040692 PR Practical in numerical Astronomy
Dr. Elke Pilat-Lohinger & Maximilian Zimmermann, MSc

Two-body problem

Dawid Stepanovic

December 30, 2021

Two-body problem:

Write a program for a two-body problem (in Python, Fortran or C/C++) and use as numerical method the **leap frog method**. Calculate various orbits of a planet and consider the **Sun** ($m_{Sun} = 1$) and a **Jupiter-mass planet** ($m_{Jupiter} \sim 0.001\ m_{Sun}$). The planet orbits the Sun at a **semi-major axis** with $a = 1$ au (astronomical units). Consider different **eccentricities e**, i.e. e $\in \{0.00, 0.20, 0.50\}$ and test different **step-size s** for the integration, i.e. s $\in \{0.01, 0.05, 0.10\}$. Implement the two-body problem for a computation time of **10 periods** and study the **constants of motion** (Energy, angular momentum).

The initial velocity of the more massive body (Sun) is zero, and the less massive body (Jupiter) is located at its apoapsis (the point of greatest distance form the Sun). There, the location and the velocity are given by the vis-viva-equation:

$$x = a(1+e) \qquad y = 0$$

$$v_x = 0 \qquad v_y = \sqrt{\frac{G(m_1+m_2)}{a}\frac{1-e}{1+e}}$$

The code units are scaled in a way such that one period is 2π and where the Gauß gravitational constant k is given by

$$k = \sqrt{G}$$

$$k = 0.01720209895 \frac{1}{\text{day[in sec]}}\sqrt{\frac{au^3}{m_{Sun}}} = \frac{2\pi}{\text{y [in days]}}\frac{1}{\text{d [in sec]}}\sqrt{\frac{au^3}{m_{Sun}}},$$

where G is the gravitational constant, au is the astronomical unit, m_{Sun} is the solar mass, d is the mean solar day [in sec] and y is the mean solar year [in days].

Leap-frog method:

The leap-frog method is in general described as follows:

$$\begin{aligned}
&\text{First step:}\\
&r_{1/2} = r_0 + v_0\frac{\Delta t}{2},\\
&a_{1/2} = a(t_{1/2}, r_{1/2})\\
&\text{Regular steps:}\\
&v_{n+1} = v_n + a_{n+1/2}\Delta t,\\
&r_{n+3/2} = r_{n+1/2} + v_{n+1}\Delta t\\
&\text{Last step:}\\
&r_{n+1} = r_{n+1/2} + v_{n+1}\frac{\Delta t}{2},
\end{aligned}$$

where $t_{n+1/2} = t_n + \frac{\Delta t}{2}$.

To implement the leap-frog method we have modified the indices and shift the velocity by a 1/2 step from v_n to $v_{n-1/2}$. Thus we use a Runge-Kutta scheme of the second order to initialize the leap-frog method. The initial step is a first order ODE problem of the form

$$\frac{dr(t)}{dt} = v(r(t), t),$$

where at t_0, $r(t_0) = 0$. By using the Runge-Kutta scheme of the second order we get the intermediate estimate of the velocity v(t) at $t = t + \frac{\Delta t}{2}$. The Runge-Kutta of the second order is defined as

$$\begin{aligned}
k_1 &= v(r(t_0), t_0), && \text{estimate of derivative at } t_0,\\
r_1\big(t + \frac{\Delta t}{2}\big) &= r(t_0) + k_1\frac{\Delta t}{2}, && \text{intermediate estimate at } t = t_0 + \frac{\Delta t}{2},\\
k_2 &= v\big(r_1\big(t + \frac{\Delta t}{2}\big), t_0 + \frac{\Delta t}{2}\big), && \text{estimate of slope at } t = t_0 + \frac{\Delta t}{2},\\
r(t_0 + \Delta t) &= v(t_0) + k_2\Delta t, && \text{estimate of } r(t_0 + \Delta t).
\end{aligned}$$

To the shift velocity from v_n to $v_{n-1/2}$ we need to evaluate k_2, where $v(r(t_0), t_0)$ is the initial velocity such that

$$k_2 = v\big(r_1\big(t - \frac{\Delta t}{2}\big), t - \frac{\Delta t}{2}\big) = v(r(t), t) - a\big(r_1\big(t - \frac{\Delta t}{2}\big)\big), t - \frac{\Delta t}{2}\big)\frac{\Delta t}{2},$$

for $t = t_0$. Translate the above derivation in an algorithmic language we basically use the following relation,

$$v_{n+1/2} = v_n + a_{n+1/2}\frac{\Delta t}{2} \Longleftrightarrow v_{n-1/2} = v_n - a_{n-1/2}\frac{\Delta t}{2}.$$

