Project 5, FYS 3150 / 4150, fall 2013

Student #

December 7, 2013

1 Introduction

Open clusters are groups of up to a few thousands stars, held together by mutual gravitational attraction. Open clusters generally last for a few hundred milions years. In it also important to underline that the parameters of the cluster's stars are kind of constant, since they are made from the same material. We want to build a model for such a cluster, and study its "cold collapse". To do so, we will begin with the approximation that stars are particles, ponctual points, then we will use a simple smoothing algorithm to increase the numerical stability of our system.

Numerous algorithms could have been used to simulate the behavior of the cluster. We chose to focus on the Leap-frog and on the forth-order Runge Kutta ones.:if each one of the numeric methods has its pros and cons, we want here to find the one which gives priority to the stability instead of the short-term accuracy. Indeed, due to the number of particles in our cluster, we will be interested in the statistical properties of our system instead of in the specific characteristics of each particles.

Our goal in this report is to simulate and to study the behavior of a cluster, after having paid attention to the algorithm used to derive the results. Thus, the first part of the report will be a discussion on the algorithms, and on the methods we implemented to help us, then, we will talk about the cluster's behavior itself.

2 Theory

2.1 Derivation of the expression for τ_{crunch}

Start with Elgarøy's notes (where $t = 0$ at the Big Bang singularity for our sub-universe) and get

$$R(\psi) = a(\psi)R_0 = \frac{R_0\Omega_{m0}}{2(\Omega_{m0}-1)}(1-\cos\psi) \tag{1}$$

where $a(\psi)$ is the dimensionless scale factor and

$$t(\psi) = \frac{\Omega_{m0}}{2H_0(\Omega_{m0}-1)^{3/2}}(\psi - \sin\psi). \tag{2}$$

From these results we see that $t_{max} = t(\psi = \pi)$ and $t_{crunch} = t(\psi = 2\pi)$, thus the elapsed time between these events is

$$\tau_{crunch} = t_{crunch} - t_{max} = \frac{\pi\Omega_{m0}}{2H_0(\Omega_{m0}-1)^{3/2}}.$$

The mass parameter is defined by

$$\Omega_{m0} = \frac{8\pi G\rho_0}{3H_0^2}$$

and for readability we will make the substitution

$$u^2 = 8\pi G\rho_0 - 3H_0^2$$

thus

$$(\Omega_{m0}-1)^{-3/2} = (\frac{u^2}{3H_0^2})^{-3/2} = 3\sqrt{3}H_0^3u^{-3}$$

$$\frac{\Omega_{m0}}{(\Omega_{m0}-1)^{3/2}} = \frac{8\pi G\rho_0}{3H_0^2}3\sqrt{3}H_0^3u^{-3} = 8\sqrt{3}\pi G\rho_0 H_0 u^{-3}$$

$$\tau_{crunch} = \frac{\pi}{2H_0}8\sqrt{3}\pi G\rho_0 H_0 u^{-3} = 4\sqrt{3}\pi^2 G\rho_0 u^{-3}$$

Now we remember that at the time when $\rho = \rho_0$, everything is at rest, so we have $H_0 = (\frac{\dot{a}}{a})_{\tau=0} = 0$. Inserting this, we get $u^2 = 8\pi G\rho_0$, and

$$\tau_{crunch} = 4\sqrt{3}\pi^2 G\rho_0(8\pi G\rho_0)^{-3/2} = \sqrt{\frac{4^2 3\pi^4 G^2\rho_0^2}{8^3\pi^3 G^3\rho_0^3}} = \sqrt{\frac{3\pi}{32G\rho_0}} \tag{3}$$

which is what we wanted to show.

2.2 Lack of a singularity in our model

TODO: Use Elgarøy II and Peacock as sources.

