University of Oslo

FYS3150

Computational Physics

Project 3 - Building a model for the solar system using ordinary differential equations

Author:
Vebjørn Gilberg

Author:
Trond-Wiggo Johansen

October 2016

Contents

Abstract

In numerical mathematics, the study of how well integration methods perform given a particular section of tasks is widespread. This is of great interest in physics as well, where we deal with Hamiltonian systems. The solar system is an example of such a system where both energy and angular momentum is conserved which are quantities we test in this project. We simulate different parts of the solar system using both Euler's method and the Velocity Verlet method. The first system we simulate is a two-body system with the Sun and the Earth, moving on to a three-body system including Jupiter. Furthermore, we solve the many-body problem for the whole solar system, including Pluto for historical reasons. The perihelion precession of Mercury is a special case in comparison to the other planets and is described by General Relativity, which is also included.

Github repository:
`https://github.com/wiggoen/UiO/tree/master/FYS3150/Project_3`

1 Introduction

One can simulate the solar system by solving coupled ordinary differential equations (ODE) with the Velocity Verlet method and Euler's method. We're particularly interested in these types of equations in solar system dynamics due to the nature of Newton's law of gravitation which is what we use in our simulation. First we begin with the hypothetical solar system consisting of the Earth orbiting around the Sun, moving on to the three-body problem with Jupiter and then the many-body problem with all of the planets including the dwarf planet Pluto. Furthermore, we compare Euler's method and the Velocity Verlet method as far as runtime, stability and floating point operations (FLOPS) are concerned. One of the critical elements for the two methods is whether they conserve energy and angular momentum or not, so we test the two algorithms to see if the quantities are in fact conserved. In 1915, Albert Einstein published his article on the motion of Mercury around the Sun where he said that General Relativity can explain its perihelion motion. We take this into account when we extend the program to include Mercury with an extra term in the force between the two bodies.

2 Theory

2.1 The N-body problem

2.1.1 ODE's and the two-body problem

Although some of the methods in Calculus can be difficult to grasp, the fundamental and quite simple idea is to explain change and the rate of change. This field of mathematics was developed as far back as 1687[1] by Isaac Newton[1]. To make a simulation of the solar system we have to solve ordinary differential equations on the form

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

Newtons second law of motion is what we use to describe the law of gravitation, that is

$$m\frac{d^2x}{dt^2} = -kx$$

where k is a constant of force. Newtons law of gravitation is written as

$$F_G = \frac{GM_\odot M_{Earth}}{r^2} \qquad\qquad G = 6.674 \cdot 10^{-11}\frac{\mathrm{m}^3}{\mathrm{kg}\cdot\mathrm{s}^2}$$

where M_{Earth} and $M_\odot$ is the mass of the Earth and the Sun, respectively, whereas G is the gravitational constant and r is the distance between the two masses at hand. As the law stated above implies, there is a force between both of the masses and so it is not only the Earth that moves, but they both move about a common center of mass. We omit the motion of the Sun because of the vast difference in mass between the two bodies. The Sun is a magnitude of a millon heavier than the Earth. If we use Newtons second law of motion in all three spacial directions we get

$$\frac{d^2x}{dt^2} = \frac{F_{G,x}}{M_{Earth}} \qquad \wedge \qquad \frac{d^2y}{dt^2} = \frac{F_{G,y}}{M_{Earth}} \qquad \wedge \qquad \frac{d^2z}{dt^2} = \frac{F_{G,z}}{M_{Earth}}$$

where $F_{G,x}, F_{G,y}, F_{G,z}$ is the force in the x, y and z direction, respectively. This gives us three coupled differential equations that we have to solve, namely

$$\frac{dx}{dt} = v_x \qquad\qquad \wedge \qquad\qquad \frac{dv_x}{dt} = -\frac{GM_\odot}{r^3}x$$

[1] As well as Leibniz in 1693 which is the formal notation we use today.

which is identical to the equations for the y and z directions. From elementary mechanics, we know that a body in circular[2] motion follows the relation where the acceleration is described as the square of the velocity times the inverse of the distance from the center to the object in orbit. Mathematically, this means that

$$F_G = M_{Earth}\frac{v^2}{r} = \frac{GM_\odot M_{Earth}}{r^2}$$

To make calculations easier and better fit the solar system, we scale the mass with respect to the mass of the Sun, use astronomical units (AU) for length and years as our unit of time. This gives us $M_\odot = 1.0$, 1 AU $= 1.5 \cdot 10^{11}$ m and 1 yr $= 31557600$ s. The velocity of earth is described as $v = 2\pi$ AU/yr. From this, we can see that

$$GM_\odot = v^2 r = 4\pi^2 \ \frac{\text{AU}^3}{\text{yr}^2}$$

where $r = 1$ AU, and so the coupled equations for the different velocities becomes

$$\frac{dx}{dt} = v_x \qquad \wedge \qquad \frac{dv_x}{dt} = -4\pi^2\frac{x}{r^3} \tag{1}$$

which also applies to y and z.

2.1.2 Three-body problem

The three-body problem is a special case of the N-body problem and dates back to the 1680s [2]. Around 1887 the mathematicians Heinrich Bruns and Henri Poincaré showed that there is no general analytical solution for the three-body problem given by algebraic expressions and integrals [3]. However there exists 16 families of special solutions to the three-body problem and they can be viewed online in the three-body gallery[3].

In the last section we talked about the Earth-Sun system, but now we must add another planet, namely Jupiter. This means that for three celestial bodies we have 18 coupled equations due to the three spacial dimensions. If we add Jupiter we will have to think about both the distance from Jupiter to Earth and the Sun. This is accomplished by considering the vectors between the planets and the Sun. Let the distance from Earth to Jupiter be denoted $r_{EJ} = \sqrt{(x_E - x_J)^2 + (y_E - y_J)^2 - (z_E - z_J)^2}$, then the distance from Earth to Jupiter would be $x_E - x_J$, hence

$$F_x^{EJ} = -\frac{GM_J M_E}{r_{EJ}^3}(x_E - x_J)$$

[2]In this case near circular.

[3]Three-body gallery: http://suki.ipb.ac.rs/3body/

and similarly for the the other coordinates. The coupled equations for the acceleration in the x-direction is then

$$\frac{dv_x^E}{dt} = -\frac{4\pi^2 x_E}{r^3} - 4\pi^2 \left(\frac{M_J}{M_\odot}\right) \frac{x_E - x_J}{r_{EJ}^3} \qquad \wedge \qquad \frac{dx_E}{dt} = v_x^E \tag{2}$$

where we have multiplied the second term in the equation for the acceleration with $M_\odot/M_\odot$ to obtain the factor $4\pi^2(\text{AU})^3/\text{yr}^2$. We would get similar equations for Jupiter which would typically look like

$$\frac{dv_x^J}{dt} = -\frac{4\pi^2 x_J}{r^3} - 4\pi^2 \left(\frac{M_E}{M_\odot}\right) \frac{x_J - x_E}{r_{JE}^3} \qquad \wedge \qquad \frac{dx_J}{dt} = v_x^J \tag{3}$$

This means that we get 18 coupled equations when we include the y and z direction.

2.1.3 N-body problem

Consider a group of N point masses m_i where $i = 1, 2, \ldots, N$ in an inertial frame of reference in three dimensional space $\mathbb{R}^3$ where all of the masses influence each other gravitationally. If we let each mass have a position vector $\boldsymbol{r}_i$, Newton's second law of motion yields $m_i d^2\boldsymbol{r}_i/dt^2$ which is the sum of the forces at play. Let F_{ij} be the gravitational force between two masses m_i and m_j, that is

$$\boldsymbol{F}_{ij} = \frac{Gm_i m_j(\boldsymbol{r}_i - \boldsymbol{r}_j)}{||\boldsymbol{r}_i - \boldsymbol{r}_j||^3}.$$

This, together with the second law of motion yields

$$m_i \frac{d^2 r_i}{dt^2} = \sum_{i=1, j\neq i}^{N} \frac{Gm_i m_j(\boldsymbol{r}_i - \boldsymbol{r}_j)}{||\boldsymbol{r}_i - \boldsymbol{r}_j||^3} = \frac{\partial U}{\partial \boldsymbol{r}_i}$$

where

$$U = \sum_{i<j} \frac{Gm_i m_j}{||\boldsymbol{r}_i - \boldsymbol{r}_j||}$$

Hamilton's equations tells us that the translational symmetry of this problem results in the constant velocity of the center of mass movement. Furthermore, the angular momentum is constant due to the rotational symmetry. Hence,

$$\boldsymbol{C} = \sum_{i=1}^{N} \boldsymbol{r}_i \times \boldsymbol{p}_i$$

This problem becomes extremely chaotic, but we can represent the solution to both the 3-body and the N-body problem through convergent power series expansions. While these do exists, any practical use of them are quite worthless due to the fact that one needs millions of terms to explain the system for short amounts of time. For this reason it is ***not*** wise to make use of the expansions, thus numerical integration is the best way to handle these kinds of problems. In our case we will end up with $N = 10$ celestial bodies which gives us 60 coupled equations to compute.

