FYS3150/FYS4150
Autumn 2016

Project 3: The solar system

Anisa Yaseen(worked with Solveig Andrea Fjeld and Sarah Rastad)

University of Oslo

October 25, 2016

Abstract

This project gives an overview on some of the most used methods to solve ordinary differential equations. We focus on initial value problems and present some of the most commonly used methods for solving such problems numerically. The aim of this project is to make a simulation of the solar system using gravitational forces. It will be written object oriented. We build a model for the solar system using ordinary differential equations, starting with the Earth-Sun system in 2-dimensions, keeping the sun fixed as the center of the mass of the system. Differential equations will be solved by using two different methods/algorithms, namely Euler's method and the velocity Verlet method. Tasks of this project involve writing an object oriented code for two-body system, testing the algorithms and finding the escape velocity. Further study of solar system (i.e. a three-body problem) demands modifying differential equations in order to accommodate both the motion of the Earth and Jupiter(included into former 2-body system). In the end one would want it to be possible to add all celestial objects of the solar system, check the stability of the Verlet method, as well as the perihelion precession of Mercury.

All source codes and programs used are hosted on Github, at:
`https://github.com/sarahbra/Project3/tree/master/solar-system-fys3150`

Contents

1 Motivation

One can obtain detailed dynamical information about interacting classical systems from molecular dynamics simulations, which require integrating Newton's equations of motion over a long period of time starting from some initial conditions. One might be interested, for example, in following; the motion of the solar system for a long period of time, or consider the evolution of a galaxy by following the motions of its constituent stars. As another example (from fluid dynamics), one might want to integrate the motion of atoms in a fluid. This project discuss an algorithm, called The veloctiy Verlet, which is particularly suited for these kind of simulations because (i) it is simple, and (ii) it has stability.

2 Introduction

In this project we will study and develope a code for stimulating the solar system using the so-called veloctiy Verlet algorithm, a widely used algorithm for solving coupled ordinary differential equations. In first part of the project we test the algorithm by litmting ourselves to a hypothetical solar system with only earth orbiting around the sun. Thereby focusing only on the gravitational force between the Earth and Sun. We discretize the differential equations and set up an algorithm for these equation by using different methods namely Euler's forward algorithm and co-called veloctiy Verlet. Our next task is to wirte an object oriented code for the Earth-Sun system which solves the differential equations using Euler's method and velocity Verlet method. By doing so we will also try to figure out which parts and operations can be written as classes and generalized, and can be reused. Further we will test the stability of our algorithm as function of different time steps Δt and find out which intitial value for the velocity gives a circular orbit around the sun. We will aslo make a plot of obtained result for Earth's orbital position around the Sun, and check if total engery(both kinetic and potential) as well as the momentem is conserved. Moreover we will give an explanation of why these identities are conserved and eventually discuss the differences between above two algorithms(Verlet algoritm and Euler's algorithm). For the remaining part of the project we will use the velocity Verlet algorithm. By using this method we will find out the initial velocity at which the planet can escape sun's gravitational pull. Our next part of the project invovles three-body problem, where we study three-body problem with sun kept as the center of mass of the system and including Jupiter(as mentioned in abstract). We will here find out how much Jupiter shifts the Earth's position. We repeat our task as given in section 3c(in the project) but this time focuing on three-body problem. At last but not least using Verlet algorithm we study a real three-body problem, where all three planets, the Earth, Jupitar and the Sun are in motion. In the end we study the perihelion precession of Mercury, We will run a simulation over once century of Mercury's orbit around the Sun whith no other planets present in the system, starting with Mercury at prihelion on the x-axis and check the value of perihelion angle. Our finnal target is to explain if the perihellion precession can be explained by the general theory of relativity.

3 Theory

3.1 A hypothetical solar system

In a hypothetical solar system with Earth only orbiting around the Sun, the only force between these two bodies is gravitational force, which is given by Newton's law of gravitation:

$$F_G = \frac{GM_{Sun}M_{Earth}}{r^2} \tag{1}$$

where M_{Sun} is the mass of the Sun and M_{Earth} is Earth's mass. G is gravitational constant and r is the distance between the Earth and the Sun. Since the mass of the Sun is much larger than that of the Earth we can thereby neglect the motion of the Sun in this problem. By using different methods for solving differential equations we compute the motion of the Earth around the Sun. We have assumed that the motion of the Earth is co-planar(lies in the same plane) and we take this to be the xy-plane. We have limited ourselves to a co-planer motion and used only the x and y coordinates in setting up the equations. Implying newton's second law of motion yeilds:

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

and

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

where $F_{G,x}$ and $F_{G,y}$ are x and y components of the gravitational force. We used astronomical units when writting our equations, 1 AU($1\text{AU} = 1.5 \times 10^{-11} m$) is the average distance between the Sun and Earth. In order to start differential equation solver we have extraced the initial conditions from NASA's webpage given in theory section of the project.

3.2 The perihelion precession of Mercury

As it is stated in the project "An important test of the general theory of relativity was to comapre its prediction for the perihelion precession of Mercury to the observed value...", unfortunatly our lack of astro-physical knowledge gives rise to a question "what does perihelion precession mean?". Well the answer to this question is as following; The thoery of general relativity explaines slight alternations in Mercury's orbit around the Sun. Since Mercury's orbit is elliptical, the orientation of this ellipse's long axis slowly rotates around the sun. This process is knows as the "Precession of the perihelion of Mercury" in astronomical terms. As the closets planet to the Sun, Mercury orbit in a region in the solar system where spacetime is distrubed by the Sun's mass. Thus Mercury's elliptical path shifts slitghly with each orbit aroun the Sun such that its nearest point to the Sun shift forwards with each time pass. In other words the point of closets approach of Mercury to the Sun does not always occur at the same place but that it slowly moves around the Sun. As seen from Earth the precession of Mercury's orbit is measured to be 5600 seconds of arc per century, where one second of arc = $\frac{1}{3600}$ degrees. The precession is a result of a classical behavior, almost all of the movement of the perihelion (around 5030 arcseconds per century) is present in a two-body system with point masses for the Sun and Mercury. Gravitational effects of the other planets around the system cause another 530 arcseconds per century. Leaving us with 40 arcseconds per century of unexplained movement. The observed value of 5599.7 arcseconds per century is measured very accurately, to within 0.04 arcseconds per century, so this means a significant deviation. It turns out that 43 arcseconds per century are expected to result from general relativity, or another way of explaining this is that the curvature of spacetime itself by the two bodies causes some changes to the gravitational potential, so it isn't really exactly $\frac{GM_{Sun}M_{Mercury}}{r^2}$. By adding the general relativity correction it is given as:

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

4 Algorithms and Methods

As mentioned above in introduction we implement two methods and compare them. The first one namely Euler's forward algorithm while the second is so-called velocity Verlet.

