Gymnasium Ernestinum
Rinteln

Subject area Physics
Thesis

Simulation of the Two-Body problem

Author: Schweighöfer, Connor
Tutor: Dr. Christopher Kranz

Submission date: Monday, 20th March 2023

Contents

1 Introduction

The study of celestial mechanics has long been of great importance in understanding the motion of objects in space. By modeling the behavior of celestial bodies, we can gain insights into the workings of the universe, facilitate space exploration, and develop technologies such as satellite communication and GPS.

One of the fundamental problems in celestial mechanics is the two-body problem, which involves modeling the motion of two celestial bodies under the influence of gravitational attraction. While the two-body problem has been studied for centuries, it remains a topic of great interest and significance in contemporary research.

However, the two-body problem is a simplified version of the more complex n-body problem, which involves modeling the motion of three or more celestial bodies under the influence of gravitational attraction. While the two-body problem can be solved analytically, the n-body problem has proven to be much more difficult to solve. As a result, numerical integration methods are often used to simulate the n-body problem.

In this paper, I present a user friendly simulation tool for the two-body problem and discuss its development and implementation. The focus is on the numerical integration method used to simulate the motion of celestial bodies, and the features of the simulation tool. This provides a platform for others to explore and investigate celestial mechanics. The goal is to contribute to a better understanding of the dynamics of celestial bodies and provide a valuable resource for studying and exploring the universe.

2 Physical fundamentals of the two-body problem

To simulate the real-world interactions between two celestial bodies, a fundamental understanding of the physical principles governing their behavior is necessary. The two-body problem, which deals with the motion of two celestial objects under the influence of gravity, has been an important topic in physics for centuries. By studying the physical principles behind the two-body problem, we can gain insight into the behavior of celestial bodies and use that knowledge to simulate their interactions in a computer program.

2.1 Newton's Law of Gravity

In 1687, Sir Isaac Newton published his famous work "Philosophiæ Naturalis Principia Mathematica" which laid the foundation for classical mechanics and introduced the laws of motion and the law of universal gravitation. The law of gravity states that every particle in the universe attracts every other particle with a force that is directly proportional to the product of their masses and inversely proportional to the square of the distance between their centers.

Mathematically, this can be expressed as:

$$F_G = G \cdot \frac{m_1 \cdot m_2}{r^2} \tag{1}$$

Where F_G is the force of gravity, G is the gravitational constant ($6.67 \cdot 10^{-11} N \cdot \frac{m^2}{kg^2}$) [2], m_1 and m_2 are the masses of the two objects, and r is the distance between their centers.

2.2 Newton's Laws of Motion

Newton's Laws of Motion consists of three laws which also published in the "Philosophiæ Naturalis Principia Mathematica". These laws form the basis of classical mechanics and are essential to understanding the behavior of objects in motion [12].

First Law: Inertia principle
An object at rest will remain at rest unless a force acts upon it, and an object in motion will continue moving at the same speed and in the same direction unless a force acts upon it.

Second Law
If a force (F) acts upon an object of mass (m), the object will experience an acceleration (a) given by:

$$a = \frac{F}{m} \tag{2}$$

This equation is often written as F = ma, and it applies only to situations in which the mass does not change during the acceleration.

Third Law
If one object produces a force on a second object, the second object produces an equally strong force back onto the first object in the opposite direction. These are sometimes called "equal and opposite forces," and it's very important that you remember that these two forces act on two different objects.

2.3 Kepler's Laws

Kepler's laws are of central importance for the two-body problem. The three laws describe the motion of planets and other celestial bodies around their sun and thus lay the foundation for the laws of celestial mechanics. The laws are named after the German astronomer Johannes Kepler, who formulated them in the 17th century on the basis of observations and calculations [11] [2].

First Law
The shape of a planet's orbit is an ellipse with his sun at one focus, so the distance between a planet and his sun is not, in general, constant.

Second Law
An imaginary line between a planet and his Sun sweeps out equal areas in equal times, so a planet moves faster when it's in the portion of its orbit closer to his Sun [1].

