FYS4150 - Project 3

Simen Nyhus Bastnes & Eirik Ramsli Hauge

24. October 2016

Abstract

In this project we will use Euler's forward method and Velocity Verlet to simulate the solar-system from given initial values. We will start off by looking at a solar system which only contains the Earth and the Sun, and then expand it by including other planets, starting with Jupiter. We will also include tests for conservation of kinetic energy, potential energy, momentum and angular momentum and find the escape velocity of the Earth in a single planet solar system. Lastly, we will add a general relativistic correction to the gravitational force, and compare the perihelion angle with that given from classical Newtonian gravitation. We found that Velocity Verlet is quite stable for time steps as large as $dt = 10^{-2}$, while Forward Euler requires $dt = 10^{-4}$ to reproduce the same results. The Velocity Verlet algorithm was found to be approximately twice as slow as Euler. The initial velocity of the Earth should be $v = 2\pi$ and the escape velocity $v = 2\sqrt{2}\pi$. Adding Jupiter doesn't change Earth's motion much, but increasing Jupiter's mass does. Adding more planets was not hard as the code was object oriented. The difference in perihelion angle was 43 arcseconds between the general relativity case and the classical one.

Introduction

Differential equations are a big part of physics, and one of the most known differential equation in physics is Newton's second law of motion. Many of our known differential equations can not be solved analytically for more than a few special cases. However, with the advent of computers, we have been able to solve increasingly complex systems of differential equations where no analytical solution exists, by utilizing the numerical power and speed of computers. To be able to solve differential equations numerically is therefore a great tool for any physicist, and this project is designed to give us a basic understanding of different methods for solving differential equations numerically, as well as how to write object oriented code in C++.

Our aim with this project is to simulate our own solar system with the two numerical methods, forward Euler and Velocity Verlet. We will start with a system only containing the Sun and the Earth and compare our two methods. Afterwards we will run a test to see if the values for kinetic energy, potential energy, momentum and angular momentum are conserved. Thirdly we will see if we can find the escape velocity of the Earth with trial and error, and check how that escape velocity matches what we expect analytically.

The Sun-Earth system will be the basis for the rest of our solar system. When we are sure the smaller system works, we will add more planets (and Pluto) starting with Jupiter. Using object orientation, and the fact that the gravitational force is the only force acting between the celestial bodies,

we will find the sum of all forces working on a celestial body from the other celestial bodies. By accessing initial conditions from NASA [3] we will attempt to simulate a relatively accurate solar system.

In the final part of the project we will add a general relativistic correction to the gravitational force and compare the perihelion angle between the classic and the relativistic corrected case for a Sun-Mercury system. If the difference is about 43 arcseconds after a century, we will have an affirmation of Einstein's theory of general relativity.

Theory

The goal of this project, is to develop a model of our solar system using the so-called velocity Verlet algorithm for solving coupled ordinary differential equations. For our solar system model, the only force interacting on it is gravity.

The Earth-Sun system

To start off, we simplify by looking at a hypothetical solar system with only the Earth orbiting around the Sun, where any solar motion is small enough to be neglected. Since the only force working on the system is the gravitational force between the Sun and the Earth, Newton's law of gravitation gives us that

$$F_G = \frac{GM_\odot M_\mathrm{E}}{r^2}$$

where $M_\odot$ is the solar mass, M_E is the mass of the Earth, G the gravitational constant, and r is the distance between the Earth and the Sun. Written on vector form, F_G takes the following form

$$F_{G,\mathbf{r}} = -\frac{GM_\odot M_\mathrm{E}}{r^3}\,\mathbf{r}$$

where $\mathbf{r} = x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$ is the vector from the Sun to the Earth.

$$F_{G,\mathbf{r}} = -\frac{GM_\odot M_\mathrm{E}}{r^3}\,(x\mathbf{i} + y\mathbf{j} + z\mathbf{k}) \tag{1}$$

Using Newton's second law of motion, we can then get the following differential equations for the motion of the Earth.

$$\frac{d^2x}{dt^2} = \frac{F_{G,x}}{M_E} \tag{2}$$

$$\frac{d^2y}{dt^2} = \frac{F_{G,y}}{M_E} \tag{3}$$

$$\frac{d^2z}{dt^2} = \frac{F_{G,z}}{M_E} \tag{4}$$

where $F_{G,x}$, $F_{G,y}$, and $F_{G,z}$ are the components of the gravitational force, given by (1). We can assume that the orbit of the Earth (and other objects in the solar system) around the Sun is mostly co-planar, which we could take to be the xy-plane, reducing our differential equations to equations

(2) and (3). However, since looking at the system in three dimensions isn't that much more work, we will continue doing so.

We can obtain mass units from assuming that the Earth's orbit is almost circular around the Sun. For circular motion, we know that the force obeys the following relation

$$F_G = \frac{M_{\mathrm{E}} v^2}{r} = \frac{G M_\odot M_{\mathrm{E}}}{r^2}$$

where v is the velocity of the Earth. This can be shown to give us the useful relation

$$v^2 r = G M_\odot = 4\pi^2 \mathrm{AU}^3/\mathrm{yr}^2 \tag{5}$$

which we can use to scale both distance and time units to more ideal orders of magnitudes. One astronomical unit (1 AU) is defined as the average distance from the Earth to the Sun, or roughly 10^{11} m. Inserting this into equation (1), gives us

$$F_{G,\mathbf{r}} = \frac{4\pi^2 \mathrm{AU}^3/\mathrm{yr}^2 \, M_{\mathrm{E}}}{r^3}(x\mathbf{i} + y\mathbf{j} + z\mathbf{k}) \tag{6}$$

where r now is given in units AU, and time is given in years. Using years instead of seconds matches the time evolution of our solar system better, as the Earth for example orbits the Sun once per year. By looking at equation (5), we realize that the velocity required for circular orbit falls out

$$\begin{aligned} v_{\mathrm{circ}}^2 r &= 4\pi^2 \mathrm{AU}^3/\mathrm{yr}^2 \\ v_{\mathrm{circ}} &= 2\pi \sqrt{\frac{\mathrm{AU}^3}{r}}\, \mathrm{yr}^{-1} \end{aligned} \tag{7}$$

For our Earth, which lies at $r = 1$ AU, equation (7) gives us $v_{\mathrm{circ}} = 2\pi$.