4.1 Euler's method

The Euler's algorithm is possibally the simplest way to solve a differential equation. It numerically approximate solution of first order ordinary differential equations with a given intial value. The ODE has to be provided in the following form:

$$\frac{dy(t)}{dt} = f(t, y(t)) \tag{5}$$

with intial value, $y(t_0) = y_0$. To get a numerical solution, we replace the derivative on the left hand side with a finite difference approximation () as:

$$\frac{dy(t)}{dt} \approx \frac{y(t+h) - y(t)}{h} \tag{6}$$

then we solve for $y(t+h)$; $y(t+h) \approx y(t) + h\frac{dy(t)}{t}$.

The rule if iterative solution is thus;

$$y_{n+1} = y_n + hf(t_n, y) \tag{7}$$

which is same as:

$$y(t+h) \approx y(t) + hf(t_n, y_n) \tag{8}$$

Here we have assumed the time $t_n = nh$ moves ahead in unifrom "ticks" h, meaning that the parameter h is step size length, the most relevant parameter for accuracy of the solution. A smaller step size increases the accuracy but at the computation cots, therefor it has always has to be hand picked according to the problem. To find y_{n+2} we only have to replace y_n with y_{n+1} and thereby progressively step outwards to the solution for arbitrary values of t.

Here are our implementation of Euler's method:

```
 //Initiating variables
Particle *p = particles.at(i);
double m = p->getMass();

vec3 F = p->getForce();
vec3 a = F.operator /=(m);
vec3 v = p->getVelocity();
double dt = getDt();

//Euler algorithm (the assignment said Euler, which is why we chose this and not Euler Cro
p->getPosition().operator +=(v.operator *=(dt));
p->getVelocity().operator +=(a.operator *=(dt));
```

4.2 Verlet method

In Molecular dynamics, the most widely used time integartion algorithm is the so-called Verlet algorithm. The basic idea is to write two-third order Taylor expansions for the position $r(t)$, one forward and one backward in time. If we now call v the velocity, a the acceleration and b the third derivaties of r with respect to to, we have:

$$r(t+\Delta t) = r(t) + v(t)\Delta t + \frac{1}{2}a(t)\Delta t^2 + \frac{1}{6}b(t)\Delta t^3 + O(\Delta t^4)$$
$$r(t-\Delta t) = r(t) - v(t)\Delta t + \frac{1}{2}a(t)\Delta t^2 - \frac{1}{6}b(t)\Delta t^3 + O(\Delta t^4)$$

If we add the above two expressions we get:

$$r(t+\Delta t) = 2(t) - r(t-\Delta t) + a(t)\Delta t^2 + O(\Delta t^4) \tag{9}$$

Since we are integrating Newton's equations, $a(t)$ is just the force devided by the mass thus the force becomes a function of the positions $r(t)$:

$$a(t) = -\frac{1}{m}\Delta V(r(t)) \tag{10}$$

We can immediatly see the truncation error of the algorithm when evovling the system by Δt is of the order of Δt^4, even if the third derivative do not appear expicitly. Verlet algorithm is numecially accurate, stable and easy to implement. We consider a second order differential equation like Newton's law($F = ma$) whose one-dimensional equation reads as;

$$m\frac{d^2x}{dt^2} = F(x,t) \tag{11}$$

We rewrite above equation in terms of two coupled differantial equations: $\frac{dx}{dt} = v(x,t)$ and $\frac{F(x,t)}{m} = a(x,t)$ By performing Tayler expansion we get:

$$x(t+h) = x(t) + hx^{(1)}(t) + \frac{h^2}{2}x^{(2)}(t) + O(h^3) \tag{12}$$

where the second derivative in our case is knows as $x^{(2)}(t) = a(x,t)$. A similar expression is obtained for the term $x(t-h)$. Fruther more we can add those two expressions and discretize them by using $x(t_i + h) = x_{i+1}$, $x_i = x(t_i)$ and arrive at:

$$x_{i+1} = 2x_i - x_{i-1} + h^2x_i^{(2)} + O(h^4) \tag{13}$$

We can see that the problem with this version of Verlet algorithm is that velocity term is not directly included. We can compute it by a formula:
$x_i^{(1)} = \frac{x_{i+1}-x_{i-1}}{2h} + O(h^2)$ We see now that the velocity has a truncation error which is of order of $O(h^2)$. A corresponding term can be obtained for the velocity. Newton's law gives us an analytical expression for the derivative of the velocity; $v_i^{(1)} = \frac{d^2x}{dt^2}|_i = \frac{F(x_i,t_i)}{m}$. We now add the corresponding expansion for the derivative of the velocity and keep only terms up to second derivatie of the velocity since we know that the order of error is $O(h^3)$ thus we have:
$hv_i^{(2)} \approx v_{i+1}^{(1)} - v_i^{(1)}$ one can again rewrite velocity term in the following final form:

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

Our final equation for the position is thus:

$$x_{i+1} = x_i + hv_i + \frac{h}{2}v_i^{(1)} + O(h^3) \tag{15}$$

Here the term $v_i^{(1)}$ has positions dependence at x_{i+1}, meaning that one needs to calculate the position at the updated time t_{i+1} before computing the next v.

