Worksheet 01

Lena Weber, Niklas Abraham, Cedric Förch

November 9, 2024

Institute for Computational Physics, University of Stuttgart

Contents

1 Cannonball

1.1 Simulating a cannonball

The simulation uses the Euler algorithm with time steps of $\Delta t = 0.1\,\mathrm{s}$ from $t = 0$ until the cannonball hits the ground again. The force on the cannonball is $\mathbf{F}(t) = (0, -mg)$ with the acceleration $g = 9.81\,\mathrm{m/s^2}$ and the mass m. The initial conditions for position and velocity at time $t = 0$ are $\mathbf{x}(0) = \mathbf{0}$ and $\mathbf{v}(0) = (60, 60)^\intercal\ \frac{\mathrm{m}}{\mathrm{s}}$.
The simulation result can be found in Figure 1. The trajectory does not depend on the mass of the cannonball.
The code for one time step of the Euler algorithm is as follows:

```
def step_euler(x, v, dt, mass, gravity, f):
    #position update
    x += v*dt
    #velocity update
    v += f*dt/mass

    return x, v
```

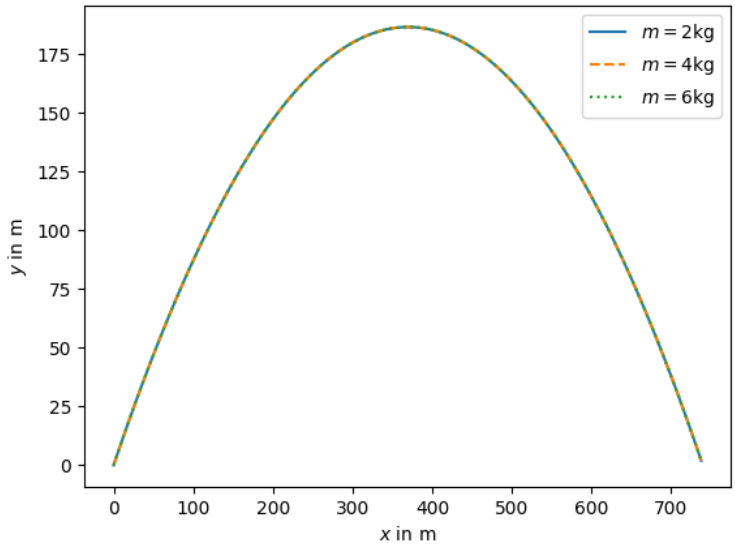
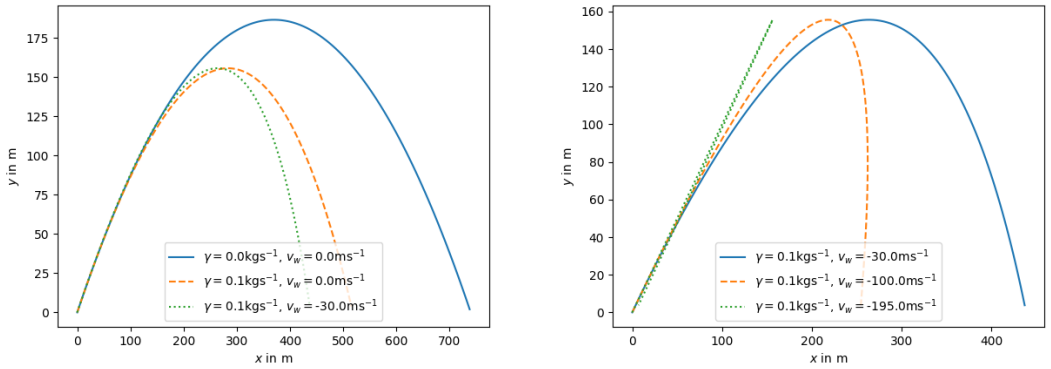


Figure 1: Simulated trajectory of the cannonball for different masses m.