The reason we do not see a singularity in our model is that we have assumed pressureless (i.e. collisionless) matter in the Friedmann equations. In our simulation we will see internal pressure as kinetic energy from the collapse is turned into random motions by near-collisions between particles. This will act as a kind of pressure in our simulation and will halt the collapse, causing the gravitationally bound particles to form more or less stable orbits around the center of mass. Keep in mind that the kinetic energy is not evenly distributed, so occasionally particles that receive more than their fair share will become unbound and may escape from the system before they have time to lose their energy. (SOURCE: Given) predicts that this should happen to roughly ?? % of the particles. (TODO: rewrite?)

We say that the system is stable when the virial theorem

$$2\langle K\rangle = -\langle P\rangle$$

where $\langle P\rangle$ and $\langle K\rangle$ are the averages of the potential and kinetic energies (actually, these are supposed to be time averages for the total energies of the entire system, but it turns out that the averages of a point in time of the energies per particle is a good approximation to this (SOURCE). (SOURCE: Elgarøy II) shows that his happens at time $\tau_{vir} = 0.81\tau_{crunch}$ when the sphere has collapsed to half its initial size, so we see that τ_{crunch} is a natural time scale for virialization to occur (some sources do in fact use τ_{crunch} to mark the point when the system is virialized).

2.3 G in units of ly, $M_{\odot}$ and τ_{crunch}

With τ_{crunch} given in years, we can rewrite equation 3 as

$$G_{yr} = \frac{3\pi}{32\tau_{crunch}^2\rho_0}.$$

Switching time units to τ_{crunch}, we get that $\tau_{crunch} = 1$ in these units, hence

$$G = \frac{3\pi}{32\rho_0} = \frac{\pi^2 R_0^3}{8\mu N} \tag{4}$$

where for the latter equality we have used the definitions of average mass $\mu = \frac{M}{N}$ and initial mass density for a sphere $\rho_0 = \frac{M}{V_0} = \frac{\mu N}{V_n(R_0)}$, where $V_n(R_o)$ is the volume of the n-dimensional ball with radius R_0.

This means our gravitational constant and our time unit both depend on N, R_o and μ. We can verify that the units for G are now correct with R_0 given in light years and μ given in solar masses.

2.4 Calculating ϵ automatically

The challenge with the ϵ values is that we want it to be as small as possible (to give more realistic results), yet make the number of ejected particles as small as possible, conserving as much of the total energy as we can. We chose to determine a good fit for ϵ experimentally (see TODO: FIGURE), and then we determined how this value would scale to different simulations in the following way:

The basic premise is that when $r = \epsilon$ (i.e. when the ϵ term starts dominating the gravitational potential), we want to set an upper limit on how much a particle's velocity can change during one time step due to the gravitational attraction from one other particle. We want this upper limit to be invariant across simulations:

$$\Delta v_1 = \Delta v_2$$

$$a_1 \Delta t_1 = a_2 \Delta t_2$$

We will assume all particles have the average mass μ and set $r = \epsilon$:

$$\frac{G_1 \mu_1 \Delta t_1}{2\epsilon_1^2} = \frac{G_2 \mu_2 \Delta t_2}{2\epsilon_2^2}$$

Now, inserting the value of G from 4 with initial radii R_1 and R_2, we get:

$$\frac{\pi^2 R_1^3 \mu_1 \Delta t_1}{2\epsilon_1^2 \cdot 8\mu_1 N_1} = \frac{\pi^2 R_2^3 \mu_2 \Delta t_2}{2\epsilon_2^2 \cdot 8\mu_2 N_2}$$

$$\frac{R_1^3 \Delta t_1}{\epsilon_1^2 N_1} = \frac{R_2^3 \Delta t_2}{\epsilon_2^2 N_2}$$

giving us this handy formula for ϵ_2 if we have a good match for ϵ_1:

$$\epsilon_2 = \sqrt{\frac{N_1}{N_2} \left(\frac{R_2}{R_1}\right)^3 \frac{\Delta t_2}{\Delta t_1}} \cdot \epsilon_1 \tag{5}$$

