BUILDING A MODEL FOR THE SOLAR SYSTEM USING ORDINARY DIFFERENTIAL EQUATIONS

Linus Ekstrm, Joakim Flatby
https://github.com/Linueks/fys3150 and
https://github.com/jflatby/fys3150

ABSTRACT

This article studies two methods for integrating the acceleration of gravitationally attracting celestial bodies: the forward Euler and Verlet velocity method. We used an object oriented approach, with three python classes, for our numerical simulations. We used a solver class containing the integration methods, a planet class to initialize the initial conditions for the celestial bodies, and a solar system class to set up and run the simulation. Further, we compared the efficiency of the forward Euler and velocity Verlet method as well as their numerical stability in terms of energy conservation. A couple of different scenarios were studied: the Earth-Sun system, the Earth-Sun-Jupiter system, an n-body simulation of the full solar system, and finally we added the relativistic correction for the precession of Mercury's orbit. Analysis of the integration methods showed that the forward Euler method solves for the motion faster than the velocity Verlet for a given time step. However, the forward Euler method requires a smaller time step to reach the same precision as the velocity Verlet. Thus, we conclude that the velocity Verlet method is the way to go when running n-body simulation.

Subject headings:

1. INTRODUCTION

The motion of the planets in our solar system is governed by gravity. Gravity can be modeled using *Newton's law of universal gravitation*. Through Newton's law we can extrapolate a differential equation which we can then discretize to simulate the motion of the planets. There are several different ways of numerically solving differential equations and throughout this article we will take a look at two different methods, the *forward Euler method* and the *velocity Verlet method*. Through other known laws of physics like *conservation of energy* we will then test whether our simulations produce the results we expect. For the first part of this project we find it sufficient to keep the Sun fixed in the center of the solar system, but in the end we will also take a look at including the full N-body problem. When implementing movements for all the bodies a couple more ideas need to be introduced like calculating the center of mass for the system. Lastly we will use our simulated environment to look at the perihelion precession of Mercury.

2. METHOD

2.1. *The Earth-Sun system*

2.1.1. *Newton's law of gravitation*

Before we can write our code, we need to figure out how we are going to simulate the motion of the planets. To do this, we are going to use Newton's law of gravitation

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

where F_G is the gravitational force acting between the two objects with mass m_1 and m_2, and r is the distance between their centers of mass. G is the gravitational constant.

To begin with we just want to simulate a simple Earth-Sun system. We assume that the Sun has a much larger mass than the Earth, so we neglect the motion of the sun and keep it fixed in the center. We also assume the orbit of the Earth to be co-planar and call it the xy-plane. From Newton's second law we know that $a = \frac{F}{m}$, so inserting the Sun and Earths masses, we get

$$\frac{d^2\vec{r}}{dt^2} = \frac{GM_{\text{Sun}}}{r^2}\hat{r} \tag{2}$$

Lastly we assume near circular orbit for the Earth, so we can use the relation

$$F_G = \frac{M_{\text{Earth}}v^2}{r} = \frac{GM_{\text{Sun}}M_{\text{Earth}}}{r^2} \tag{3}$$

From this we can see that

$$v^2 r = GM_{\text{Sun}} = 4\pi^2\Big[\frac{AU^3}{yr^2}\Big] \tag{4}$$

so all we need to solve for this approximation is

$$\frac{d^2\vec{r}}{dt^2} = \frac{4\pi^2}{r^2}\hat{r} \tag{5}$$

2.1.2. *Euler and velocity Verlet*

To simulate this Earth-Sun system we are going to discretize the differential equation (5) in Python, using two diffrent methods. First we create a python script which loops over a set amount of time steps decided by $\frac{\text{Total time}}{\Delta t}$ where Δt is the step size. We define a method which finds the acceleration based on Eq. (5), making sure to keep everything in AU and years.

