040692 PR Practical in numerical Astronomy

Dr. Elke Pilat-Lohinger & Maximilian Zimmermann, MSc

3-body problem (N-body problem)

Dawid Stepanovic

February 7, 2022

Exercise 2a & 2b: Stability and Chaos in the three body problem

Develop an N-body simulation program to integrate the trajectories of various bodies under their mutual gravitational influence. Use the barycenter coordinate system. The code should additionally calculate the total energy E, the specific angular momentum j, the Runge-Lenz vector e, and the motion of the center of mass. Implement the following time integrators: Runge-Kutta of 4th order.

Study the following planet pairs:

- Group A: Jupiter-Saturn
- Group B: Venus-Earth

The parameters a (semi-major axis) and e (eccentricity) for the inner planet are given in Figure 1. Further determine the Hill radius Δ_c and reduce the distance between the two planets such that the difference equals $3 \times \Delta_c$.

Planetary Mean Orbits (J2000)
(epoch = J2000 = 2000 January 1.5)
(see the table of rates below)

Planet (mean)	a AU	e	i deg	Omega deg	~omega deg	L deg
Mercury	0.38709893	0.20563069	7.00487	48.33167	77.45645	252.25084
Venus	0.72333199	0.00677323	3.39471	76.68069	131.53298	181.97973
Earth	1.00000011	0.01671022	0.00005	-11.26064	102.94719	100.46435
Mars	1.52366231	0.09341233	1.85061	49.57854	336.04084	355.45332
Jupiter	5.20336301	0.04839266	1.30530	100.55615	14.75385	34.40438
Saturn	9.53707032	0.05415060	2.48446	113.71504	92.43194	49.94432
Uranus	19.19126393	0.04716771	0.76986	74.22988	170.96424	313.23218
Neptune	30.06896348	0.00858587	1.76917	131.72169	44.97135	304.88003
Pluto	39.48168677	0.24880766	17.14175	110.30347	224.06676	238.92881

Figure 1: orbital parameter

Exercise 2a is implemented with following parameters:

- a_1 = see Figure 1, $a_2 = a_1 + 3 \times \Delta_c$
- e_1 = see Figure 1, $e_2 = e_1 + 0.1$, $e_3 = e_2 + 0.2$
- period = sidereal period of inner planet (Venus: 224.701 [in days], Jupiter: 4332.589 [in days])
- barycenter implementation
- Using a Runge-Kutta of 4th order

The Runge-Kutta of 4th order is defined as

$$\begin{aligned} k_1 &= f(t_k, y_k) \\ k_2 &= f\left(t_k + \frac{1}{2}h, y_k + \frac{1}{2}k_1\right) \\ k_3 &= f\left(t_k + \frac{1}{2}h, y_k + \frac{1}{2}k_2\right) \\ k_4 &= f\left(t_k + h, y_k + k_3\right). \end{aligned}$$

We then use a weighted sum of these slopes to get our final estimate by

$$y_{k+1} = y_k + \frac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right).$$

Exercise 2b is implemented with the same parameters as given in exercise 2a but with an adaptive step size. The time step can be chosen to be fixed for the whole simulation

$$\Delta t = \eta. \tag{1}$$

Remark: However, to improve the accuracy of the simulation, the time step can also be adaptive and change during the simulation. The time step may for example depend on the current curvature of the trajectory. In this case, the time step is calculated from the accelerations and the jerks

$$\Delta t_{acc} = \tilde{\eta} \min_{i=1,\dots,N} \frac{|\mathbf{a}_i(t_n)|}{|\dot{\mathbf{a}}_i(t_n)|}. \tag{2}$$

We also have implemented an adaptive step size simulation based on the position and the velocity

$$\Delta t_{vel} = \tilde{\eta} \min_{i=1,\dots,N} \frac{|\sum_{i \neq j} r_j - r_i|}{|\sum_{i \neq j} v_j - v_i|}. \tag{3}$$

This simulation uses larger time steps for particles that are far apart or moving slowly with smaller time steps when particles are closer together or moving quickly and is basically build on the same intuition as eq. (2). Additionally to the previous the adaptive step size methods we have implemented a Runge-Kutta-Fehlberg method. This method compares two different approximations for the solution and if the two answers do not agree to a specified accuracy (ϵ),

the step size is reduced, but if the answers agree to more significant digits than required, the step size is increased. The Runge-Kutta-Fehlberg method is specified as follows

$$
\begin{aligned}
k_1 &= hf(t_k, y_k) \\
k_2 &= hf\left(t_k + \frac{1}{4}h, y_k + \frac{1}{4}k_1\right) \\
k_3 &= hf\left(t_k + \frac{3}{8}h, y_k + \frac{3}{32}k_1 + \frac{9}{32}k_2\right) \\
k_4 &= hf\left(t_k + \frac{12}{13}h, y_k + \frac{1932}{2197}k_1 - \frac{7200}{2197}k_2 + \frac{7296}{2197}k_3\right) \\
k_5 &= hf\left(t_k + h, y_k + \frac{439}{216}k_1 - 8k_2 + \frac{3680}{513}k_3 - \frac{845}{4104}k_4\right) \\
k_6 &= hf\left(t_k + \frac{1}{2}h, y_k - \frac{8}{27}k_1 + 2k_2 - \frac{3544}{2565}k_3 + \frac{1859}{4104}k_4 - \frac{11}{40}k_5\right)
\end{aligned}
$$

Then an approximation to the solution is made by using a Runge-Kutta method of 4th order and a Runge-Kutta method of 5th order:

$$y_{k+1} = y_k + \frac{25}{216}k_1 + \frac{1408}{2565}k_3 + \frac{2197}{4101}k_4 - \frac{1}{5}k_5$$

$$x_{k+1} = x_k + \frac{16}{135}k_1 + \frac{6656}{12825}k_3 + \frac{28561}{56430}k_4 - \frac{9}{50}k_5 + \frac{2}{55}k_6.$$

The optimal step size δh can be determined by multiplying the scalar δ times the current step size h, where δ is defined as

$$\delta = \left(\frac{\epsilon h}{2|x_{k+1} - y_{k+1}|}\right)^{1/4} \approx 0.84\left(\frac{\epsilon h}{|x_{k+1} - y_{k+1}|}\right)^{1/4}.$$

The Hill radius is given by

$$R_{Hill} = a\left(\frac{\mu}{3}\right)^{1/3}$$

and the Laplace radius is given by

$$R_{Laplace} = a\left(\mu\right)^{2/5},$$

where $\mu = \frac{m}{M_{sun}}$. Both radius calculations are implemented.

The N-body function is implemented as follows:

Nbodyproblem(method,nbody,planet,period,step_val,norbit_val,hillradius,shift_hill,e_val, shift_eval, epsilon,simulation)

- method: We have implemented four methods. A Runge-Kutta of order 4, a Runge-Kutta of order 4 with an adaptive step size using eq. (2), a Runge-Kutta of order 4 with an adaptive step size using eq. (3) and a Runge-Kutta-Fehlberg method, which can be initialized by "rk4", "rk4_acc", "rk4_vel", "rk45_fehlberg". Remark: For implementing a Runge-Kutta of order 4 with an adaptive step size using eq. (2) and a Runge-Kutta of order 4 with an adaptive step size using eq. (3) we need to add an multiplying factor (multiplier - code line 270 and code line 310) to obtain an reasonable step size. However, for both methods $\tilde{\eta}$ is assumed to be a fixed value, such that $\tilde{\eta} = \eta \times$ multiplier, where η is the initial step size representing a fraction of the orbital period. In case of using Runge-Kutta of order 4 with an adaptive step size implemented by eq. (2) we also have to control the evolution of the step size by restricting the actual value not to be greater than twice as the previous step, since the step size development for this method is quite unstable and volatile. This problem can be resolved by using Runge-Kutta of order 4 with an adaptive step size implemented by eq. (3) which shows a more "tempered" step size evolution. For implementing the Runge-Kutta-Fehlberg method we use an accuracy ϵ of 10^{-6}. The accuracy ϵ has to be adapted to 10^{-8} when implementing the setting of group A (Jupiter-Saturn).
- nbody: Number of bodies in the system.
- planet: An array that stores the parameter of the system. For this reason one has to define an array of **np.zeros((nbody - 1,8))**. The order of the first six parameter in the array is structured as in Figure 1. The last two parameter are the weighted mass $\mu = \frac{m}{M_{sun}}$ and the sidereal orbit period [in days] of the planet.
- period: The sidereal orbit period [in days] of the inner planet.
- step_val: step size
- norbit_val: Number of orbits based on the sidereal orbit period of the inner planet.
- hillradius: The implemented Laplace and Hill radius which is initialized by "hill" and "laplace". For deactivating the hill radius modus pass False.
- shift_hill: For changing the radius of the inner planet pass 0 and for changing radius for the outer planet pass 1.
- e_val: Pass the additional change in eccentricity e.
- shift_eval: For changing the eccentricity of the inner planet pass 0 and for changing eccentricity for the outer planet pass 1.
- epsilon: Accuracy ϵ of the Runge-Kutta-Fehlberg simulation.
- simulation: Activate or deactivate the simulation modus by passing True or False.

