Amanda Di Nitto – 260689696
PHYS 512 – Project

Set Up:

In this problem, the potential of a single particle needs to be calculated by way of Green's theorem. For one particle, the Laplacian can be described via the following potential:

$$U = \frac{1}{4\pi r}$$

Where *r* is the distance between particles and not the radius. In order to avoid problems at *r = 0* a softening term was included (softening was kept at 0.1 most of the time).

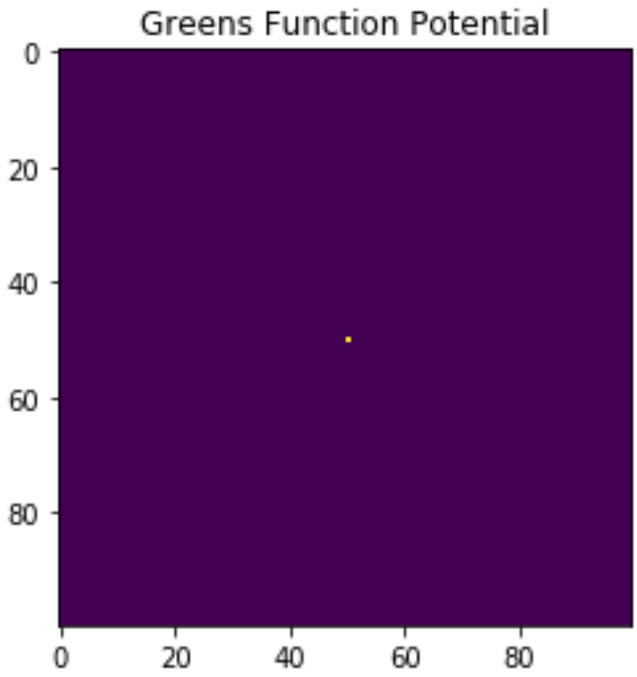

Figure 1: Plot of the potential calculated with Green's function

The next thing required was a density grid which houses all the particle locations. This grid is used to calculate the potential of all the particles. The grid was feed the input values of *x,y* and *z* and returned a matrix of specified size with the particles occupying specific bins based on their coordinates. The grid was created using numpy's histogramdd, which returns a grid based on the the points, the specific number of bins and the size of the grid. For simplicity, the grid was made to have bin spacing always equal to one; for an *NxNxN* matrix, there were *N* bins. With the density grid and the potential of a single particle, the overall potential could now be calculated using an inverse Fourier transform, after Fourier transforming the two matrices before-hand. The gradient of the potential was calculated for all three directions using numpy's gradient function. This provided the acceleration for the whole grid, but since we were only interested in specific particles, the bin locations of the particles were used to gather all the specific accelerations

required. These accelerations were then divided by the mass and this gave the resulting force for each direction.

Unfortunately, I did not manage to make my script work in three dimensions (3D), but it did work in two dimensions (2D) so the remaining write up will be able the 2D code. The 3D code will be posted as well in git hub.

Part A:

The particle was placed at the center of the grid and the leapfrog method was run for 20 iterations. The gif titled *Single Particle.gif* displays the fact that the particle did not move. Energy remained at 217.63 meaning the conservation of energy is being correctly calculated in the script.

Part B:

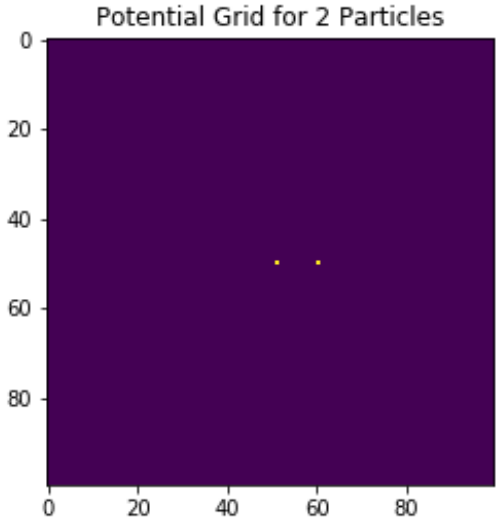

Figure 2: Total potential energy plotted for 2 particles

Two particles were placed in the middle of the plot about 10 spaces apart and evolved with time. The resulting orbit [Figure 3] is not a perfect circle (hinting that conservation of energy is not perfect) but they did orbit for over 800 iterations. The total energy did hover around 897 but there were changes at the decimal level which could account for the not as perfect circle. The Greens function was changed by removing the factor of 4π from r. This was purely scaling and was done so because the scale resulted in a larger grid being necessary and a very long computing time therefore to display the proper relationship the factor was removed, and the Greens function remained at $1/r$ for the remainder of the project.