The total energy of the Earth can be given as $E_{\mathrm{tot,E}} = T_{\mathrm{E}} + U_{\mathrm{E}}$, where

$$T_{\mathrm{E}} = \frac{1}{2} M_{\mathrm{E}} v_{\mathrm{E}}^2$$

is the kinetic energy for the Earth, and the potential energy U of the Earth in our system is given as

$$\begin{aligned} U_{\mathrm{E}} &= \frac{G M_\odot M_{\mathrm{E}}}{r} \\ &= \frac{4\pi^2 \mathrm{AU}^3/\mathrm{yr}^2 \, M_{\mathrm{E}}}{r} \end{aligned}$$

where r is the distance given in AU. From this we can find the escape velocity, which is the initial velocity an object needs to escape from its orbit. This occurs when the kinetic energy is equal the potential energy. For the Earth, this is

$$\begin{aligned} T_{\mathrm{E}} &= U_{\mathrm{E}} \\ \frac{1}{2} M_{\mathrm{E}} v_{\mathrm{esc}}^2 &= \frac{G M_\odot M_{\mathrm{E}}}{r} \\ v_{\mathrm{esc}} &= \sqrt{\frac{2 G M_\odot}{r}} \end{aligned} \tag{8}$$

Inserting $r = 1$ AU and our result from equation (5), we get

$$v_{\text{esc}} = 2\sqrt{2}\,\pi\,\text{AU/yr}$$

or expressed in terms of the circular orbit velocity $v_{\text{esc}} = \sqrt{2}\,v_{\text{circ}}$.

In order to solve the differential equations (equations (2), (3), and (4)) numerically, we need to discretize them. For simplicity, we will only derive the discretized expression for the x-direction, though the derivation is the same for the other directions. In the non-discretized situation, the position of an object in x is given by $x(t)$, where t is the time. We start by discretizing the time by defining the time steps t_i in the interval $t_0 \dots t_n$, so

$$t_i = t_0 + ih$$

where h is the step length and $i = 0, 1, \dots, n$, so that we can write $t_{i+1} = t_i + h$. Now, we can write $x(t)$ in terms of the discretized time, $x(t_i)$. We can simplify the notation by writing $x(t_i)$ as x_i. Inserting this into equation (1) gives us

$$F_{G,x_i} = -\frac{GM_\odot M_{\text{E}}}{r^3}\,x_i$$

We then get the discretized differential equation for x from equation (2)

$$x_i'' = -\frac{GM_\odot}{r^3}x_i$$

Or when inserting the expression for $GM_\odot$ from equation (5)

$$x_i'' = -\frac{4\pi^2\text{AU}^3/\text{yr}^2}{r^3}\,x_i \tag{9}$$

Numerical methods for solving ordinary differential equations

In order to solve differential equations like equation (9), we need to use some numerical algorithm. Euler's forward method [1], is a well known numerical method used to solve differential equations. The method is fairly simple, basing itself on that the next step is the former step added with its own derivative times a steplength. By defining h as the step length we can write Euler's forward method as follows:

$$f_{i+1} = f_i + f_i^{'} \cdot h + \mathcal{O}(h^2) \tag{10}$$

where we define f_i as $f(i)$, $f_{i+1} = f(i+h)$ and $i = 0, 1, 2..., n$.

In our case we use Newton's second law and we want to find the position and velocity of our celestial bodies. Since it's easy to find the acceleration from equation (6), and we know that $a = \frac{\partial^2 x}{\partial t^2}$ and $v = \frac{\partial x}{\partial t}$ where x is position, v is velocity and a is acceleration, Euler's forward method becomes:

$$v_{i+1} = v_i + h \cdot a_i + \mathcal{O}(h^2) \tag{11}$$

$$x_{i+1} = x_i + h \cdot v_i + \mathcal{O}(h^2) \tag{12}$$

We can implement the algorithm as follows:

Algorithm 1 Forward Euler

1: **for** $i = 0, n$ **do**
2: $\quad v_{i+1} = v_i + h \cdot a_i$
3: $\quad r_{i+1} = r_i + h \cdot v_i$
4: **end for**

Where we calculate F_{G,r_i} with equation (6). We count the floating point operations of this algorithm to be $4n$ FLOPs.

The Velocity Verlet method presents a different approach to solve the differential equations in this project. While Euler's method is applicable to many differential equations, the Verlet methods are specified to solve Newton's equations of motion [4]. The Velocity Verlet method is derived by first Taylor expanding x_{i+1} ad x_{i-1}:

$$x_{i-1} \simeq x_i - v_i h + \frac{1}{2} a_i h^2$$
$$x_{i+1} \simeq x_i + v_i h + \frac{1}{2} a_i h^2$$

By adding the two expressions we can find an expression for x_{i+1}

$$x_{i+1} + x_{i-1} = 2x_i + a_i h^2$$
$$x_{i+1} = 2x_i - x_{i-1} + a_i h^2$$

and by subtracting them we can find an expression for v_i

$$x_{i+1} - x_{i-1} = 2hv_i$$
$$v_i = \frac{x_{i+1} - x_{i-1}}{2h}$$

These expressions for x_{i+1} and v_i are called the Verlet method. However, we want to have an expression for v_{i+1}.