Third Law
The square of a planet's orbital period is proportional to the cube of the semi-major axis of the planet's orbit, so planets far from the Sun take longer to complete one orbit than planets close to the Sun.

2.4 Numerical Integration Method: Euler Method

The Euler method is a simple and intuitive numerical integration method that can be used to solve ordinary differential equations. In the context of simulating the two-body problem, it can be used to approximate the position and velocity of each body at each time step [4].

The basic idea of the Euler method is to use the current position and velocity of a body to estimate its position and velocity at the next time step, based on the acceleration that the body is experiencing. This estimate is then used as the new *current* position and velocity, and the process is repeated to simulate the motion of the body over time.

In the context of simulating the two-body problem, the Euler method can be implemented as follows:

1. Set the initial position and velocity of each body.
2. Choose a time step Δt.
3. For each time step:
    (a) Calculate the acceleration experienced by each body, based on the gravitational force between the two bodies.
    (b) Update the velocity of each body using the current acceleration and the estimated change in velocity over the time step.
    (c) Update the position of each body using the current velocity and the estimated change in position over the time step.
    (d) Repeat for specific number of time steps.

The Euler method is a first-order method, which means that its accuracy depends on the size of the time step Δt. Smaller time steps generally lead to more accurate results, but at the cost of increased computational time. Additionally, the Euler method can be subject to numerical instability if the time step is too large, which can cause the simulation to diverge over time.

There are more advanced numerical integration methods, such as the forth-order Runge-Kutta method [13], which are more accurate and stable than the Euler method. However, the Euler method is still useful in certain contexts, such as simple simulations or as a starting point for more advanced methods.

3 Simulation

In order to study the behavior of the two-body problem, numerical methods are often employed due to the complexity of the equations involved. In this chapter, we will describe a simulation of the two-body problem using numerical integration methods, specifically the Euler method. We will discuss the implementation of the simulation and present the results obtained from the simulation. You can find the source code for this project on my GitHub repository, with a specific focus on the `thesis` branch, which contains the version used for this paper and is separate from any newer versions [6] [7].

3.1 Description

3.1.1 Technologies

The Two-body simulation was implemented using `Java` and the `JavaFX` library. Java is a general-purpose programming language that is commonly used in enterprise and scientific applications [9]. `JavaFX`, on the other hand, is a set of libraries for building desktop and mobile applications that run on the Java Virtual Machine (JVM). `JavaFX` provides tools for creating graphical user interfaces (GUIs), animations, and multimedia content. The simulation runs as a `JavaFX` application, with a `Canvas` element used to render the bodies and their trajectories [10]. I additionally used Maven in my project to manage dependencies, compile source code, and package the application into an executable JAR file. With Maven, I was able to easily manage the project's build lifecycle and automate many common development tasks, making the development process more efficient and streamlined [3].

3.1.2 Vectors

In my simulation, I needed to perform various operations on vectors such as addition, multiplication, and normalization. While Java does have a `Vector` class, it is not a mathematical vector but rather a data structure for storing and manipulating elements of an array in a thread-safe manner. Therefore, I implemented my own `Vector2D` class that would allow me to perform mathematical operations on vectors.

My `Vector2D` class includes methods for adding and multiplying vectors, computing the magnitude of a vector, normalizing a vector, and setting the magnitude of a vector. This implementation allowed me to perform the necessary operations on vectors within my simulation and provide a more readable code.

3.1.3 Body

The `Body` class models a celestial body within the two-body simulation. It comprises of the following properties:

- `G`: A constant value representing the gravitational constant $6.67 \cdot 10^{-11} \frac{m^3}{kg \cdot s^2}$.
- `mass`: A `double` value representing the mass of the body.
- `position`: A `Vector2D` object that represents the current location of the body in 2D space. The unit of measurement is in meters, where an `x` and `y` value of `10` denotes that the body is located `10` meters to the right and `10` meters down. It should be noted that the coordinate system used considers the origin to be positioned at the top left corner of the canvas, where the y-axis increases as it moves downwards, and the x-axis increases as it moves to the right [5].
- `velocity`: A `Vector2D` object that represents the current velocity of the body in $\frac{m}{s}$.