2.2 Escape velocity

A body can escape the gravitational pull of the mass of which it is orbiting at a certain tangential velocity characterized as the escape velocity. In order to find this velocity for the system consisting of some planet and the Sun we can consider a planet with mass m at a distance of $r = 1$ AU (like Earth). If we imagine a mass m which escapes a larger body of mass $M_\odot$, the potential and kinetic energy is zero at infinity and by using the conservation of energy we get

$$\begin{aligned} \frac{1}{2}mv_{escape}^2 - \frac{-GM_\odot m}{r} &= 0 + 0 \\ \frac{1}{2}mv_{escape}^2 &= \frac{GM_\odot m}{r} \\ v_{escape} &= \sqrt{\frac{2GM_\odot}{r}} \end{aligned}$$

Given the values of G, $M_\odot$ and r we get

$$\begin{aligned} v_{escape} &= \sqrt{\frac{2GM_\odot}{r}} \\ &= 2\sqrt{2}\pi \ \tfrac{\text{AU}}{\text{yr}} \\ &\approx 8.9 \ \tfrac{\text{AU}}{\text{yr}}. \end{aligned}$$

2.3 The perihelion precession of Mercury

When Isaac Newton published his work on Calculus and the motion of the planets, people began to realize that the motion of Mercury didn't fit his calculations and rightly so, because Einstein was needed to explain that sort of motion through

space-time with his General theory of Relativity (GR). When Einstein published his article, *Explanation of the Perihelion Motion of Mercury from General Relativity Theory*[4], he claimed that the general theory of relativity could explain the perihelion precession of about 43 arc seconds ($43''$) per century with a varying angle θ_p. That is,

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

where x_p and y_p denotes the position of Mercury at perihelion, respectively.

The characteristic difference between the motion of Mercury and the other planets is that it does not appear in exactly the same position it started in after one complete orbit around the Sun. This is a feature we must add to the standard gravitation law since any correction to the pure $1/r^2$ law will lead to such an orbit. This means that we must add a general relativistic term to the law of gravitation, namely

$$F_G = \frac{GM_\odot M_{Mercury}}{r^2}\left[1 + \frac{3l^2}{r^2c^2}\right] \tag{4}$$

where $l = |\vec{l} \times \vec{v}|$ is the magnitude of Mercury's orbital angular momentum per unit mass, and c is the speed of light in vacuum. One can explain the change in it's perihelion position by considering a far away observer who sees Mercury as it's at the perihelion. Because it's closer to the Sun, and thereby farther down in the gravitational field, an observer would see Mercury spending more time there than the other places in the orbit which moves the perihelion position. The effect is minuscule, but it's there.

3 Methods and algorithms

3.1 Euler's method

In any rudimentary course on Calculus one finds that the differential equations solved are quite basic. That's not to say that all of them are easy to solve, but they are *exactly solvable* by analytical methods. In reality, few differential equations can be solved this way, and this is not a new concept. Leonhard Euler devised a method christened Euler's method around 1770 to integrate numerically. The

method consists of discretizing the variable x to x_i and v to v_i where we find a relation between the two through the Taylor expansion[4] of x around $t_i + h$, that is

$$x_{i+1} = x(t_i + h) = x_i + h \cdot \underbrace{x_i'}_{v_i} + \frac{h^2}{2} \underbrace{x_i''}_{a_i} + \mathcal{O}(h^3)$$
$$v_{i+1} = v_i + h \underbrace{v_i'}_{a_i} + \frac{h^2}{2} v_i'' + \mathcal{O}(h^3)$$

where the step length h is defined as $h = \frac{t_{final} - t_0}{n}$ and t_i as $t_i = t_0 + i \cdot h$. Furthermore, we must use the initial conditions $v(t = 0) = v_0$ and $x(t = 0) = x_0$. It is common to know the acceleration a_i, which is obtained from Newton's law of gravitation in our case. Ignoring the higher order terms, as well as the quadratic term gives rise to Euler's method. This is because this method is based on a two-point formula for the first derivative, i.e.

$$x_i' \approx \frac{x_{i+1} - x_i}{h} \qquad \wedge \qquad v_i' \approx a_i \approx \frac{v_{i+1} - v_i}{h}$$

which yields

$$x_{i+1} = x_i + hv_i + \mathcal{O}(h^2)$$
$$v_{i+1} = v_i + ha_i + \mathcal{O}(h^2)$$

If we use the equations from the theory section in the x-direction, we can see that the acceleration given above is discretized as

$$a_{x,i} = -4\pi^2 \frac{x_i}{r_i^3}$$

So the final equations to solve becomes

$$v_{x,i+1} = v_{x,i} - h\frac{4\pi^2}{r_i^3} x_i + \mathcal{O}(h^2)$$
$$x_{i+1} = x_i + hv_{x,i} + \mathcal{O}(h^2),$$

which also applies to the other two spacial coordinates. The code will ultimately represent the same, but it looks a bit different. The implementation of Euler's method in C++ looks like this:

```
for(CelestialBody &body : system.bodies()) {
    body.position += body.velocity*m_dt;
    body.velocity += body.force/body.mass * m_dt;
}
```

[4]One can also derive this from a geometrical description.

3.2 The Velocity Verlet method

When it comes to integrating the equations of motion, Verlet integration is well suited for problems in Molecular Dynamics and simulations of the solar system[5]. There are three versions of Verlet integration, namely the Position, Leapfrog and the Velocity. We will consider the Velocity method in this project because it gives both the position and the velocity at the same time step, which also makes it the most complete form of the Verlet methods. It all boils down to manipulations of the Taylor expansion of $v_i' = a_i$, that is

$$x_{i+1} = x(t_i + h) = x_i + h \cdot \underbrace{x_i'}_{v_i} + \frac{h^2}{2} \underbrace{x_i''}_{a_i} + \mathcal{O}(h^3)$$

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

and the corresponding expansion of the derivative of v_{i+1}

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

Because our error goes as $\mathcal{O}(h^3)$, we only have to retain the terms up to the second derivative. That is,

$$hv_i'' \approx v_{i+1}' - v_i'$$

Combining these equations yields

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

and so the position becomes

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

where $\mathcal{O}(h^3)$ is the local truncation error at each step. It is important to note that a_{i+1} depends on x_{i+1} because a_i and v_{i+1} is used in x_{i+1}. This again, can be reused in v_{i+1}. The key thing about this method is it conserves the energy of the system,

[5] Hopefully.

i.e. the system's Hamiltonian remains invariant. Another convenient fact about this Verlet algorithm is that it's very stable numerically and there is a compromise between FLOPS and numerical precision. In both of these algorithms we need to know the acceleration of the system, and we do! As mentioned earlier, the discretized accelerations is

$$a_i = -4\pi^2 \frac{x_i}{r_i^3}$$

and likewise for the y and z-direction. When we know the acceleration we can calculate the position and velocity of the object(s) which is what this and Euler's method does for us. We can thus plug the acceleration into the equations above which yields

$$v_{x,i+1} = v_{x,i} - 2\pi^2 h \left(\frac{x_{i+1}}{r_{i+1}^3} + \frac{x_i}{r_i^3} \right) + \mathcal{O}(h^3)$$
$$x_{i+1} = x_i + h v_i - 2\pi^2 h^2 \frac{x_i}{r_i^3} + \mathcal{O}(h^3)$$

As stated in the section for Euler's method, the code in the program will represent the same thing, but will look a bit different. The implementation of the Velocity Verlet method in C++ looks like this:

```
    for(CelestialBody &body : system.bodies()) {
        body.velocity += (body.force / body.mass) *m_dt/2.0;
        body.position += body.velocity*m_dt;
    }

    // F(t + dt)
    for(CelestialBody &body : system.bodies()) {
        body.velocity += (body.force / body.mass) *m_dt/2.0;
    }
```

4 Code Validation

As stated earlier, one of the most important aspects of the integration methods we use is how well they conserve energy and angular momentum. If not, our simulations will be way off. In our program we calculate the potential and kinetic energy, and to test whether the functions do their job we can calculate the total energy analytically by $E_{tot} = \frac{1}{2}mv^2 - \frac{GM_\odot m}{r}$ and the angular momentum by $l = |\vec{r} \times \vec{v}|$ and compare them to the values calculated in the program. Since we can't test the angular momentum directly we use the length of the angular momentum. An efficient and easy way to do this is by using Catch, hence

```
TEST_CASE( "Conservation of angular momentum and total energy", "
    Approximation" )
{
    // Setting up system
    int numTimesteps = 1000;
    double dt = 0.001;
    string integrator = "Verlet";
    int printEvery = 1;
    bool withGr = false;

    // Initializing SolarSystem
    SolarSystem solarSystem;

    // Setting numTimesteps, dt and integrator
    solarSystem.setNumTimesteps(numTimesteps);
    solarSystem.setDt(dt);
    solarSystem.setIntegrator(integrator);

    // Adding celestial bodies to system
    // Sun
    solarSystem.createCelestialBody( vec3(0,0,0), vec3(0,0,0), 1.0, "
        Sun");

    // Earth
    solarSystem.createCelestialBody( vec3(1.0, 0.0, 0.0), vec3(0.0, 2*
        M_PI, 0), 3.0e-6, "Earth");


    // Integrate system
    solarSystem.integrate(printEvery, withGr);

    // Computed total energy and length of angular momentum
    double totalEnergy = solarSystem.totalEnergy();
    double LengthOfAngularMomentum = solarSystem.angularMomentum().
        length();

    // Analytical values for the kinetic energy, potential energy and
        angular momentum of the Sun-Earth system
    // Kinetic energy: K_E = 0.5*m*v**2 = 6*pi^2 * 10^(-6) = 5.922e-5
    double A_kineticEnergy = 0.5*3e-6*4*M_PI*M_PI;
    // Potential energy: P_E = -(G*M*m)/r = -12*pi^2 * 10^(-6) = -1.184
        e-4
    double A_potentialEnergy = -4*M_PI*M_PI*1.0*3e-6/1.0;
    // Total energy: Kinetic energy + Potential energy
    double A_totalEnergy = A_kineticEnergy + A_potentialEnergy;
    // Length of angular momentum: |l| = ||r x v|| = 2*pi
    double A_LengthOfAngularMomentum = 2*M_PI;
```

```
    // Conservation of energy
    REQUIRE(totalEnergy == Approx(A_totalEnergy));
    // Conservation of angular momentum
    REQUIRE(LengthOfAngularMomentum == Approx(A_LengthOfAngularMomentum
        ));
}
```

All tests pass, which means that our solar system works as it should.