$$v_{i+1} = \frac{x_{i+2} - x_i}{2h}$$

We can easily find x_{i+2} by using the expression above

$$x_{i+2} = 2x_{i+1} - x_i + a_{i+1} h^2$$

Inserting this back into v_{i+1} gives us:

$$v_{i+1} = \frac{2x_{i+1} - x_i + a_{i+1} h^2 - x_i}{2h}$$
$$v_{i+1} = \frac{x_{i+1} - x_i}{h} + \frac{1}{2} a_{i+1} h$$

Inserting the Taylor expanded x_{i+1} for v_i dependency

$$v_{i+1} = \frac{x_i + v_i h + \frac{1}{2}a_i h^2 - x_i}{h} + \frac{1}{2}a_{i+1}h$$
$$v_{i+1} = v_i + \frac{1}{2}a_i h + \frac{1}{2}a_{i+1}h$$

This gives us the equations for position and velocity in the Velocity Verlet method:

$$x_{i+1} = x_i + v_i h + \frac{1}{2}a_i h^2 + \mathcal{O}(h^3) \tag{13}$$

$$v_{i+1} = v_i + \frac{1}{2}h(a_i + a_{i+1}) + \mathcal{O}(h^3) \tag{14}$$

As we can see, the term for v_{i+1} is dependent on a_{i+1}. Therefore the algorithm must go like this:

Algorithm 2 Velocity Verlet

1: **for** $i = 0, n$ **do**
2: $r_{i+1} = r_i + v_i h + \frac{1}{2}a_i h^2$
3: $a_{i+1} = F_{G,r_{i+1}}/m$
4: $v_{i+1} = v_i + \frac{1}{2}h(a_i + a_{i+1})$
5: **end for**

Where we calculate $F_{G,r}$ with equation (6). By counting the floating point operations, we can see that it goes like $11n$ FLOPs which is a bit more than the double of the forward Euler algorithm.

The three-body problem

Up until now, we have only looked at a simple Earth-Sun solar system. As our own solar system contains a few more celestial bodies of significant size, we should expand upon our model by adding more planets, and see how that affects the stability of the Earth's orbit.

For simplicity, we will start by adding Jupiter, as it is both the largest planet, and fairly close to the Earth. Looking at the Earth's motion, we note that adding another body changes the forces affecting Earth. The forces acting on the Earth is then the gravitational force between Earth and the Sun, as well as the force between Earth and Jupiter. The first we found earlier as equation (6), and the interaction between the Earth and Jupiter can be written as

$$F_{\text{E-J}} = \frac{GM_{\text{E}}M_{\text{J}}}{r_{\text{E-J}}^2} \tag{15}$$

where M_{J} is the mass of Jupiter, M_{E} is the mass of the Earth. G is the gravitational constant, and $r_{\text{E-J}}$ is the distance between Earth and Jupiter. We can use equation (5) to express equation (15) in more appropriate units.

$$F_{\text{E-J}} = \frac{GM_{\text{E}}M_{\text{J}}}{r_{\text{E-J}}^2}\frac{M_\odot}{M_\odot}$$

Written on vector form

$$= -\frac{GM_\odot M_{\text{E}}}{r_{\text{E-J}}^3}\frac{M_{\text{J}}}{M_\odot}\,\mathbf{r}_{\text{E-J}}$$

Like before, we will for simplicity only look at the x-direction

$$F_{\text{E-J},x} = \frac{4\pi^2\text{AU}^3/yr^2\,M_{\text{E}}}{r_{\text{E-J}}^3}\,x_{\text{E-J}}$$

We now use Newton's second law to give us the updated differential equation for x.

$$\frac{d^2x}{dt^2} = \frac{F_{G,x} + F_{\text{E-J},x}}{M_{\text{E}}}$$

Written fully out, we get the differential equation

$$\frac{d^2x_{\text{E}}}{dt^2} = -4\pi^2\text{AU}^3/\text{yr}^2\left(\frac{x_{\text{E}}}{r_{\text{E}}^3} + \frac{M_{\text{J}}}{M_\odot}\frac{x_{\text{E-J}}}{r_{\text{E-J}}^3}\right) \tag{16}$$

which can be expanded to three dimensions and discretized like discussed earlier.

We observe that when adding another planet (planet 2) to the system, the changes to the motion of an object (planet 1) can be expressed by adding a term on the form $4\pi^2\text{AU}^3/\text{yr}^2(M_2/M_\odot)\cdot(x_{21}/r_{21}^3)$, where M_2 is the mass of planet 2, and x_{21} and r_{21} is the distance from planet 2 to planet 1.

Perihelion precession

Lastly we want to study the perihelion angle of Mercury. This was an important test of the general relativity. When one compared the observed value of the perihelion precession to all classical effects, there was a difference of about 43 arcseconds after a century. This difference was not understood until Einstein predicted it with general relativity [5]. Einstein found that the perihelion precession could be predicted by:

$$\epsilon = 24\pi^3\frac{\alpha^2}{T^2c^2(1-e^2)} \tag{17}$$

Where α is the major semi-axis of the planets orbit, e is the eccentricity of the orbit, T the period of revolution and c the speed of light. By inserting the right values for Mercury, it was found that for a year the precession was about 0.104 arcseconds and after a (Earth) century, Mercury would have revolved about 415 times around the sun giving a precession of about 43 arcseconds which coincides with the difference between the observed and classically calculated value thus proving general relativity.

It became clear that the Newtonian force of gravity needed a relativistic correction. In our numerical calculations, we will therefore add a correction setting the gravitational force to:

$$F_G = \frac{GM_\odot M_{Mercury}}{r^2}\left[1 + \frac{3l^2}{r^2c^2}\right] \tag{18}$$

Where l is the orbital angular momentum given as $l = |\vec{r} \times \vec{v}|$ for Mercury and c is once again the speed of light. We can then find the perihelion angle by

$$\tan \theta_p = \frac{y_p}{x_p} \tag{19}$$

where y_p and x_p are y and x respectively at perihelion.

