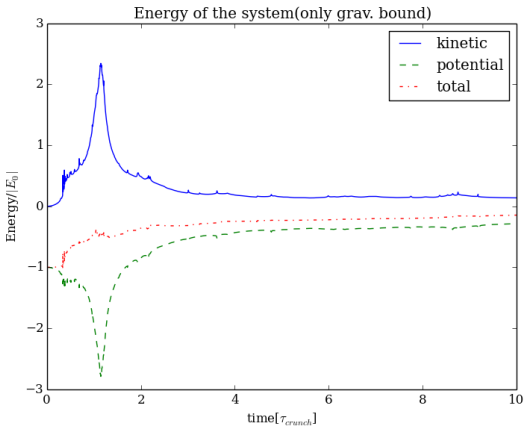

Fig. 19. The figures shows the same as the previous, but this is for $\epsilon = 0.01$.

gravitational system in equilibrium we have

$$2 < K >= - < V >, \tag{7}$$

where $< K >$ is the average (over time) kinetic energy of the system and $< V >$ is the average potential energy. We rearrange the equation to $2 < K > + < V >= 0$ and divide by the initial total energy E_0. We expect this quantity to oscillate around 0 after the system virializes. The smoothing parameter ϵ also affects whether the theorem holds or not. We plot the virialization for the same ϵ values as we did previously.

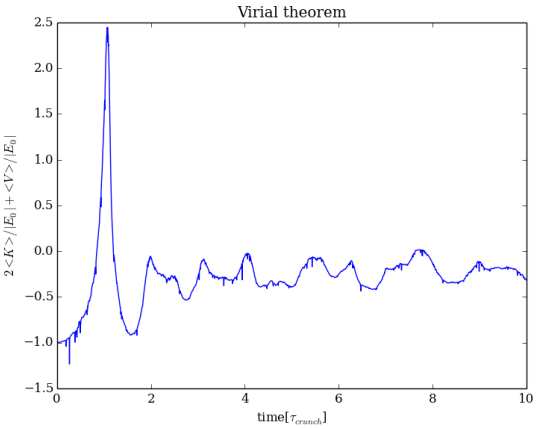

Fig. 20. This figure shows the quantity $2 < K > + < V >$ for $\epsilon = 1$.

The effect of ϵ is shown in figures 20, 21 and 22. As we can see, it has the opposite effect on the virial theorem than on energy conservation. For low ϵ the virialization is not affected, but as we increase the value to $\epsilon = 1$ we see that the graph starts oscillating around some other value than zero. This indicates that the virial theorem is no longer fulfilled.

The conclusion from this is that we get results that are both consistent with energy conservation and the virial theorem as long as we keep $\epsilon \approx 0.01$ ly.

It would be interesting to study the number density of the objects in the halo. To do this we calculate the mass center of the system and make a histogram of the number of objects within a radial distance in steps of 2 light years. The results is figure 23. However, this does not take into account that when increasing the distance form the center, each shell contains a larger volume than

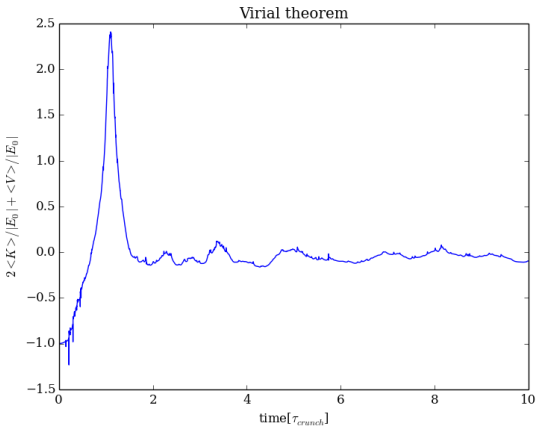

Fig. 21. This figure shows the quantity $2 < K > + < V >$ for $\epsilon = 0.1$.

