N-Body simulation using Particle-Mesh Method (NOISE)

Hitesh Kishore Das
(4th Year UG) 11-01-00-10-91-16-1-13491

Under supervision of
Prof. Prateek Sharma

Introduction

N-body simulations are widely used in many fields of physics like soft condensed matter and astrophysics. In many such simulations in astrophysics, the main force acting on the particles is the gravitational force. Some of the methods for such N-body simulations are:

- **Direct method**: Direct calculation of forces on each particle due to every other particle
- **Particle-Mesh method**: Calculation of forces from potential obtained by a method employing FFT, IFFT and density field.
- **Tree method**: Direct force calculation for nearby particles and approximations for the ones far away.

The Direct N-body method is computationally expensive for large number of particles. So, alternative methods are used to reduce this computational cost. This project is on implementing the **Particle-Mesh method** for N-body simulations. The main focus is on astrophysical circumstances, hence the code units used are of astrophysical scale and gravitational force is taken as the central force.

I structured the code similar to other simulation codes that I have used, like PLUTO code and LAMMPS. Following the tradition of giving peculiar names to Astrophysical surveys and codes, I named this set of code "**NOISE**".

The code is available on Github: `https://github.com/HiteshKishoreDas/NOISE`.

1 What's happening under the hood?

The structure of this code set is shown as a flowchart in Fig. 1.

1.1 Simulation Parameters (`config.py`)

For every simulation, one needs to define some parameters like box size, resolution, timestep size, etc. In this code, all such parameters are defined in `config.py`. Following are the different defined parameters and corresponding explanation:

- **`unit_mass`**: Mass set to 1.0 in code. Set to 1 solar mass in kilograms by default.
- **`unit_length`**: Length set to 1.0 in code. Set to 1 parsec in metres by default.
- **`unit_time`**: Time set to 1.0 in code. Set to 10^6 years in seconds by default.
- **`G`**: Value of universal gravitational constant in code units.
- **`x_start`**: Starting point of x-axis. `y_start` and `z_start` are set to be equal to `x_start`.
- **`x_end`**: Starting point of x-axis. `y_end` and `z_end` are set to be equal to `x_end`.
- **`stop_time`**: Time to stop the simulation in code units.
- **`step_length`**: Grid cell size in code units.
- **`step_time`**: Timestep size in code units.
- **`N_particles`**: Number of particles in the simulation.
- **`max_vmag0`**: Maximum magnitude of velocity in initial conditions.
- **`mass_assign_scheme`**: Assignment scheme to be used for mass assignment to grid cells.
  Options: "NGP"(Nearest Grid Point), "CIC"(Cloud-in-Cell).
- **`force_assign_scheme`**: Assigment scheme to be used for interpolating force.
  Options: "NGP"(Nearest Grid Point), "CIC"(Cloud-in-Cell).

