

Exercise No. 2

David Bubeck, Pascal Becht, Patrick Nisblè

May 3, 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

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4
5 #function that integrates the 2 - body - problem using the forward
6 #euler algorithm. The initial velocity needs to be two dimensional
7 #vector, the step size h can be choosen. The program breaks when the
8 #orbit is complete.
9 def euler(v_0, h, s_0 = np.array([0, 1])):
10     S = [s_0] #list for spatial coordinate
11     V = [v_0] #list for relative velocity
12     E = [] #list for the Energy
13     test = 0 #variable to check if the orbit is complete
14     step = 0
15     maxstep = 1000000
16
17     while((test < 2) and (step < maxstep)):
18         S.append(S[-1] + h * V[-1])
19         V.append(V[-1] - h * S[-1] / (np.sqrt(S[-2][0]**2 + S[-2][1]**2)**3))
20         E.append((0.5 * (V[-1]**2)[0] + (V[-1]**2)[1]) - (1/((S[-1]**2)[0] + (S
21             [-1]**2)[1])**0.5))
22
23         if((S[-1][0] < 0):
24             test = 1
25         if(test == 1 and (S[-1][0] > 0):
26             test = 2
27
28     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.

Listing 2: Exercise02a.py

```

30 #for the given initial values the eccentricity is zero,
31 #therefore the orbit is circular
32 S, V, E = euler(np.array([1, 0]), 0.0001)
33
34 plt.figure(figsize = (6.5, 10))
35 plt.plot(S[:, 0], S[:, 1])
36 plt.xlabel('x')
37 plt.ylabel('y')
38 plt.show()
39
40 energyerror = []
41 H = [0.01, 0.001, 0.0001, 0.0001, 0.00001]
42 for h in H:
43     S, V, E = euler(np.array([1, 0]), h)
44     energyerror.append((abs(E[-1] - E[0])) / (abs(E[0])))
45
46 plt.plot(H, energyerror)
47 plt.xscale('log')
48 plt.yscale('log')
49 plt.xlabel('Stepsize_h')
50 plt.ylabel('error_in_energy')
51 plt.show()