The gravitational constant is converted to

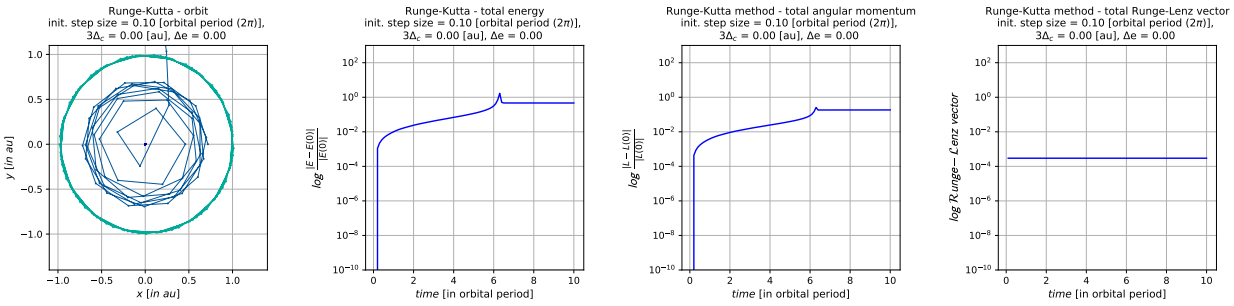
$$G_{new} = \left(\frac{G_{SI}}{au^3} \times \text{mean solar day[in sec]} \times m_{Sun} \right)$$

to represent the orbital evolution in days. To start the orbital movement of the bodies from the opposite side we set the argument of periapsis for the Earth (group A: Saturn) (ω) to 180°. The plot output shows in addition the mean total energy, the mean angular momentum, the number of steps, the minimum and maximum step size [in days] and the orbit period of the inner planet [in days].

Conclusion: We see, as expected, that the adaptive step size method produce much more stable orbits. Moreover, the higher the change in eccentricity the more stable orbits become. The Laplace radius implementation provides similar results. However, we see that the Runge-Kutta of order 4 with an adaptive step size implemented by eq. (3) provides the most effective results regarding the overall number step sizes, but fails for small step size values. In opposite, we see that the Runge-Kutta-Fehlberg method produces very good results for small initial step size values. Due to the very large amount of images we only present the required plots.

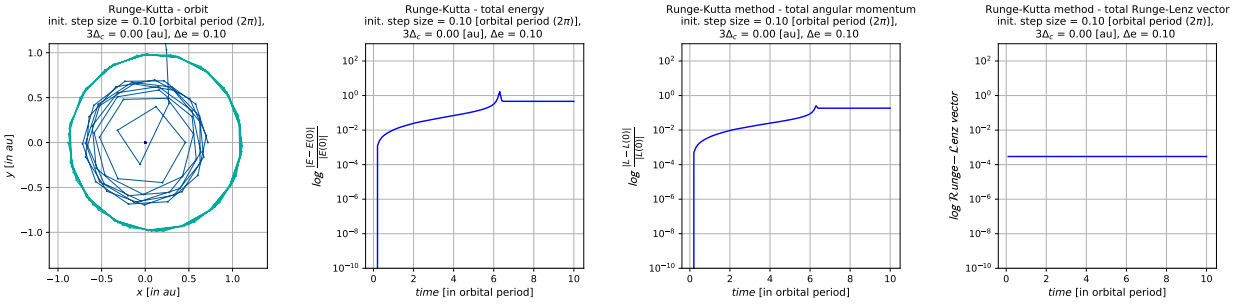
Plots of exercise 2a:

- Plots for a step-size s = 0.10 and a eccentricity Δe $\in \{0.00, 0.10, 0.30\}$ for Runge-Kutta of order 4.



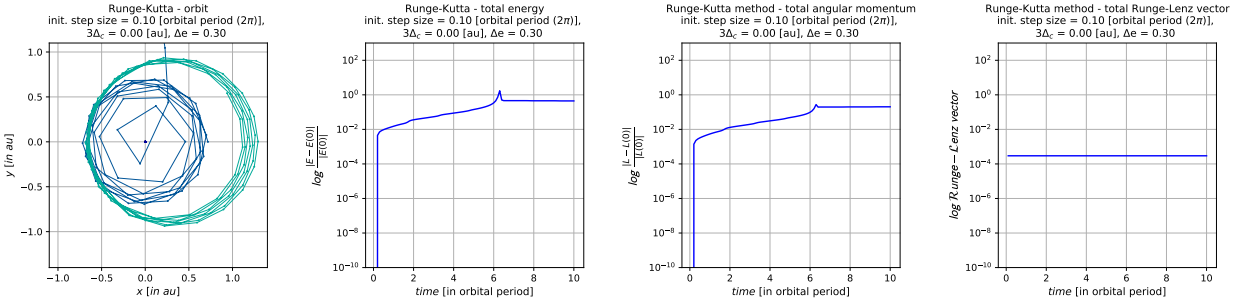
mean total energy: -8.455518193448771e-10
mean total angular momentum: 7.992209240863144e-08

steps: 99, min. step [in days]: 22.470100000000002, max. step [in days]: 22.470100000000002
period of inner orbit [in days]: 224.701



mean total energy: -8.471921931388928e-10
mean total angular momentum: 7.953209534208699e-08

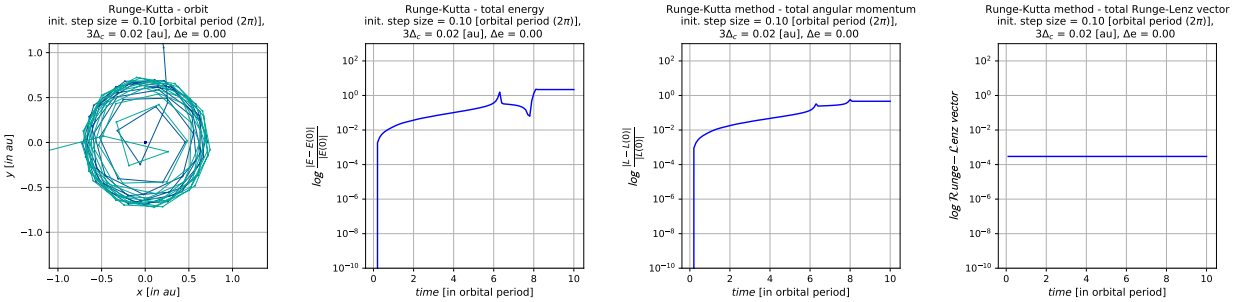
steps: 99, min. step [in days]: 22.470100000000002, max. step [in days]: 22.470100000000002
period of inner orbit [in days]: 224.701



mean total energy: -8.58694223547624e-10
mean total angular momentum: 7.67162106924786e-08

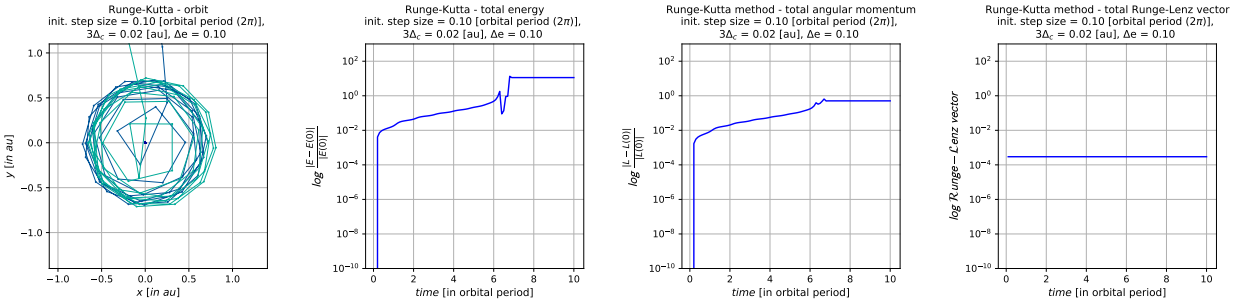
steps: 99, min. step [in days]: 22.470100000000002, max. step [in days]: 22.470100000000002
period of inner orbit [in days]: 224.701

- Plots for a step-size s = 0.10 and a eccentricity Δe $\in \{0.00, 0.10, 0.30\}$ for Runge-Kutta of order 4 with an adapted semi-major axis a of the outer planet that differs by $3R_{Hill}$ from the inner planet. $3\Delta_c$ denotes the Hill radius shift.