5 Visualization

5.1 Ovito

To view our setup go to the Appendix section 8.3 'Using Ovito'.

Using Ovito[6] as a visualization tool is great because it's simple to use, have a lot of features and gives you animation quickly. It's made for visualizing and analyzing atomistic data, but it works great for simulating the solar system as well. One of the features is to create trajectory lines for the solar system. For Pluto to make one orbit around the Sun we have to use about 250000 time steps, but with that amount of time steps Mercury is making a wobbly trajectory and it doesn't look good. A snapshot of the animation of the solar system without trajectory lines is shown in figure 1.

If you only plot every five hundred time step, it will look like Mercury is orbiting slowly around the Sun while the other planets orbits very fast, but this is because the sampling rate does not follow the Nyquist-Shannon[7] sampling theorem. This means that the samples (every five hundred time step) does not capture all the information about the computed orbits. Since we loose information about our orbits, we get an effect of Mercury orbiting slowly around the Sun.

[6]OVITO is a scientific visualization and analysis software for atomistic simulation data developed by Alexander Stukowski at Darmstadt University of Technology, Germany. It can be found at: http://www.ovito.org

[7]The Nyquist-Shannon sampling theorem honors Harry Nyquist and Claude Shannon. The theorem was also discovered independently by other scientists like E. T. Whittaker and Vladimir Kotelnikov.

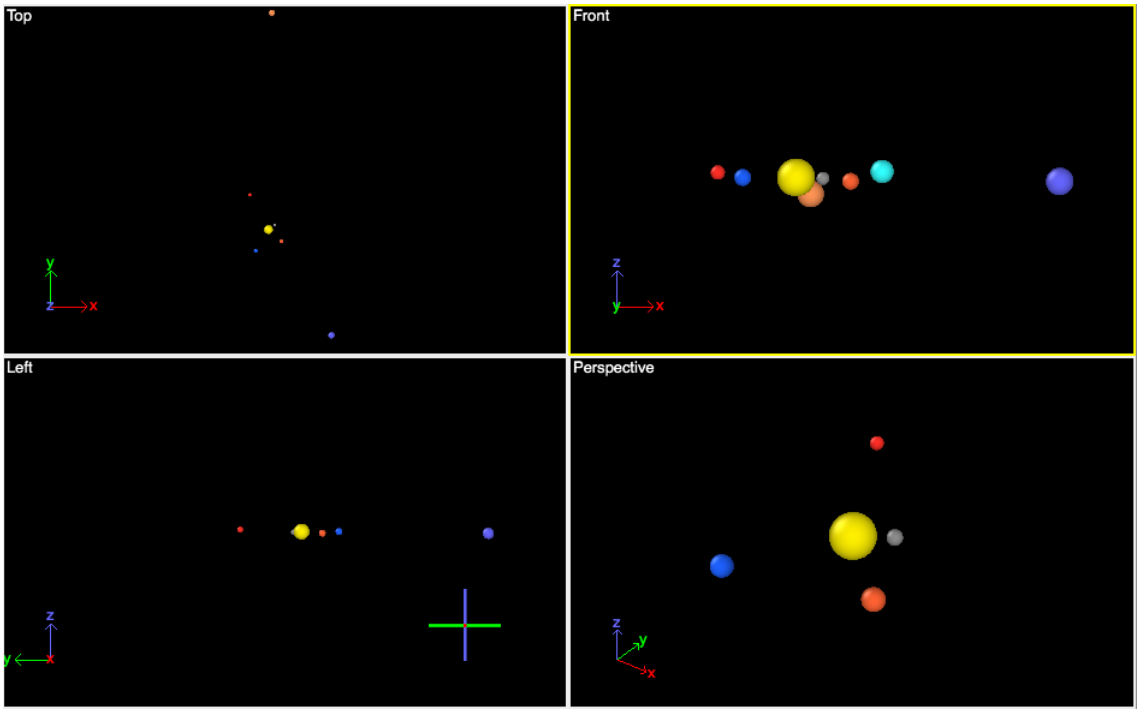

Figure 1: Animating the solar system in three dimensions using Ovito. The four blocks shows the solar system from different angles. In the upper right block you can see Mars (red), Earth (blue), the Sun (yellow), Saturn (light orange), Mercury (gray), Venus (orange), Uranus (cyan) and Jupiter (violet). Pluto is not visible in this picture, and it's far away from the Sun anyways.

5.2 Python

Using Python as visualization tool we can make plots in 2D and 3D. Trying to simulate the whole solar system in a 3D plot is quite difficult since the distance from the Sun to Pluto is enormous. When trying to plot the whole system we only get a view of the orbits from Mars to Pluto. This is shown in figure 2.

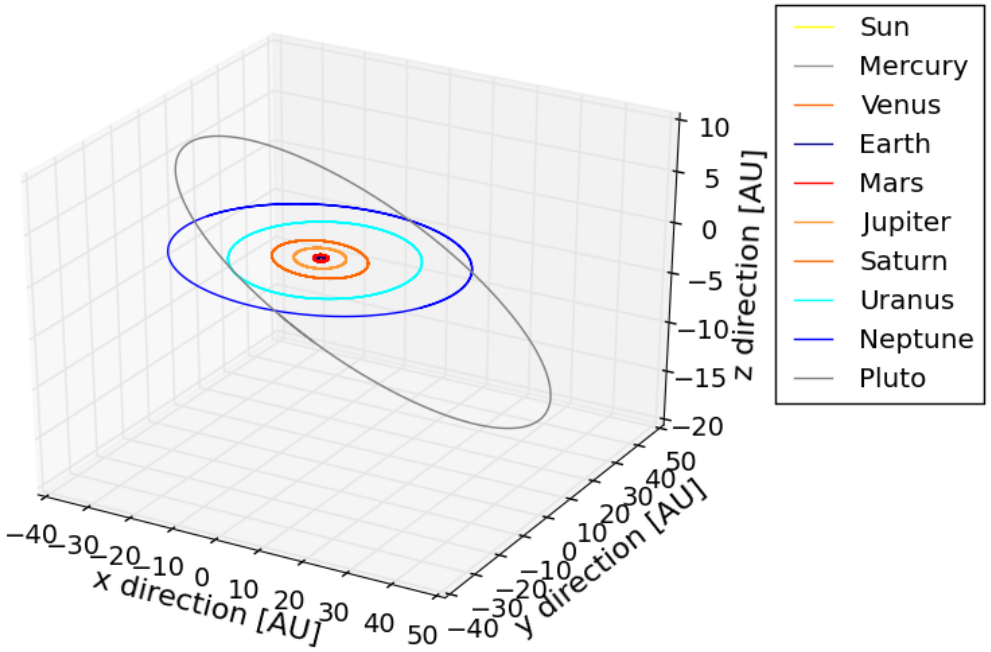

Figure 2: Simulation of the solar system in three dimensions using Python with 250000 time steps. It's quite difficult to see the orbits of the inner celestial bodies, from Earth to the Sun. The inner red circle defines Mars' orbit around the Sun.

6 Results and discussion

6.1 The two-body problem

The most famous family of integrators is the Runge-Kutta family and the one we use is Euler's method. In general, to see if something is good or bad we need something to compare it to. In our case that something is the Velocity Verlet method and the system we test them on is the solar system. Euler's method can reproduce the orbit of the Earth around the Sun to some extent. We can see in figure 3 that it does not perform satisfactory for 10000 time steps and different dt's. This is due to the local error in Euler's method which runs as $\mathcal{O}(h^2)$. This is not the case for the Verlet method, since it has an error of order $\mathcal{O}(h^3)$, so the Euler algorithm becomes unsatisfactory quicker than Verlet. This is illustrated in figure 3 and 4.

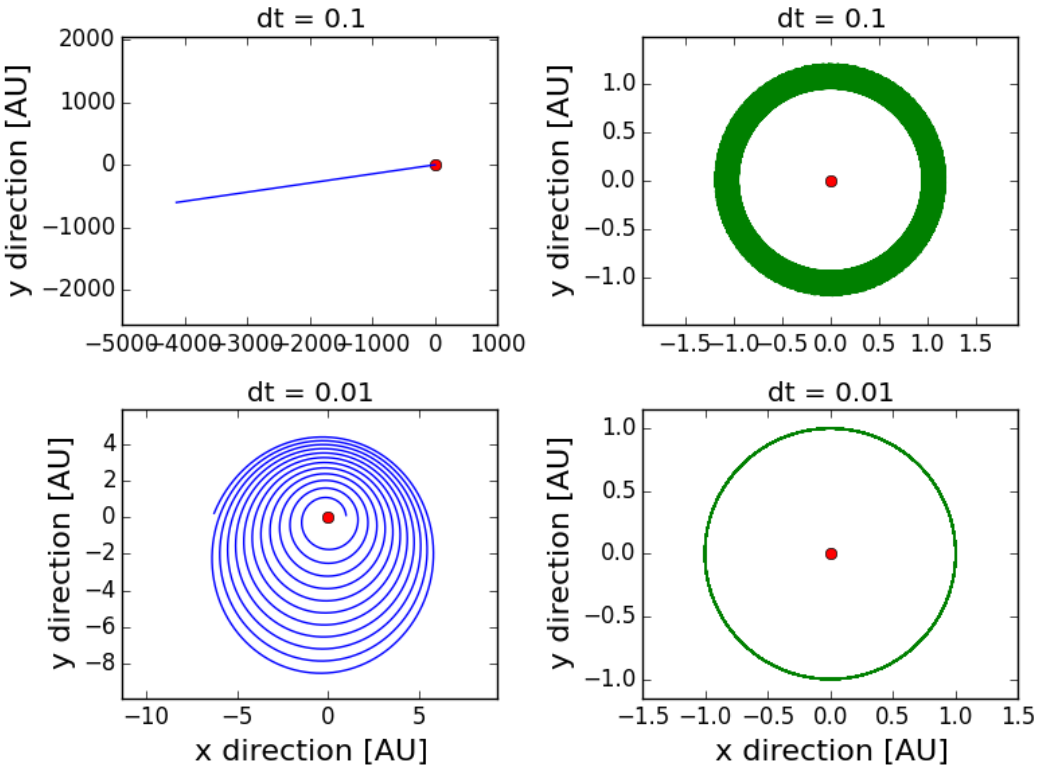

Figure 3: The solar system consisting of the Earth and the Sun. Euler's method is used in the blue plots, and Verlet in the green ones for 10000 time steps and different dt's.