2.5 Gravitational potential with modified gravity

Starting with the magnitude of the force

$$F = \frac{GMm}{r^2 + \epsilon^2} = \frac{GMm}{\epsilon^2} \frac{1}{(\frac{r}{\epsilon})^2 + 1} = \frac{GMm}{\epsilon^2} \frac{1}{u^2 + 1}$$

where we have used the subsitution $u = \frac{r}{\epsilon}$ which gives $\frac{du}{dr} = \frac{1}{\epsilon}$, hence $dr = \epsilon \cdot du$, and so

$$E_p = m\Phi = \int F dr = \frac{GMm}{\epsilon^2} \int \frac{1}{u^2+1} \epsilon du = \frac{GMm}{\epsilon}(\arctan(u) + C)$$

We want $E_p \to 0$ as $r \to \infty$, and since $\arctan(u) \to \frac{\pi}{2}$ as $u \to \infty$, we achieve this by choosing $C = -\frac{\pi}{2}$:

$$E_p = \frac{GMm}{\epsilon}(\arctan(\frac{r}{\epsilon}) - \frac{\pi}{2}) \qquad (\epsilon > 0) \tag{6}$$

2.6 Volume of the n-ball

To calculate the gravitational constant with τ_{crunch} as the time unit in any dimension, we need the volume of the sphere in n dimensions to calculate the initial mean density ρ_0. This is accomplished by the following formula:

$$V_n = \frac{\pi^{\frac{n}{2}}}{\Gamma(\frac{n}{2}+1)} \tag{7}$$

(Source.)

2.7 Uniform distribution in the n-ball

Our dimension-independent algorithm for generating uniformly distributed points inside the n-ball is:

1. Generate random points on the surface of the unit n-ball (i.e. randomize the directions of the unit vectors):
   (a) Generate normally distributed n-dimensional vector $\mathbf{x}_n = [x_1, x_2, \ldots, x_n]$ where the normal distribution has $\mu = 0$ and $\sigma = 1$.
   (b) Calculate the n-dimensional norm of the vector $|\mathbf{x}_n|$. We chose to let Armadillo handle this, but another easy way to do it is using the n-dimensional dot product and taking the square root of this: $|\mathbf{x}_n| = \sqrt{\mathbf{x}_n \cdot \mathbf{x}_n} = \sqrt{\sum_i x_i^2}$.
   (c) Turn it into a unit vector: $\mathbf{u}_n = \frac{\mathbf{x}_n}{|\mathbf{x}_n|}$. We refer to (source) for the proof that this is uniformly distriuted in terms of direction.
2. Generate a radius that results in a uniform distribution within maximum radius R_0:
   (a) Generate a uniformly distributed value $r \in [0, 1]$.

(b) The desired radius that takes into account that the outer spherical shells have a larger surface area is $R = \sqrt[n]{r} \cdot R_0$, where n is the number of dimensions.

(Source.)

2.8 Algorithms

For both of our algorithms, the step intervalle looks like $[t_0; t_{max}]$. We will work on n_{Steps}. Let's define h as $h = \frac{t_{max} - t_0}{n_{Steps} - 1}$.

2.8.1 Runge-Kutta

We have$t_i = t_0 + i.h$ for each i belonging to $[0, n_{Steps}]$. We can now define $x_i = x(t_i)$.

The general formulae of Simpson's rule is: $\int_{t_i}^{t_i+1} f(t,x)dt = \frac{1}{6}h(f(t_i, x_i) + 2f(t_{i+\frac{1}{2}}, x_{i+\frac{1}{2}}) + f(t_{i+1}, x_{i+1}))$.

