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 Euler-Cromer and the velocity Verlet integration methods. In spite of the the velocity Verlet method using two more FLOPS than the Euler-Cromer 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 planets then we gradually expand to the whole solar system.

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 Euler-Cromer 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 constrain this to be grater than zero and insert values for the quantities in units of AU, years and Sun masses, $G = 4\pi^2$, as we will show later, 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 defines as:

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

From the Appendix **??** we have that for this 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

By doing a Taylor expansion of the field equations of Albert Einstein's general theory of relativity (GR) to first order in $1/c^2$, Newton's gravitational law gets an additional relativistic correction:

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

This extra term will give a contribution to the gravitational force which changes the position of the perihelion point of orbit over time. As the additional term is often negligible, the effect is small, but can still be observed for fast moving planets, with a short period. The contribution to the perihelion precision from GR is given by (**?**)

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

where e is the orbital eccentricity, a the semi-major axis, and T the orbital period. The planet with the smallest orbital period is Mercury. When inserting values for all these quantities, one get that the perihelion precision of Mercury due to relativistic corrections 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 Euler-Cromer 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:

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

[H] Algorithm for computing the the total mechanical energy in a modeled solar system.

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

5 Results

5.1 Two-body system

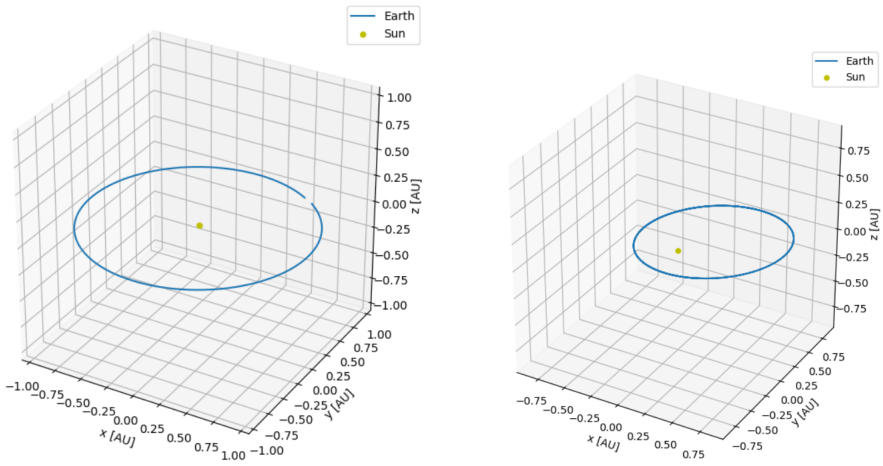

Figure 1: plots that compares the precision between the Forward-Euler method (left) and the Velret method (right) . The time step is $dt = 10^{-4}$ and the number of points is $N = 10^4$

TEST OF STABILITY WITH DIFFERENT DT

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

```
Energies printed in C++:

Energies for Earth-Sun
EULER-CROMER
Angular momentum: 39.4784*m
Step: 1000000  E = -2e-05   Ek =1.9739e-05  Ep =-3.9478e-05
VERLET
Angular momentum: 39.4784*m
```

```
Step: 1000000   E = -2e-05   Ek =1.9739e-05  Ep =-3.9478e-05

Energies for three-body system
Step: 1000000   E = -0.004   Ek = 0.0040197  Ep =-0.0077056
For 10m_j
Step: 1000000   E =  -0.04   Ek =  0.041006  Ep = -0.077361
For 1000m_j
Step: 100000    E =     -4   Ek =     3.314  Ep =     -6.944

Energies for multi-body system dt=1e-5 N=1e5
Step: 100000    E = -0.004   Ek = 0.0040429  Ep =-0.0084804
```

Table 2: The average increase of speed shown by the table below.