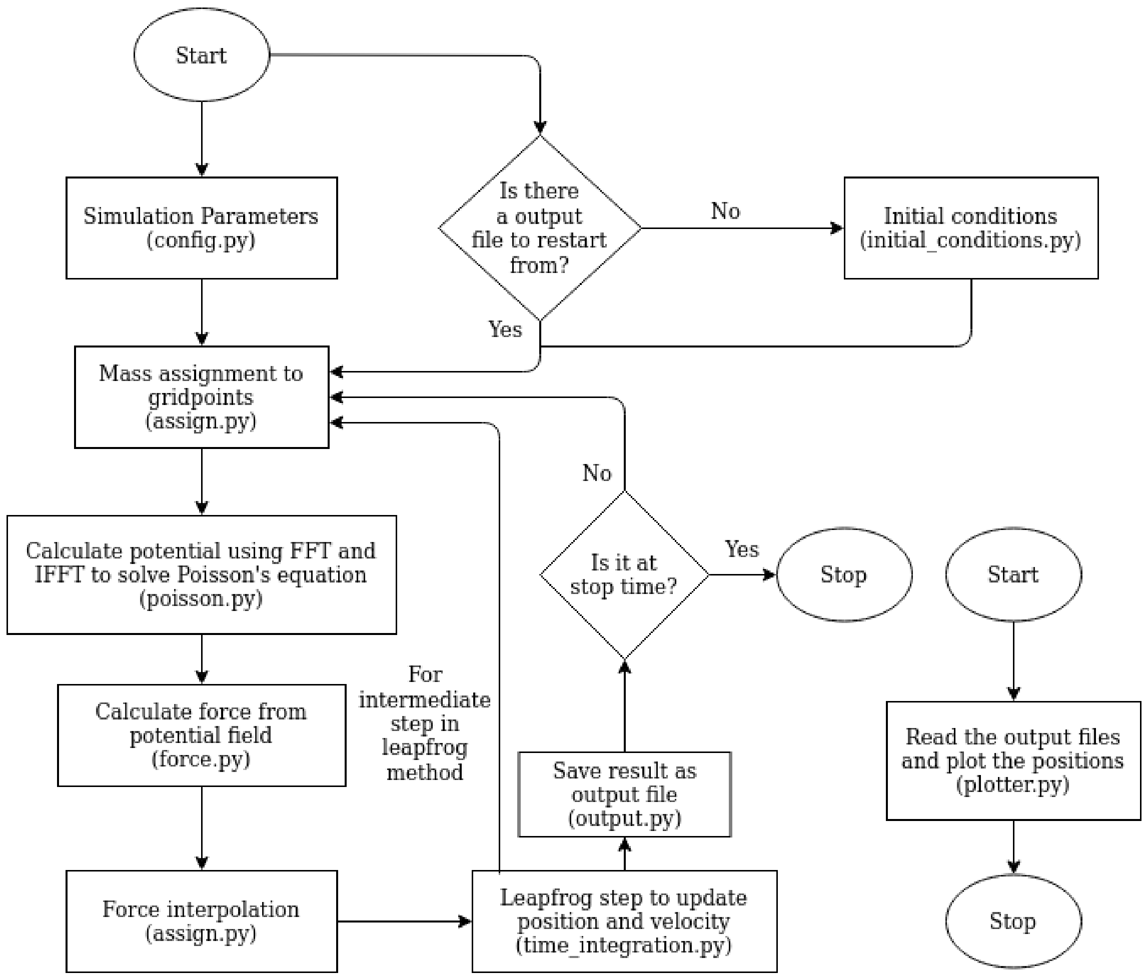
Start
Simulation Parameters (config.py)
Is there a output file to restart from?
No
Yes
Initial conditions (initial_conditions.py)
Mass assignment to gridpoints (assign.py)
Calculate potential using FFT and IFFT to solve Poisson's equation (poisson.py)
Calculate force from potential field (force.py)
For intermediate step in leapfrog method
Force interpolation (assign.py)
Leapfrog step to update position and velocity (time_integration.py)
Save result as output file (output.py)
Is it at stop time?
No
Yes
Stop
Start
Read the output files and plot the positions (plotter.py)
Stop

Figure 1: Flowchart diagram showing general structure of the code.

- **`time_integration_scheme`**: Time integration method to be used for getting the positions and velocities of particles in next timestep. Options: "LEAPFROG".
- **`parent_dir`**: Directory where the "output" directory and "Plots" directory will be created.
- **`output_dt`**: Time interval (in code units) between outputs into "output" directory.
- **`h`**: Dimensionless Hubble constant
- **`t_0`**: Calculated age of universe (used to calculate scale factor)
- **`a_p`**: Power in relation for scale factor. Is equal to 2./3. for matter-dominated universe and 1./2. for radiation-dominated universe. Set to 0. for no expansion.

It contains following function:

- **`print_config()`**: Run this module as a program to get a list of defined simulation parameters

This module also contains a test block which is executed if the module is run individually. The test block calls the `print_config()` function.