Moreover, the `Body` class offers the following methods:

- `draw(graphics: GraphicsContext)`: This method is responsible for rendering the body on the canvas using the provided `GraphicsContext` object. It first calculates the appropriate pixel location to display the body because the user has the option to modify the scale of the entire canvas.
- `attract(other: Body, timeStep: double)`: This method updates the acceleration of the current body based on the gravitational force exerted by another celestial body. It then applies the calculated acceleration to the velocity of the body and subsequently updates the position of the body based on its new velocity, all scaled by the time step (Δt) to ensure correct values.

Below is the implementation of the `attract` method of the `Body` class:

```
// Body#attract
public void attract(Body other, double timeStep) {
    Vector2D force = Vector2D.subtract(other.position, position);
    double fg = G * mass * other.mass;
    double magnitudeSq = force.magnitudeSquared();
    fg /= magnitudeSq;
    force.setMagnitude(fg);

    force = force.divide(mass).multiply(timeStep);
    velocity.add(force);
    position.add(Vector2D.multiply(velocity, timeStep));
}
```

The `attract` method of the `Body` class calculates the gravitational force exerted by another body on the current body. This is done by first creating a `Vector2D` object named `force` that represents the vector pointing from the current body to the other body. Then, the magnitude of the gravitational force between the two bodies is calculated using the formula derived from Newton's Law of Gravity (1). The magnitude of the `force` vector is set to this value. Next, Newton's second law (2) is applied to update the acceleration of the current body by adding the force vector divided by the mass of the current body to the current acceleration vector. Finally, the acceleration is scaled by the time step and added to the current velocity, which is then multiplied by the time step and applied to the position to update its value.

3.1.4 TimeMode

The `TimeMode` enum is responsible for controlling the scale of time in the simulation. It allows the user to adjust the speed of the simulation by increasing or decreasing the time scale. The simulation offers the user a choice of six modes (`SECOND`, `MINUTE`, `HOUR`, `DAY`, `MONTH`, `YEAR`) to adjust the time scale. By selecting a mode, one second of real time is mapped to the corresponding duration in the simulation. The simulation also provides a slow mode feature that reduces the simulation time by a factor of 10. For instance, if the current mode is set to `HOUR` and the slow mode is activated, one second of real-time is equivalent to one-tenth of an hour, which means 6 minutes per real time second. It should be noted that these times are only approximate and may contain some inaccuracies.

3.1.5 DistanceMode

The `DistanceMode` enum is responsible for regulating the scale of the simulation's full canvas[1]. By increasing or decreasing the scale of the canvas, it allows the user to control the level of detail in the simulation. The user is presented with a selection of seven modes (`KILOMETER`, `LUNAR`, `TWO_LUNAR`, `ASTRONOMICAL_UNIT`, `TWO_ASTRONOMICAL_UNITS`, `LIGHT_YEAR`, `PARSEC`), which they can use to adjust the distance for the entire canvas. It is worth noting that while the user can modify the distance, there may be some discrepancies in the exact measurements due to potential errors in time.

3.1.6 Simulation and Simulation Loop

The `Simulation` and `SimulationLoop` classes are an essential part of the simulation application. They work together to provide a realistic and interactive simulation of gravitational forces between two celestial bodies.

The `Simulation` class extends the `AnimationTimer` class, which allows it to run continuously and update the canvas with each iteration. It uses a `ScheduledExecutorService` to schedule the `SimulationLoop` to run at fixed intervals. Separating the drawing of the bodies and the calculation of their positions into two continuous running loops was necessary due to certain limitations. For instance, the `AnimationTimer` does not allow setting specific intervals, which are necessary for time step calculation. Furthermore, the `ScheduledExecutorService` runs on a different `Thread`, and thus drawing cannot be performed in that loop. This is because `JavaFX` does not support multi-threaded drawing other than using the `Platform.runLater` method which would on the other hand slow down the simulation.