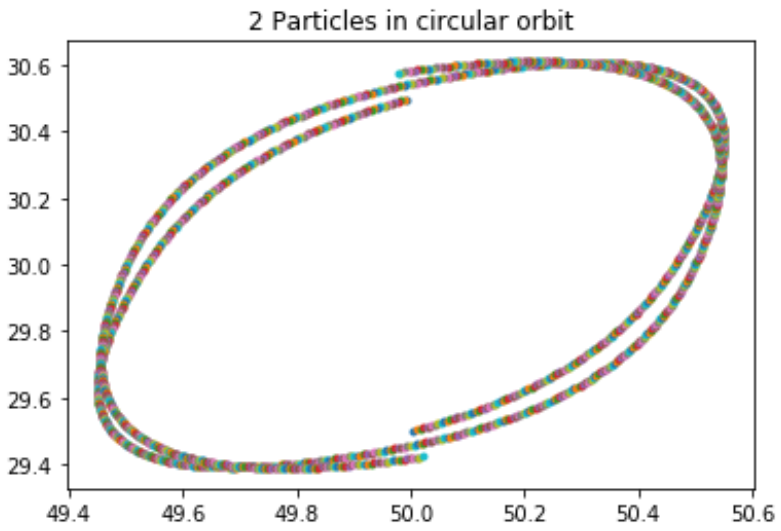

Figure 3: Circular orbit for 2 particles after 800 iterations.

Part C:

In order the run the code, it had to be set up with periodic boundary conditions. This is because, if my particle attempted to exit the grid an error would occur saying the indexing location given to find the particle in the force matrix is outside the bounds of the grid. Therefore, after every leapfrog step for the position, the position was divided by the modulo operator; this meant that the particle would be placed back inside the grid. The N-body was run for 3000 iterations with 100,000 particles