1.2 Initial conditions (`initial_condition.py`)

This module contains the function for defining the initial condition for the simulation. The particle properties are defined as a 2D numpy array (called `particle` in the code). Each row contains all the properties of a given particle. Starting from index 0, the columns in the array are- mass, x-coordinate, y-coordinate, z-coordinate, x-velocity, y-velocity, z-velocity, x-force, y-force and z-force. This module won't be executed if a restart is being used with `run.py`.

Functions in the module:

- **`initial()`**: Function for defining the initial conditions. It takes no input and returns a 2D array with the particle properties.
- There may be additional user-defined functions that are used in `intial()`, depending simulation setup.

The test block shows the mass distribution, position scatter and velocity scatter of initial condition.

1.3 Mass assignment (`assign.py`)

The mass of the particles need to be assigned to the grid cells before the calculation of potential. There are multiple schemes available for this task:

- **Nearest Grid Point**: Mass of the particle is assigned to the nearest grid cell.
- **Cloud-in-Cell**: Mass fraction assigned to a grid cell is depends on the intersected volume between a grid cell sized cube around the particle and the grid cell. A 2D version of this scheme is shown in Fig. 2.

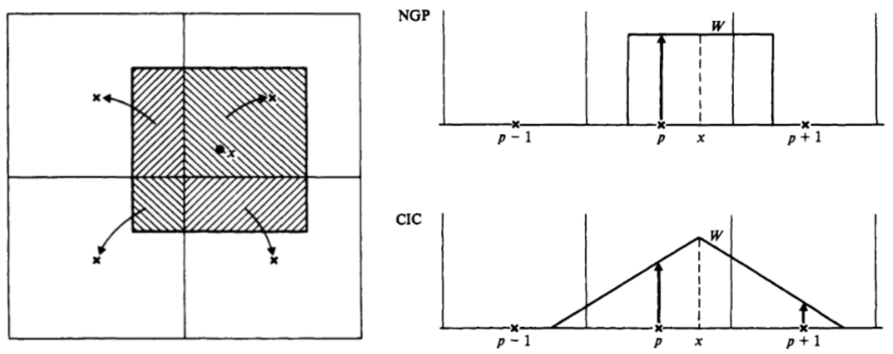

Figure 2: **(Left)** 2D version of the CIC scheme. **(Right)** Plot of assignment functions for NGP and CIC. *Source: Computer simulation using particles, Hockney and Eastwood*

These schemes are implemented using assignment functions, $W(x)$. The value of mass assigned to the grid cells depends on the distance of particle from the grid cell centre, given as x. The assignment functions are shown in Fig. 2 Related functions in this module:

- **`top_hat()`**: Top-hat function for assignment function. It takes an array as input and returns corresponding function value
- **`triangle()`**: Triangle function for assignment function. It takes an array as input and returns corresponding function values as an array.
- **`W_NGP()`**: Assignment function for Nearest grid-point assignment scheme.
- **`W_CIC()`**: Assignment function for Cloud-in-cell assignment scheme.
- **`assign_scheme ()`**: Returns the required assignment function and extent of influence, according to the simulation parameters in `config.py`. It returns the assignment function for both mass assignment and force assignment. Force assignment function is later used in `force_assign()` function.
- **`mass_assign()`**: Function for mass assignment to grid cells. It takes particle and grid properties as input and returns a density field array.

Function relevant to force interpolation will be mentioned in Force interpolation subsection (1.6). The test block in this module shows the time taken for mass and force assignment in a timestep.

