The Solar System

Brandt, Samuel   Davidov, Aleksandar   Hemaz, Said
github.com/aleksda

October 2019

1 Abstract

In the third project we build a an easy and reusable object oriented template to simulate the solar system. For the calculation of the planet orbits we use the Forward-Euler and the Velocity-Verlet integration methods. In spite of the the velocity Verlet method using a few more FLOPS than the Forward-Euler algorithm, it proves to be a more efficient algorithm in terms of precision. Using the reusable nature of object oriented programming we start of with two celestial bodies then we gradually expand to the whole solar system.

Contents

2 Introduction

The purpose of this project is to demonstrate the advantages of object oriented programming , and compare the efficiency of certain algorithm simulating an object oriented framework for the solar system. Newton's theorized his gravitational law to explain how bodies moves in general. He also developed a set of differential equations ,able to explain how the celestial bodies around the solar system orbit. We will solve the differential equations using algorithms that has the property of conserving the total energy and angular momentum in a precise manner. The Forward-Euler and velocity Verlet algorithms will then be compared for their conservation properties, and also their ability to output the position. furthermore, the amount of FLOPS as well as computational time will also be compared. In our framework we will be solving the coupled differential equation in three dimensions with N amount of planets starting from two and gradually increasing N to include the whole solar system. Additionally we will also examine Einstein's general relativity test on the perihelion precision of Mercury.

3 Theory

3.1 The differential equations

The planets orbit around the sun can be described in terms of Newtonian gravitation , which states

$$\vec{F}_{ij} = G\frac{M_i M_j}{r^3}\vec{r}, \tag{1}$$

where G is the gravitational constant, M_i and M_j are the masses of the given objects and r is the distance between them.

Newton's second law yields the relation between the acceleration of an object and the force acting on it.

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

Furthermore the relations between acceleration, velocity and position is given by the following differential equations

$$\begin{aligned} \frac{d\vec{v}}{dt} &= \vec{a}, \\ \frac{d\vec{r}}{dt} &= \vec{v}. \end{aligned} \tag{3}$$

With this information we can process it further to solve a set of coupled differential equations to find the position with respect to the time from the initial position and the calculated force.

3.2 Escape velocity

The escape velocity of the Earth can be found by setting the mechanical energy of the Earth to be greater than zero, to get it out of a bound state. When considering the following system

$$E_{Earth} = K + P = \frac{1}{2}M_{Earth}v^2 - G\frac{M_\odot M_{Earth}}{r^2}. \tag{4}$$

If we make this is to be grater than zero and insert values in units of AU, years and Sun masses, $G = 4\pi^2$, we have

$$\frac{1}{2}M_{Earth}v^2 - G\frac{M_\odot M_{Earth}}{r^2} > 0 \tag{5}$$

$$v > \sqrt{2G\frac{M_\odot}{r^2}} \tag{6}$$

$$v > 2\pi\sqrt{2}AU/year. \tag{7}$$

In order for earth to escape the sun's gravitational pull, it needs an initial velocity at least $2\pi\sqrt{2}$AU/year, if it start 1 AU away.

3.3 The Lagrangian

By modifying newtons gravitational law 1 to fit the sun as the centre of mass with the earth in its system , we get:

$$\vec{F} = G\frac{M_\odot M_{Earth}}{r^\beta}\vec{e}_r, \tag{8}$$

$\vec{r}$ will be the position vector of the Earth, and can also be expressed as

$$\vec{r} = r(\cos\theta, \sin\theta). \tag{9}$$

To find the potential we integrate and get the following potential :

$$V = \frac{1}{\beta - 1}\frac{GM_\odot M_{Earth}}{r^{\beta-1}} = \frac{1}{\alpha}\frac{GM_\odot M_{Earth}}{r^\alpha}, \tag{10}$$

where $\alpha = \beta - 1$.

the Lagrangian is the kinetic minus the potential energy of the system:

$$\mathscr{L} = \frac{1}{2}M_{Earth}\left(\dot{r}^2 + r^2\dot{\theta}^2\right) - \left(-\frac{1}{\alpha}\frac{GM_\odot M_{Earth}}{r^\alpha}\right). \tag{11}$$