Runge-Kutta's method defines four quantities, to predict and correct the value of x_{i+1}:

$$k_1 = f(t_i, x_i)$$

$$k_2 = f(t_{i+\frac{1}{2}}, x_{i+\frac{1}{2}}) \qquad x_{i+\frac{1}{2}} = x_i + \frac{h}{2}k_1$$

$$k_3 = f(t_{i+\frac{1}{2}}, x^*_{i+\frac{1}{2}}) \qquad y_{i+\frac{1}{2}} = x_i + \frac{h}{2}k_2$$

$$k4 = f(t_{i+1}, x_{i+1}) \qquad x_{i+1} = x_i + hk_3$$

In our case, we will have to compute this results six times by time step: once for the x,y and z positions, and once for the x,y and z velocities.

This Runge-Kutta algorithm is especially interesting since we only need initial conditions to unfold it. Its approximation error runs like $\mathcal{O}(\Delta h^2)$.

2.8.2 Leapfrog algorithm

The Leapfrog algorithm is given by the trhee following steps:

$$x^{(1)}(t+\frac{h}{2}) = (x^{(1)}(t) + \frac{h}{2} * x^{(2)}(t) + \mathcal{O}(\Delta h^2)$$

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

$$x^{(1)}(t+h) = x^{(1)}(t+\frac{h}{2}) + \frac{h}{2} * x^{(2)}(t+h) + \mathcal{O}(\Delta h^2)$$

Now, let's take a look at the approximation error for this algorithm. The first step to derive this algorithm is to use Taylor expansion on the position:

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

In the same way, we have for the velocity:

$$x^{(1)}(t+\frac{h}{2}) = x^{(1)}(t) + \frac{h}{2} * x^{(2)}(t) + \mathcal{O}(\Delta h^2)$$

$$x^{(1)}(t-\frac{h}{2}) = x^{(1)}(t) - \frac{h}{2} * x^{(2)}(t) + \mathcal{O}(\Delta h^2)$$

$$x^{(1)}(t+\frac{h}{2})-x^{(1)}(t-\frac{h}{2}) = x^{(1)}(t)+\frac{h}{2}*x^{(2)}(t)+\mathcal{O}(\Delta h^2)-(x^{(1)}(t)-\frac{h}{2}*x^{(2)}(t)+\mathcal{O}(\Delta h^2)) = h*x^{(2)}(t)+\mathcal{O}(\Delta h^2)$$

So

$$x^{(1)}(t+h) = x^{(1)}(t+\frac{h}{2}) + \frac{h}{2} * x^{(2)}(t+h) + \mathcal{O}(\Delta h^2)$$

We can rewrite the first step thanks to the expression of $x^{(1)}(t+\frac{h}{2})$:

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

In the end, we can see that the approximation error runs like $\mathcal{O}(\Delta h^2)$ for the Leapfrog algorithm.

3 Implementation

3.1 Overview

The code is designed to be highly modular and independent of dimension to maximise reusability.

Figure 1: Class Diagram

3.2 SolarSystem class

Container class for the entire N-body system. This class is dimension independent (number of dimensions is given as a parameter), to make it as general and reusable as possible. Does not specify a method to solve the equations of motion, so any numerical method can be used on this class to update the positions every time step (by iterating over the celestial bodies contained in this class). A SolarSystem can make deep copies of itself (and all its CelestialBody and Gravity objects), which we make use of to be able to run different algorithms on identical copies of a system (for comparison of results and stability analysis).

3.3 CelestialBody class

Particle class of our N-body simulation, with position and mass. Handles all calculations on the individual particle level (e.g. kinetic energy) and can be set to fixed if desired. Also stores properties calculated by the encompassing SolarSystem (like potential energy) so these will not have to be calculated on the fly every time they are needed. This class inherits dimensionality from the SolarSystem it belongs to, so we avoid creating a sub-dimensional celestial body by accident.

3.4 Gravity class

This class allows each SolarSystem to use a different gravity. One can change both the value of the gravitational constant to fit the time unit of choice and set an ϵ value to dampen collisions. For future use it is possible to extend this class to include any form of modified gravity (which is of interest in cosmology, SOURCE). The only thing the SolarSystem class requires of this object is that it is able to return a force and potential energy when two CelestialBody objects are given as input. This way, a SolarSystem does not need to handle the specifics of gravitational forces itself, making the code more modular.