Experimental

The programs used in this project can be found in the GitHub repository [2], in the `/src/` folder. When running the program, it takes 4 command line arguments, `task`, dt, `task_parameter` and `method`, however the last two are optional. The program has four different runmodes, which we will discuss in some more detail. All data written to file by the program are stored in `/benchmarks/` with different subfolders for each runmode. For every `task`, `method` can be set to either Euler, Verlet or VerletREL where the two first uses forward Euler or Velocity Verlet respectiveley, and the latter uses Velocity Verlet with a relativistic correction to the gravitational force.

During the development stage of the code, the total kinetic and potential energy for the system, as well as the momentum and angular momentum were calculated to make sure they are conserved. For every 10 or 100 iterations (depending on dt), the values were printed out in terminal, and checked to see if the properties were conserved.

The Earth-Sun system

Setting `task` to `ES` runs the program for a two-body solar system containing the Earth and the Sun, which is fixed in origo. The Earth starts at a distance of 1 AU from the Sun along the x-axis and `task_parameter` gives its initial velocity in the y-direction. The position vectors of the Sun and the Earth are saved in the file `ES/pos_<method>_dt<dt>_yrs<number of years>_vy<initial velocity in y-direction>.xyz`
where everything inside a `<>` means the value given for that run through.

In order to calculate the CPU time for the Earth-Sun system, the writing to file is commented out, as writing to file is slow. The code required for the timing is commented out as it is unnecessary to do so while running the code normally and writing positions to file.

The Earth-Sun-Jupiter system

Setting `task` to `ESJ1` runs the program for a three-body solar system containing the Earth, Jupiter and the Sun which is fixed in origo. Setting `task` to `ESJ2` sets origo as center-of-mass for the same system, and alters the momentum of the Sun so that the total momentum of the entire system is zero. `task_parameter` gives a factor to multiply with Jupiters mass. Positions of all celestial bodies are written either
`ESJ1/pos_dt<dt>_yrs<number of years>_m<mass factor>.xyz` or
`ESJ2/pos_dt<dt>_yrs<number of years>_m<mass factor>.xyz` depending on `task`.

The whole Solar System

Setting `task` to `WS` runs the program for a solar system containing the Sun, all the planets and Pluto revolving around the centre of mass. `task_parameter` sets the number of years one wants the simulation to run. The position of all celestial bodies are written to the file
`WS/pos_dt<dt>_yrs<number of years>.xyz`

Perihelion precession

Setting `task` to `GR` runs the program for a two-body solar system containing Mercury and the Sun. `task_parameter` sets the number of years one wants the simulation to run. The perihelion angle is saved to the file `GR/ang_per<method>_dt<dt>`

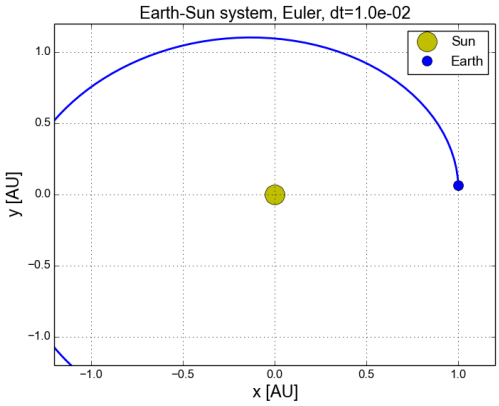
An error while writing to file

One of the main goals of our implementation, is to write the positions of the celestial bodies to file, however for some unknown reason, the first positions saved to file is not always the initial positions.

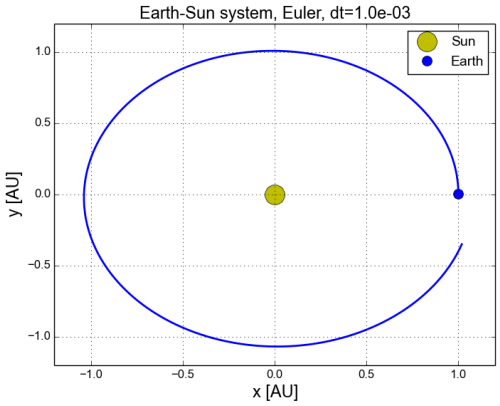
Results

The Earth-Sun system

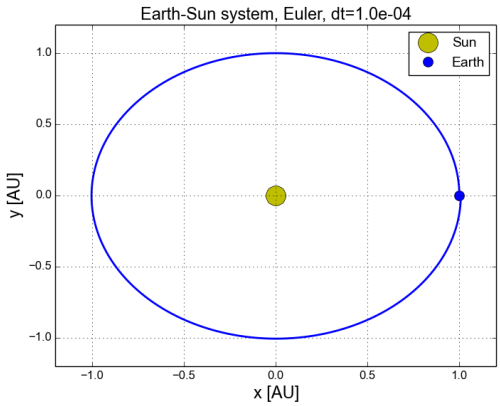
By initializing a system containing the Earth and the Sun and simulating it using forward Euler with different time steps we got the plots shown in figure 1. By doing the same only with Velocity Verlet we achieved the plots in figure 2.