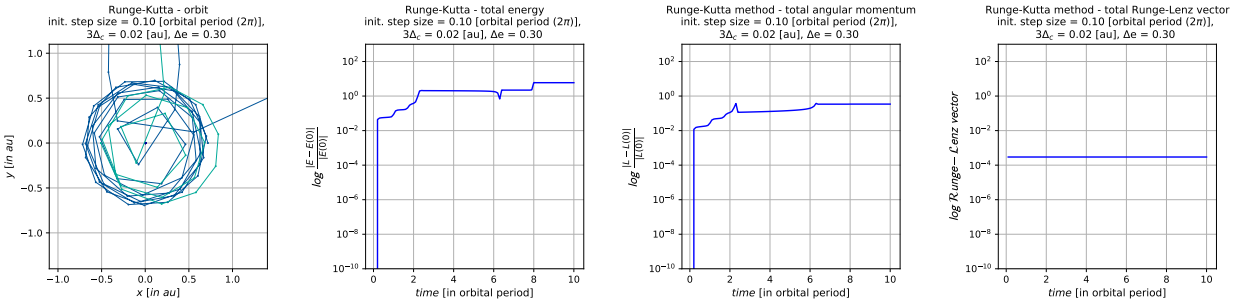
mean total energy: -6.844055290111573e-10
mean total angular momentum: 6.631841953711349e-08

steps: 99, min. step [in days]: 22.470100000000002, max. step [in days]: 22.470100000000002
period of inner orbit [in days]: 224.701



mean total energy: 2.8825323888998275e-09
mean total angular momentum: 6.218082874719717e-08

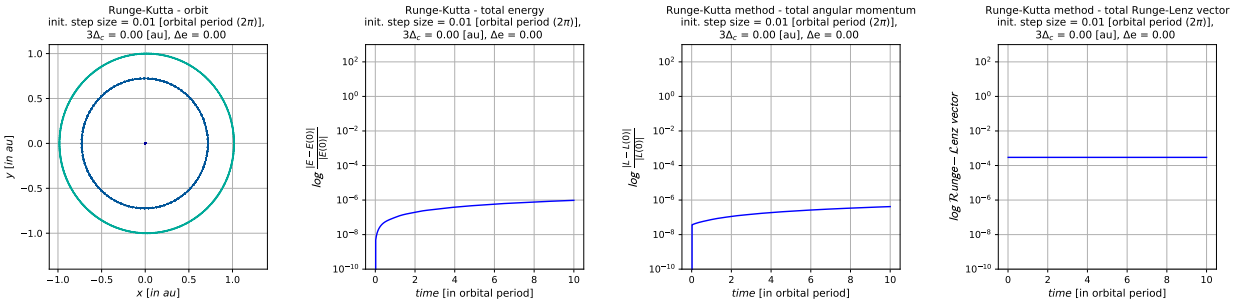
steps: 99, min. step [in days]: 22.470100000000002, max. step [in days]: 22.470100000000002
period of inner orbit [in days]: 224.701



mean total energy: 1.4537096722191 53e-09
mean total angular momentum: 6.239033826712669e-08

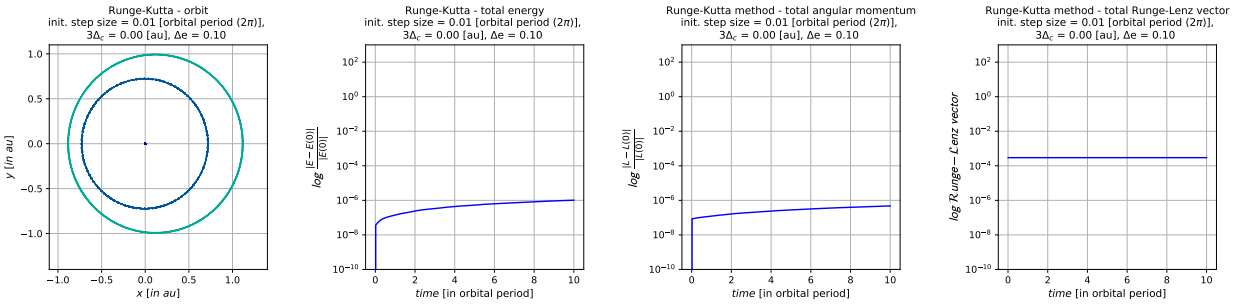
steps: 99, min. step [in days]: 22.470100000000002, max. step [in days]: 22.470100000000002
period of inner orbit [in days]: 224.701

- Plots for a step-size s = 0.01 and a eccentricity Δe $\in \{0.00, 0.10, 0.30\}$ for Runge-Kutta of order 4.



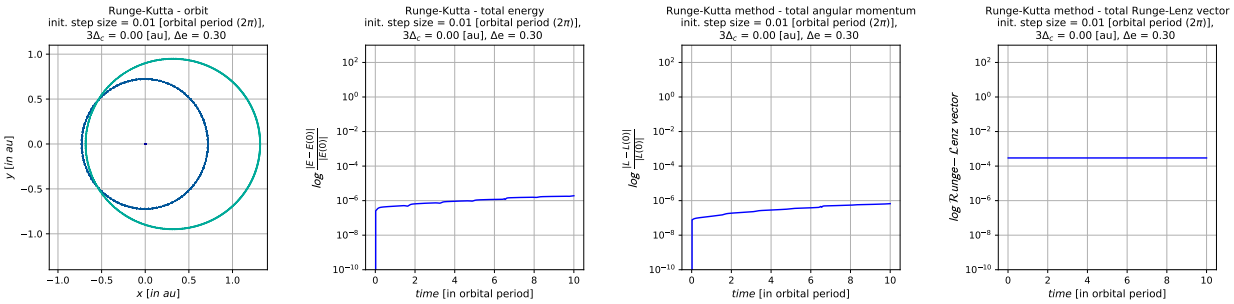
mean total energy: -9.45089353815702e-10
mean total angular momentum: 8.74651506558244e-08

steps: 999, min. step [in days]: 2.24701, max. step [in days]: 2.24701
period of inner orbit [in days]: 224.701



mean total energy: -9.450927013779987e-10
mean total angular momentum: 8.711929449570481e-08

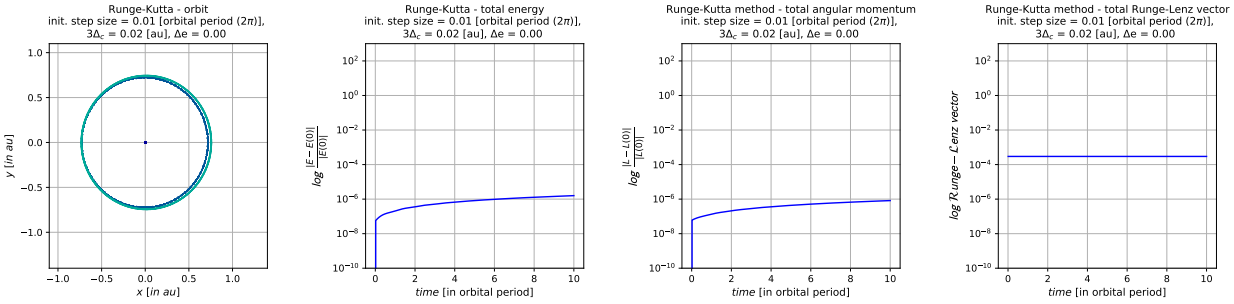
steps: 999, min. step [in days]: 2.24701, max. step [in days]: 2.24701
period of inner orbit [in days]: 224.701



mean total energy: -9.451026945994325e-10
mean total angular momentum: 8.48128579417106e-08

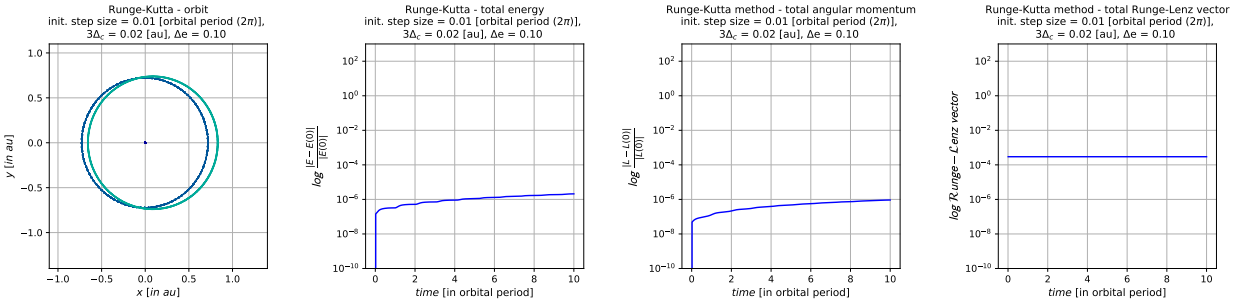
steps: 999, min. step [in days]: 2.24701, max. step [in days]: 2.24701
period of inner orbit [in days]: 224.701

- Plots for a step-size s = 0.01 and a eccentricity Δe $\in \{0.00, 0.10, 0.30\}$ for Runge-Kutta of order 4 with an adapted semi-major axis a of the outer planet that differs by $3R_{Hill}$ from the inner planet. $3\Delta_c$ denotes the Hill radius shift.