3.5 CelestialBodyInitializer class

Sets up a uniform position distribution given an initial max radius (generalized for any number of dimensions). Also generates normal distributed random masses. Doing this in a separate class allows the SolarSystem class to be as general as possible (we might not want a random distribution every time, but when we do, this class will do the job). Keeping this class dimension independent makes it more general and easier to use in other projects.

3.6 Solvers class

Contains code to iterate over a SolarSystem object over a number of time steps, using the Leapfrog, Runge-Kutta (4th order) and Euler-Cromer algorithms to solve the equations of motion. Creates copies of the system given as a parameter to do this, so we can keep the original system unchanged and also solve using several numerical methods simultaneously and compare the results. Keeping this in a separate class allows adaptation of this code to use other objects than the SolarSystems we employ here.

3.7 GaussPDF class

This is simply a static class wrapper for the code given at:

https://www.uio.no/studier/emner/matnat/fys/FYS3150/h13/gaussiandeviate.cpp

It provides random numbers in the uniform and normal probability distributions.

4 Results and analysis

4.1 Benchmarks

Reproduce project 3 2-body problem.

Center of mass conserved.

Energy mostly conserved, especially with gravitational correction.

We saw in the theory part that the forth-order Runge Kutta has an approximation error smaller than the Leapfrog algorithm. It basically means that the local data about each particles will be more accurate but since the Leapfrog method is more stable than the Runge Kutta method, and gives satisfying results for small enough time steps. The Runge Kutta method is more accurate to compute the velocity, but here, it does not matter so much. It is also important to underline the fact that the Runge Kutta method generates an oscillation on the energy, while the energy is stable with the Leapfrog one. And if the time step gets to big, for both of our algorithm, everything becomes messy, and not usable.

Execution time (s)	For 1000 steps, two bodies	For 1000 steps, 100 bodies
Leap-frog	19,18	86,35
Forth-order Runge Kutta	30,85	171,25

Table 1: Elapsed Time to process the two body system

We can see here that the Runge Kutta algorithm takes more time to process the same set of data than the Leapfrog. For small data set, the time spent by the forth order Runge Kutta algorithm is twice as much as the time spent by the Leapfrog method. And when we increase the number of steps, or the number of bodies, we increase the time spent during processing. Thus, the difference in the execution time between Runge Kutta and Leapfrog becomes more and more significant.

According to the time required to process equivalent results, the Leapfrog algorithm seems more interesting. Furthermore, we are going now to process on a N-body system, where N is not smaller than 100. Thus, we will be more interested in the statistical properties of our system, than in the local accuracy of the position or the velocity.

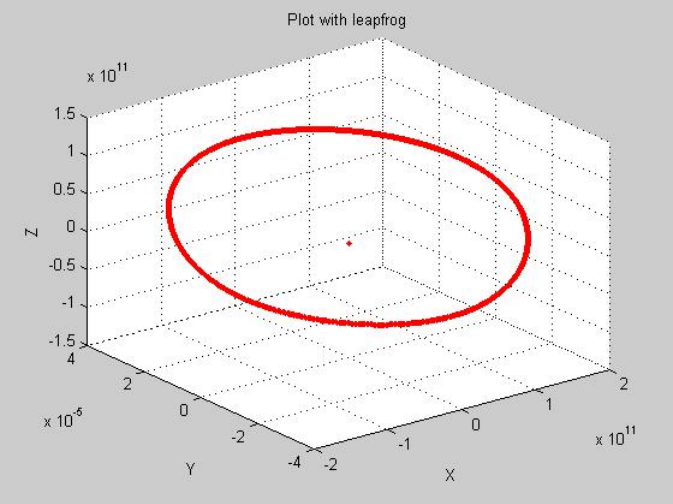

Figure 2: Conservation of the Energy – Leap-frog – Length = 2 years ~

```
E_k before: 1.33206e+033
E_p before: 0
E_tot before: 1.33206e+033
Entering the Benchmark part
Using Leap-frog
Aaand ... We're done
E_tot after: -1.62098e+033
E_k after: 1.37681e+033
E_p after: -2.9978e+033
```

