University of Oslo

FYS3150

Computational Physics

Modelling the Solar System
A Numerical Study of Differential Solvers

Authors
Lars Kristian Skaar
Håkon Lindholm
Brage Brevig
https://github.uio.no/lkskaar/FYS3150

October 28, 2020

1 Abstract

The scope of this numerical study is two-fold; for one we wish to delve into the study of three popular algorithms for solving ordinary differential equations, namely the Euler - Forward and Euler - Cromer methods, as well as the Velocity Verlet method. Secondly, it is an exercise in object oriented programming, in which the end result is a working model for our solar system. In our studies, we have concluded that the Verlocity Verlet algorithm is the optimal choice for solving the problem at hand, as it commits an error decreasing twice as fast as the errors committed by the Euler methods for increasing temporal resolution, all the while having an insignificantly longer run time compared to the Euler methods, differing by roughly 10%. Secondly, we find that devising three main classes in our program, it is very easy to extend the problem to large systems, and more interestingly to extend our program to solve different physical systems seamlessly. In order to survey how realistic our model is, we have conducted several simulations in which we may compare our numerical results to observed and theoretical values, all which produce results enforcing the credibility of our model.

Contents

2 Introduction

Setting out, this numerical project has a three-fold goal; studying three differential equation solvers, designing and devising an object oriented code and lastly making a model for our Solar System. We choose to look at this project as an exercise in object orientation, as this is an important programming model which allows developers to design software in which large data bases may be treated seamlessly and with high precision - when a certain data type, or *object* is known, its inherent properties and behaviour may be cast into a class of objects displaying similar properties or behaviour. This allows the same logical structures to be devised across different, yet similar objects. In the first part of this article, we shall more formally define the problem at hand and how we object orient our code and implement the numerical schemes for solving N - body motion in a system of celestial bodies. In this part we also explain how we have performed surveys of our model using logical test functions in lack of analytical solutions we may compare our model against. We later go on to present the myriad of results we have procured in this project and discuss these in light of the goals of the study. We will do this in an orderly fashion. Lastly we make some conclusive remarks on our findings and shed light on prospective changes.

3 Theory & Background

3.1 Newton's Universal Law of Gravitation

In this study, we assume that all massive bodies are only under the influence of the gravitational force exerted on them by other massive bodies - a force which is modelled by Newton's universal law of gravitation on its most general vector form

$$\mathbf{F}_{ij} = -\frac{GM_iM_j}{|r_{ij}|^3}\mathbf{r}_{ij} \tag{1}$$

for some massive body i and j. Here, G is the gravitational constant[1], $|r_{ij}|$ is the vector distance between the two bodies, and $\mathbf{r}_{ij}$ is the radial vector pointing from body i to body j. The minus sign indicates that

[1] Its value in SI units is $6.67{\cdot}10^{-11}$ m^3 kg^{-1} s^{-2}

$\mathbf{F}$ is attractive. Should the massive body i be in proximity of N other massive bodies, its motion will be influenced by the gravitational force exerted on it by all of them, such that

$$\mathbf{F}_i^{(\mathrm{NET})} = -\sum_{j=1}^{N} \frac{GM_iM_j}{|r_{ij}|^3}\mathbf{r}_{ij}$$

where $\mathbf{F}_i^{(\mathrm{NET})}$ should be read as the net force acting on a massive body i. By Newton's second law of motion we may easily determine the motion of any massive body i by

$$\frac{\partial^2}{\partial t^2}\mathbf{r}(t) = -\frac{1}{M_i}\sum_{j=1}^{N} \frac{GM_iM_j}{|r_{ij}|^3}\mathbf{r}_{ij} = \frac{1}{M_i}\mathbf{F}_i^{(\mathrm{NET})} \tag{2}$$

The integration is however a two-step process, as for some velocity vector $\mathbf{v}_i(t)$ and some position vector $\mathbf{r}_i(t)$, both belonging to body i we have

$$\begin{cases} \frac{\partial}{\partial t}\mathbf{v}_i(t) = \frac{1}{M_i}\mathbf{F}_i^{(\mathrm{NET})} \\ \frac{\partial}{\partial t}\mathbf{r}_i(t) = \mathbf{v}_i(t) \end{cases} \tag{3}$$

where we have devised the fundamental laws of motion[2]. The equations in (3) are easily implemented on a computer.

3.2 A Numerical Scheme for the Equations of Motion

Let $p_k := \mathbf{r}(t_k)$ and $u_k := \mathbf{v}(t_k)$ define numerical solutions to the differential equations in (3), where $t_k = t_0 + h$ for some temporal step size $h = t_N/N$ where N is a chosen number of numerical grid points such that $k = 1, 2, 3, ..., N$. We may then discretize the equations in (3) by

$$\begin{cases} u_{k+1}^{(i)} - u_k^{(i)} = (h/M_i)\mathbf{F}_i^{(\mathrm{NET})}(t_n, u_k) \\ p_{k+1}^{(i)} - p_k^{(i)} = hu_k^{(i)}(t_n, p_k) \end{cases} \tag{4}$$

where the superscripts denote that a solution belongs to body i. The discrete equations in (4) are on a very general form, as the three different algorithms we shall encounter later treat them differently. We state them here, however, to give the reader a feeling of how we view the evolution of position and velocity numerically.

[2]Velocity is the rate of change of position, whilst acceleration is the rate of change of velocity.

3.3 Astronomical Framework and Units

The solar system is undoubtedly a computationally demanding architecture; it is extremely vast, and contains objects of incredible size. In order to reduce the risk of numerical overflow arising from arithmetic operations with very large numbers, we introduce the astronomical unit (AU) for length, in which the distance from the Earth to the Sun is defined as 1 AU. We shall also consider time in terms of Julian years, where one Julian year is defined as 365.25 days. Lastly, but not least, we shall define all celestial body masses in terms of the solar mass $M_\odot$, by letting $M_\odot := 1$.

Table 1: Tabular overview of observable values for each planet in the Solar System in astronomical units.

Celestial Body	Mass [kg]	Mass $[M/M_\odot]$	Distance to the Sun [AU]
Sun	$1.99{\cdot}10^{30}$	1	-
Mercury	$3.30{\cdot}10^{23}$	$1.66{\cdot}10^{-7}$	0.39
Venus	$4.87{\cdot}10^{24}$	$2.44{\cdot}10^{-6}$	0.72
Earth	$5.97{\cdot}10^{24}$	$3.00{\cdot}10^{-6}$	1.00
Mars	$6.40{\cdot}10^{23}$	$3.22{\cdot}10^{-7}$	1.52
Jupiter	$1.90{\cdot}10^{27}$	$9.54{\cdot}10^{-4}$	5.20
Saturn	$5.70{\cdot}10^{26}$	$2.86{\cdot}10^{-4}$	9.54
Uranus	$8.70{\cdot}10^{25}$	$4.37{\cdot}10^{-5}$	19.19
Neptune	$1.02{\cdot}10^{26}$	$5.13{\cdot}10^{-5}$	30.06
Pluto	$1.46{\cdot}10^{22}$	$7.34{\cdot}10^{-9}$	39.53

Working within this frame also allows us to deal with the gravitational constant in a much computationally friendly way. The first part of the project will be concerned with the Earth-Sun system, where the force F_G acting on the Earth from the Sun is given by

$$F_G = \frac{GM_\odot M_\oplus}{r^2} \tag{5}$$

where $M_\odot$ is the mass of the Sun, $M_\oplus$ is the mass of the Earth, and r is the length of the radial vector pointing from the Earth to the Sun. Here we also omit the minus sign, assuming it is known by now that F_G is attractive. Suppose now that the Earth is constrained to a perfectly circular orbit around the Sun. In this respect, we may observe that the Earth is held in orbit by some centripetal force such that

$$M_\oplus r\omega^2 = \frac{GM_\odot M_\oplus}{r^2}$$

where $\omega = 2\pi/T$ is the angular frequency of Earth and T is its orbital period being one Julian year.

Observe now that

$$G = \frac{4\pi^2 r^3}{T^2 M_\odot} \tag{6}$$

Since we have defined r, T and $M_\odot$ already, we have $G = 4\pi^2$ AU^3 M$_\odot^{-1}$ yr^{-2}. Similarly, we may define the centripetal force acting on Earth by $M_\oplus \nu^2 / r$, where ν is the orbital velocity of Earth, leading to

$$\nu^2 = \frac{GM_\odot}{r}$$

or $\nu = 2\pi$ AU yr^{-1} by the previous definitions.

3.4 Escape Velocity

The escape velocity of a planet in a simple binary planet-Sun system, is the minimum velocity required to escape the Sun's gravitational pull. This is achieved when the sum of the kinetic and potential energy of a planet is greater than zero. In this study we shall only devise this theory as a benchmark test to see whether or not our program correctly computes the motion of the Earth when only under the influence of the gravitational pull from the Sun, meaning that

$$\frac{1}{2} M_\oplus v^2 - G\frac{M_\odot M_\oplus}{r^2} r \geq 0 \tag{7}$$

$$\tag{8}$$

$$v \geq \sqrt{2G\frac{M_\odot}{r}} \tag{9}$$

By inserting G from (6), $M_\odot := 1$ and $r = 1$, we find that theoretically that a minimum velocity of $\sqrt{8}\pi$ is needed in order for the Earth to escape its orbit around the Sun. We may now use this to see at what initial velocities the Earth 'stays or strays' in the model we shall implement numerically, and compare this to the theoretical value in order to validate the model.

