PHYS281: Coding and Modelling Project Report

Deneth Weerasinghe,
Lancaster University
d.weerasinghe@lancaster.ac.uk

Github repository:
`https://github.com/deneth-weerasinghe/PHYS281_Simulation-Project`

December 23, 2021

Abstract

In this report, we describe to which degree a simple, 3D n-body system of gravitating particles, written in Python, can simulate Newtonian gravitation. This is done by modelling a simulation of the solar system as 10 point-like particles, consisting of the Sun, the planets and Pluto. The position of the particles are updated using a variety of integration methods, being Euler, Euler-Cromer, Euler-Richardson and Verlet, each with their sets of accuracies. Results of this simulation using each method is compared to real data of the positions and velocities of solar objects from NASA's Jet Propulsion Laboratory (JPL) dataset[1]x and further insight is gained from analysing the conserved quantities of total energy and linear momentum.

Contents

1 Introduction

1.1 Newtonian gravitation

This project attempts to simulate the motion of any number of bodies using Newtonian gravity as its underlying model, thus operating under classical mechanics.
In this model of gravitation, the gravitational force $\vec{F}$ acting on a body of mass m_1 due to a body of mass m_2 is proportional to the product of those masses and is a function of the distance $|\vec{r}|$ between those two bodies, as shown in the following[1]:

$$\vec{F} = \frac{Gm_1m_2}{|\vec{r}|^2}\hat{r} \tag{1}$$

Here, G is the constant of proportionality of this relation, known as the gravitational constant. This value has been derived empirically. One of the first times it was derived was via the 1798 Cavendish experiment, which indirectly determined the value of G as $6.74 \times 10^{-11}\mathrm{m}^3\,\mathrm{kg}^{-1}\,\mathrm{s}^{-1}$[2].
As seen, this is an example of the inverse square law: the force does not scale linearly but rather decreases quadratically as distance between bodies increase. As such, the resulting gravitational field is radial and infinite; no matter how small the masses involved are or how large the distances between bodies are, the force of gravity still acts on an object.
The gravitational acceleration $\vec{g}$ for a body of mass m_1 is then simply derived

from $\vec{F} = -m_1\vec{g}$ as per Newton's second law:

$$\vec{g} = \frac{-Gm_2}{|\vec{r}|^2}\hat{r} \tag{2}$$

For the simulation of n-bodies, it is needed to take into account every gravitational force acting on a particle. Therefore the resultant acceleration on a body is equal to the sum of all the contributions from all other particles.

1.2 Modelling and limitations

True for all simulations, there are abstractions to be used in order to greatly simplify the model. This will inevitably give rise to error when comparing the results to empirical data which will have to be taken into account when discussing results.
The first abstraction is to model all gravitating objects as point-like particles with no physical dimensions. The simplification is justified in this model due to the scale used for the simulation. In celestial contexts such as the solar system, the lengths of object is negligible when compared to the vast distances between each of them. For instance, consider the radius of the largest body in this simulation, the Sun, 696,342 km[3]. This is only 1.2% of Mercury's semi-major axis of 57,909,050 km[4]. As a result, this leads to inaccurate results when objects enter the volume that would be occupied by another object, as collision is what should happen.
Another abstraction is disregarding relativistic effects and continuing to use classical mechanics despite it being superseded by general relativity. This is justified by relativistic contributions being negligible as no object travels at relativistic speeds in this simulation. However, an observable incongruity with Newtonian mechanics is the precession of the perihelion of Mercury's orbit, which has been demonstrated to have been caused by the significant curvature of spacetime around the Sun[5]. This will not be reflected in the simulation.

1.3 Numerical Methods

In order to simulate the motion in a system, that is to calculate the positions, velocities and accelerations of all bodies at all times t, we need to estimate them using numerical methods for ordinary first-order differential equations. These are only approximations of the equations of motion and will give rise to inevitable margins of error, different for each of the methods used.
Each of the methods make use of the previous state of the system and a

time step Δt in order to estimate the current state i.e. they're all iterative methods.

