Worksheet 1

Jens Jakschik, Fabio Oelschläger

November 16, 2020

Source code via Github
https://github.com/bliepp/Simulation-Methods-in-Physics-Exercises

1 Exercise 2 - Cannonball

1.1 Exercise 2.1 - Simulating a cannonball

Goal of this exercise was to simulate the motion of a cannonball with set x ([0,0]) and v([50,50]) start parameters until it hits the ground, i.e. reaches a y-value smaller than zero. For this, first the forces acting on the cannonball have to be calculated. Here, friction is still ignored, therefore the only force acting on the cannonball is gravity, which can easily be calculated with the following function:

```
1 def force(mass, gravity):
2     return np.array([0,-1*mass*gravity])
```

Here, mass stands for the mass of the cannonball, while gravity stands for the gravitational acceleration. The change of x and v over time in dependence of the force acting on the cannonball, is calculated using the Euler scheme. This is done using the following function:

```
1 def step_euler(x,v,dt,mass,gravity,f):
2     _x = x + v*dt
3     _v = v + f/mass*dt
4     return _x,_v
```

The Euler step is then repeated until the cannonball hits the ground. The result of this simulation is depicted in figure 1. As expected, the trajectory of the cannonball describes a parabolic shape.

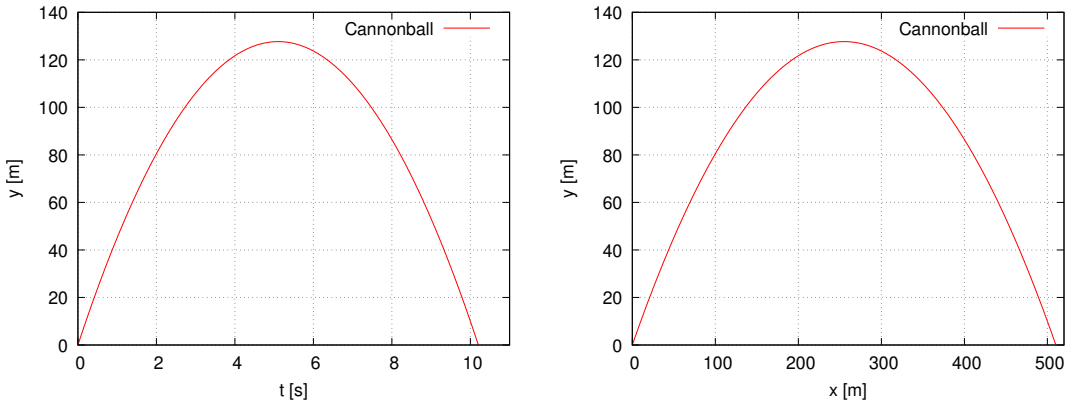

Figure 1: Height of the cannonball over time (left) and over distance (right).

1.2 Exercise 2.2 - Influence of friction and wind

Here, the simulation done in the previous section is expanded by including air resistance/friction and wind speed. For this, the function to calculate the forces acting on the ball is changed to:

```
1 def force(mass, gravity, v, gamma, v_0):
2     return ex_2_1.force(mass,gravity) - gamma*(v-v_0)
```

This results in the simulation depicted in figure 2. As expected, the distance the cannonball travels gets reduced when considering friction and wind.

In addition to this, it also should be determined how fast the wind has to be so the cannonball hits the ground where it initially started, at $x = 0$. Simulating the throw distance for different wind speeds and using a linear regression (figure 3), this is determined to be the case at $v_w = 195.99\,\mathrm{m\,s^{-1}}$.