Through some derivations related to the angular momentum we can reach the one-dimensional version of the Lagrangian:

$$\mathscr{L} = \frac{1}{2}M_{Earth}\dot{r}^2 - V_{\text{eff}}(r), \tag{12}$$

The effective potential is defined as:

$$V_{\text{eff}}(r) = \frac{1}{2}\frac{l^2}{M_{Earth}}\frac{1}{r^2} - \frac{1}{\beta - 1}\frac{GM_{\odot}M_{Earth}}{r^{\beta-1}}. \tag{13}$$

For the effective potential to have any stable equilibrium, we must have

$$\beta < 3. \tag{14}$$

The stability of the orbit of the Earth will vary for different values of β, and for the orbit to be stable $\beta < 3$.

3.4 Newton's gravitational law in the light of Einstein's general relativity

A modification of Albert Einstein's general theory of relativity (GR) to first order in $1/c^2$, will give us the following relativistic correction on Newton's gravitational law:

$$\vec{F}_{ij} = G\frac{M_i M_j}{r_{ij}^2}\left[1 + \frac{l_{ij}^2}{r_{ij}^2 c^2}\right]. \tag{15}$$

This gives a contribution that changes the position of the perihelion point of orbit over time. The effects are more observable for fast moving planets. The contribution to the perihelion precision is given by [3]

$$\epsilon = 24\pi^3 \frac{a^2}{T^2 c^2 (1 - e^2)}, \tag{16}$$

where e is the orbital eccentricity, a the semi-major axis, and T the orbital period. The fastest moving planet is Mercury. The perihelion precision of Mercury is $42.97''$ per century.

4 Methods, algorithms and implementation

4.1 Implementation

This source code contains two classes for simulating a solar system. The first is called Planet and it contains information on the planet of choice. The second is called System and is used for solving and plotting the equations needed for the system.

The System class contains two functions: solve and plot. The solve function has four inner functions:a_g, forward_euler, euler_cromer and velocity_verlet. The a_g calculates the acceleration and is used by the forward_euler, euler_cromer and velocity_verlet functions to calculate the next position . A more detailed description of the algorithmic nature and implementation of the Forward-Euler method (FE) and the Velocity-Verlet method (VV) below.

4.2 Forward-Euler

The Forward-Euler method is derived from the first two terms of the Taylor expansion and, can be expressed in the following general algorithm as:

```
compute $h$
initialize $\vec{x}_0$ and $\vec{v}_0$
**for** n = 0, 1, 2, ..., N-1 **do**
    compute $\vec{a}_n$ from forces
    $\vec{x}_{n+1} = \vec{x}_n + h\vec{v}_n$
    $\vec{v}_{n+1} = \vec{v}_n + h\vec{a}_n$
**end for**
```

Here, the a_n is the acceleration, v_n is the velocity and x_n is the position, after a certain number of steps n, while dt is the time step ,and t_n is the time. The algorithm is not very costly in terms of time-complexity, we can see from the algorithm that it has $4n$ FLOPS. In Python the code of the function forward_euler is implemented like this:

```
def forward_euler(steps, N, m, x, v, dt, G, origin_mass):
            total = (steps-1)*N
            count = 0
            perc = 0
            perc_new = 0

            for i in range(steps-1):
                for j in range(N):
                    r = x[i] - x[i,j]
                    r_norm = np.sqrt(np.sum(r**2, axis = 1))
                    v[i+1,j] = v[i,j] + a_g(m, r, r_norm, G)*dt
                    if origin_mass != 0:
                        x_norm = np.sqrt(np.sum(x[i,j]**2, axis =
    0))
                        v[i+1,j] = v[i+1,j] - G*origin_mass*x[i,j]*
    dt/x_norm**3
                    x[i+1,j] = x[i,j] + v[i,j]*dt

                    count += 1
                    new_perc = int(100*count/total)
                    if new_perc > perc:
                        perc = new_perc
                        print(perc)
            return x,v
```

4.3 Velocity Verlet

The algorithm for VV is derived from the first three terms of the Taylor expansion ,and looks like this, [1]

```
compute $h$
initialize $\vec{x}_0$ and $\vec{v}_0$
```

```
for n = 0, 1, 2, ..., N-1 do
    compute $\vec{a}_n$ from forces
    $\vec{x}_{n+1} = \vec{x}_n + h\vec{v}_n + \frac{h^2}{2}\vec{a}_n$
    compute $\vec{a}_{n+1}$
    $\vec{v}_{n+1} = \vec{v}_n + \frac{h}{2}(\vec{a}_{n+1} + \vec{a}_n)$
end for
```

