Project 5, FYS 3150 / 4150, fall 2013

Student #

November 29, 2013

1 Introduction

?

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

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

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

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

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. (We can also check our virialization radius based on this.)

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.

We say that the system is stable when the virial theorem (...) is satisfied. 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

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

where for the latter equality we have used the definitions of average mass $\mu = \frac{M}{N}$ and initial mass density for a sphere $\rho_0 = \frac{M}{V_0} = \frac{\mu N}{\frac{4}{3}\pi R_0^3} = \frac{3\mu N}{4\pi R_0^3}$.

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 μgiven in solar masses.

2.4 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 $C = -\frac{\pi}{2}$:

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

2.5 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:

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

(Source.)

2.6 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.)

3 Implementation

3.1 Overview

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

(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. Currently returns the resulting system of the Leapfrog algorithm, since that is the one we use the most (but this behaviour is easily changed if desired). 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/FYS3
It provides random numbers in the uniform and normal probability distribu-
tions.

4 Results and analysis

4.1 Benchmarks

Reproduce project 3 2-body problem.

Center of mass conserved.

Energy mostly conserved, especially with gravitational correction.

5 Conclusion

What we learned:

- ?

5.1 Critique

- ?