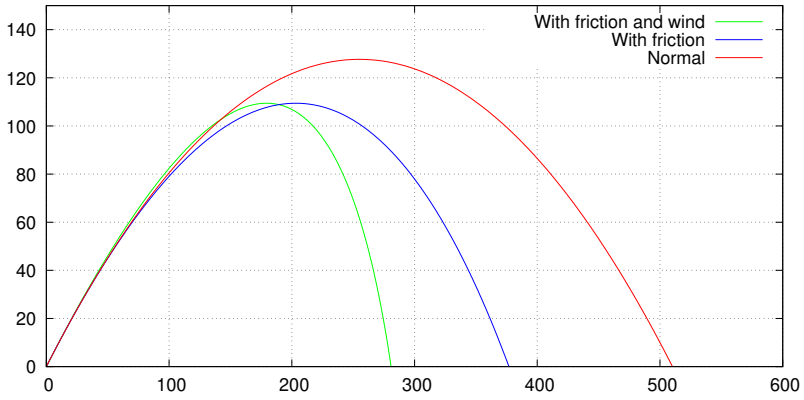

Figure 2: Trajectory of the cannonball when considering friction and air speed, only friction, or none of those.

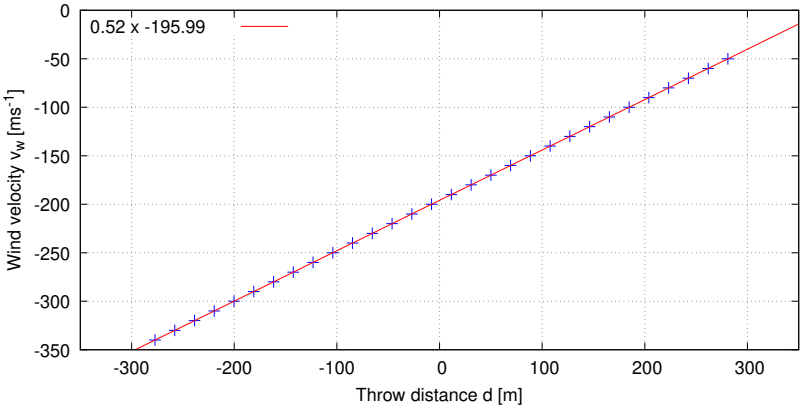

Figure 3: The end point of the cannonball's trajectory (throw distance) for different wind speeds.

2 Exercise 3 - Solar system

2.1 Exercise 3.1 - Simulating the solar system with the Euler scheme

Goal of this exercise is to simulate the planetary motion of the planets in our solar system. For this, the same Euler step function used in exercise 2.1 is used to calculate the next place and velocity of the different planets. Here, two important factors have to be considered. The first one is, that not only the sun enacts a gravitational pull on all the planets but also the planets act on each other. In addition to this Newtons third law also has to be considered. The total forces acting on each planet are calculated like in code 2.

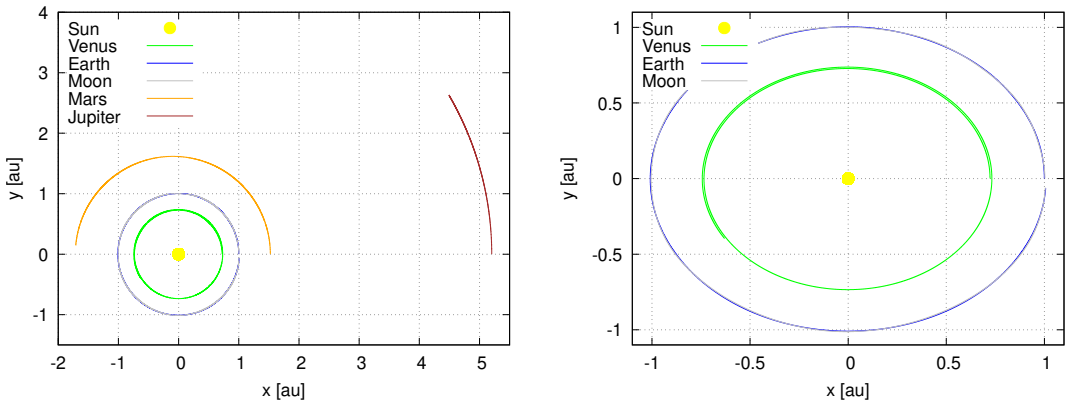

Figure 4: Results of the first solar system simulation

To determine the quality of this simulation, a closer look at the relation between the earth and the moon is taken. Initially, they seem to describe the same path, but this is only due to the large scale of the simulation. A closer look reveals that the moon orbits around the earth, as it should. This initial first look can be seen in figure 5, while the determined orbit, or rather the oscillation of the moon around earth can be seen in figure 6. This was obtained by determining the relative position of the moon to earth, thus mostly eliminating the effect the sun has on the moon.