1.3.1 Euler

The Euler method is the simplest approximation used in this simulation. It is derived from the Taylor expansion of the equation of motions for constant acceleration[6]. For instance for position $\vec{x}$, the expansion is:

$$\vec{x}(t + \Delta t) = \vec{x}(t) + \dot{\vec{x}}(t)\Delta t + \frac{1}{2}\ddot{\vec{x}}(t)(\Delta t)^2 + O((\Delta t)^3) \tag{3}$$

The Euler method itself is:

$$\vec{x}_{n+1} \approx \vec{x}_n + \vec{v}_n \Delta t \tag{4}$$

$$\vec{v}_{n+1} \approx \vec{v}_n + \vec{a}_n(\Delta t) \tag{5}$$

Where $\vec{x}_{n+1}$ denotes the next position after an interval Δt, calculated from the current position $\vec{x}_n$ and the current velocity $\vec{v}_n$.
Likewise with the next velocity $\vec{v}_{n+1}$ using the current acceleration $\vec{a}_n$, assuming acceleration is constant within the interval Δt.
As demonstrated, the Euler method is the Taylor expansion with higher order terms removed. Additionally, using the positions and velocities at the end of the previous iteration will lead to the simulated values to drift compared to the actual values. This is what introduces the main source of error in the simulation, which will have an error of order $(\Delta t)^2$.
However the aforementioned assumption of constant velocity also introduces its own error contribution as the intermediate acceleration is skipped over. Both sources can be mitigated with smaller Δt, increasing accuracy.

1.3.2 Euler-Cromer

The Euler-Cromer method[7] is similar to the Euler method in the form it takes, however $\vec{v}_{n+1}$ is calculated first and then this same value is used to calculate $\vec{x}_{n+1}$:

$$\vec{v}_{n+1} \approx \vec{v}_n + \vec{a}_n(\Delta t) \tag{6}$$

$$\vec{x}_{n+1} \approx \vec{x}_n + \vec{v}_{n+1} \Delta t \tag{7}$$

This should result in a more accurate simulation as updating velocity before the next position is calculated will smooth out the path taken by particles, thus leading to more stable systems. In the case of this simulation, the

orbits will be less likely to drift and more likely to conserve energy as the simulation is run for longer. More formally, the Euler-Cromer method is a symplectic integrator, which is the reason why energy is more likely to be conserved long term.

1.3.3 Euler-Richardson

As mentioned previously, the Euler method fails at taking into account the intermediary steps between iterations unless Δt is decreased further, causing the curve to not be as smooth. The Euler-Richardson introduces a solution by computing velocity and position in the middle of the time step using the original Euler method[8]:

$$\vec{a}_n = \vec{F}(\vec{x}_n, \vec{v}_n, t_n) \tag{8}$$

$$\vec{v}_{mid} \approx \vec{v}_n + \frac{1}{2}\vec{a}_n \Delta t \tag{9}$$

$$\vec{x}_{mid} \approx \vec{x}_n + \frac{1}{2}\vec{v}_n \Delta t \tag{10}$$

$$\vec{a}_{mid} \approx \frac{\vec{F}(\vec{x}_{mid}, \vec{v}_{mid}, t + \frac{1}{2}\Delta t)}{m} \tag{11}$$

$$\vec{v}_{n+1} \approx \vec{v}_n + \vec{a}_{\text{mid}} \Delta t \tag{12}$$

$$\vec{x}_{n+1} \approx \vec{x}_n + \vec{v}_{\text{mid}} \Delta t \tag{13}$$

This is more useful for velocity dependent forces but gravity is not such a force, so it's effectiveness is limited in this model. However, this method should smooth out the paths like the aforementioned Euler-Cromer method.

