Critiques of a Computational Model of the Solar System - Project Report for Computer Simulation

Luke Jennings

4th April 2025

---

1 Introduction

This report aims to demonstrate an object-oriented model to simulate the behavior Newtonian gravity has in maintaining stable orbits in the solar system. Using this model, I conduct the following experiments and discuss their results;

1. Accuracy of the modeled orbital period of each planet compared to scientifically accepted results.
2. Energy conservation properties and the behavior of alternative integration methods.
3. Occurrences of planetary alignment.

Experiments 1 and 2 aim to demonstrate the accuracy of the model by comparing computed data with accepted data. For instance, the total energy of any closed system should be constant, and so we expect that of the model; any significant deviation would contribute to evidence of unreliability. Using NASA's observations and accepted physical laws, this report intends to establish a precise model for our solar system and investigate its limits by comparing the results of the above experiments with reality.

Throughout this text, **bold** terms denote a Class, and *italic* terms denote a method or variable.

---

2 Methods

2.1 Design Overview

In designing a way to implement and interface with the model, I implement two key classes: a **Simulation** class in which all UI interfaces and rendering processes are handled, and a **Space** class which handles the behavior of the solar system under given parameters. While any **Simulation** requires reference to a **Space** instance, **Spaces** instead function independently of any **Simulation** method; this autonomy is intended to produce consistent and reproducible results regardless of the way in which it is simulated.

2.2 Component Methods

A component here describes a specific aspect of the physical model, which in the context of the solar system consists of the star and all its planets. To model these components, I first implement an **Object** class which represents an arbitrary physical instance in 2D space; a point mass with position, momentum, and acceleration. **CelestialBody** inherits object to represent an arbitrary body, which itself is inherited by **Planet** to specifically contain orbital period data for experiment 1.

2.2.1 Object

The **Object** class should maximally abstract context to create a framework for any component required for the solar system. Its constructor takes a reference space and initial parameters such as name, mass and initial dynamical properties. To integrate an Object's dynamics, three methods define unique integration types: *beeman_update* implementing Beeman integration, *euler_cromer_update*, and *direct_euler_update* implementing their respective techniques.

Object also contains *get_acceleration*, a method which computes acceleration of an object under the influence of the iterable input variable *focus_objects*, which should include all objects that cause any meaningful gravitational attraction. By adding the magnitude of force each focus object attracts the object by (given by Newton's law of gravitation), this method returns acceleration according to Newton's second law. That is, for an object (mass M_i, distance r_i) from *focus_objects* with vector $\underline{r_i}$ from the current object (mass m) the resultant acceleration is given by

$$\underline{F}_i = \frac{GM_i m}{r_i^2}\hat{\underline{r}}_i = m\underline{a}_i \quad \Longrightarrow \quad \underline{a}_i = \frac{GM_i}{r_i^2}\hat{\underline{r}}_i$$

which gives a total acceleration computed by the sum

$$\underline{a} = G \cdot \sum_{\text{focus\_objects}} \frac{M_i}{r_i^2}\hat{\underline{r}}_i.$$

2.2.2 Vector2D

To handle all vector operations so as to keep other classes as abstract as possible, **Vector2D** stores the two components of a vector and contains methods to perform arithmetic. This includes vector addition, scalar multiplication, normalization and most notably an *angle_between_xaxis* method to evaluate the angle between the vector and the positive x-axis, (1,0). For a normalized vector $\langle x, y\rangle$ this calculation reduces to

$$\langle x, y\rangle \cdot \langle 1, 0\rangle = \|\langle x, y\rangle\| \cdot \|\langle 1, 0\rangle\| \cdot \cos\theta$$

$$\Longrightarrow \quad \theta = \begin{cases} \arccos x & \text{if } y \geq 0 \\ 2\pi - \arccos x & \text{otherwise} \end{cases}$$

which defines an invaluable method for evaluating planetary alignment in experiment 4.