The `Simulation` class takes care of starting, stopping, and handling the animation loop. It also initializes the `TimeManager` to control the time scale of the simulation. When the simulation is started, the `Simulation` class initializes the `TimeManager` and sets up the `SimulationLoop` to run at fixed intervals.

The `SimulationLoop` class implements the `Runnable` interface and calculates the gravitational forces between the two celestial bodies in a loop. It takes in the two Body objects and the time step as parameters. The run method of the `SimulationLoop` updates the position of the two celestial bodies based on the gravitational forces acting on them and runs continously with a delay of 10 milliseconds.

Together, the `Simulation` and `SimulationLoop` classes provide a simulation of gravitational forces between celestial bodies. In order to make the simulation look more smoothly you could reduce the delay between each calculation.

[1] 1000px by default

3.2 Results

After implementing the simulation, it's time to test it out and see the results. To demonstrate the simulation's capabilities, a pre-configured scenario of the Sun and Earth has been provided as a preset. The values were obtained from various sources, including NASA and other astronomical data sources [8] [14].

Sun

- `Position`: The `x` and `y` positions of the Sun are both set to $1.495978707 \cdot 10^{11}$, which places the Sun at the center of the simulation.
- `Velocity`: In this preset, the Sun is stationary, which is not realistic, but it simplifies the visualization of the Sun-Earth orbit.
- `Mass`: The mass of the Sun in this preset is approximately $1.989 \cdot 10^{30}$ kilograms.
- `Color`: The color of the Sun in the simulation is red.

Earth

- `Position`: The `x` position of the Earth is set to 0, and the `y` position is set to $1.495978707 \cdot 10^{11}$, which places the Earth on the left side of the simulation canvas and approximately 1 astronomical unit away from the Sun.
- `Velocity`: The `x` velocity of the Earth is set to 0, and the `y` velocity is set to $30 \cdot 10^3 \frac{m}{s}$.
- `Mass`: The mass of the Earth in this preset is approximately $5.972 \cdot 10^{24}$ kilograms.
- `Color`: The color of the Earth in the simulation is blue.

The choice of `DistanceMode` is `TWO_ASTRONOMICAL_UNITS` in order to illustrate the complete orbit, as one astronomical unit corresponds to the radius of the sun-earth orbit. On the other hand, the `TimeMode` is set to `MONTH` without slowmode.

