Exercise No. 2

David Bubeck, Pascal Becht, Patrick Nisblè

May 4, 2017

2 - Error analysis of the Euler scheme

a)

At first we write the code for the 2 - body problem by using the forward Euler algorithm. We integrate the problem for one orbit and plot it on a double - logarithmic scale. The error will be plotted as a function of Δt. Therefore we will be using three different eccentricities and various different time steps.

Function that integrates the 2 - body problem using the forward Euler algorithm.

Listing 1: Exercise02a.py

```
import numpy as np
import matplotlib.pyplot as plt
plt.figure(figsize=(8,6))

#function that integrates the 2-body-problem using the forward
#euler algorithm. The initial velocity needs to be two dimensional
#vector, the step size h can be choosen. The program breaks when the
#orbit is complete.
def euler(v_0, h, s_0 = np.array([0, 1])):
    S = [s_0]    #list for spatial coordinate
    V = [v_0]    #list for relative velocity
    E = []       #list for the Energy
    test = 0     #variable to check if the orbit is complete
    step = 0
    maxstep = 1000000

    while((test < 2) and (step < maxstep)):
        S.append(S[-1] + h * V[-1])
        V.append(V[-1] - h * S[-1] / (np.sqrt(S[-2][0]**2 + S[-2][1]**2)**3))
        E.append((0.5 * (V[-1]**2)[0] + (V[-1]**2)[1]) - (1/((S[-1]**2)[0] + (S
            [-1]**2)[1])**0.5))

        if((S[-1])[0] < 0):
            test = 1
        if(test == 1 and (S[-1])[0] > 0):
            test = 2

    return np.array(S), np.array(V), np.array(E)
```

Now we plot a circular orbit with the error in energy in a additional plot. The values that were used are given in the code and We repeat the calculation for two more different initial velocities.

Listing 2: Exercise02a.py

```
#for the given initial values the eccentricity is zero,
#therefore the orbit is circular
orbits = [1, 0],[1.3,0],[.8,0]

for o in orbits:
    S, V, E = euler(np.array(o), 0.0001)
    plt.plot(S[:, 0], S[:, 1], label="$v_{init}$=" + "{}".format(o[0]))

plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.axis('equal')
plt.savefig('fig_a1.png')
plt.clf()
# plt.cla()
# plt.close()

for o in orbits:
    energyerror = []
    H = [0.01, 0.001, 0.0001, 0.0001, 0.00001]
    for h in H:
        S, V, E = euler(np.array(o), h)
        energyerror.append((abs(E[-1] - E[0])) / (abs(E[0])))

    plt.plot(H, energyerror, label="$v_{init}$=" + "{}".format(o[0]))

plt.xscale('log')
plt.yscale('log')
plt.xlabel('Stepsize h')
plt.ylabel('error in energy')
plt.legend()
plt.savefig('fig_a2.png')
```