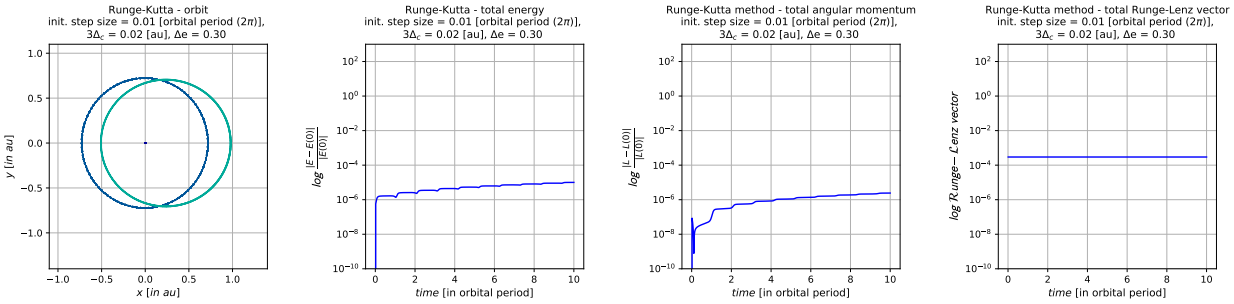
mean total energy: -1.0983111802651275e-09
mean total angular momentum: 8.03535444559539e-08

steps: 999, min. step [in days]: 2.24701, max. step [in days]: 2.24701
period of inner orbit [in days]: 224.701



mean total energy: -1.0983159332344595e-09
mean total angular momentum: 8.005530350472864e-08

steps: 999, min. step [in days]: 2.24701, max. step [in days]: 2.24701
period of inner orbit [in days]: 224.701

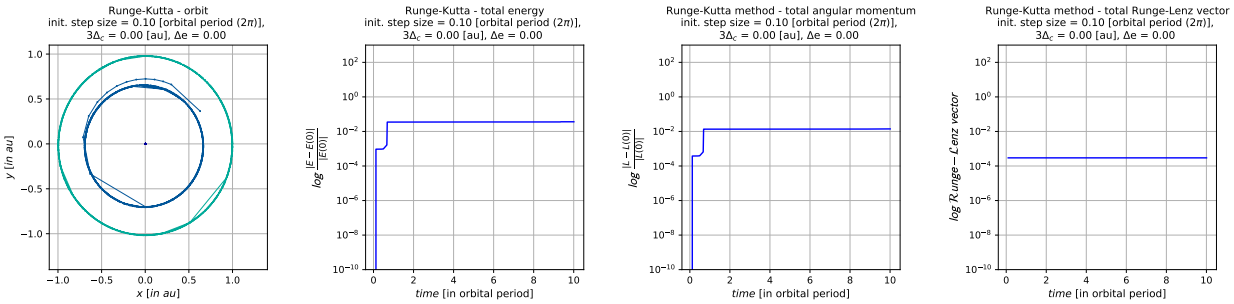


mean total energy: -1.0983334325927373e-09
mean total angular momentum: 7.80663525200942e-08

steps: 999, min. step [in days]: 2.24701, max. step [in days]: 2.24701
period of inner orbit [in days]: 224.701

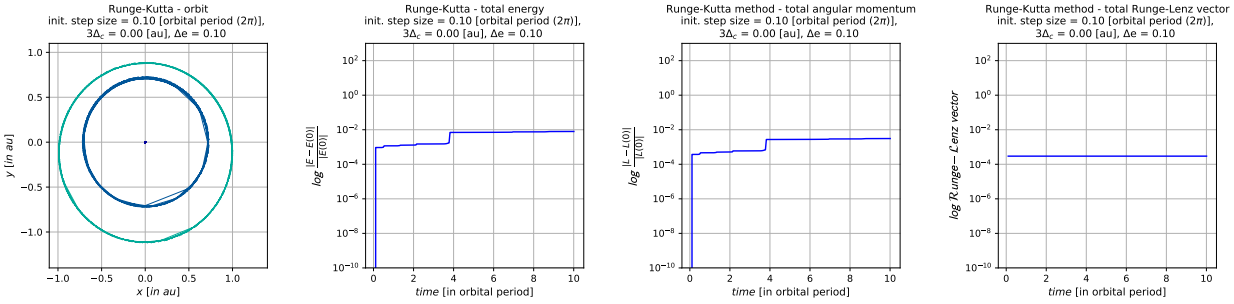
Plots of exercise 2b:

- Plots for a step-size s = 0.10 and a eccentricity $\Delta e \in \{0.00, 0.10, 0.30\}$ for Runge-Kutta of order 4 with an adaptive step size implemented by eq. (2).



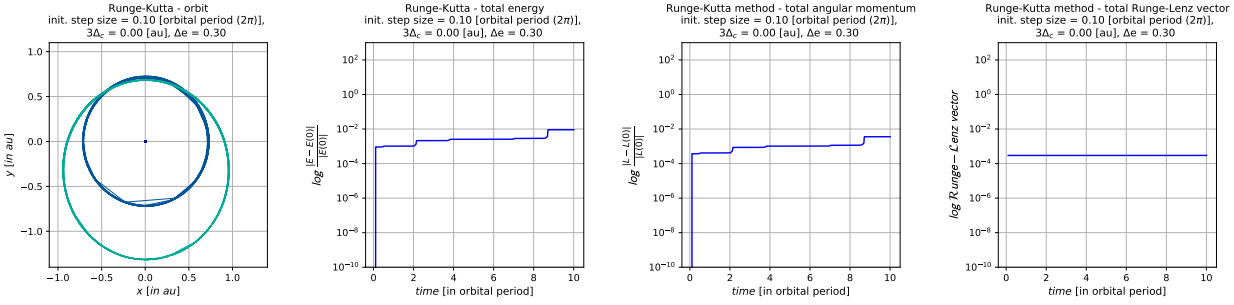
mean total energy: -9.779358702899723e-10
mean total angular momentum: 8.624835244625718e-08

steps: 841, min. step [in days]: 0.7026731258754995, max. step [in days]: 39.25628507760085
period of inner orbit [in days]: 224.701



mean total energy: -9.50287500206 54e-10
mean total angular momentum: 8.689526037742716e-08

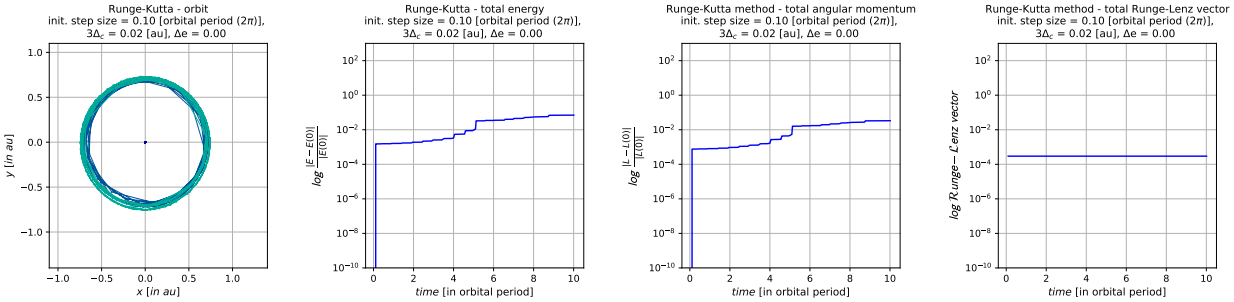
steps: 742, min. step [in days]: 0.6014244879654724, max. step [in days]: 29.224970727394673
period of inner orbit [in days]: 224.701



mean total energy: -9.480244550157043e-10
mean total angular momentum: 8.467103190327894e-08

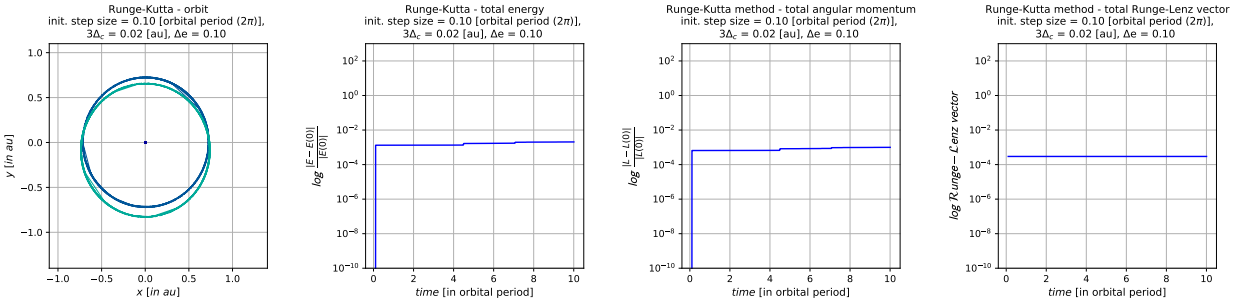
steps: 2245, min. step [in days]: 0.07889974651602169, max. step [in days]: 29.137170085332652
period of inner orbit [in days]: 224.701

- Plots for a step-size s = 0.10 and a eccentricity Δe $\in \{0.00, 0.10, 0.30\}$ for Runge-Kutta of order 4 with an adaptive step size implemented by eq. (2) and with a adapted semi-major axis a of the outer planet that differs by $3R_{Hill}$ from the inner planet. $3\Delta_c$ denotes the Hill radius shift.