Euler's method does not do the job for a time step of either $dt = 0.01$ or $dt = 0.1$ because the error becomes quite large very quickly. This is why other integration methods are preferred over Euler's method. The Verlet method does a much better job at $dt = 0.01$, however it's faulty at $dt = 0.1$ which is understandable because dt is too big.

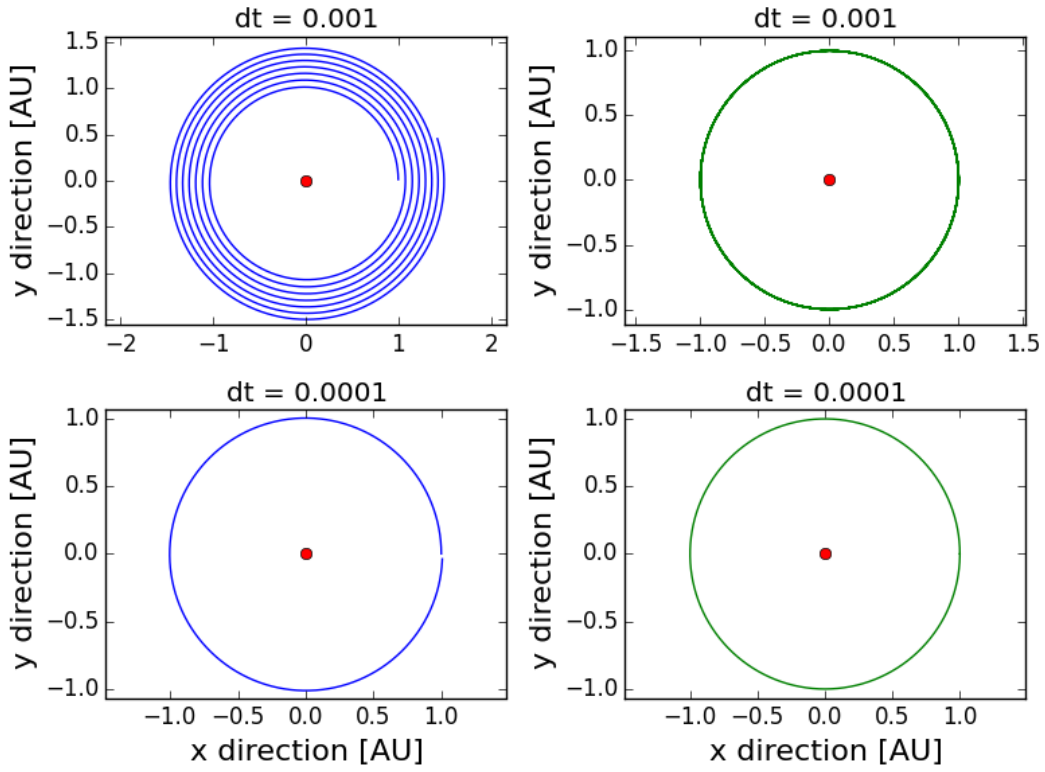

Figure 4: The solar system consisting of the Earth and the Sun. Euler's method is used in the blue plots, and Verlet in the green ones for 10000 time steps and different dt's.

As we discussed in figure 3, Euler's method does not produce a satisfactory orbit, but it do perform a bit better for $dt = 0.0001$. While Verlet also obviously improves for a smaller time step, it's not hard to see which of the algorithms is preferred when we want to simulate a Hamiltonian system like this. The neat thing about the Verlet method is that it has a decent compromise between FLOPS and numerical precision. By that we mean that there's a tradeoff between the number of significant digits in the calculations of the position of the planets and the number of floating point operations when we do so, and this relationship is satisfactory. In table 1 below, we have represented the floating point operations of the two methods.

	Euler's method	Velocity Verlet method
Memory Reads	$7N$	$10N$
Memory Writes	$2N$	$3N$
Divisions	N	$4N$
Additions	$2N$	$3N$
Multiplications	$2N$	$3N$
Subtractions	0	0

Table 1: Floating point operations with with Euler's method and the Velocity Verlet method as seen in the listings in the section for methods and algorithms. We include the operations that are read from and written to the memory as well.

As stated several times earlier, one important motivation for choosing the right integrator for a Hamiltonian system is to choose the one that keeps the Hamiltonian invariant. While the Hamiltonian in Euler's method varies, it is constant in the Velocity Verlet method. This is due to the symplecticity of the algorithm.[8]

When a program has to run for weeks or even months, optimization of the code is extremely important. One can optimize programs like the ones we encounter in this paper by some simple optimization flags for the g++ compiler. In table 2 we have included the runtimes for both Euler's method and the Velocity Verlet method with and without compiler optimization flags.

N	Runtime for Euler's method without optimization flag [s]	Runtime for Euler's method with optimization flag '-O3' [s]	Runtime for the Verlet method without optimization flag [s]	Runtime for the Verlet method with optimization flag '-O3' [s]
100000	1.209053	0.728423	1.324063	0.812690
150000	1.824935	1.103554	1.968575	1.221159
200000	2.422072	1.469343	2.612698	1.634270
250000	2.990832	1.839864	3.239193	2.023163
300000	3.601328	2.235404	3.890702	2.409948
400000	4.789730	2.908766	5.172844	3.226968

Table 2: Table of runtimes in seconds for different time steps N. The values are taken as an average of 10 program runs. For large N we can see that the runtime increases which is completely natural. Note that there's an increase of about 1.08 times in the runtime for the Verlet method compared to the Euler method.

[8]We encourage the reader to have a closer look at symplectic integrators and Hamiltonian systems in the appendix.

The fact that the runtime is about 1.3 to 1.2 second slower for the Verlet method is the result of the compromise in flops as we would expect. The algorithm is a bit larger for the Verlet method but the runtime is not affected that much in comparison to the results it produces. In other words, we can live with an extra ~ 0.5 s due to the fact that it conserves energy well.

6.2 Escape Velocity

By trial and error, with 400000 time steps using the Verlet algorithm, we can test at what speed the planet escapes and if we do so we find that the escape velocity has to be $v_y \approx 8.9 \ \frac{\text{AU}}{\text{yr}}$ before it can escape the gravitational pull of the Sun.

To create celestial bodies in our solar system we have to give the initial values for the position, velocity, mass and give the body a name

```
solarSystem.createCelestialBody(vec3 position, vec3 velocity, double
    mass, std::string name)
```

Thus, we must use the escape velocity as we explained above when we plug in the values for the velocity. The initial position of the planet is simply 1 AU from the Sun. Moreover, the mass given is the Earth's mass scaled by the mass of the Sun.

```
solarSystem.createCelestialBody(vec3(1.0, 0.0, 0.0), vec3(0.0, 8.9,
    0.0), 3.0e-6, "Planet");
```

The orbit in figure 5 is most likely correct and our plot shows that the planet is thrown out to about 350 AU in the x direction and over 50 AU in the y direction from the Sun. For comparison, Pluto's orbit is at a radial distance of 39.53 AU from the Sun. Our planet were thrown about 1.2 times this in the y direction and almost 9 times this in the x direction. In radial distance this is about 353 AU, which is about 9 times further out than Pluto's orbit. We would say that this is far enough to escape the gravity of the Sun.

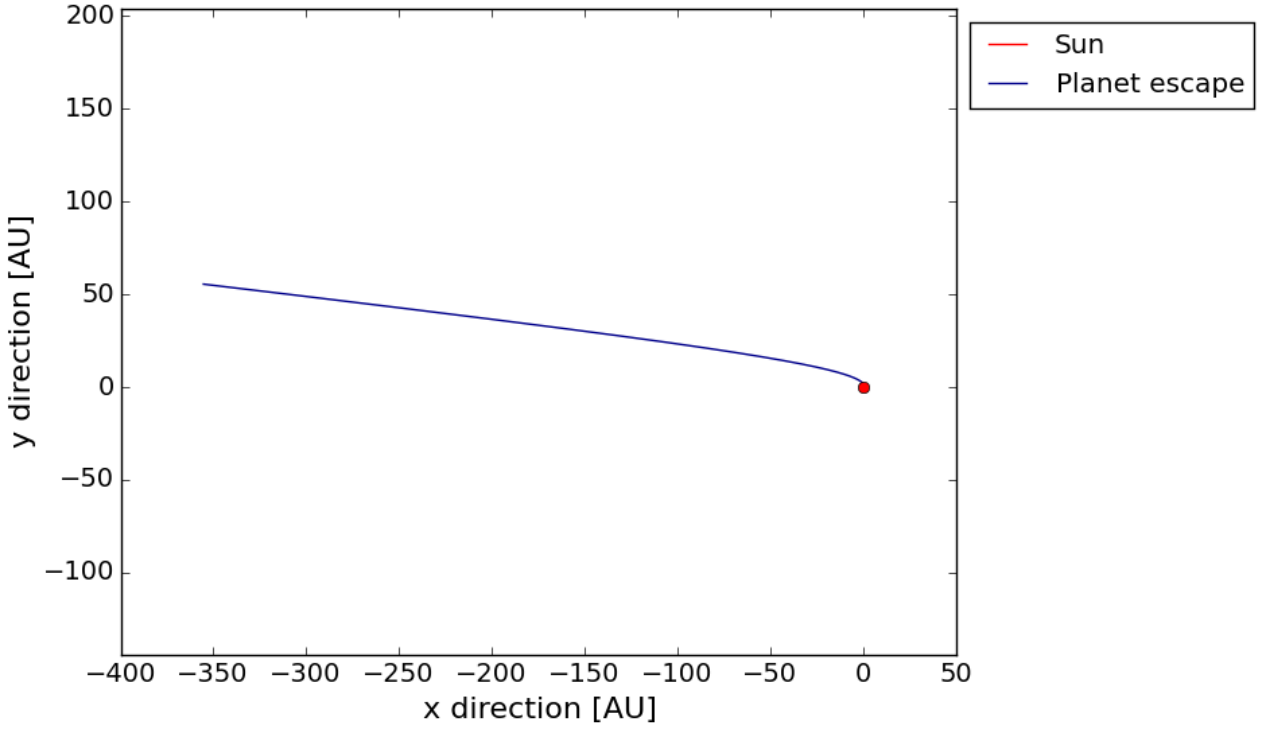

Figure 5: Simulation of a planet (like Earth) escaping the gravity of the Sun using 400000 time steps and $dt = 0.001$.