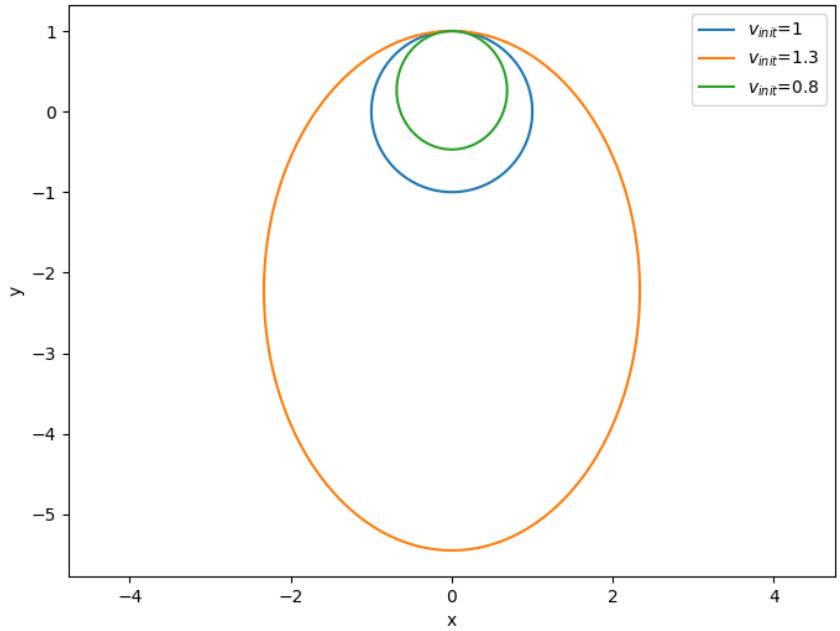

Figure 1: First orbit with forward Euler algorithm

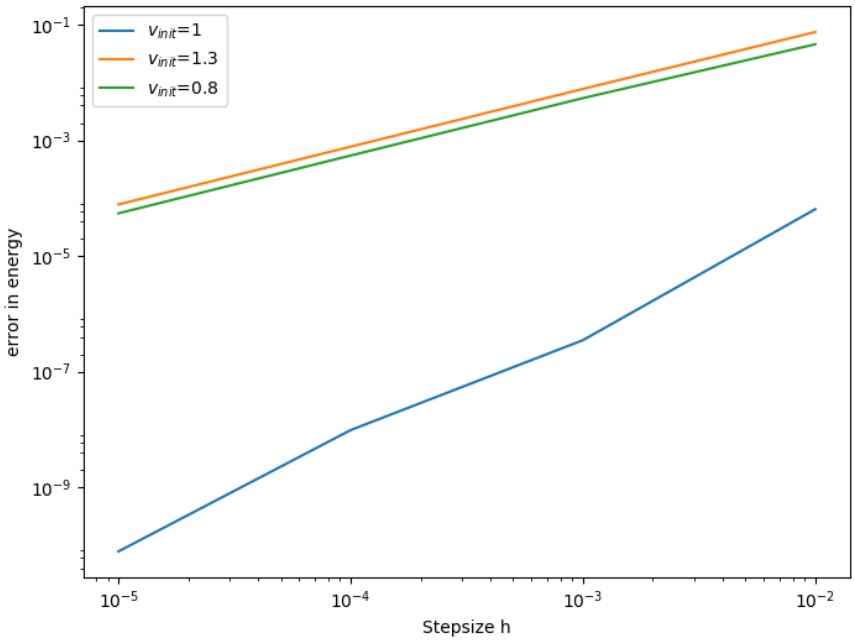

Figure 2: First orbit energy error with forward Euler algorithm

b)

Now we write the code for the 2 - body problem by using the Leapfrog algorithm. We integrate the problem for one orbit and plot it on a double - logarithmic scale. The error will be plotted as a function of Δt. Therefore we will be using three different eccentricities and various different time steps.

Function that integrates the 2 - body problem using the Leapfrog algorithm.

Listing 3: Exercise02b.py

```
import numpy as np
import matplotlib.pyplot as plt
plt.figure(figsize=(8,6))


#function that integrates the 2 - body - problem using the leapfrog
#algorithm. The initial velocity needs to be two dimensional
#vector, the step size h can be choosen. The program breaks when the
#orbit is complete.
def leapfrog(v_0, h, s_0 = np.array([0, 1])):
    S = [s_0]    #list for spatial coordinate
    V = [v_0]    #list for relative velocity
    E = []       #list for the Energy
    #first value for the acceleration
    a = [-1*(np.array([(S[-1])[0], (S[-1])[1]]))/(((S[-1]**2)[0] + (S[-1]**2)[1])
        **1.5)]
    test = 0     #variable to check if the orbit is complete
    step = 0
    maxstep = 1000000

    while((test < 2) and (step < maxstep)):
        vhalf = V[-1] + 0.5*h*a[-1]
        S.append(S[-1] + h*vhalf)
        a.append(-1*(np.array([(S[-1])[0], (S[-1])[1]]))/(((S[-1]**2)[0] + (S
            [-1]**2)[1])**1.5))
        V.append(vhalf + 0.5*h*a[-1])
        E.append((0.5*(V[-1]**2)[0] + (V[-1]**2)[1]) - (1/((S[-1]**2)[0] + (S
            [-1]**2)[1])**0.5))

        if ((S[-1])[0] < 0):
            test = 1
        if (test == 1 and (S[-1])[0] > 0):
            test = 2
```

Now we plot a circular orbit with the error in energy in a additional plot. The values that were used are given in the code. To be precise we going to use the same values as before and we also repeat the calculation for two more different initial velocities.

Listing 4: Exercise02b.py

```
34
35 #for the given initial values the eccentricity is zero,
36 #therefore the orbit is circular
37 orbits = [1, 0],[1.3,0],[.8,0]
38
39 for o in orbits:
40     S, V, E = leapfrog(np.array(o), 0.0001)
41     plt.plot(S[:, 0], S[:, 1], label="$v_{init}$=" + "{}".format(o[0]))
42
43 plt.xlabel('x')
44 plt.ylabel('y')
45 plt.legend()
46 plt.axis('equal')
47 plt.savefig('fig_b1.png')
48 plt.clf()
49 # plt.cla()
50 # plt.close()
51
52 for o in orbits:
53     energyerror = []
54     H = [0.01, 0.001, 0.0001, 0.0001, 0.00001]
55     for h in H:
56         S, V, E = leapfrog(np.array(o), h)
57         energyerror.append((abs(E[-1] - E[0])) / (abs(E[0])))
58
59     plt.plot(H, energyerror, label="$v_{init}$=" + "{}".format(o[0]))
60
61 plt.xscale('log')
62 plt.yscale('log')
63 plt.xlabel('Stepsize_h')
64 plt.ylabel('error_in_energy')
65 plt.legend()
66 plt.savefig('fig_b2.png')
```

