Exploring an aggregate many-body problem using iterative numerical integration in Cartesian vector spaces.

Sakarias Frette
Anders Vestengen
William Hirst

October 27, 2020

Abstract

In the following report we have modeled the Solar system using the Velocity Verlet method of integration to solve the many-body problem. We used our model to examine many properties of gravity and other natural phenomena. We found that when modeling a binary system with the sun and the earth we were able to conserve energy up to at least $\frac{1}{150}$'th of the original energy(after 10 years) and $\frac{1}{100}$'th of the original angular momentum (after 1 year). We also found that the code was able to find a shift in the perihelion angle of mercury to be 45.55" when classical effects were subtracted, which is an accuracy of $\pm 2.5''$.

1 Introduction

In undergraduate physics we learn that most natural phenomena are extremely complex. To learn about these phenomena we reduce them down to simpler numerical models, which we can work out and solve for more definite answers. One of the most interesting examples of this is the planetary orbits of our own solar system. This is a very complicated problem, but its also a type of problem which can be successfully scaled down in complexity while remaining very relevant in teaching us about the actual solar system. It is also interesting in the sense that there are many observed values of our solar system that makes it ideal when creating tests for the model.

In this project we will be using the Velocity-Verlet algorithm for numerical integration, because this project is more reliant on the physics itself, rather than the numerical-precision or the speed of the computation. One example of this is that we need our solver to be symplectic, which we will get into later. This project also allowed us to explore several different subjects, like the movement of Mercury's perihelion angle with and without the effect of general relativity, the inverse-square relationship in gravitational force and the simulation of planetary orbits in our solar system over hundreds of years.

2 Theory

2.1 Velocity-Verlet

To simulate the positions of the planets we use the Velocity Verlet method. This method is energy conserving and has an error equal to $O(h^3)$. Unlike most known solvers, this method calculates first the position of the object, then the acceleration and then the velocity. To see how the Verlet method works for calculating the paths of planets we start with a more general differential equation, like Newton's second law in one dimension:

$$\frac{d^2x}{dt^2} = \frac{F(x,t)}{m} \tag{1}$$

with the following relations:

$$\frac{dx}{dt} = v(x,t)$$

$$\frac{dv}{dt} = F(x,t)/m = a(x,t)$$

Now, lets Taylor expand Newton's second law. We then get:

$$x(t+h) = x(t) + hx^{(1)}(t) + \frac{h^2}{2}x^{(2)}(t) + O(h^3) \tag{2}$$

If we discretize this equation with these relations, $x(t \pm h) = x_{i\pm1}$ and $x(t_i) = x_i$, we get

$$x_{i+1} = x_i + hx_i^{(1)} + \frac{h^2}{2}x_i^{(2)} + O(h^3) \tag{3}$$

Now, as with position, we also have the same relation for the velocity, given by:

$$v_{i+1} = v_i + hv_i^{(1)} + \frac{h^2}{2}v_i^{(2)} + O(h^3) \tag{4}$$

If we now derive the velocity equation, and only use term up to the second order, we get this equation:

$$v_{i+1}^{(1)} = v_i^{(1)} + hv_i^{(2)} + O(h^2) \tag{5}$$

We see here that the second derivative of the velocity can be approximated to the difference in the derivative of the velocity for this and the next timestep, divided by the timestep. We can then insert this relation into our equation and then get

$$v_{i+1} = v_i + hv_i^{(1)} + \frac{h}{2}(v_{i+1}^{(1)} - v_i^{(1)}) + O(h^2) \tag{6}$$

The first derivative of the velocity is just the acceleration, and so our final set of equations is then:

$$x_{i+1} = x_i + hv_i + \frac{h^2}{2}a_i \tag{7}$$

$$v_{i+1} = v_i + \frac{h}{2}\left(a_{i+1} + a_i\right) \tag{8}$$

2.2 Orbital mechanics

2.2.1 Units

When calculating distances and velocities at the planetary level, km and km/s are quite useless as units, when Mercury, the closest planet to the

sun, is at least 190 million km away from the sun. We use instead AU (Astronomical unit) for length, and AU/year for velocity. One AU is the average distance from the earth to the sun, or about 150 million kilometers. Mass is here measured in solar masses, where our sun has mass $M = 1_{\odot}$.