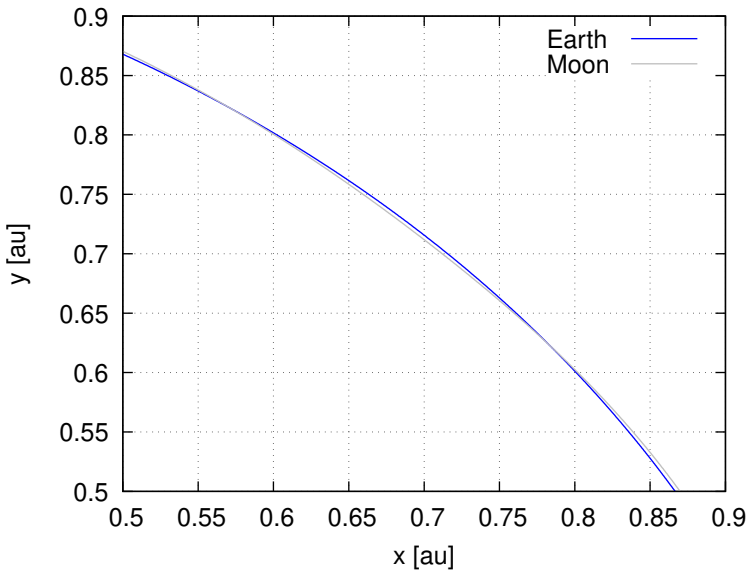

Figure 5: Closer look at the orbit of the earth and the moon around the sun

An interesting aspect of this plot is that the moon seems to increase its orbital radius (figure 6). Therefore the moon is going to get out of range of the gravitational potential of the other astronomical objects.

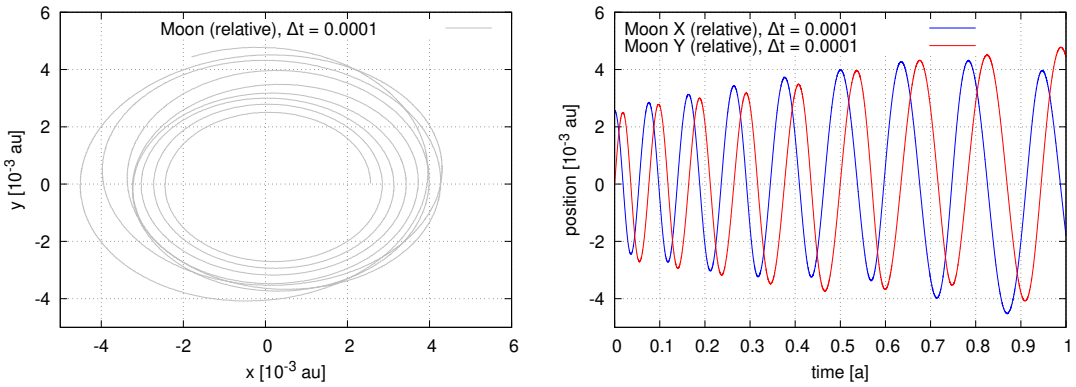

Figure 6: Relative orbit of the moon around earth and the oscillation of this relative position

Comparing it with data calculated with a bigger time step, which can be seen in figure 7, makes it obvious that this is because the used Euler algorithm is not symplectic.