These results can be checked by finding the escape velocity analytically as we did in the theory section and they match well. We see that the mass of the planet doesn't matter, the escape velocity is only dependent on the mass of the Sun and the distance between the bodies. To get a better view of how fast this actually is, we can represent the velocity in a more intuitive unit, with $M_\odot = 2 \cdot 10^{30}$ kg, $G = 6.674 \cdot 10^{-11}$ $\frac{\text{m}^3}{\text{kg s}^2}$ and $r = 1.5 \cdot 10^{11}$ m, this means that the escape velocity is

$$\begin{aligned} v_{escape} &= 8.9 \cdot \frac{1.5 \cdot 10^{11} \text{ m}}{3600 \cdot 24 \cdot 365 \text{ s}} \\ &\approx 42.2 \ \frac{\text{km}}{\text{s}} \end{aligned}$$

6.3 The three-body problem

Although the Sun is the primary body in the solar system, as far as mass and thereby it's gravity is concerned, the other planets do have an effect on each other as well. The most massive planet in the solar system is Jupiter, and it's interesting to see how much of a gravitational effect on Earth's orbit Jupiter has. In figure 6 we have illustrated the shift in the orbit.

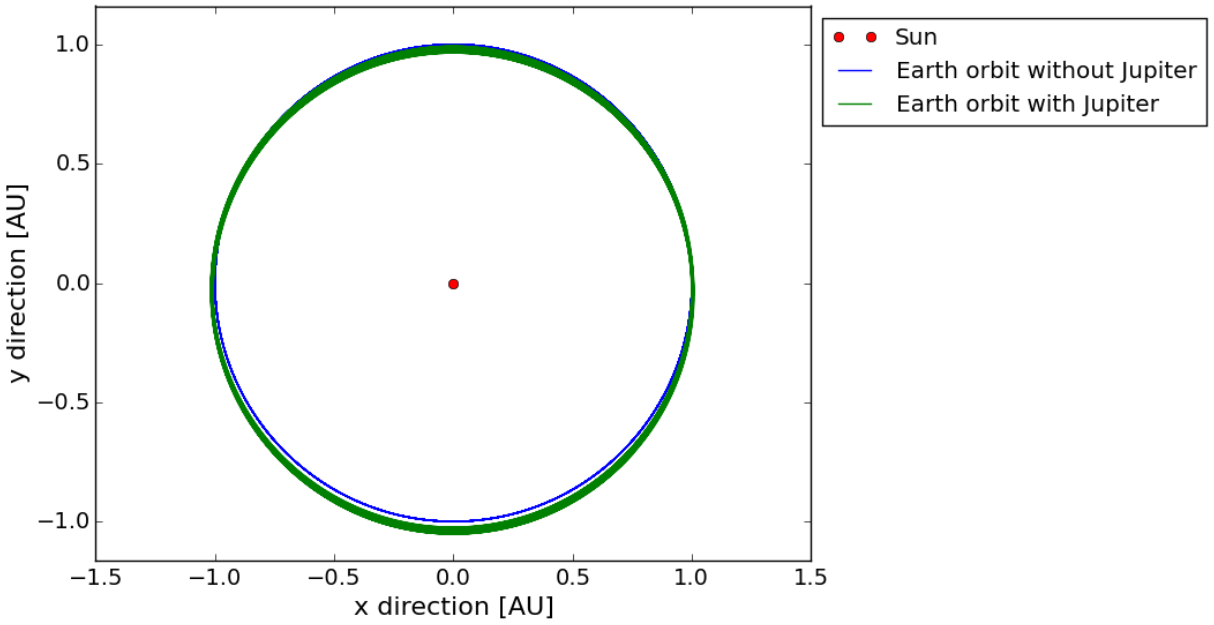

Figure 6: The shift in Earth's orbit is caused by the gravitational pull of the planet Jupiter. However, this effect is quite small due to the fact that the Sun is about 1000 times more massive than Jupiter. The simulation is run with 12000 time steps (enough for Jupiter to have one orbit around the Sun).

If we now increase the mass by a factor of 10 followed by 1000 we get some odd effects as shown in figure 7 and 8, respectively.

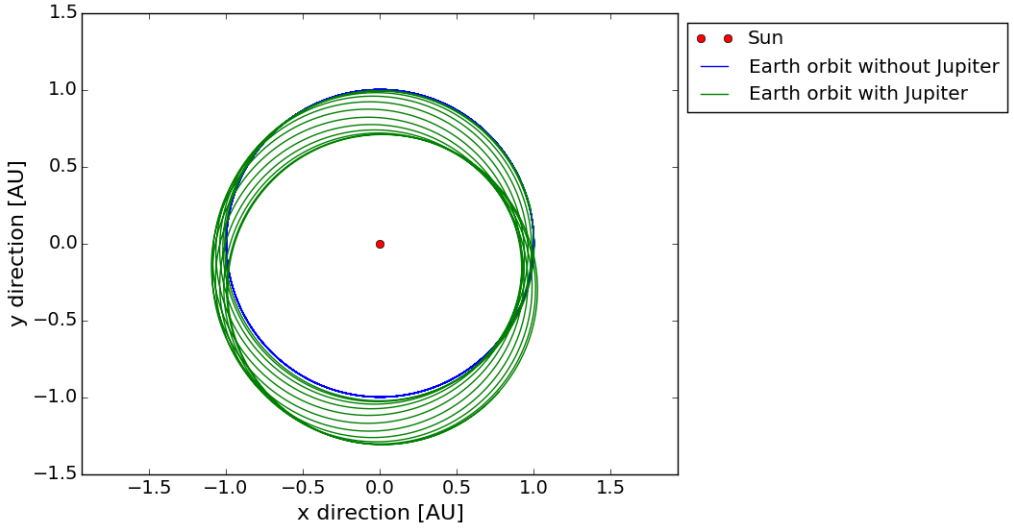

Figure 7: The Sun and the Earth with and without Jupiter. In figure 6 we used Jupiter's scaled mass, but in this figure that mass is multiplied by 10. The simulation is run with 12000 time steps.

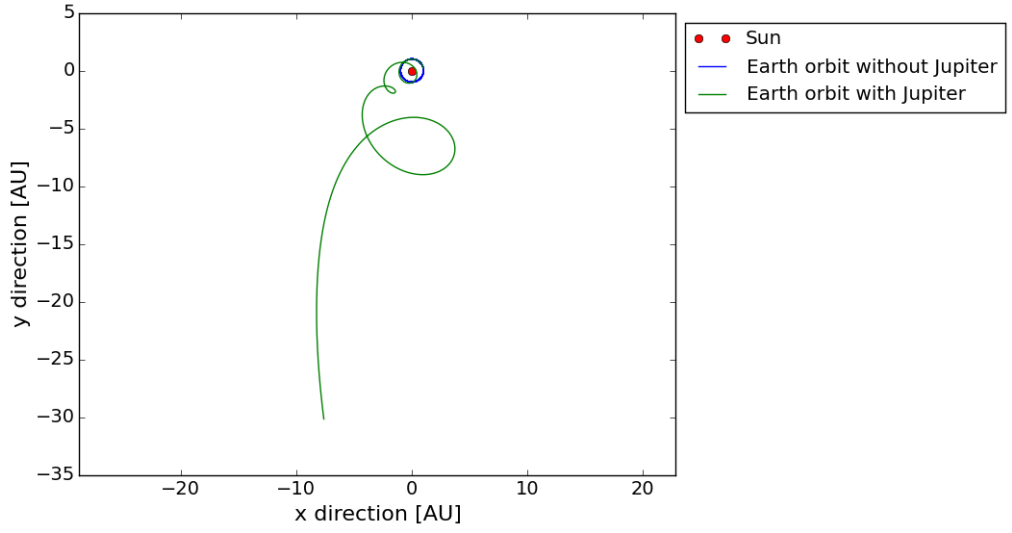

Figure 8: The Sun and the Earth with and without Jupiter. This time the mass of Jupiter is multiplied by 1000, which is almost equal to the mass of the Sun. The simulation is run with 12000 time steps.

As discussed in the section above, the stability of the Verlet method is quite satisfactory for $dt = 0.001$ which is what we have used in our simulations with and without

Jupiter. The orbit in figure 6 is quite stable because the mass of Jupiter is set to it's correct scaled value. In the figures 7 and 8 however, the mass of Jupiter scales up unproportionally with the other masses and so energy is no longer conserved. That is, if we scale up one mass, we have to do the same with the others. Although Verlet is still the same energy conserving method, the increase in mass of Jupiter leads to the fact that the energy is no longer conserved because we're changing Jupiter's momentum, and hence we end up with unstable solutions. Particularly in figure 8.