For the implementation of this algorithm is to calculate $\vec{a}_{n+1}$ between the calculation of $\vec{x}_{n+1}$ and the calculation of $\vec{v}_{n+1}$, since we only need $\vec{x}_{n+1}$ for the calculation of $\vec{a}_{n+1}$, while we need $\vec{a}_{n+1}$ for $\vec{v}_{n+1}$.

This algorithm yields more precise results and have better conservation properties compared to FE. High accuracy comes naturally with the cost of higher amount of FLOPS, which in this case will be $9n$

Here is the following implementation of the algorithm using Python :

```
def velocity_verlet(steps, N, m, x, v, dt, G, origin_mass):
            total = (steps-1)*N
            count = 0
            perc = 0
            perc_new = 0

            for i in range(steps-1):
                for j in range(N):
                    r = x[i] - x[i,j]
                    r_norm = np.sqrt(np.sum(r**2, axis = 1))
                    v_half = v[i,j] + 0.5*a_g(m, r, r_norm, G)*dt
                    if origin_mass != 0:
                        x_norm = np.sqrt(np.sum(x[i,j]**2, axis =
    0))

                        v_half = v_half - G*origin_mass*x[i,j]*dt/
    x_norm**3
                    x[i+1,j] = x[i,j] + v_half*dt

                    r = x[i+1] - x[i+1,j]
                    r_norm = np.sqrt(np.sum(r**2, axis = 1))
                    v[i+1,j] = v_half + a_g(m, r, r_norm, G)*dt
                    if origin_mass != 0:
                        x_norm = np.sqrt(np.sum(x[i+1,j]**2, axis =
     0))

                        v[i+1,j] = v[i+1,j] - G*origin_mass*x[i+1,j
    ]*dt/x_norm**3

                    count += 1
                    new_perc = int(100*count/total)
                    if new_perc > perc:
                        perc = new_perc
                        print(perc)
            return x,v
```

4.4 Energy and angular moment

[H] Algorithm for calculating the the total energy in the solar system. [2]

```
function Ek($M_p, v$):
    return $\frac{1}{2}mv^2$
end function
function Ep($M_p, r_{sp}$):
    return $-G\frac{M_sM_p}{r_{sp}}$
end function
EnergyTot = 0
for j=1:PlanetCount do
    Potential = Ep($M_p$[j], $r_{sp}$[j]
    Kinetic = Ek($M_p$[j], $v$[j])
    EnergyTot += Kinetic + Potential
end for
```

The angular momentum can also be calculated using this algorithm. We simpliy implemnt it by summing total angular momentum instead of summing the kinetic and potential energy. Angular is momentum given by:

$$\vec{l} = \vec{r} \times \vec{v} \tag{17}$$

5 Results

We begin with a simple two-body system of the solar system. With this simple model, comes unrealistic results, but as we gradually increase the amount of planets in the system we will get closer to something more realistic model.

5.1 Two-body system

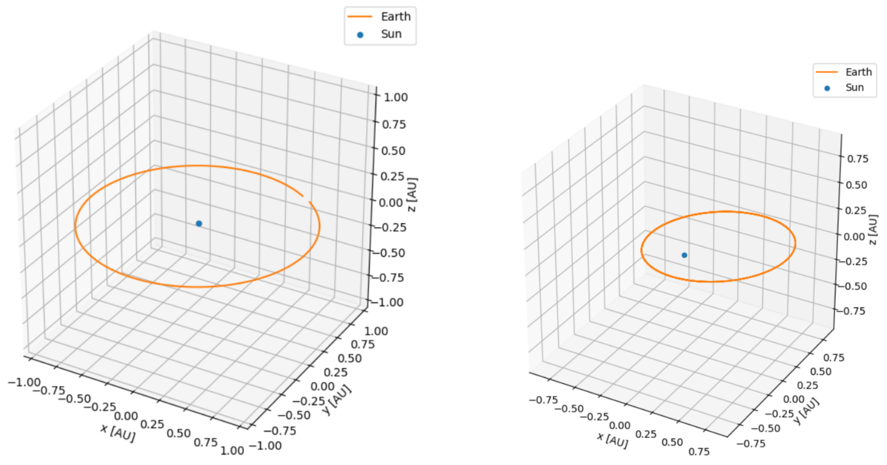

Figure 1: plots that compares the precision between the Forward-Euler method (left) and the Velret method (right) . The time step is $dt = 10^{-4}$ and the number of points is $N = 10^4$