The *forward Euler method* is a simple first order numerical integration method

$$\begin{aligned}\vec{r}(t+\Delta t) &= \vec{r}(t) + \vec{v}(t)\Delta t \\ \vec{v}(t+\Delta t) &= \vec{v}(t) + \vec{a}(t)\Delta t\end{aligned} \tag{6}$$

All we need to implement this into python is to define arrays to store the positions and velocities for each time step, and calculate the acceleration every time based on the position $\vec{r}$ at time t.

The *velocity Verlet method* is another numerical method which should have a much better numerical stability, and is commonly used in molecular dynamics trajectory-simulations and computer graphics.

$$\begin{aligned}\vec{r}(t+\Delta t) &= \vec{r}(t) + \vec{v}(t)\Delta t + \frac{1}{2}\vec{a}(t)\Delta t^2 \\ \vec{v}(t+\Delta t) &= \vec{v}(t) + \frac{1}{2}\big(\vec{a}(t) + \vec{a}(t+\Delta t)\big)\Delta t\end{aligned} \tag{7}$$

As we can see the velocity Verlet method requires us to keep an array containing the accelerations as well, since we now need access to the acceleration for the last and current time step.

2.1.3. *Writing an object-oriented code for the Earth-Sun system*

To object orient our code we chose to create a Solver class, containing the velocity Verlet method and the forward Euler method, as well as the time step info. Next we have a Planet class which contains mass, positions, etc. and finally we have a SolarSystem class which has an instance of the solver class, and an array of Planet-instances.

2.1.4. *Test of the algorithms*

Now that we have a class which can simulate a planet orbiting a fixed star, we need some initial values to get us started. We measure distance in AU so Earth's initial position can be $(1, 0)$. We know from equation (4) that $v^2 r = 4\pi^2$ and from this we get

$$v = \sqrt{\frac{4\pi^2}{r}} = 2\pi[\text{AU/yr}] \tag{8}$$

since $r = 1$AU. So Earth's initial velocity is set to $(0, 2\pi)$ to get a counterclockwise orbit.

To further test whether our simulation behaves as we expect we can check if the potential and kinetic energies are conserved, as well as the angular momentum. Since we are in an isolated system, and no external forces other than gravity are applied to the planet, we expect the total mechanical energy and the angular momentum to be conserved. If the orbit is nearly circular, we expect the potential and kinetic energies to stay relatively constant.

As mentioned in section 2.1.2 we expect the velocity Verlet integration method(7) to be the best choice. We can test this by comparing the results for some Δt's, as well as timing how long they each take to execute. Our Euler algorithm only has 4 FLOPS, and only requires the acceleration calculated from the current position. The velocity Verlet algorithm on the other hand, has about 10 FLOPS, and needs to store the accelerations in an array because it needs both the preceding and current accelerations to calculate velocity. Therefore the Euler method is probably faster when using the same Δt, however, the Verlet method should be able to provide a better approximation at higher Δt, which could mean even faster run time.

2.1.5. *Escape velocity*

Using our simulation we can now play around with the initial velocity of the Earth until we get an escape trajectory out of the gravitational influence of the Sun. To find the exact answer to this we use conservation of energy

$$U_i + K_i = U_f + K_f \tag{9}$$

where U is potential energy and K is kinetic energy. This gives

$$\frac{1}{2}mv_e^2 + \frac{-GMm}{r} = 0 + 0 \tag{10}$$

$$v_e = \sqrt{\frac{2GM}{r}} = \sqrt{2} \cdot 2\pi \tag{11}$$

We also see what happens when we change the gravitational force so

$$F_G = \frac{GM_\odot M_{\text{Earth}}}{r^\beta} \tag{12}$$

where $\beta \in [2, 3]$.

2.2. ***The three-body problem***

Now that we have achieved simulating an orbit using the gravitational pull between the sun and the Earth, we want to extend our program to be able to apply the gravitational force from all other planets in the system in each time step. To do this, we keep an array in our SolarSystem class called "planets", which contains instances of the planet class. The get_acceleration function in the Planet class now takes an array of planets as an argument, and calculates the gravitational pull from all planets except for itself.