To compute the total energy E(t) and the total angular momentum L(t), we need to evaluate the velocities and the positions at the same time. Using a drift-kick-drift leap-frog scheme (or kick-drift-kick leap-frog scheme) we update the position (or velocity) by a 1/2 step and calculate the total energy E and the total angular momentum L at same time step and complete the leap-frog step by updating the position (or velocity) by the remaining 1/2 step. In our case we have implemented the drift-kick-drift leap-frog scheme which is a re-arranged and synchronised form of the general scheme described above. The drift-kick-drift leap-frog scheme can written as

$$\begin{aligned} r_{n+1} &= r_n + v_i \Delta t + \frac{1}{2} a_n (\Delta t)^2, \\ v_{n+1} &= v_n + \frac{1}{2}(a_n + a_{n+1}) \Delta t, \end{aligned}$$

which can be re-arranged to

$$\begin{aligned} r_{n+1/2} &= r_n + v_{n+1/2} \frac{\Delta t}{2}, && \text{drift step,} \\ v_{n+1/2} &= v_{n-1/2} + a_{n+1/2} \Delta t, && \text{kick step,} \\ r_{n+1} &= r_{n+1/2} + v_{n+1/2} \frac{\Delta t}{2}, && \text{re-synchronize the drift step.} \end{aligned}$$

The kick-drift-kick leap-frog scheme can be written similarly by starting with an initial Euler kick step, then calculating the position drift and close by re-synchronizing the open kick step.

Period:

To scale the code in a way such that one period equals 2π we need to convert the gravitational constant G to 1 to describe the period in units of 2π such that G is unit-less and where the astronomical units au and the solar mass m_{Sun} and the time in day [in sec] equals 1. This can be achieved by

$$G_{new} = \left(\frac{G_{SI}}{au^3} \times \text{mean solar day[in sec]} \times m_{Sun} \right) / k^2 \approx 1,$$

where G_{SI} $(= 6.67408 \times 10^{-11})$ is gravitational constant in SI units and k $(= \frac{2\pi}{\text{mean solar year}} = \frac{2\pi}{365.2568983263281} = 0.01720209895)$ Gaussian gravitational constant.

Further remarks:

For m_i the mass and $\mathbf{r}_i(t)$ the vector of spatial coordinates depending on time of the i-th celestial body we define the velocity $\mathbf{v}(t)$ and acceleration $\mathbf{a}(t)$ of the i-th celestial body as follows

$$\frac{d\mathbf{r}_i(t)}{dt} = \mathbf{v}_i(t), \qquad \mathbf{a}_i(t) = \sum_{i \neq j}^{N} \frac{G m_j \mathbf{r}_{ij}(t)}{||\mathbf{r}_{ij}||_2^3},$$

where $\mathbf{r}_{ij}(t) = \mathbf{r}_i(t) - \mathbf{r}_j(t)$ and $i \in \{1, 2\}$. Additionally, we know from the lecture material that the conversation of total energy for a two-body problem is given by

$$\frac{dE(t)}{dt} = \frac{d}{dt} \left(\overbrace{\frac{m_1(\mathbf{v}_1(t))^2}{2} + \frac{m_2(\mathbf{v}_2(t))^2}{2}}^{\text{kinetic energy}} - \overbrace{\frac{G m_1 m_2}{||\mathbf{r}_{ij}||_2}}^{\text{potential energy}} \right) = 0,$$

and the conversation of the total angular-momentum for the system is defined as

$$\frac{dL}{dt} = \frac{d}{dt}\big(m_1\mathbf{r}_1(t) \times \mathbf{v}_1(t) + m_2\mathbf{r}_2(t) \times \mathbf{v}_2(t)\big) = 0.$$