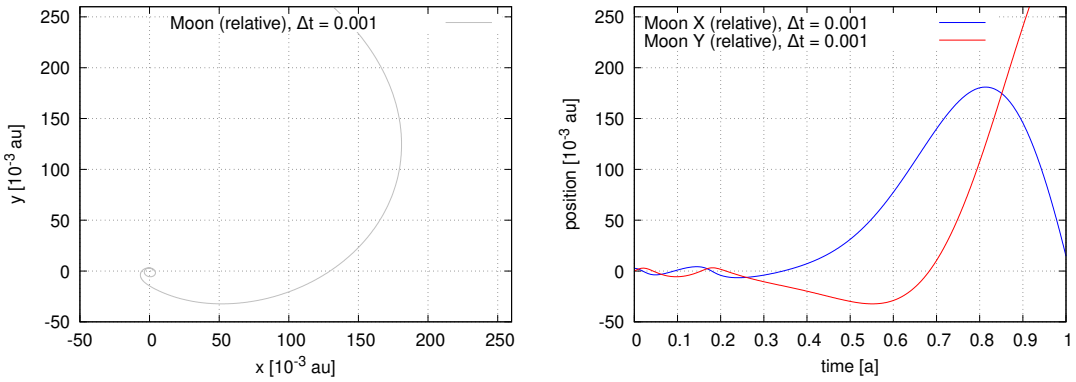

Figure 7: The moon escapes its relative orbit around earth if the time step is increased.

2.1.1 Computing time

When thinking about large scale particle simulations with a few million to billion particles the calculation will be the most expensive part. Since the program has to iterate over every other particle to get the force on one particle this part of the code has the worst scaling capability. An unoptimized algorithm like code 1 would take $N_c = N(N-1)$ ($\rightarrow N^2$ for large N) calculations considering N particles.

Using Newton's third law ($F_{ij} = -F_{ji}$) the number of calculations N_c needed can be reduced based on equation (1) like shown in code 2.

Code 1: Unoptimized force calculation (pseudocode)

```
def forces():
    for i=0; j<N; i++:
        for j=0; j<N; j++:
            if i != j:
                F[i] += force(i, j)
```

Code 2: Optimized force calculation (pseudocode)

```
def forces():
    for i=0; j<N-1; i++:
        for j=i+1; j<N; j++:
                F[i] += force(i, j)
                F[j] -= force(i, j)
```

$$N_\mathrm{c} = \sum_{n=1}^{N-1} n \overset{\text{Gauss}}{=} \frac{N(N-1)}{2} \to \frac{N^2}{2} \tag{1}$$

By writing the absolute values of the forces in the shape of a lookup table/matrix $M_{ij} = |F_{ij}|$ like in equation (2) it becomes obvious that $\underline{\underline{M}}$ is orthogonal. This is one way to demonstrate why the calculations are cutted in half.

$$\underline{\underline{M}} = \begin{pmatrix} 0 & |F_{12}| & |F_{13}| & \cdots & |F_{1N}| \\ |F_{21}| & 0 & |F_{23}| & & |F_{2N}| \\ |F_{31}| & |F_{32}| & \ddots & & |F_{3N}| \\ \vdots & & & 0 & \vdots \\ |F_{N1}| & |F_{N2}| & |F_{N3}| & \cdots & 0 \end{pmatrix} \tag{2}$$

Not only by cutting the number of calculations in half but also by getting rid of the branch caused by the if statement to check for itself the calculation time decreases drastically.

2.2 Exercise 3.2 - Integrators

2.2.1 Symplectic Euler algorithm

Although the Euler scheme is probably the simplest intergrator to solve inital value problems, it is not the best. Other alternatives are the symplectic Euler algorithm or the Verlet algorithm. The symplectic Euler algorithm is simply the Euler scheme with the order of operations switched, making it symplectic.

2.2.2 Velocity Verlet algorithm

To derive the Velocity Verlet algorithm, a Taylor expansion of the position and the velocity to the second order after an infinitesimal time step is used.

$$\underline{x}(t+\Delta t) = \underline{x}(t) + \underbrace{\dot{\underline{x}}(t)\Delta t}_{=\underline{v}(t)\Delta t} + \underbrace{\ddot{\underline{x}}(t)\frac{\Delta t^2}{2}}_{=\underline{a}(t)\frac{\Delta t^2}{2}} \tag{3}$$

$$\underline{v}(t+\Delta t) = \underline{v}(t) + \underbrace{\dot{\underline{v}}(t)\Delta t}_{=\underline{a}(t)\Delta t} + \underbrace{\ddot{\underline{v}}(t)\frac{\Delta t^2}{2}}_{=\dot{\underline{a}}(t)\frac{\Delta t^2}{2}} \tag{4}$$

To determine an expression for the $\dot{\underline{a}}(t)$ term, a Taylor expansion to the first order of $\underline{a}(t+\Delta t)$ is used.

