Great job overall. I'd like to see a bit more (not lots but a bit more) numerical detail overall. Still . . . it looks like you have the concepts down.

Score: 9.5/10+9/10+10/10+10/10+9/10+8/10

ASTRO 645
Homework 4

Numerical Computation of Orbits

Sandra Bustamante

1 Turning points in spherical systems

As we know, the apocenter r_a and pericenter r_p of an orbit in a spherical stellar system is given by the roots of

$$2(E - V(r)) - \frac{J^2}{r^2} = 0. \tag{1}$$

To derived equation 1 twice, we need to substitute r=1/u and apply the chain rule.

$$2(E - V(1/u)) - J^2u^2$$
$$\frac{d}{du}\left(2(E - V(1/u)) - J^2u^2\right)$$
$$-2\frac{dV(f(u))}{df(u)}\frac{df(u)}{du} - 2J^2u$$
$$2\frac{1}{u^2}\frac{dV(f(u))}{df(u)} - 2J^2u$$

For the second derivative we need apply the chain rule again,

$$\frac{d}{du}\left(2\frac{1}{u^2}\frac{dV(f(u))}{df(u)} - 2J^2u\right)$$
$$2\frac{d}{df(u)}\left(\frac{1}{u^2}\frac{dV(f(u))}{df(u)}\right)\frac{df(u)}{du} - 2J^2$$
$$2\frac{-1}{u^2}\frac{d}{df(u)}\left(\frac{1}{u^2}\frac{dV(f(u))}{df(u)}\right) - 2J^2$$
$$-2r^2\frac{d}{dr}\left(r^2\frac{dV(r)}{dr}\right) - 2J^2$$

where $f(u) = 1/u = r$ and $f'(u) = -1/u^2 = -r^2$. This last equation is similar to Poisson equation for spherical coordinates

$$\nabla^2 V = r^2\frac{d}{dr}\left(r^2\frac{dV}{dr}\right) = 4\pi G\rho \tag{2}$$

So by usign Poisson equation,

$$-2r^2\frac{d}{dr}\left(r^2\frac{dV(r)}{dr}\right) - 2J^2$$
$$-2(4\pi G\rho) - 2J^2$$

we can note that the second derivative is negative. This tells us that the function has a local maximum. By taking the limits of 1 when $r \to [0, \infty]$ we find that:

$$\lim_{r\to 0} 2(E - V(r)) - \frac{J^2}{r^2} = -\infty$$
$$\lim_{r\to \infty} 2(E - V(r)) - \frac{J^2}{r^2} = 2(E - V(r))$$

In the latter case, we know that $(E < 0)$ since the orbit is bound making the expression also negative.

Using both of these methods (second derivative and limits) being negative, we showed that the function is concave down from $r = 0$ to infinity. If the local maximum is above the plane, then the function will have two roots and if it is below the plane it will have zero roots.

Yes, very good, but maybe a bit more physical interpretation? Note that you can find the value of u at the maximum of f(u) by solving df(u)/du=0. This shows that u = v_c/J, where v_c = is the velocity of the circular orbit. At this point there is a single root. Well to be very mathematically precise, this single root is a double degenrate root owing to the vanishing of second derivative. For other values, there are two roots, corresponding to the turning points of the rosette. Or zero roots, if there are no bound orbits at the given value of J.

2 Symplectic integration.

A symplectic integrator is an integrator that conserve phase-space volume and Poincaré invariants [1]

Leapfrog integrator is a second order method that integrates ordinary differential equations. It updates the position and velocity at interleaved times. The algorithm is being implemented is of the sequence drift-kick-drift. A drift is when the position $\vec{q}$ changes but not the momentum $\vec{p}$ and a kick is the inverse, momentum changes but not the position. The algorithm is

$$\begin{aligned}\vec{q}_{1/2} &= \vec{q}_0 + \frac{1}{2}h\vec{p}_0 \\ \vec{p} &= \vec{p}_0 + hF(\vec{q}_{1/2}) \\ \vec{q} &= \vec{q}_0 + \frac{1}{2}h\vec{p}\end{aligned}$$

where $F(\vec{q}_{1/2})$

We can test the leapfrog algorithm using the pendulum. The equation of motion of the pendulum is given by

$$\ddot{\theta} = -\sin\theta. \tag{3}$$

We can determine 3 scenarios:

- Rotation
  It is when the total energy given by

  $$E = \frac{1}{2}\dot{\theta}^2 + (1+\cos\theta) \tag{4}$$

  is bigger than the critical energy. The critical energy is given by the energy evaluated in equilibrium. For this case, is 2. If thinking of a physical pendulum, we can make the pendulum go to rotation when we start it with an added velocity.