6.4 The full model of the solar system

We want to simulate the solar system as it is, and that implies that the total momentum must be zero. That is

$$\sum_i m_i v_i + p_\odot = 0$$

When we do this using data from NASA we find that the velocity of the Sun must be $\vec{v} \approx 365 \cdot (-1.8 \cdot 10^{-6}, 6.8 \cdot 10^{-6}, 3.2 \cdot 10^{-8})$AU/yr. In figure 2 in section 5.2 we have simulated the entire solar system. Since we cannot see too much of the inner planets, figure 1 in section 5.1 is added in order to illustrate these.

6.5 The perihelion precession of Mercury

We can test Einstein's claim[9] in our program using three points and their distances from the Sun. When Mercury is at it's perihelion position, it has the unique property where the distances between the Sun, the former and the next points are longer than the perihelion point. When Mercury is at this point, we make use of the tangent of the angle θ_p. The calculation should produce $43'' \approx 0.000208$ radians, over a period of 100 earth years, and in figure 9 we have plotted the angle as a function of time. We can see that there is a correspondence between our calculations and his explanation which we expected to see.

To make sure that the time resolution used to simulate the effect of the perihelion precession is sufficient, we check that the perihelion precession we get with a pure Newtonian force is at least a few orders of magnitude smaller than the observed perihelion precession of Mercury. Our program calculated this value to be $\sim 2 \cdot 10^{-2}$, which is indeed a few orders of magnitude smaller.

[9]This has been checked thoroughly through the 101 year period since his publication of course.

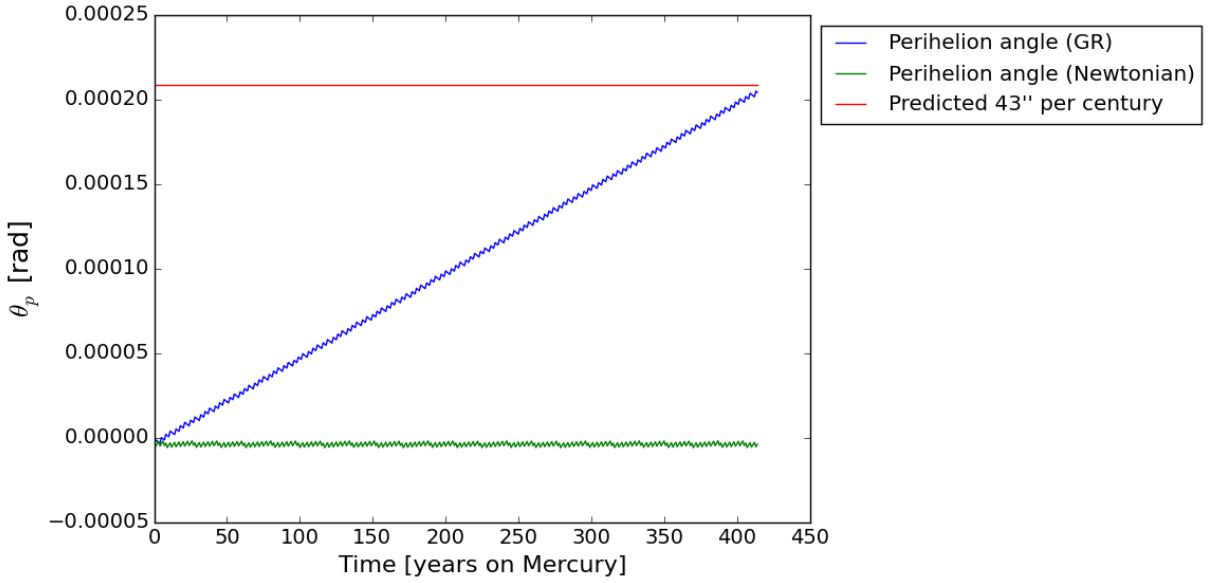

Figure 9: The perihelion angle of Mercury as a function of time. The simulation runs over 414 Mercury years which approximately corresponds to a century here on Earth. The simulation was run with 10^9 time steps and $dt = 10^{-7}$.

7 Conclusion

When we simulate anything in nature on a computer, we want the simulation to be as close to the real thing as possible. Often that simulation involves some kind of change, and in physics we often want to see if that change is conserved. Even though the positions of the planets do change, we want their orbits to be periodic and stable and one of the goals of this project was to check whether the Velocity Verlet method, together with Newton's laws can produce stable orbits over time. It performs considerably better than Euler's method and with almost no cost in runtime even though it requires more FLOPS. Since the conservation of energy and angular momentum is vital in order to simulate such a system, and Euler's method doesn't conserve energy, we continued to use the Verlet method. The perihelion precession of Mercury is one of the things such a model should be able to reproduce taken General Relativity into account. We calculated the perihelion angle over a period of 100 Earth-years which corresponds well with the calculated 43 arc seconds that Einstein explained with his theory.

8 Appendix

8.1 Symplectic integrators and Hamiltonian systems

Newton's laws of motion are viewed as fundamental laws in physics. There is however, a set of equations from which these laws can be derived. William Rowan Hamilton proved in 1843 that with a function we today call the *Hamiltonian*, one can derive Newton's equations given that F is conservative for a system of n particles. This equation takes the form

$$H(q,p) = \sum_{i=1}^{n} \frac{|p_i|^2}{2m_i} + V(q_1, q_2, \ldots, q_n)$$

where p_i is the the canonical momentum $p_i = m_i\dot{q}_i$. Newtons laws of motion can then be derived from the equations

$$\dot{q} = \frac{\partial H}{\partial p}, \qquad\qquad \dot{p} = -\frac{\partial H}{\partial q}$$

This set of equations is viewed as more fundamental than Newton's equations because we're using arguments of energy rather than force. Note that because of the requirement that F is conservative the Hamilton-Lagrange[10] formalism of mechanics cannot handle systems which involve friction and dissipation. The systems they do handle we refer to as *Hamiltonian systems*.

When it comes to integrating Hamiltonian systems, the basic Runge-Kutta methods are not ideal. The fear when integrating such a system is that it will obtain non-Hamiltonian pertubations which makes the integration method unstable. The result of this is that we will will see a very different long-term behavior than one would expect of the given system. It is this fundamental problem that has led to the idea of symplectic integrators, which, by symplectic (canonical) transformations in each step of the integration avoids non-Hamiltonian pertubations[11].

Definition (1). *A linear mapping $A : \mathbb{R}^{2n} \mapsto \mathbb{R}^{2n}$ is called symplectic if*

$$A^T J A = J,$$

[10] Joseph Lagrange and William Hamilton came up with two different but equivalent formulations of the same thing.

[11] The full, rigorous mathematical explanation of symplecticity will not be given here.

where $J = \begin{pmatrix} 0 & I \\ -I & 0 \end{pmatrix}$, *$I$ is the identity matrix and A is the Jacobian matrix of the map for each integration step.*

To illustrate this mapping we can construct a parallelogram[5] spanned by two vectors

$$\xi = \begin{pmatrix} \xi^p \\ \xi^q \end{pmatrix} \quad , \quad \eta = \begin{pmatrix} \eta^p \\ \eta^q \end{pmatrix}$$

in the (p, q) space where $\xi^p, \xi^q, \eta^p, \eta^q \in \mathbb{R}^{2n}$ and $n \in \mathbb{Z}$ represents the number of degrees of freedom. These vectors are spanned as

$$P = \{t\xi + s\eta | 0 \leq t \leq 1, 0 \leq s \leq 1\}.$$

If we consider the oriented area for $n = 1$ we get

$$\text{or.area}(P) = \begin{vmatrix} \xi^p & \eta^p \\ \xi^q & \eta^q \end{vmatrix} = \xi^p \eta^q - \xi^q \eta^p$$

For higher dimensional space we sum the oriented area from $i = 1 \to n$ and this is defined as

$$\Phi(\xi, \eta) \equiv \sum_{i=1}^{n} \begin{vmatrix} \xi_i^p & \eta_i^p \\ \xi_i^q & \eta_i^q \end{vmatrix} = \sum_{i=1}^{n} \xi_i^p \eta_i^q - \xi_i^q \eta_i^p \tag{5}$$

An equivalent statement the the definition of the linear mapping above would be $\Phi(A\xi, A\eta) = \Phi(\xi, \eta)$, $\forall \eta, \xi \in \mathbb{R}^{2n}$. The best way, in my view, to illustrate this mapping is by a figure, and in figure 10 below we can see that the symplecticity of the linear mapping simply means that the area is conserved under the transformation. For higher dimensions, symplecticity means that the areas for the transformed parallelograms $A(P)$ is the same as the sum given in (5).

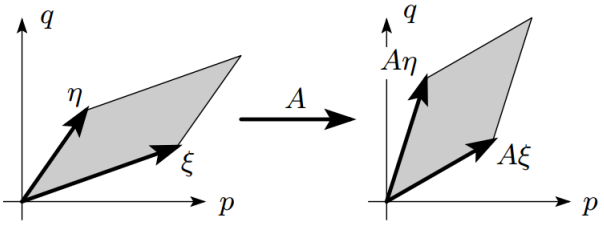

Figure 10: The area of this linear mapping is conserved under the transformation given in the definition.

These mappings however, are exact in the sense that they conserve the areas perfectly. Since we're dealing with non-linear dynamics we have to move on to non-linear mappings. Linear mappings are still useful because differentiable functions can be approximated by linear mappings locally.