$$\underline{a}(t+\Delta t) = \underline{a}(t) + \dot{\underline{a}}(t)\Delta t \qquad \Rightarrow \qquad \dot{\underline{a}}(t)\Delta t = \underline{a}(t+\Delta t) - \underline{a}(t) \tag{5}$$

This can then be combined with equation (4) to derive the velocity in the Velocity Verlet algorithm, as shown in equation (6).

$$\underline{v}(t+\Delta t) = \underline{v}(t) + \frac{\underline{a}(t) + \underline{a}(t+\Delta t)}{2}\Delta t \tag{6}$$

2.2.3 Verlet algorithm

The standard Verlet algorithm in equation (12) is equivalent to the Velocity Verlet algorithm and can be constructed by shifting equation (3) back by one step resulting in equation (7). On the other hand the same equation can be rearranged to separate $\underline{x}(t)$ as shown in equation (8).

$$\underline{x}(t) = \underline{x}(t-\Delta t) + \underline{v}(t-\Delta t)\Delta t + \frac{\underline{a}(t-\Delta t)}{2}\Delta t^2 \tag{7}$$

$$\underline{x}(t) = \underline{x}(t+\Delta t) - \underline{v}(t)\Delta t - \frac{\underline{a}(t)}{2}\Delta t^2 \tag{8}$$

Both resulting equations now have to be added together. This way equation (9) is constructed.

$$2\underline{x}(t) = \underline{x}(t-\Delta t) + \underline{x}(t+\Delta t) + [\underline{v}(t-\Delta t) - \underline{v}(t)]\Delta t + \frac{\underline{a}(t-\Delta t) - \underline{a}(t)}{2}\Delta t^2 \tag{9}$$

By reducing (6) by one step, too, a term for $\underline{v}(t-\Delta t)-\underline{v}(t)$ can be constructed. Inserting it into equation (9) gives the resulting equation (11), which is a rearranged form of (12).

$$\underline{v}(t-\Delta t) - \underline{v}(t) = -\frac{\underline{a}(t-\Delta t) + \underline{a}(t)}{2}\Delta t \tag{10}$$

$$2\underline{x}(t) = \underline{x}(t-\Delta t) + \underline{x}(t+\Delta t) - \underline{a}(t)\Delta t^2 \tag{11}$$

$$\underline{x}(t+\Delta t) = 2\underline{x}(t) - \underline{x}(t-\Delta t) + \underline{a}(t)\Delta t^2 \tag{12}$$

Another (some might say simpler) way to derive the standard Verlet algorithm is to combine two Taylor expansions of $\underline{x}$ to the third order in positive and negative direction, namely $\underline{x}(t+\Delta t)$ and $\underline{x}(t-\Delta t)$. The terms of uneven orders ($\underline{v}$ and $\dot{\underline{a}}$) are canceling each other out leaving just the acceleration and the position vectors while still being of $\mathcal{O}(\Delta t^4)$. Therefore the Verlet and the Velocity Verlet algorithm are of $\mathcal{O}(\Delta t^4)$, too.

2.2.4 Disadvantage of the Verlet algorithm

The disadvantage of the standard Verlet algorithm compared to the Velocity Verlet algorithm is that due to the term $\underline{x}(t - \Delta t)$ it is not self starting. This means, that a simple set of start parameters is not sufficient. Therefore, two start parameters of $\underline{x}$ (but no $\underline{v}$) are needed to be known to start the algorithm.

2.2.5 Implementation in the code

These two new possible algorithms should then be implemented into the existing simulation from exercise 3.1. First is the symplectic Euler, which is straightforward to implement, as merely the order of operations have to be switched, so that the velocity is calculated first.

```
1 def step_symplectic_euler(x, v, dt, mass, g, forces):
2     _v = v + forces(x, mass, g)*dt/mass
3     _x = x + _v*dt
4     return _x, _v
```

To implement the Velocity Verlet algorithm, equation (3) is used to calculate the new position, while equation (6) is used to calculate the new velocity. Here, an additional step compared to the other algorithms has to be included, as the equation for the velocity includes the forces acting on the planets before and after the position step.

```
1 def step_velocity_verlet(x, v, dt, mass, g, forces):
2     f = forces(x, mass, g) # may be reused from previous iteration
3     _x = x + v*dt + 0.5*f*dt*dt/mass
4     _v = v + 0.5*f*dt/mass
5     # recalculate the acceleration
6     f = forces(_x, mass, g) # to reuse save globally or return
7     _v += 0.5*f*dt/mass
8     return _x, _v
```

To test these algorithms, a simulation over one year with a time step of $\Delta t = 0.01$ is done and the trajectory of the moon relative to earth is plotted. The results of these simulations can be seen in the following figure.

