PROJECT 2 – COSMOLOGICAL SIMULATIONS

The goal of this project is to create, in a group, a realistic simulation of the universe on cosmological scales, starting from an early time (redshift $z = 30$) and stopping at the current time ($z = 0$). You should write a short paper describing your work, your results, and your conclusions, based on the tasks below. All code should be on GitHub in a suitably named directory.

A. A VERY BIG BOX

First, let's begin with the entire universe itself. Start with these calculations:

- Make a list of the currently accepted values for the following parameters: H_0, $\Omega_{m,0}$, $\Omega_{\Lambda,0}$, $\Omega_{0,\text{tot}}$, and any others you think you might need.
- calculate the critical density of the universe (today),

$$\rho_{\text{c},0} = \frac{3c^2}{8\pi G} H_0^2,$$

  and how much of that is matter, $\rho_{\text{m},0}$. Your final answer should be in units of $\text{M}_\odot/\text{Mpc}^3$.
- Calculate the current age of the universe t_0 ($a = 1$ or $z = 0$, billions of years) and the age of the universe at redshift $z = 30$ (in millions of years). To do this you'll need to rearrange the Friedmann equation for t to get

$$H_0 t = \int_0^a \left[\frac{\Omega_{\text{m},0}}{a} + \Omega_{\Lambda,0} a^2 + (1 - \Omega_{\text{m},0} - \Omega_{\Lambda,0}) \right]^{-1/2} da,$$

  and use the relationship between redshift z and scale factor a:

$$a = \frac{1}{1+z}.$$

- Next, consider a cubic section of the universe that today measures 50 Mpc by 50 Mpc by 50 Mpc. What is the total mass (of matter) inside this box?

B. PARTICLE-MESH CODE

This is the main part of the project and involves the most coding. I'll guide you through it, though. Create a Python class called `Mesh`. The class should contain the following methods:

1. `__init__`, which initializes the necessary variables for the class. This should include, at minimum, the length of the simulation box L, the number of grid points per dimension N_g, and the cosmological parameters $\Omega_{m,0}$, $\Omega_{\Lambda,0}$, and H_0. You can use reasonable defaults for these in the code if you want.
2. `f`, which takes as parameter the scale factor a and returns the function $f(a)$ (see notes for that function).
3. `G`, which takes as parameters the integers ℓ, n, and m, along with the scale factor a, and returns the Green's function $G_{\ell mn}$ (see notes for the function itself).
4. `calc_density`, which takes as a parameter a `System` object and returns a three-dimensional array δ_{ijk} representing the normalized density $\delta(\mathbf{x})$ in each cell.

5. `calc_potential`, which takes as parameter the density δ_{ijk} and the scale factor a and returns the three dimensional potential ϕ_{ijk} for each cell. (This will involve a complex FFT, so make sure you return only the real part.)

6. `calc_accels`, which takes as parameter a `System` object and the scale factor a and updates the acceleration of each particle based on the potential of each cell. It returns nothing. This function should call on its own the functions `calc_density` and `calc_potential`.

7. `move_particles`, which takes as parameters a `System` object, the scale factor a, and the "timestep" da, and updates the position and velocity of each particle based on the calculated accelerations. Be careful of periodic boundary conditions.

Given the above methods in the class, your main loop to evolve the PM system would look something like:

```
while a <= 1.0:
        accels = mesh.calc_accels(s, a)
        mesh.move_particles(s, accels, a, da)

        s.write(f"output_{a:5.3f}")
        a += da
```

C. Tests

In addition to the above code, you'll also need to write a program to generate initial conditions. Do so for two cases, using $N_p = 32$ particles per dimension:

1. A system of particles with positions evenly spaced, forming a three dimensional grid of points. Set the velocities to zero.

2. A system of particles with random positions within a cube of length L. Set the velocities to zero.

With the test initial conditions, test your code carefully. Make sure each function is doing what it should, and describe your testing procedure in your paper (e.g., how do you know your density assignment scheme is working properly?). When run fully, the grid-like initial conditions should stay exactly in place, and for the random case, you should see particles clumping together in random clusters due to gravity.

D. A (somewhat) Realistic Simulation

I'll provide a set of more realistic set of initial conditions, generated by the software MUSIC. Run your code with my initial conditions and a mesh size of $N_g = 64$, and try to answer the following questions:

1. What's the mass of the largest structures that form in your simulation? Although there are sophisticated ways of finding structures, I think in this you can try to estimate the size of a structure by counting the mass of the particles approximately within a certain region of your simulation.

2. What happens if you make the mesh smaller ($N_g = 32$) or larger (careful, it will take longer to run)?

3. What happens if you vary $\Omega_{m,0}$ or $\Omega_{\Lambda,0}$? If your universe is highly curved? The differences might be subtle unless the changes are large.

4. Finally, try the larger initial conditions I provide ($N_p = 64$) with a correspondingly larger mesh ($Ng = 128$). How does it look? Can you perform a similar analysis for structure mass that you did with the smaller simulation? Are there any major differences?

I'll leave it up to you as to how you'll display results and organize your short paper.