3.5 The Perihelion Precession of Mercury

When Albert Einstein founded the General Theory of Relativity, he proposed three important tests to the theory - amongst which one is to explain the anomalous precession of Mercury's perihelion[3] as it orbits the Sun [1]. The currently accepted value of Mercury's perihelion precession is 43" (arc seconds) per century, a value which classical Newtonian laws fail to predict. In fact, for some time

[3]The perihelion is the spatial point on an objects orbit where it is closest to the Sun.

before Einstein's work, esteemed astronomers like Le Verriere proposed that unobserved 'ghost' planets - or a conglomerate of non-planetary massive objects - within Mercury's orbit was causing its orbital precession, as their observations of Mercury was incompatible with their classical predictions [2]. In this project we shall study the orbit of Mercury around the Sun in light of its relativistic motion given by

$$F_G = \frac{GM_\odot M_{Mercury}}{r^2}\left[1 + \frac{3l^2}{r^2c^2}\right] \tag{10}$$

where r is the distance between Mercury and the Sun, c is the speed of light in vacuum and $l = |\mathbf{r} \times \mathbf{v}|$ is the magnitude of Mercury's orbital angular momentum per unit mass[4]. Closed elliptical orbits is a special feature of the inverse square law described in equation (1). Adding corrections to this law causes open orbital behaviour in which objects on the orbit will displace in each orbital period. Very small corrections, such as the one in equation (10), will lead to nearly closed elliptical orbits. In this respect, we may consider the orbit of Mercury to slowly precess around the Sun. This phenomenon is difficult to analyse visually, though, which is why it will be useful to calculate the angle of Mercury's perihelion

$$\vartheta_p = \arctan\left(\frac{y_p}{x_p}\right) \tag{11}$$

during each of Mercury's orbital periods and rather study this relationship.

4 Numerical Methods and Implementation

4.1 Implementation and Class Hierarchy

As was mentioned in the introduction of the article, the goal of this numerical project is not only to study differential solvers and the solar system, but also an exercise in designing object oriented code. We therefore see it as necessary to briefly explain the class hierarchy of our program, and how this implementation simultaneously is the foundation for an object oriented, 'recyclable' code which may be extended to study systems beyond our Solar System.

Our program is based on three abstract classes; `System`, `Planet` and `Solver`. The class `System` contains settings needed in place to add objects to a list. Said objects are then contained in the system we wish to study. The more interesting class `Planet` constructs and declares the planets - or objects - we wish to add to our system. Each planet has three fundamental attributes - position, velocity and mass - all

[4] In order to get dimensionless units.

which are defined in this class. This class illustrates why our program could be cast into different physical systems. Given that a system contains objects of similar attributes as the planets we are studying in this project, they would be easily defined numerically in `Planet` - a class which by all means could be called `Particles` or `Objects`. It should be noted, however, that Newton's law of gravitation is implemented in `Planet` as the force acting on a planet from all other planets in the same system. This would naturally have to be changed given that a different physical system is under the influence of a different (or several other) forces when re-devising the program. Lastly, the class `Solver` is where the magic happens. In this class, the `EulerForward()`, `EulerCromer()` and `VelocityVerlet()` - solvers are all declared. This class inherits objects from `Planet` and systems of said objects from `System`, allowing us to perform operations on them to produce results.

4.2 A Psuedocode for Calculating the Net Force on a Given Planet

Going forward, we shall shed light on the numerical implementation of the three algorithms we are studying. In each of them, the calculation of the net force acting on a given planet in a given system acts as the backbone of the algorithm, which is why we introduce it here. We will also present the implementation of each algorithm as psuedocode, which is why we encourage the reader to visit our GITHUB repository linked on the first page in the article to view all code.

Algorithm 1: Calculate Net Force

input : System

output: Net force on Planet ($\mathbf{F} \in \mathbb{R}^3$)

Var:

i, n, planet_idx, otherPlanet_idx, System.TotalPlanets $\in \mathbb{N}$

System.AllPlanets $\in \mathbb{N}^{1\times P}$, $P =$ Number of planets in System

for i **to** $n-1$ **do**

  for *Planet_ idx* **to** *System.TotalPlanets-1* **do**

    current_planet = System.AllPlanets[planet_idx]

    for *otherPlanet_ idx* **to** *System.TotalPlanets - 1* **do**

      current_planet.Force += Force_From_OtherPlanet

    end

  end

end

As we see in Algorithm 1, it takes `System` as input, in which there exists information about the total number of planets we are dealing with (`System.TotalPlanets`) and a ordered list of planets (`System.AllPlanets`).

`Force_From_OtherPlanet` is naturally the gravitational force exerted from one planet onto another as in equation 1 at the position the planet is currently in.

4.3 The Euler-Forward Algorithm

The Euler-Forward algorithm may be given as a set of discrete recursive relations as

$$\begin{cases} \mathbf{r}_{k+1} = \mathbf{r}_k + h\mathbf{v}_k \\ \mathbf{v}_{k+1} = \mathbf{v}_k + (h/M_i)\mathbf{F}_i^{(\text{NET})} \\ \mathbf{r}_0 = (x_0, y_0, z_0), \quad \mathbf{v}_0 = (v_0^{(x)}, v_0^{(y)}, v_0^{(z)}) \end{cases} \tag{12}$$

Counting the number of FLOPs to $5N$, we may observe that this implementation comes at low computational cost. One could compute the acceleration outside of the algorithm to reduce the number of FLOPs to $4N$. This however only applies to the iterative recursion, not the program itself! In reality, one should also count the FLOPs involved in calculating the gravitational force between planets to get a true value of the number of FLOPs for the algorithm in its entirety. The FLOPs we present are however representative of the order of the number of FLOPs, and are really only used for comparison between the different algorithms. The Euler - Forward method also carries a global truncation error $\mathcal{O}(h)$, ensuring a linear relationship between the numerical error and the step size h. It should also be noted that the Euler-Forward method is *non-symplectic*, meaning that it does not belong to the class of energy-conserving integrators. We shall see the consequence of this in the Results - section. The algorithm may be implemented as

Algorithm 2: Euler-Forward Algorithm

input : System

output: Current planet position at time t_i ($\mathbf{r} \in \mathbb{R}^3$)

Var:

h $\in \mathbb{R}$

top: current_planet.Force $\rightarrow$ Calculate_Net_Force()

update curret_planet.position $\rightarrow$ h·current_planet.velocity

update current_planet.velocity $\rightarrow$ h·current_planet.Force/current_planet.mass

goto: **top**

4.4 The Euler - Cromer Algorithm

The Euler - Cromer -, or semi-implicit Euler method is a first order symplectic algorithm, meaning that it commits a global truncation error of $\mathcal{O}(h)$. Its symplecticity is said to *almost* conserve physical quantities in conservative fields, making it a suitable candidate for solving the problem at hand. This algorithm may be implemented as a set of discrete recursive relations by

$$\begin{cases} \mathbf{v}_{k+1} = \mathbf{v}_k + (h/M_i)\mathbf{F}_i^{(\text{NET})} \\ \mathbf{r}_{k+1} = \mathbf{r}_k + h\mathbf{v}_{k+1} \\ \mathbf{r}_0 = (x_0, y_0, z_0), \quad \mathbf{v}_0 = (v_0^{(x)}, v_0^{(y)}, v_0^{(z)}) \end{cases} \tag{13}$$

The number of FLOPs required to performed N iterations with the Euler-Cromer scheme is, as for the Euler - Forward scheme $5N$, making it a low cost algorithm, too. The schemes are remarkably similar, as the major difference is that in the Euler Cromer scheme, the position vector is overwritten using the velocity vector one time step ahead, whilst in the Euler Forward scheme it is overwritten using the current velocity vector. This algorithm is implemented as

Algorithm 3: Euler-Cromer Algorithm

input : System

output: Current planet position at time t_i ($\mathbf{r} \in \mathbb{R}^3$)

Var:

h $\in \mathbb{R}$

top: current_planet.Force $\rightarrow$ Calculate_Net_Force()

update current_planet.velocity $\rightarrow$ h·current_planet.Force/current_planet.mass

update curret_planet.position $\rightarrow$ h·current_planet.velocity

goto: **top**

4.5 The Velocity Verlet Algorithm

Now, onto the Velocity Verlet algorithm. This algorithm belongs to a whole class of Verlet integrators, and has the initial advantage over the Euler schemes of being a second order symplectic integrator. This means that, in contrasts to the Euler schemes, the Verlet algorithm commits a global truncation error of $\mathcal{O}(h^2)$ such that a ten-fold increase in the temporal resolution leads to a hundred-fold increase in the numerical precision. Simultaneously, its symplecticity ensures conservation of physical quantities in conservative fields. Adding to the list of why the Verlet schemes are interesting, they allow for time-

reversibility as the first and final step of the scheme are exactly the same. This means that the integrator could potentially be extended to Hamiltonian systems in which negative time exists! However, departing from the small side track, the Velocity Verlet algorithm may be implemented as a set of discrete recursive relations as

$$\begin{cases} \mathbf{r}_{k+1} = \mathbf{r}_k + h\mathbf{v}_k + (h^2/2)\mathbf{a}_k \\ \mathbf{v}_{k+1} = \mathbf{v}_k + (h/2)\left(\mathbf{a}_{k+1} + \mathbf{a}_k\right) \\ \mathbf{r}_0 = (x_0, y_0, z_0), \quad \mathbf{v}_0 = (v_0^{(x)}, v_0^{(y)}, v_0^{(z)}) \\ \mathbf{a}_0 = -\sum_{j=1}^{N} GM_j(\mathbf{r}_{ij}^{(0)}/|\mathbf{r}_{ij}^{(0)}|^3) \end{cases} \tag{14}$$