2.2.2 Newtonian mechanics

All bodies, such as bugs, humans, planets and galaxies all experience gravity. For most cases a gravitational force between two objects can be explained by Newtonian gravity, which is given as:

$$\vec{F_G} = G\frac{M_1 M_2}{R^2}\hat{R} \tag{9}$$

, where G is the gravitational constant which can be approximated to $G \approx 4\pi^2 AU^3/yr^2$, M_1 and M_2 is the mass of the two objects, R^2 is the square distance between them and $\hat{R}$ is the unit-vector which decides the direction of the force.

To calculate the path of a planet(p), we use (1), which means we need to find the force from all the surrounding planets(p_i). The force from N surrounding planets, F can be expressed as

$$\vec{F} = \sum_{i}^{N} G\frac{M M_i}{R_i^2}\hat{R}_i \tag{10}$$

Where M is the mass of the planet p, M_i is the mass of p_i, and R is the distance between p and p_i.

2.2.3 Conservation of Angular Momentum

Angular momentum is a conserved quality, given by the newtons first law for rotation, which says that the net torque is zero. This gives the following relation

$$\sum \tau = \frac{d\vec{L}}{dt} = 0$$

which gives us that

$$\vec{L} = constant \tag{11}$$

2.2.4 Einsteinian gravity

Newtonian mechanics is however an approximation, as shown with Mercury's orbit, which we will discuss later. When taylor expanding Einsteins general theory [1] of relativity we get a more accurate expression of the gravitational force:

$$\vec{F_G} = G\frac{M_1 M_2}{R^2}\left[1 + \frac{3l^2}{R^2 c^2}\right]\hat{R} \tag{12}$$

where l is the angular momentum per unit mass, $\hat{R}$ is the unit vector for the distance vector between the two objects and c is the speed of light.

2.2.5 Conservation of Energy

Total energy is a conserved quantity for a closed system[2]. In our case with a planetary system without outside influence from other stars, this means that the sum of potential and kinetic energy always is conserved.

$$V_{before} + K_{before} = V_{after} + K_{after} \tag{13}$$

However, due to numerical error, they may differ slightly in our model.

2.2.6 Perihelion and Aphelion

In orbital mechanics we denote the two most extreme points on a elliptic curve where the object is farthest and closest to its host, as aphelion and perihelion. This phenomenon is very noticeable with Mercury's orbit. Correctly predicting Mercury's moving perihelion and aphelion was one of the proofs for General Relativity. The perihelion angle is given by

$$\tan\theta_p = \frac{y_p}{x_p} \tag{14}$$

Where x_p and y_p are the Cartesian coordinates for the perihelion position. For Mercury it is known that when all classic effects are subtracted, the perihelion angle should move 43"(arc seconds) per century.

2.2.7 Keplers second law

Keplers third law is defined as the following;
A line joining a planet and the Sun sweeps out equal areas during equal intervals of time[3]

[1] More on Einsteins general theory in the two-body problem
[2] More on conservation of energy here
[3] Keplers laws of planetary motion

. Through geometry it can be shown that the area sweeped out by this line, ΔA (from t_i and t_{i+1} can be written as:

$$\Delta A = \frac{r^2}{2}\Delta\theta \tag{15}$$

, where r is the distance between the earth and the sun and θ is the angle between the planets position in t_i and t_{i+1}. This can be rewrtten as

$$\Delta A = \frac{L}{2m} \tag{16}$$

, where L is the angular momentum and m is the mass of the planet. In other words; if the area covered by the line between the earth and the sun is constant for the all time-intervals equal in size, the angular momentum is constant.

By assuming that the angle θ is very small, we can write $\Delta\theta$ as $\approx \sin\Delta\theta = \frac{vdt}{r}$. This gives us:

$$\Delta A = \frac{vr}{2}dt \tag{17}$$

3 Implementation

Our code can be found on this Github link in the "Project 3" directory.