L	**Forward-Euler**	**Velocity-Verlet**	**Speed increase**
E	3.92506	2.00267	0.5102
Ep	7.62552	3.66395	0.4804
Ek	12.7915	5.4596	0.4268
I	19.8124	7.95295	0.4014

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 2

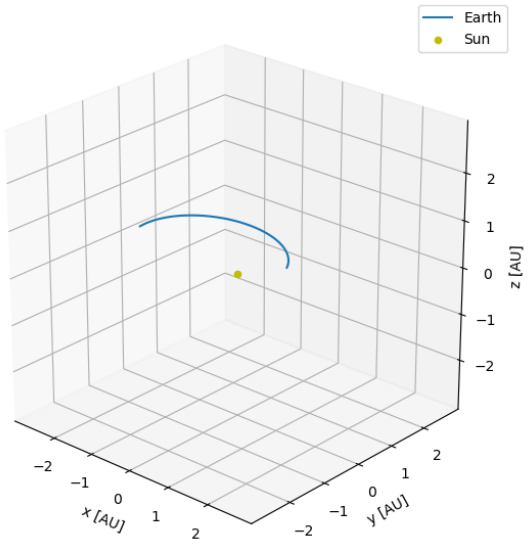

Figure 2: Plot of Earths escape velocity. The value used was $v_{ev} \approx sqrt2\pi$ AU / year

5.4 Three-body system

By using our code, a three-body model with the sun, earth and Jupiter was made. In Figure 3 bellow, you can see how this system looks like using the Velocity-Verlet algorithm.

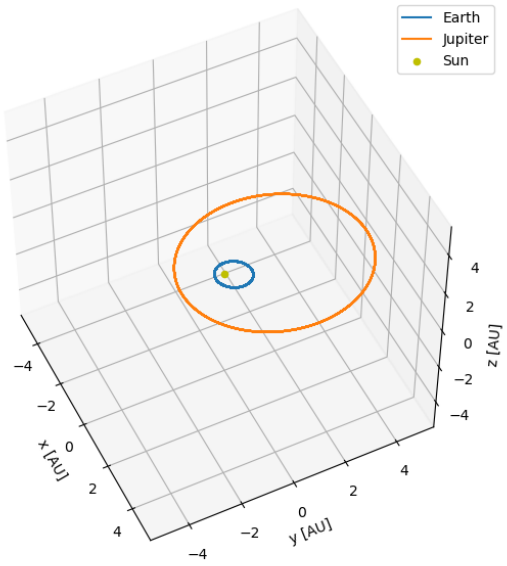

Figure 3: Plot of Earths escape velocity. The value used was $v_{ev} \approx sqrt2\pi$ AU / year

5.5 Multi-body system

5.6 Relativity

6 Discussion

6.1 The methods

We have seen that the velocity Verlet algorithm is much more precise than the Euler-Cromer method (figure **??**). In addition it appears that it may also be quicker (table **??**), which is a surprise since the Verlet algorithm seems to include more FLOPS than the Euler-Cromer method. This leads us to believe that the velocity Verlet may not be that much faster, at least we can conclude that the results are not statistically significant since we only have one observation per method per value for N. An improvement on this study would have been to conduct such a test. We propose larger sampling of code run time and, for simplicity, a paired Student's t-test as a start to answer this question(**?**).

The discussion of an Euler method versus a Verlet method is a rather big one. Both methods are quite quick cheap which is what you want, and they are

usually the two methods of choice if one does not need entirely exact results, but results that are "good enough". A field where this discussion prominent is in physics engine design for video games. With more computing power one would probably go for a more sophisticated method like a foruth degree Runge-Kutta method(**?**).

It is worth mentioning something about the symplecticity of the velocity Verlet method. As stated previously, a symplectic method will be good at simulating systems with energy conservation. We believe that the solar system has conserved energy, but our model of it may not be. Looking further into this matter we have found that both energy and angular momentum is conserved for the two-body Earth-Sun system. This would warrant further use of the velocity Verlet scheme. See appendix 5.2 for a sample readout of energies and angular momenta.

6.2 Two-Body method

6.3 Earths escape velocity

6.4 Three-Body method

6.5 Multi-Body method

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. From our testing, it seems like this model fits very well with reality.