```

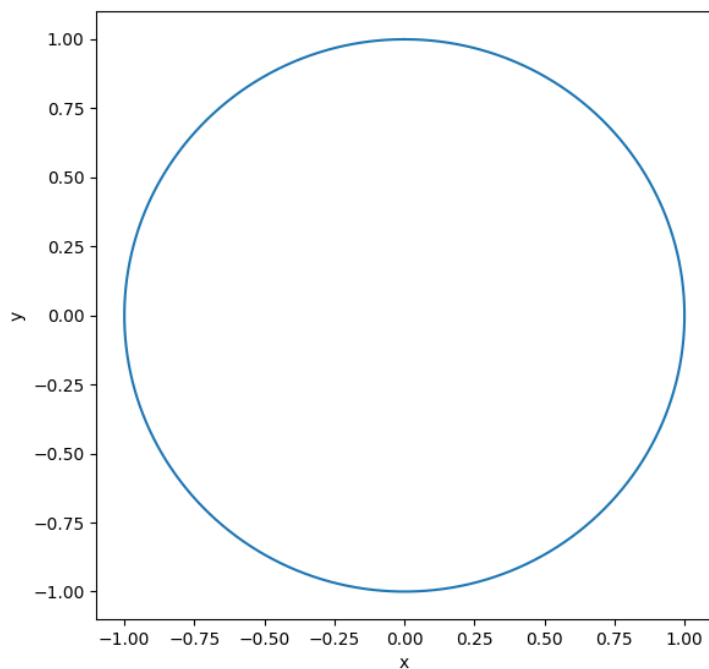


Figure 1: First orbit with forward Euler algorithm

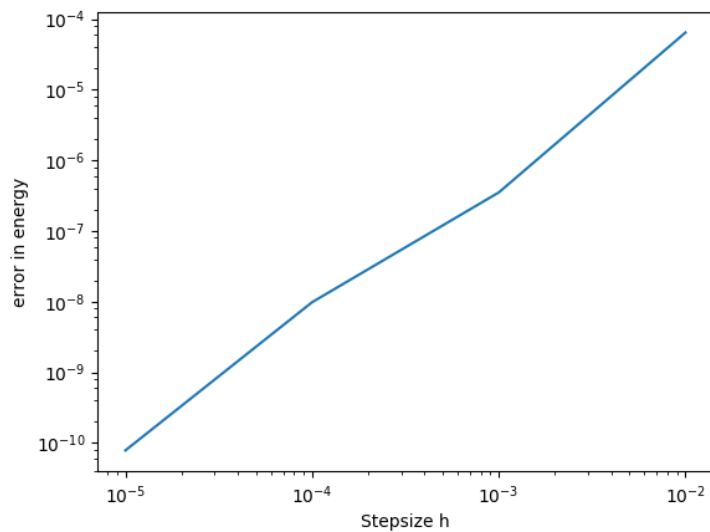


Figure 2: First orbit energy error with forward Euler algorithm

We repeat the calculation for two more different initial velocities.
For the second orbit we chose:

Listing 3: Exercise02a.py

```

54 #use different initial velocity to achieve another orbit
55 S, V, E = euler(np.array([1.3, 0]), 0.0001)
56
57 plt.figure(figsize = (6.5, 10))
58 plt.plot(S[:, 0], S[:, 1])
59 plt.xlabel('x')
60 plt.ylabel('y')
61 plt.show()
62
63 energyerror = []
64 H = [0.01, 0.001, 0.0001, 0.0001, 0.00001]
65 for h in H:
66     S, V, E = euler(np.array([1.3, 0]), h)
67     energyerror.append((abs(E[-1] - E[0])) / (abs(E[0])))
68
69 plt.plot(H, energyerror)
70 plt.xscale('log')
71 plt.yscale('log')
72 plt.xlabel('stepsize_h')
73 plt.ylabel('error_in_energy')
74 plt.show()

```

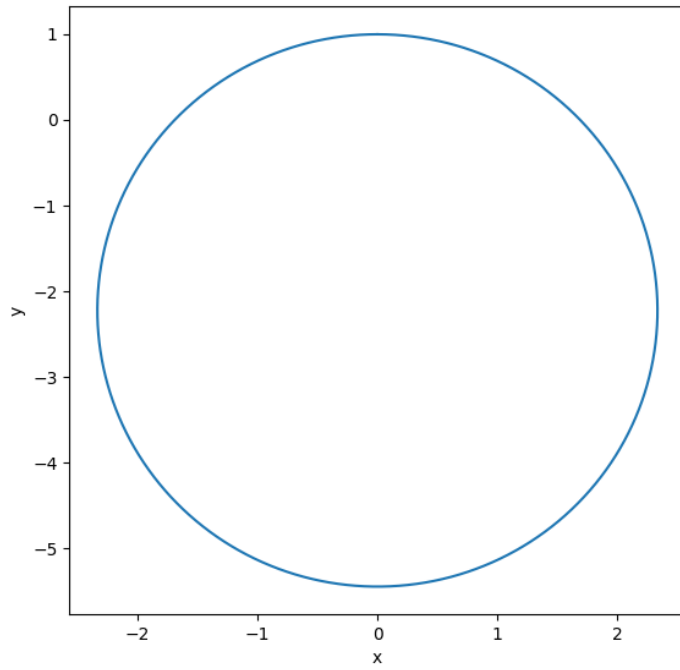


Figure 3: Second orbit with forward Euler algorithm

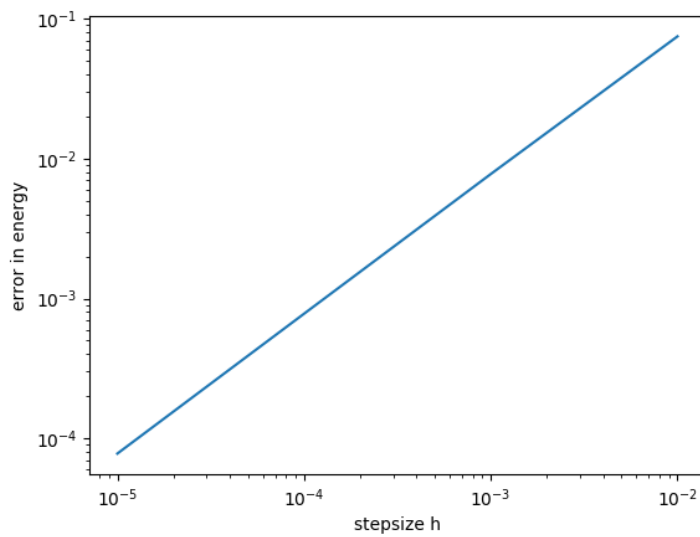


Figure 4: Second orbit energy error with forward Euler algorithm

For the third orbit we chose:

Listing 4: Exercise02a.py

```

77 #do the same for the third orbit with again another initial velocity
78 S, V, E = euler(np.array([0.8, 0]), 0.001)
79
80 plt.figure(figsize = (6.5, 10))
81 plt.plot(S[:, 0], S[:, 1])
82 plt.xlabel('x')
83 plt.ylabel('y')
84 plt.show()
85
86 energyerror = []
87 H = [0.01, 0.001, 0.0001, 0.0001, 0.00001]
88 for h in H:
89     S, V, E = euler(np.array([0.8, 0]), h)
90     energyerror.append((abs(E[-1] - E[0])) / (abs(E[0])))
91
92 plt.plot(H, energyerror)
93 plt.xscale('log')
94 plt.yscale('log')
95 plt.xlabel('stepsize_h')
96 plt.ylabel('error_in_energy')
97 plt.show()

```

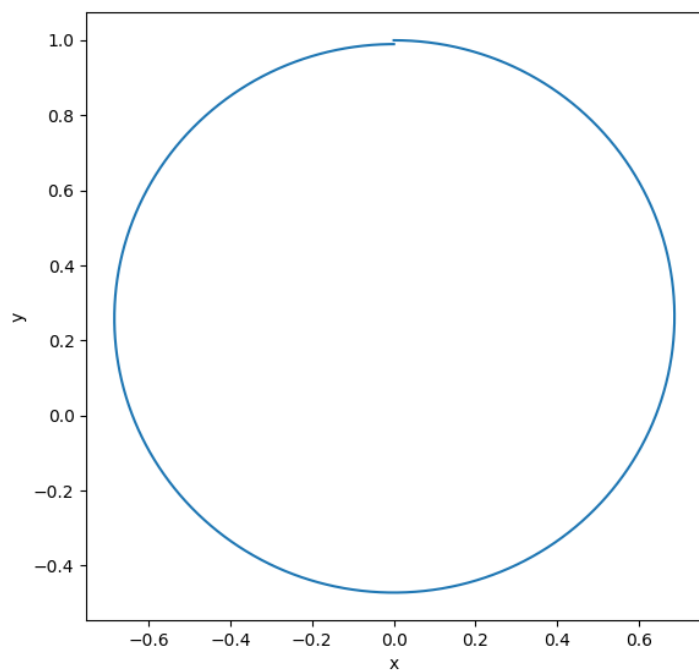


Figure 5: Third orbit with forward Euler algorithm

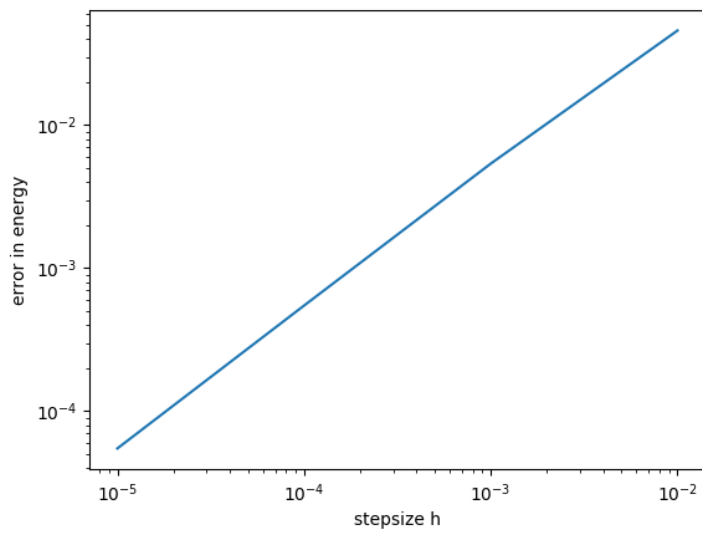


Figure 6: Third orbit energy error with forward Euler algorithm

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 5: Exercise02b.py

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4
5 #function that integrates the 2 - body - problem using the leapfrog
6 #algorithm. The initial velocity needs to be two dimensional
7 #vector, the step size h can be choosen. The program breaks when the
8 #orbit is complete.
9 def leapfrog(v_0, h, s_0 = np.array([0, 1])):
10     S = [s_0] #list for spatial coordinate
11     V = [v_0] #list for relative velocity
12     E = [] #list for the Energy
13     #first value for the acceleration
14     a = [-1*(np.array([(S[-1])[0], (S[-1])[1]])/(((S[-1]**2)[0] + (S[-1]**2)[1])
15         **1.5))]
16     test = 0 #variable to check if the orbit is complete
17     step = 0
18     maxstep = 1000000
19     while((test < 2) and (step < maxstep)):
20         vhalf = V[-1] + 0.5*h*a[-1]
21         S.append(S[-1] + h*vhalf)
22         a.append(-1*(np.array([(S[-1])[0], (S[-1])[1]])/(((S[-1]**2)[0] + (S
23             [-1]**2)[1]) **1.5))
24         V.append(vhalf + 0.5*h*a[-1])
25         E.append((0.5*(V[-1]**2)[0] + (V[-1]**2)[1]) - (1/((S[-1]**2)[0] + (S
26             [-1]**2)[1]) **0.5))
27
28         if ((S[-1])[0] < 0):
29             test = 1
30         if (test == 1 and (S[-1])[0] > 0):
31             test = 2
32
33     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. To be precise we going to use the same values as before.

Listing 6: Exercise02b.py

```

34 #for the given initial values the eccentricity is zero,
35 #therefore the orbit is circular
36 S, V, E = leapfrog(np.array([1, 0]), 0.0001)
37
38 plt.figure(figsize = (6.5, 10))
39 plt.plot(S[:, 0], S[:, 1])
40 plt.xlabel('x')
41 plt.ylabel('y')
42 plt.show()
43
44 energyerror = []
45 H = [0.01, 0.001, 0.0001, 0.0001, 0.00001]
46 for h in H:
47     S, V, E = leapfrog(np.array([1, 0]), h)
48     energyerror.append((abs(E[-1] - E[0])) / (abs(E[0])))
49
50 plt.plot(H, energyerror)
51 plt.xscale('log')
52 plt.yscale('log')
53 plt.xlabel('Stepsize_h')
54 plt.ylabel('error_in_energy')
55 plt.show()