3.1 The code/algorithm

The code partitioned into two classes. One stores the different planetary objects and holds a function to compute the distance. This class is called object. The second class is the solver, made to be as generalized as possible. It is also made to be versatile, with different inputs to turn on or off several components of the solution, or switch numerical solvers if we want to compare performance metrics for different methods (for example Forward Euler vs Verlet). The code then scales easily with different amounts of planets. This code is generalized to the point where it can model any manybody-system, as long as gravity is the only interacting force, which is the solver class.
The following algorithm shows our implementation of the Velocity-Verlet method:

```
Initialize matrix for acceleration;
Initialize variables and arrays for calculation;
Initialize file to write;
while t < t_max do
    for i=0; i<totalplanets; i++ do
        Update current to planet(i);
        Update forces on current planet (i);
        Update position for current planet (i);
        Update forces on current planet (i);
        Update velocity for current planet (i);
        Write position and energy to file;
    end
    Update time;
end
```

First, we define the objects we wish to use, in our case the planets. Once they are added we run the solver algorithm. Here we initialize to matrices for current and next acceleration, and define all needed variables, as well as set up a file where we will write our calculated values. Then for each time step, t_i we do the following;

- Update which planet you want to calculate the position of.
- Calculate the forces from all the other planets using equation (10) and using equation (1) to find the acceleration, a_i.
- Use a_i, velocity v_i and equation (7), to find x_{i+1}.
- Use new position, x_{i+1} to find acceleration a_{i+1}.
- Use v_i, a_i, a_{i+1} and equation (8) to find new velocity v_{i+1}.
- Repeat process over all planets.

3.2 Parameters

As discussed above, the solver class is written to be very general and very flexible. When calling the Verlet method in the solver class, there are many parameters we must define. Three of these parameters define the system we want to solve. The first parameter is called "beta". We can rewrite (10)

to be:

$$\vec{F}_i = \sum_i^N G \frac{M_j M_i}{R_i^\beta} \hat{R}_i \tag{18}$$

, where β is beta. This allows us to examine to which extent the gravitational force is an inverse-square force.

The second parameter is called "fixed". Fixed is an integer and is defined in such a way that if $fixed \neq 1$ then the sun is in origin and the planets do not effect the sun. If $fixed = 1$ the center of mass is in origin and the planets effect the sun. This means we calculate the position of the center of mass for each time-step and subtract it from all the planets. This allows us to examine the effect the suns movement has on the earth.

The third parameter is alpha. If $alpha \neq 1$, we only use Newtonian gravity (10). If $alpha = 1$ then we add an element from Einsteins general relativity, changing our expression for gravity, from (10) to (12). Finally this allows us to examine the limitations of Newtonian mechanics.

3.3 Perihelion angle

We added a function to find the perihelion angle for the innermost planet, Mercury. We do this by setting three variables, Monday, Tuesday and Wednesday, where they are all the distance between the planet and the sun but for time-step t_{-1}, t_i and t_{i+1}. The function will then check for the following; if the absolute relative distance from the planet to the star is less on Tuesday than it was on Monday, and less on Tuesday than it is on Wednesday, then we have found the perihelion position, and can calculate the angle using equation (14). We know this to be the perihelion position, because only when the planet is nearest to the sun, will this be true.

3.4 The Unit Tests

It is important to test the code, especially in the early stages of the project. We chose to run tests on a two body system (earth and the sun) with a common center of mass. The unit-test tests three aspects of the code; if the total energy of the planet is conserved, if the angular momentum of the planet is conserved and if the orbit itself is stable over time. Since the code models a physical system, we can determine the codes accuracy by testing

the codes ability to upholds physical laws.
We chose to test energy conservation by examining if the max-value of the total energy of the earth was relatively equal in the first and second half of the simulation time. By doing this we are testing if the total energy is periodic for each year, thereby making it conserved. The threshold for the values being equal was chosen to be under one 150'th of the maxvalue of the second half. In other words,we checked that the following was held:

$$\mid E_{tot,2} \mid - \mid E_{tot,1} \mid < \frac{\mid E_{tot,2} \mid}{150} \tag{19}$$

, where $E_{tot,1}$ is the max-value for the energy in the first half and $E_{tot,2}$ is the max-value in the second.

Secondly we test conservation of angular momentum. We do this by comparing the angular momentum in the first time-step to the angular momentum in the last time-step after one year. The test is passed if the difference between the two values is smaller then 1% of the angular momentum in the first time-step. The unit test is therefore passed if:

$$\mid l_0 \mid - \mid l_{N/ft} \mid < \frac{\mid l_0 \mid}{100} \tag{20}$$

, where l_0 is the angular momentum in the first time-step and $t_{N/ft}$ is the angular momentum in the last time-step after one year. Alternatively to this, we could see if Keplers second law holds, by using the expression we found in (2.2.7). We chose not to use this as a unit-test. We will discuss this further down.