The vector setting was chosen in a compact form which stores for every step that is given by the number of orbits times the integer value of 2π divided by the step size ($=$ norbits $\times$ int($\frac{\text{tperiod}}{\text{dt}}$)), and for every celestial body the coordinates in $\mathbb{R}^3$ (x,y,z). We have plotted the orbit in the x-y plane of the center-of-mass frame and we have plotted the relative error in the total energy $\big(\frac{|E-E(0)|}{|E(0)|}\big)$ versus time by period, and the relative error in the total angular momentum $\big(\frac{|L-L(0)|}{|L(0)|}\big)$ versus time by period for a eccentricity e $\in$ {0.00, 0.20, 0.50} and for a step-size s $\in$ {0.01, 0.05, 0.10}, which are illustrated down below. We calculated the specific orbital energy (or vis-viva energy) E_{spec} and the specific angular momentum L_{spec} of a two-body problem, which are defined as,

$$\begin{aligned} E_{spec} &= -\mu\frac{GM}{2a}, \\ L_{spec} &= \mu\sqrt{(1-e^2)GMa}, \end{aligned}$$

where $M(= m_1 + m_1)$ is the total mass and $\mu = \frac{m_{m2}}{m_1+m_2}$ is the reduced mass, to compare it with the calculated total energy and total angular momentum and to provide a sanity check. The results are reported below the plots for every variable specification. The calculated total energy and calculated total angular momentum is constant over time. However, since the calculation deviate because of the approximational character we calculate the mean total energy and the mean total angular momentum to make it comparable.

The plots show that the relative error of the total energy and the relative error of the total angular momentum is as expected constant over time. Nevertheless, we see that the leapfrog method becomes unstable in highly eccentric cases unless Δt is very small. The relative error behavior seems to be independent of the step-size and changes with a higher eccentricity. However, it evolves in regular, constant and similar patterns over different eccentricities and it only differs in magnitude, when changing the step-size. This changes can also be seen in the orbit plots where we observe an apsidal precession for larger step-sizes with higher eccentricities.

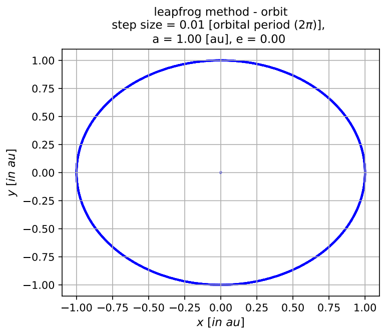
Program function:
The program function **twobodyproblem** has been initialized by the following parameters:

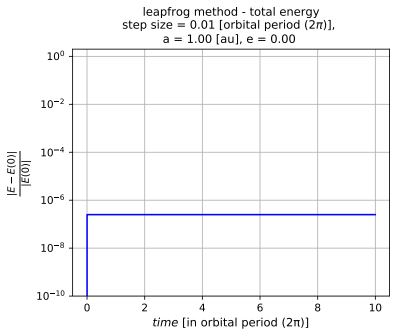
twobodyproblem(steps,e_value,a_val,norbit_val,simulation),

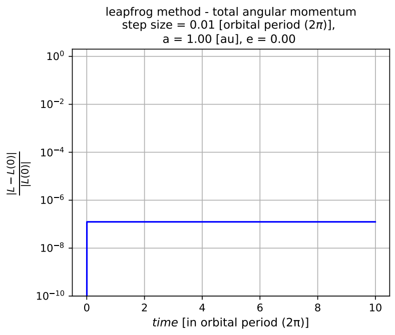
where **steps** (int) is the time step, **e_value** (int) is the eccentricity, **a_val** (int) is the semi-major axis [au], **norbit** (int) are the number of orbits and the **simulation** flag (boolean) starts a small orbital simulation.

Plots:

- Plots for a eccentricity e = 0.00 and a step-size s ∈ {0.01, 0.05, 0.10}

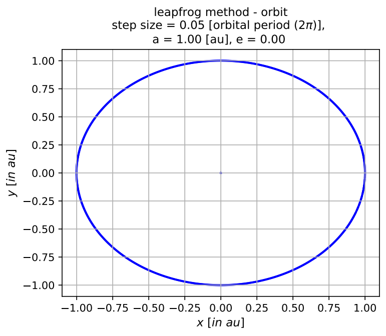


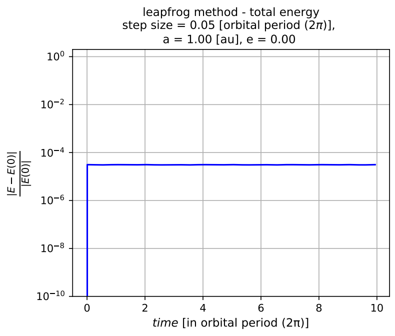


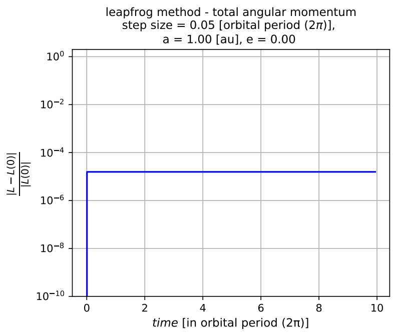


Mean total energy: -0.00047682727966982835
Specific orbital energy: -0.00047728274253704497

Mean total angular momentum: 0.0009550043696930709
Specific angular momentum: 0.0009540937754773542

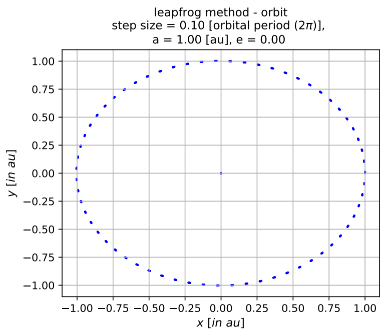


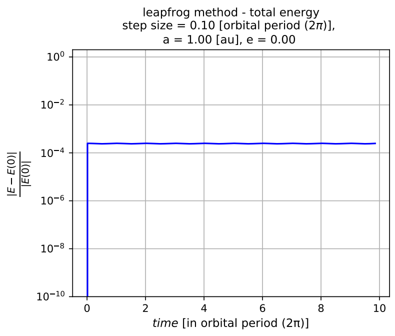


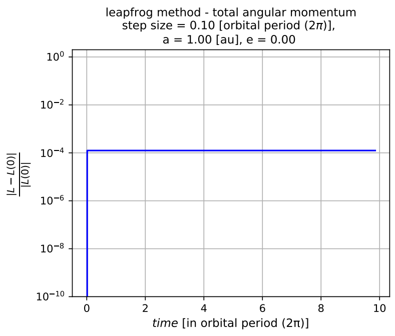


Mean total energy: -0.00047684186297776063
Specific orbital energy: -0.00047728274253704497

Mean total angular momentum: 0.0009549895785401992
Specific angular momentum: 0.0009540937754773542



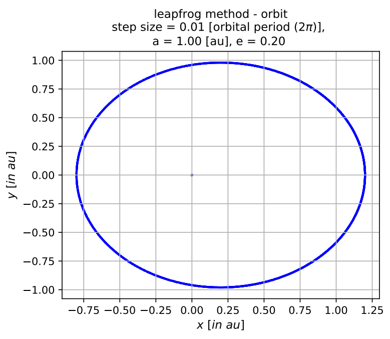


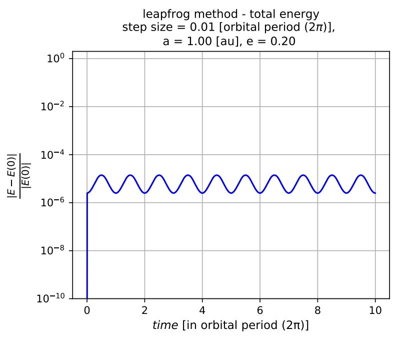


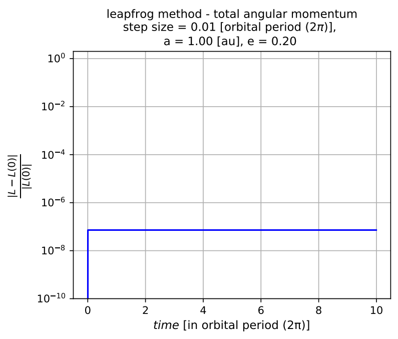
Mean total energy: -0.0004769432444303867
Specific orbital energy: -0.00047728274253704497

Mean total angular momentum: 0.0009548853030131586
Specific angular momentum: 0.0009540937754773542