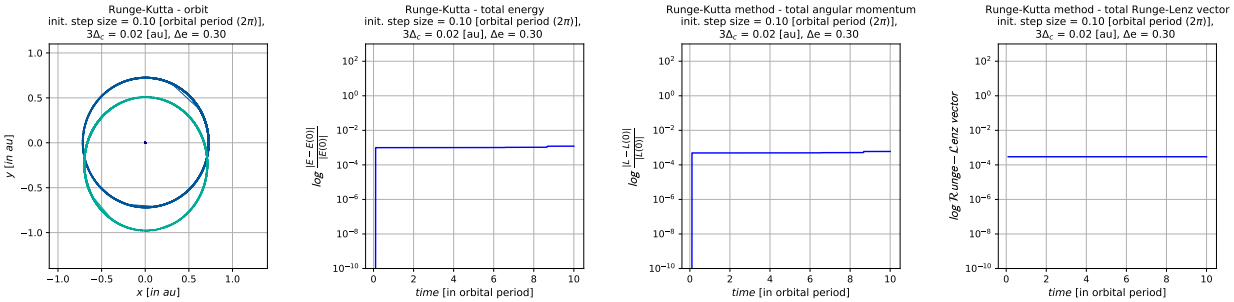
mean total energy: -1.121914834039176e-09
mean total angular momentum: 7.948732947214029e-08

steps: 282, min. step [in days]: 2.1672832839260807, max. step [in days]: 33.61910961160592
period of inner orbit [in days]: 224.701



mean total energy: -1.0999692705428207e-09
mean total angular momentum: 7.996099899554504e-08

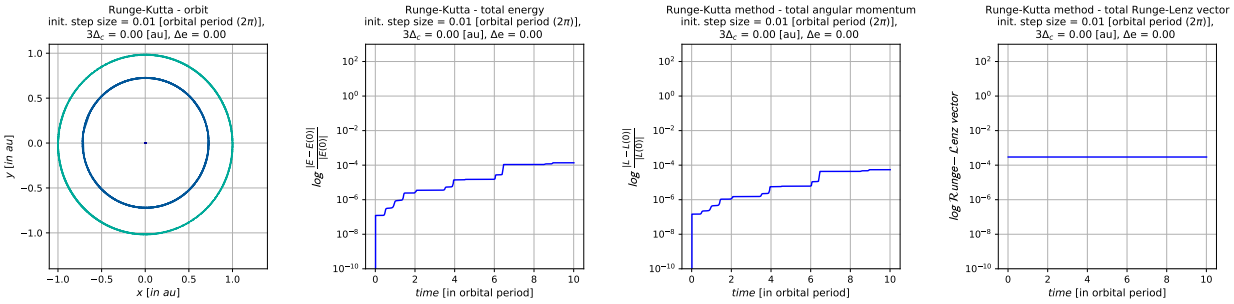
steps: 1445, min. step [in days]: 0.300911526450354, max. step [in days]: 22.470100000000002
period of inner orbit [in days]: 224.701



mean total energy: -1.099423157646272e-09
mean total angular momentum: 7.799327928399651e-08

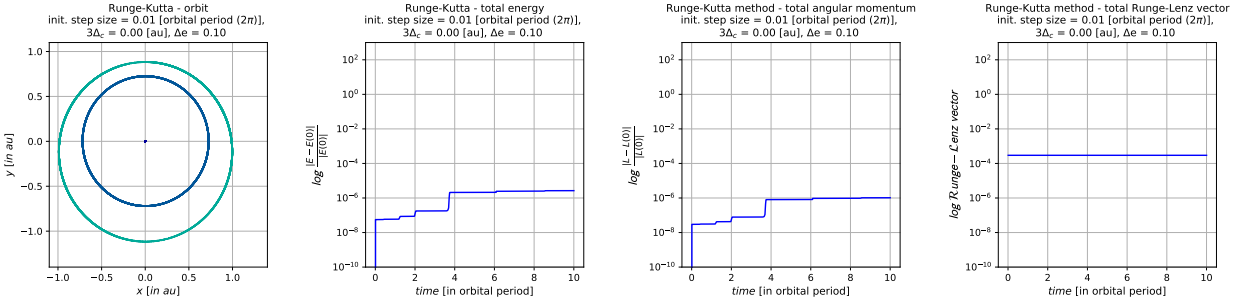
steps: 4600, min. step [in days]: 0.026402414247045685, max. step [in days]: 22.470100000000002
period of inner orbit [in days]: 224.701

- Plots for a step-size s = 0.01 and a eccentricity $\Delta e \in \{0.00, 0.10, 0.30\}$ for Runge-Kutta of order 4 with an adaptive step size implemented by eq. (2).



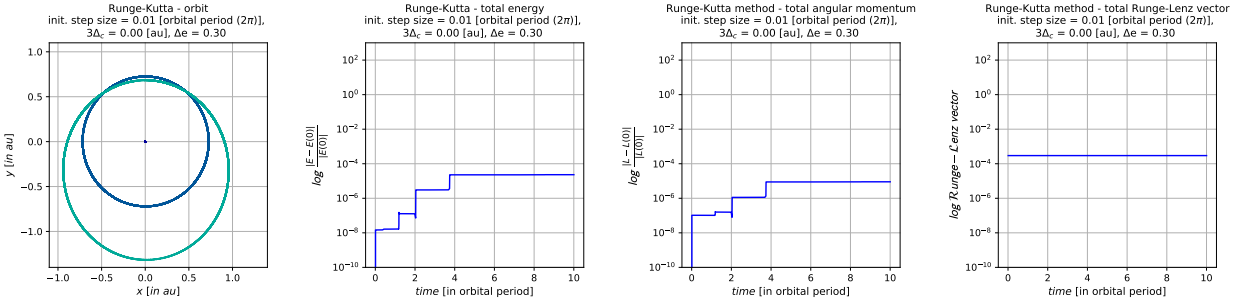
mean total energy: -9.4513136642180 85e-10
mean total angular momentum: 8.742644603737261e-08

steps: 1421, min. step [in days]: 0.47740976690768944, max. step [in days]: 13.337764023873346
period of inner orbit [in days]: 224.701



mean total energy: -9.450885167848324e-10
mean total angular momentum: 8.708225126979101e-08

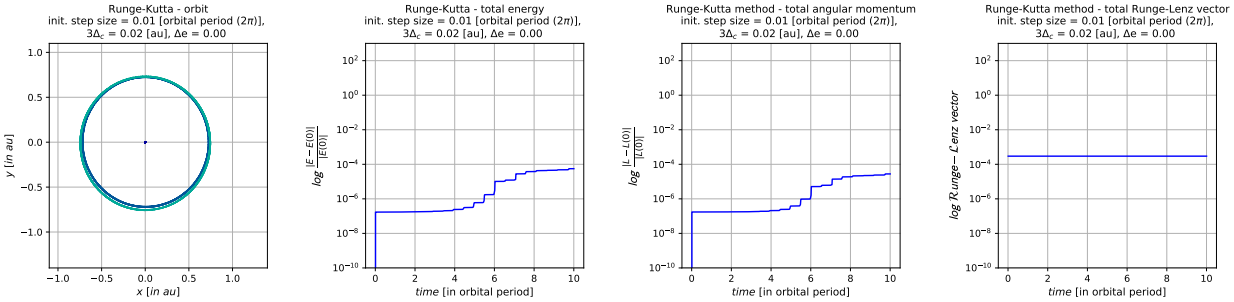
steps: 7109, min. step [in days]: 0.07176164353286535, max. step [in days]: 6.974259215264113
period of inner orbit [in days]: 224.701



mean total energy: -9.451012210820493e-10
mean total angular momentum: 8.4775993229439e-08

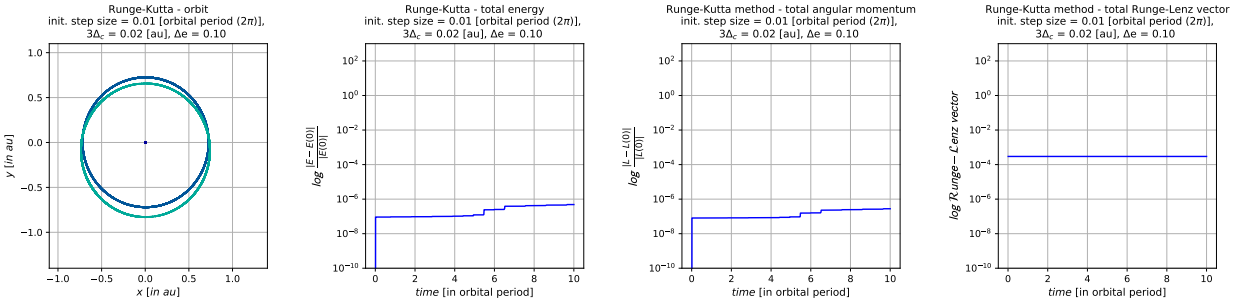
steps: 27440, min. step [in days]: 0.004788185895728438, max. step [in days]: 11.691677020004509
period of inner orbit [in days]: 224.701

- Plots for a step-size s = 0.01 and a eccentricity Δe $\in \{0.00, 0.10, 0.30\}$ for Runge-Kutta of order 4 with an adaptive step size implemented by eq. (2) and with a adapted semi-major axis a of the outer planet that differs by $3R_{Hill}$ from the inner planet. $3\Delta_c$ denotes the Hill radius shift.