Finally we check that the orbit of the earth is stable when we set the initial values of earth to be values we know result in a stable orbit. We do this by running the Verlet method and examine if the final distance from the sun is smaller den twice the distance from the sun for the first time-step. This results in the following requirement:

$$R^{end}_{earth,rel} < 2 * R^{start}_{earth,rel} \tag{21}$$

, where $R^{end}_{earth,rel}$ is the distance from the earth to the sun at the final distance and $R^{start}_{earth,rel}$ is the distance at the first time-step.

4 Results

After initially building our implementation we chose to start by comparing a few solvers. If we try to solve a binary system (the sun being in the

origin) with the same initial conditions, but with two different solvers, we find the following: From the two plots we see that the Verlet method calcu-

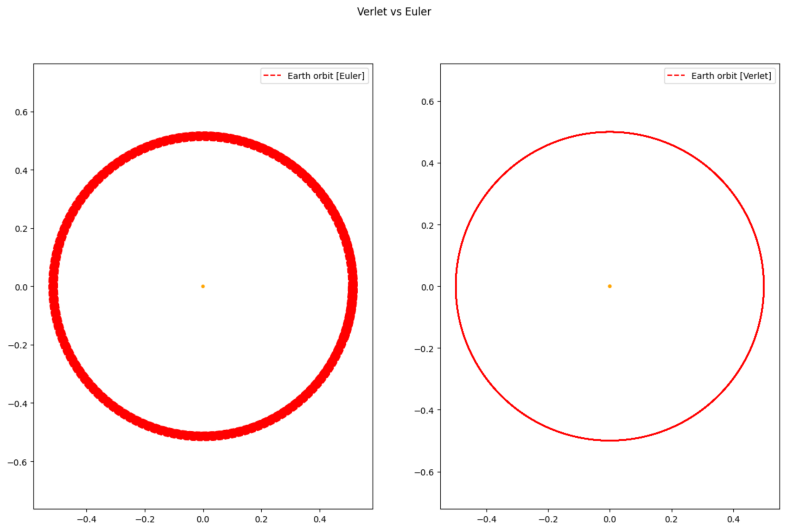


Figure 1: Figure showing the Earth-Sun system, with Euler(left) and Verlet(right) solvers both running for 150 years.

lates a much more stable orbit, and the Forward Euler method calculates a orbit which is falling towards the sun.

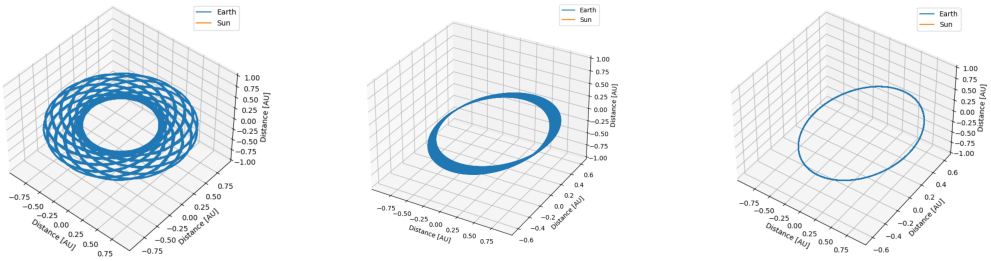
When using (17) for a system with the sun and the earth (sun being constantly in origin), we found the following result:

Timeintervall[Years]	[0.19,.2]·FinalTime	[0.69,0.70]·FinalTime
Area Sweeped [Au^2]	9.853	9.898

Table 1: A table of area swiped by a line between the earth and the sun. Initial distance for earth being 1Au, and velocity being 5 Au/years.

The table shows that the area swept is approximately equal. It is equal up to 2 digits.

Now that we have shown the Verlet methods superiority over Forward Euler, we look at the numerical stability of the Verlet solver. Like all integration methods, its accuracy is dependent of the size of its time-step. Choosing an initial velocity of $5AU/yr$ and staring distance of 1Au, we vary different time-steps(by changing integration points). From the plots



(a) Figure showing the Earth-Sun system, 1e3 integration points

(b) Figure showing the Earth-Sun system, 1e4 integration points

(c) Figure showing the Earth-Sun system, 1e5 integration points

Figure 2: Three plots for a binary system with the earth and the sun, running for 50 years.

we can see that the third plot is more stable. This means that for our current system we needed at least 1e5 integration points per 150 years.