Due to its coarse time step it is obvious that the symplectic Euler algorithm has more fluctuations than the Velocity Verlet algorithm. In figure 8 this can be seen by the width

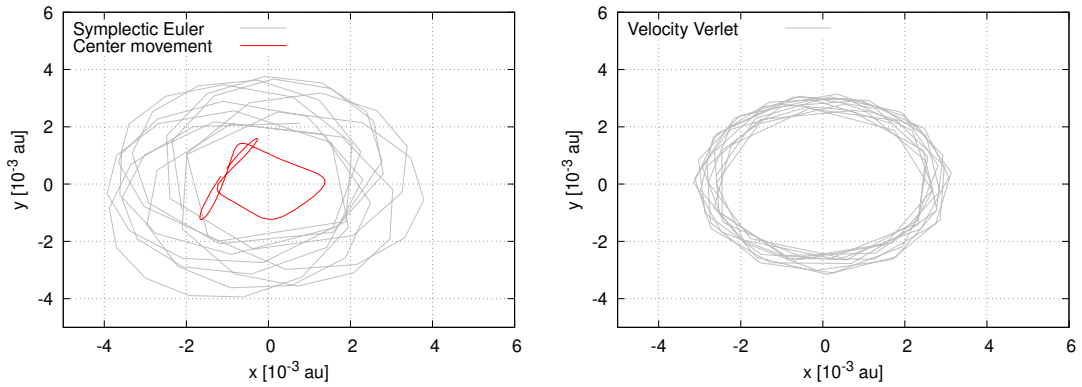

Figure 8: Relative position of the moon with respect to earth calculated with two different algorithms.

change of the orbit. Besides the moon's almost perfect circlular movement there's an additional movement of the center point. While this is barely noticable in the Velocity Verlet algorithm it has a big influence on the results of the symplectic Euler algorithm.

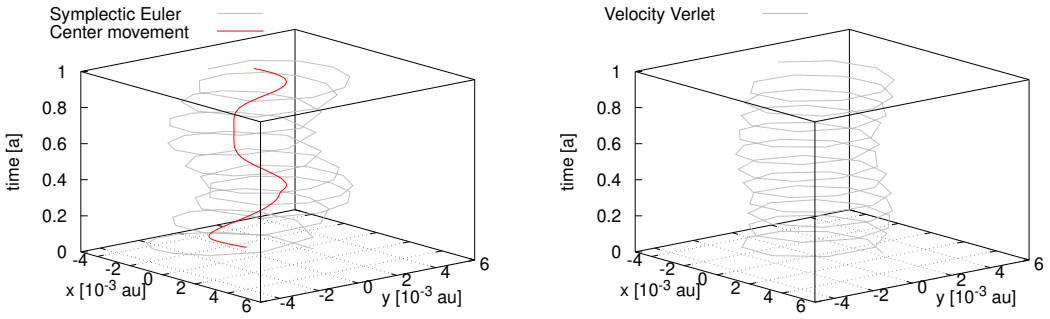

Figure 9: The same trajectory as in fig. 8 but over time.

Having a look of the trajectory over time (figure 9) it can be determined that the orbit radius does not change while only the center moves.

Calculating the center point movement is pretty simple. As seen figure 10 the signals can be identified as almost periodic signals. Applying a low pass filter via a Fourier transform returns a signal without the high frequency of the moon "oscillating" around the earth. The result is the low frequency movement of the moon which is the approximately the movement of the orbits center point.