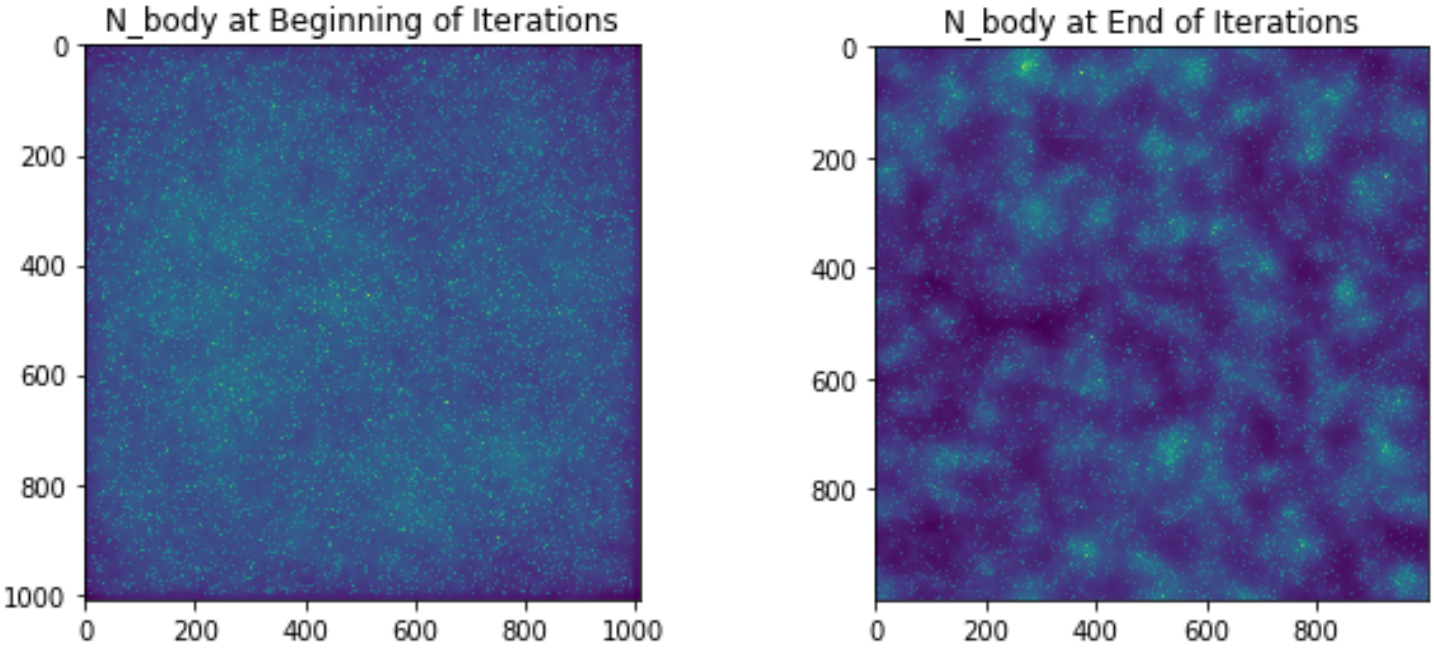

Figure 4: N_body simulation for periodic BC's at beginning and at end of 3000 iterations.

The energy conservation was okay with an initial value of 35315936 and a final of 354456908. Obviously there was a large increase, and it is hard to tell where the conservation of energy is going wrong. On a positive note, there were no large breaks in the conservation of energy, so although the grid appears to be gaining energy as a whole, it is small enough that there were no big breaks to be concerned about. In order to create periodic boundaries, the wrap around effect from the Fourier transforms had to be eliminated. To do this, the potential was calculated for a larger matrix than necessary, and then cut to the actual matrix size. By making it quite a bit bigger, the periodicity of the code should not be noticeable because all the points on the end are not going to be looked at by our leapfrog step meaning. Unfortunately, the non-periodic conditions did not end up working. I kept running into an indexing error for the velocity leapfrog step whenever the particles went beyond the limits of my smaller matrix. I tried solving it by using np.delete to take out the particles that were causing issues but it did not end up working. I also tried using a mask instead of just splicing the matrix down to what I wanted but that also did not fix my indexing problem. There was clearly an issue with the way I cut the matrix since the plot for the starting point seemed to display a square with more particles than the surrounding area.

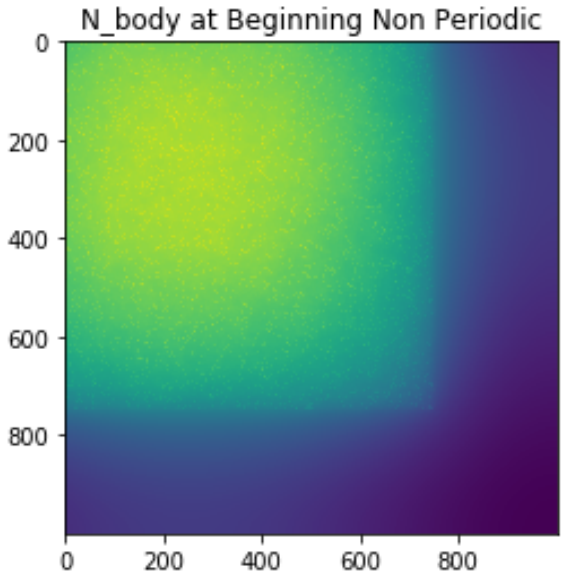

Figure 5: Incorrect plot for N-body Non-Periodic simulation

Part D:

The k^{-3} dependency was included by creating an array for the mass rather than one single value. The array was given Gaussian noise and then the inverse Fourier transform of the mass was taken. The mass noise can be seen in figure 6 below. The ends of the mass array were cut since the ends had large spikes in the values and it was not a uniform white noise spectrum. Since the array was the same length as the number of particles a random m was given to each particle meaning this should create a universe with a changing mass. The code was run with

periodic conditions and for 3000 iterations to be able to better compare to the uniform mass simulation.