Here is our implementation of the Velocity Verlet Algorithm:

```
//Defining variables

Particle *p = particles.at(i);
double dt = getDt();
double m = p->getMass();
vec3 v = p->getVelocity();


//Calculating the acceleration
vec3 F1 = p->getForce();
vec3 a1 = F1.operator /=(m);

//calculating the position
double temp = dt/2;
```

```
v.operator *=(dt);

p->getPosition().operator += (v);
p->getPosition().operator +=(F1.operator *=(temp*dt));

//computing the force to calculate the new velocity
m_system->computeForces();

//calculating the velocity
vec3 a2 = p->getForce().operator /=(m);
a2.operator +=(a1);
a2.operator *=(temp);

p->getVelocity().operator +=(a2);
```

5 Building a code for solar system

5.1 The Earth sun system

We first start with a hypothetical solar system, the Earth-Sun system in two dimensions. The gravitational force acting between these two bodies is given by $F_G = \frac{GM_{Sun}M_{Earth}}{r^2}$. We assume that the Earth's orbit is circular around the Sun thereby the force must obey following relation:

$$F_G = \frac{M_{Earth}v^2}{r} = \frac{GM_{Sun}M_{Earth}}{r^2} \tag{16}$$

where v is Earth's velocity, G is the gravitational constant, $M_{Earth} = 6 \times 10^{24}Kg$ is mass of the earth and $M_{Sun} = 2 \times 10^{30}Kg$ is the mass of the Sun. The above equation can be rewritten in one dimensional as:

$$\frac{d^2x}{dt^2} = \frac{F_x}{M_{Earth}} \tag{17}$$

and

$$\frac{d^2y}{dt^2} = \frac{F_y}{M_{Earth}} \tag{18}$$

in x and y direction. By using polar coordinates we can rewrite:

$$F_x = -\frac{GM_{Sun}M_{Earth}}{r^2}cos(\theta) = -\frac{GM_{Sun}M_{Earth}}{r^3}x, \tag{19}$$

and

$$F_y = -\frac{GM_{Sun}M_{Earth}}{r^2}sin(\theta) = -\frac{GM_{Sun}M_{Earth}}{r^3}y, \tag{20}$$

These two expressions for force are general and can be used in all the cases, so this is the force we implemented in our program. since $x = rcos(\theta)$, $y = rsin(\theta)$ and $r = \sqrt{x^2 + y^2}$. Moreover rewriting above equations as four first-order coupled differential equations we have:

$$\begin{aligned} \frac{dv_x}{dt} &= -\frac{GM_{Sun}}{r^3}x, \\ \frac{dx}{dt} &= v_x, \\ \frac{dv_y}{dt} &= -\frac{GM_{Sun}}{r^3}y, \\ \frac{dy}{dt} &= v_y. \end{aligned}$$

As mentioned in the beginning of this section our equation from circular motion is of form:

$$\begin{aligned} F_G &= M_{Earth}a \\ \frac{M_{Earth}v^2}{r} = F_G &= \frac{GM_{Sun}M_{Earth}}{r^2} \\ v^2r = GM_{Sun} &= \frac{4\pi^2AU^3}{yr^2} \end{aligned}$$

where we have used centripetal acceleration $a^{=}\frac{v^2}{r}$. After rearranging we have $GM_{Sun} = v^2 r$, since earth's velocity is $v = \frac{2\pi r}{yr} = \frac{2\pi AU}{yr}$ due to the fact that we have assumed circular motion. We end up with following relation:
$GM_{Sun} = v^2 r = 4\pi^2 \frac{(AU)^3}{yr^2}$ where we have used the average distance between the Sun and Earth converted in astronomical units (AU). $1AU = 1.5 \times 10^{11} m$. From this expression we find the gravitational constant that we use in our program. The gravitational constant are given in units $\frac{AU^3}{yr^2 M_{sun}}$

5.2 Test of the algorithms

From our calculation we have found that the initial value at which we get circular orbit is: $V_{initial} = 2\pi$.