For the stability test result of each method we have the figure 2 below:

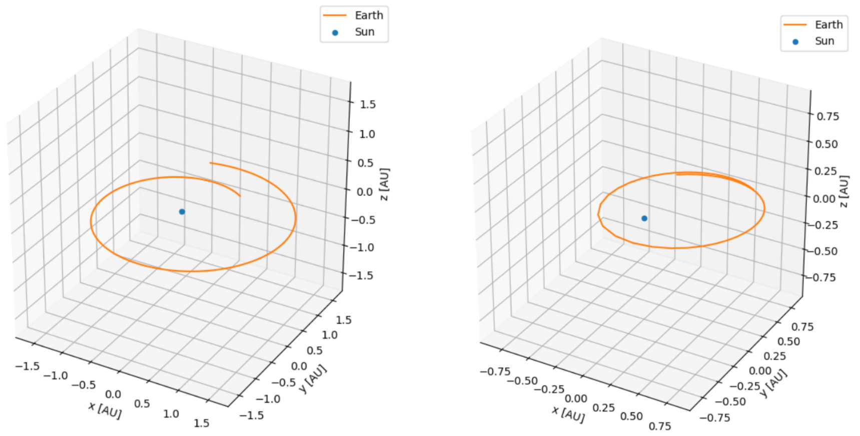

Figure 2: plots that shows the breaking point for Forward-Euler method (left) and the Velret method (right) . The time step is $dt = 10^{-2}$ and the number of points is $N = 50$ (Verlet) and $N = 200$ (Forward Euler)

Table 1: Comparison of time for Forward-Euler and velocity Verlet applied to a two-body system.

N	Forward-Euler [t/s]	Velocity Verlet [t/s]
10^3	4.631	4.092
10^4	4.661	4.125
10^5	5.144	4.601
10^6	10.410	7.515

5.2 Energy and angular moment

Using the algorithm we got the following printout of the energy and angular momentum values in the figure below:

Table 2: The different energies and the angular moment for Earth's orbit. This is done with 10^4 steps.

L	**Forward-Euler/Velocity-Verlet**
E	-0.00005
E_k	$1.96*10^-5$
E_p	$3.93*10^-5$
I	39.46*m

5.3 Earths escape velocity

Since our object oriented code was build in a flexible way, it made it easy for us to get any information we need about a planet by simply changing some values. For example, Earths escape velocity relative to the sun. See figure 3

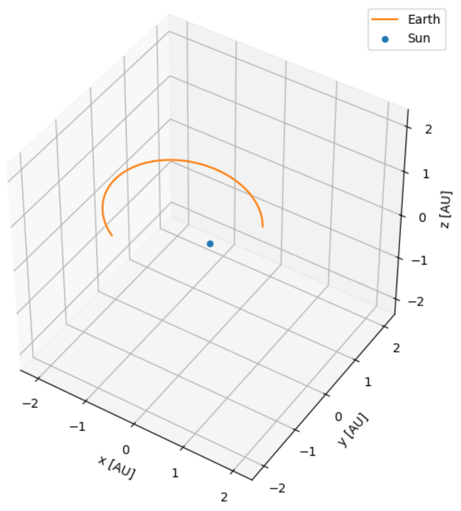

Figure 3: Plot of Earths escape velocity. The value used was $v_{ev} \approx sqrt2\pi$ AU / year

5.4 Three-body system

By using our code, a three-body model with the sun, earth and Jupiter was made. In Figure 6 bellow, you can see how this system looks like using the Velocity-Verlet algorithm.