```

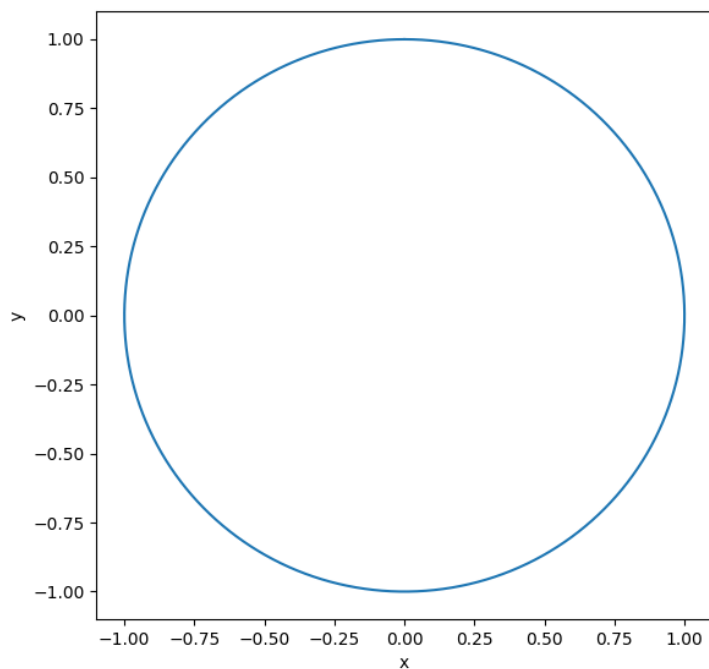


Figure 7: First orbit with leapfrog algorithm

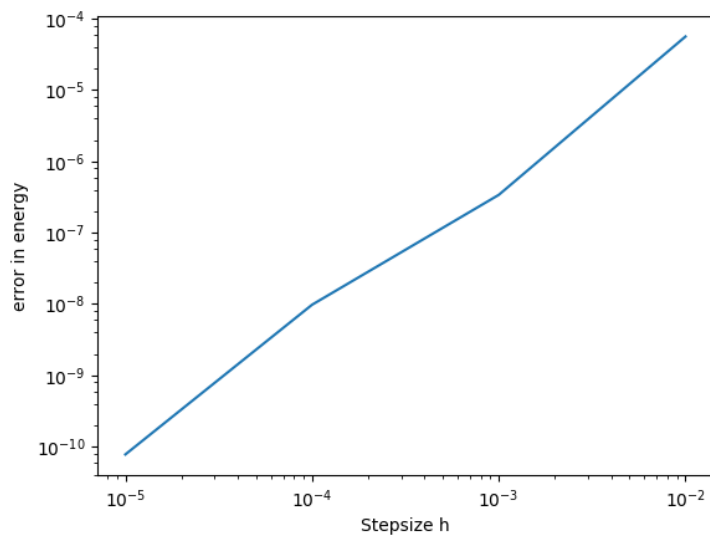


Figure 8: First orbit energy error with leapfrog algorithm

We repeat the calculation for two more different initial velocities.
For the second orbit we chose:

Listing 7: Exercise02b.py

```

58 #use different initial velocity to achieve another orbit
59 S, V, E = leapfrog(np.array([1.3, 0]), 0.0001)
60
61 plt.figure(figsize = (6.5, 10))
62 plt.plot(S[:, 0], S[:, 1])
63 plt.xlabel('x')
64 plt.ylabel('y')
65 plt.show()
66