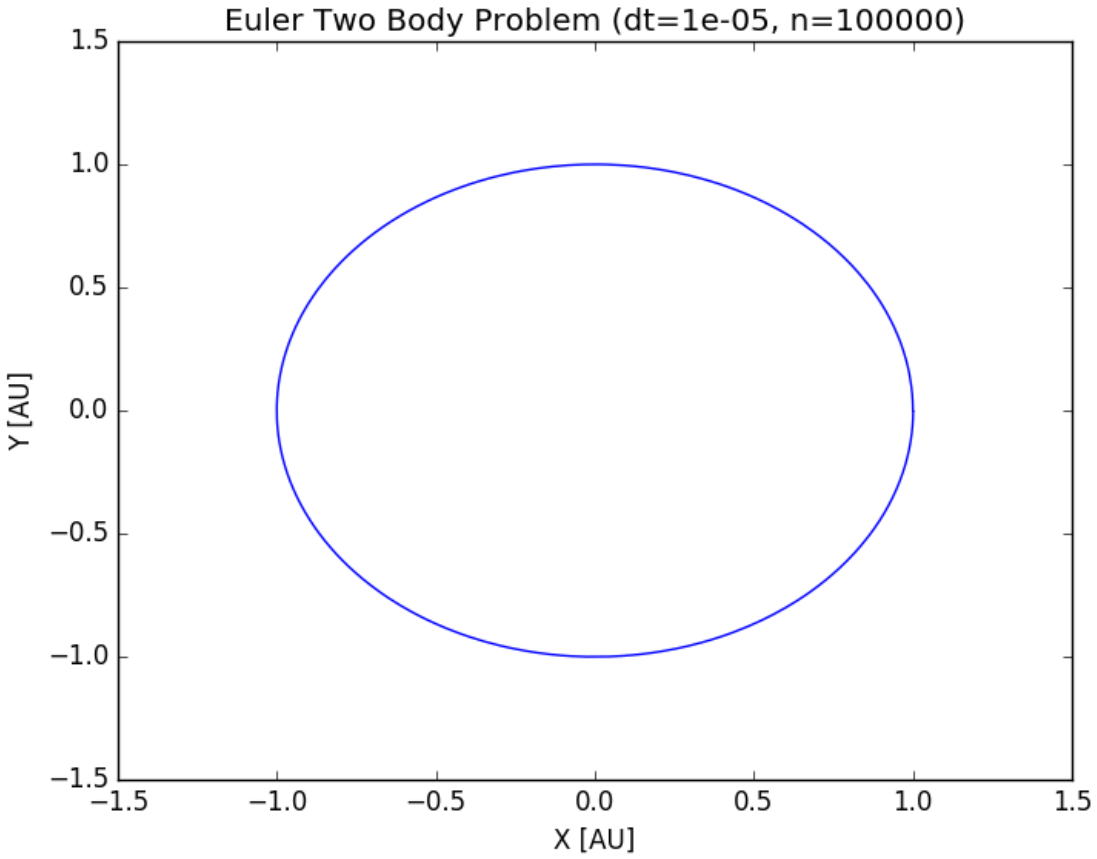

Figure 1: Two-body system: using Euler's algorithm

For the case of a circular orbit the total mechanical energy for an object in a central gravitational field gives a beautiful simple result. As we know the mass of Earth is smaller than the mass of the Sun, we have ignored the motion of the Sun as an approximation, and held the center of mass as the origin of our coordinate system. Since gravity is the only force doing work: **The total mechenical energy $E = E_k + E_p$ of the system is conserved**. The total mechanical energy for a planet with mass, M_1 in a circular orbit with radius, R around a body with mass M_2 can be written:

$$E_{total} = -\frac{GM_2M_1}{R} + \frac{1}{2}mv^2 \tag{21}$$

We eliminate v by equating the net force in circular motion to the gravitational force

$$m\frac{v^2}{R} = G\frac{M_2M_1}{R^2} \Rightarrow mv^2 = G\frac{M_2M_1}{R} \tag{22}$$

Inserting this into equation 20 we find that for a circular orbit, the kinetic energy is $\frac{1}{2}$ the size of potential energy.

$$E_{total} = -\frac{GM_2M_1}{R} + \frac{1}{2}G\frac{M_2M_1}{R} = -G\frac{M_2M_1}{2R} \tag{23}$$

Why is total energy or kinetic and potential energy conserved? it is becuase; When in circular motion, an object remains the same distance above the surface of the planet; that is, its radius of orbit is fixed. Furthermore, its speed remains constant. The speed at all positions at circular orbit are the same. The heights above the Sun's surface at all positions are also the same. Since kinetic energy is dependent upon the speed of an object, the amount of kinetic energy will be constant throughout the Earth /planet's motion. And since potential energy is dependent upon the height(distance in this case) of an object, the amount of potential energy will be constant throughout the Earth's motion. So if the kinetic and the potential remain constant, it is quite reasonable to believe that the total mechanic energy remains constant.

Moreover, the gravitational force acts along the line between the two bodies, so there is no torque about th origin. This means that: **The total angular momentum L of the system about the origin is also conserved**. Recall that $\sum T_{external} = \frac{dL}{dt}$. In light of this equation, consider the special case of when there is no net torque acting on the system. In this case, $\frac{dL}{dt}$ must be zero, implying that the total angular momentum of a system is constant. We can state this verbally: "If no net external torque acts on a system, the total angular momentum of the system remains constant".

Differences between the Verlet algorithm and the Euler algorithm: The Euler method is a first order integration scheme, i.e. the total error is proportional to the step size, however it can be numerically unstable. Meaning that the accumulated error can overwhelm the calculation. The instability can occur regardless of how small we make the step size. What happens with Euler is that one calculate a velocity by dividing by the timestep, then next frame, one will end up calculating a change in position by multiplying the timestep. The further away from 1.0f(with Euler we chose time-step not close to 1.0f) this timestep is, the more precision we lose in that mulitpication or division.

The Velocity Verlet assumes that the timestep is the same, so it just disposes with the multiplication or division altogether, which means that it doesn't suffer that precision loss, but also means one need a fixed timestep. The Verlet provides both the positions and velocities synchronously (at the same time-steps) and also requires only the initial positions and initial velocities to initiate. In contrast with Euler's method, the Verlet method is a second order integration and is good at simulating systems with energy conservation. The Verlet method is sympletic and time reversible, desirable properties that reflect the physical reality of certain simulation problems. Sympletic means that when the methods are guaranteed to conserve total energy in conservative simulation problems. For instance taking the simulation of the earth spinning around the sun. Without an external dissipation of energy, the earth is guaranteed to remain indefinitely in orbit. But using Euler to perform the simulation, the earth would eventually leave orbit or crash into the sun, due to a small numerical drift in the total energy that builds up over a long period of simulation. Symplectic methods like Verlet explicitly preserve the total energy, so the numerical solution is also guaranteed to keep the earth indefinitely in orbit. Time-reversibility means that the (fixed-step) method can take i steps forward in time, followed by i steps backwards in time, and arrive at the same initial conditions used to start the simulation. We often apply time-reversible numerical methods to time-reversible physical processes; we then expect the numerical solution to have long-time behavior similar to that of the exact solution.

So a good method to check the stability of the system is to check if the total energy is zero, and the momentum is constant. So in our code we calculate the potential when we calculate the force, we also have a function that calculates the kinetic energy. We print both of these in the terminal window. We also print the total energy in the terminal. We have also made a function in our system which calculates the total momentum of the system and also print this in the terminal window.

5.3 Two-Body problem

In figure 2 we have plotted the circular orbit of the earth around the Sun, and in figure 3 we have zoomed in so that we can see the stability of the Euler algorithm. We can see that the Euler algorithm carry over error so that the distance from the sun gets continually larger.