- Libration
  It is when the total energy is smaller than the critical energy. This is where the pendulum oscillates with small amplitudes. We can achieve this when we drop the pendulum at an angle close to equilibrium.

- Equilibrium
  It is where the pendulum stays in equilibrium. This can happen when the pendulum is straight down or straight up. When it is straight up it is called an unstable equilibrium since a very small force can take it out of that state.

	θ	$\dot{\theta}$
Rotation	-4.08	1.30
Libration	0.50	0.00
Near Uns Eq	3.13	0.00

Table 1: Initial conditions use for the integration of the pendulum for the case of rotation, libration and near the unstable equilibrium.

These 3 scenarios can be created by using the initial conditions listed in table 1.

Figure 1 shows the phase space of each scenario using leapfrog integration and 4th order Runge-Kutta. When the pendulum is in libration it forms concentric circles near the stable equilibrium while in rotation it forms curve lines above the critical energy. When the pendulum is near the unstable equilibrium it forms an oblate shape where it almost touches the unstable equilibrium point.

Figure 2 shows the errors in position and velocity of leapfrog with different time steps. The error was calculated by subtracting the final value of position at a time step minus the final value at the previous time step, where the time step at each iteration gets halved.

Note that at smaller time steps the error $\mathcal{O}(h^3)$ ie. the error is of second order represented by

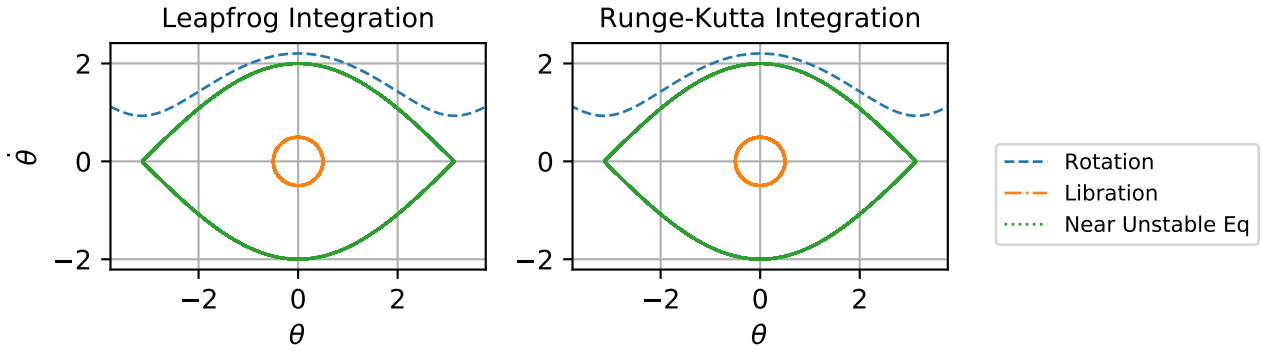

Figure 1: Plot of the phase space for each scenario using Leapfrog and 4th order Runge-Kutta (RK4).