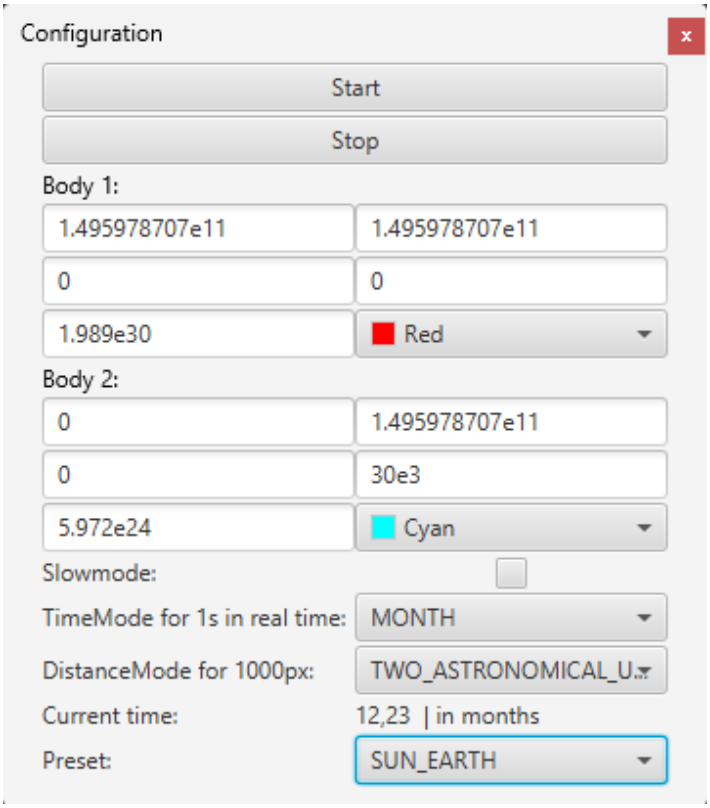

Figure 1: Configuration Dialog using the Sun-Earth preset

When running the simulation with the `Sun-Earth` preset, we can visualize the interaction between the two celestial bodies. The Earth orbits around the Sun, while the Sun remains stationary at the center of the simulation. However, this is just one possible scenario, and the simulation's features allow us to create and explore different configurations by adjusting the parameters and selecting different celestial bodies.

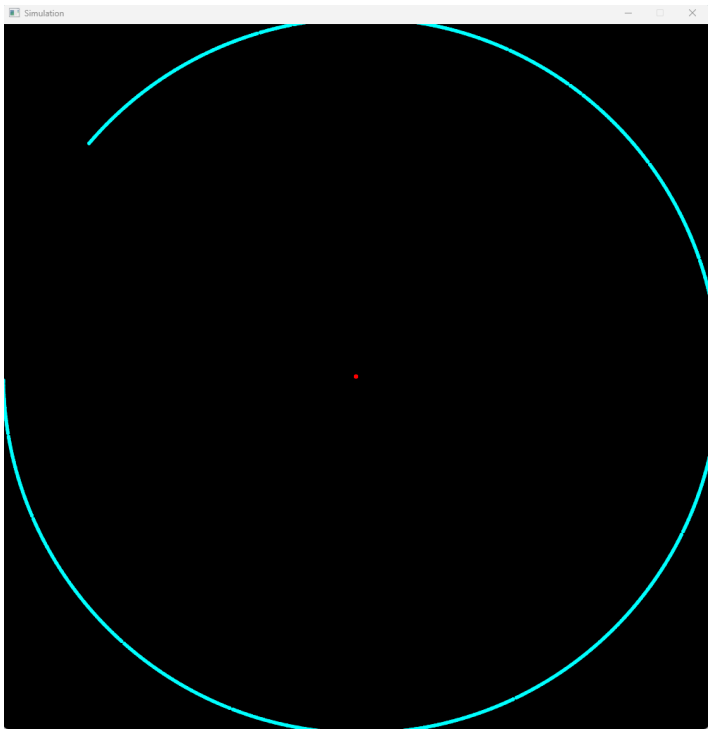

Figure 2: Sun-Earth orbit result

It's worth noting that the accuracy of the simulation is limited by the precision of the calculations and the parameters used. As a result, it's not a precise model of the universe, but rather a tool for visualization and exploration.

In summary, the simulation is a valuable resource for studying celestial mechanics and investigating the behavior of different celestial objects. By using the provided presets and features, we can observe the motion of planets and stars and gain a better understanding of the fundamental forces that govern our universe.

4 Summary

This project involves the creation of a simulation tool for visualizing the two-body problem. The tool allows users to simulate the motion of celestial bodies such as planets and stars, and observe their interactions.

Various parameters such as the masses, positions, velocities, and gravitational forces of the celestial bodies can be adjusted in the simulation. The tool also provides presets, including the Sun-Earth preset, which allows users to quickly set up simulations of well-known celestial systems.

While the simulation tool has proven to be useful for visualizing and studying celestial mechanics, there are potential areas for improvement. For example, the accuracy of

the simulations could be increased by implementing more precise calculations using the forth-order Runge-Kutta method and modeling additional physical phenomena. To further enhance the simulation's performance and accuracy, it could be beneficial to utilize a lower-level programming language, such as `C++`, which would allow for the use of smaller time steps and faster calculations. One possible enhancement for the user experience would be to enable the use of presets in `JSON` format, which would allow users to create and share their own configurations more easily. Another potential improvement would be to make the diameter proportional to the mass of each celestial body, which would enable visualization of the mass as well. One direction for future development of the simulation tool is to integrate n-body problem simulation, which would allow us to study the interactions between multiple celestial bodies. Another potential improvement is to extend the visualization to 3D space, which would enable us to better understand the spatial relationships between different celestial objects.