Definition (2)**.** *A differentiable map $f : U \mapsto \mathbb{R}^{2n}$, where $U \subset \mathbb{R}^{2n}$ is an open set, is called symplectic if the Jacobian matrix f'(p, q) is everywhere symplectic, that is, if*

$$f'(p,q)^T J f'(p,q) = J \;\; or \;\; \Phi(f'(p,q)\xi, f'(p,q)\eta) = \Phi(\xi,\eta)$$

One could (and probably should) go on to give a geometrical interpretation of the non-linear mappings with Riemannian manifolds in order to explain the symplecticity of the Hamiltonian systems, but as stated earlier we will not give a rigorous mathematical account of this concept as a whole. The result tells us that all symplectic mappings, including non-linear ones, preserve areas.

We will, however, give a definition of the *flow* of the Hamiltonian system. By flow we simply mean the mapping which advances the solution by a time t, that is $\Psi_t(p_0, q_0) = (p(t, p_0, q_0), q(t, p_0, q_0))$ with initial values $p(0) = p_0$ and $q(0) = q_0$.

The area conservation of the flow can be illustrated by an example where we consider the pendolum which have kinetic and potential energies given by

$$T = \frac{1}{2}m(\dot{x}^2 + \dot{y}^2) = m\ell^2\dot{\alpha}^2/2 \;\; , \;\; U = mgy = -mg\ell\cos\alpha,$$

and so the Lagrange equations become

$$-mg\ell\sin\alpha - m\ell^2\ddot{\alpha} = 0$$

We can set $m = \ell = g = 1$. Since $q = \alpha$ and $p = \dot{\alpha}$, the Hamiltonian is given by

$$H(p,q) = \frac{p^2}{2m} - \cos q$$

Figure 11 shows the conservation of the area of the time flow Ψ_t. That is, although the shapes appear different, their area is the same.

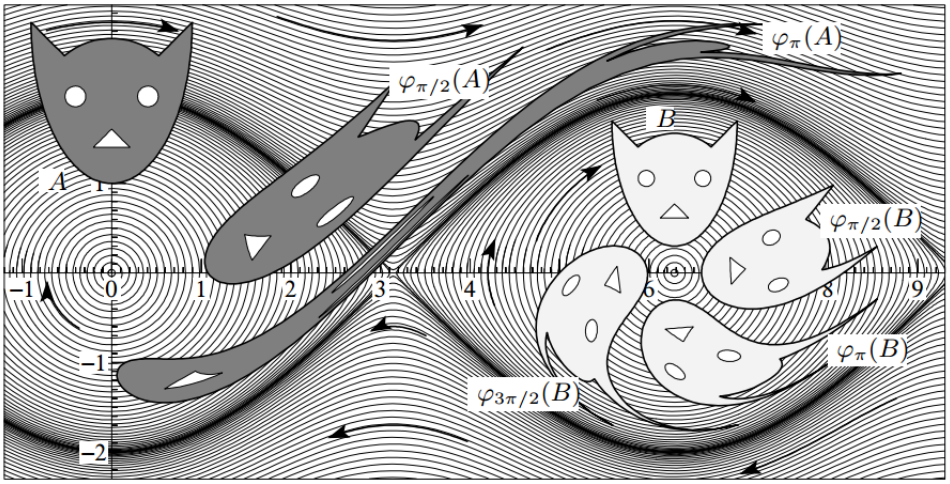

Figure 11: Level curves in the phase space of the pendolum where the shapes represents the areas that are conserved. This, however, is the ideal case. All credit goes to Lubich [5].

It is interesting to see how well these integration methods actually perform, due to the non-linearity of the Hamiltonian system. This is illustrated in figure 12. The white areas represent the error in the conservation of the area. That is, the implicit Euler scheme is definitely not symplectic. These are computed with step sizes $h = \pi/4$ for the first order methods and $h = \pi/3$ for the second order methods.

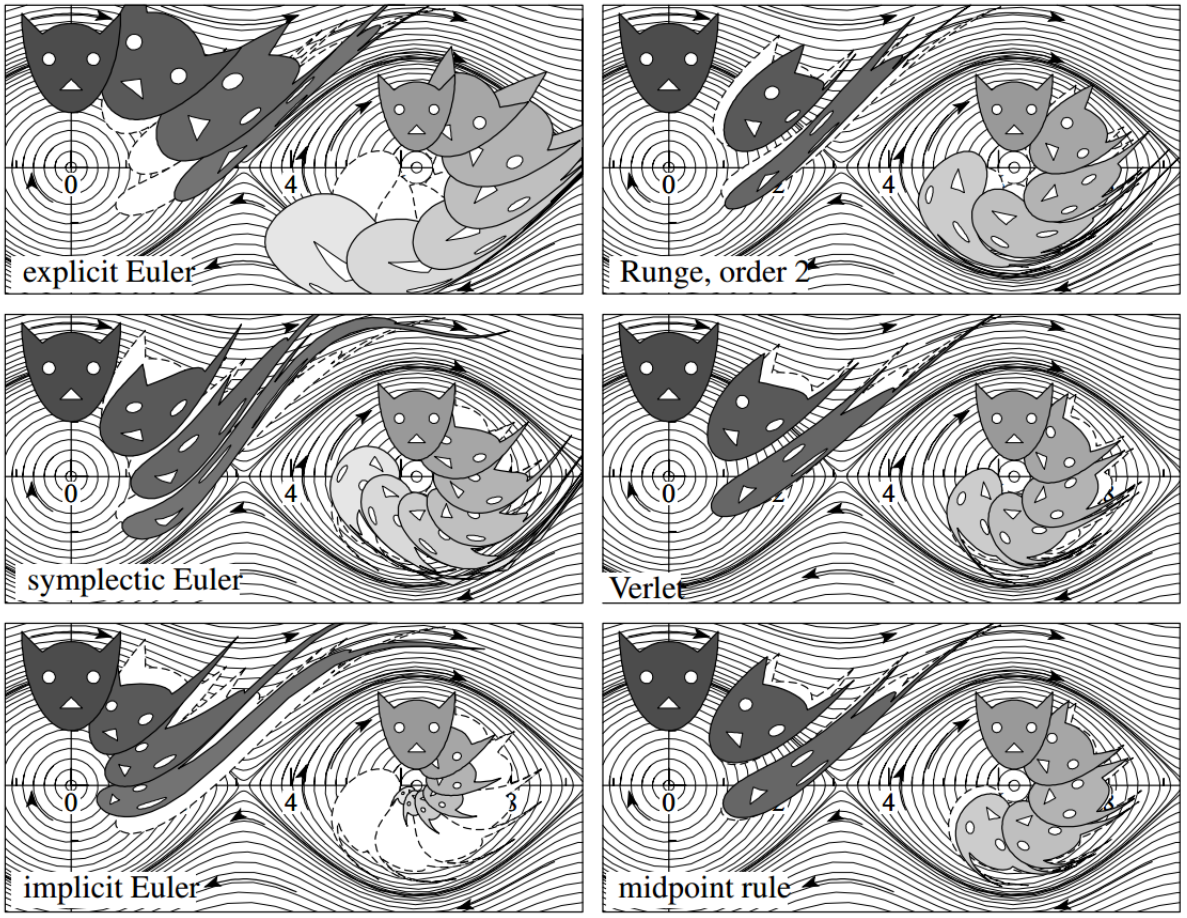

Figure 12: Level curves in the phase space of the pendolum where the author added a visualization of the area conservation with different integration schemes. Note that the Verlet method which we used in the report conserves the area fairly well. The explicit Euler however, does not do a good job.

When we look at figure 12 it becomes apparent that the symplectic integrators are not perfect, but they do avoid non-Hamiltonian pertubations in the integration.

8.2 Schwarzchild's letter to Einstein

Einstein published a paper in 1915 that would explain the perihelion precession of Mercury with an estimated $45'' \pm 5''$ per century[4]. Karl Schwarzchild sent a letter to Einstein dated 22 December 1915 which was sent from the Russian front[12]. This is the letter he sent[13].

Esteemed Mr. Einstein.

In order to become versed in your gravitation theory, I have been occupying myself more closely with the problem you posed in the paper on Mercury's perihelion and solved to 1st-order approximation. Initially, one factor made me very confused. I found for the coefficients $g_{\mu\nu}$ in first-order approximation, in addition to their solutions, also the following second one:

$$g_{\rho\sigma} = -\frac{\beta x_\rho x_\sigma}{r^5} + \delta_{\rho\sigma}\left[\frac{\beta}{3r^3}\right], \qquad g_{44} = 1$$

According to this there would be a second α in addition to yours and the problem would be physically ambiguous. Thereupon, I took my chances and made an attempt at a complete solution. A not overly lengthy calculation yielded the following result: There is only one line element that satisfies your conditions (1) to (4)[4], aside from the field and determinant eqs., and is singular at the origin and only at the origin.

$$\text{Let} \quad x_1 = r\cos\phi\cos\theta \quad x_2 = r\sin\phi\cos\theta \quad x_3 = r\sin\theta$$
$$R = (r^3 + \alpha^3)^{1/3} = r\left(1 + \frac{1}{3}\frac{\alpha^3}{r^0}\ldots\right),$$

then the line element reads:

$$ds^2 = \left(1 - \frac{\gamma}{R}\right)dt^3 - \frac{dR^2}{1 - \frac{\gamma}{R}} - R^2(d\theta^2 + \sin^2\theta d\phi^2).$$

R, θ, ϕ are not "admissible" coordinates with which the field equations could be formed, because they do not have the determinant 1, but the line element is written most neatly in them.
The equation for the orbit remains exactly the one obtained by you in first-order approximation (11), except under x not $\frac{1}{r}$, but $\frac{1}{R}$ must be understood, which is a difference of the order of 10^{-12}, thus practically absolutely irrelevant. The problem

[12]He was probably injured.
[13]Which is translated from german.