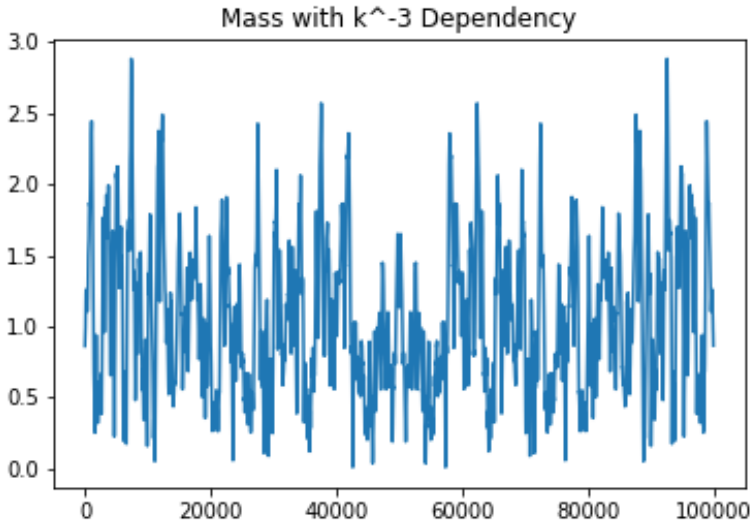

Figure 6: Gaussian noise plot for the mass with k^{-3} dependency

Looking at the end plot [Figure 7], it does seem like there are regions of denser particles than there was with the uniform mass. This would make sense since the universe's evolution displays similar grouping leading to formation of galaxies and such. The areas with less particle density do appear larger as well so all these factors do seem to point to the fact that the k^{-3} did improve our simulation. Interestingly enough the conservation of energy actually got worse with the mass dependency, starting at 353159480 and ending at 335893553. This is a much bigger gain than the uniform mass run. It could be that the change in mass value exacerbated the issue which is causing the lack of conservation of energy which would mean we are probably gaining energy during the leapfrog step. This is rather unexpected since the leapfrog scheme is supposed to inherently conserve energy but based on the difference it seems clear that at some point during either the step or the gradient calculation (since rho and greens do not change in magnitude) energy must be added.

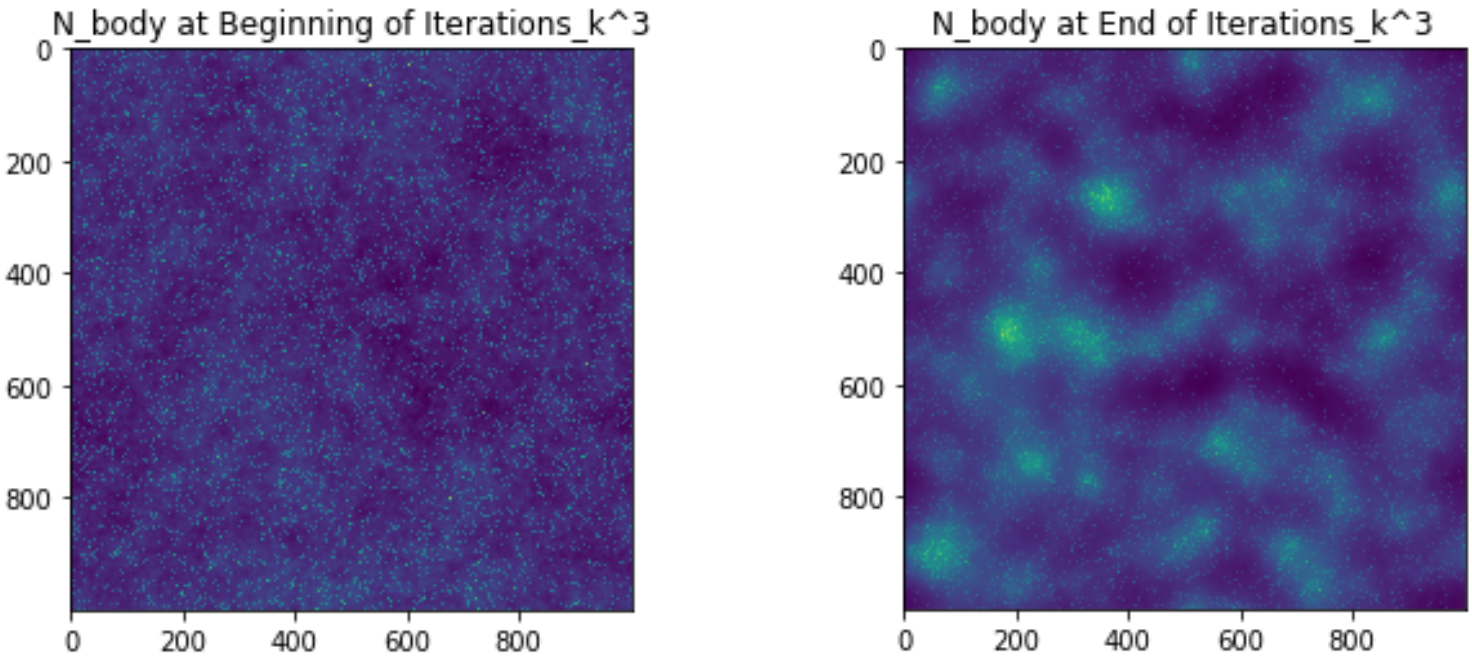

Figure 7: N-Body simulation with k^{-3} dependency for the mass with periodic boundary conditions