1.2 Influence of friction and wind

The friction is introduced as the force $F_{\text{fric}}(\mathbf{v}) = -\gamma(\mathbf{v} - \mathbf{v}_0)$ with the friction coefficient $\gamma = 0.1\,\text{kg/s}$ and the wind speed $\mathbf{v}_0 = (0, v_w)^\intercal\ \frac{\text{m}}{\text{s}}$. This friction model a linear drag model (linearly dependent on the velocity) which also considers wind. The results seem mostly realistic, but this model is only an approximation which does not contain the shape of the cannonball for example.
The mass of the cannonball is $m = 2.0\,\text{kg}$.
The simulation result can be found in Figure 2. The maximum height the cannonball reaches is lower in the case with friction compared to without. The wind influences the shape of the trajectory, it no longer has the shape of a parabola, but does not affect the height. For a wind speed of $v_w = -195.0\ \frac{\text{m}}{\text{s}}$ the cannonball hits the ground close to the initial position.
The friction model is included into the code as follows:

```
def force(mass, gravity, v, gamma, v_0):
    #calculate gravitational force and drag
    return ex_2_1.force(mass, gravity) - gamma*(v - v_0)
```



(a) (b)

Figure 2: Simulated trajectory of the cannonball for different friction coefficients γ and wind velocities v_w.

2 Solar system

2.1 Simulating the solar system with Euler scheme

The trajectories of the whole solar system, simulated using the Euler algorithm, can be seen in Figure 3. The simulation is for a whole year, with time steps of $\Delta t = 0.0001$ a.

The trajectory of the moon in the rest frame of the earth is simulated and visualized in Figure 4 for different time steps Δt. For larger time steps the trajectory leaves the orbit faster and the simulation therefore is more unstable and inaccurate. This is because the errors of the Euler algorithm accumulate with each time step and are larger for larger time steps.
The total force calculation is implemented as follows:

```
def forces(x, masses, g):
    Fs = np.zeros((2,6))
    #calculate all forces, in both directions (i -> j and j -> i)
    for i in range(N):
        for j in range(i+1,N):
            Fs[:,i] += force(x[:,i]-x[:,j], masses[i], masses[j], g)
            Fs[:,j] += -force(x[:,i]-x[:,j], masses[i], masses[j], g)
    return Fs
```

2.2 Integrators

Symplectic Integrators are for example the Verlet algorithm and the Velocity Verlet algorithm. They are mathematically the same. We start with the Taylor expansion of

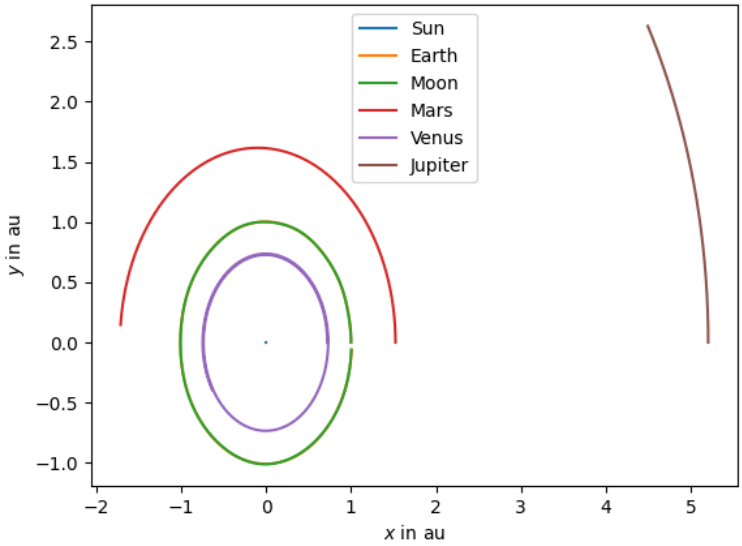
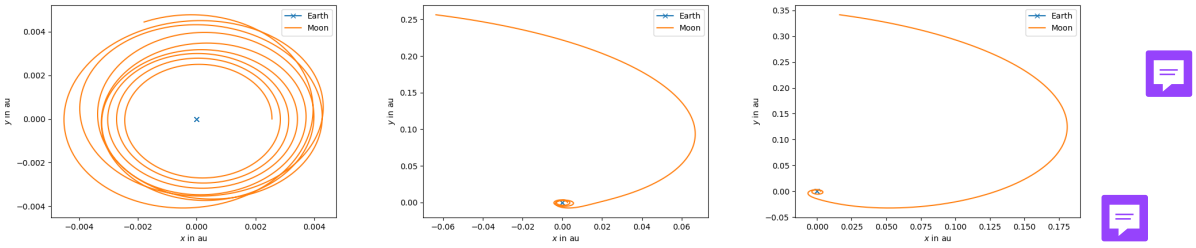


Figure 3: Simulated trajectory of the planets of the solar system.



(a) $\Delta t = 0.0001\,\mathrm{a}$. (b) $\Delta t = 0.0005\,\mathrm{a}$. (c) $\Delta t = 0.001\,\mathrm{a}$.