1.3.4 Verlet

This method strongly resembles the actual equations of motions for constant acceleration[9]. It differs from the other mentioned methods by recalculating acceleration after the position is updated when calculating the next velocity $\vec{v}_{n+1}$:

$$\vec{x}_{n+1} \approx \vec{x}_n + \vec{v}_n \Delta t + \frac{1}{2}\vec{a}_n (\Delta t)^2 \tag{14}$$

$$\vec{v}_{n+1} \approx \vec{v}_n + \frac{1}{2}(\vec{a}_{n+1} + \vec{a}_n)\Delta t \tag{15}$$

As seen, both the estimated acceleration $\vec{a}_{n+1}$ and the current acceleration $\vec{a}_n$ are averaged out to calculate $\vec{v}_{n+1}$. This estimated acceleration must

be calculated separately from the current acceleration stored in the Python class that handles particles.
Therefore the path should also be smoothed out under this method. Similar to Euler-Cromer, this method preserves the "symplectic form of phase space" i.e. it's a symplectic integrator.

2 Results

2.1 Progression

The following figures will show how the simulation evolved as it got more complex, starting with simple projectile motion under a constant gravitational field and ending with an n-body simulation of the solar system.

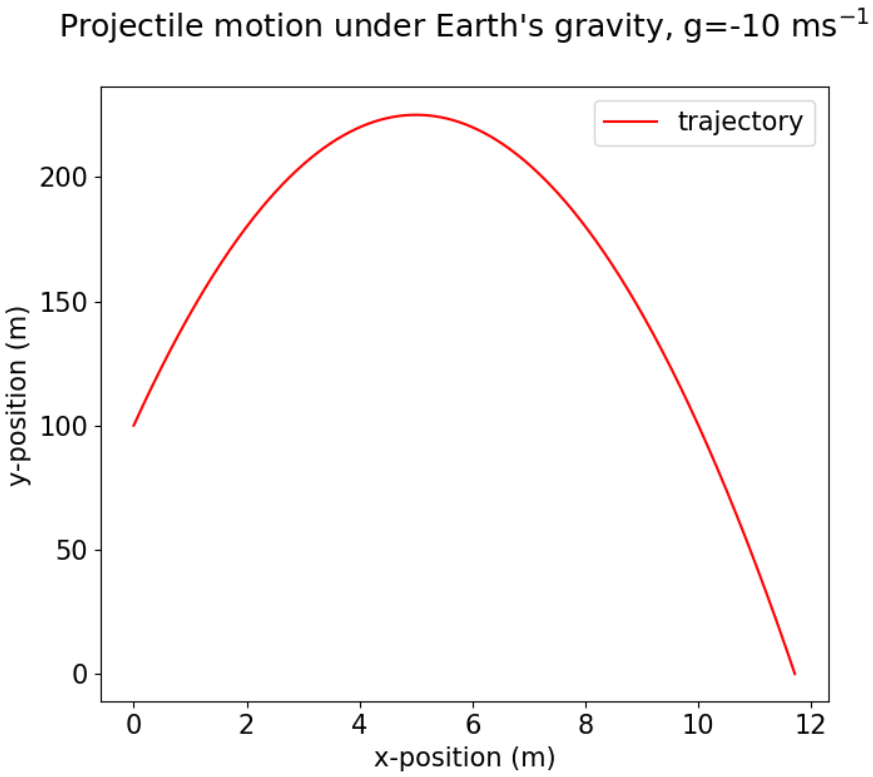

Figure 1: Plot of the trajectory of a projectile under a constant gravitational field, calculated using the Euler method. The initial conditions are: $\vec{x} = 100\hat{j}$ m and $\vec{v} = (20\hat{\imath} + 50\hat{j})$ ms^{-1}; time step is $1e-3$s

.

As shown in Fig. 1, the generated path is an arc stretched along the x-direction. This is because the projectile has an initial velocity but only the $\hat{j}$ component of velocity will be changed by $\vec{g}$.