If we now include Jupiter in the simulation, we should see an effect on Earth's orbit due to the gravitational pull between Earth and Jupiter. However the changes are too small to visually see on a plot on the scale of Jupiter's orbit. Therefore we will test this effect to the extreme by increasing Jupiter's mass by 10 and 1000 times and running the simulation again.

2.3. ***N-Body problem***

2.3.1. *Mercury's Precession*

It is known that Mercury's orbit is not properly accounted for in Newtonian gravity. Therefore, an additional relativistic correction is required to obtain a more realistic model of the Solar system. For Mercury, the gravitational force is better approximated by

$$F_G = \frac{GM_{\text{Sun}}M_{\text{Mercury}}}{r^2}\left(1 + \frac{3l^2}{r^2c^2}\right) \tag{13}$$

where l is the angular momentum of Mercury around the sun and c is the speed of light in a perfect vacuum.

3. RESULTS

3.1. ***The Earth-Sun system***

3.1.4. *Test of the algorithms*

Using 2π as initial velocity gave us a circular orbit, shown in Fig. 1.

As we can see in Fig. 2, kinetic energy, potential energy and angular momentum are conserved in a circular orbit. In Fig. 3 we can see that angular momentum is conserved in an elliptic orbit as well, and that the total mechanical energy is conserved by the changes in potential energy canceling out the changes in kinetic energy.

Our results from timing the integration methods confirms what we expected, the forward Euler algorithm is faster than the velocity Verlet algorithm for a set amount of time steps. However, to achieve a good enough approximation so that our planet completes one circular orbit in one year, the Euler method needs a Δt as small as $1 \cdot 10^{-5}$, while the Verlet method is accurate enough all the way up to $\Delta t = 1 \cdot 10^{-3}$. This dramatic difference in

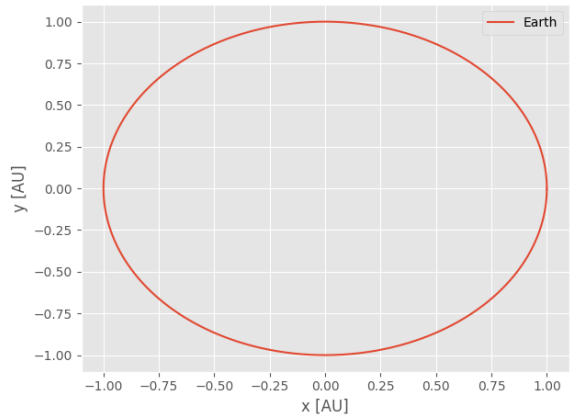

FIG. 1.— First simulated circular orbit

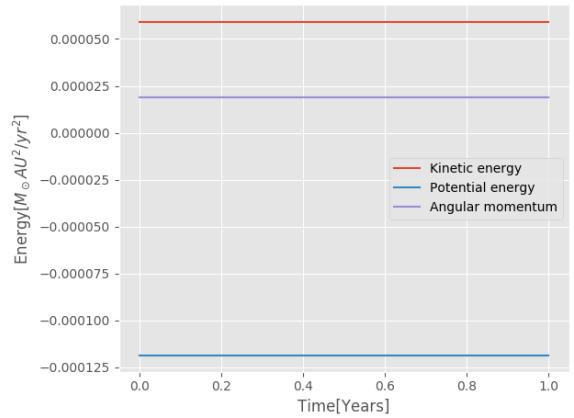

FIG. 2.— Kinetic and potential energy and angular momentum of Earth plotted over one year

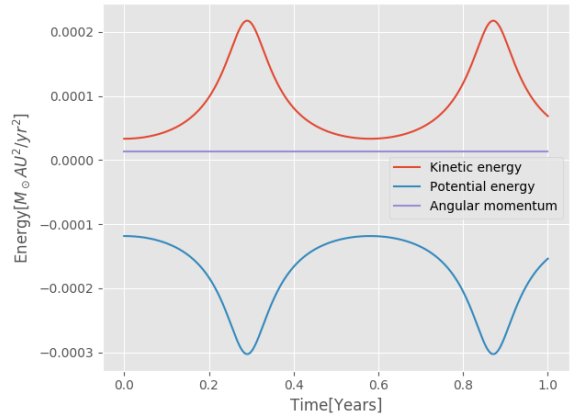

FIG. 3.— Kinetic and potential energy and angular momentum plotted over one year with an elliptic orbit