- Plots for a eccentricity e = 0.20 and a step-size s ∈ {0.01, 0.05, 0.10}

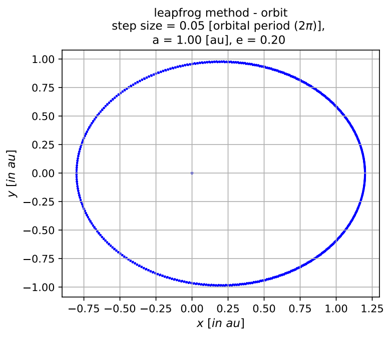


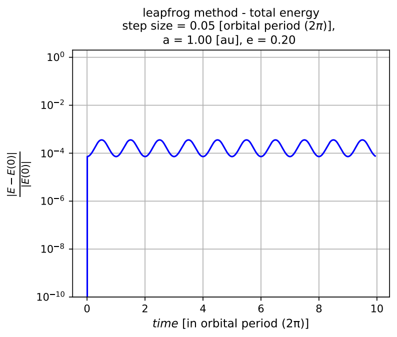


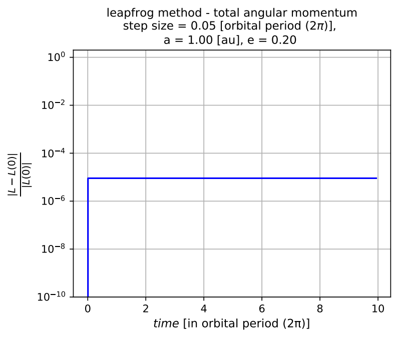


Mean total energy: -0.00047698241208874993
Specific orbital energy: -0.00047728274253704497

Mean total angular momentum: 0.0009357094124207466
Specific angular momentum: 0.0009348171666740224

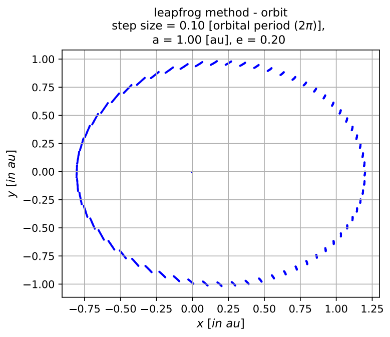


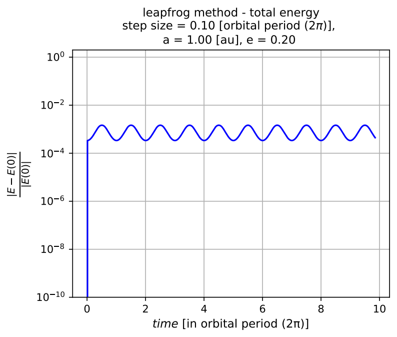


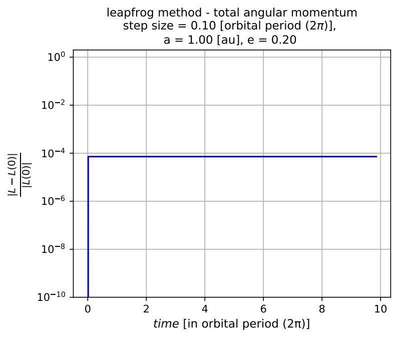


Mean total energy: -0.00047706844483920685
Specific orbital energy: -0.00047728274253704497

Mean total angular momentum: 0.0009357010256644191
Specific angular momentum: 0.0009348171666740224



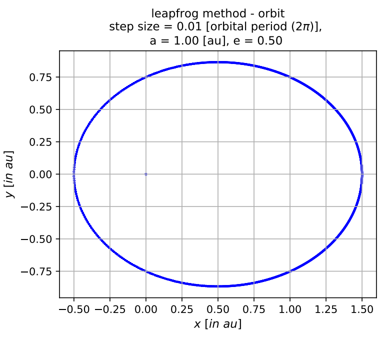


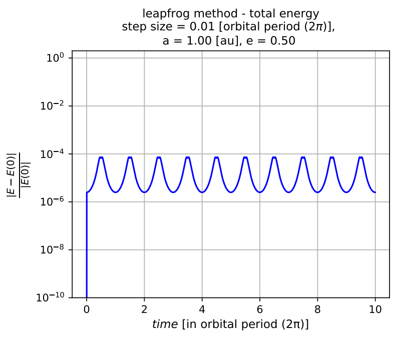


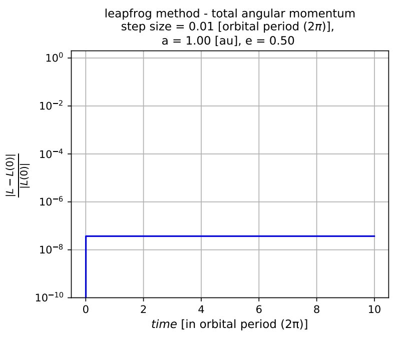
Mean total energy: -0.00047735890358424653
Specific orbital energy: -0.00047728274253704497