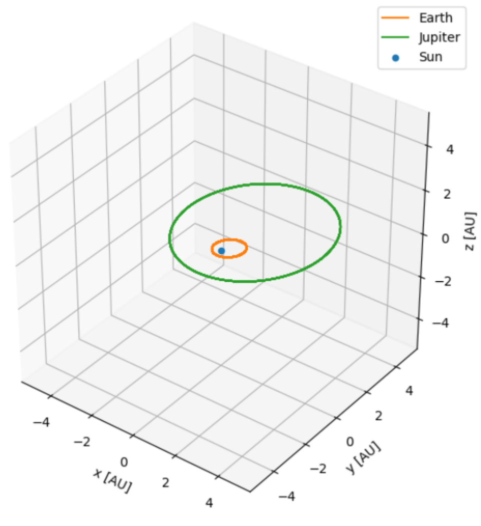

Figure 4: Plot of the Earth-Jupiter-Sun system with the VV-algorithm , $T = 40yr$ and $dt = 10^{-4}$.

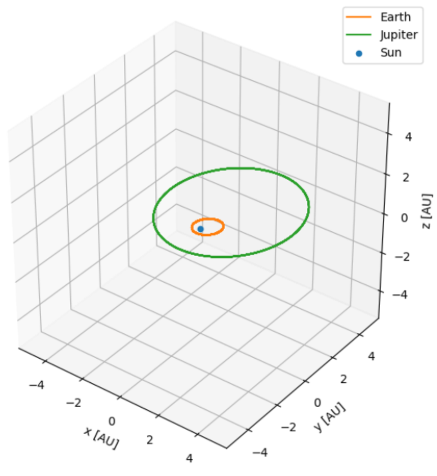

Figure 5: Plot of the Earth-Jupiter-Sun system with the VV-algorithm , $T = 40yr$, $dt = 10^{-4}$ and 10x mass .

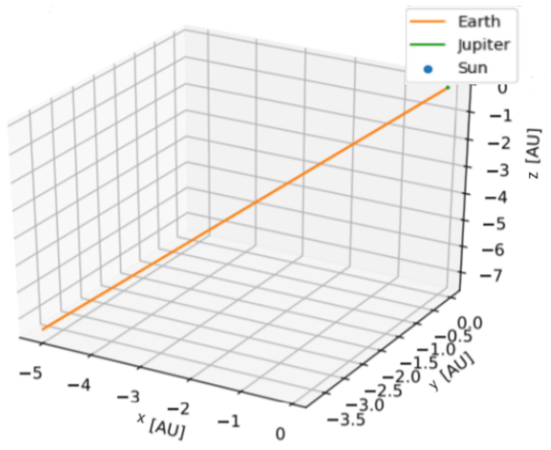

Figure 6: Plot of the Earth-Jupiter-Sun system with the VV-algorithm , $T = 40yr$, $dt = 10^{-4}$ and 1000x mass .

5.5 Multi-body system

Lastly, for the main event, a model for the seven planets in our solar system together with Pluto was constructed. The Velocity-Verlet algorithm was once again used.

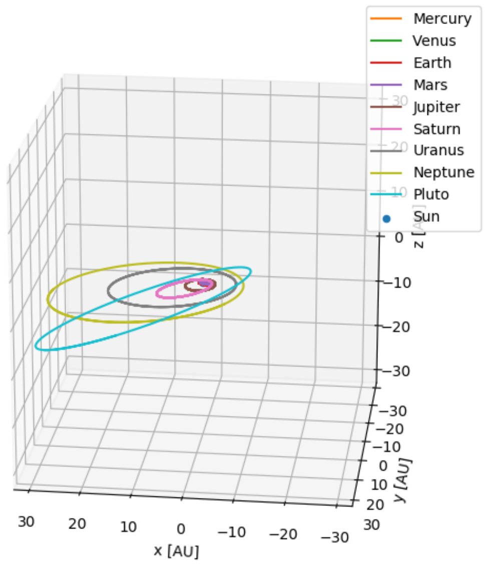

Figure 7: Simulation of the enitre Solar system using Velocity-Verlet algorithm with the values $T = 40yr$, $dt = 10^{-4}$

6 Discussion

6.1 The methods

When we compared our methods, what we noticed was that the Velocity Verlet algorithm was superior when compared to the Euler method in terms of precision. This is clear in figure 1, the Velocity Verlet method has a much narrower orbit than the forward-Euler method, this an indication of precision. The results on the time was however quite surprising (from Table 1). We expected the velocity Verlet would be slower because it has slightly more on the FLOPS count. One very probably reason for this this error in the results, would be small testing sample. When we took a time test, we only took the first result we got. Trying out multiple time tests on the same method and time steps, then taking the average of it, would have yielded more accurate results.