Δt has much more impact on how long it takes to run the algorithm than the difference in the amount of FLOPS, so in the end the velocity Verlet method is definitely the way to go. Results are shown in Table 2

3.1.5. *Escape velocity*

After playing around with our simulation, plotting 200 years to be certain, with $\Delta t = 1 \cdot 10^{-3}$ and the Verlet method, we found that escape out of the solar system was achieved at about $8.8 AU/yr$. Calculating the exact number using Equation (11) gives $v_e = 8.886$. This is a great indication that our simulation works just like we want it to, even with a pretty high Δt to make it go quick when plotting 200 years, the movement of the planet was simulated closely enough so that we were able to find the escape velocity just by looking at the graph.

Δt	Forward Euler	Velocity Verlet
$1{\cdot}10^{-3}$	0.018s	0.023s
$1{\cdot}10^{-5}$	1.97s	2.42s

TABLE 1
INTEGRATION TIMES FOR THE FORWARD EULER AND VELOCITY VERLET METHODS FOR TWO DIFFERENT VALUES OF Δt

When we start changing the power of r in the gravitational equation (12) from 2 to 3 the orbit becomes more and more unstable. Plotting 20 years using $\beta = 2$ gives almost no visual difference from plotting 1 year since the lines are basically on top of each other on that scale. However when we use $\beta = 3$ the entire orbit looks smudged out because the planet no longer travels in a perfect circle.

3.2. ***The three-body problem***

As we can see in Fig 4, the difference in Earth's orbit is almost not noticeable at this scale over 10 years. However when compared to Fig. 5 we can see that with a 10 times as massive Jupiter, our orbit is shifted enough for a noticeable difference in the plot. In Fig. 6 we see that with Jupiter 1000 times as massive, Earth's orbit is pulled all over the place, getting a different ellipse around the Sun each orbit. When simulating this scenario the Earth would get shot out of the solar system through a gravity slingshot close to the sun if the Δt was too high. But with sufficiently low Δt we ended up with the result in Fig. 6.

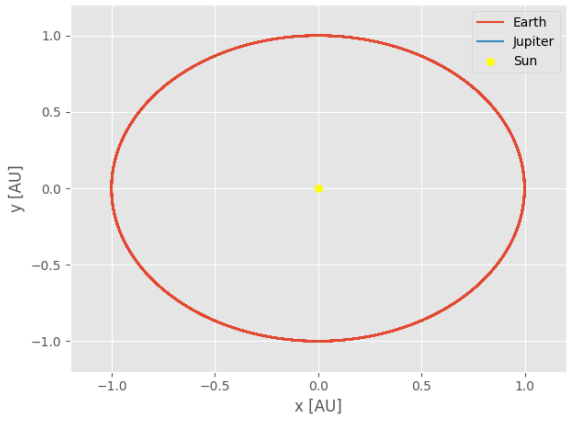

FIG. 4.— Earth's orbit with included gravitational pull from Jupiter. There is a difference, but too small to notice at this scale.