Figure 4: Simulated trajectory of the moon in the rest frame of the earth for different time steps Δt.

$\mathbf{x}(t + \Delta t)$

$$\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + \mathbf{v}(t)\Delta t + \frac{1}{2}\mathbf{a}(t)\Delta t^2 + O(\Delta t^3). \tag{1}$$

That is the position update of the Velocity Verlet algorithm. For the Velocity Verlet algorithm we need to expand $\mathbf{v}(t + \Delta t)$

$$\mathbf{v}(t + \Delta t) = \mathbf{v}(t) + \mathbf{a}(t)\Delta t + \frac{1}{2}\frac{d^2\mathbf{v}(t)}{dt^2}\Delta t^2 + O(\Delta t^3). \tag{2}$$

We now need to expand $\frac{d\mathbf{v}(t+\Delta t)}{dt}$ to the first order

$$\frac{d\mathbf{v}(t + \Delta t)}{dt} = \frac{d\mathbf{v}(t)}{dt} + \frac{d^2\mathbf{v}(t)}{dt^2}\Delta t + O(\Delta t^2) \tag{3}$$

$$\leftrightarrow \frac{d^2\mathbf{v}(t)}{dt^2} = \frac{d}{dt}\mathbf{a}(t) = \frac{\mathbf{a}(t + \Delta t) - \mathbf{a}(t)}{\Delta t} + O(\Delta t). \tag{4}$$

Combined this results in the Velocity Verlet algorithm

$$\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + \mathbf{v}(t)\Delta t + \frac{1}{2}\mathbf{a}(t)\Delta t^2 \tag{5}$$

$$\mathbf{v}(t + \Delta t) = \mathbf{v}(t) + \frac{1}{2}(\mathbf{a}(t) + \mathbf{a}(t + \Delta t)\Delta t. \tag{6}$$

For the second task we prove that the Velocity Verlet algorithm is equivalent to the Verlet algorithm. First we make a Taylor expansion of $\mathbf{x}(t + 2\Delta t)$, which results in **??** with a shift of Δt

$$\mathbf{x}(t + 2\Delta t) = \mathbf{x}(t + \Delta t) + \mathbf{v}(t + \Delta t)\Delta t + \frac{1}{2}\mathbf{a}(t + \Delta t)\Delta t^2. \tag{7}$$

From Equation 5 we can rearrange

$$\mathbf{x}(t) = \mathbf{x}(t + \Delta t) - \mathbf{v}(t)\Delta t - \frac{1}{2}\mathbf{a}(t)\Delta t^2. \tag{8}$$

Add Equation 7 and Equation 8 to get

$$\begin{aligned}\mathbf{x}(t + 2\Delta t) + \mathbf{x}(t) = 2\mathbf{x}(t + \Delta t) + \mathbf{v}(t + \Delta t)\Delta t + \frac{1}{2}\mathbf{a}(t + \Delta t)\Delta t^2 \\ -\mathbf{v}(t)\Delta t - \frac{1}{2}\mathbf{a}(t)\Delta t^2.\end{aligned} \tag{9}$$

Rearrange Equation 6 to

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

Replace the velocity terms in Equation 9 with Equation 10 to obtain

$$\mathbf{x}(t + 2\Delta t) + \mathbf{x}(t) = 2\mathbf{x}(t + \Delta t) + \frac{\mathbf{a}(t) + \mathbf{a}(t + \Delta t)}{2}\Delta t^2 + \frac{1}{2}\mathbf{a}(t + \Delta t)\Delta t^2 - \frac{1}{2}\mathbf{a}(t)\Delta t^2 \tag{11}$$

$$\Leftrightarrow \mathbf{x}(t + 2\Delta t) + \mathbf{x}(t) = 2\mathbf{x}(t + \Delta t) + \mathbf{a}(t + \Delta t)\Delta t^2. \tag{12}$$

We now shift Equation 12 by $-\Delta t$

$$\mathbf{x}(t+\Delta t) + \mathbf{x}(t-\Delta t) = 2\mathbf{x}(t) + \mathbf{a}(t)\Delta t^2 \tag{13}$$

$$\Leftrightarrow \mathbf{x}(t+\Delta t) = 2\mathbf{x}(t) - \mathbf{x}(t-\Delta t) + \mathbf{a}(t)\Delta t^2. \tag{14}$$