(a) dt = 1.0e-02

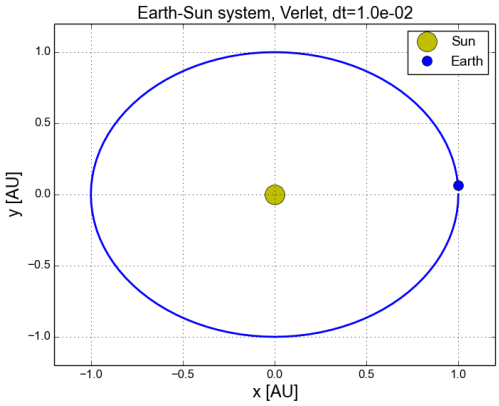


(b) dt = 1.0e-03

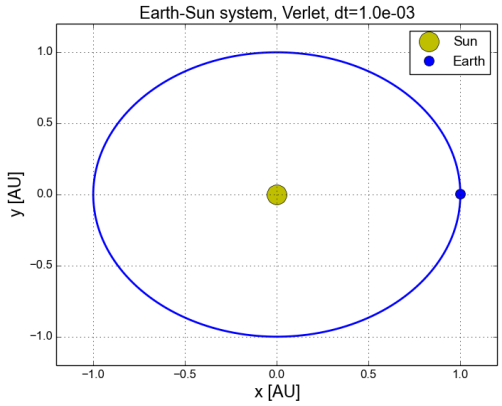


(c) dt = 1.0e-04

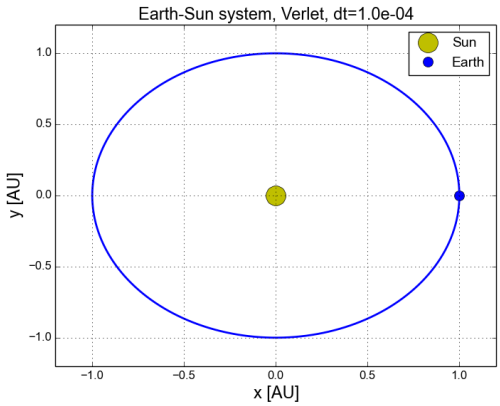
Figure 1: Simulation of the Earth-Sun system using forward Euler with different time steps. The Earth's initial velocity is in all cases set to 6.28 AU/years in the y-direction, the Sun is fixed in origo and the Earth starts in (1, 0, 0) AU



(a) dt = 1.0e-02



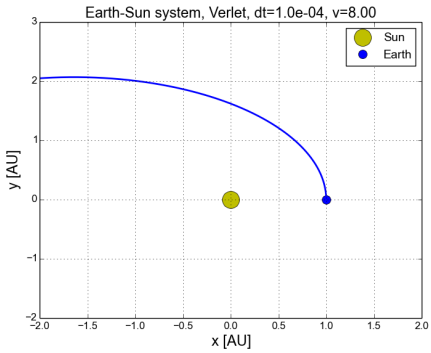
(b) dt = 1.0e-03



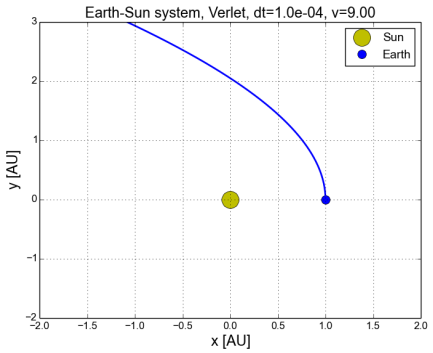
(c) dt = 1.0e-04

Figure 2: Simulation of the Earth-Sun system using Velocity Verlet with different time steps. The Earth's initial velocity is in all cases set to 6.28 AU/years in the y-direction, the Sun is fixed in origo and the Earth starts in (1, 0, 0) AU

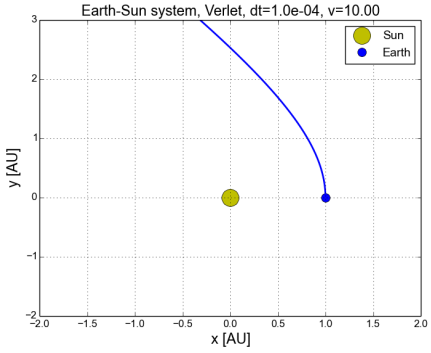
Analytically we found that the Earth's escape velocity was 8.89 AU/years. Some of our trial and errors are shown in figure 3:



(a) $v = 8.00$ AU/years



(b) $v = 9.00$ AU/years



(c) $v = 10.00$ AU/years

Figure 3: Simulation of the Earth-Sun system using Velocity Verlet with different initial velocity for the Earth in the y-direction. The Sun is fixed in origo and the Earth starts in (1, 0, 0) AU. Simulates a time period of two years.

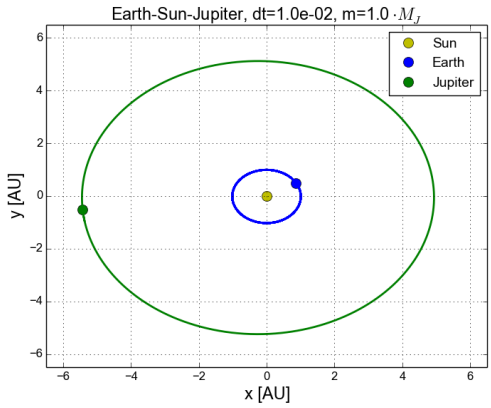
The differences in execution time for the forward Euler and velocity Verlet algorithms for various time steps are shown in table 1.

Table 1: Execution time for the two numerical algorithms for time steps dt. All values are given in units [second].

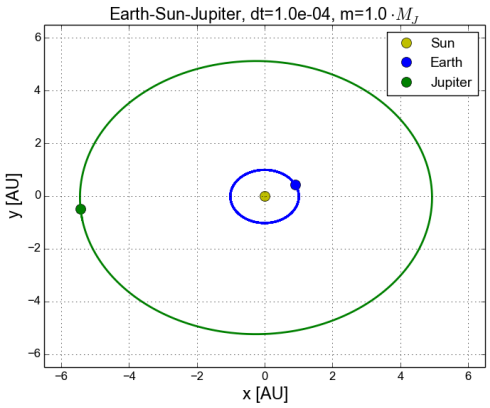
dt	Forward Euler	Velocity Verlet
1e-1	4.00e-5	4.10e-5
1e-2	1.23e-4	2.14e-4
1e-3	9.56e-4	2.85e-3
1e-4	8.63e-3	1.55e-2
1e-5	9.03e-2	1.58e-1
1e-6	0.896	1.542
1e-7	11.8	18.4
1e-8	83.8	150.7