1.4 Solving Poisson's equation (`poisson.py`)

Completing the mass assignment to gridpoints, gives a density field which can now be used to calculate the potential field using Fast Fourier Transform (FFT) and Inverse FFT (IFFT).

$$\Phi(\mathbf{r}) = -G \iiint \frac{\rho(\mathbf{r}')}{\|\mathbf{r}-\mathbf{r}'\|} d^3\mathbf{r}'$$

$$\implies \widetilde{\Phi}(\mathbf{k}) = -G\widetilde{\rho}(\mathbf{k}) \left[\widetilde{\frac{1}{r}}\right](\mathbf{k}) \tag{1}$$

Due to lower time complexity of FFT and IFFT ($O(N \log N)$), it is faster for larger number of particles than Direct N-body simulation. The density field from `mass_assign()` in `assign.py` is used for the calculation in Eq. (1). Functions in this module:

- **`onebyr()`**:Function to calculate 1/r for Eq. (1). It takes grid cell centers as input and returns 1/r array.
- **`poisson()`**: Function for calculation of $\Phi(\mathbf{r})$ by obtaining $\widetilde{\Phi}(\mathbf{k})$ using Eq. (1) and taking an IFFT. It takes density field array from `mass_assign()` and grid cells properties to return potential field.

The test block returns the time taken for solving Poisson's equation, projected potential field and projected particle distribution.

1.5 Force calculation `force.py`

$$F_i = \frac{\rho\left(\Phi_{i-1} - \Phi_{i+1}\right)}{2\Delta_i} \tag{2}$$

This module uses finite difference approximation for calculating force from potential field using Eq. (2) in all three directions. This module also contains a function that combines all the steps into one function.

The functions contained in this module are:

- **`force()`**: Function to calculate forces in three directions using finite difference approximation as shown in Eq. (2). It takes potential and density field to return three force components.
- **`particle_to_force()`**: Function to combine the steps of mass assignment, solving Poisson's equation, force calculation and force interpolation (explained later) into a single function. It takes particle and grid properties to return particle properties with force values for each particle.

The test block is the time taken for force calculation and plots all three projected force components distribution.

1.6 Force interpolation (`assign.py`)

This module interpolates the forces obtained on grid cell positions from `force()` to the position of particles using the same schemes as mentioned in section 1.3 about Mass assignment. Force on each particle is calculated as the sum of forces in grid cells, weighted with assignment function as a function of distance from the particle. Function `energy_cons()` in `conservation.py` is used to maintain energy conservation after force interpolation.

Related function in `assign.py`:

- **`force_assign()`**: Function for interpolating forces to particle positions. It returns particle properties with the corresponding forces included.

The test block already explained in section 1.3 about Mass assignment.

1.7 Time integration (`time_integration.py`)

Leapfrog method is used for time integration in this code. Other methods like 4th-order Runge-Kutta method can be added via an additional function.

Functions in the module:

- **`scale_factor()`**: Function to calculate scale factor for any given timestep.
- **`leapfrog.py`**: Function to use leapfrog method to calculate position and velocities at different timesteps. The forces at each timestep are calculated using `particle_to_force()` in `force.py`. It takes starting particle and grid properties. It writes an output file at time intervals according to simulation parameters in `config.py` using functions in `output.py` (explained later).
- **`time_int_func()`**: Function that returns the appropriate time intergration method function according to the simulation parameters in `config.py`.

The test block shows the time taken for one time integration step.

1.8 Output (`output.py`)

This module contains the functions for saving output from the time integration steps for later processing. The functions in the module are:

- **`output()`**: Function to save particle properties 2D array as ".npy" files in a "output" directory inside the parent directory as defined in `config.py`. If the "output" directory does not exist, it creates the directory.
- **`output_info()`**: Function to save info files which contains time and timestep size for each output in a dictionary format as ".npy" format in the same "output" directory.

1.9 `main.py` and `run.py`

`main.py` is the module that brings the whole simulation process into a single function, `main_func()`, which can be called by other programs.

`run.py` is the python file that is run in the terminal. It has a -restart flag, which enables the user to restart the simulation from a existing output file in the "output" folder in the parent directory. This file calls the `main_func()` function defined in `main.py` module.