Here, $\mathbf{r}_{ij}^{(0)}$ should be read as the radial vector pointing from body i to body j when body i is in its initial position $\mathbf{r}_0 = (x_0, y_0, z_0)$. Counting the number of FLOPs, we see that this algorithm commits a total number of $9N$ FLOPs. Comparing this against the number of FLOPs committed by the Euler schemes, we see that the Velocity Verlet method is slightly less efficient. The following psuedocode gives an idea of how this is implemented on a computer

Algorithm 4: Velocity Verlet Algorithm

input : System

output: Current planet position at time t_i ($\mathbf{r} \in \mathbb{R}^3$)

Var:

h $\in \mathbb{R}$

top: current_planet.Force $\rightarrow$ Calculate_Net_Force()

update current_planet.position $\rightarrow$ h·current_planet.velocity + $(h^2/2)$current_planet.Acceleration

set curret_planet.Force_New $\rightarrow$ Calculate_Net_Force.at(current_planet.position)

update current_planet.velocity $\rightarrow$ $h/2$(current_planet.Acceleration_New + current_planet.Acceleration)

goto: top

4.6 Unit Tests and Logical Test Functions

Unit tests are an essential part of program development as they ensure each part of the implementation produces the results it is supposed to. For this project, we have implemented many such tests, but none of these are worth paying attention to in this article - consult `unittests.cpp` under the Code-folder in our GITHUB - repository to see these tests.

It is worthwhile however, to briefly explain how and why we have implemented logical test functions in our program. In the Results - section we exhibit data about the numerical errors associated with each of the three solvers we study in this project, and a natural question should be how this error computation was performed. Consider the simple Earth-Sun system in which we assume the Earth's orbit is circular. We study this system with good reason, as it allows us not only to compare run times for large N within a suitable time frame, but also because it allows us to design a logical operation for determining the maximum relative error in order to compare numerical precision. Given a circular orbit, the length of Earth's position vector should for each time step have a magnitude of one. Using this, our program contains a test function `TEST_circular()` which compares the length of each position vector and the theoretical length. The absolute value of this difference is what we take to be a numerical error. The program then goes on to find the maximum error and returns this.

Furthermore, our program incorporates Kepler's second law to determine whether the angular momentum of a planet's orbit is conserved. The magnitude of the position vector relative to the Sun is stored for the previous and current time step, along with the distance travelled along the orbit for the time step. The area of the triangle these vectors make is stored for each time step, and compared to the area of the previous triangle in the function `TEST_kepler()`. The areas are compared, and if they differ by more than a set tolerance[5], the angular momentum for the system is not conserved.

5 Results

All simulations and consequent plots and figures up until Figure 16 has been done with the Sun fixed to the origin, unable to move as a result of the force acting on it from other bodies in the system.

5.1 Numerical error and run times

Figure 1 and Table 2 both illustrate the numerical efficiency of each of the different solvers we have studied in this project. The main observation here is that the elapsed run times follow a linear relationship with the number of integration points which is what we would expect from the number of FLOPs each solver carries with them. Second to that we see that there are very small differences in the data, reflecting the relatively small differences in FLOPs for each algorithm.

[5] In our case, we have set a tolerance of 10^{-10}

Table 2: Tabular overview of sequential run times for increasing number of integration points N for each solver when solving the motion equation for Earth in the Earth - Sun system.

N	Euler-Forward [s]	Euler-Cromer [s]	Velocity Verlet [s]
10^1	$3.04 \cdot 10^{-4}$	$1.57 \cdot 10^{-4}$	$2.68 \cdot 10^{-4}$
10^2	$9.29 \cdot 10^{-4}$	$6.80 \cdot 10^{-4}$	$7.23 \cdot 10^{-4}$
10^3	$6.70 \cdot 10^{-3}$	$6.89 \cdot 10^{-3}$	$8.50 \cdot 10^{-3}$
10^4	$5.96 \cdot 10^{-2}$	$6.18 \cdot 10^{-2}$	$6.65 \cdot 10^{-2}$
10^5	$5.98 \cdot 10^{-1}$	$5.64 \cdot 10^{-1}$	$6.54 \cdot 10^{-1}$
10^6	$5.98 \cdot 10^{0}$	$5.58 \cdot 10^{0}$	$6.79 \cdot 10^{0}$
10^7	$6.60 \cdot 10^{1}$	$6.59 \cdot 10^{1}$	$6.75 \cdot 10^{1}$

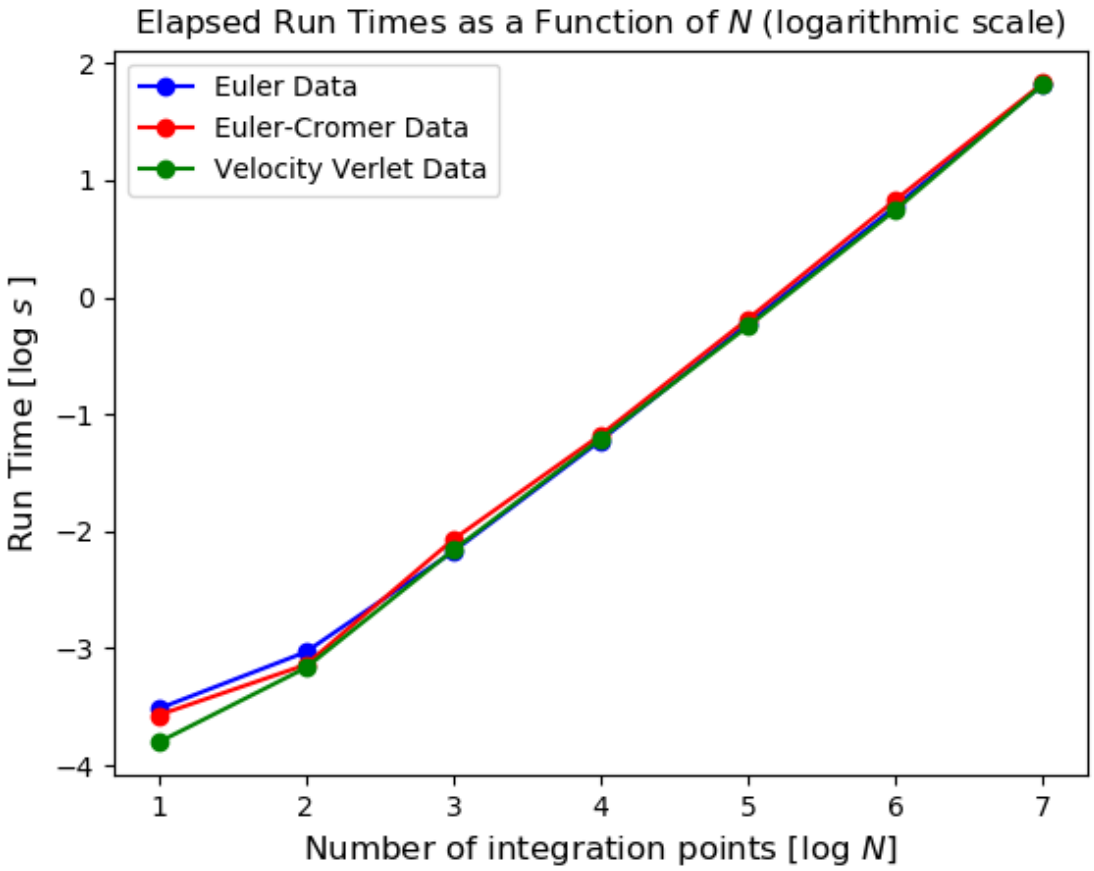


Figure 1: A plot showing the relationship between the number of integration points N and the elapsed run times for each of the three solvers as to illustrate the different solvers' efficiency.

Table 3: Tabular overview of the maximum numerical error for a given number of integration points N for each solver when solving the motion equation for Earth in the Earth - Sun system.

N	Euler-Forward	Euler-Cromer	Velocity Verlet
10^1	$4.19 \cdot 10^{0}$	$5.31 \cdot 10^{-1}$	$1.87 \cdot 10^{-1}$
10^2	$7.17 \cdot 10^{-1}$	$3.34 \cdot 10^{-2}$	$1.97 \cdot 10^{-3}$
10^3	$7.69 \cdot 10^{-2}$	$3.14 \cdot 10^{-3}$	$1.97 \cdot 10^{-5}$
10^4	$7.87 \cdot 10^{-3}$	$3.15 \cdot 10^{-4}$	$1.97 \cdot 10^{-7}$
10^5	$7.89 \cdot 10^{-4}$	$3.29 \cdot 10^{-5}$	$1.97 \cdot 10^{-9}$
10^6	$7.89 \cdot 10^{-5}$	$1.74 \cdot 10^{-5}$	$1.97 \cdot 10^{-11}$
10^7	$1.40 \cdot 10^{-5}$	$2.02 \cdot 10^{-5}$	$7.65 \cdot 10^{-14}$