The Earth-Sun-Jupiter system

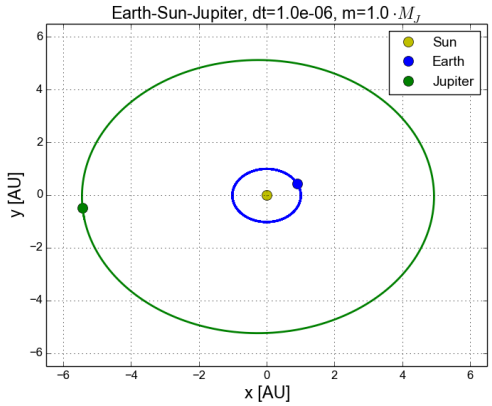
Adding Jupiter to our system and keeping the Sun in origo we achieved the plots shown in figure 4. By letting the Sun move around the center-of-mass, we got the plots in figure 5. The plots in figure 6 were drawn by letting the mass of Jupiter increase while the Sun was fixed in origo. An animation of the events leading up to figure 6c can be found in `/figures/esj.gif` in our GitHub repository [2].



(a) dt = 1.0e-02

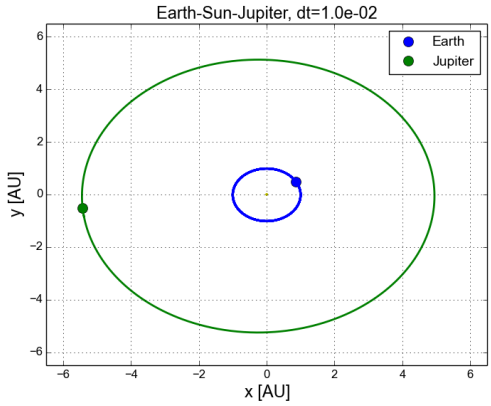


(b) dt = 1.0e-04

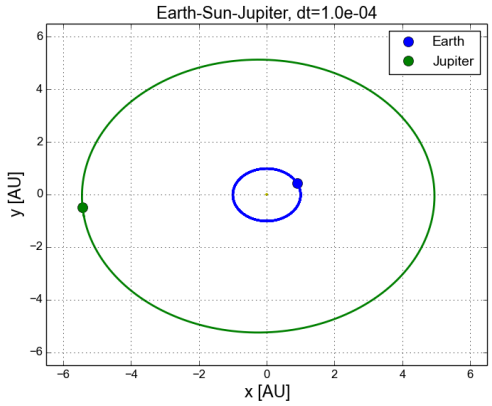


(c) dt = 1.0e-06

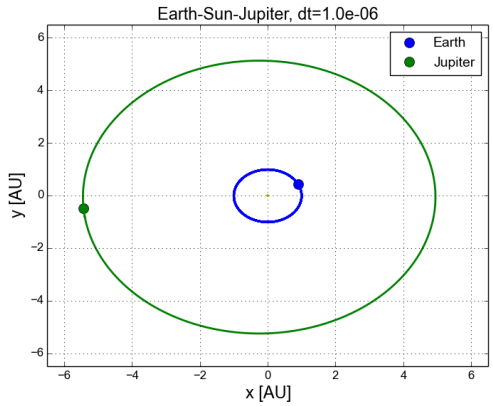
Figure 4: Simulation of the Earth-Sun-Jupiter system using Velocity Verlet with different time steps. Initial velocity and position for the Earth and Jupiter are retrieved from NASA [3] and the Sun is fixed to Origo.



(a) dt = 1.0e-02

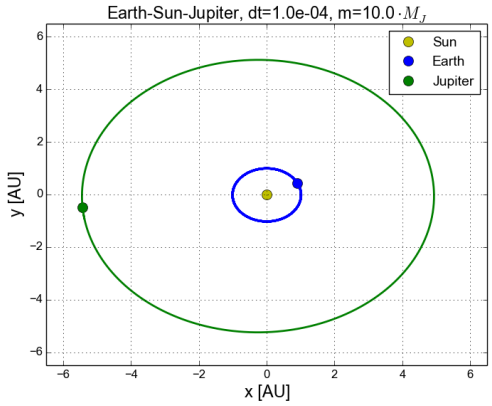


(b) dt = 1.0e-04

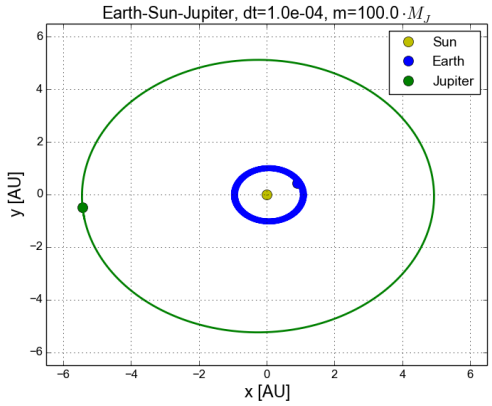


(c) dt = 1.0e-06

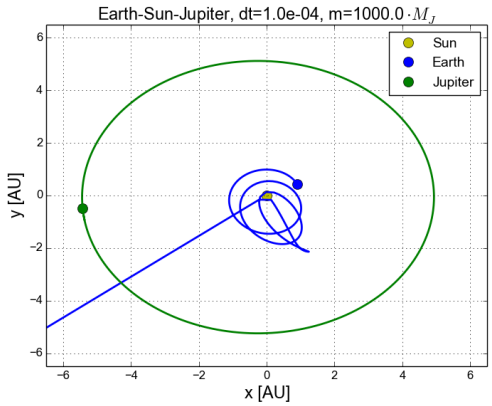
Figure 5: Simulation of the Earth-Sun-Jupiter system using Velocity Verlet with different time steps. Initial velocity and position for the Earth, the Sun and Jupiter are retrieved from NASA [3] and the center-of-mass is in origo.