In addition to its current uses, the simulation tool has the potential to be applied in various areas such as education and research. For instance, the tool could be used to teach students about celestial mechanics and the fundamental forces that govern the universe. Researchers could also use the tool to study and analyze various celestial systems.

Overall, the simulation tool serves as a valuable resource for understanding and exploring celestial mechanics. Its potential for future development and application in different fields makes it a promising tool for the scientific community.

References

[1] Antonio González Fernández. *Animation of the Kepler's second law for a planetary orbit*. https://commons.wikimedia.org/wiki/File:Kepler-second-law.gif#/media/File:Kepler-second-law.gif. Online, accessed 10-March-2023. 10 October 2010, 10:19 (UTC).

[2] Daniel Fleisch and Julia Kregenow. *A Student's Guide to the Mathematics of Astronomy*. Cambridge University Press, 2013.

[3] Apache Software Foundation. *Apache Maven*. https://maven.apache.org/. Online, accessed 13-March-2023.

[4] Nicholas J. Giordano and Hisao Nakanishi. "Computational Physics, Second Edition". In: (2006), p. 115.

[5] Oracle. *JavaFX JavaDocs*. https://openjfx.io/javadoc/17/index.html. Online, accessed 11-March-2023.

[6] Connor Schweighöfer. *SquidXTV/TwoBodyProblem*. https://github.com/SquidXTV/TwoBodyProblem/tree/thesis. Online, accessed 19-March-2023.

[7] Connor Schweighöfer. *SquidXTV/TwoBodyProblem Release*. https://github.com/SquidXTV/TwoBodyProblem/releases/tag/v1.0.0. Online, accessed 19-March-2023.

[8] Wikipedia. *Earth's orbit*. https://en.wikipedia.org/wiki/Earth's_orbit. Online, accessed 18-March-2023.

[9] Wikipedia. *Java (programming language)*. https://en.wikipedia.org/wiki/Java_(programming_language). Online, accessed 10-March-2023.

[10] Wikipedia. *JavaFX*. https://en.wikipedia.org/wiki/JavaFX. Online, accessed 11-March-2023.

[11] Wikipedia. *Kepler's laws of planetary motion*. https://en.wikipedia.org/wiki/Kepler's_laws_of_planetary_motion. Online, accessed 10-March-2023.

[12] Wikipedia. *Newton's laws of motion*. https://en.wikipedia.org/wiki/Newton's_laws_of_motion. Online, accessed 10-March-2023.

[13] Wikipedia. *Runge–Kutta methods*. https://en.wikipedia.org/wiki/Runge%E2%80%93Kutta_methods. Online, accessed 10-March-2023.

[14] Wikipedia. *Sun*. https://en.wikipedia.org/wiki/Sun. Online, accessed 18-March-2023.

List of Figures

5 Erklärung zur Selbstständigkeit

Hiermit versichere ich, dass ich die Arbeit selbstständig angefertigt, keine anderen als die angegebenen Hilfsmittel benutzt und die Stellen der Facharbeit, die im Wortlaut oder im wesentlichen Inhalt aus anderen Werken übernommen wurden, mit genauer Quellenangabe kenntlich gemacht habe. Verwendete Informationen aus dem Internet sind dem(r) Lehrer(in) vollständig zur Verfügung gestellt worden.

Rinteln, der 19. März 2023 _____

Connor Schweighöfer

6 Veröffentlichungserklärung

Hiermit erkläre ich, dass ich damit einverstanden bin, wenn die von mir verfasste Facharbeit der schulinternen Öffentlichkeit zugänglich gemacht wird.

Rinteln, der 19. März 2023 _____

Connor Schweighöfer