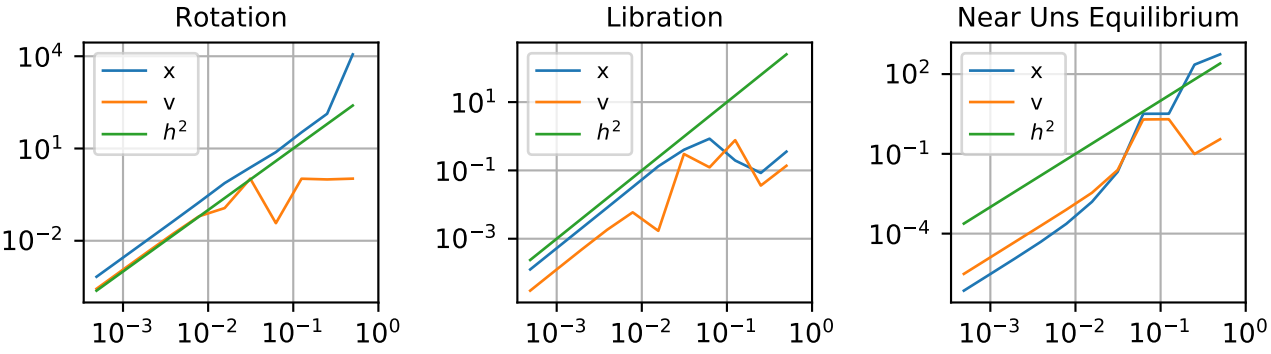

Figure 2: Plots of errors at different time steps h for each scenario: rotation, libration and near unstable equilibrium.

the line h^2. At larger time steps, equivalently the time steps is almost the same as the orbit period so it will suffer from variation equivalent to under sampling a signal in digital systems.

Figure 3 shows the total energy of the pendulum of the 3 scenarios over 10,000 orbits. It shows that when using the leapfrog integrator, the energy has small variations but remains constant in a small range through time, in other words, energy is being conserved, while the total energy using RK4 shows a drift, energy is not conserved. This is one of the advantages of leapfrog over RK4 for dynamical systems where the energy needs to be conserved.

Nice overall. In order to look at the error scaling, you either need an exact result, or a result with very small step size to compare. Otherwise, the differences themselves could depend in some way on h. In other words, looking at successive errors in phase-space coordinates has some scaling with h. That said, your scaling as h^2 looks right.

I would have expected a bit more width in the leapfrog conservation because we are only using an approximate Hamiltonian as I described in class. Not sure how you got such thin curves . . .

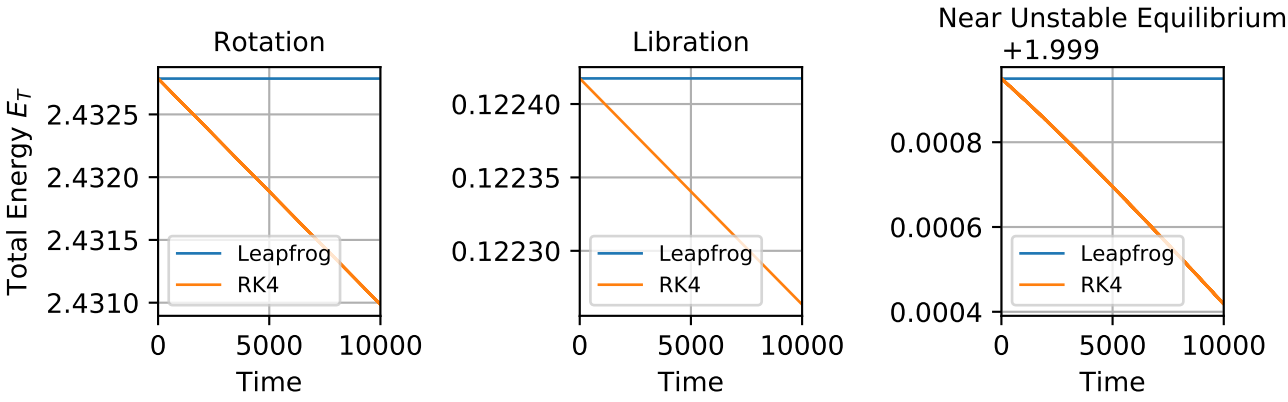

Figure 3: Plots comparing the calculated total energy through time using leapfrog integrator and RK4 integrator for each scenario of the pendulum system.

3 Orbits in an axisymmetric potential

Considering an axisymmetric potential for a simple model of a galaxian disk first proposed by Toomre (1964, ApJ):

$$U(r) = -(1 + r^2)^{-1/2}, \tag{5}$$

We can find the equation of motion by,

$$F = -\frac{dU(r)}{dr} = -\frac{r}{(1 + r^2)^{3/2}}. \tag{6}$$

This is a differential equation that can be solved using the leapfrog integrator.