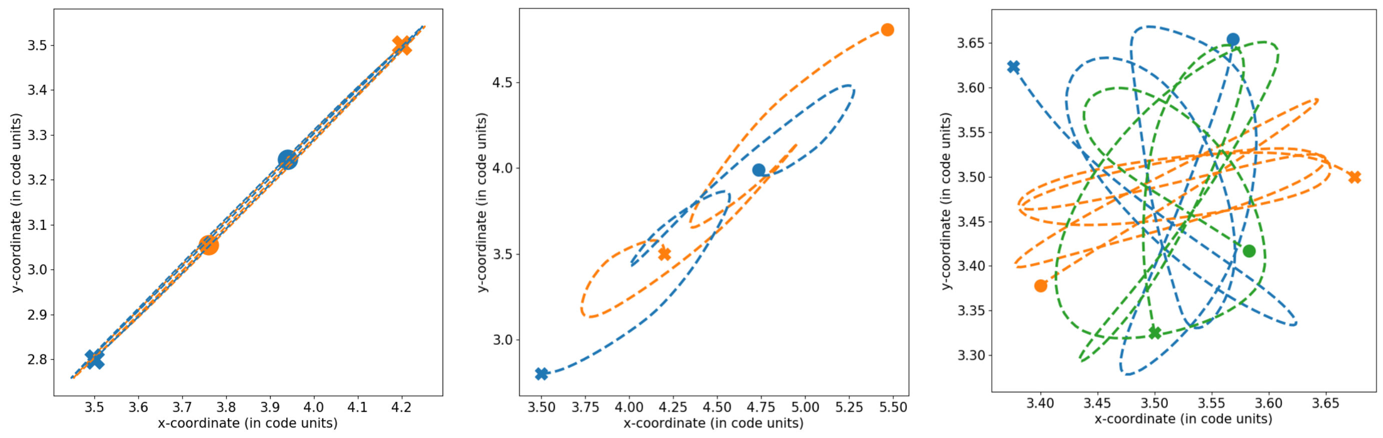

Figure 3: **(Left)** 2-body simulation with no initial velocities. **(Middle)** 2-body simulation with initial velocities. **Right** 3-body simulation with no initial velocities. **Note**: Solid cross - Starting position of particles, Solid circle - Last position

2 Does it work?

2.1 2-body and 3-body simulations

2-body simulations with and without any initial velocities is run to check the outputs. A 3-body simulation is also run without any initial velocities to check if the trajectories are chaotic as expected. Note that, Direct N-body method is better for 2-body and 3-body simulations as very close interactions are not resolved accurately in particle-mesh simulations. The results don't show any peculiar behaviour and are as expected.
Fig. 3 (Left) shows that the two particles just move almost on the line joining them, as expected. Fig. 3 (Middle) shows how the orbits change as the center of mass is moving. Fig. 3 (Left) shows the chaotic trajectories for particles, as expected.

2.2 Large scale structure formation

At very large length scales, web-like structures are observed. In the last row of Fig. 4, the left picture shows the observed distribution of galaxies near Coma cluster, given in [2] and such large web-like structures can be easily seen in this image.

To simulate these large-scale structures, I start with a uniform distribution of particles in the box. To include expansion of space, the box is kept fixed, but the forces and velocities are scaled accordingly using the calculated scale factor. There are two simulations with expansion, constituting of different number of particles. The results are very similar with minor differences between the simulations. Snapshots of particle positions projected in z-directions are shown in first two rows of Fig. 4. We can see the formation of large-scale structures in these snapshots.