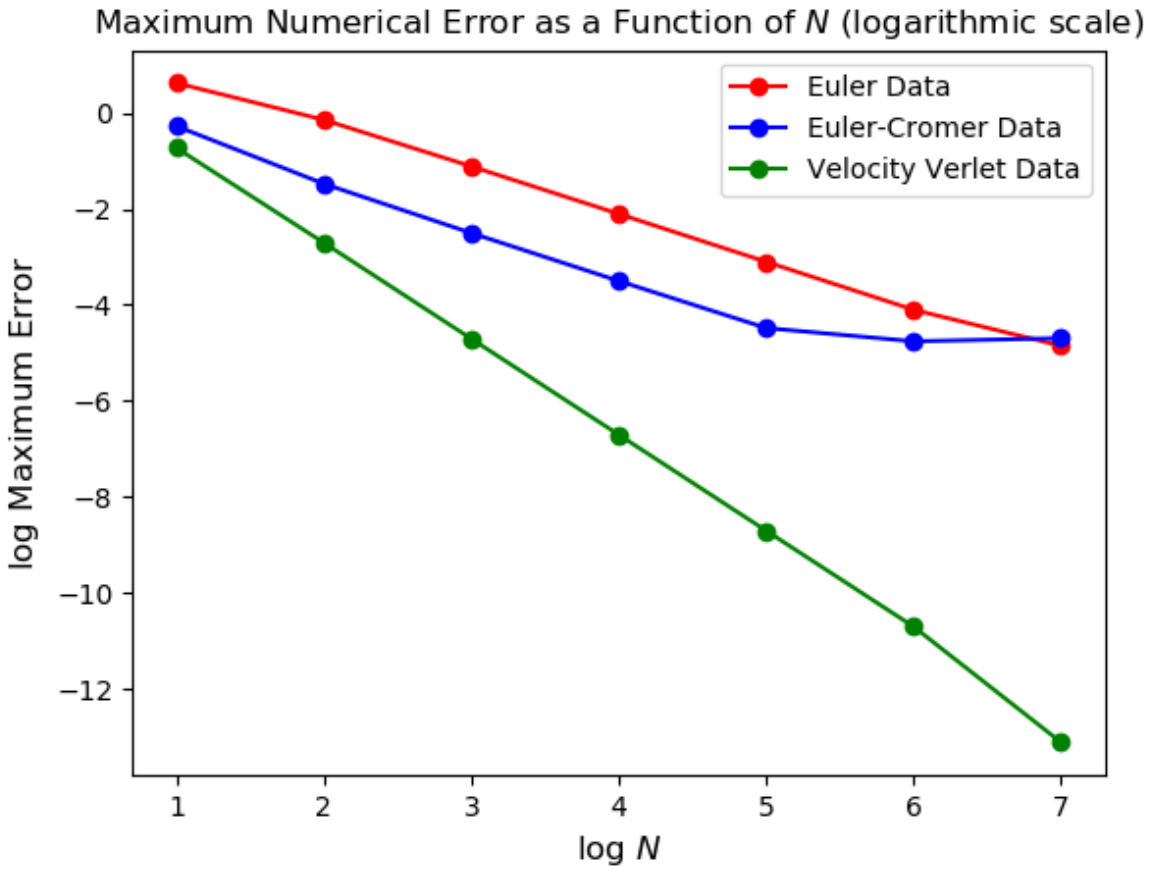


Figure 2: A plot showing the relationship between the number of integration points N and the maximum numerical error for each of the three solvers as to illustrate the different solvers' precision.

Table 4: Tabular overview of the maximum numerical error for a given number of years for each solver when solving the motion equation for Earth in the Earth - Sun system with a constant $N = 10^5$.

Years	Euler-Forward	Euler-Cromer	Velocity Verlet
10^0	$7.89 \cdot 10^{-4}$	$3.29 \cdot 10^{-5}$	$1.97 \cdot 10^{-9}$
10^1	$7.33 \cdot 10^{-2}$	$3.15 \cdot 10^{-4}$	$1.97 \cdot 10^{-7}$
10^2	$1.94 \cdot 10^{0}$	$3.14 \cdot 10^{-3}$	$1.97 \cdot 10^{-5}$
10^3	$2.58 \cdot 10^{1}$	$4.2 \cdot 10^{-2}$	$1.97 \cdot 10^{-3}$
10^4	$4.16 \cdot 10^{4}$	$3.25 \cdot 10^{3}$	$1.88 \cdot 10^{-1}$
10^5	$3.99 \cdot 10^{6}$	$3.99 \cdot 10^{6}$	$2.06 \cdot 10^{6}$

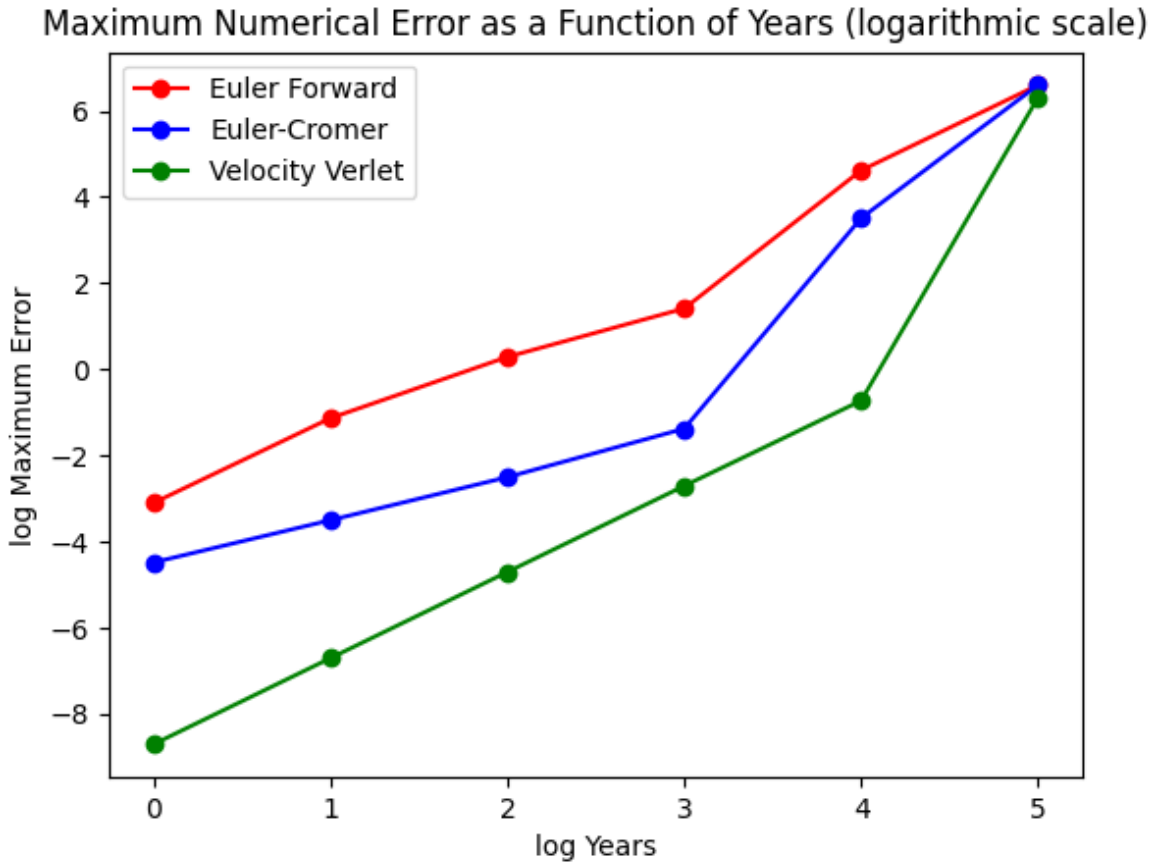


Figure 3: A plot showing the relationship between the number of Years and the maximum numerical error for each of the three solvers

Figure 2 and Table 3 illustrate the numerical precision of each of the three solvers. The data in Table 3 is in itself implicative of the Verlet - algorithms superiority in this setting. As was mentioned in the sections for each of the different algorithms, the two Euler methods both commit a first-order global truncation error $\mathcal{O}(h)$, whilst the Velocity Verlet method commits a second-order global truncation error $\mathcal{O}(h^2)$ - these are all reflected in Figure 2. It is easy to extract by visual observation that the error in the two Euler methods decreases linearly with a gradient close to one, and that the error in the Verlet method decreases twice as fast with a gradient close to two. The computed gradients are -0.94, -1.04 and -2.04 for the Euler-Forward, Euler-Cromer and Velocity Verlet algorithms respectively. Here we have omitted the anomalous data for the error in the Euler-Cromer method for $N > 10^5$ when making linear fits.

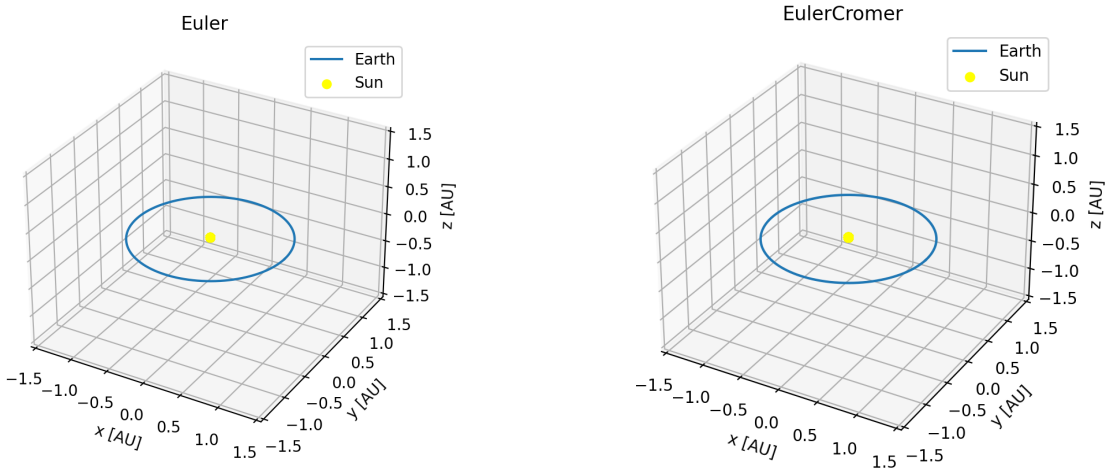


Figure 4: 3D plots of Earth's annual orbit around the Sun as calculated by the Euler - Forward (left) and Euler - Cromer (right) algorithms for $N = 10^6,\ h = 10^{-6}$.