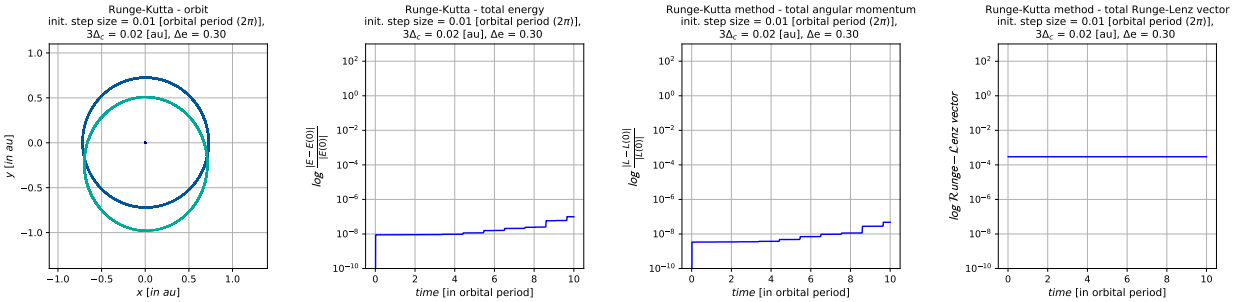
mean total energy: -1.0983157864377999e-09
mean total angular momentum: 8.031849024802796e-08

steps: 2336, min. step [in days]: 0.1997501051216557, max. step [in days]: 10.250336314555405
period of inner orbit [in days]: 224.701



mean total energy: -1.0983081140032286e-09
mean total angular momentum: 8.002059778850846e-08

steps: 15561, min. step [in days]: 0.014754613618203977, max. step [in days]: 4.916125099139917
period of inner orbit [in days]: 224.701



mean total energy: -1.0983044243419892e-09
mean total angular momentum: 7.803242430307483e-08

steps: 47675, min. step [in days]: 0.0028355870993996054, max. step [in days]: 4.121796648532125
period of inner orbit [in days]: 224.701

