Pegasus

Óscar Castro Remesal, Raúl Penagos Solórzano, Gustavo Gancedo Crespo

Pegasus is a Python-based astronomical simulation tool that models the motion of celestial bodies in the solar system using numerical integration methods. It can simulate planetary orbits, calculate eclipses, determine moon phases, and verify Kepler's laws.

Features

- Simulates motion of multiple celestial bodies using Velocity Verlet integration
- Calculates and detects solar and lunar eclipses
- Determines moon phases
- Verifies Kepler's laws
- Generates animations of the simulated system
- Saves position and state data for analysis
- Configurable simulation parameters
- Configurable initial astro parameters
- Energy and angular momentum conservation tracking

Installation

The only required dependencies are numpy and matplotlib. They can be installed using pip:

```
pip install numpy matplotlib
```

Usage

Configuration Files

The simulation requires two configuration files:

1. `initial_conditions.ini` - Defines the celestial bodies and their initial states:

```
# name, mass, x, y, z, vx, vy, vz, [fixed]
sun, 1.98847E30, 0, 0, 0, 0, 0, 0, true
earth, 5.97219E24, 1.496e11, 0, 0, 0, 29783.9, 0
```

It is important to notice that the program asumes all data introduced is in the International System of Units (SI):

- Position in meters
- Velocity in m/s
- Mass in kg

The `fixed` parameter is optional and indicates whether that astro will be fixed during the simulation or not.

2. `simulation_parameters.conf` - Controls simulation settings:

```
# Time step in seconds
Delta_time          3600
# Total simulation steps
Number_steps        24*345
# Steps between data saves
Interval_data_save  10
# Steps between animation frames. If zero or not especified, animation won't be created
animation_step      100
# Astros to check for eclipses, moon phases and Kepler.
star sun
satellite moon, 1737500
planet earth, 6378000
# Whether graphs are displayed after the simulation. The will be saved anyway.
show_plots true
```

The configuration files support two key features: line-based comments that can be added using the `#` symbol, and the ability to perform simple mathematical operations like addition or multiplication within parameter values.

Running the Simulation

Run the main script:

```
python main.py
```

Output

The simulation generates several outputs in the `output_data` directory:

- Data files for each celestial body (positions, velocities, force, ...)
- Positions along time of all the astros
- Energy and angular momentum conservation plots
- Kepler's law verification plots
- Moon phase visualization
- Center of mass movement
- Animation of the system (optional)

Code Structure

- `main.py` - Main simulation driver
- `astros.py` - Classes for astronomical bodies and systems
- `verlet.py` - Velocity Verlet integration implementation
- `file_io.py` - Configuration file handling
- `animation.py` - Visualization and animation
- `eclipse_search.py` - Eclipse detection
- `moon_phase.py` - Moon phase calculation
- `kepler.py` - Kepler's laws verification
- `save_state.py` - Data output handling

`main.py`

The core orchestrator of the simulation that:

1. Reads configuration files
2. Initializes the system
3. Runs the time evolution simulation
4. Handles data collection and output generation

Key functions:

- `main()` : Primary execution function that coordinates the entire simulation flow

`astros.py`

Contains the fundamental classes for representing astronomical objects.

Astro class

Represents individual celestial bodies with properties:

- Position
- Velocity
- Mass
- Forces
- Potential energy
- Name

AstroList class

Manages collections of astronomical objects and their interactions:

- Handles both fixed and free-moving bodies
- Calculates gravitational interactions
- Updates system state
- Tracks total energy and momentum

`verlet.py`

Implements the Velocity Verlet integration algorithm.

Verlet class

- `__init__(stepsize)` : Initializes integrator with given time step
- `advance_time(astrolist, t_final)` : Propagates system state using the Velocity Verlet method:

```
 r += v*dt + a*dt**2/2
v += a/2 * dt
a = calculate_acceleration()
v += a/2 * dt
```

`file_io.py`

Handles configuration file reading and parsing.

ConfigParams class

Stores simulation parameters:

- Time step size
- Number of steps
- Output intervals
- Animation settings

Key functions:

- `load_initial_condition(file)` : Reads astronomical body definitions
- `load_configuration(file)` : Reads simulation parameters

`animation.py`

Creates visual representations of the simulation.

Key functions:

- `create_animation(lists, filename)` :
  - Creates 3D visualization of celestial bodies
  - Generates animated GIF of system evolution
  - Handles color coding and labeling

`eclipse_search.py`

Implements eclipse detection algorithms.

Key functions:

- `eclipse_check(astrolist, last_time_eclipse)` :
  - Calculates geometric conditions for eclipses
  - Distinguishes between solar and lunar eclipses
  - Outputs eclipse timing and type
- `seconds_to_years()` : Converts simulation time to human-readable format

`moon_phase.py`

Calculates and visualizes moon phases. Cropped picture of the Moon from the original: Luc Viatour / https://Lucnix.be https://creativecommons.org/licenses/by-sa/3.0/

Key functions:

- `get_moon_phase(system, filename)` :
  - Determines moon phase from Earth-Moon-Sun geometry
  - Creates visual representation of current phase
  - Labels phase (New Moon, First Quarter, etc.)

`kepler.py`

Verifies Kepler's laws through simulation data.

Key functions:

- `new_year()` : Detects orbital periods
- `kepler()` :
  - Analyzes orbital data
  - Creates plots to verify Kepler's third law
  - Calculates orbital parameters

`save_state.py`

Manages data output and storage.

Key functions:

- `save_positions(list_astros)` :
  - Creates organized data files for each body
  - Stores position, velocity, and force data
  - Maintains simulation state history

License

MIT License - see `LICENSE` for details.

Authors and Responsabilities

All the members of the team have contributed to the development of the project in all its parts, but each one has focused on some specific parts of the code. The main responsabilities are:

- Óscar Castro Remesal (main, readme, savestate, file_io, save_state, animation)
- Raúl Penagos Solórzano (eclipse_search, moon_phases, save_state, data_tables_reader, inital_conditions)
- Gustavo Gancedo Crespo (main, astros, file_io, verlet, kepler, animation)