Mean total angular momentum: 0.0009356418999084442
Specific angular momentum: 0.0009348171666740224

- Plots for a eccentricity e = 0.50 and a step-size s $\in \{0.01, 0.05, 0.10\}$

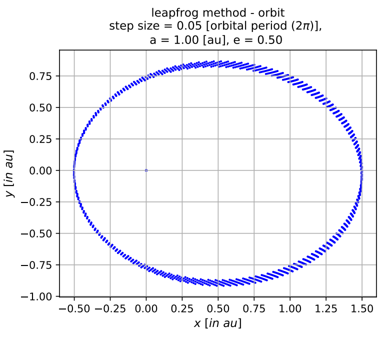


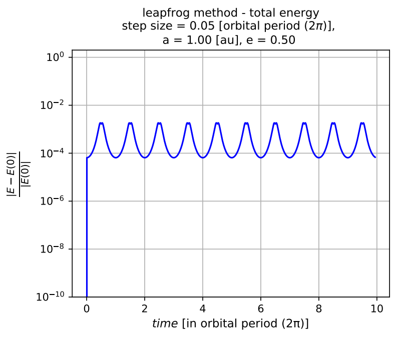


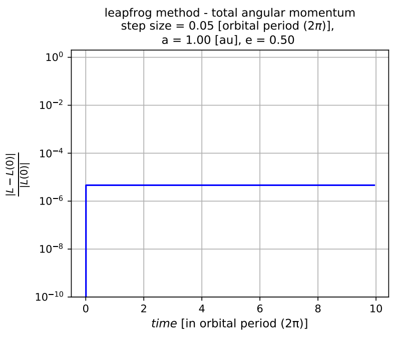


Mean total energy: -0.00047714008642819905
Specific orbital energy: -0.00047728274253704497

Mean total angular momentum: 0.0008270581176207549
Specific angular momentum: 0.0008262694471559953

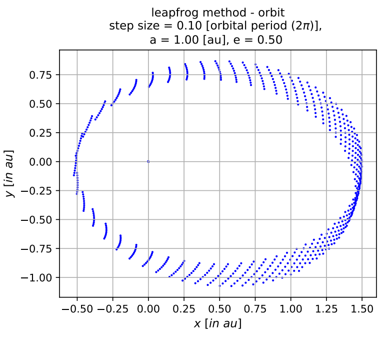


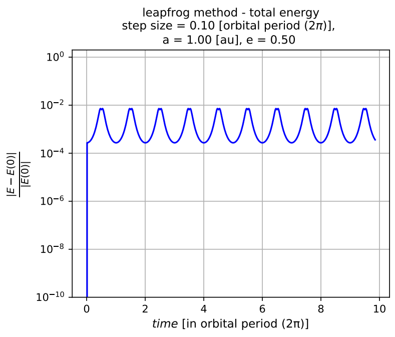


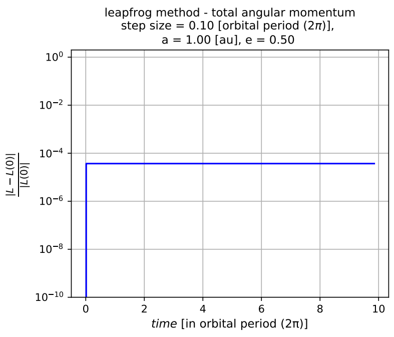


Mean total energy: -0.00047736217748455874
Specific orbital energy: -0.00047728274253704497

Mean total angular momentum: 0.0008270543222073277
Specific angular momentum: 0.0008262694471559953







Mean total energy: -0.00047805680506609195
Specific orbital energy: -0.00047728274253704497

Mean total angular momentum: 0.0008270275648570992
Specific angular momentum: 0.0008262694471559953

Python code:

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

def twobodyproblem(step_val,e_val,a_val,norbit_val,simulation=False):