Python code:

```
# -*- coding: utf-8 -*-
"""
@author: Dawid Stepanovic
"""
from matplotlib import pyplot as plt
import numpy as np
import coconv as con
from numpy.linalg import norm
np.set_printoptions(precision=20)

# sidereal orbit parameter retrieved from: https://astronomy.swin.edu.au/cosmos/s/Sidereal+Period
solar       = 1.988544*10**30                   # in kg
jup         = 1898.13*10**24                    # in kg
ear         = 5.97219*10**24                    # in kg
sat         = 568.319*10**24                    # in kg
ven         = 4.86732*10**24                    # in kg

# parameter for assignment 2a:
venus      = np.array((0.72333199,0.00677323,0.,0.,0.,0.,ven/solar,224.701))
earth      = np.array((1.00000011,0.01671022,0.,0.,180.,0.,ear/solar,365.2568983263281))
jupiter    = np.array((5.20336301,0.04839266,0.,0.,0.,0.,jup/solar,4332.589))
saturn      = np.array((9.53707032,0.05415060,0.,0.,180.,0.,sat/solar,10759.22))

# parameter for assignment 2b:
#venus       = np.array((0.72333199,0.00677323,3.39471,76.68069,131.53298,181.97973,ven/solar,224.701))
#earth       = np.array((1.00000011,0.01671022,0.00005,-11.26064,102.94719,100.46435,ear/solar
    ,365.2568983263281))
#jupiter     = np.array((5.20336301,0.04839266,1.30530,100.55615,14.75385,34.40438,jup/solar,4332.589))
#saturn     = np.array((9.53707032,0.05415060,2.48446,113.71504,92.43194,49.94432,sat/solar,10759.22))

method      = "rk4"                             # implemented methods: "rk4", "rk4_acc", "rk4_vel", "
    rk45_fehlberg"
nbody       = 3                                 # number of bodies
planet      = np.zeros((nbody-1,8))             # define storage for two planets since the sun is in
    the program include
planet[0]   = venus                             # venus parameter
planet[1]   = earth                             # earth parameter
period      = planet[0,-1]                      # sidereal orbit period
step_val    = 0.01                              # initial step size
norbit_val = 10                                 # number of orbits
hillradius = "hill"                             # implemented methods: "hill" (Hill radius), "laplace"
    (Laplace radius), False (neither of both)
shift_hill = 1                                  # 0 = change radius for inner planet (Hill radius) 1 =
    change radius for outer planet (Hill radius)
e_val       = 0.                                # eccentricity
shift_eval = 1                                  # 0 = change ecc. for inner planet, 1 = change ecc. for
     outer planet
simulation = False                              # de/activating simulation
epsilon = 1e-6                                  # accuracy for the Runge-Kutta-Fehlberg method

def Nbodyproblem(method,nbody,planet,period,step_val,norbit_val,hillradius,shift_hill,e_val,shift_eval
    ,epsilon,simulation):

    # general setting
    dt       = period*step_val                   # time step
    norbits = norbit_val                        # number of orbits
    au       = 1.49597870700*10**11              # in m
    G_SI     = 6.67408*10**-11                   # gravitational constant [in m^3/(kg*s^2)]
    daysec   = 86400.                            # in seconds
    msun     = 1.                                # mass [in solar mass]
    sun      = np.array((0,0,0,0,0,0,1))

    # convert the gravitational constant
    G_new = G_SI/au**3*solar*(daysec)**2
    #G_new = np.square(2*np.pi/365.2568983263281)

    # Hill/Laplace radius
    if hillradius == "laplace":
        radius = planet[0,0]*(planet[0,-2])**(2/5)
        radius = 3*radius
        planet[shift_hill,0] = planet[0,0]+radius
    elif hillradius == "hill":
        radius = planet[0,0]*(planet[0,-2]/3)**(1/3)
        radius = 3*radius
        planet[shift_hill,0] = planet[0,0]+radius
    else:
        radius = 0;
        planet[shift_hill,0] = planet[shift_hill,0]+radius

    # shift eccentricity
    if e_val != 0:
        planet[shift_eval,1] = planet[shift_eval,1]+e_val

    # variable setting
    steps = int(100*period*norbits) #int(norbits*(period/dt))
    t = np.zeros(steps); dtstore = np.zeros(steps); energy = np.zeros(steps); moment  = np.zeros(steps
    )
    rungelenz = np.zeros(steps); a_time = np.zeros(3)

    # r array stores the steps, the number of bodies in the system and the (x,y,z) array
    r  = np.zeros((steps,nbody,3))
    v  = np.zeros_like(r)
    r0 = np.zeros_like(r)
    r1 = np.zeros_like(r)
    v0 = np.zeros_like(r)
    v1 = np.zeros_like(r)
```

```
89
90      # gravitational constant*mass
91      m = []
92      m.append(G_new*msun)
93      for i in range(np.shape(planet)[0]):
94           m.append(G_new*planet[i,-2])
95      m = np.array(m)
96
97      # initial position and velocity
98      for i in range(np.shape(planet)[0]):
99              r[0,i+1,:] = con.kepToCart(planet[i,:-1],sun[-1])[0];
100             v[0,i+1,:] = con.kepToCart(planet[i,:-1],sun[-1])[1];
101
102     # acceleration for j != i
103     def acceleration(i,r):
104       a      = np.zeros_like(r[0])
105       r_ij   = np.zeros_like(r[0])
106       l = [k for k in range(nbody) if k != i]
107       for j in l:
108         r_ij = r[j]-r[i]
109         a += m[j]/(norm(r_ij))**3*r_ij
110       return a
111
112     # jerk for j != i
113     def acceleration_dot(i,r,v):
114       a_dot = np.zeros_like(r[0])
115       r_ij  = np.zeros_like(r[0])
116       v_ij  = np.zeros_like(r[0])
117       l = [k for k in range(nbody) if k != i]
118       for j in l:
119         r_ij = r[j]-r[i]
120         v_ij = v[j]-v[i]
121         a_dot += m[j]/(norm(r_ij))**3*v_ij-(3*np.dot(r_ij,v_ij)/(norm(r_ij))**5*r_ij)
122       return a_dot
123
124     # total (kinetic + potential) energy
125     def calc_energy(r,v):
126       kin = 0.;pot = 0.
127       r_ij  = np.zeros_like(r[0])
128       for i in range(nbody):
129         kin += 1/2*m[i]*(norm(v[i]))**2)
130         l = [k for k in range(nbody) if k > i]
131         for j in l:
132           r_ij = r[j]-r[i]
133           pot  -= (m[i]*m[j])/norm(r_ij)
134       return (kin+pot)/G_new
135
136     # total angular momentum
137     def calc_moment(r,v):
138       L = np.zeros_like(r[0])
139       for i in range(nbody):
140         L += m[i]/G_new*np.cross(r[i],v[i])
141       return norm(L)
142
143     # Runge-Lenz vector
144     def diff(i,vr):
145       vr_ij  = np.zeros_like(vr[0])
146       l = [k for k in range(nbody) if k != i]
147       for j in l:
148         vr_ij += vr[j]-vr[i]
149       return vr_ij
150
151     # min Runge-Lenz vector
152     def diffmin(i,vr):
153       vr_ij  = np.zeros(nbody)
154       l = [k for k in range(nbody) if k != i]
155       for j in l:
156         vr_ij[j] = norm(vr[j]-vr[i])
157       return vr_ij
158
159     # Runge-Lenz vector
160     def calc_rungelenz(r,v):
161         RL = np.zeros_like(r[0])
162         L  = np.zeros_like(r[0])
163         for i in range(nbody):
164             L = m[i]/G_new*np.cross(r[i],v[i])
165             RL += np.cross(L,diff(i,v))-m[i]*diff(i,r)/norm(diff(i,r))
166         return norm(RL)
167
168     # Barycenter
169     def barycenter(r,v):
170         rbar = np.zeros(3)
171         vbar = np.zeros(3)
172         for i in range(nbody):
173             rbar += m[i]*r[i]
174             vbar += m[i]*v[i]
175         for i in range(nbody):
176                 r[i] = r[i]-(rbar/G_new)/np.sum(m/G_new)
177                 v[i] = v[i]-(vbar/G_new)/np.sum(m/G_new)
178         return rbar, vbar
179
180     # initial total energy
181     energy[0] = calc_energy(r[0],v[0])
182
183     # initial angular momentum
184     moment[0] = calc_moment(r[0],v[0])
185
```

```
    # initial Runge-Lenz vector
    rungelenz[0] = calc_rungelenz(r[0],v[0])

    # general settings for the RK-methods
    n = 0
    t[n] = dt
    dtstore[n] = dt


    k1 = np.zeros((nbody,6))
    k2 = np.zeros_like(k1)
    k3 = np.zeros_like(k1)
    k4 = np.zeros_like(k1)
    k5 = np.zeros_like(k1)
    k6 = np.zeros_like(k1)
    delta_r = np.zeros(nbody)
    delta_v = np.zeros(nbody)

    # initial barycenter values
    barycenter(r[0],v[0])

    if(method == "rk4"):
        while(t[n]<period*norbits):
            for i in range(nbody):
                k1[:,0:3] = dt*v[n,:]
                k1[i,3:6] = dt* acceleration(i,r[n])

                k2[:,0:3] = dt*(v[n,:]+0.5*k1[:,3:6])
                k2[i,3:6] = dt*acceleration(i,r[n]+0.5*k1[:,0:3])

                k3[:,0:3] = dt*(v[n,:]+0.5*k2[:,3:6])
                k3[i,3:6] = dt*acceleration(i,r[n]+0.5*k2[:,0:3])

                k4[:,0:3] = dt*(v[n,:]+k3[:,3:6])
                k4[i,3:6] = dt*acceleration(i,r[n]+k3[:,0:3])

                # next acceleration and position
                r[n+1,i]  = r[n,i]+1/6*(k1[i,0:3]+2*k2[i,0:3]+2*k3[i,0:3]+k4[i,0:3])
                v[n+1,i]  = v[n,i]+1/6*(k1[i,3:6]+2*k2[i,3:6]+2*k3[i,3:6]+k4[i,3:6])

            # barycenter
            barycenter(r[n+1],v[n+1])
            # compute total energy of system
            energy[n+1] = calc_energy(r[n+1],v[n+1])
            # compute angular momentum of system
            moment[n+1] = calc_moment(r[n+1],v[n+1])
            # compute Runge-Lenz vector
            rungelenz[n+1] = calc_rungelenz(r[n+1],v[n+1])
            # time step
            t[n+1] = t[n]+dt
            dtstore[n+1] = dt
            n = n+1

    elif(method == "rk4_vel"):
        while(t[n]<period*norbits):
            for i in range(nbody):
                k1[:,0:3] = dt*v[n,:]
                k1[i,3:6] = dt* acceleration(i,r[n])

                k2[:,0:3] = dt*(v[n,:]+0.5*k1[:,3:6])
                k2[i,3:6] = dt*acceleration(i,r[n]+0.5*k1[:,0:3])

                k3[:,0:3] = dt*(v[n,:]+0.5*k2[:,3:6])
                k3[i,3:6] = dt*acceleration(i,r[n]+0.5*k2[:,0:3])

                k4[:,0:3] = dt*(v[n,:]+k3[:,3:6])
                k4[i,3:6] = dt*acceleration(i,r[n]+k3[:,0:3])

                # next acceleration and position
                r[n+1,i]  = r[n,i]+1/6*(k1[i,0:3]+2*k2[i,0:3]+2*k3[i,0:3]+k4[i,0:3])
                v[n+1,i]  = v[n,i]+1/6*(k1[i,3:6]+2*k2[i,3:6]+2*k3[i,3:6]+k4[i,3:6])

            # barycenter
            barycenter(r[n+1],v[n+1])
            # compute total energy of system
            energy[n+1] = calc_energy(r[n+1],v[n+1])
            # compute angular momentum of system
            moment[n+1] = calc_moment(r[n+1],v[n+1])