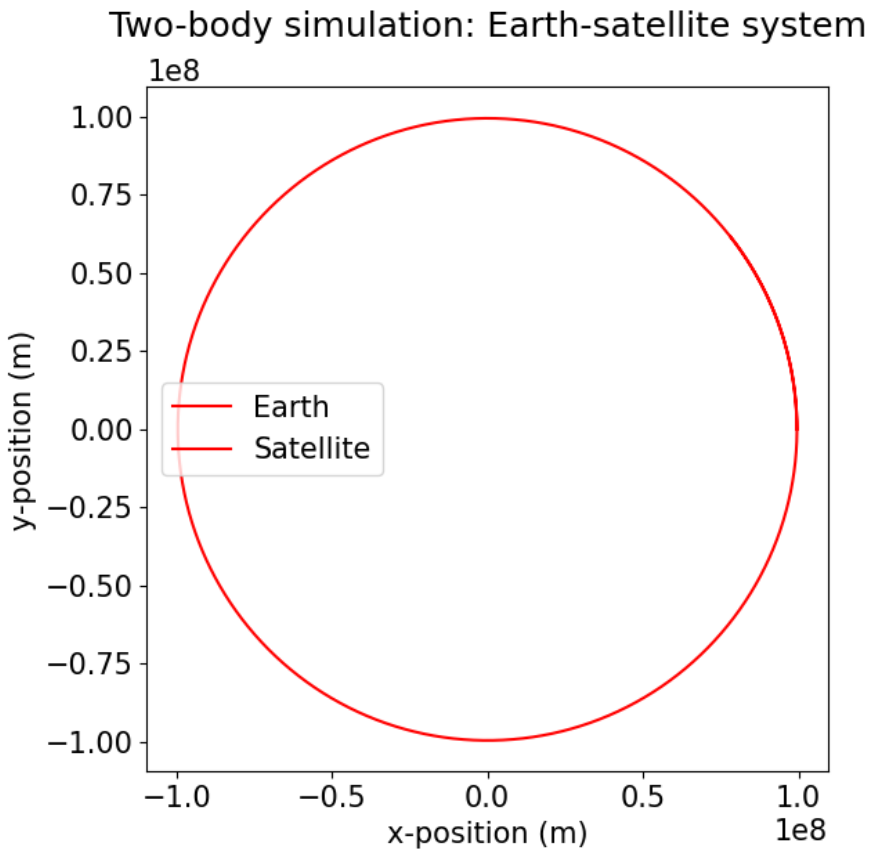

Figure 2: Orbit of a satellite around the Earth, serving as an example of a two-body system, calculated using the Euler method. The initial conditions are $\vec{x} = \vec{0}$, $\vec{v} = \vec{0}$, $\vec{a} = \vec{0}$ for the Earth and $\vec{x} = (R + 35786 \times 1e3)\hat{\imath}$, $\vec{v} = G\frac{M}{R+35786*1e3}\hat{\jmath}$ ms^{-1}, $\vec{a} = \vec{0}$ for the satellite, with time step of 1 s and 345600 iterations.

As demonstrated in Fig. 2, according to the simulation, particles in radial gravitational fields follow elliptical paths as all components of velocity and position are changed, with the exception of the $\hat{k}$ component, which is not affected by the gravitational force at all. This is due the particle orbiting in the x-y plane. This qualitative result is as expected.