67 energyerror = []
68 H = [0.01, 0.001, 0.0001, 0.0001, 0.00001]
69 for h in H:
70     S, V, E = leapfrog(np.array([1.3, 0]), h)
71     energyerror.append((abs(E[-1] - E[0])) / (abs(E[0])))
72
73 plt.plot(H, energyerror)
74 plt.xscale('log')
75 plt.yscale('log')
76 plt.xlabel('stepsize_h')
77 plt.ylabel('error_in_energy')
78 plt.show()

```

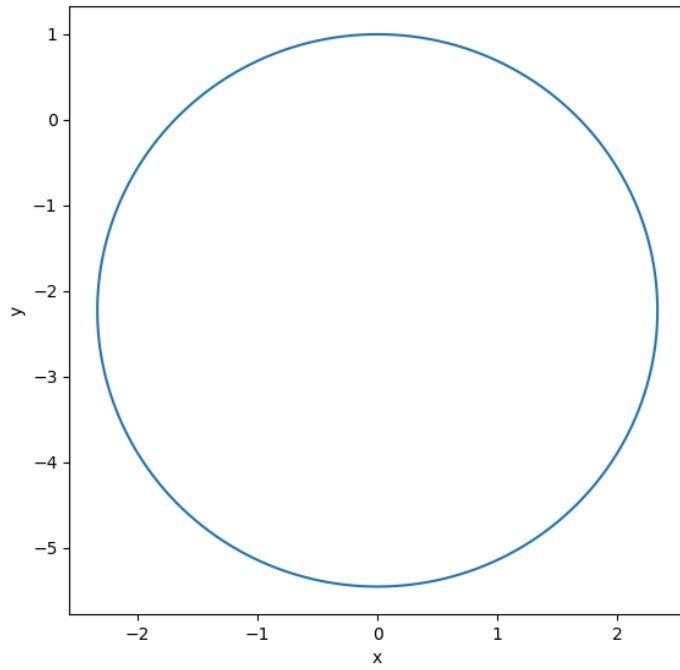


Figure 9: Second orbit with leapfrog algorithm

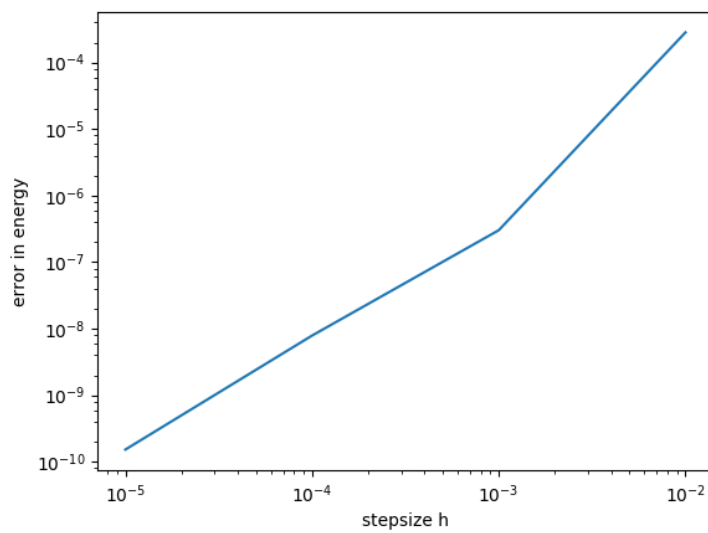


Figure 10: Second orbit energy error with leapfrog algorithm

For the third orbit we chose:

Listing 8: Exercise02b.py

```

81 #do the same for the third orbit with again another initial velocity
82 S, V, E = leapfrog(np.array([0.8, 0]), 0.001)
83
84 plt.figure(figsize = (6.5, 10))
85 plt.plot(S[:, 0], S[:, 1])
86 plt.xlabel('x')
87 plt.ylabel('y')
88 plt.show()
89
90 energyerror = []
91 H = [0.01, 0.001, 0.0001, 0.0001, 0.00001]
92 for h in H:
93     S, V, E = leapfrog(np.array([0.8, 0]), h)
94     energyerror.append((abs(E[-1] - E[0])) / (abs(E[0])))
95
96 plt.plot(H, energyerror)
97 plt.xscale('log')
98 plt.yscale('log')
99 plt.xlabel('stepsize_h')
100 plt.ylabel('error_in_energy')
101 plt.show()

```

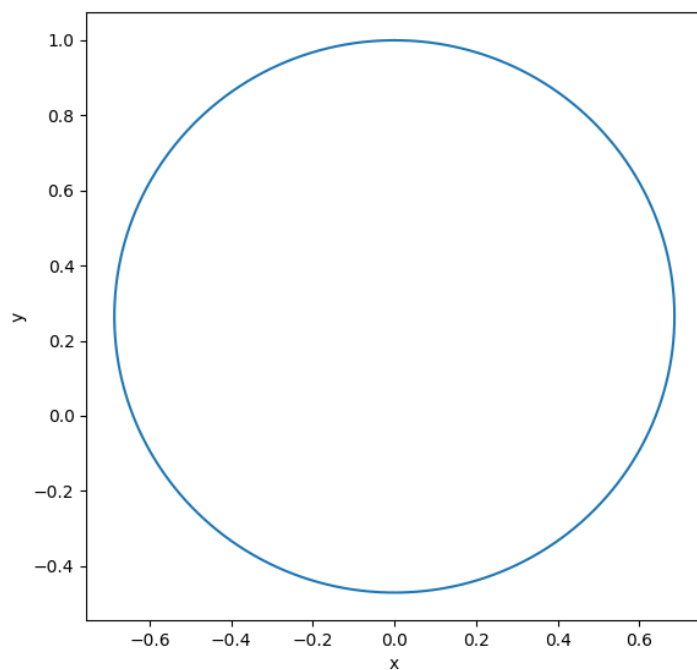


Figure 11: Third orbit with leapfrog algorithm

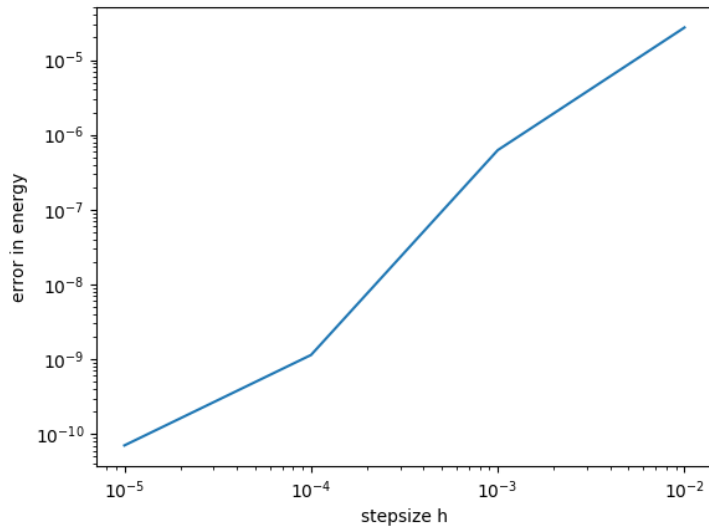


Figure 12: Third orbit energy error with leapfrog algorithm

The errors decrease with decreasing time steps for both integration schemes. For the Euler algorithm we expect a decreasing with $O(h^2)$, for the Leapfrog algorithm $O(h^3)$. This is what we get. 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 expect. Another difference between the error - plots, of the two integration variations, is, that for the Euler integration we get straight lines whereas for the Leapfrog integration the errors seem to scatter around the expected line.