of the two arbitrary constants α and β, which the first-order approximation had yielded, is solved in that β must have a specific value of the order of α^4, the way α is given, otherwise in continuing the approximations the solution would be divergent. Thus the uniqueness of your problem is also in the best of order. It is a wonderful thing that the explanation for the Mercury anomaly emerges so convincingly from such an abstract idea.
As you see, the war is kindly disposed toward me, allowing me, despite fierce gunfire at a decidedly terrestrial distance, to take this walk into your land of ideas.[6]

8.3 Using Ovito

When writing data of the solar system to file using outputmode 'Ovito', our output looks like figure 13. The file contains the time series of our choice (in our case 250000 time steps). We had to scale the radius of the celestial bodies so they would fit the orbits around the Sun. To be able to see most of the celestial bodies we chose a logarithmic scaling shown in table 3.

```
10
The rows are the celestial bodies stacked, and the columns are x, y and z positions.
Sun -2.35527e-09 -6.32439e-10 6.71973e-11 0.188
Mercury -0.160985 0.27831 0.0374057 0.066
Venus 0.0171951 -0.723464 -0.0109114 0.085
Earth 0.979025 0.222714 -0.000176143 0.086
Mars 1.08097 -0.866353 -0.0448267 0.073
Jupiter -5.43314 -0.387141 0.123115 0.138
Saturn -2.31456 -9.76285 0.261867 0.134
Uranus 18.4773 7.52937 -0.211412 0.116
Neptune 28.2515 -9.94992 -0.446184 0.116
Pluto 9.39533 -31.8205 0.687287 0.05
10
The rows are the celestial bodies stacked, and the columns are x, y and z positions.
Sun -5.29848e-09 -1.42458e-09 1.51061e-10 0.188
Mercury -0.171784 0.27327 0.0379846 0.066
Venus 0.0245248 -0.723311 -0.0113324 0.085
Earth 0.977524 0.228816 -0.000176398 0.086
Mars 1.08437 -0.861932 -0.0448173 0.073
Jupiter -5.43298 -0.389758 0.123122 0.138
Saturn -2.31269 -9.76332 0.261801 0.134
Uranus 18.4767 7.53064 -0.211401 0.116
Neptune 28.2518 -9.94883 -0.446215 0.116
Pluto 9.39645 -31.8204 0.686954 0.05
```

Figure 13: The 24 first lines of the output file 'positions.xyz' when writing a file for simulating the solar system using Ovito.

The file format, called the xyz file format, is a lot different from what we're used to, but it's not that hard to understand. The formatting is a repeating pattern with

the following rows

```
<number of atoms>
comment line
<element> <X> <Y> <Z>
...
<element> <X> <Y> <Z>
```

The elements is in our case the celestial bodies in the solar system and the X, Y and Z defines the position in the X, Y and Z direction respectively.

Celestial body	Symbol of radius	Radius [km]	Radius [AU]	Scaled radius $(1 + \log \frac{R}{R_{Pluto}})/10$
Sun	$R_\odot$	695500	$4.637 \cdot 10^{-3}$	0.188
Mercury	$R_{Mercury}$	2440	$1.627 \cdot 10^{-5}$	0.066
Venus	R_{Venus}	6052	$4.035 \cdot 10^{-5}$	0.085
Earth	R_{Earth}	6371	$4.247 \cdot 10^{-5}$	0.086
Mars	R_{Mars}	3390	$2.260 \cdot 10^{-5}$	0.073
Jupiter	$R_{Jupiter}$	69911	$4.661 \cdot 10^{-4}$	0.138
Saturn	R_{Saturn}	58232	$3.882 \cdot 10^{-4}$	0.134
Uranus	R_{Uranus}	25362	$1.691 \cdot 10^{-4}$	0.116
Neptune	$R_{Neptune}$	24624	$1.642 \cdot 10^{-4}$	0.116
Pluto	R_{Pluto}	1195	$7.967 \cdot 10^{-6}$	0.050

Table 3: Table of the radius of the celestial bodies in the solar system. 1 AU is the average distance between the Sun and Earth and is given by 1 AU $= 1.5 \cdot 10^{11}$ m. Note: Pluto is no longer considered a planet, but a dwarf planet and we include it for historical reasons. The radius of the celestial bodies are collected from NASA[7].

To visualize the solar system in Ovito we had to load the file 'positions.xyz' containing the output. A setup-menu popped up and we chose the settings shown in table 4. Since Ovito is made for atomistic simulation data, our celestial bodies became particles.

File column	Particle property	Component
Column 1	Particle Type	
Column 2	Position	X
Column 3	Position	Y
Column 4	Position	Z
Column 5	Radius	

Table 4: Setting up Ovito to simulate the solar system.

After this we had to check a box down in the right lower corner under 'XYZ' → 'Timesteps' to tell Ovito that our file contains time series. We also had to uncheck a box in the right upper corner to tell Ovito that we didn't want a simulation cell (a box around our system).

When running the program for the whole solar system we had 250000 time steps and that runs slowly in Ovito. So to make the simulation go a bit faster we changed the animation settings by pushing the clock button (animation settings) and changed the values of 'Frame per second' to '60' and 'Playback speed in viewports' to '×20'. There is also a choice of render settings to only view every N-th frame.

The celestial bodies (particles) didn't look like the solar system as default, but to make them look more like the solar system we changed the color of the objects to the ones in figure 14.

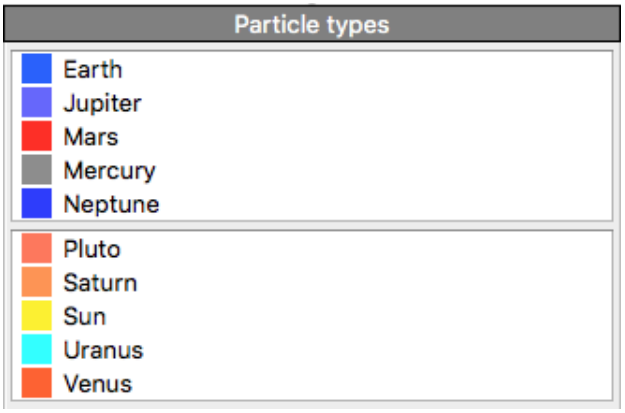

Figure 14: Changing the colors of the celestial bodies.

To play the animation we just pushed the play button, and by zooming and moving the picture with the hand button we got the simulation shown in figure 1 in section

5.1.

The masses and distances used are shown in table 5 below.

Celestial body	Symbol of mass	Mass [kg]	Mass scaled by mass of the Sun	Symbol of distance	Distance from the Sun [AU]
Sun	$M_\odot$	$2 \cdot 10^{30}$	1.0	-	-
Mercury	$M_{Mercury}$	$2.4 \cdot 10^{23}$	$1.2 \cdot 10^{-7}$	$r_{\text{Sun-Mercury}}$	0.39
Venus	M_{Venus}	$4.9 \cdot 10^{24}$	$2.45 \cdot 10^{-6}$	$r_{\text{Sun-Venus}}$	0.72
Earth	M_{Earth}	$6 \cdot 10^{24}$	$3.0 \cdot 10^{-6}$	$r_{\text{Sun-Earth}}$	1
Mars	M_{Mars}	$6.6 \cdot 10^{23}$	$3.3 \cdot 10^{-7}$	$r_{\text{Sun-Mars}}$	1.52
Jupiter	$M_{Jupiter}$	$1.9 \cdot 10^{27}$	$9.5 \cdot 10^{-4}$	$r_{\text{Sun-Jupiter}}$	5.20
Saturn	M_{Saturn}	$5.5 \cdot 10^{26}$	$2.75 \cdot 10^{-4}$	$r_{\text{Sun-Saturn}}$	9.54
Uranus	M_{Uranus}	$8.8 \cdot 10^{25}$	$4.4 \cdot 10^{-5}$	$r_{\text{Sun-Uranus}}$	19.19
Neptune	$M_{Neptune}$	$1.03 \cdot 10^{26}$	$5.15 \cdot 10^{-5}$	$r_{\text{Sun-Neptune}}$	30.06
Pluto	M_{Pluto}	$1.31 \cdot 10^{22}$	$6.55 \cdot 10^{-9}$	$r_{\text{Sun-Pluto}}$	39.53

Table 5: Table of the masses and the distances in the solar system. 1 AU is the average distance between the Sun and Earth and is given by 1 AU = $1.5 \cdot 10^{11}$ m. Note: Pluto is no longer considered a planet, but a dwarf planet and we include it for historical reasons. The mass of the celestial bodies are collected from NASA[7].

References

[1] Morten Hjort-Jensen. Computational physics, lecture notes fall 2015. Department of Physics, University of Oslo, 2015. https://github.com/CompPhysics/ComputationalPhysics/blob/master/doc/Lectures/lectures2015.pdf.

[2] Jon Cartwright. Physicists discover a whopping 13 new solutions to three-body problem. http://www.sciencemag.org/news/2013/03/physicists-discover-whopping-13-new-solutions-three-body-problem.

[3] J. J. O'Connor and E. F. Robertson. Bruns biography. University of St Andrews, Scotland. http://www-history.mcs.st-andrews.ac.uk/Biographies/Bruns.html.

[4] Albert Einsten. Explanation of the perihelion motion of mercury from general relativity theory. 25 November 1915 in Königlich Preußische Akademie der Wissenschaften, Republished by Anatoli Andrei Vankov, IPPE, Obninsk, Russia; Bethany College, KS, USA.

[5] Christian Lubich. From quantum to classical molecular dynamics: reduced models and numerical analysis. European Math. Soc., 2008. https://na.uni-tuebingen.de/~lubich/chap6.pdf.

[6] Karl Schwarzschild. Volume 8: The Berlin Years: Correspondence, 1914-1918 (English translation supplement), Princeton. http://einsteinpapers.press.princeton.edu/vol8-trans/191?ajax.

[7] National Aeronautics and Space Administration (NASA). Horizons web-interface. http://ssd.jpl.nasa.gov/horizons.cgi#top, using Ephemeris Type: Vectors.