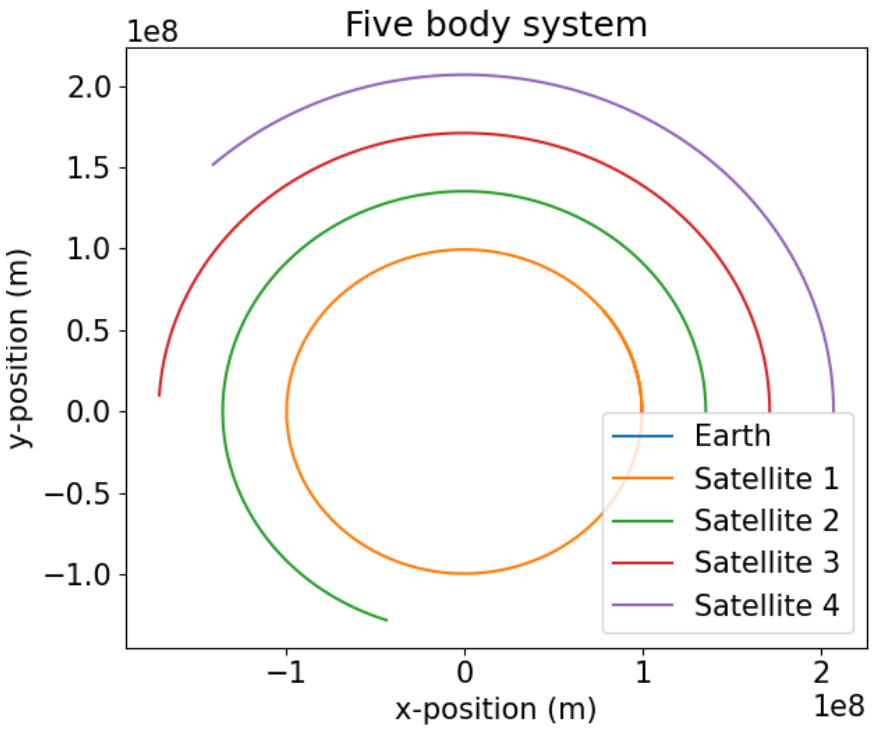

Figure 3: An extension to the case of Fig. 2 where n-bodies are simulated. In this case 4 identical satellites orbiting the Earth at specific distance intervals from each other: $\vec{x} = (R + 35786i \times 1e3)\hat{\imath}$, where i is the iteration of the for loop that generates these objects. Otherwise initial conditions are identical to the 2-body case.

Here, in Fig. 3, the simulation has demonstrated to hold for an arbitrary amount of bodies and is able to successfully simulate a system for 345600 iterations.

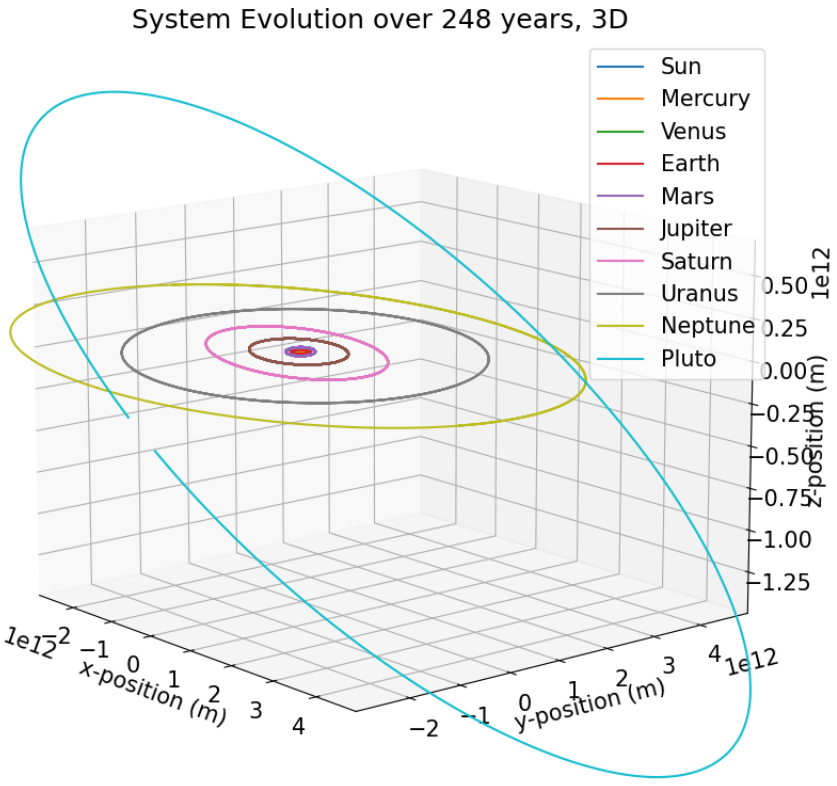

Figure 4: A full 3D simulation of the solar system using initial conditions from the JPL database, using the Verlet method, run for the equivalent of 248 years with time step of 86400 seconds (a Juliam day). Data was retrieved using the Python modules astropy[10, 11], poliastro[12] and spiceypy[13]/SPICE Toolkit[14, 15].

