

Introduction to Computational Physics - Exercise 2

Simon Groß-Böltting, Lorenz Vogel, Sebastian Willenberg

8. April 2020

In the first exercise from the worksheet we worked on the following code:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 class Body:
5     def __init__(self, mass, position, velocity=np.array([0., 0., 0.])):
6         self.mass = mass           # mass of the body
7         self.position = position   # initial position vector
8         self.velocity = velocity   # initial velocity vector
9
10    def forward_euler(body1, body2, G, dt, N):
11        ''' This function computes the relative motion of two point-like bodies
12            under their mutual gravitational influence using a
13            forward Euler integration procedure
14            Input: two bodies of the class "Body", gravitational constant G,
15                   time steps dt and number of time steps N
16            Output: location vector s and velocity vector w '''
17
18        # compute the total mass of the two bodies and set the
19        # characteristic length scale as the initial separation
20        M = body1.mass+body2.mass
21        R0 = np.linalg.norm(body1.position-body2.position)
22        h = dt/np.sqrt(R0**3/(G*M)) # compute the dimensionless step size
23
24        s = np.zeros((int(N),3))      # create array for the location vectors
25        s[0] = (body1.position-body2.position)/R0
26
27        w = np.zeros((int(N),3))      # create array for the velocity vectors
28        w[0] = (body1.velocity-body2.velocity)/np.sqrt(G*M/R0)
29
30        for i in range(1,int(N)):    # compute the relative motion
31            s[i] = s[i-1]+(w[i-1]*dt)
32            w[i] = w[i-1]-(s[i-1]/np.linalg.norm(s[i-1]))**3.*dt
33        return (s,w)
34
35    def get_acceleration(body1, body2, G, position):
36        return -(body1.mass+body2.mass)*G*position/np.linalg.norm(position)**3
37
38    def leapfrog(body1, body2, G, dt, N):
39        ''' This function computes the relative motion of two point-like bodies
40            under their mutual gravitational influence using a Leapfrog scheme
41            Input: two bodies of the class "Body", gravitational constant G,
42                   time steps dt and number of time steps N
43            Output: location vector s and velocity vector w '''
44
45        # compute the total mass of the two bodies and set the
46        # characteristic length scale as the initial separation
47        M = body1.mass+body2.mass
48        R0 = np.linalg.norm(body1.position-body2.position)
49        h = dt/np.sqrt(R0**3/(G*M)) # compute the dimensionless step size
```