2.3 Simulation Methods

A simulation method is one which resides in a class representing some abstract component of the model; these classes consist of the **Space** and **Simulation** classes which, using the component classes, complete the model.

2.3.1 Space

Space is designed to simulate a solar system independently of rendering, doing so by handling the physical modeling of all celestial bodies based purely on parameters from a JSON file. Its constructor method instantiates each body, calculates interactions between them, and evaluates their greatest orbital radii R, a value that allows scaling of the trail limit of all bodies for ease of visibility of outer planets. This function $f : [0, R] \to \mathbb{N}$ is defined

$$f(r) = 50\lceil \frac{6r}{R} + 1 \rceil.$$

The class includes two methods for conducting experiments, *determine_alignment* considers planetary angles and the deviation from their mean; the number of planets and tolerance of alignment are parameters given by the JSON file. *get_energy_levels* returns a tuple of the total kinetic and potential energies of the system through the sum of each body's energies, determined as follows:

$$K_{total} = \sum_{\text{body} \in \text{bodies}} \frac{m_{body} \cdot \|\bar{x}_{body}\|^2}{2}$$

$$P_{total} = M_{star} G \sum_{\text{body} \in \text{planets}} \frac{m_{body}}{\|\bar{x}_{body}\|}.$$

where $\bar{x}_{body}$ denotes the position vector of a body.

In the *update* method, the instance of each body is integrated over a time_step and subsequently checked if the first orbit is complete. A planet's first orbit is decided based on the previous position; by the mean value theorem, if the body was in the fourth quadrant before integration, and is now in the first quadrant (relative to an axis centered on the star), the planet must have crossed the positive x-axis. Since all planets initially reside on the positive x-axis, the orbital period of each planet is the current elapsed time (evaluated by adding the current simulation's time_step to the variable elapsed_time every frame).

The result of both the constructor and update method is a complete physical model of bodies in the solar system and a way to simulate their behavior over time. Since the time between frames is variable, time scaling is possible at real-time, but the time step should always be far less than the inner-most planet's orbital period to maintain an accurate approximation. This time scaling allowed me to investigate the results of experiments

2.3.2 Simulation

A **Simulation** instance references a **Space** instance to construct a UI dependent on the composition of the **Space**. Contained within are 5 methods, including the constructor method which initializes all matplotlib axes and figures.

The update method of **Simulation** first calls the Space's update method, and thus updates the patch center of each object's artist and their trail which now contains the new position. Once rendering is complete, the time taken is recorded and added to an array storing the time taken each frame; this array is sampled to give an estimate of frame rate. This among other simulation and experiment parameters are then returned from *generate_output*.

Once both **Space** and **Simulation** classes are initialized, calling *run* in **Simulation** will begin a matplotlib **FuncAnimation** sequence of it's update method using parameters passed in construction, and pause after the iteration limit is reached.

---

3 Results

What follows is each experiment's results presented by means of excel plotting, with brief comments on the data collection procedure and how results are plotted.

3.1 Experiment 1 - Orbital Periods

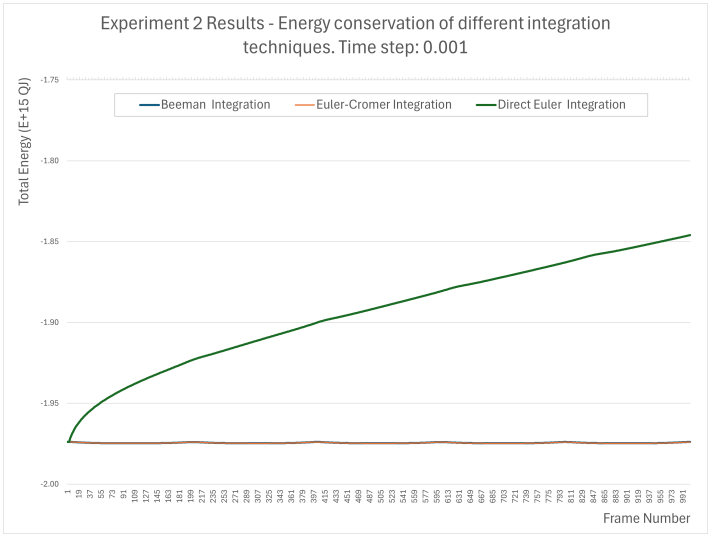
Experiment 1 investigates the simulated orbital period of each planet in the solar system and compares to the result found by NASA's collected data. Figure 3 plots the difference between the orbital period simulated by the model and the orbital period observed by NASA to three decimal places. Since the data was so minimal, data collection reduced to copying console output to excel to create plots.