The Verlet algorithm is given by

$$\mathbf{x}(t+\Delta t) = 2\mathbf{x}(t) - \mathbf{x}(t-\Delta t) + \mathbf{a}(t)\Delta t^2 + \mathcal{O}(\Delta t^4). \tag{15}$$

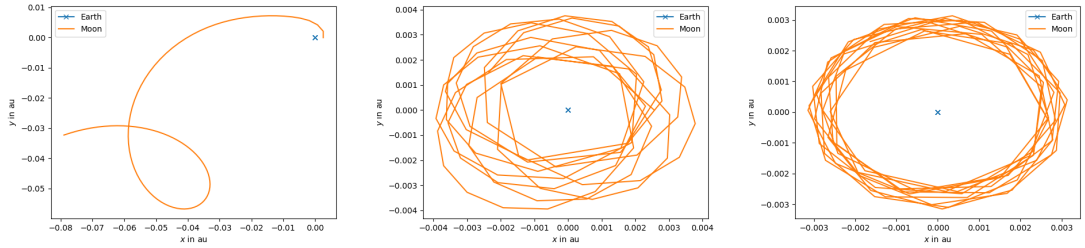
It requires information of the previous step at time $t - \Delta t$ in order to calculate the next step at $t + \Delta t$. The first time step can not be calculated like this, if only information for the initial conditions at $t = 0$ are given.

Figure 5 shows the simulation of the moon's trajectory in the rest frame of earth using different integrators. The Euler algorithm is the least accurate, the moon almost immediately leaves the orbit around earth. For both the symplectic Euler and Velocity Verlet algorithm an orbit is visible, although it is more precise for the Velocity Verlet algorithm.
The symplectic Euler and Velocity Verlet algorithms are implemented as follows:

```
def step_symplectic_euler(x, v, dt, mass, g):
    #current forces on all particles
    f = ex_3_1.forces(x, mass, g)
    #velocity update first
    v += f*dt/mass
    #position update last
    x += v*dt

    return x, v

def step_velocity_verlet(x, v, dt, mass, g, force_old):
    #old accelerations
    a = force_old/mass
    #position update
    x += v*dt + a*dt**2/2
    #first half of velocity update
    v += a*dt/2
    #new accelerations
    a = ex_3_1.forces(x, mass, g)/mass
    #second half of velocity update
    v += a*dt/2
    force_old = a*mass

    return x, v, force_old
```

2.3 Long-term stability

The simulation is run for 20 years with $\Delta t = 0.01\,\mathrm{a}$ with each integrator. The distance between earth and the moon in seen in Figure 6. The Euler algorithm is the most unstable, since the errors add up with every time step. The symplectic Euler algorithm



(a) Euler. (b) Symplectic Euler. (c) Velocity Verlet.

Figure 5: Simulated trajectory of the moon in the rest frame of the earth for different integrator algorithms.

is stable for almost 6 years, but then the moon leaves the orbit as well. The Velocity Verlet algorithm remains stable for the whole simulated time frame. There are no odd-degree terms in the algorithm, this symmetry leads to reduced errors.

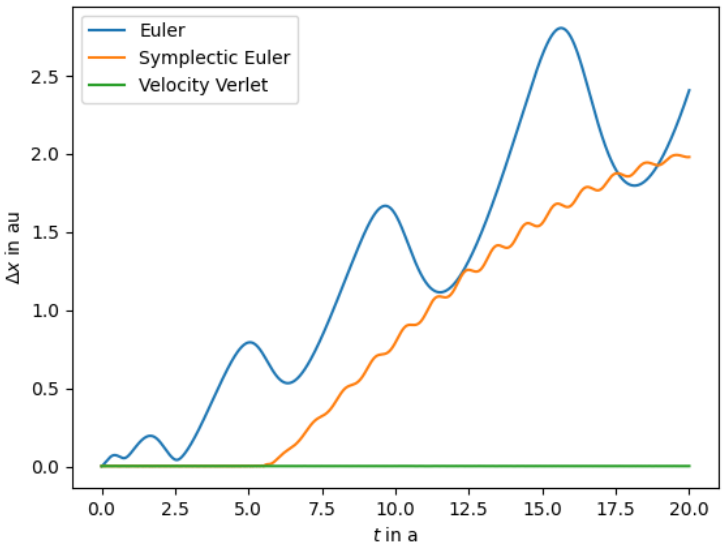


Figure 6: Distance between the moon and the earth in a long-term simulation for different algorithms.