Now that we have shown why we use the Verlet method, we extend it two a three body problem by including Jupiter. Giving Jupiter an initial velocity of 3 Au/year and a distance of 5.2 Au away from the sun, we get the following result:

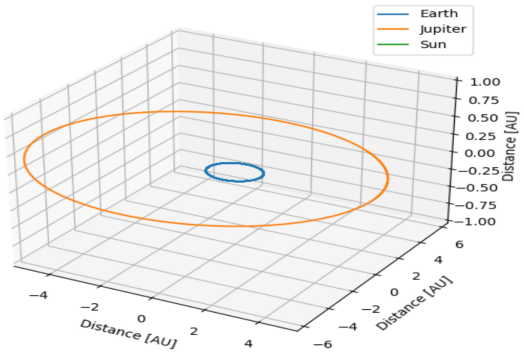
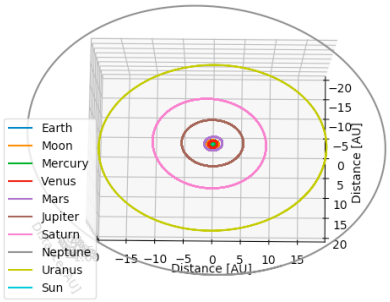


Figure 3: Figure showing the Earth-Jupiter-Sun system, solved with Verlet for 18 years.

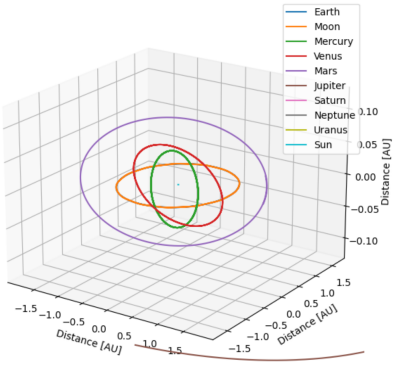
The sun is set in origin and is therefore not visible in plot.

Now that the code is tested, we want to simulate the entire solar-system.

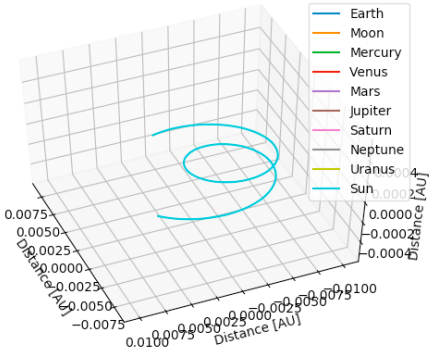
Therefore we add all the planets(and the moon) to our model using initial values from NASA and simulate. The first plot (a) shows all the planets



(a) Figure showing the full solar system with a run time of 180 years



(b) Figure showing inner planets of the solar system with a run time of 20 years



(c) Figure showing the sun with a run time of 20 years

Figure 4: Three plots show out model with all the planets and the center of mass in the origin. The plots our created by running for 180years (a) and 20 years (b,c)

for 180 years. We had to run this long so that all the planets made at least one orbit. The second and third plot (b,c) shows the planets after 20 years. This is because showing the inner planets after 180 years makes the plot hard to read.

Finally we want to check to see the difference in the perihelion angle of

Mercury after a 100 years with and without relativity (alpha = 1 and alpha=0). We find the following result with 7·1e7 integration points:

Timeintervall[Years]	Alpha =0	Alpha =1
New Perihelion angle [arc seconds"]	28.690	74.243

Table 2: The table shows two Perihelion angles after 100 years. Mercury started with initial distance 0.3075Au and initial velocity 12.44Au/years.

When subtracting the angle of alpha =0 from alpha=1 we find that it is approximately 45.55". This is quite similar to the observed value of 43". We choose not to add the plots of two different situations because the two plots are indistinguishable.

5 Discussion and analysis

5.1 The code

5.1.1 Object orientation

We made the code as generalized as possible, and a consequence of that is that it took a while to understand and plan for how to structure and design the code. We had done a similar code project in another course, so we used some of that code to help plan and then design this project. Based on our intuitive understanding of orbital mechanics and comparing this to our results, the code seems to work very well for this project.

Our biggest issue when building the code was finding a balance between a general code with many parameters and a "messy" solver-class or a specialized code with fewer parameters and a smaller solver-class. We choose to make the solver-class as generalized as possible. We choose this because we knew we wanted to examine many properties of our model and the easiest way to do this was making the code able to simulate as many scenarios as possible (two-body, three-body, Newtonian vs Relativity etc.). The only downside to the general nature of our code is the amount of parameters. This can make the code complicated to use, and sometimes even make the code a little slower. But, we found that for the amount of work we wanted to do(with the code) and with a thorough README-file this was the best way to do this.

