Objective:

Simulation of Multi-Body Evolution due to Gravity/Electrostatic Force given Mass, Charge, Position, and Velocity

How to Use Package:

For N_Body.py -

- **Input:** CSV File named "N_Body" with data columns - 'Mass, Charge, Position_X, Position_Y, Position_Z, Velocity_X, Velocity_Y and Velocity_Z' (the CSV file shouldn't have any header row)
- **Output:** 'n' CSV Files with data columns - 'Time(in seconds), Position_X, Position_Y and Position_Z'

Arguments:

- time_step (int): Steps size of discretising time (lower⇒more accurate): Force considered constant during this period and Output given at each time step.
- t_final (int): Time till which system is made to evolve
- Engine (GravitySim or ElectroStaticSim): Specify whether to simulate Gravity or Electrostatic force, taking the other to be negligible [either GravitySim() or ElectroStaticSim()]
- filepath (string): Path of input CSV

Example execution from command-line:

```
python ./N_Body.py --time_step 10000 --t_final 5000000 --engine GravitySim --input .
```

For Animate.py –

Code to animate and export video of the evolution of the system.

Arguments:

- n (int): Number of Particles
- filepath (string): Path of folder containing input CSVs
- output (string): Path of folder for output Video
- in_step (int): Adjust resolution of animation. Skip (in_step-1) many lines from CSV between each input line. 0 is for no skips and higher in_step gives faster computation
- interval (int): Interval in msec between each frame for animation
- frames (int): Number of frames in output video

Example execution from command-line:

```
python ./Animate.py --n 9 --input ./Output --output . --in_step 10 --interval 100 --frames 50
```

For TimePeriods.py –

Code to find the time period of revolution of every particle.

Arguments:

- n (int): Number of Particles
- filepath (string): Path of folder containing input CSV

Example execution from command-line:

```
python ./TimePeriods.py --n 9 --input ./Output
```

For Eccentricity.py –

Code to find the eccentricity of orbit of every particle.

Arguments:

- n (int): Number of Particles
- filepath (string): Path of folder containing input CSV

Example execution from command-line:

```
python ./Eccentricity.py --n 9 --input ./Output
```

For ExecutionTime.py –

Code that takes both a function and its arguments as its own arguments and returns the result of the function along with the time required to execute that function in seconds.

Arguments:

- func (function): Function whose exececution time is to be found
- *args (*args): Arguments of the required function in order

Returns:

- float: Execution Time of function

Description of Solution:

We tackle this problem by discretising time and taking the force to be constant for a ‘time_step’ time. At every instant, for all particles, we find the net force on that particle due to all other particles and update its position and velocity due to this net force. Letting this evolve gives us our required simulation.

We first define a ‘Vector’ class followed by a ‘Particle’ class with parameters – Mass, Charge, Position (Vector), and Velocity (Vector). We define functions to read and write these variables and operator overloading functions on the class.

Then we define a Simulation Engine class consisting of a Force vector and two virtual functions – force() and run().

Two new Engine classes inherit this class – Gravity Simulator and Electrostatic Force Simulator, each of which have a different force function.

- **force()** – Takes two particles as arguments and calculates the force between them
- **run()** – Takes a particle and time_step as arguments and updates the position and velocity of the particle due to the Net force acting on it

We run three nested ‘for’ loops. The innermost loop is to calculate the force on one particle due to all other particles using the force() function. The middle loop uses the net force on a particle to update its position and velocity using the run() function and does this for all particles. The outermost loop is to go through the time instances, making the system evolve.

Tests Performed:

Solar System Simulation –

The data on the position and velocity of planets of the solar system were obtained from ‘https://ssd.jpl.nasa.gov/horizons/app.html#/’ and used to simulate the evolution of the system. The simulation was performed for a time_step (time discretisation) of 1000 sec and for a t_final (length of simulation) of 5 years.