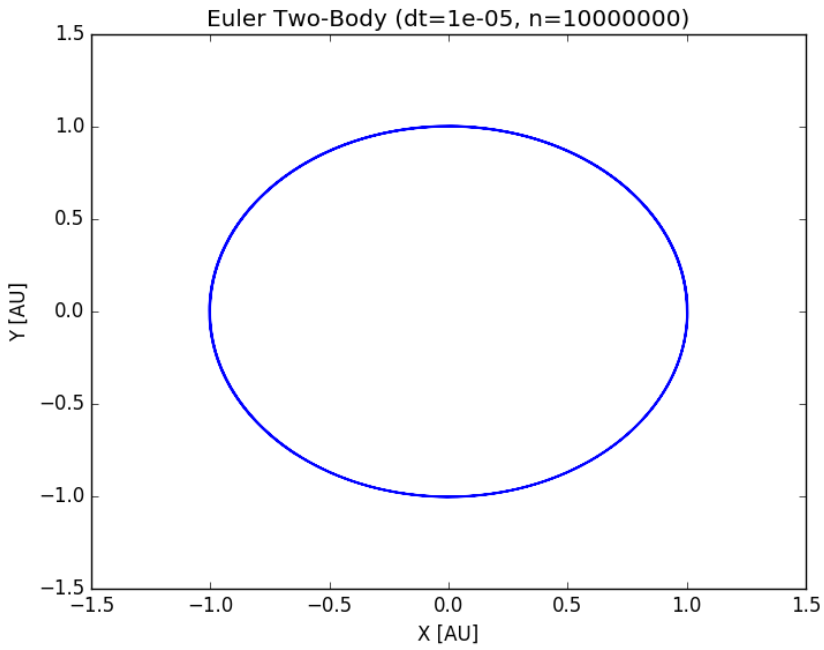

Figure 2: Two-body system: using Euler's algorithm

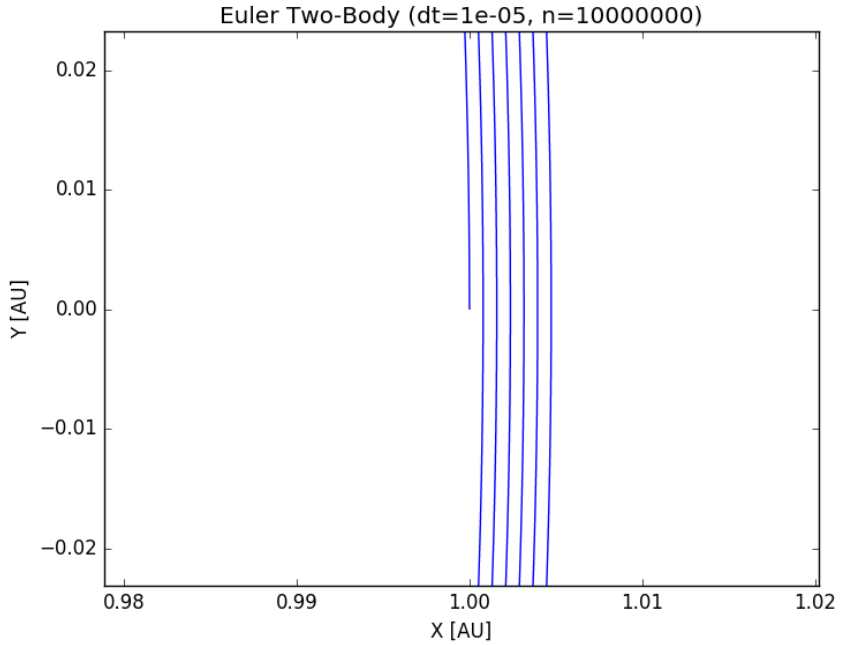

Figure 3: Two-body system: using Euler's algorithm

When we consider the energy and momentum of the system we get:

```
STARTING INTEGRATION
------------------------
  o Number of steps:     1000000
  o Time step, dt:       1e-05
  o Initial condition:   Two-body
  o Number of particles: 2
  o Potential in use:    Newtonian gravity
  o Integrator in use:   Euler

Step: 100000   E =-0.00000   Ek = 0.00012   Ep =-0.00012   M= 0.00002
Step: 200000   E =-0.00000   Ek = 0.00012   Ep =-0.00012   M= 0.00002
Step: 300000   E =-0.00000   Ek = 0.00012   Ep =-0.00012   M= 0.00002
Step: 400000   E =-0.00000   Ek = 0.00012   Ep =-0.00012   M= 0.00002
Step: 500000   E =-0.00000   Ek = 0.00012   Ep =-0.00012   M= 0.00002
Step: 600000   E =-0.00000   Ek = 0.00012   Ep =-0.00012   M= 0.00002
Step: 700000   E =-0.00000   Ek = 0.00012   Ep =-0.00012   M= 0.00002
Step: 800000   E =-0.00000   Ek = 0.00012   Ep =-0.00012   M= 0.00002
Step: 900000   E =-0.00000   Ek = 0.00012   Ep =-0.00012   M= 0.00002
Step: 1000000   E =-0.00000   Ek = 0.00012   Ep =-0.00012   M= 0.00002
Time computations took:    6.63305s
```

Which shows that the total energy is equal to zero as expected, and the total momentum is equal to a constant as expected.

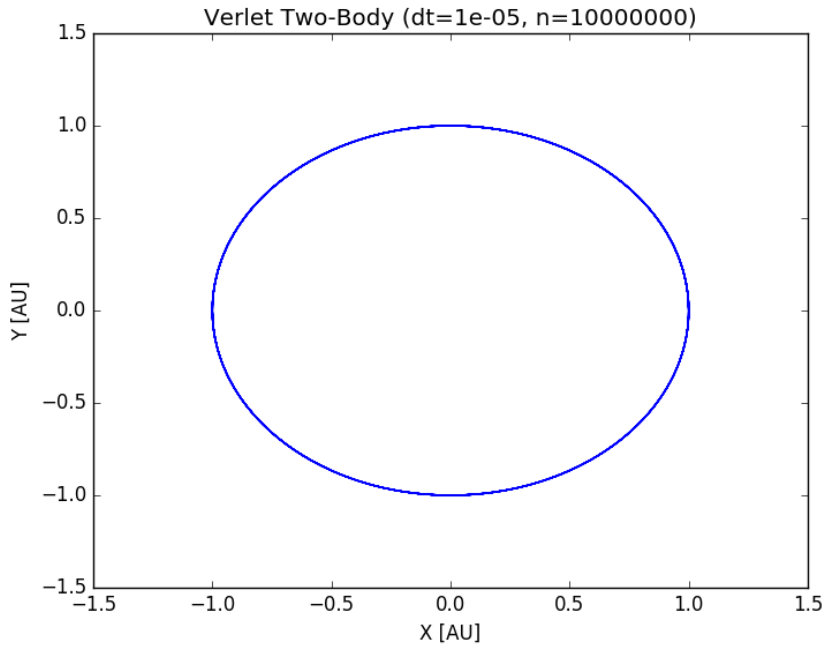

Figure 4: Two-body system: using Verlet's algorithm

We have plotted the Earth's orbit around the Sun using the Velocity Verlet algorithm in figure 4 and figure 5. These obserevation reveale that the verlet is alot more stable than Euler since one can asily see that the orbit is completely circular. One can also state that the energy and momentum is conserved:

```
 STARTING INTEGRATION
------------------------
  o Number of steps:     1000000
```