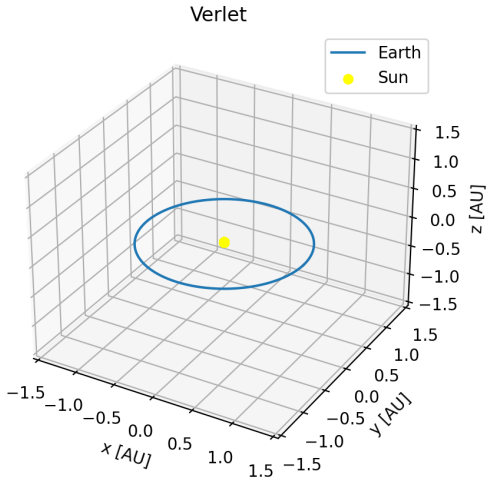


Figure 5: 3D plot of Earth's annual orbit around the Sun as calculated by the Velocity Verlet algorithm for $N = 10^6,\ h = 10^{-6}$.

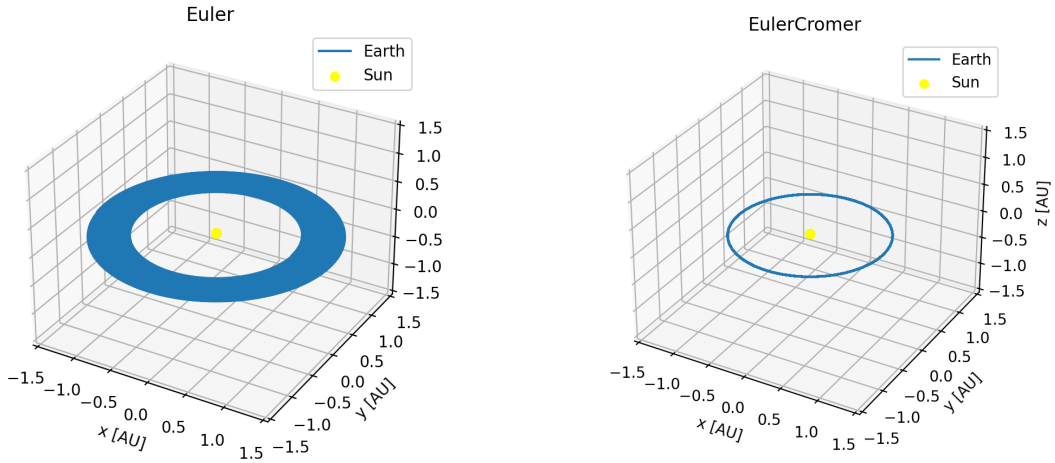


Figure 6: 3D plots of Earth's centennial orbit around the Sun as calculated by the Euler - Forward (left) and Euler - Cromer (right) algorithms for $N = 10^6,\ h = 10^{-4}$.

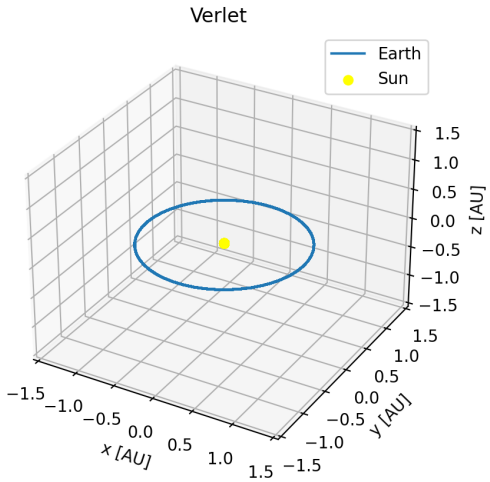


Figure 7: 3D plot of Earth's centennial orbit around the Sun as calculated by the Velocity Verlet algorithm for $N = 10^6,\ h = 10^{-4}$.

Figures 4 and 5 are simulations of Earth's annual motion around the Sun for a high temporal resolution $h = 10^{-6}$. In these plots, we should observe that the numerical precision is visually indistinguishable for the three solvers, and we would have to consult Tables 3 and 4 to conclude that the Velocity Verlet algorithm indeed produces the most precise result. Now, Figures 6 and 7 are simulations of Earth's centennial motion around the Sun, with a lower temporal resolution $h = 10^{-4}$. We immediately see that the Euler-Forward algorithm is non-symplectic as stated in Section 4.2. The orbital "width" indicates a motion in which the Earth strays from its (initialized) circular orbit of radius 1. The symplectic (energy conserving) Euler-Cromer and Velocity Verlet algorithms are still stable for this temporal resolution, and we see from Tables 3 and 4, as well as Figures 2 and 3 that a simulation of Earth's millennial motion in the Earth-Sun system would result in noticeably imprecise system, especially when devising the Euler-Cromer algorithm.

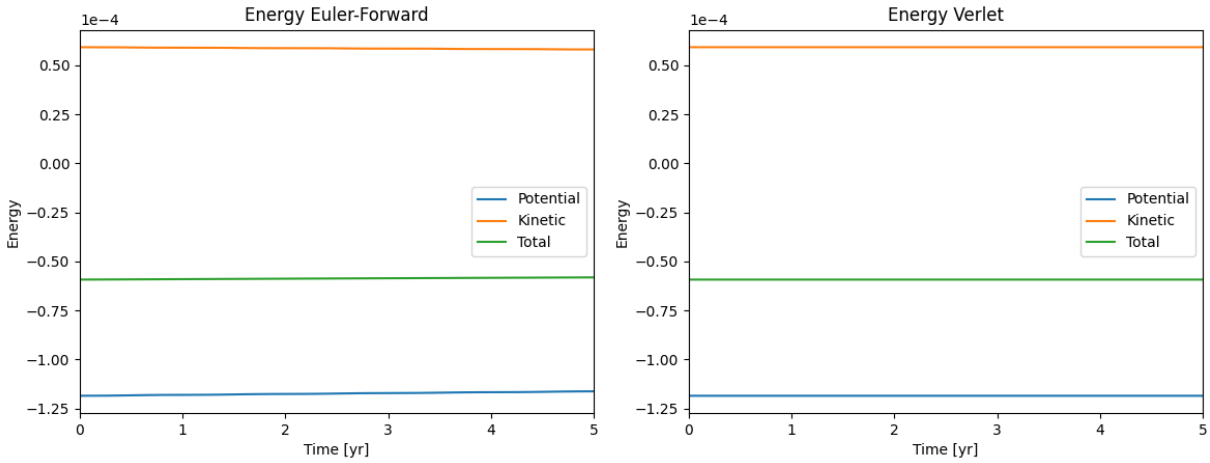


Figure 8: Plots of kinetic, potential and total energy for the Earth-Sun system for Euler-Forward (left) and Velocity Verlet (right) with $h = 5 \cdot 10^{-5}$.

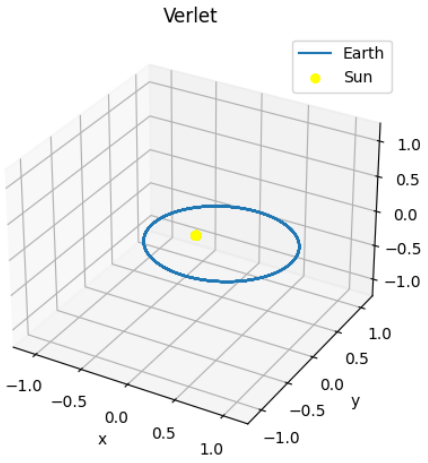
Figure 8 exhibit the non-symplecticity of the Euler-Forward scheme, as can be seen, just by putting some good will to use, by the increase in total energy. For the Velocity Verlet algorithm, the total energy is conserved.

Table 5: Tabular overview of the time step needed to have conservation of angular momentum with a tolerance of 10^{-10} for the Earth-Sun system.

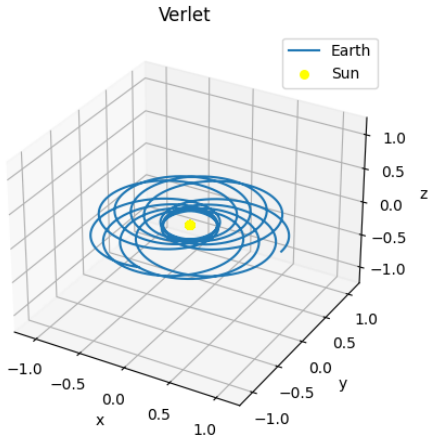
Years	h
1	$2.3 \cdot 10^{-3}$
10	$2.3 \cdot 10^{-3}$
100	$2.1 \cdot 10^{-3}$
1000	$1.1 \cdot 10^{-3}$

Table 5 shows roughly what step length is needed to achieve conservation of angular momentum for the Earth-Sun system.