Using the initial values listed on table 2, we can calculate the orbits for each case. Figure 4 shows that the orbits are not close but they precess. These types of orbits are called rosette orbits because of the shape or tube orbits because they have an inner and outer boundary.

These systems have two conserved quantities, the total energy E_T and angular momentum L. Using the initial conditions, one can calculate the total energy of the system given by,

$$E_T = K + U(r) = \frac{1}{2}v^2 + U(r) \tag{7}$$

where U is the potential energy and K is the kinetic energy. The angular momentum by

$$L = r^2\dot{\theta} \tag{8}$$

where

$$r^2 = x^2 + y^2 \tag{9}$$

$$\dot{\theta} = r \times \dot{r} \tag{10}$$

$$\dot{r}^2 = v_x^2 + v_y^2 \tag{11}$$

.

Figure 5 shows that the total energy and the angular momentum of these rosette orbits are being conserved through time.

The inner and outer radius of the orbits are found by finding the roots of

$$E_T - U(r) - \frac{L^2}{r^2} = 0 \tag{12}$$

where E and L were determined by using the initial conditions. The values for each orbit are shown in Figure 5.

The bisect method was used to find the roots within a range. The range for r was determined by first plotting the equation 12 and determining

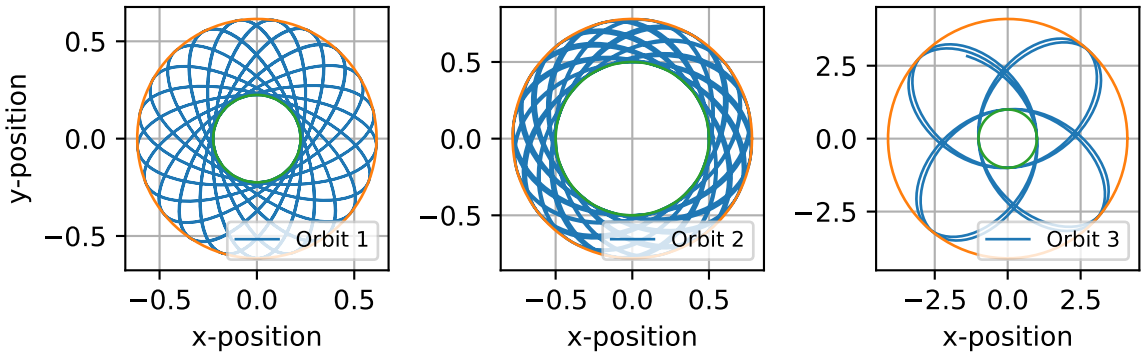

Figure 4: Rosette Orbits using initial conditions listed in Table 2.

	x	y	v_x	v_y	r_{inner}	r_{outer}	E_T	L
Orbit 1	0.30	0.00	0.30	0.40	0.22	0.61	-0.83	0.12
Orbit 2	0.00	0.50	0.60	0.00	0.50	0.78	-0.71	-0.30
Orbit 3	1.00	0.00	0.00	1.00	1.00	4.10	-0.21	1.00

Table 2: Initial conditions for the orbits in Figure 4 and the calculated value of the inner and outer radius.

visually were a sign change occurred. The value for inner and outer radius for each trial orbit is listed in table 2 and is shown in Figure 4 in orange and green. You can note that the inner and outer radius agrees with the orbit integration.

Very nice work here.

4 Non-axisymmetric orbits

A potential of a non-axisymmetric case with two fixed point masses is given by:

$$U(x,y) = -[(x-a)^2+y^2]^{-1/2} - [(x+a)^2+y^2]^{-1/2}. \tag{13}$$

By integrating the potential you can find the equations of motions. In this case, they are

$$F = -\nabla U(r) = -\left(\frac{\partial U(r)}{\partial x} + \frac{\partial U(r)}{\partial y}\right)$$

$$\ddot{x} = -\left[\frac{x-a}{(x-a)^2+y^2} + \frac{x+a}{(x+a)^2+y^2}\right]$$

$$\ddot{y} = -\left[\frac{y}{(x-a)^2+y^2} + \frac{y}{(x+a)^2+y^2}\right]$$

It can integrated using the initial conditions listed in table 3 and $a = \frac{1}{2}$.