```

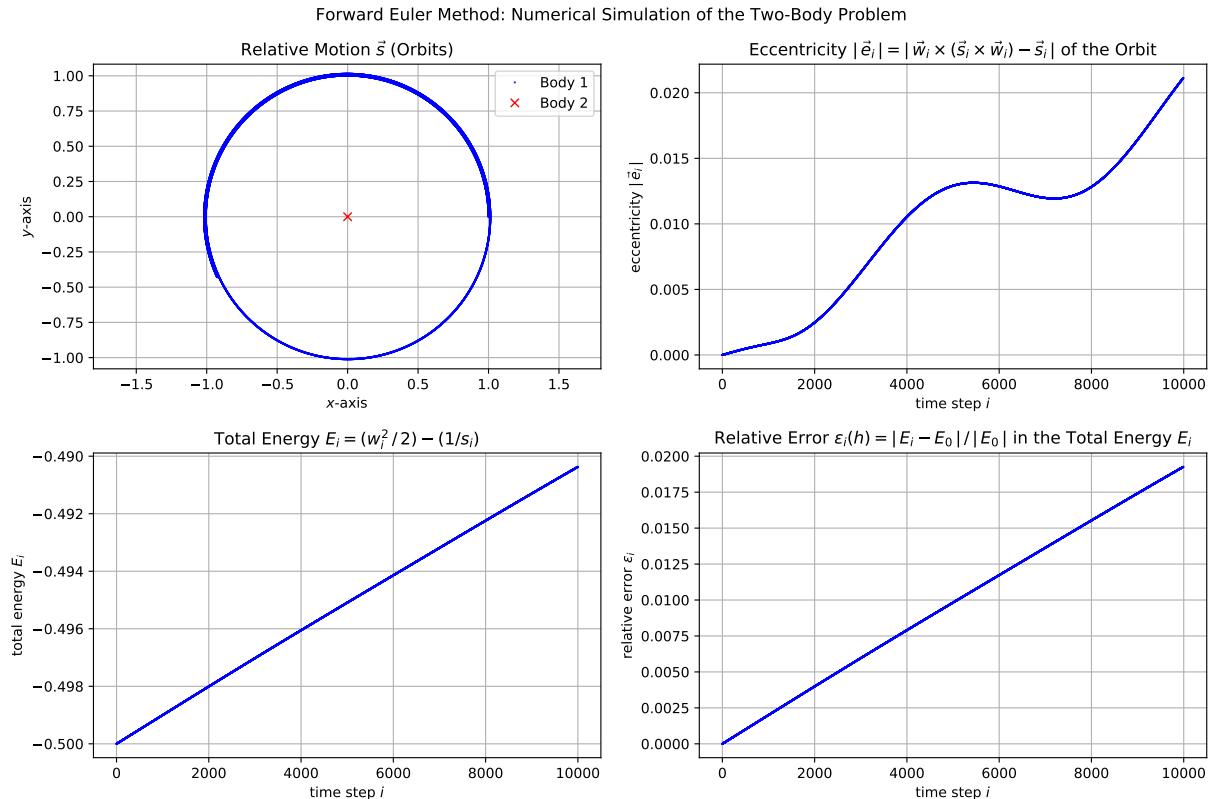
50
51     s = np.zeros((int(N),3))      # create array for the location vectors
52     s[0] = (body1.position-body2.position)/R0
53
54     w = np.zeros((int(N),3))      # create array for the velocity vectors
55     w[0] = (body1.velocity-body2.velocity)/np.sqrt(G*M/R0)
56
57     w[0] -= (s[0]/np.linalg.norm(s[0]))**3)*h/2
58     for i in range(1,int(N)):
59         w[i] = w[i-1]+get_acceleration(body1, body2, G, s[i-1])*h
60         s[i] = s[i-1]+w[i]*h
61
62     return (s,w)
63
64 def total_energy(s,w):
65     ''' Function to compute the total energy of the system from the
66         dimensionless vectors s (location) and w (velocity) '''
67     energy = np.zeros(np.shape(s)[0])
68     for i in range(0,np.shape(s)[0]):
69         energy[i] = (0.5*np.linalg.norm(w[i])**2)-(1/np.linalg.norm(s[i]))
70
71     return energy
72
73 def relative_error(energy):
74     ''' Function to compute the relative error in the total energy at
75         step i compared to the initial value 0 '''
76     rel_error = np.zeros(np.shape(s)[0])
77     for i in range(0,np.shape(s)[0]):
78         rel_error[i] = abs(energy[i]-energy[0])/abs(energy[0])
79
80     return rel_error
81
82 def angular_momentum(s,w):
83     ''' Function to compute the angular momentum'''
84     angular_momentum = np.zeros((np.shape(s)[0],3))
85     for i in range(0,np.shape(s)[0]):
86         angular_momentum[i] = np.cross(s[i],w[i])
87
88     return angular_momentum
89
90 def eccentricity(s,w):
91     ''' Function to compute the eccentricity
92         from the Laplace-Runge-Lenz vector '''
93     LRL = np.zeros((np.shape(s)[0],3))
94     for i in range(0,np.shape(s)[0]):
95         LRL[i] = np.cross(w[i], np.cross(s[i],w[i]))-s[i]
96
97     return np.linalg.norm(LRL, axis=1)
98
99 # set the initial properties of the two-body system
100 G = 1. # set the gravitational constant
101 body1 = Body(1., np.array([1.,0.,0.]))
102 body2 = Body(1., np.array([0.,0.,0.]))
103 dt = 1e-3; N=1e4 # set step length and number of steps
104
105
106 ## Numerical Simulation of the Two-Body Problem using Forward Euler Method
107 fig, axs = plt.subplots(2,2,figsize=(12,8), constrained_layout=True)
108 fig.suptitle(r'Forward Euler Method: Numerical Simulation of the Two-Body
109 Problem')
110 s, w = forward_euler(body1, body2, G, dt, N)

```