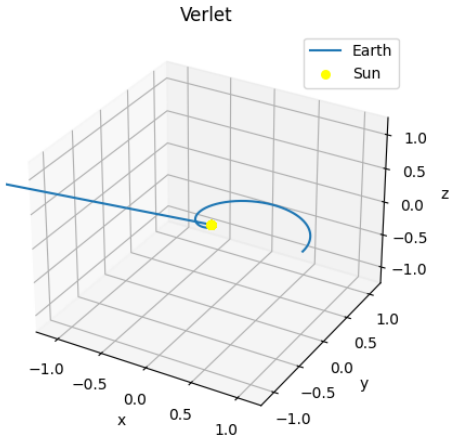
The program has been written such that a value β can be passed. This value corresponds to the exponent of r in the gravitational force term. Values of $\beta \in [2, 3]$ have been tested, in order to find out whether there can be forces other than the inverse-square force sustaining a stable Earth-Sun system.



(a) $\beta = 2$



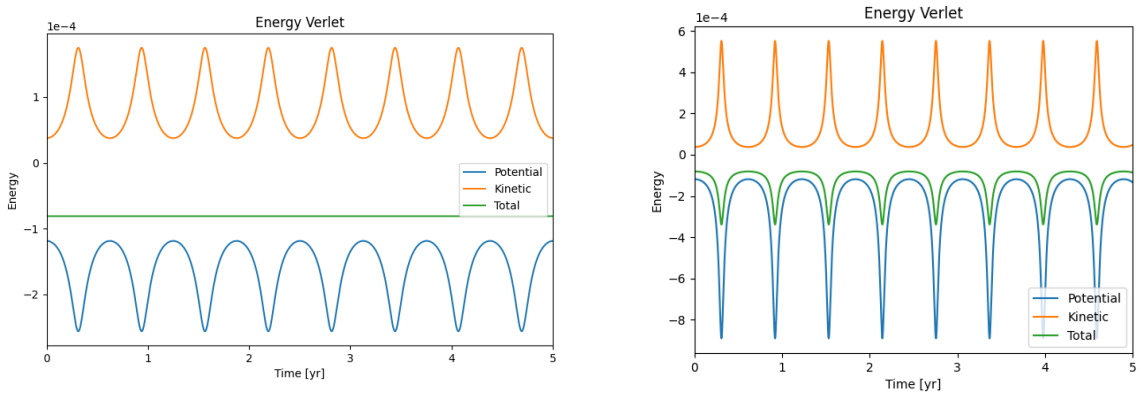
(b) $\beta = 2.5$



(c) $\beta = 3$

Figure 9: Plots of the Earth-Sun system with a gravitational force $F \propto 1/r^{\beta}$, where $\beta \in [2, 3]$. Earth's initial position is (0, 1 AU, 0) and velocity (5 AU/yr, 0, 0)

In Figure 9, Earth is initiated with an elliptical orbit. For $\beta = 2.5$, the orbit looks chaotic and far from the observed orbit. For $\beta = 3$, Earth has a rapidly declining orbit, and since our program has no functionality to detect planets colliding, Earth moved past the Sun's center, and experiences a huge force which throws it out of the Solar system in the other direction.



(a) $\beta = 2$

(b) $\beta = 2.5$

Figure 10: Energies of elliptical orbit with varying β

An inspection of the total energies (Figure 10 for the systems in Figure 9 reveals that with $\beta = 2.5$, the energy for the system is not conserved. It is however conserved with $\beta = 2$, as expected. We consider now a varying β with an Earth initiated with a circular orbit.

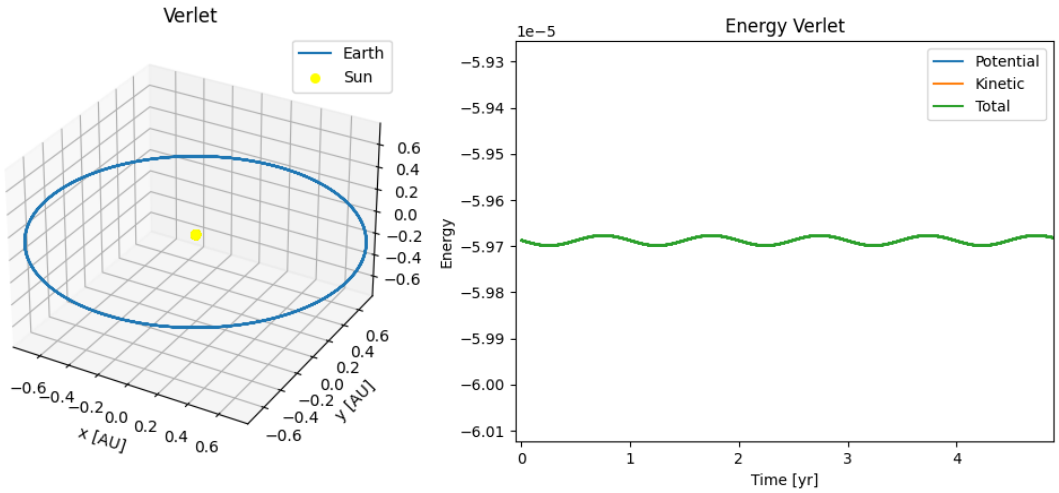
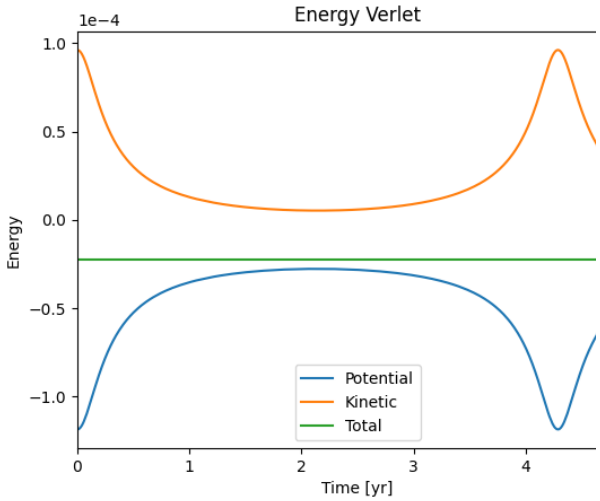


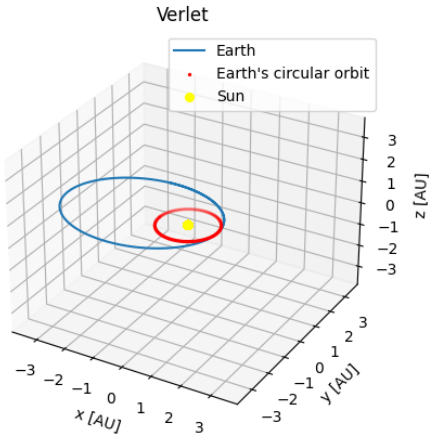
Figure 11: Plots of position (left) and energy (right) for Earth-Sun system with NASA data for initial position and velocity and $\beta = 2.01$

The position plot in Figure 11 does not raise cause for concern upon inspection. The energy plot tells a different story, as the total energy for the system can be seen to fluctuate in an oscillating fashion; the energy for the system is not conserved. We see no reason to include plots for higher values of β, as this energy fluctuation only increase in magnitude for increasing β. The program output also shows that angular momentum is not conserved for $\beta > 2$, and this is true for however small step size is chosen.

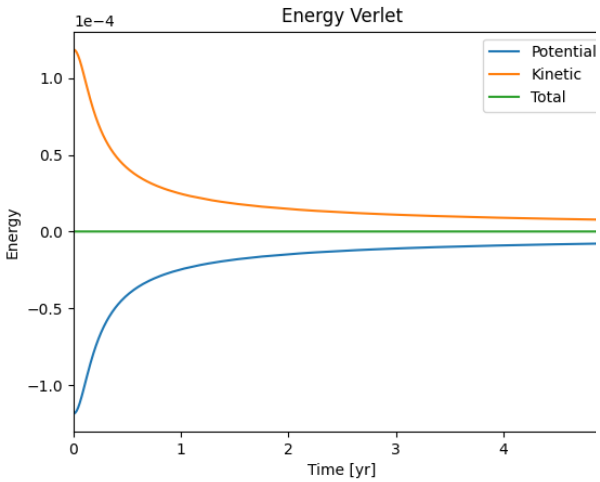
5.2 Escape Velocity



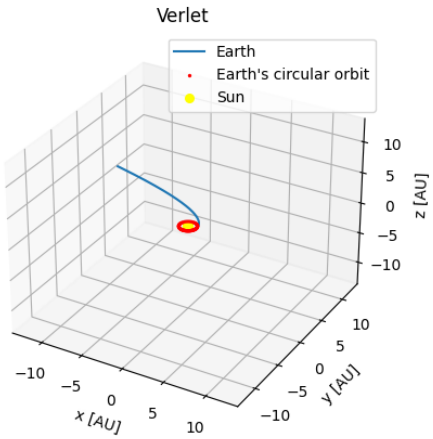
(a) Energy with V_0 = 8 AU/yr



(b) Position with V_0 = 8 AU/yr



(c) Energy with V_0 = 8.89 AU/yr



(d) Position with V_0 = 8.89 AU/yr

Figure 12: Plots of the position and energy for different initial velocities

Figure 12 shows two runs with different initial velocities for Earth. In the top two plots, Earth has a initial velocity in y-direction of 8 AU/yr. This produces an orbit with a semi major axis of more than 3 AU, but the Earth is still in orbit. In the bottom two plots, the initial velocity in y-direction is set to 8.89 AU/yr. This is below the analytical value found in the Theory-section, and as can be seen from the energy plot, the total energy is above 0. This means that Earth is able to overcome the Sun's gravitational pull and escape the Solar system.