(a) 10 Jupiter masses



(b) 100 Jupiter masses

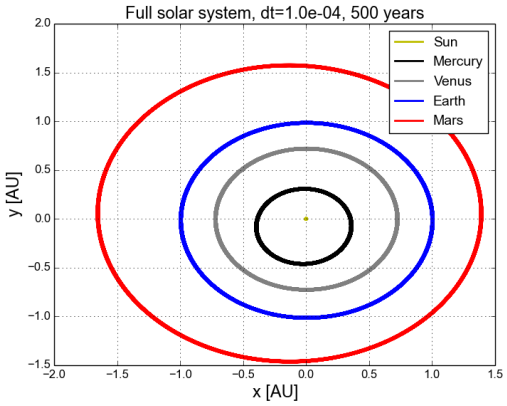


(c) 1000 Jupiter masses

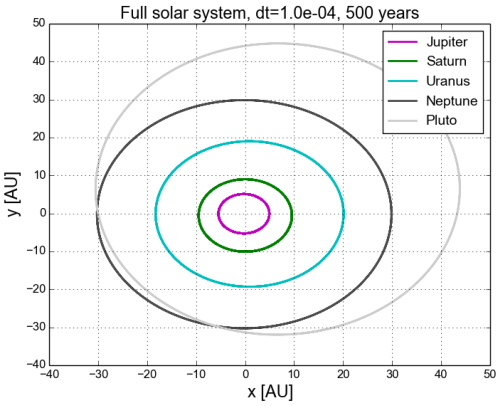
Figure 6: Simulation of the Earth-Sun-Jupiter system using Velocity Verlet with different mass of Jupiter. Initial velocity and position for the Earth and Jupiter are retrieved from NASA [3] and the Sun is fixed to Origo.

The whole Solar System

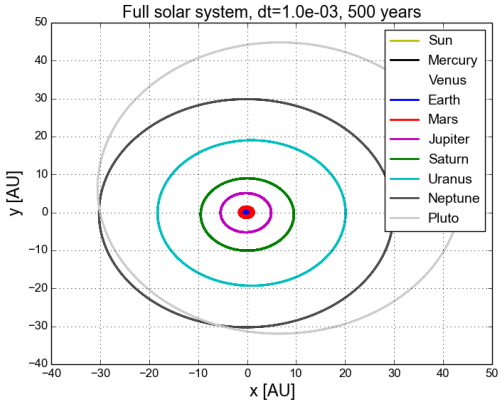
Simulating the Solar System with the Sun, all eight planets and Pluto gave us the results presented in figure 7



(a) Inner Solar System, $dt = 1.0e - 04$



(b) Outer Solar System, $dt = 1.0e - 04$



(c) Whole Solar System, $dt = 1.0e - 03$

Figure 7: Simulation of the whole Solar System including Pluto using Velocity Verlet with different time steps. Initial velocity and position for all celestial bodies are retrieved from NASA [3] and the center-of-mass is in origo. Simulates a time span of 500 years.

Perihelion Precession

By using Verlet twice, first with acceleration given by equation (6) and then by the component version of equation (18), we can compare the perihelion angles. With dt = $1e-7$, initial velocity in y-direction for Mercury = 12.44 AU/year, initial position for Mercury = (0.3075, 0, 0) and letting the simulation run for a century we found the results presented in figure 8

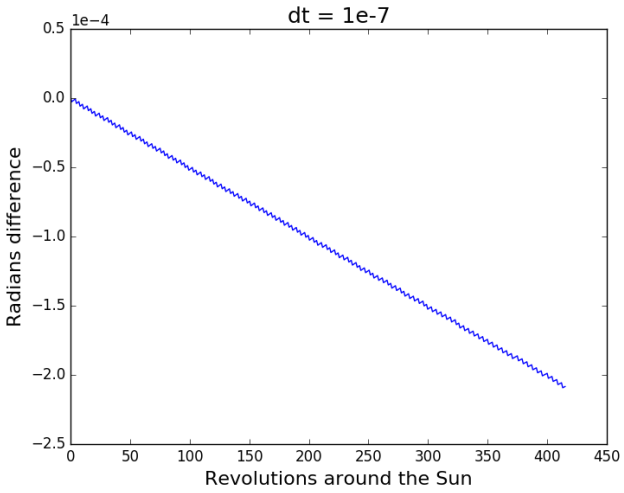


Figure 8: The difference of perihelion angle for the classical and the general relativistic case over a century with time step $= 1 \cdot 10^{-7}$. The x-axis gives revolutions around the Sun and the y-axis shows the difference in radians.

As we can see from figure 8 the difference is about $-2.0 \cdot 10^{-4}$ after Mercury has revolved about 415 times around the Sun.

Discussion

As we can see there are several differences between forward Euler and Velocity Verlet. Regarding FLOPs we can see that Euler has less than half the amount of FLOPs compared to Velocity Verlet. Here it is important to note that the FLOPs we count are only the FLOPs from the method in itself. The program also has calculation of, for example, a_i adding many more FLOPs to the real calculation. However, we want to compare the two methods and have therefore only included the FLOPs from the methods.

The fact that Velocity Verlet has 11 FLOPs compared to Eulers 4 is again shown by the time usage of the two methods in table 1. Forward Euler uses significantly less time compared to Velocity Verlet, but the effectiveness of the algorithm comes at a price. As we can see throughout the results, Velocity Verlet is the far better alternative when it comes to accuracy. This is evident from the algorithm as Forward Euler bases its position on a Taylor expansion stopping after the first derivative, Velocity Verlet takes the second derivative into account. The error is therefore dependent on $\mathcal{O}(h^2)$ for forward Euler and $\mathcal{O}(h^3)$ for Velocity Verlet.

If one studies the code, you will see that the Velocity Verlet could have been efficient. As the program is now, we calculate a_i and a_{i+1} for each loop, but a more efficient code would reuse the term a_{i+1} the next time it goes through the loop. If we were to do this project again, we would look into how to implement this in an efficient way.

