Overview:

This program is at its core an N Body simulation, used to model the orbits of the Solar System’s planets, Pluto, Earth’s Moon and Halley’s comet. This program utilises the input data from files of the initial position, velocity and mass to calculate the orbital motion around the Sun using Newton’s Gravitational Force of Attraction and the Velocity Verlet method.

Note: In this document (and in the program) the objects being simulated are referred to as planets for the sake of simplicity, even though some are non-planet orbital bodies (Halley’s comet etc.).

This document gives a description of the input and output of the program, detailing the module structure and the methods implemented within them, and outlines the main code and how the program meets the task at hand.

Velocity Verlet:

The motion of the planets is simulated numerically using the velocity Verlet algorithm, which follows these steps:

1. The position of the planets is updated, based on a second order expansion:
2. The force on the planets is calculated at this new position.
3. The velocity is updated using the average of the new force and its previous value:

This occurs repeatedly for small timesteps dt, and so maps out the trajectory of the planets over time.

Module Structure:

planetsim.py

particle3D.py

NumPy

The program has a main module planetsim.py, a helper module particle3D.py, and uses the standard module NumPy.

Inputs and Outputs

The input data for the: initial position, initial velocity, and mass for each orbiting object was retrieved from the NASA/JPLS/HORIZONS system using the following settings:

* Ephemeris type: Vectors
* Coordinate origin: Solar System Barycentre
* Units: km, km/s, kg

The program reads in a file containing this data, along with a file containing the simulation parameters, which includes the time step, the initial time and the length of the simulation.

The position data is output in a VMD-compliant format for visualisation. There is another output file containing the apsides (extreme orbital points) for all bodies. (For most bodies this concerns the maximal/minimal distance from the Sun, but for the Moon it concerns the maximal/minimal distance from the Earth.)

|  |  |  |
| --- | --- | --- |
| **File** | **Type** | **Contents** |
| Bodies (.txt) | Input | Label (str), Position (3xfloat), Velocity (3xfloat), Mass (float) |
| Simulation Parameters (.txt) | Input | Time Step (float), Initial Time (float), Length of simulation (float) |
| Trajectory (.xyz) | Output | The positions over the course of the simulation. |
| Apsides (.txt) | Output | The extreme points for the orbits. |

Module Descriptions:

**particle3D CLASS**

Each instance of this class represents a particle with the following properties:

|  |  |  |
| --- | --- | --- |
| **Property** | **Type** | **Notes** |
| Label | string | Particle Name |
| Mass | float | Particle Mass |
| Position | numpy.array [float] | 3-element array of position |
| Velocity | numpy.array [float] | 3-element array of velocity |

Initialisation:

Particle3D(label, position, velocity, mass), creates a particle with those parameters.

**Methods:**

* **\_str()\_**

*Returns a string containing the particle position (and name) that can be used by VMD to model the system.*

* **kinetic\_energy()**

*Returns the particle’s kinetic energy as a float using:*

* **leap\_velocity(float dt, numpy.array[float] force)**

*Updates the velocity by first order methods, using equation:*

* **leap\_pos2nd(float dt, numpy.array[float] force)**

*Updates the position by second order methods, using the equation:*

* **from\_file(fname)**

*Static method to create particles from a file. Files must be in the format:*

# <label> <x-pos> <y-pos> <z-pos> <x-vel> <y-vel> <z-vel> <mass>

**planetsim CLASS**

An instance of this class is a N-body simulation with a list of planets and parameters.

|  |  |  |
| --- | --- | --- |
| **Property** | **Type** | **Notes** |
| planet\_list | List | A list of instances of Particle3D |
| dt | float | Timestep |
| t0 | float | Initial t |
| sim\_time | float | Simulation length |
| forces | list of numpy.array[float] | 3D vector force for each planet. |

Initialisation:

PlanetSim(planet\_list, dt, sim\_time, t0)

Creates a simulation instance using the planet and the given input parameters.

**Methods:**

* **force(Particle3D p0, Particle3D p1)**

*Takes two planets and returns a NumPy array describing the force between them as a consequence of Newton’s Gravitational Force of Attraction:   
where , are the masses of* **p0, p1,**  *is the vector between them and G is the gravitational constant.*

* **potential(Particle3D p0, Particle3D p1)**

*Takes two planets and returns the gravitational potential:*

* **update\_forces()**

*Loops through every pair of planets and updates the force respective of their current position.*

* **total\_kinetic()**

*Loops through every planet and calls their kinetic\_energy() function, returning the sum of the results.*

* **total\_potential()**

*Loops through every pair of planets and calls the potential() function, returning the sum of the results.*

* **run(output file)**

*Uses the velocity Verlet method and the parameters given to simulate the planets’ orbits. It saves the trajectory to the output file for later use in the VMD display (example.xyz). It also calculates the apsides and writes them to a file apsides\_example.txt, where ‘example’ is the same name as the trajectory file.*

**main():**

Due to the modular design of the program, the main() function has to do very little:

* When the program is called on the command line it is passed the names of three files: the particle file, the parameter file and the trajectory output file.
* A list of planets is created by using the particle3D.from\_file() method on the first file, and simulation parameters are taken from the second.
* A planetSim() instance is created with the planet\_list and parameters.
* The run() function is called on the instance with the name of the trajectory output file as an argument.