```

111 energy = total_energy(s,w) # compute the total energy
112 rel_error = relative_error(energy) # compute the relative error
113 angular_momentum = angular_momentum(s,w) # compute the angular momentum
114 eccentricity = eccentricity(s,w) # compute the eccentricity
115
116 axs[0,0].plot(s[:,0], s[:,1], 'b.', markersize=1, label='Body 1') # plot Body 1
117 axs[0,0].plot([0], [0], 'rx', label='Body 2') # plot Body 2
118 axs[0,0].set_title(r'Relative Motion $\vec{s}$ (Orbits)') # set title
119 axs[0,0].set_xlabel(r'$x$-axis'); axs[0,0].set_ylabel(r'$y$-axis') # set axes labels
120 axs[0,0].axis('equal'); axs[0,0].legend(loc='best') # set axis equal and add legend
121
122 axs[0,1].plot(range(0,len(eccentricity)), eccentricity, 'b.', markersize=1) # plot Eccentricity
123 axs[0,1].set_title(r'Eccentricity $|\vec{e}_i|$ of the Orbit') # set title
124 +r'$(\vec{s}_i \times \vec{w}_i) - \vec{s}_i$'
125 axs[0,1].set_xlabel(r'time step $i$') # set x-axis label
126 axs[0,1].set_ylabel(r'eccentricity $|\vec{e}_i|$') # set y-axis label
127
128 axs[1,0].plot(range(0,len(energy)), energy, 'b.', markersize=1) # plot Total Energy
129 axs[1,0].set_title(r'Total Energy $E_i = (w_i^2/2) - (1/s_i)$') # set title
130 axs[1,0].set_xlabel(r'time step $i$'); axs[1,0].set_ylabel(r'total energy $E_i$') # set axes labels
131
132 axs[1,1].plot(range(0,len(rel_error)), rel_error, 'b.', markersize=1) # plot Relative Error
133 axs[1,1].set_title(r'Relative Error $\epsilon_i(h) = |E_i - E_0| / |E_0|$ in the Total Energy $E_i$') # set title
134 +r' in the Total Energy $E_i$'
135 axs[1,1].set_xlabel(r'time step $i$'); axs[1,1].set_ylabel(r'relative error $\epsilon_i$') # set axes labels
136
137 for ax in fig.get_axes():
138     ax.grid()
139 fig.savefig('figures/Two-Body-Problem_Euler-Method.pdf', format='pdf', dpi=100)

```



2 Error Analysis of Euler Scheme

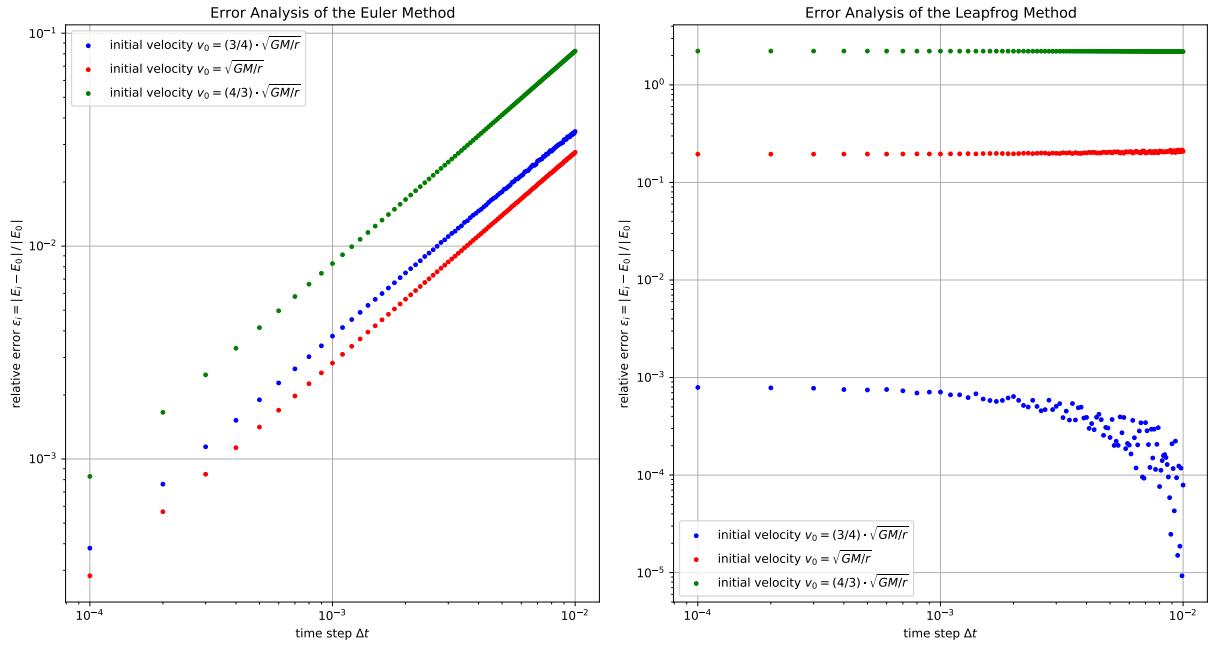
- (a) After varying the initial velocity three times we can see that the double logarithmic plot of the function is always a straight line with the same slope but different starting points. Therefore all the lines are parallel. This differs quite a lot from what we would expect. The law of conservation of energy predicts, that the energy in the system should remain constant. But we actually see that the energy in the system increases. This also becomes apparent in the plot with the relative motion. We can see that the path of the second body deviates slightly from the circular path it is supposed to take when the initial velocity is set to $v_0 = 2\sqrt{\frac{GM}{R_0}}$. That means that the distance between both bodies increases with each orbit.