3.2 Experiment 2 - Energy Conservation

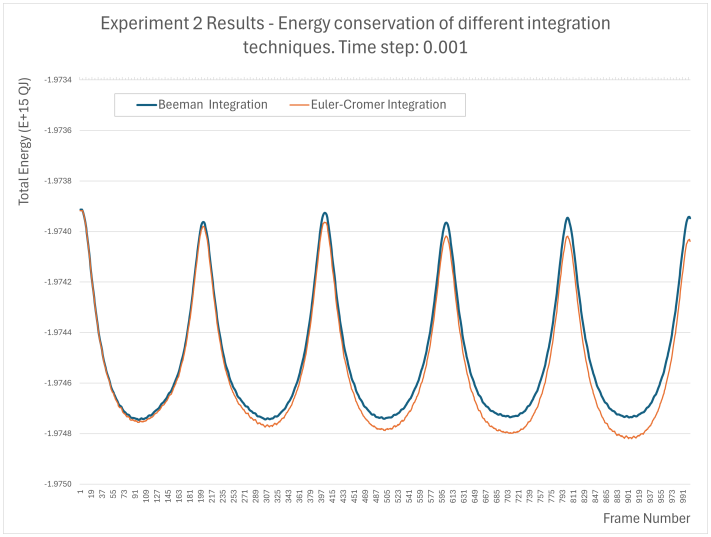
Experiment 2 measures the energy level over a simulation's lifespan and graphically compares the influence each integration method has on energy conservation. Two plots were created, one which shows the inconsistency in direct-Euler integration and one which shows the variation in energy levels of a more precise model. Data was collected and processed by writing basic data to a text file and importing to excel.

3.3 Experiment 4 - Planetary alignment

To collect results for experiment 4, the console output of **Simulation** includes occurrences of planetary alignment as determined during the processes of **Space**.



(a) Comparison of energy levels of all three integration methods



(b) Above plot with direct Euler integration excluded.

Figure 2: Plot of energy level of the solar system using different integration techniques.

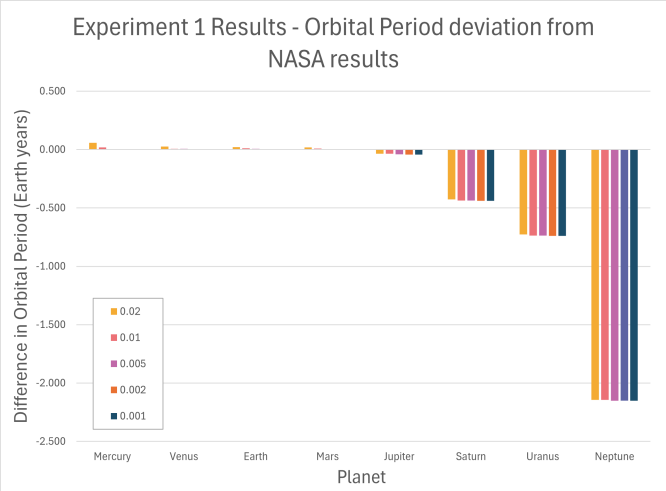


Figure 3: Excel plot of results from experiment 1 over five different simulation time steps.