5.3 Three-Body System

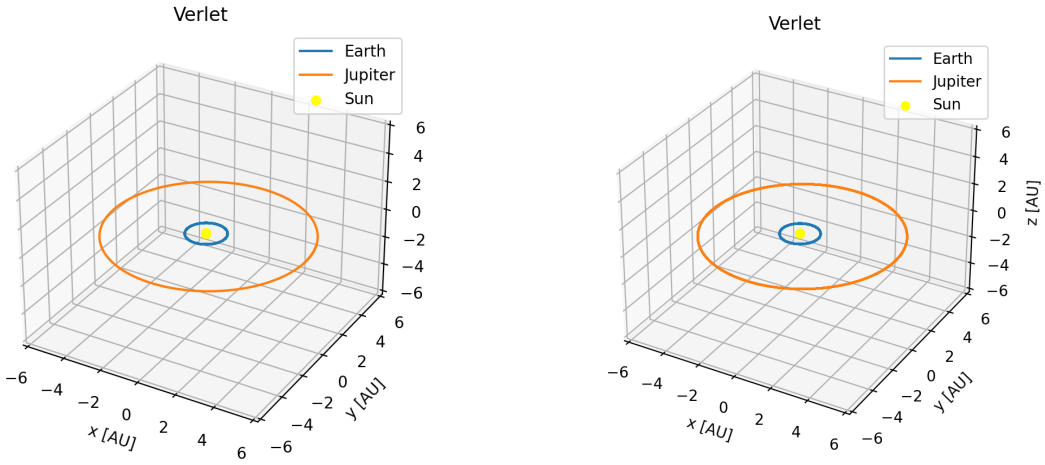


Figure 13: 3D plot of Jupiter's annual (left) and centennial (right) orbit around the Sun as calculated by the Velocity Verlet algorithm for respective temporal resolutions $h = 1.2 \cdot 10^{-5}$ and $h = 1.2 \cdot 10^{-4}$.

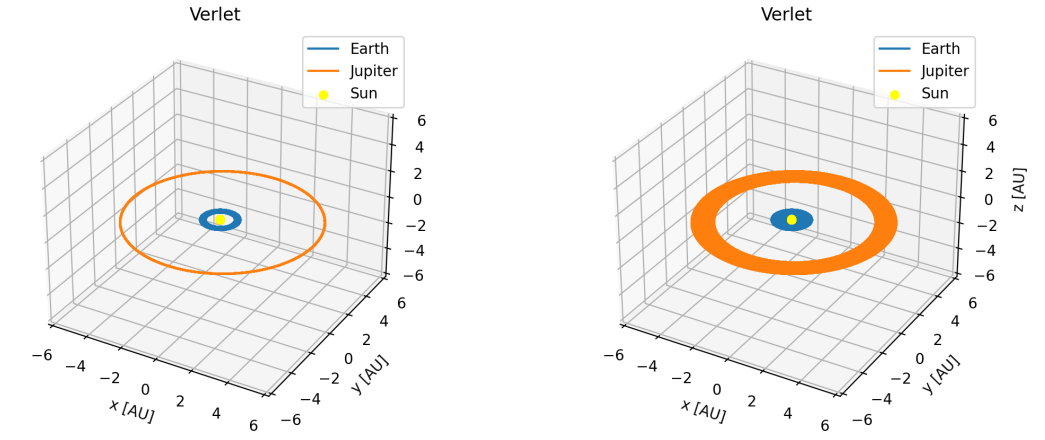
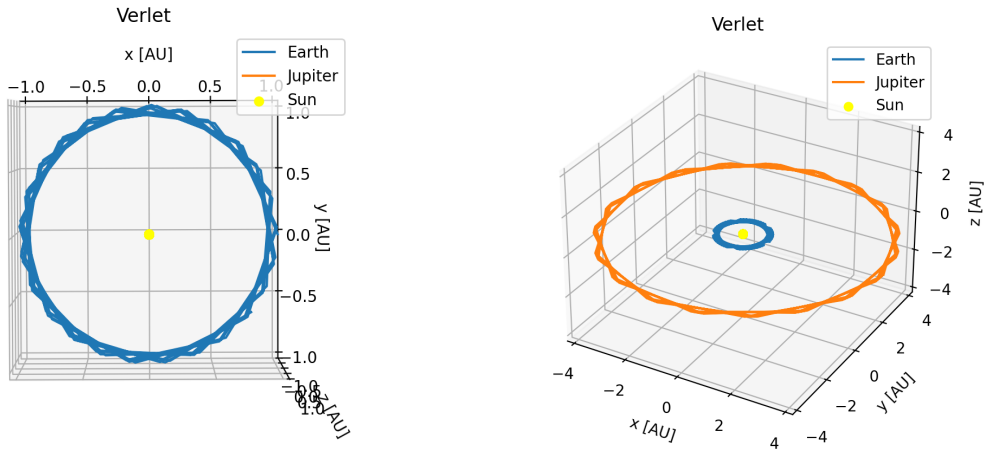


Figure 14: 3D plots of Jupiter's millennial (left) and 12th millennial (right) orbit around the Sun as calculated by the Velocity Verlet algorithm for respective temporal resolutions $h = 1.2 \cdot 10^{-3}$ and $h = 1.2 \cdot 10^{-2}$.

Figures 13 and 14 are results obtained from simulating the motion of both the Earth and Jupiter for a system in which the Sun is fixed to the origin, devising the Velocity Verlet algorithm each time. Here we may observe that, much like for the binary Earth-Sun system, the semi-like[6] three - body systems are well behaved for high temporal resolutions ($h < 10^{-3}$), and exhibit noticeable numerical imprecision for low temporal resolutions ($h > 10^{-3}$) as the orbital circumferences increase over time. Although this system is not a 'real' three-body system, it remains physically interesting as we may use it to study how Jupiter influences the motion of the Earth while still under the influence of the gravitational force it experiences from the Sun.



(a) 12 years

(b) 120 years

Figure 15: 3D plot of Earth's orbit while under influence of Jupiter (left) and Jupiter's orbit (right) with $h = 1.2 \cdot 10^{-5}$ and Jupiter's mass increased by a factor of 10

Figure 15 shows the system when Jupiter's mass is increased 10-fold. As can clearly be seen from the plots, this has a huge influence on Earth's orbit after 12 years, and an influence on Jupiter's orbit after 120 years.

[6] As the Sun is rigidly fixed, this is not a real three-body simulation, but the system exists however of three bodies.

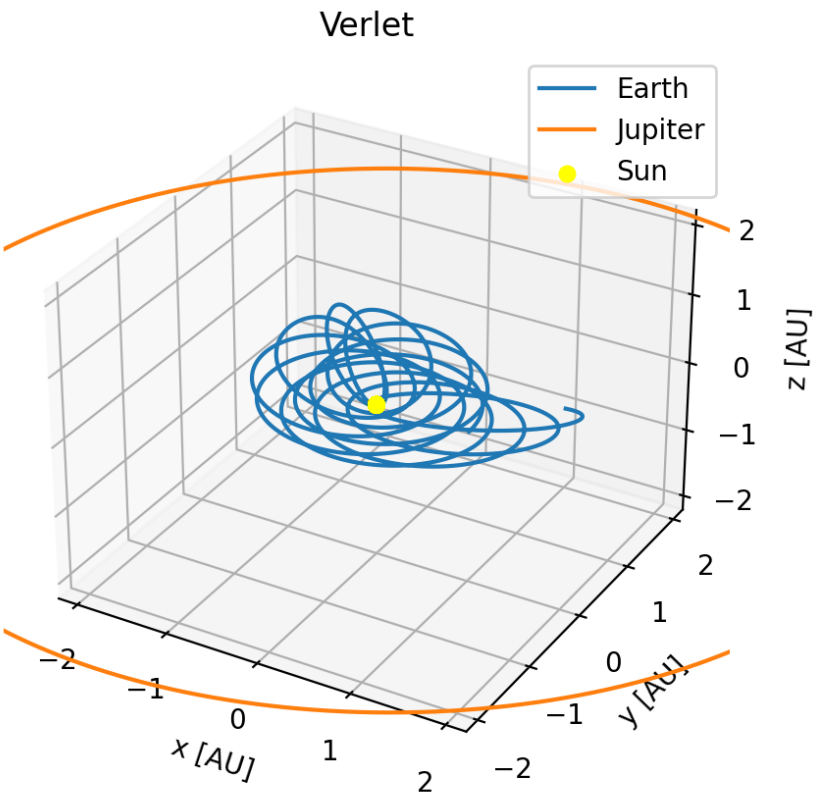


Figure 16: 3D plot of the three-body system with Jupiter's mass increased by a factor of 1000

From Figure 16, Earth's orbit quickly deteriorates, and we've elected to leave out results for runs with larger final times, as 12 years is sufficient to illustrate that the system is no longer stable.

5.4 The N-body System

Now we expand our model to contain all of the planets in the solar system and get the following result when we plot the position using the Velocity Verlet method:

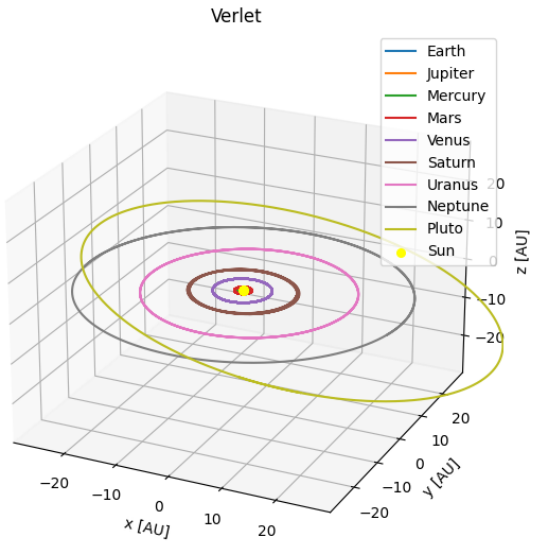


Figure 17: 3D of the entire Solar system, including Pluto. No moons have been included. Time-span of 250 years, with $h = 2.5 \cdot 10^{-4}$ and the Solar system Barycenter as origin.