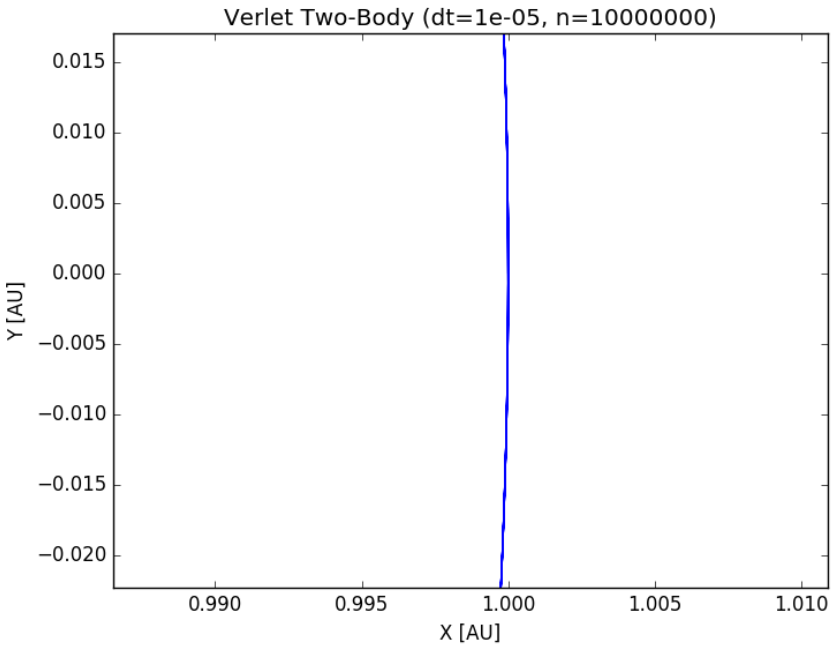

Figure 5: Two-body system: using Verlet's algorithm (zoomed

```
  o Time step, dt:       1e-05
  o Initial condition:   Two-body
  o Number of particles: 2
  o Potential in use:    Newtonian gravity
  o Integrator in use:   Velocity verlet

Step: 100000   E = 0.00000   Ek = 0.00012   Ep =-0.00012   M= 0.00002
Step: 200000   E = 0.00000   Ek = 0.00012   Ep =-0.00012   M= 0.00002
Step: 300000   E = 0.00000   Ek = 0.00012   Ep =-0.00012   M= 0.00002
Step: 400000   E = 0.00000   Ek = 0.00012   Ep =-0.00012   M= 0.00002
Step: 500000   E = 0.00000   Ek = 0.00012   Ep =-0.00012   M= 0.00002
Step: 600000   E = 0.00000   Ek = 0.00012   Ep =-0.00012   M= 0.00002
Step: 700000   E = 0.00000   Ek = 0.00012   Ep =-0.00012   M= 0.00002
Step: 800000   E = 0.00000   Ek = 0.00012   Ep =-0.00012   M= 0.00002
Step: 900000   E =-0.00000   Ek = 0.00012   Ep =-0.00012   M= 0.00002
Step: 1000000   E =-0.00000   Ek = 0.00012   Ep =-0.00012   M= 0.00002
Time computations took:    7.67353s
```

Algorithm	FLOPs	Time(n=100000)
Euler	10	6.633[s]
Verlet	35	7.673[s]

Table 1: FLOPs and computing time.

The number of FLOPs for the Verlet methods is in total 35; Note that the FLOPs for just the position in Verlet are 10, remanding comes from the fact that we must calculate the force of the system twice in the loop, since we need the new acceleration to calculate the new velocity. Since we hav eto calculate the Force twice it becomes a lot more demanding computationally

By comparing the number of FLOPs and computing time of both methods we see that the Verlets method took longer time than Eulers, reason is that Verlet has much more FLOPs than Euler, but still it is quite fast and more stable. We can thereby conclude that the Verlet method

is much sufficient then Euler's.

5.4 Escape velocity

We know that if the kinetic energy of the an object with mass M_1 from a planet of mass M_2 is equal in magnitude to the potential energy then in the absence of friction resistance it could escape from the planet's gravitational pull. The escape velocity depends on the distance from sun. For a circular orbit around Sun the escape velocity is given by: $U = -\frac{GM_1M_2}{R} = \frac{1}{2}M_1v^2$

$$V_{escape} = \sqrt{\frac{2GM_2}{R}} \tag{24}$$

where G is gravitational constant, M_2 is the mass of the Sun, and R is Earth's distance from the Sun. If we set gravitational force to be equal to centripetal force: $\frac{GM_1M_2}{R^2} = \frac{M_1v^2}{R}$ thus from this relation we have that the escape velocity is $v_{escape} = \sqrt{2}\sqrt{\frac{GM_2}{R_{Sun}}}$. We have our values given by: $G = 6.64 \times 10^{-11}[m^3kg^{-1}s^{-2}]$, $M_{Sun} = 2 \times 10^{30}[kg]$ and $R = r = 1.5 \times 10^{11}[m]$. Inserting the values we get: $V_{escape} = 42079[\frac{m}{s}] \approx 42.1[\frac{km}{s}] \approx 8.8763[\frac{AU}{yr}]$. By trails and error we have found that the numerical value of escape velocity is $V_{escape} = 2\pi + 2.6 \approx 8.8831[\frac{AU}{Yr}]$, a deviation equal to a value $0.0068 \approx 0.007$. Hence our numerical value matches the exact value of escape velocity quite well.

5.5 The three-body problem

In study of three-body system we include Jupiter in our 2-body system. Jupiter having a mass 1000 times smaller than the mass of the Sun, does effect the position of the Earth in solar system. Our aim is to find how Jupiter alters Earth's position. We modify our program by adding the gravitational force between The Earth and jupiter, and plot the result in figure. We then repeat calculation by increaing the mass of jupiter by a factor of 10 and 1000 and plotted the result. See figure 6, 7, and 8.