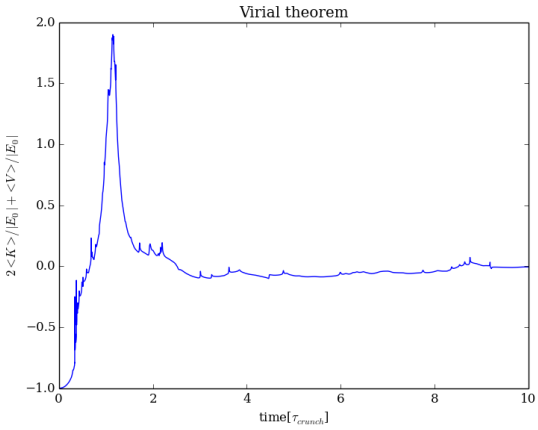

Fig. 22. This figure shows the quantity $2 < K > + < V >$ for $\epsilon = 0.01$.

those closer to the center. When we take this into account we get figure 24 where we have chosen to plot the number density for each distance range as a point half way between the distances. (The number density of objects between 0 and 2 light years is marked with a dot at 1 light years and so on). From this figure we see that we need a lot more objects to obtain any more information than that there seems to be a much larger number density close to the center of the system.

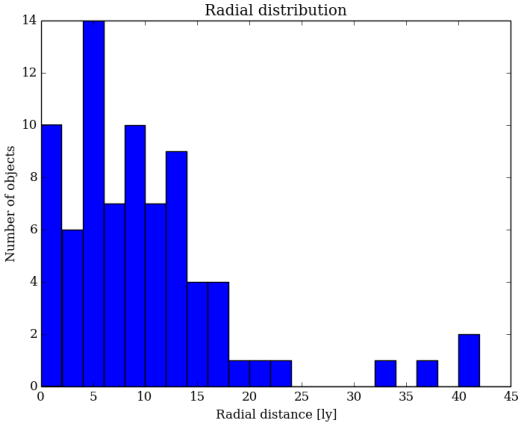

Fig. 23. Radial distribution of objects with $N = 100$.

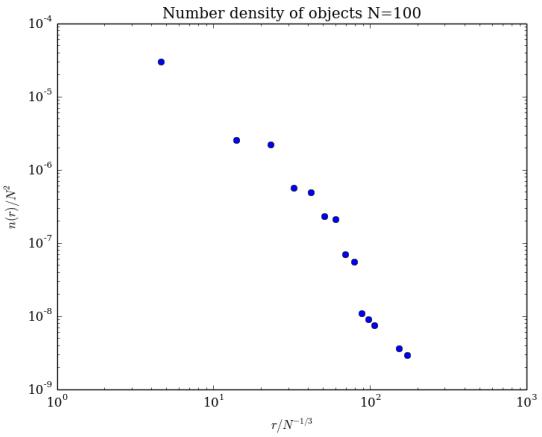

Fig. 24. Number of objects within given distances. It becomes clear than the number density is largest close to the mass center, and it diminishes quickly.

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

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

Attempting to fit this to our curve reveals that setting $n_0 = 0.8$ and $r_0 = 1.5$ly works fairly well. The average distance from the mass center is $r_{avg} = 9.9$ly, with a standard deviation $\sigma = 8.5$. See figure 25. It is still clear that we should increase the number

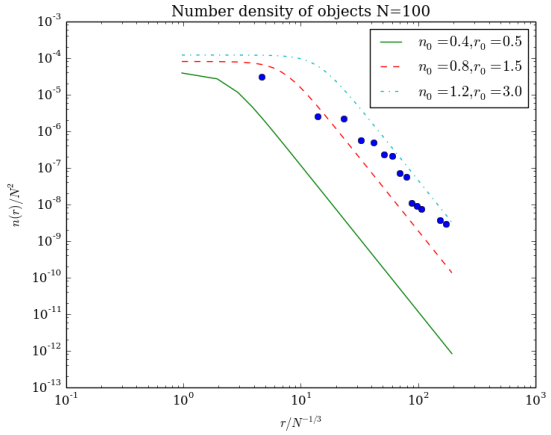

Fig. 25. Number density with a curve fit to the data. This is for $N = 100$ objects.