```

1 ## Error Analysis of the Euler and the Leapfrog Scheme
2 v0_range = [(3/4)*v0, v0, (4/3)*v0]                      # set the initial
   velocities
3 dt_range = np.linspace(1e-4, 1e-2, int(1e2))      # set different time steps
4
5 # compute the relative error in the energy for Euler and Leapfrog
6 rel_error_euler = np.zeros((len(v0_range), len(dt_range)))
7 rel_error_leapfrog = np.zeros((len(v0_range), len(dt_range)))
8
9 for i in range(0, len(v0_range)):
10    for j in range(0, len(dt_range)):
11        N = int(2*np.sqrt(1.**3/(1.*2.))/dt_range[j])
12        body1.velocity = np.array([0., v0_range[i], 0.])
13
14        s, w = forward_euler(body1, body2, G, dt_range[j], N)
15        rel_error_euler[i,j] = relative_error(total_energy(s,w))[N-1]
16
17        s, w = leapfrog(body1, body2, G, dt_range[j], N)
18        rel_error_leapfrog[i,j] = relative_error(total_energy(s,w))[N-1]
19
20 # plot the relative error in the energy for Euler and Leapfrog
21 fig, axs = plt.subplots(1,2, figsize=(14,8), constrained_layout=True)
22 fig.suptitle('Numerical Simulation of the Two-Body Problem: Error Analysis
   of the Integration Schemes\n
   +r'relative error $\epsilon_i=\sqrt{|E_i-E_0|}, \sqrt{|E_i-E_0|}, \sqrt{|E_i-E_0|}, \sqrt{|E_i-E_0|}$ in the energy'
   +r' at the end of the second orbit as a function of the time
   step $\Delta t$')
23 label = [r'initial velocity $v_0=(3/4)\cdot\sqrt{GM/r}$',
24          r'initial velocity $v_0=\sqrt{GM/r}$',
25          r'initial velocity $v_0=(4/3)\cdot\sqrt{GM/r}$']
26 color = ['b.', 'r.', 'g.']
27 axs[0].set_title('Error Analysis of the Euler Method')
28 axs[1].set_title('Error Analysis of the Leapfrog Method')
29
30 for i in range(0,3):
31     axs[0].plot(dt_range, rel_error_euler[i,:], color[i], label=label[i])
32     axs[1].plot(dt_range, rel_error_leapfrog[i,:], color[i], label=label[i])
33
34 for ax in fig.get_axes():
35     ax.set_xlabel(r'time step $\Delta t$')
36     ax.set_ylabel(r'relative error $\epsilon_i=\sqrt{|E_i-E_0|}, \sqrt{|E_i-E_0|}, \sqrt{|E_i-E_0|}, \sqrt{|E_i-E_0|}$')
37     ax.grid(); ax.legend(loc='best')
38     ax.set_xscale('log'); ax.set_yscale('log')
39
40 fig.savefig('figures/Two-Body-Problem_Error-Analysis.pdf', format='pdf',
41 dpi=100)
42 plt.show(); plt.clf(); plt.close()

```

Numerical Simulation of the Two-Body Problem: Error Analysis of the Integration Schemes
 relative error $\varepsilon_i = |E_i - E_0| / |E_0|$ in the energy at the end of the second orbit as a function of the time step Δt



- (b) After implementing the Leapfrog scheme we can see, that the energy remains constant for initial velocities that are smaller or equal to $v_0 = 2\sqrt{\frac{GM}{R_0}}$. This means that the conservation of energy applies here. If we plot the motion in this case we can see that the circular remains equidistant to the center. But if we look at initial velocities that are greater than $v_0 = 2\sqrt{\frac{GM}{R_0}}$ we can see that the energy of the Leapfrog scheme doesn't remain constant. The energy in the system actually decreases in this case.