3.3. ***N-body Simulation***

3.3.1. *Mercury's precession*

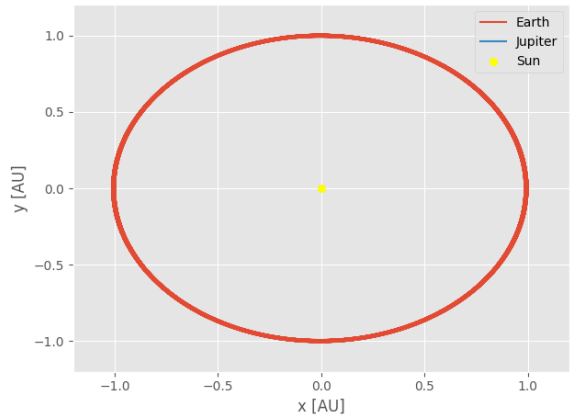

FIG. 5.— Earths orbit with gravitational pull from Jupiter with 10x mass

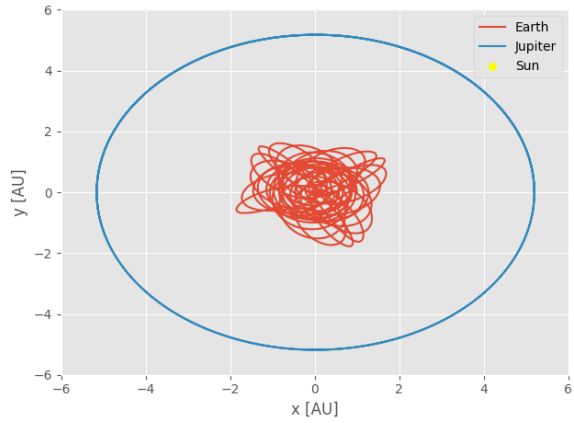

FIG. 6.— Earths orbit with gravitational pull from Jupiter with 1000x mass

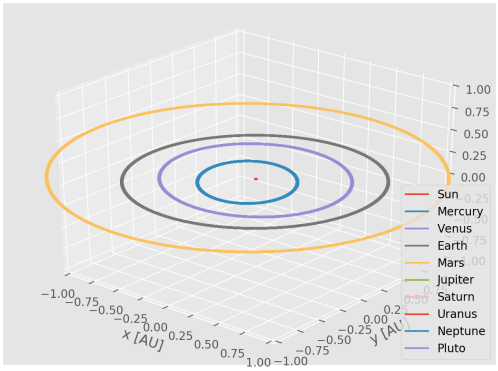

FIG. 7.— 3D plot of the N-body simulation with 1 AU on each axis

When simulating Mercury's position using the relativistic expression there was little visual difference in the orbit, which makes sense since the perihelion is only supposed to be moving at 44 arcseconds per century. By tracking the perihelion of the orbit, and comparing the position after 100 years, we see how many arcseconds it moved. When using $\Delta t = 10^{-4}$ we got around 1500 arcseconds, and with $\Delta t = 10^{-5}$ we got 121 arcseconds, which leads us to the conclusion that it probably converges towards 43 as we increase the accuracy of the simulation. However, the simulation would take very long, so we were not able to test this.

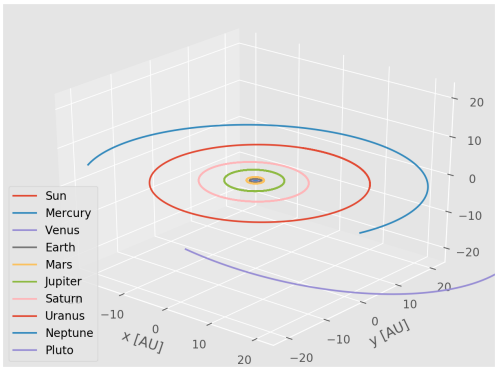

FIG. 8.— 3D plot of the N-body simulation viewed at an angle

Δt	perihelion movement after 100 years
$1{\cdot}10^{-4}$	1542.8"
$1{\cdot}10^{-5}$	121.37"

TABLE 2

4. CONCLUSIONS

From running the simulations of the Earth-Sun system, the Earth-Sun-Jupiter system and the full Solar system we have concluded that for exploratory calculations that the velocity Verlet method is quite a bit more effective than the forward Euler. Our main reason for thinking this is the accuracy gained for the close to zero implementational difficulties. In addition, the velocity Verlet method conserves energy so it is a lot more stable over longer simulations as well. However, to be accurate enough to accurately simulate Mercury's perihelion precession over 100 years, we would need to run the simulation for a very long time.

REFERENCES

[1]Wikipedia *Verlet integration* Last update Sep. 2018 https://en.wikipedia.org/wiki/Verlet_integration
[2]Wikipedia *Newton's law of universal gravitation* Last update Oct. 2018 https://en.wikipedia.org/wiki/Newton%27s_law_of_universal_gravitation