PHYS512 Project: N-Body Simulation

Jehan Dastoor

(Dated: December 19, 2020)

To produce an n-body simulation, we proceed with the particle-mesh method whereby we calculate the density of the particles on a grid and the potential over that grid by convolving the density with the potential for a single particle, using the green's function. The green's function for the gravitational Poisson equation $\nabla^2\phi = 4\pi G\rho$, where ρ represents the density on the grid, can be solved as:

$$Greens = -\frac{1}{4\pi G\sqrt{k_x^2 + k_y^2 + k_z^2 + soft^2}}, \tag{1}$$

where $soft$ is a softening value used to reduce errors and k_x, k_y and k_z are the distance from the particle (or we can think of it as the corners of the mesh squares). Furthermore, we can center our green's function at the center of our mesh by subtracting from each k half the grid size and adding 0.5 so that the potential is centered in the middle of each cell. We can determine the total potential by convolving Eq. 1 with the density of particles on this grid (which can be determined by computing a 3D histogram). Ideally, we could calculate this density using a Cloud in a Cell (CIC) model, however given time constraints a basic nearest cell model was used. Finally, the acceleration can be calculated by taking the negative gradient of the potential to get the force, and then dividing this by the mass of each particle. Position can now be determined using a Leap-frog method whereby,

$$x_{i+1} = x_i + v_i\Delta t + a_i\frac{\Delta t}{2}, \tag{2}$$

$$v_{i+1} = v_i + (a_i + a_{i+1})\frac{\Delta t}{2}, \tag{3}$$

where x represents position, v represents velocity, Δt is our time-step and a represents our acceleration. The entire code can be seen in "nbody_class.py" (where "animation_class.py" is used to evolve the system). We now proceed with the questions. Periodic boundary conditions are applied by taking the modulo of our position at each time step to make sure that if it had left the grid we would push it back to the other side. Non-periodic boundary conditions were applied by first padding our gradient so that there are no border effects from the other side of the grid (which result from doing a discrete Fourier transform) and then by removing particles from the simulation if they left the grid using np.argwhere and np.delete. Details again can be seen primarily in "nbody_class.py". Note, $G = 1$ for the entire simulation to avoid using very large masses for no reason. Furthermore, the simulation is designed such that all grids are cubes, the grid spacing is always fixed at 1 and only even sided grids can be used. Heat maps were produced along the way to make sure that the potential produced was accurate (this code is commented out still in "nbody_class.py").

1. We can plot the position of a single particle given no initial velocity to see if the particle remains still. As we can see from "problem_1.gif", the particle remains still. Great! We can also see that the total energy remains constant which is good as it tells us that there is no slight movement of the particle which we are not detecting.

2. To get a circular orbit, we assume that the particle moves only in the $x - y$ plane and does not move in the z-direction. Furthermore, let's assume that the first particle starts in the middle of the grid, separated by a distance r_1 in the y-direction (r_2 for the second particle), i.e. all our velocity is in the x-direction (take all particle masses to be 1). The particles have mass m_1 and m_2, respectively. We can then calculate the velocity using the equation:

$$v = \sqrt{\frac{G(m_1 + m_2)}{(r_1 + r_2)^3}} r_1. \tag{4}$$

Giving particle 1 $v_x = v$ and particle 2 $v_x = -v$, we produce the following gif that can be seen in "problem_2.gif" (with a softening value of 0.1, total grid size of 6x6x6.). Note, that due to other obscurities in the code, we multiply this by a factor so that we do get as close to a circular orbit as possible. The fact that the grid spacing is fixed at 1 does not help in this section as the particles need to be close to interact nicely, yet if they get too close they both fall into the same bin and do not get the correct forces applied to them (plus they no longer experience a $\sim 1/r$ type potential at very close distances). This is why we tend to see this curve in our total energy as the particles are being pushed apart and pulled together due to the large grid spacing. In the future, the spacing could be decreased so that we could see more minuscule details.

3. Both of these simulations were done with a softening value of 0.01, a size of 50x50x50 and 150,000 particles. We set our system to have periodic boundaries first. As we can see from "problem_3_periodic.gif", the particles seem to form clumps in local areas. Meanwhile, total energy has remained fairly constant and only just starts creeping up. Furthermore, these clumps are unstable and break down fairly quickly. As they break down, total energy starts to rise very quickly as the particles quickly speed up going through one side and coming out the other. Hence, energy is not conserved that well. However, the rise in total energy does peak and begin to fall again, looking as if to repeat the cycle that already happened.