The Earth-Sun system

Figures 1 and 2 illustrates the difference in accuracy for the two different methods. As we can clearly see, Velocity Verlet is the most accurate. While forward Euler needs $dt = 1.0e-4$ to start being stable, Velocity Verlet is nearly stable at $dt = 1.0e-2$ and stable at $dt = 1.0e-3$. This coincides perfectly with our discussion above. It is worth pointing out that the small discrepancy at $dt = 1.0e-2$ for Velocity Verlet is due to the error in the implementation, discussed earlier, where the initial positions are not always saved to file.

Finding the escape velocity through trial and error is hard. As we can see from figure 3a it seems as though the velocity given here could be an escape velocity. However, this is not the case. The simulations in figure 3 runs for two years, and then stops. If we had run the simulation for a longer time period, we would see that all velocities below 8.89 AU/year would eventually come back and all equal or above would not. Our numerical results seems to fit the analytical result, but running the simulations for longer time period would have been ideal. For velocities close to the analytical escape velocity, the time required to see if the object escapes gets quite high, as the orbit could potentially be a very elliptical orbit.

The Earth-Sun-Jupiter system

Comparing figure 2, 4 and 5 we can see clearly that adding Jupiter almost doesn't change Earth's motion at all.
The difference between figure 4 and 5 is that the former has the Sun in origo and the latter has center-of-mass in origo. As we can see they are in almost all manners, completely alike. This is because the Sun is so massive that it will stay close to the center-of-mass even though we don't stick it to origo. By zooming in on the plots in figure 5 we saw that the Sun did move around origo, but the movement was so small that it would have moved within its own radius.
It is more precise to let the Sun move because of the gravitational forces from Earth and Jupiter, but it is practically the same as letting it stay in origo. Therefore, it depends on what you want to use the program for. If your goal is to study the Sun's movement around the center-of-mass or you need the simulation as precise as possible, let the Sun move around. If not, you could just keep it still and maybe find a way to not have to calculate the force effecting the Sun and save time.

Changing the mass of Jupiter gave us some interesting results. As one can see from figure 6 the Earth's trajectory does not really change much for a Jupiter that is up to 100 times heavier, but increasing the mass further to 1000 times normal, gives us the result in 6c. Here we can see that the Earth starts out with a relative normal trajectory. But it very soon comes so close to the Sun that it recieves a gravity assist and is, after a little piroutte, shot out of the Solar System. By zooming in close we could see that the Earth would actually crash into the Sun if we had defined the Sun as a 3D object with a radius. To really get a good understanding of Jupiter's movement and it's effect on Earth we made a `.gif` for the 1000 times mass case. This file can be found in `/figures/esj.gif` in our GitHub repository [2]. As we can see from this animation the Earth goes back and forth between Jupiter and the Sun for a little while and goes around the Sun every time Jupiter gets a bit far away. The third time the Earth goes around the Sun it gets shot out of the system.

The whole Solar System

The whole Solar System, including Pluto as we all love Pluto, is shown in figure 7. We had to divide the System into its inner and outer part as the distances traveled by the outer planets are very large compared to those of the inner ones. These simulations were very easily made as our code was already object oriented. When we knew that our smaller system worked, we just added more planets by calling the right function and giving them initial position, velocity and mass. This shows one of the great strengths of object oriented programming as we could have added all the celestial bodies we wanted as long as we knew position, velocity and mass by only adding three more lines of code. And as we can see from 7 the calculations still seem valid even though we have added a lot of planets.

Perihelion Precession

From figure 8 it becomes clear to us that the difference is about $-2.0 \cdot 10^{-4}$ radians after a century. This coincides almost perfectly with the expected value of 43 arcseconds which is the same as $2.1 \cdot 10^{-4}$ radians. (We can neglect the minus sign since we are only interested in the difference.) As we have discussed in the theory, this effect is explained well by general relativity and our results coincides with the analytical value as well. Our plot in figure 8 has a little noise and could have been better with a lower dt-value. However, this would have taken a long time to run and since we have a pretty good result with this time resolution, we decided this was good enough. If we were to publish or use the result for something which required high accuracy, we would run the program again with lower dt.

It has come to our attention that the original mass of Mercury was wrong and we used $2.4 \cdot 10^{23}$ kg when we should have used $3.3 \cdot 10^{23}$kg. The difference caused by this is most likely small, as we scale the mass with the solar mass. As our results seems to replicate the expected result very well, and the execution time for this calculation is fairly long, rerunning the program with the correct mass is not a main priority at this point.

Conclusion

In the end this has been a fun project. We found that Velocity Verlet was the more accurate method as it was stable at time steps as large as $dt = 10^{-2}$ while forward Euler needs $dt = 10^{-4}$ to be stable, but forward Euler requires less time to calculate. We found that the initial velocity of the Earth should be $v = 2\pi$ and that the escape velocity has a threshold at $v = 2\sqrt{2}\pi$. By adding Jupiter we almost didn't change the Earth's motion but by increasing the mass of Jupiter to 1000 times more changed the Earth's motions so much it shot out of the system. Adding more planets to our system was easy since we had an object oriented code and the system was stable for the entire solar system over a time span of 500 years. The difference in perihelion angle between the classic case and the general relativity one was almost exactly 43 arcseconds and thus affirming general relativity as this coincides with the observed difference from the classical case.

References

[1] Euler method. `https://en.wikipedia.org/wiki/Euler_method`.

[2] Github repository. `https://github.com/scuper42/fys4150-project-3`.

[3] Nasa horizons. http://ssd.jpl.nasa.gov/horizons.cgi.

[4] Newton's laws of motion. https://en.wikipedia.org/wiki/Newton%27s_laws_of_motion.

[5] Perihelion precession. https://en.wikipedia.org/wiki/Apsidal_precession.