    # general setting
    nbody   = 2                                # number of celestial objects
    dt      = step_val                         # time step
    e       = e_val                            # eccentricity
    a       = a_val                            # semi-major axis [au]
    norbits = norbit_val                       # number of orbits
    tperiod = 2*np.pi                          # orbital period
    solar   = 1.988544*10**30                  # in kg
    jup     = 1898.13*10**24                   # in kg
    au      = 1.49597870700*10**11             # in m
    G_SI    = 6.67408*10**-11                  # gravitational constant in m^3/(kg*s^2)
    k_const = (2*np.pi/365.2568983263281)      # 2pi/(year[in days])
    yearsec = 86400.*365.2568983263281         # in seconds
    daysec  = 86400.                           # in seconds
    earth   = 5.97219*10**24                   # in kg
    msun    = 1.                               # mass [in solar mass]
    mplanet = jup/solar                        # jupiter mass [in solar mass]

    # convert the gravitational constant G to 1 to scale the period to 2*pi
    G_new  = G_SI/au**3*solar*(daysec)**2/k_const**2
    #G_new = G_SI/au**3*solar*(yearsec)**2/(2*np.pi)**2

    m  = G_new*np.array([msun,mplanet])        # gravitational constant*mass
    r0 = a*(1+e)                               # initial position [au]
    v0 = np.sqrt(((m[0]+m[1])/a)*((1-e)/(1+e))) # initial velocity [au]

    # variable setting
    steps = norbits*int(tperiod/dt)
    t = np.zeros(steps); energy = np.zeros(steps); moment  = np.zeros(steps)
    # 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)

    # initial position and velocity
    r[0,1,0]  = r0;
    v[0,1,1]  = v0;

    # acceleration for j != i
    def acceleration(i,r):
      a     = np.zeros_like(r[0])
      r_ij  = np.zeros_like(r[0])
      l = [k for k in range(nbody) if k != i]
      for j in l:
        r_ij = r[j]-r[i]
        a += m[j]/(norm(r_ij))**3*r_ij
      return a

    # total (kinetic + potential) energy
    def calc_energy(r,v):
      kin = 0.;pot = 0.
      r_ij  = np.zeros_like(r[0])
      for i in range(nbody):
        kin += 1/2*m[i]*(norm(v[i])**2)
        l = [k for k in range(nbody) if k > i]
        for j in l:
          r_ij = r[j]-r[i]
          pot  -= (m[i]*m[j])/norm(r_ij)
      return (kin+pot)/G_new

    # total angular momentum
    def calc_moment(r,v):
      L = np.zeros_like(r[0])
      for i in range(nbody):
        L += m[i]/G_new*np.cross(r[i],v[i])
      return norm(L)

    # initial total energy
    energy[0] = calc_energy(r[0],v[0])

    # initial angular momentum
    moment[0] = calc_moment(r[0],v[0])

    # first step of leapfrog integration - acceleration for initial velocity by RK 2
    for i in range(nbody):
      acc = acceleration(i,r[0]+1/2*dt*(-1/2*dt*v[0]))

      # velocity at t[n-1/2] for n=1
      v[0,i] -= 1/2*dt*acc

    # regular steps of leapfrog integration
    for n in range(steps-1):

      # velocities from n-1/2 to n+1/2
      for i in range(nbody):
        # acceleration
        acc = acceleration(i,r[n])
        v[n+1,i] = v[n,i] + dt*acc
        # position change from n to n+1/2 by velocities at n+1/2
        r[n+1,i] = r[n,i] + 1/2*dt*v[n+1,i]

      # 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])

      # finishing leapfrog integration step to the position n+1 using velocities at n+1/2
      for i in range(nbody):
        r[n+1,i] += 1/2*dt*v[n+1,i]

      # time step
      t[n+1] = t[n] + dt

    # check E and L
    mu = msun*mplanet/(msun+mplanet)             # reduced mass
    M  = (msun + mplanet)                        # total mass

    # specific orbital energy
    energy_check = -mu*G_new*M/(2*a)
    # specific angular momentum
    moment_check =  mu*np.sqrt((1-e**2)*G_new*M*a)

    # exit the simulation by closing the figure
    def handle_close(evt):
      raise SystemExit('Closed figure, exit program.')