Figure 3: Conservation of the Energy – Leap-frog

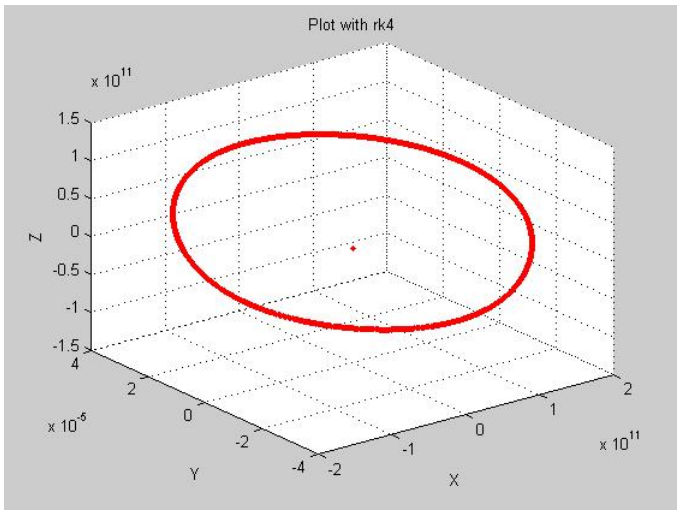

Figure 4: Conservation of the energy – forth-order Runge Kutta = Length = 2 years ~

```
E_k before: 0
E_p before: -2947.33
E_tot before: -2947.33
E_k before (bound): 0
E_p before (bound): -2947.33
E_tot before (bound): -2947.33
Center of mass (all):   -0.8268
   1.0058
  -2.3687

Running simulation.....................................................
.....................................DONE!

E_k after (bound): 3816.15
E_p after (bound): -6691.61
E_tot after: -931.65
E_tot after (bound): -2875.46
E_k after: 5246.59
```

Figure 5: Conservation of the energy – forth-order Runge Kutta – A changer !!!

Length - Time step size	2 years - 1 day			300 years - 1 day		
Leap-frog	1,33206e33-1,33206e33					
Runge Kutta 4						

Table 2: Not so satisfying é_è

4.2 Application to a multi-bodies system

4.2.1 Evaluation of G

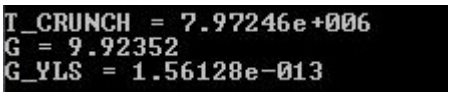

Figure 6: Computation of G

With a system of $N = 100$ particles, an initial radius of $R_0 = 20$ light years, we found a τ_{crunch} of nearly 8 millions of years. In the figure above, the variable called G_YLS is the gravitational constant, uses meters, seconds, and kg. G is computed in ly, $M_{\odot}$ and τ_{crunch}

4.2.2 Energy conservation (finding a reference value for ϵ)

With a system of $N = 100$ particles, an initial radius of $R_0 = 20$ light years and a time step of $\frac{1}{250}\tau_{crunch}$, we did a series of simulations where we varied the gravitational correction parameter in the interval $\epsilon \in [0,0.15]$. According to equation 5, the values are supposed to be independent of mass, but the masses used were a normal distribution with mean $\mu = 10$ and $\sigma = 1$ (both in solar masses).

The criteria for accepting an ϵ value are that the number of bound particles are as high as possible, while the total energy, at least for the bound particles, is as conserved as possible. In the process, we discovered that these criteria are actually mutually exclusive: The energy conservation improves if we eject many particles. Our interpretation of this result is that particles that are bound but close to being unbound will impact the energy bound conservation negatively: If these particles are actually ejected, the remaining particles' total energy is better conserved. Thus, we need to look at both criteria simultaneously when deciding on a value for ϵ. Additionally, we want ϵ to be as close to 0 (and the Newtonian limit) as possible, so we get more realistic results.