The results also look very similar to the simulation results from [2] and [1] shown in middle and right image of last row in Fig. 4. The web-like structures in these simulations are present snapshots in last column of upper two rows. The presence of more cavities in simulation snapshots from [2] is due to the fact that the galaxies are magnitude-limited, that is, only galaxies above a specific magnitude has been shown.
For more quantitative comparison, I compare the two-point correlation function given in [2] with the two sets of simulations shown in Fig. 4. The two-point correlation function has been calculated for the last snapshot in first three rows of Fig. 4. We see that the shape of two point correlation function matches the one given in [2]. The minor differences may be due to less particles than that used in literature ($\sim 10^6$). The disturbances in two point correlation function at small values of r decrease with increase in number of particle.

3 Conclusion

In this project the N-body simulation is implemented using the Particle-Mesh method. In this report, I have explained the details about the process of different calculations made in this implementation. I have also tested this implementation through different test cases, namely, 2-body, 3-body and structure formation simulations. In all the test cases, the results from this implementation matches the expected results and results given in literature ([2],[1]). There are a lot improvements that can be done over the current implementation, like other types of time integration methods, more kinds of boundary conditions, adding provision for parallel computing, adaptive resolution and although ambitious, maybe general relativistic simulation with weak-field approximation using this particle-mesh framework.