    # simulation
    if simulation:
        fig = plt.figure(figsize=(7,5))
        fig.canvas.mpl_connect('close_event', handle_close)
        for i in range(nbody):
            x = r[:,i,0]-r[:,0,0]
            y = r[:,i,1]-r[:,0,1]
        for m in range(steps):
            plt.cla()
            plt.ylim([-1.5,1.5])
            plt.xlim([-1.2,1.75])
            plt.title(r"leapfrog method - orbit""\n"\
                      "step size = %1.2f [orbital period (2$\pi$)], " %dt +\
                      "\n""a = %1.2f [au], " %a + "e = %1.2f " %e,fontsize=11)
            plt.xlabel(r"$x~[in~au]$",fontsize=11)
            plt.ylabel(r"$y~[in~au]$",fontsize=11)
            plt.grid()
            plt.scatter(x[1:m],y[1:m],s=1,color='blue')
            plt.scatter(x[m],y[m],s=40,color='darkblue')
            plt.pause(0.00001)
    else:
        # plots
        # figure 1 - orbit
        plt.figure(figsize=(20,5))
        plt.subplots_adjust(wspace = 0.275,left=1/8, right=1-1/8, bottom=1/4.8, top=1-1/7.5)
        plt.subplot(131)
        for i in range(nbody):
            plt.scatter((r[:,i,0]-r[:,0,0]),(r[:,i,1]-r[:,0,1]),s=1,color='b')
        plt.title(r"leapfrog method - orbit""\n"\
                  "step size = %1.2f [orbital period (2$\pi$)], " %dt +\
                  "\n""a = %1.2f [au], " %a + "e = %1.2f " %e,fontsize=11)
        plt.xlabel(r"$x~[in~au]$",fontsize=11)
        plt.ylabel(r"$y~[in~au]$",fontsize=11)
        plt.grid()

        col = 'dimgrey'
        plt.annotate('Mean total energy: '+str(np.mean(energy))\
                     , xy=(0.0, -0.25), xycoords='axes fraction',color=col)
        plt.annotate('Specific orbital energy: '+str(energy_check)\
                     , xy=(0.0, -0.3), xycoords='axes fraction',color=col)
        plt.annotate('Mean total angular momentum: '+str(np.mean(moment))\
                     , xy=(1.1, -0.25), xycoords='axes fraction',color=col)
        plt.annotate('Specific angular momentum: '+str(moment_check)\
                     , xy=(1.1, -0.3), xycoords='axes fraction',color=col)

        # figure 2 - total energy
        plt.subplot(132)
        plt.semilogy(t/(2*np.pi),abs(energy-energy[0])/abs(energy[0]),'k',color='b')
        plt.title(r"leapfrog method - total energy""\n"\
                  "step size = %1.2f [orbital period (2$\pi$)], " %dt +\
                  "\n""a = %1.2f [au], " %a + "e = %1.2f " %e,fontsize=11)
        plt.xlabel(r"$time~[{\rm in~orbital~period~(2\pi)}]$",fontsize=11)
        plt.ylabel(r"$\frac{|E-E(0)|}{|E(0)|}$",fontsize=13.5)
        plt.ylim([1e-10,2])
        plt.grid()

        # figure 3 - total angular momentum
        plt.subplot(133)
        plt.semilogy(t/(2*np.pi),abs(moment-moment[0])/abs(moment[0]),'k',color='b')
        plt.title(r"leapfrog method - total angular momentum""\n"\
                  "step size = %1.2f [orbital period (2$\pi$)], " %dt +\
                  "\n""a = %1.2f [au], " %a + "e = %1.2f " %e,fontsize=11)
        plt.xlabel(r"$time~[{\rm in~orbital~period~(2\pi)}]$",fontsize=11)
        plt.ylabel(r"$\frac{|L-L(0)|}{|L(0)|}$",fontsize=13.5)
        plt.ylim([1e-10,2])
        plt.grid()
        # save figures in one pdf
        plt.savefig("leapfrog_"+str(step_val)+"_"+str(e_val)+".pdf")
        plt.show()

# run the function twobodyproblem(step_val,e_val,a_val,norbit_val)
for steps in [1/10,1/20,1/100]:
    for e_value in [0.,0.2,0.5]:
            twobodyproblem(steps,e_value,a_val=1,norbit_val=10,simulation=False)

# run simulation for e = 0.5 and s = 0.10
twobodyproblem(0.1,0.5,a_val=1,norbit_val=10,simulation=True)
```

Python code 1: Two-body problem