of objects to get better data. To save computation time we reduce the simulation time to $5\tau_{crunch}$. This is still well within the time frame after the system has virialized and reached equilibrium. With this setup we get an average distance $r_{avg} = 9.4$ly, and standard deviation $\sigma = 10.2$. Figure 26 shows the number density scaled with N^2 as a function distance scaled by $N^{-1/3}$. From the figure we see that the curve with $n_0 = 0.9$ and $r_0 = 3$ is the best fit.

Unfortunately the computer we are using can not handle any more objects than this. Not only does the computation time increase too much. There is not enough memory to read the data files using python. Because of this we are unable to find many n_0 and r_0 values for different numbers of objects N.

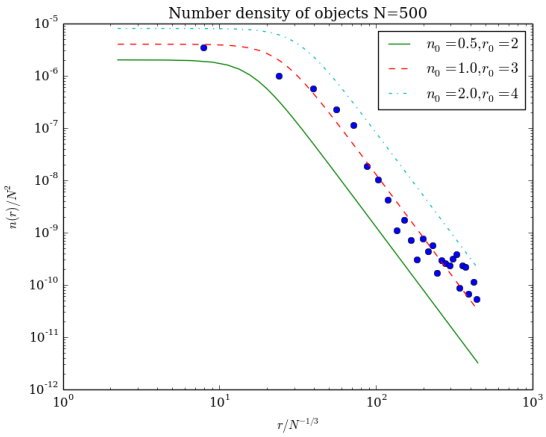

Fig. 26. Number density with curves to fit the data. This is for $N = 500$ objects.

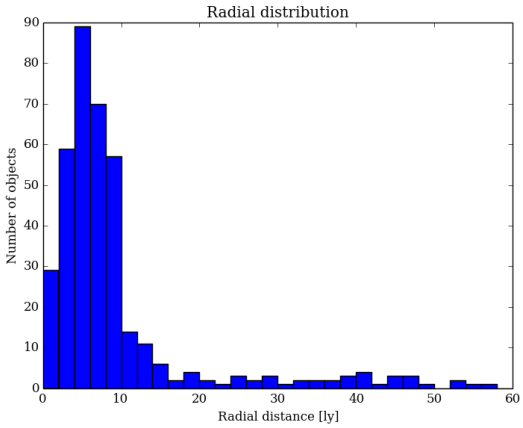

Fig. 27. Radial distribution of objects with $N = 500$.

4. Methods

4.1. Runge Kutta 4th order

The Runge-Kutta of 4th order is a method of numerically integrating ordinary differential equations. A collaboration between calculations in intermediate steps of an interval cancel out the lower order error terms. The final step is obtained by using Simpson's rule, integrating at $t + \Delta t/2$. It can be approximated by

$$\int_t^{t+\Delta t} f(t,x)dt \approx \frac{1}{6}h(f(t,x)+4f(t+\frac{1}{2}h, x+\frac{1}{2}h)+f(t+h, x+h) \quad (9)$$

where h is the step length.

Numerically, the RK4 can be implemented by four steps, each step slightly improves the accuracy by using the result from previous steps.

First, we use Euler's method to compute step $f(t, x)$

$$k1 = hf(t,x) \quad (10)$$

Second, similar to the first step, only this time we compute the slope at the midpoint.

$$k2 = hf(t + \frac{1}{2}h, x + \frac{1}{2}k1) \quad (11)$$

Again we compute the midpoint slope, but improve it by using our previous result k2.