5.1.2 Solvers

One of the problems with using numerical tools over such a long time is the fact that some solvers like Forward Euler or Runge-Kutta4[4] are non-symplectic, which means the system loses some energy after every numerical integration. Forward-Euler method and our preferred Velocity-Verlet solver was tested in this project, and from the plots in figure 6 we observe that the orbit computed with the Velocity Verlet method is relatively stable, while the orbit calculated from the Euler-forward implementation keeps losing energy. We can see the system loosing energy by the planet falling towards the sun.

There are however costs for the choice of the symplectic solver. A solver such as Forward Euler has a significant speed advantage to make up for the lack of accuracy, due to the low number of calculations needed. On the other side, we have solvers like Runge-Kutta2 and Runge-Kutta4, who, all though are slow due to the amount of operations needed to calculate, have great accuracy.

Some numerical solvers might exhibit unstable solutions, due to stiffness[5] in the equations. This means that the Velocity Verlet method might make poor calculations for varying timesteps, in for example collisions compared to orbit calculations. In figure (2) we showed the relationship between the time-step and the accuracy. We see that even for a time-step equal to 1e4 per 50 years the solution is not very accurate.

5.1.3 β testing

From the implementation section we discussed how our solver allows us to solve a gravitational system in a very general sense. This means we can run the system for different values. In all the plots in our result-section $\beta = 2$, but in reality β is not exactly 2(although very close). Therefore it could be interesting to see what would happen if we changed the β value from 2 to 3. Thereby checking whether there is a case for using an inverse-cubed or an inverse-proportional force. For β = 3: It seems like if gravity worked like this, we would have to be much closer to the sun. From a distance like the earth it seems to fall of so rapidly we never achieve a stable orbit.

[4]More on Runge Kutta methods can be read here.

[5]More on numerical solutions to differential equations here.

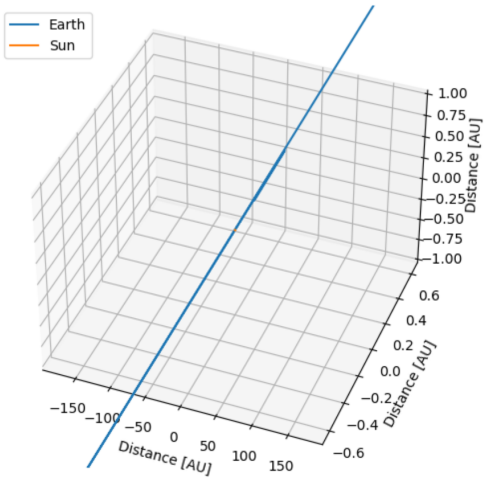


Figure 5: Figure showing the Earth-Sun system with an inverse-cubed ($\beta = 3$) force. Integration time of 50 years. Integration points N = 1e5

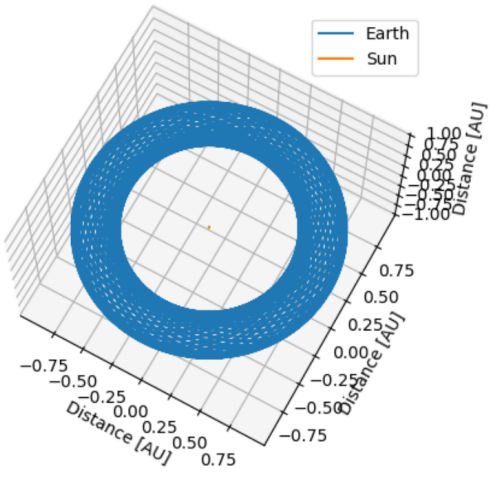


Figure 6: Figure showing the Earth-Sun system with an inverse-proportional ($\beta = 1$)force. for N = 1e7 integration points and 50 years of integration time.

In this case we see the orbit is much less stable, and the orbit itself seems to shift around in a much bigger pattern. To clarify we tested this for a higher number of integration points (N=1e7) than we previously es-

tablished as the minimum (N=1e5) required for a stable calculation, this is because we found this plot to be similar to Figure 2a, which is an unstable orbit due to a lack of integration points.