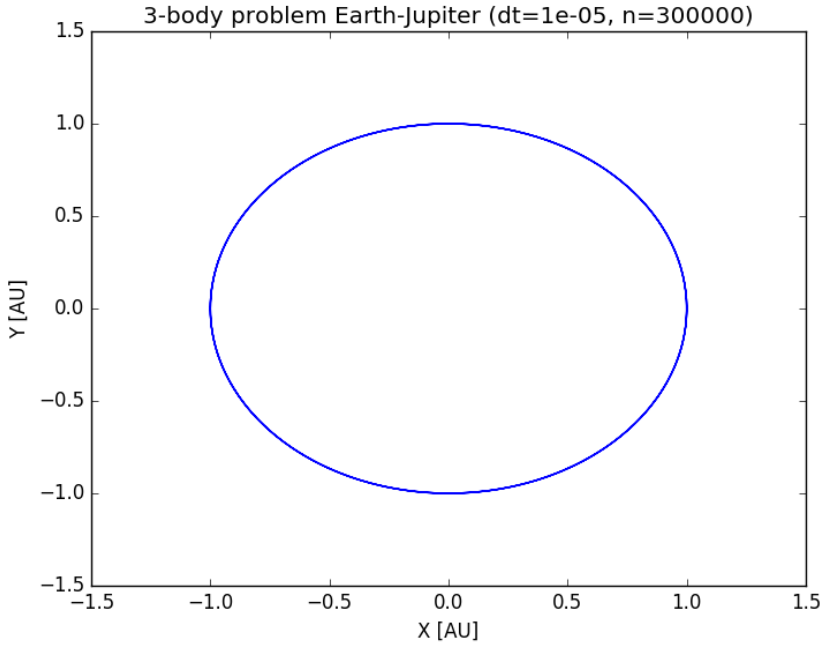

Figure 6: Three-body system: using Verlet algorithm

In figure 6 we have plotted the orbit of earth when it is also affected by the gravitational pull of Jupiter. We see that the orbit isn't much affected by the pull of Jupiter. The inital velocity have been taken from the NASA site given in the project. Here is the total energy and total from this run:

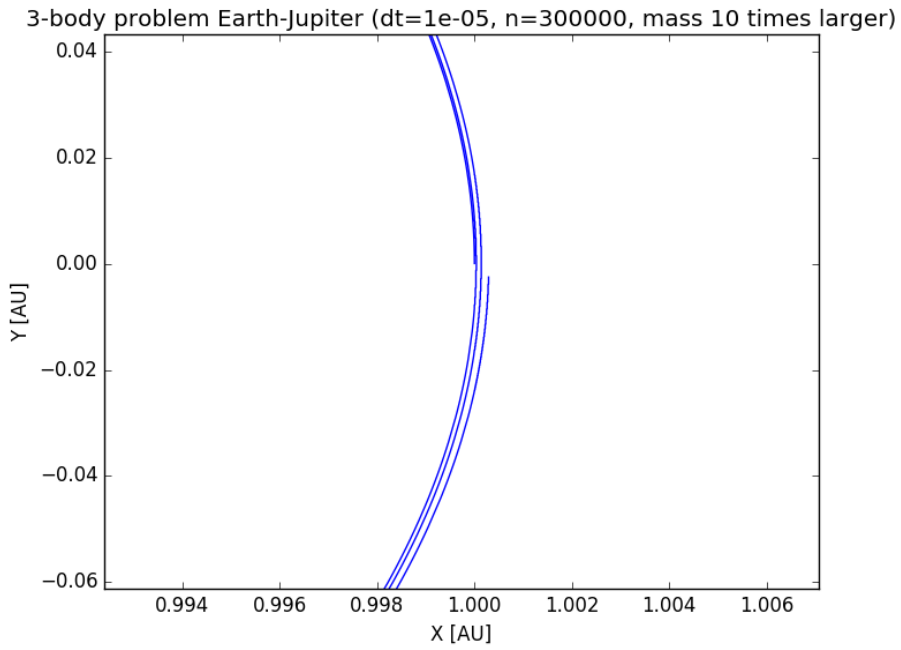

Figure 7: Three-body system:

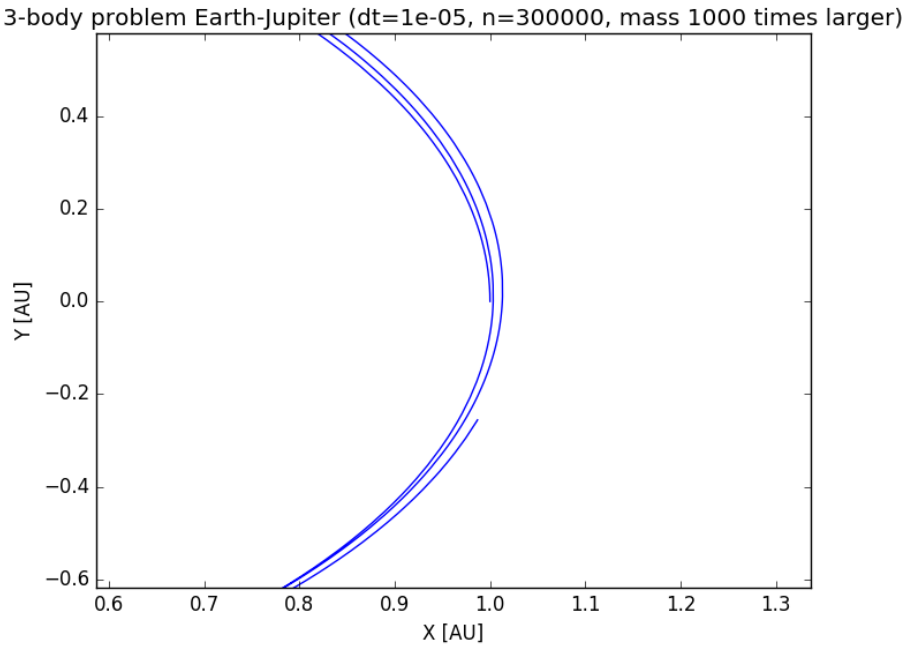

Figure 8: Three-body system:

```
 STARTING INTEGRATION
-------------------------
  o Number of steps:     100000
  o Time step, dt:       1e-05
  o Initial condition:   Three-body
  o Number of particles: 3
  o Potential in use:    Newtonian gravity
  o Integrator in use:   Velocity verlet

Step: 10000   E =-0.00036   Ek = 0.00698   Ep =-0.00733   M= 0.00509
Step: 20000   E =-0.00035   Ek = 0.00698   Ep =-0.00733   M= 0.00524
Step: 30000   E =-0.00035   Ek = 0.00699   Ep =-0.00734   M= 0.00538
Step: 40000   E =-0.00034   Ek = 0.00700   Ep =-0.00734   M= 0.00551
Step: 50000   E =-0.00034   Ek = 0.00701   Ep =-0.00734   M= 0.00563
Step: 60000   E =-0.00033   Ek = 0.00702   Ep =-0.00735   M= 0.00575
Step: 70000   E =-0.00032   Ek = 0.00703   Ep =-0.00736   M= 0.00588
Step: 80000   E =-0.00031   Ek = 0.00705   Ep =-0.00736   M= 0.00601
Step: 90000   E =-0.00030   Ek = 0.00707   Ep =-0.00737   M= 0.00615
Step: 100000   E =-0.00029   Ek = 0.00709   Ep =-0.00738   M= 0.00629
Time computations took:    1.18598s
Press <RETURN> to close this window...
```

We then calculated it with Jupiter ten times larger, this is visualized in figure 7. Here we have zoomed in so you can see the effect of Jupiter pull. And 8 shows the orbit of earth zoomed in when Jupiter is 1000 times larger. You see that Jupiters pull affect eart but not that much. Which is expected since the force of gravity decays with $\frac{1}{r^2}$. And jupiter is 5.2 AU distance from the sun. So even if the mass of Jupiter is almost the same as the Sun (as you can see in the figure 8), the force from Jupiter would be a lot less.

5.6 Final model for all planets of the solar system

Here we have build our final model; a system with all the celestial bodies included into the solar system. In figure figure 9 we observe the orbit of the sun and Earth. We see that the system drifts away as the system gets to large values of T. But the orbit of the earth is as expected. The reason why the system drifts away we think are because of round off error when the algorithm runs, the algortihm also takes alot of time as you see here:

5.7 Perihelion percession of Mercury

We were here asked to run a simulation over a timespan of one century of Mercury's orbit around the Sun, starting with Mercury at perihelion postion and with no other planets present in the system. We can find the value of perihelion angle θ_p using;

$$tan\theta_p = \frac{y_p}{x_p} \Rightarrow \theta_p = arctan\frac{y_p}{x_p} \tag{25}$$

where $x_p(y_p)$ is the $x(y)$ position of Mercury at perihelion, i.e. at the point closets to the Sun. Values for Perihelion speed of Mercury and its distance from the Sun are given by; $V_{peri} = 12.44[\frac{AU}{yr}]$ and $r = 0.3075AU$.

We didn't manage to get a working code of the perihelion problem so we can not make any discussions of our results.

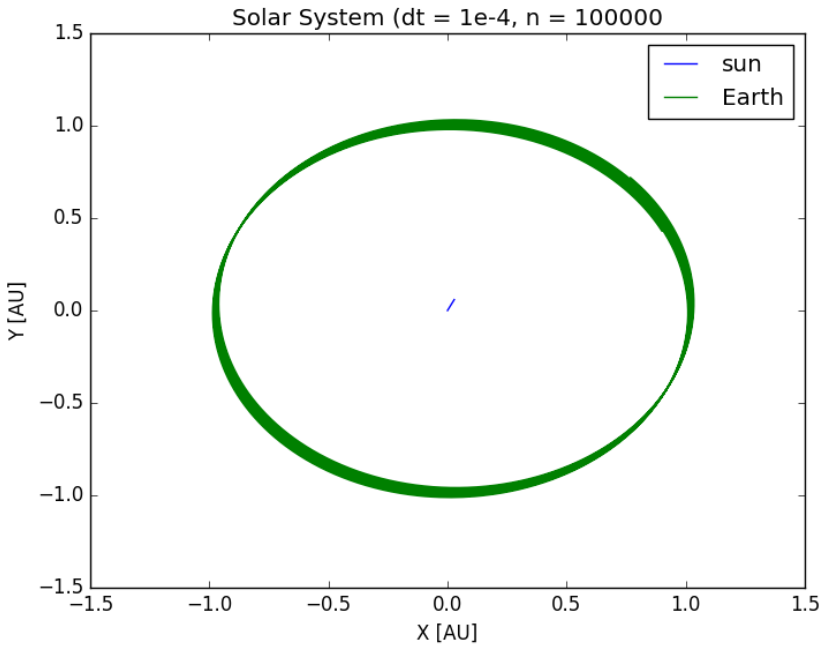

Figure 9: The orbit of the Sun and Earth with the forces from all of the planets in solar system (including Pluto

6 Conclusion

Symplectic algorithms such as Verlet algorithm have many advantages; they are not very complicated but have good accuracy and stability. The velocity Verlet algorithm provides both the objects positions and velocities at the same instant of time, and for this reason may be regarded as the most complete form of Verlet algorithm. For this reason Velocity Verlet plays a major role in molecular dynamic simulations. And we observed that it gave the correct orbit for the two body problem, thus supporting our argument that the Velocity Verlet is a lot more stable when calculating a circular orbit. It is also easy to implement as shown before. The only downside is the need to calculate the force twice in the algorithm, but it is a trade off of more stability versus computation time and the Velocity Verlet was a lot more accurate. We also obseverd that in the calculation of the entire solar system that the system drifts away. We think this comes from round of error and that we didn't have as accurate initial conditions that we could have. We also noticed that the calculations for the whole solar system took a long time, which is expected but to get a high precision it took too long time , so we had to take lesser precision.

So for calculations speed Euler is far superior but less accurate than Velocity Verlet.

Velocity Verlet will be using a little more memory because it have to calculate the force twice. But it won't use a lot more than Euler, since the values we save are ca. the same. And when we look at our code now we see that we probably could optimize the Velocity Verlet algorithm even more so that it would use less memory.

And we didn't manage to implement the perihelion code so we cant draw any conclusion directly from that, but we have tried to give an explanation in theory section.