$$k3 = hf(t + \frac{1}{2}h, x + \frac{1}{2}k2) \tag{12}$$

The fourth step then calculates the step $x(t + \Delta t)$ by

$$k4 = hf(t + h, x + k3) \tag{13}$$

The final step can then be computed:

$$x(t + h) = x(t) + \frac{1}{6}(k1 + 2k2 + 2k3 + k4) \tag{14}$$

4.2. Velocity-Verlet

Another approach to integrate Newtons equation of motion is by the method Velocity-Verlet. This algorithm is very similar to the Euler method. The only difference is that the Velocity Verlet calculates an intermediate midpoint velocity step.

$$v(t + \frac{1}{2}\Delta t) = v(t) + \frac{1}{2}a(t)\Delta t \tag{15}$$

This midpoint velocity step is calculated using the acceleration derived from the interacting potential at the present position. With this velocity the next position is calculated.

$$x(t + \Delta t) = x(t) + v(t + \frac{1}{2}\Delta t)\Delta t \tag{16}$$

The methods are similar up to this point. From here we derive a new acceleration $a(t+\Delta t)$ with the new position, and finally the next step velocity.

$$v(t + \Delta t) = v(t + \frac{1}{2}\Delta t) + \frac{1}{2}a(t + \Delta t) \tag{17}$$

In the long term, this extra step improves the result. The Velocity Verlet also has an advantage over RK4. It calculates half the amount of steps as RK4 and thus takes half the time to finish.

5. Conclusions

We decided that the best numerical approach for our case was by using the Velocity-Verlet method. The reason for this was that the fourth order Runge Kutta, although close to perfect at small time steps, turned out to lose energy as we used appropriate time steps for our large scale system. Quite opposite, the Velocity-Verlet is not as accurate in computing the various trajectories, but the method is symplectic and thus conserves energy.

Open clusters are key to understanding stellar evolution. Through internal close encounters stars gain momentum and are ejected out of the cluster. Our simulation lasts over large time scales. To complete the computations in reasonable time the time steps cannot be too small. As a consequence, a numerical instability arise. At close interactions stars gain massive acceleration which last for too long. The result was that they got ejected out of the cluster. We introduced a smoothing function as a simple numerical trick to fix this problem. This trick involved modifying the gravitational force with a constant ϵ. We found that to minimize the disruption of other parameters the best value was $\epsilon = 0.1$.

In our study we have found that the ejected mass and energy loss decreases as we apply more objects to our system. The remaining cluster collapses and starts to virialize. At this point it has reach an equilibrium and is stable against further loss. In reality most clusters are located in the galactic plane and will be exposed to radiation pressure, dust and other environmental forces. These external disruptions can cause instability in the cluster and may catalyze further ejections.

We obtained our best resolution when we randomly distributed 500 objects inside a sphere of radius 20 light years. It takes about 8 million years to collapse. After virialization, most of the remaining objects is located inside a radius of 10 light years. The object farthest away is at 58 light years. In comparison, observed open clusters have a radius up to 180 light years. However, apart from environmental forces, these clusters have up to a few thousand stars.

6. Codes

Here follows a list of programs used to generate our results. The main program that runs functions found in main.cpp and p5funcs.cpp

- main.cpp - main program
- p5funcs.cpp - functions to solve problem
- lib.cpp - library file
- earthsun_plot.py - plots earth-sun system (method 0)
- energyplot.py - plots energy of system
- mass_plot.py - plots mass distribution
- position_plot.py - plots position and saves image files to folder
- moviecreator.py - creates movie from image files in same folder
- radial_density.py - plots radial density

Video files:

- RK4N100dt0_001.avi
- verletN100dt0_001.avi
- verletN100dt0_001_5Tcrunch.avi

7. References

- Morten Hjorth-Jensen, Computational Physics, Lecture notes Fall 2015 (2015)
- Øystein Elgarøy, AST4220: Cosmology I (2009)
- M. Joyce, B. Marcos, and F. Sylos Labini, Cold uniform spherical collapse revisited, AIP Conf. Proc. 1241, 955 (2010),[http://arxiv.org/abs/1011.0614].
- http://www.fisica.uniud.it/ ercolessi/md/md/node21.html