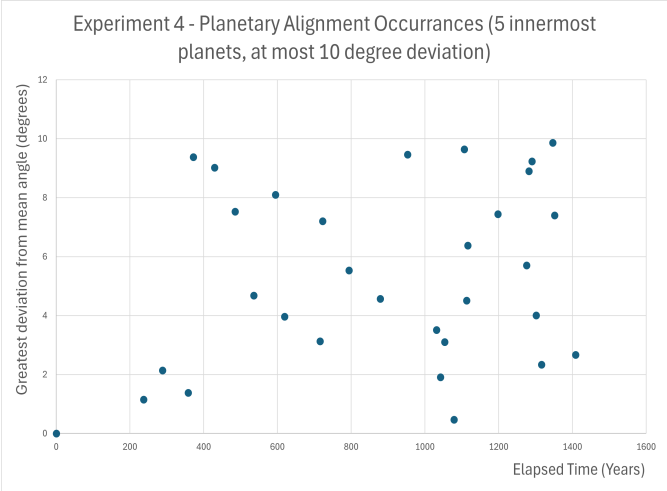


Figure 4: Scatter plot of alignment occurrences over 1500 years, simulated with time step 0.005.

---

4 Discussion

4.1 Orbital Periods

To demonstrate deviation of orbital period from accepted results, Figure 3 contains a bar plot of the difference between simulated orbital period and accepted orbital period. As seen, the difference gets increasingly greater as orbital radius increases to Neptune, as expected since these outer planets take much longer to orbit and thus experience approximation error more considerably in their first orbit. Note also that each planet has results for over a range of time steps, all of which give almost the same result. This supports the reliability of *time_step* variation if not the whole simulation, since the results given are consistent over the different simulation parameters. An unexpected feature of this data is the significant difference under a *time_step* of 0.02 at inner-most planets, indicating that the time increment is too large to give reliable results. For this reason, further experiments are simulated at time increments of at most 0.005.

4.2 Energy Conservation

Experiment 2 demonstrates the effectiveness of the various integration methods by graphically demonstrating the variation in total energy. Figure 2 contains two plots showing different results of the experiment. Plot 2a shows that the direct-Euler method of approximating gravitational motion does not conserve energy nearly as much as other methods; this indicates that the direct Euler technique is not physically accurate and thus will not provide reliable results. For this reason, plot 2b excludes the data from direct Euler, and thus shows the similarity between Beeman and Euler-Cromer integration. Despite the clear variation from some constant energy total, the deviation appears periodic (especially Beeman integration; the Euler-Cromer results begin to deviate from such periodicity since the start of the simulation), and remains within a range differing by only the fourth significant figure, suggesting this deviation from average is relatively small. In addition, since the peaks and troughs of Beeman integration results appear more consistent than those from Euler-Cromer, it is clear that Beeman integration provides a system in which energy is conserved within a reasonable margin of error, producing an accurate integration scheme to model the solar system.

4.3 Planetary Alignment

Results from Experiment 4 show a plot with little to no recognizable pattern except for the initial few hundred years, where little to no alignments occur. I suspect this may be due to the artificial placement of planets along the x-axis at the simulation's start. The apparent randomness of the proceeding data, however, suggests this could be an unlikely but natural absence of occurrences. The clear absence of periodicity in occurrences indicates that no bias significantly influences the overall behavior of the model, and thus supports the reliability of the model's ability to produce consistent results.

5 Conclusion

From the results presented in section 4, the model described constructs a consistent simulation of a stable solar system configuration (since experimental data aligns with both NASA measurements and conservation laws), producing physically accurate data to take measurements from. Despite this, modification to include any moon or other non-planet component of the solar system proves difficult, and so in any similar project proceeding this one I intend to more carefully design the functionality of each class to allow an easier integration of new components. Additionally the behavior of the model is not perfectly accurate due to the variation of energy, which would not occur in a closed system like our solar system. To conclude, the model provided will give a precise rendering of orbital motion in the solar system, but is limited by both its inability to easily integrate unique components and inexact integration techniques causing energy fluctuation, hence generating a simplistic but reasonably consistent simulation.