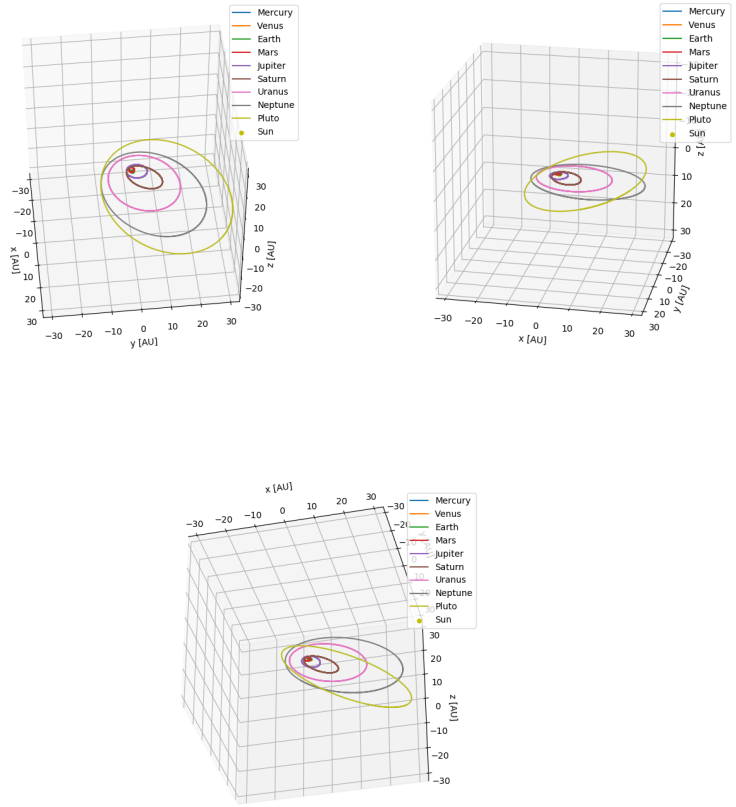

Figure 4: yooo

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. MORE TO ADDThe latter proved to be better in terms of stability and accuracy At last we tried to include relativity for Mercury's orbit, the numerical results where not as expected.....MORE TO ADD

COPIED """ We have implemented two integrators, the Euler-Cromer and the velocity Verlet. The velocity Verlet proved to be more stable than the Euler-

Cromer method. The reason for this is because Verlet integration methods are symplectic and therefore perfect for system with conserved angular momentum and energy. Additionally, the velocity Verlet method seemed in theory to be more CPU costly than the Euler-Cromer method, but proved just as fast.

By solving coupled differential equations, through discretizing the continuous equations and using natural units to scale them, we have been able to reproduce analytical results and solve analytically unsolvable problems for the Solar system. Using the results we are able to quantiavly compare two algorithms, forwarad Euler and velocity Verlet. For the Solar system both energy and angular momentum should be conserved. We find that velocity Verlet conserve both of these quantites a factor of more than 10^4 more accurately, as well as producing more accurate positions, as well as it's accuracy not deteriorating as we add more celestial bodies. The only downside to using velocity Verlet over forward Euler is the number of FLOPS, which is around a factor of two larger, resulting in around twice the computational time. Still velocity Verlet is superior due to it's accuracy.
Studying the three body problem of the Sun, the Earth and Jupiter, we test our assumption that the center of mass is fixed at the center of the Sun. We find that this assumption changes the position of planets by around 0.008AU, and should not be assumed to get precice results. When computing the center of mass of the Sun-Jupiter system, we find the center of mass to be 0.005AU away from the origin, which is quite consistent with what we observed.
Using the bisection method we are able to find the escape velocity of the Earth. Analytically the escape velocity is $2\sqrt{2}\pi \approx 8.8857$AU/year, numerically we find it to be equal to 8.8851(5), which results to a relative deviation of 0.0068%.
We alter Newton's gravitational law and check the stability of the Earth's orbit as the exponent approches 3. We find that Earth's orbit becomes more and more unstable the larger the exponent becomes. When the exponent is exactly equal to 3 the orbit of the Earth is unstable, and diverges. This is tested versus analytical results from calssical mechanics, and we find complete accordance.
By Taylor approximation we add a relativistic term from Einstein's general theory of relativity to Newton's gravitational law. We find that with this additional tem we are able to reproduce the analytical value of Mercury's perihelion precision of $42.98''$ per centrury, numerically with $42.91(1)''$ per centruy, which is a relative deviation of 0.17%.
Using object orientation we can easily expand the Solar system to an N-body problem. And we compute the path of all planets, and some moons over the coming century. """

References

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

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