Furthermore, we use our implementation of Kepler's law to verify the validity of the different β values. We found that for $\beta = 3$, we have large errors, by about an order of magnitude. However, for $\beta = 1$ and $\beta = 2$ the differences seems to be within tolerance. We've already discussed the shortcomings of our Kepler implementation, and we believe there should be a difference between the two, we just can't see it with this implementation.

5.1.4 Perihelion Precision

As mentioned in section 2.2.4, Newtonian gravitation is an approximation to Einstein's field equations. This means that for orbits such as Mercury's, it will not predict the precise orbit over time. We tried to look at the perihelion precision of Mercury's orbit with the next term in the Taylor expansion, i.e equation 12, to see if that would help. We found that the angle we found was 2 arc seconds of the observed angle. This is a good result, and is a major confirmation of the legitimacy of our code as well as a confirmation of Einsteins theory of general relativity.

We suspect that the deviation from the observed value is due to our algorithm of finding the perihelion angle. The movement of the perihelion angle, is measured in arc seconds where one arc second is approximately 0.000277^o. To measure such a small movement we have to increase the amount of integration points(thereby decreasing the time-step). We found that when increasing the amount of integration points above a certain point (ca. 10^7), our calculated movement of perihelion angle deviated from the observed value. This is due to the fact that when the time-step is decreased, the change from one time-step to another does as well. When the change between time-step decreases so does the effect of truncation error. Therefore we are certain that when the time-step is to small, there will be several point along the orbit where $x_{i-1} < x_i < x_{i+1}$. Therefore our algorithm finds the wrong angle.

Alternatively we should have used a different algorithm to find the angle. We should have made the code find the point nearest to the sun in the last year of the simulation instead of trying to find the angle for each orbit. This would not experience the same error as described above and would

allow us to increase the amount of integration points which would give us a more accurate angle.

5.2 Unit tests

When testing our system for energy-conservation our threshold for equality in the max-value of the first and second half was chosen to be $\frac{E_{tot,2}}{150}$. Firstly we chose the threshold to be a fraction of the second value, since we do not really have an intuitive estimate for what the difference will be. Secondly this threshold is relatively large. This could be contributed to the fact that energy-conservation holds true for the hole system, not just the earth. We decided that since the earths effect on the sun will we minuscule, we did not include the sun. But, this is why the threshold was chosen to be so large.

When checking if the angular momentum was conserved, (as discussed in implementation section) we calculated the two values at different times and compared. Alternatively we could have used Keplers second law (see subsection on Keplers second law). But, to make it easier to calculate the area swiped by our "line", we found the expression (17).
From table (1) we found that when doing so, the area is relatively equal, but only by two digits. The reason being that we assumed that the angle θ(see section for Keplers second law) was very small. This will be true for planets moving slowly, but for the earth moving (relatively) fast, it is not very accurate. But, although it is not very accurate, (17) can be used to test our code. Even for elliptic orbit, the area will be relatively equal. This is because when the planet is close, the planets speed will compensate and cover an equally large area as when the planet is far away and and is moving slowly.

The unit tests chosen were solely based on physical properties of planetary systems. In order to ensure a good model, we chose our three test on the fact that they are easy to check and should hold for conservative forces, such as gravity. However there were other potential tests we could have run instead. One such test is to check that the velocity of a planet in orbit never exceeds or matched the escape velocity for the given system. This is however a another version of the third test, so there was no need to use this one. Another test would be to check that the relative distance between the planet and the sun was larger than the radius of the sun. This way, if the planet had an unstable orbit due to poor choice of initial conditions,

and ended up inside the sun, the code would kill the simulation.

6 Conclusion

We found that the Velocity Verlet method works well to solve the many body problem of our solar system. By combining object orientation and this numerical solver, we managed to simulate all the planets in our solar system, including the moon. Since we simulated a real physical system, there were many ways of testing the code, for example by checking conservation of energy and angular momentum. When using this test to create a unit-test, we discovered that the values were conserved to threshold of at least 1% of the values we compared.

We were also able to calculate the effect of Einsteins relativity on the movement of the perihelion angle of Mercury. We found the effect on the angle to be 45", which is an accuracy of $\pm 2.5''$. We attributed this deviation to truncation error. We also found that when trying to use Kepler's second law to test conservation of angular momentum, this was also affected by numerical complications, even though we found that the law was upheld up to 2 digits.

In conclusion we have found that our model up holds many physical laws and can be used to show many natural phenomena, when using the velocity verlet method with a sufficient number of integration points.