            # compute Runge-Lenz vector
            rungelenz[n+1] = calc_rungelenz(r[n+1],v[n+1])
            # time step
            for i in range(nbody):
                a_time[i] = norm(diff(i,r[n+1]))/norm(diff(i,v[n+1]))

            multiplier = 10
            if((np.min(step_val*a_time[a_time > 0])*multiplier)/(t[n]-t[n-1])> 10):
                dt = dt
            else:
                dt = np.min(step_val*a_time[a_time > 0])*multiplier
            t[n+1] = t[n]+dt
            dtstore[n+1] = dt
            n = n+1

    elif(method == "rk4_acc"):
        while(t[n]<period*norbits):
            for i in range(nbody):
                k1[:,0:3] = dt*v[n,:]
```

```
                k1[i,3:6] = dt* acceleration(i,r[n])

                k2[:,0:3] = dt*(v[n,:]+0.5*k1[:,3:6])
                k2[i,3:6] = dt*acceleration(i,r[n]+0.5*k1[:,0:3])

                k3[:,0:3] = dt*(v[n,:]+0.5*k2[:,3:6])
                k3[i,3:6] = dt*acceleration(i,r[n]+0.5*k2[:,0:3])

                k4[:,0:3] = dt*(v[n,:]+k3[:,3:6])
                k4[i,3:6] = dt*acceleration(i,r[n]+k3[:,0:3])

                # next acceleration and position
                r[n+1,i]  = r[n,i]+1/6*(k1[i,0:3]+2*k2[i,0:3]+2*k3[i,0:3]+k4[i,0:3])
                v[n+1,i]  = v[n,i]+1/6*(k1[i,3:6]+2*k2[i,3:6]+2*k3[i,3:6]+k4[i,3:6])

            # barycenter
            barycenter(r[n+1],v[n+1])
            # compute total energy of system
            energy[n+1] = calc_energy(r[n+1],v[n+1])
            # compute angular momentum of system
            moment[n+1] = calc_moment(r[n+1],v[n+1])
            # compute Runge-Lenz vector
            rungelenz[n+1] = calc_rungelenz(r[n+1],v[n+1])
            # time step
            for i in range(nbody):
                a_time[i] = norm(acceleration(i,r[n+1]))/norm(acceleration_dot(i,r[n+1],v[n+1]))

            multiplier = 1e8
            if((np.min(step_val*a_time[a_time > 0])*multiplier)/(t[n]-t[n-1])> 2):
                dt = dt
            else:
                dt = np.min(step_val*a_time[a_time > 0])*multiplier
            t[n+1] = t[n]+dt
            dtstore[n+1] = dt
            n = n+1

    elif(method == "rk45_fehlberg"):
        while(t[n]<period*norbits):
                for i in range(0,nbody):
                    k1[:,0:3]  = dt*v[n,:]
                    k1[i,3:6]  = dt* acceleration(i,r[n])

                    k2[:,0:3]  = dt*(v[n,:]+k1[:,3:6]/4)
                    k2[i,3:6]  = dt*acceleration(i,r[n]+k1[:,0:3]/4)

                    k3[:,0:3]  = dt*(v[n,:]+3/32*k1[:,3:6]+9/32*k2[:,3:6])
                    k3[i,3:6]  = dt*acceleration(i,r[n]+3/32*k1[:,0:3]+9/32*k1[:,0:3])

                    k4[:,0:3]  = dt*(v[n,:]+1932/2197*k1[:,3:6]-7200/2197*k2[:,3:6]+7296/2197*k3
[:,3:6])
                    k4[i,3:6]  = dt*acceleration(i,r[n]+1932/2197*k1[:,0:3]-7200/2197*k2
[:,0:3]+7296/2197*k3[:,0:3])

                    k5[:,0:3]  = dt*(v[n,:]+439/216*k1[:,3:6]-8*k2[:,3:6]+3680/513*k3[:,3:6]-845/4104*
k4[:,3:6])
                    k5[i,3:6]  = dt*acceleration(i,r[n]+439/216*k1[:,0:3]-8*k2[:,0:3]+3680/513*k3
[:,0:3]-845/4104*k4[:,0:3])

                    k6[:,0:3]  = dt*(v[n,:]-8/27*k1[:,3:6]+2*k2[:,3:6]-3544/2565*k3[:,3:6]+1859/4104*
k4[:,3:6]-11/40*k4[:,3:6])
                    k6[i,3:6]  = dt*acceleration(i,r[n]-8/27*k1[:,0:3]+2*k2[:,0:3]-3544/2565*k3
[:,0:3]+1859/4104*k4[:,0:3]-11/40*k4[:,0:3])

                    # next acceleration and position
                    r0[n+1,i]  = r[n,i]+25/216*k1[i,0:3]+1408/2565*k3[i,0:3]+2197/4104*k4[i,0:3]-k5[i
,0:3]/5
                    v0[n+1,i]  = v[n,i]+25/216*k1[i,3:6]+1408/2565*k3[i,3:6]+2197/4104*k4[i,3:6]-k5[i
,3:6]/5

                    r1[n+1,i]  = r[n,i]+16/135*k1[i,0:3]+6656/12825*k3[i,0:3]+28561/56430*k4[i
,0:3]-9/50*k5[i,0:3]+2/55*k5[i,0:3]
                    v1[n+1,i]  = v[n,i]+16/135*k1[i,3:6]+6656/12825*k3[i,3:6]+28561/56430*k4[i
,3:6]-9/50*k5[i,3:6]+2/55*k5[i,3:6]

                    delta_r[i] = 0.84*((epsilon*dt)/norm(r0[n+1,i]-r1[n+1,i]))**(1/4)
                    delta_v[i] = 0.84*((epsilon*dt)/norm(v0[n+1,i]-v1[n+1,i]))**(1/4)
                    r[n+1,i]   = r0[n+1,i]
                    v[n+1,i]   = v0[n+1,i]

                # time step
                if(norm(v0[n+1,i]-v1[n+1,i])/dt <= epsilon):
                    r[n+1,i] = r0[n+1,i]
                    v[n+1,i] = v0[n+1,i]
                    barycenter(r[n+1],v[n+1])
                    t[n+1] = t[n]+dt
                    dtstore[n+1] = dt
                    n = n+1
                    dt = np.min(delta_v[1:nbody])*dt
                else:
                    dt = np.min(delta_v[1:nbody])*dt

                # compute total energy of system
                energy[n] = calc_energy(r[n],v[n])
                # compute angular momentum of system
                moment[n] = calc_moment(r[n],v[n])
                # compute Runge-Lenz vector
                rungelenz[n] = calc_rungelenz(r[n],v[n])
```

```
370
371     # extract relevant non-zero values
372     r=r[:n+1]
373     t=t[:n+1]
374     dtstore=dtstore[:n+1]
375     energy=energy[:n+1]
376     moment=moment[:n+1,]
377     rungelenz=rungelenz[:n+1]
378
379     # exit the simulation by closing the figure
380     def handle_close(evt):
381         raise SystemExit('Closed figure, exit program.')
382
383     # simulation
384     if simulation:
385         fig = plt.figure(figsize=(5,5))
386         fig.canvas.mpl_connect('close_event', handle_close)
387
388         for m in range(n):
389                 plt.cla()
390                 plt.title(r"Runge-Kutta method - orbit""\n"\
391                           "step size = %1.2f [orbital period (2$\pi$)], " %dt +\
392                           "\n""$3\Delta_c$ = %1.2f [au], " %radius + "$\Delta$e =\
393                           %1.2f " %e_val,fontsize=11)
394                 plt.xlabel(r"$x~[in~au]$",fontsize=11)
395                 plt.ylabel(r"$y~[in~au]$",fontsize=11)
396                 plt.grid()
397                 plt.scatter(r[1:m,1,0],r[1:m,1,1],s=1,color='blue')
398                 plt.scatter(r[1:m,1,0],r[1:m,1,1],s=1,color='blue')
399                 plt.scatter(r[1:m,2,0],r[1:m,2,1],s=1,color='blue')
400                 plt.scatter(r[1:m,2,0],r[1:m,2,1],s=1,color='blue')
401                 plt.scatter(r[m,0,0],r[m,0,1],s=40,color='yellow')
402                 plt.scatter(r[m,1,0],r[m,1,1],s=40,color='darkblue')
403                 plt.scatter(r[m,2,0],r[m,2,1],s=40,color='darkblue')
404                 plt.pause(0.00000000001)
405     else:
406         # plots
407         # figure 1 - orbit
408         plt.figure(figsize=(22,5))
409         plt.subplots_adjust(wspace = 0.45,left=1/9, right=1-1/9, bottom=1/4.8, top=1-1/7.5)
410         plt.subplot(141)
411         for i in range(nbody):
412             plt.plot(r[:,i,0],r[:,i,1],color=(0,i/nbody,0.6,1),linewidth=.5,\
413                      marker='h',markersize=1, markevery=1)
414         plt.title(r"Runge-Kutta - orbit""\n"\
415                   "init. step size = %1.2f [orbital period (2$\pi$)], " %step_val +\
416                   "\n""$3\Delta_c$ = %1.2f [au], " %radius + "$\Delta$e = %1.2f " %e_val,fontsize=11)
417         plt.xlabel(r"$x~[in~au]$",fontsize=11)
418         plt.ylabel(r"$y~[in~au]$",fontsize=11)
419         #plt.ylim([-.02,.02])
420         #plt.xlim([-.02,.02])
421         plt.ylim([-1.4,1.1])
422         plt.xlim([-1.1,1.4])
423         plt.grid()
424
425         col = 'dimgrey'
426         plt.annotate('mean total energy: '+str(np.mean(energy))\
427                      , xy=(0.0, -0.25), xycoords='axes fraction',color=col)
428         plt.annotate('mean total angular momentum: '+str(np.mean(moment))\
429                      , xy=(0.0, -0.3), xycoords='axes fraction',color=col)
430         plt.annotate('steps: '+str(n)+ ', min. step [in days]: '+str(np.min(dtstore))+\
431                      ', max. step [in days]: '+str(np.max(dtstore)),\
432                     xy=(1.5, -0.25), xycoords='axes fraction',color=col)
433         plt.annotate('period of inner orbit [in days]: '+str(period)\
434                      , xy=(1.5, -0.3), xycoords='axes fraction',color=col)
435
436         # figure 2 - total energy
437         plt.subplot(142)
438         plt.semilogy(t/period,abs(energy-energy[0])/abs(energy[0]),'k',color='b')
439         plt.title(r"Runge-Kutta - total energy""\n"\
440                   "init. step size = %1.2f [orbital period (2$\pi$)], " %step_val +\
441                  "\n""$3\Delta_c$ = %1.2f [au], " %radius + "$\Delta$e = %1.2f " %e_val,fontsize=11)
442         plt.xlabel(r"$time~[{\rm in~orbital~period}]$",fontsize=11)
443         plt.ylabel(r"$\mathcal{log}~\frac{|E-E(0)|}{|E(0)|}$",fontsize=13.5)
444         plt.ylim([1e-10,1000])
445         plt.grid()
446
447         # figure 3 - total angular momentum
448         plt.subplot(143)
449         plt.semilogy(t/period,abs(moment-moment[0])/abs(moment[0]),'k',color='b')
450         plt.title(r"Runge-Kutta method - total angular momentum""\n"\
451                   "init. step size = %1.2f [orbital period (2$\pi$)], " %step_val +\
452                  "\n""$3\Delta_c$ = %1.2f [au], " %radius + "$\Delta$e = %1.2f " %e_val,fontsize=11)
453         plt.xlabel(r"$time~[{\rm in~orbital~period}]$",fontsize=11)
454         plt.ylabel(r"$\mathcal{log}~\frac{|L-L(0)|}{|L(0)|}$",fontsize=13.5)
455         plt.ylim([1e-10,1000])
456         plt.grid()
457
458         # figure 4 - total Runge-Lenz vector
459         plt.subplot(144)
460         plt.semilogy(t/period,rungelenz,'k',color='b')
461         plt.title(r"Runge-Kutta method - total Runge-Lenz vector""\n"\
462                   "init. step size = %1.2f [orbital period (2$\pi$)], " %step_val +\
463                   "\n""$3\Delta_c$ = %1.2f [au], " %radius + "$\Delta$e = %1.2f " %e_val,fontsize=11)
464         plt.xlabel(r"$time~[{\rm in~orbital~period}]$",fontsize=11)
465         plt.ylabel(r"$\mathcal{log~Runge-Lenz~vector}$",fontsize=13.5)
466         plt.ylim([1e-10,1000])
```

```
467             plt.grid()
468
469             # save figures as pdf
470             plt.savefig("rungekutta_"+str(step_val)+"_"+str(e_val)+"_"+str(hillradius)+"_"+str(method)+".
        pdf")
471             plt.show()
472
473 Nbodyproblem(method,nbody,planet,period,step_val,norbit_val,hillradius,shift_hill,e_val,shift_eval,
        epsilon,simulation)
```

Python code 1: N-body