Looking at (SOURCE), we see also that the number of ejected particles will fluctuate considerably between simulations with the same starting parameters. Ideally, we would run many simulations with the same epsilon value to eliminate statistical errors - this is something that could be looked at in a future study.

Equilibrium As discussed in the theory part, according to the virial theorem, the system reachs an equilibrium after

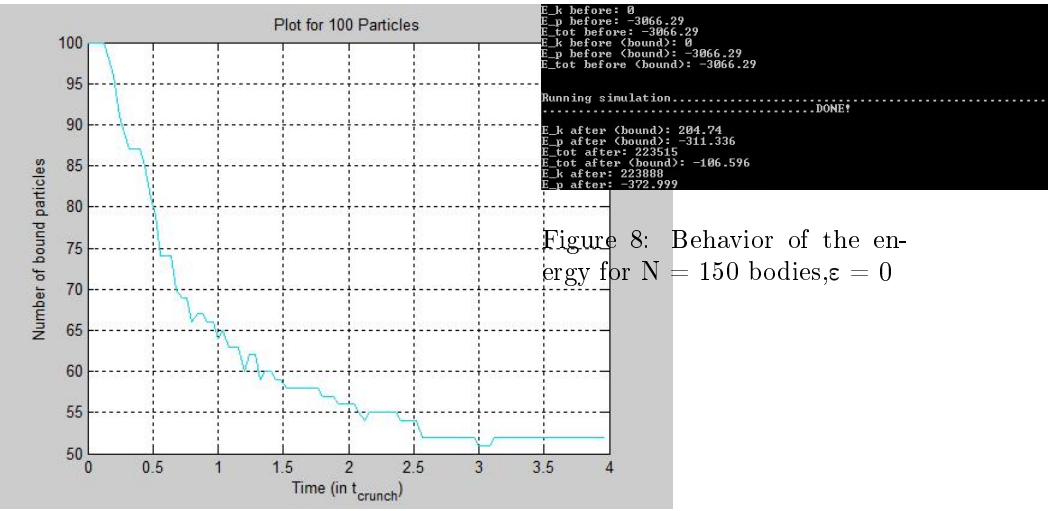

Figure 8: Behavior of the energy for N = 150 bodies,ε = 0

Figure 7: Number of bound particles for ε = 0

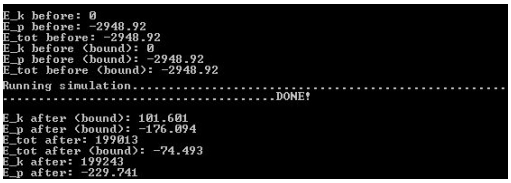

Figure 9: Behavior of the energy for N = 300 bodies,ε = 0

Particles ejection Without using the modified gravitational potential, we notice that the energy is not conserved: some particles are ejected of the system. For these particles, the kinetic energy blows up, and the total energy of theses particles is positive.

The bigger N we take, the bigger the quantity of energy the system is taken away. For the biggest N we tried, the energy taken away is nearly 10 times the energy of the initial system.