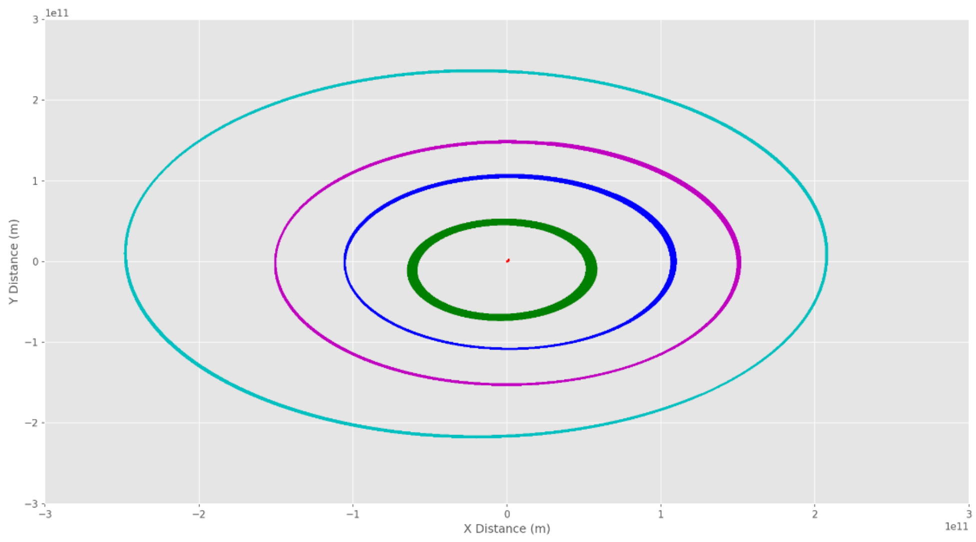
Execution Time for Simulation = 2454.72 sec

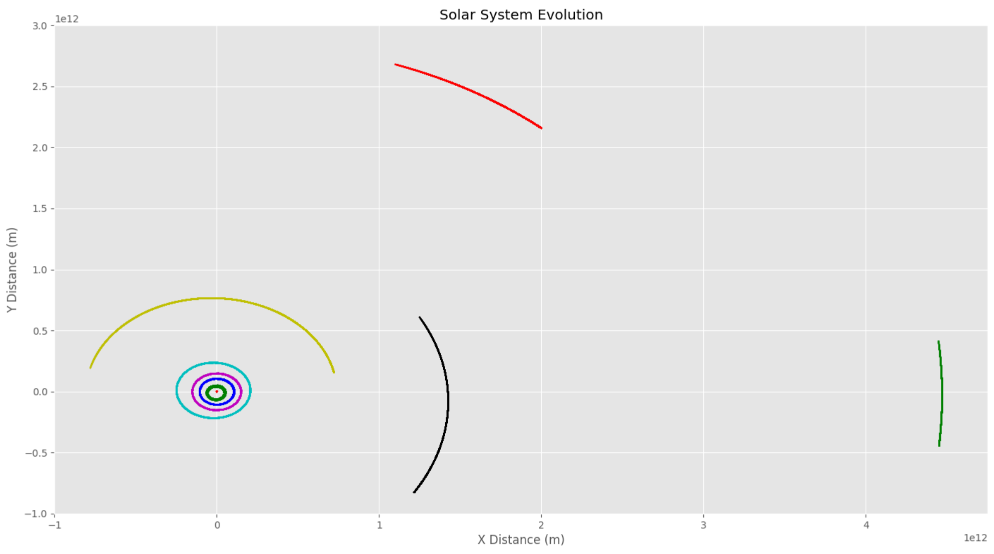
These particle evolutions were animated, and it was observed to give the expected orbits of the solar system.

Execution Time of animation code for,

Inner Planets = 11.08 sec

Solar System = 16.04 sec





The time periods of revolution and eccentricities of the orbits were found to correlate with the known periods and eccentricities of the orbit of the planets.

Time Periods from Simulation:

Mercury – 91.7 Days

Venus – 220.9 Days

Earth – 368.7 Days

Mars – 687.2 Days

Execution Time = 1.55 sec

Known Time Periods (https://spaceplace.nasa.gov/years-on-other-planets/en/):

Mercury: 88 days

Venus: 225 days

Earth: 365 days

Mars: 687 days

Eccentricities from Simulation:

Mercury – 0.1905

Venus – 0.0197

Earth – 0.0192

Mars – 0.0978

Execution Time = 1.75 sec

Known Eccentricities of Planets (https://nssdc.gsfc.nasa.gov/planetary/factsheet/):

Mercury: 0.206

Venus: 0.007

Earth: 0.017

Mars: 0.094

To reproduce these tests, obtain the data of position and velocity of planets of the solar system at the time ‘01-01-2023 00:00’ from ‘https://ssd.jpl.nasa.gov/horizons/app.html#/’ and use it to simulate the evolution of the system and get the output CSV file.

Then the tests can be replicated using the Animate.py, TimePeriods.py, and Eccentricity.py codes on the output CSV file.

Future Work:

A limitation of this simulation was that the accurate simulation of complete orbits of outer planets takes too long to compute. This is why we couldn’t find their periods or eccentricity.

We will make the computation faster using multiprocessing and GPU coding.