The symplectic Euler scheme might not be a to good replacement for non-symplectic

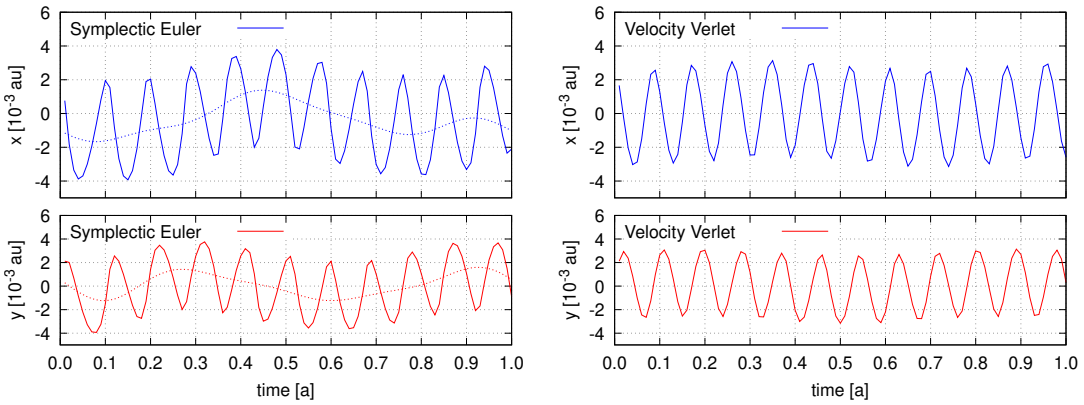

Figure 10: Coordinate over time for different algorithms. The center point movement was isolated using a Fourier transform.

algortihms such as the Euler scheme due to its high fluctuation. Simulating things like planets may be okay since the the fluctuation can be filtered like shown before, but not all physical problems may have fluctuation on a different scale than the expected signal. High precision symplectic algorithms such as the Velocity Verlet on the other hand algorithm might be the better choice in most cases because the fluctuations are barely noticable.

2.3 Exercise 3.3 - Long term stability

To test the long term stability of the different algorithms tested in section 3.2, the simulation performed there was repeated for a time period of ten years. A good measure of the stability is the distance between the earth and moon, as this is one of the most sensitive parts of this simulation. There it can be seen, that the velocity Verlet alogirithm remains by far the most stable throughout the duration of 10 years. The symplectic Euler algorithm starts loosing most of its precision after about 6 years. This showcases the importance of choosing the right algorithm, as the Euler step and the symplectic Euler were easier to implement and the symplectic Euler resulted in almost identical results compared to the velocity Verlet algorithm for short simulations or smaller time steps. Only in this case, where a long simulation with a broad time step was performed, the differences became visible.

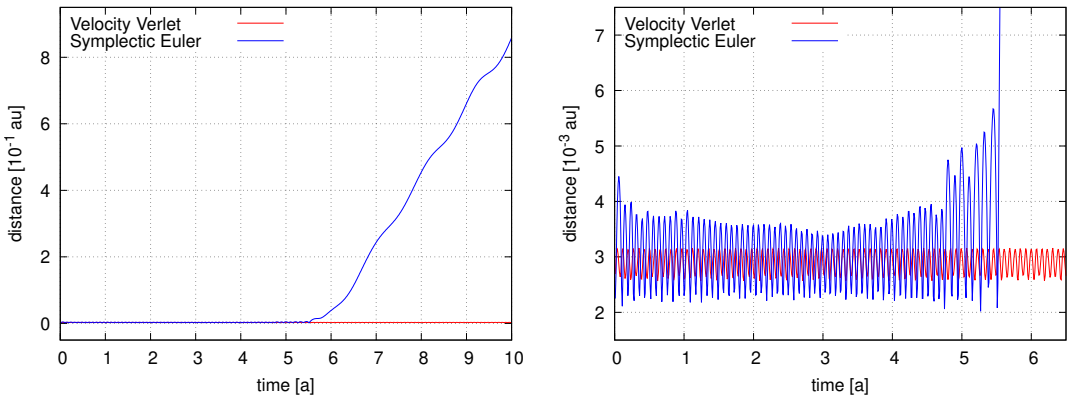

Figure 11: Distance of the moon from the earth for different algorithms.