Figure 17 shows a spatial plot of all the planets in the Solar system, including Pluto, which is included for historical reasons. The system is plotted for a period of 250 years, which roughly corresponds to one revolution for Pluto. The origin is chosen as the centre-of-mass of the entire system, or the Solar system Barycenter. This has been done by using vector data from the JPL HORIZONS Web-Interface[3]. This means that in this system, the Sun is unfixed and allowed to move.

5.5 Perihelion precession of Mercury

A function `VelocityVerletRel()` has been made which includes the correction term to the gravitational force described in (10). After the entire Velocity Verlet algorithm has been for one century, the function finds the position of Mercury at Perihelion in the 100th year, and prints these coordinates. We've run the program with the Sun both fixed to the origin, and unfixed, with the centre-of-mass as origin.

Table 6: Values of θ, the angle of perihelion precession for Mercury after one century, Sun fixed to origin

	ϑ_P	Arc seconds
With relativistic correction term	$2.08 \cdot 10^{-3}$	429”
Without relativistic correction term	$2.88 \cdot 10^{-3}$	594”
Absolute sum	$1.05 \cdot 10^{-4}$	165”

Table 7: Values of θ, the angle of perihelion precession for Mercury after one century, centre-of-mass as origin

	ϑ_P	Arc seconds
With relativistic correction term	$4.52 \cdot 10^{-4}$	93.2”
Without relativistic correction term	$3.46 \cdot 10^{-4}$	71.2”
Absolute sum	$1.05 \cdot 10^{-4}$	22”

The simulation was done with $h = 5 \cdot 10^{-5}$ for the results in Tables 7 and 6.

6 Discussion

6.1 The Different Algorithms

When designing a program, or software for that matter, choosing your main, hard-coded algorithms wisely will dramatically improve the efficiency of said program and precision of the result you wish to produce. In our case, we have studied three different numerical schemes for solving a second order differential equation by recursion; the Euler - Forward and Euler - Cromer algorithms, as well as the Velocity Verlet algorithm. As we saw in Tables 2, 3 and 4, there are clear advantages to choosing the Velocity Verlet algorithm over the Euler methods. This is in part due to the global truncation error it commits, which decreases twice as fast as the error committed by the Euler methods. What concerns efficiency, we see that the tabulated CPU timings are insignificantly different, merely resulting from the difference in associated FLOPs being $5N$ and $9N$ respectively for the Euler methods and the Velocity Verlet method. In light of this, we here conclude that there is little reason taking this aspect into consideration when choosing between the three, especially when the slight halting is compensated by a much more precise algorithm. It should be made clear, however, that the data procured about the efficiency and precision of each algorithm has no statistical gravitas. The elapsed run times and maximum errors have undergone at most two trial runs each, leaving no room for possible deviations and more precise mean values. Looking at the anomalous increase in maximum numerical error for the Euler - Cromer algorithm in this respect, it is difficult to make conclusive arguments as to why the anomaly occurs. One may argue that the sudden increase in the maximum error is due to numerical round off as this is the case for many algorithms when employing very high temporal resolutions, yet one should expect that this would be the case for the Euler - Forward algorithm, too, if not for all three algorithms.

6.2 Conservation of Angular Momentum

As described in Section 4.6, the function `TEST_kepler()` tests - by Kepler's second law - for each run whether the angular momentum was conserved or not. Table 5 shows that up until 100 years, the step sizes needed are roughly the same. For 1000 years, a step size twice as small is needed. It is however unreasonable to expect this simulation to give correct values for more than a few centuries, so the step size needed for conservation of angular momentum can be taken to be in the order of 10^{-3}. The tolerance given to the `TEST_kepler()` function has been chosen through trial and error, but as we have found, any arbitrary value can be given, and conservation of angular momentum achieved with sufficiently small time steps.

6.3 Testing Forms of the Force

With an elliptical orbit, the Earth-Sun system quickly diverges from the real system for increasing values of β in the force term. For $\beta = 3$, the Earth falls into the sun before having done one revolution around the Sun. An inspection of the total energy of the system shows clearly that $\beta > 2$ is not representative of a real physical system. The energy is conserved for $\beta = 2$, which is the normal inverse square gravitational force. With a circular orbit, we have shown data for $\beta = 2.01$, and already for this small increase of the exponent, the total energy fluctuates. This is also reflected by Kepler's second law, as the angular momentum is not conserved for the systems as β is increased beyond 2.

6.4 Escape Velocity

In the Theory-section, an analytical value for the escape velocity of Earth was found to be $\sqrt{8}\pi$. If we from trial and error use the initial velocity where the total energy is zero according to the results obtained in Figure 12, we see that the numerical value is 8.89 which corresponds closely to the analytical value from the Theory-section: $\sqrt{8}\pi \sim 8.89$.

6.5 The Three-Body System

Figures 13 and 14 shows that the Velocity Verlet solver we have made is stable on the order of centuries for the three-body problem. After one millennium, it's clear that Earth's orbit is no longer stable, and after 12 millenniums, Jupiter's orbit itself is also unstable. Considering the respective temporal resolutions $h = 1.2 \cdot 10^{-5}, 1.2 \cdot 10^{-4}, 1.2 \cdot 10^{-3}$ and $1.2 \cdot 10^{-2}$, we see that the Velocity Verlet integrator produces well behaved solutions for high temporal resolutions, and for low temporal resolutions the solutions become

highly unstable - similar to the behaviour of the Earth in the Earth-Sun system. This leads us to appreciate our program as flexible with the ability of adding bodies systemically without risking the expense of greater numerical errors.

6.6 The N-Body System

Finally, as one may see in Figure 17, we have reached the final desired result - a fully functioning model for our Solar System. The plot includes the orbits of all planets in our Solar System, including Pluto, too. In the .cpp - file `multibodysystem.cpp` we have included NASA data for our moon, Luna, as well as three of the largest moons of Jupiter (Io, Europa and Ganymede) - we have decided not to include plots of these celestial bodies, however, as they orbit so close to Jupiter that it is difficult to view the orbit of Jupiter and its moons simultaneously. Some tedious zooming have however led us to conclude that these orbits too are physically feasible. As was previously mentioned, all simulations, including the simulations of all eight planets and Pluto, are run for systems in which the sun is fixed. The reason for this in the case of the N-body system, is that our group mistook the NASA data as being given in an inertial frame. A little too late we came to the realization that this was not the case, and that we indeed should have corrected our system's inertial reference frame with a center of mass such that the system has no net angular momentum. We still see our results as indicative of a program capable of producing results for large systems, being one of the major goals in light of object orienting our code. The code is written such that object such as moons and satellites can easily be added. We've opted to not do this, but this could be done when expanding the model further.

6.7 Perihelion Precession of Mercury

A goal of this project was to have our model produce results which are measured experimentally, more specifically; the Newtonian model for gravitation predicts a perihelion precession for Mercury which is is off by 43 arc seconds from experimental data. We wished to see whether we could add a relativistic term to the gravitational force term and make up for this discrepancy between the Newtonian model and the real Solar system. The data in Table 6 gives values of the precession which obviously is wrong, as the perihelion precession for both force-models are off by one order of magnitude. Table 7 gives data which are closer to what we wished to achieve, but is still inconclusive. We have been unable to uncover why our results do not reflect what we expect.

7 Conclusive Remarks

In summary, we have made some remarkable findings in this rather large study spanning across multiple topics. Although there are some aspects of the study which failed to produce solid results, it has overall revealed profound insight into numerical differential equation solvers, object orientation and our Solar System. Firstly, we have found that the Velocity Verlet algorithm proves superior to both the Euler - Forward and Euler - Cromer method. Its error displays a linear relationship with the temporal resolution which changes twice as fast as for the two Euler methods, and this comes at the expense of a run time which is slightly longer - yet insignificantly so. We made a good decision when choosing to include the Euler - Cromer method as a part of the study, as this also allowed us to compare two symplectic integrators. From what we have seen, the committed error in the two algorithms dominates over their ability to display symplecticity, and to conduct more precise studies of possible differences one should have a computer (or several) capable of performing simulations for much greater temporal resolutions than we have. Furthermore, we have at this point designed a program which, in light of object orientation, is highly functioning. It seamlessly creates large systems and produces results which behave similarly for different temporal resolutions. Moreover, it is constructed such that it is very easy to create systems of non-planetary objects and study these instead. Concerning the results procured from this study other than those relating to the algorithms and the object orientation of the code, there are next to none. We have obtained results regarding the perihelion precession of Mercury, but the findings are inconclusive.

8 References

References

[1] Einstein, Albert (1919) as cited in a letter to *The Times*, London. Retrieved from PDF file, 3-4.
http://germanhistorydocs.ghi-dc.org/pdf/eng/EDU_Einstein_ENGLISH.pdf

[2] Pollock, Chris. (2003) *Mercury's Perihelion*. PDF file, 7-10.
http://www.math.toronto.edu/~colliand/426_03/Papers03/C_Pollock.pdf

[3] NASA, JPL Solar System Dynamics. HORIZONS Web-Interface. 19.10.2020. 15:24
https://ssd.jpl.nasa.gov/horizons.cgi