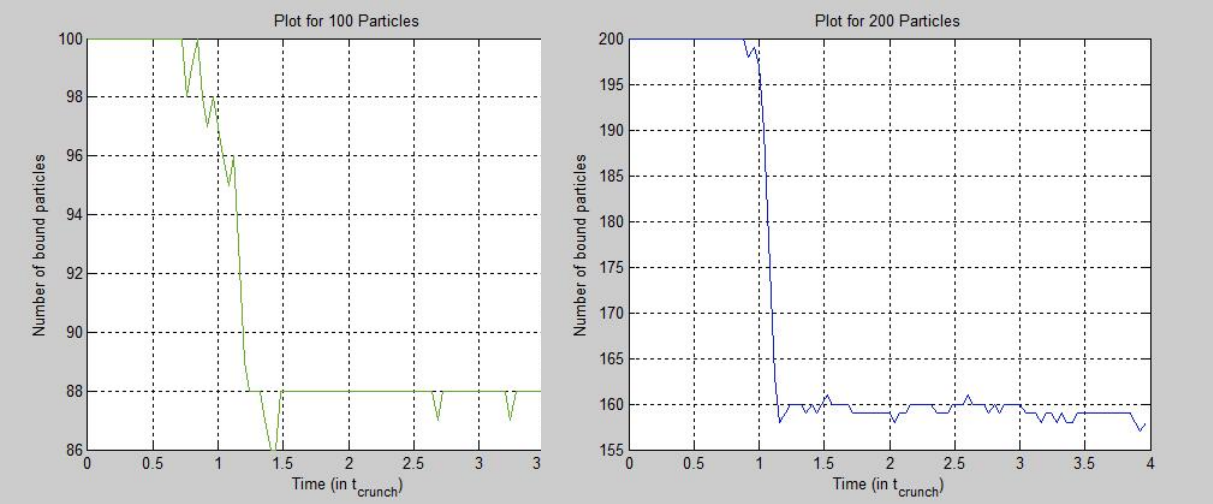

Figure 10: Number of bound particles for $\varepsilon = 0.10005$

Figure 11: Number of bound particles for $\varepsilon = 0{,}071$

```
E_k before: 0
E_p before: -2963.82
E_tot before: -2963.82
E_k before (bound): 0
E_p before (bound): -2963.82
E_tot before (bound): -2963.82
Running simulation.....................................................
.....................................DONE!

E_k after (bound): 1206.49
E_p after (bound): -2195.98
E_tot after: 631.039
E_tot after (bound): -989.484
E_k after: 2877.47
E_p after: -2246.43
```

Figure 12: Behavior of the energy for N = 300 bodies, $\varepsilon = \sqrt{0{,}0225}$

After adding the smoothing part in our gravitational potential, we can see that the quantity kinetic energy does not blow up anymore, and the quantity of energy lost is a lot smaller than before: this was expected, since we reduced the numerical instability by adding this ε factor in the calculation of our Newtonian force. Thus, the behavior of particles when they come closer to each other is more stable: we lose less particles than before.

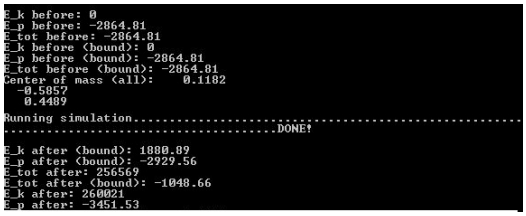

Figure 13: Evolution of Energy with the Leapfrog algorithm, ε = 0

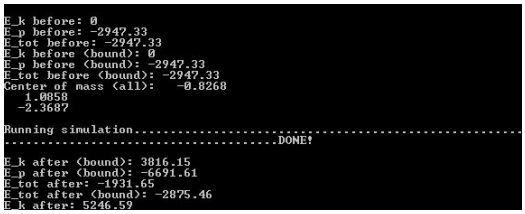

Figure 14: Evolution of Energy with the Leapfrog algorithm, ε = √0,0225

Virial Theorem After running the system for a few τ_{crunch}, we evaluate the kinetic and the potential energies for every particles of our bound system. What is displayed here is the value of the average kinetic energy, $\langle K \rangle$, and the average of the potential energy,$\langle P \rangle$, for our bound system. We can see that the virial theorem is met in the case where the smoothing function is used, with an ϵ of $\sqrt{0,0225}$, and is not met in our initial case (i.e. without using the smoothing function). The smoothing function helps us to take care of the numerical instability generated when two particles are too close. Indeed, with the standard gravitational potentiel, we lose too many particles. Thus, the introduction of ε solves this problem.

We took ε = $\sqrt{0,0225}$, which was a satisfying value for the respect of the total energy conservation.

5 Conclusion

What we learned:

- ?

5.1 Critique

- ?