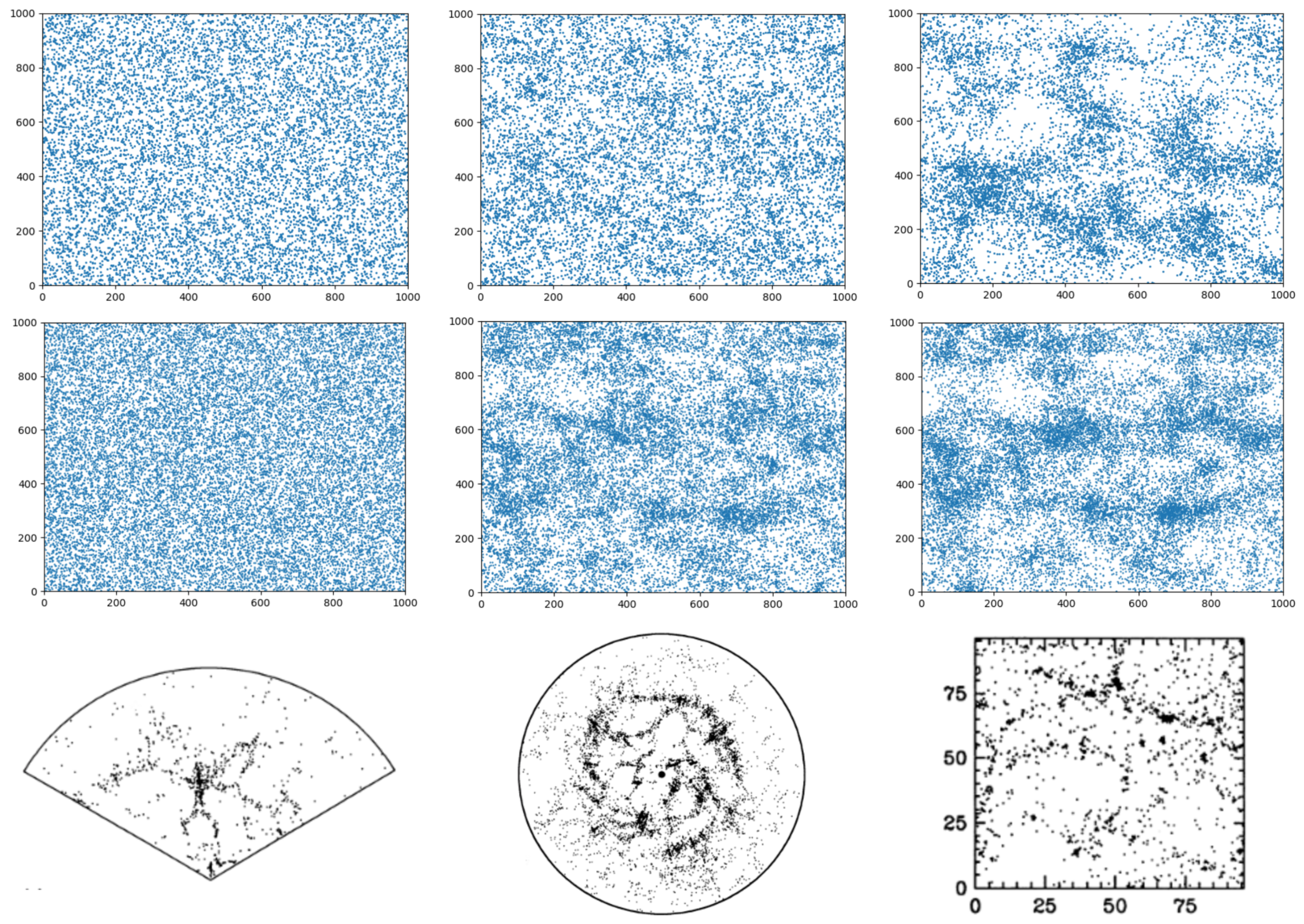

Figure 4: **(Top row)** Snapshots of simulation with expansion and 10000 particles **(Middle row)** Simulation with expansion and 20000 particles **(Bottom row, Left)** Observed magnitude-limited galaxy distribution near Coma cluster [2]. **(Bottom row, Middle)** Simulations for large scale structure formation in [2]. **(Bottom row, Right)** Simulations for large scale structure formation in [1]. **Note**: $h = 0.5$ for my simulations and those in [2].

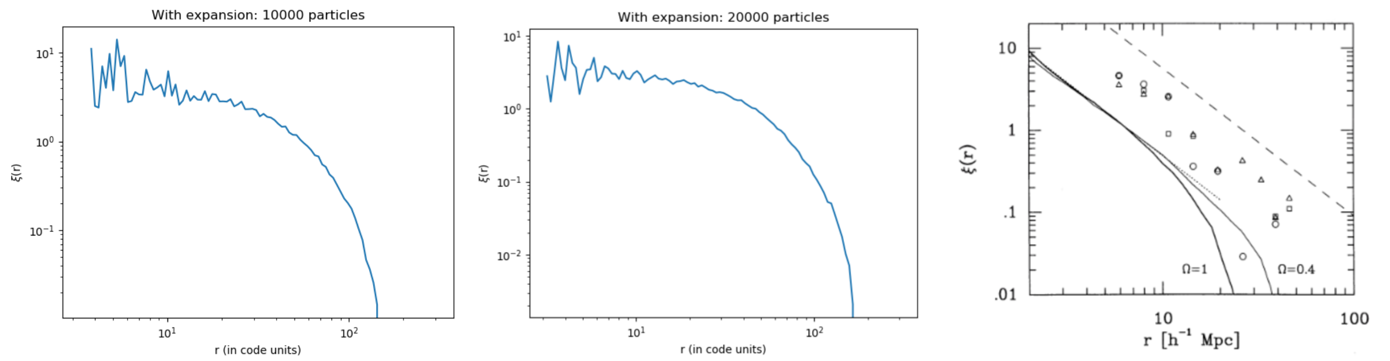

Figure 5: **(Left)** Two point correlation function (TPCF) for simulation with expansion and 10000 particles. **(Middle)** TPCF for simulation with expansion and 20000 particles. **(Right)** TPCF from [2].

Again, the code is available on GitHub `https://github.com/HiteshKishoreDas/NOISE`. I hope to add the above mentioned improvements in future.

References

[1] Arif Babul et al. "Testing the gravitational instability hypothesis?" In: *The Astrophysical Journal* 427 (May 1994), p. 1. ISSN: 1538-4357. DOI: `10.1086/174119`. URL: `http://dx.doi.org/10.1086/174119`.

[2] Changbom Park. "Large N-body simulations of a universe dominated by cold dark matter". In: 242 (Feb. 1990), 59P–61P. DOI: `10.1093/mnras/242.1.59P`.