On the contrary, when we use non-periodic boundary conditions, we see in "problem_3_nonperiodic.gif" that the particles seem to all collapse into the center before dispersing. Energy rises very quickly as the particles come together in the center, and then falls as the particles are pushed back out. The total energy then keeps falling as particles are eventually pushed out of the grid and removed from the simulation. Hence, energy is not well conserved, yet this is expected for the non-periodic boundary conditions.

4. We set up our system to have periodic boundary conditions and mass that scales according to k^{-3} such that we can simulate the start of the universe. We do this in "mass_class.py" taking inspiration from https://garrettgoon.com/gaussian-fields/. We can plot a 2D heat map in the xy-plane of our normalised inverse Fourier transform of the k^{-3} power spectrum convolved with white noise. This can be seen below.

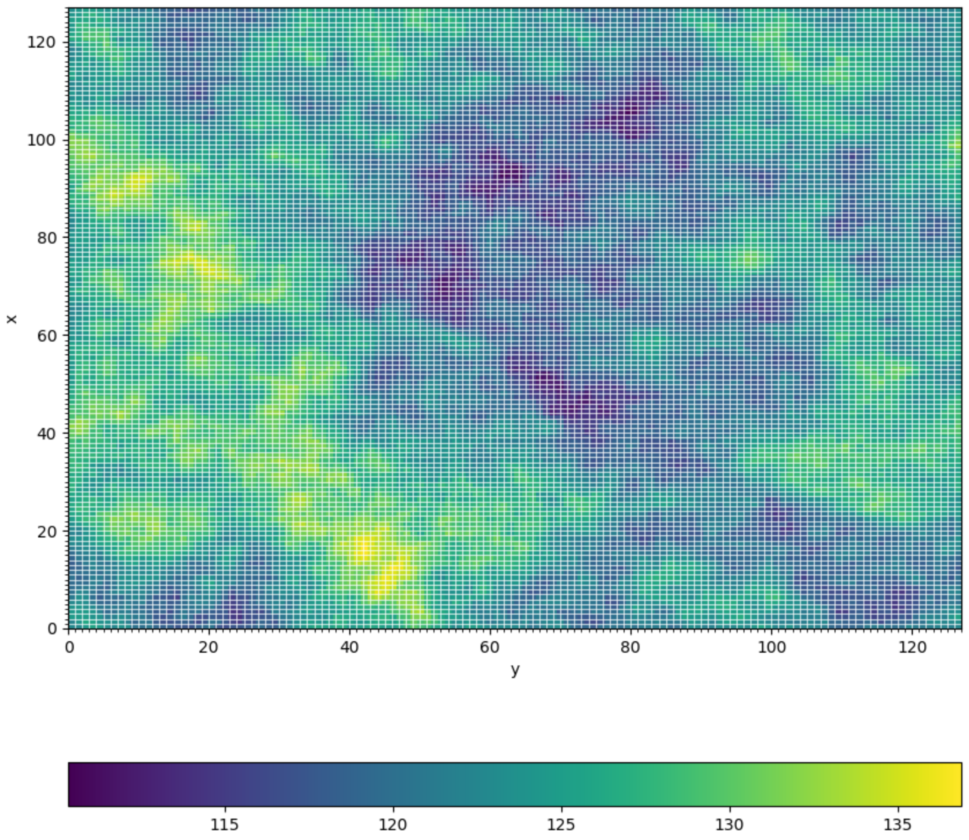

Figure 1: 2D heat map of the normalised inverse Fourier transform of the k^{-3} power spectrum convolved with white noise (Fourier transform of the normal distribution).

This heat map looks very similar to what we would expect for a k^{-3} power spectrum! We can now use this to produce masses according to a uniform distribution that has mass ranging between 0.1 and the value of this mass probability density function at the cell that the particle is in. Doing this leads to "problem_4.gif" (150,000 particles, 0.1 softening, a size of 128x128x128 and particles initially at the center of their respective cells), which as we expect behaves similarly to the periodic boundary condition in part 3. Noticeably it takes longer to reach a stage where the particles collapse in together and it forms seemingly larger groups of particles that stick together. Furthermore, the energy remains stabler for longer which is indicative of how these are generally stable structures.

Last Remarks: The simulation would ideally need more work, the energy is not stabilising as I would hope for the periodic conditions. This is probably the result of some typo in the code that I have been unable to notice. Furthermore, the simulation can be improved by increasing the density of bins per unit square on the grid such that our potential can be more accurate per particle. Lots more work required, would love to know specifically where I went wrong as I would like to come back to this and fix it up over the break.