Fig. 4 shows the simulation is successful at its final goal of emulating the orbits of the planets and Pluto, as shown by elliptical orbits, with variations in inclination. The system forms a rough, two-dimensional disk (the plane of the solar system) as expected, showing further success.
Further analysis is required to quantify the accuracy of the simulation, as demonstrated in the following subsection.

2.2 Analysis of conserved quantities

2.2.1 Euler

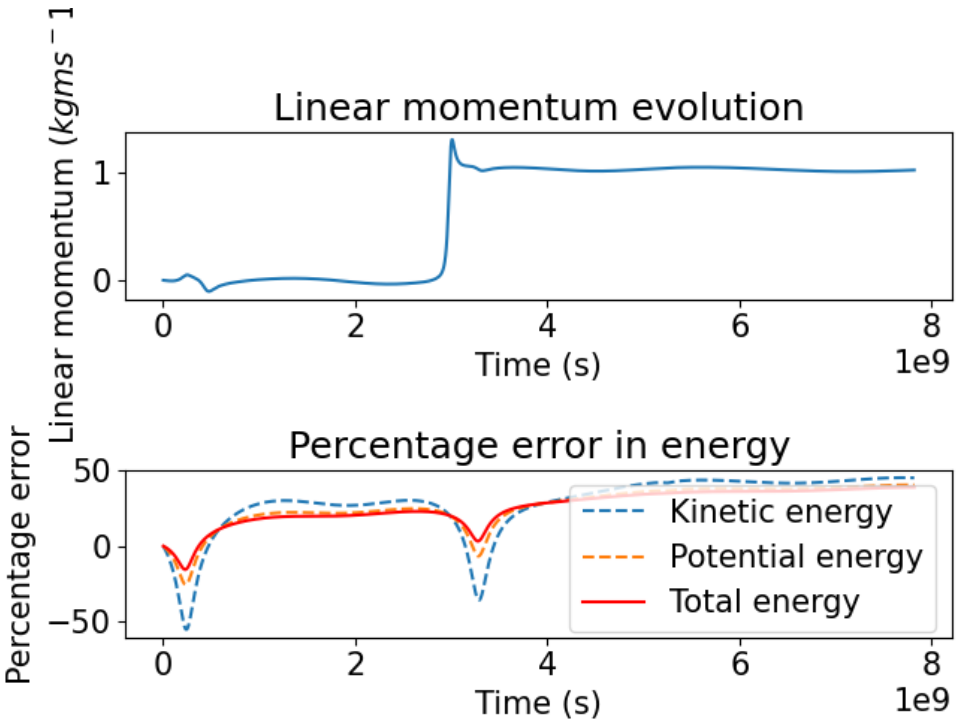

Figure 5: Graphs of the error in the system linear momentum and the system internal energy compared to their initial states at t=0 for the Euler method at time step 86400×8, iterations of $365 \times 248 \div 8$ (time step equal to 8 Julian days, run for the duration of Pluto's orbit)

As seen, the linear momentum stays relatively constant for about 3e8 seconds and then suddenly jumps up, showing a critical failure that happened in a relatively short amount of time. This can be explained by the high likelihood of drift in the orbits of the inner planets at such long iterations. The cumulative errors pile up, causing the orbits to spiral out, until they are shot out of the solar system, hence the dramatic change in momentum.

3 Conclusion

Like for all simulations, there are a wide range of errors that arose within these sets of simulated data that are derived from multiple sources. This is due to all the aforementioned assumptions and simplifications, namely modelling based on Newtonian gravitation and the usage of numerical methods whose natural error is relatively large as seen in the results section2. Therefore, the model can call upon improvements in the physics to start off with. In the future, it is expected for the model to also take into account the more accurate model of gravity: general relativity, however this might greatly increase computational power as general relativity is much more complex than the old Newtonian gravitation theory.
Furthermore, it is required to take a more in depth look into the current model using a greater variety of time steps and iterations, to see how the system behaves in other conditions. Additionally, we only calculated linear momentum and total energy and did not have time to take a look into another conserved quantity: angular momentum.