Whether the Euler algorithm or the velocity Verlet is "best" is a matter of context. They both have the advantage of being cheap in terms of time and both have relatively good precision. For this particular project, the results has to be adequate and not necessarily very precise. If we are going to take this into a deeper context the velocity Verlet method comes out on top because of its other properties. The notable property is symplecticity, is the conservation of state-space volume. This is perfect for simulating the require energy to be conserved.

Speaking of conservation, we have shown that the two-body system has conserved energy, this is shown Table 2 from the code output.

6.2 Earths escape velocity + modification of beta

As we can see from section 3.2, Earths escape velocity relative to the Sun is given by $v_{escape} = 2\sqrt{2}\pi \approx 8.89$ AU / year. This analytical result fitted perfectly when applied to our model, which was ≈ 8.89 AU / year. However, given just the methods for finding the escape velocity in the model, one can not say with when the planet will exactly escape.

From section 3.3, by doing our calculations, what we noticed was that changing β may lead to less stable orbit conditions. Also when tested, it appeared that an elliptic (not circular) orbit can not have the value β different than 2. Further, it seemed as the values changed in an oscillating manner. This behaviour, we can not justify.

6.3 The effect of Jupiter on Earth's Orbit

When we moved over to implementing Jupiter to our solar system, it was unsurprisingly, in our interest, to find out how much Earths motion is being altered by Jupiter. This was an interesting experiment because, as we know, Jupiter has the strongest gravitational pull when compared with all the other planets in our solar system.

What we noticed however, was that Jupiter's effect on the Earth was unnoticeable. You can see Figure 4 for the plot. Our conclusion to why Jupiter's interference was unnoticeable, was because the Sun itself is about thousand times more massive.

After our test, we went on to increase Jupiter's mass by ten times to see if that would alter Earth's trajectory. However, even with a ten times more massive Jupiter, it seemed like it was to small of a change to make any noticeable difference.
We therefore lastly tried to increase Jupiter's mass to thousand times it's original. When we did this, massive changes were noticeable. Namely because our Earth's orbit is dependent by two Sun-like masses. This made Earth totally lose it's orbit as seen in Figure 6

6.4 The solar system

In Figure (7), we can see a model of our solar system generated by our model using the Velocity-Verlet algorithm. As you may see, our model has managed to imitate our solar system quite well. With that, it is safe to say that our model is working as it should be and that it is numerically precise, and hence therefore, a good model for this type of project.

6.5 Mercury's Precession

In our project, we also tried testing for the perihelion precession of Mercury. However, our results did not produce something we would consider an simulation of reality. When we tried to discuss the possible errors in our script, we arrived with the some possibilities. The first two possibilities are related to computing power and numerical errors. Due to the weakness of our old laptop, we could not test for small enough step.points, maybe if we could, we would find more reasonable solutions.
Lastly, there could be some problems with interpolation when orbit points where calculated. However, this could not be verified.

7 Conclusion

In this project we have demonstrated the reusable nature of object oriented programming and its flexibility when it comes to modification. We also implemented two methods : the Forward-Euler and the Velocity-Verlet method. The latter proved to be better in terms of stability and accuracy. The velocity Verlet method is also a better candidate because of its symplectic nature, if we consider that part of the project is about conservation. If we look at the results of the Earth's escape velocity, our results where in good correspondence with the analytical solution also using velocity Verlet algorithm here. At last we tried to include relativity for Mercury's orbit, the numerical results where not as expected. When looking at relativistic effects such as the (thing) of Mercury, it appeared that or model may have not been optimized for this type of study. If we would have to build our model from the bottom-up while keeping the relativistic effects in mind, other results that better model the reality may have been modeled.

References

[1] M. Hjorth-Jensen, Computational Physics Lecture Notes Fall 2019 (2019). ørenssenørenssenørenssen

[2] Anders Malthe-Sørenssen, Elementary Mechanics Using Python (2015).

[3] Albert Einstein, The Foundation of the General Theory of Relativity , page 769-822 (1916).