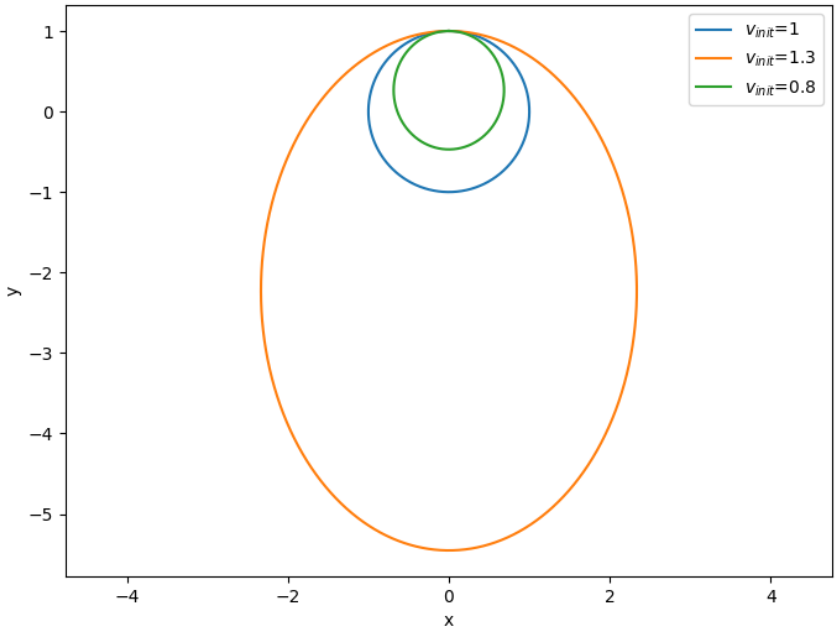

Figure 3: First orbit with leapfrog algorithm

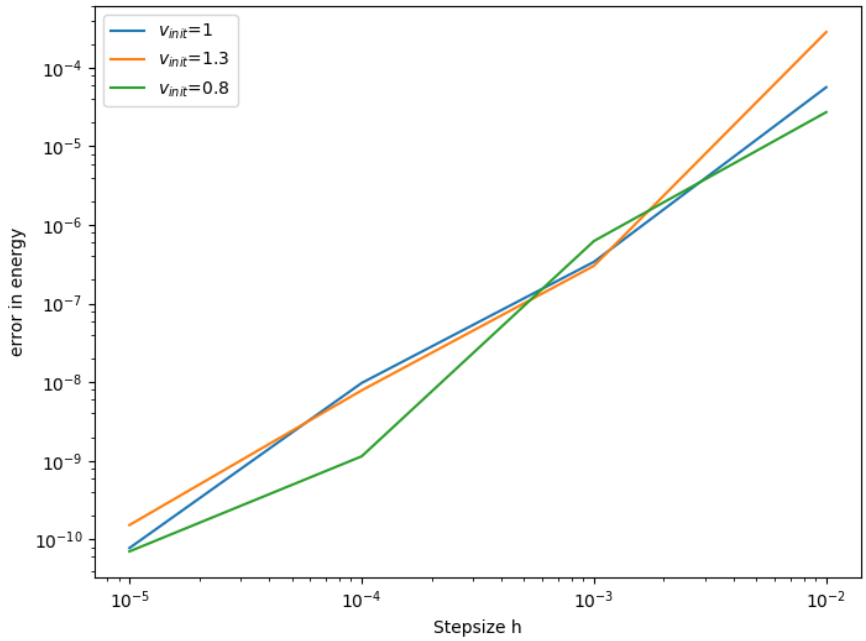

Figure 4: First orbit energy error with leapfrog algorithm

The errors decrease with decreasing time steps for both integration schemes. For the Euler algorithm we expect a drop with $O(h^2)$, for the Leapfrog algorithm $O(h^3)$. This is also what we can observe. The errors of the Leapfrog integration are much smaller than those of the Euler integration. However, the slope of the error-curve is steeper which we also expected. Another difference between the error-plots, of the two integration variants is, that for the Euler integration we get straight lines whereas for the Leapfrog integration the errors seem to scatter around the expected line.