References

[1] Newton I, Cohen IB, Whitman A. The Principia: Mathematical Principles of Natural Philosophy. The Principia: Mathematical Principles of Natural Philosophy. University of California Press; 1999. Available from: https://books.google.co.uk/books?id=kmglDQAAQBAJ.

[2] Cavendish H. XXI. Experiments to determine the density of the earth. Philosophical Transactions of the Royal Society of London. 1798;88:469-526. Available from: https://royalsocietypublishing.org/doi/abs/10.1098/rstl.1798.0022.

[3] Emilio M, Kuhn JR, Bush RI, Scholl IF. Measuring the Solar Radius from Space during the 2003 and 2006 Mercury Transits. The Astrophysical Journal. 2012 May;750(2):135.

[4] Yeomans DK. HORIZONS Web-Interface for Mercury Major Body. JPL Horizons On-Line Ephemeris. 2008. Select "Ephemeris Type: Orbital Elements", "Time Span: 2000-01-01 12:00 to 2000-01-02". ("Target Body: Mercury" and "Center: Sun" should be defaulted to.) Results are instantaneous osculating values at the precise J2000 epoch, retrieved April 7, 2008. Available from: http://ssd.jpl.nasa.gov/horizons.cgi?find_body=1&body_group=mb&sstr=1.

[5] Einstein A. Die Grundlage der allgemeinen Relativitätstheorie. Annalen der Physik. 1916 Jan;354(7):769-822.

[6] Atkinson KE. An introduction to numerical analysis. 2nd ed. New York: Wiley; 1989.

[7] Cromer A. Stable solutions using the Euler Approximation. American Journal of Physics. 1981;49:455.

[8] Pitkin M. Gravity Simulation. PHYS281: Scientific Programming and Modelling Project. 2021. Accessed December 22, 2021. Available from: http://ma.ttpitk.in/teaching/PHYS281/gravity/#different-methods-of-simulating-kinematics.

[9] Swope WC, Andersen HC, Berens PH, Wilson KR. A computer simulation method for the calculation of equilibrium constants for the formation of physical clusters of molecules: Application to small water clusters. Journal of Chemical Physics. 1982 Jan;76(1):637-49.

[10] Astropy Collaboration, Robitaille TP, Tollerud EJ, Greenfield P, Droettboom M, Bray E, et al. Astropy: A community Python package for astronomy. Astronomy and Astrophysics. 2013 Oct;558:A33.

[11] Astropy Collaboration, Price-Whelan AM, Sipőcz BM, Günther HM, Lim PL, Crawford SM, et al. The Astropy Project: Building an Open-science Project and Status of the v2.0 Core Package. The Astronomical Journal. 2018 Sep;156(3):123.

[12] Rodríguez JLC, G JM, Hidalgo A, Bapat S, Astrakhantsev N, Eleftheria C, et al.. poliastro/poliastro: poliastro 0.15.2 (astroquery edition). Zenodo; 2021. Available from: https://doi.org/10.5281/zenodo.5035326.

[13] Annex AM, Pearson B, Seignovert B, Carcich BT, Eichhorn H, Mapel JA, et al. SpiceyPy: a Pythonic Wrapper for the SPICE Toolkit. Journal of Open Source Software. 2020;5(46):2050. Available from: https://doi.org/10.21105/joss.02050.

[14] Acton CH. Ancillary data services of NASA's Navigation and Ancillary Information Facility. Planetary and Space Science. 1996;44(1):65-70. Planetary data system. Available from: https://www.sciencedirect.com/science/article/pii/0032063395001077.

[15] Acton C, Bachman N, Semenov B, Wright E. A look towards the future in the handling of space science mission geometry. Planetary and Space Science. 2018;150:9-12. Enabling Open and Interoperable Access to Planetary Science and Heliophysics Databases and Tools. Available from: https://www.sciencedirect.com/science/article/pii/S0032063316303129.