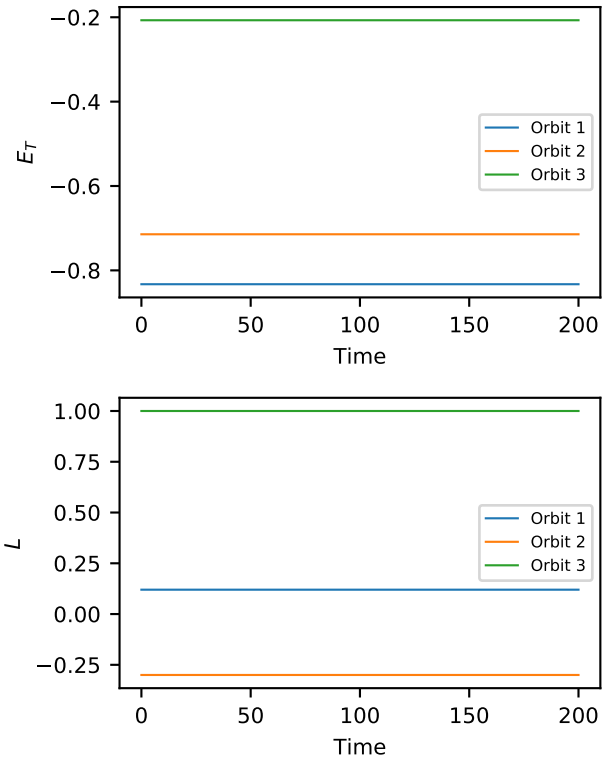

Figure 5: Left: Energy and Right: Angular Momentum of the rosette orbits in function with time.

Figure 6 shows three orbits integrated by using Leapfrog integrator and the initial conditions listed in Table 3 for 100 s. Orbit 1 and Orbit 2 are considered box orbits since they come close to center of force filling a closed area. Orbit 3 is a tube orbit.

Nice solution.

Just out of curiosity, what happens if you integrate these orbits further in time? Did you try?

	x	y	v_x	v_y	Time
Orbit 3	2.00	0.00	0.00	0.40	100.00
Orbit 4	2.00	0.00	0.00	0.50	100.00
Orbit 5	3.00	0.00	0.00	0.60	100.00

Table 3: Initial conditions used for the orbits shown in Figure 6.

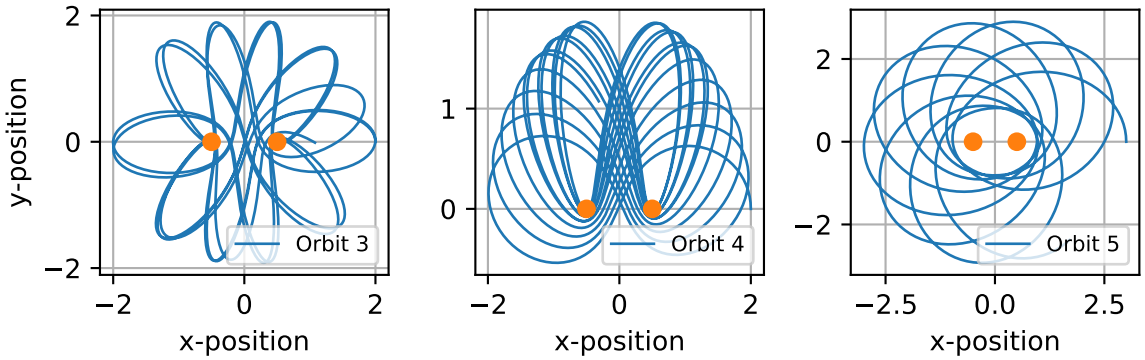

Figure 6: Orbits of a test particle under non-axisymmetric potential given by Equation 13 using initial conditions listed in Table 3.

5 Irregular orbits: Part 1

The surface of section method is a very useful technique for both determining whether an orbit is regular or irregular and classifying the type of orbit.

The logarithmic potential given by B&T equation 3-103 is

$$\Phi_L(x,y) = \frac{1}{2}v_0^2 \log\left(R_c^2 + x^2 + \frac{y^2}{q^2}\right). \quad (14)$$

The equations of motion for this system are:

$$a = -\nabla\Phi(x,y,) \quad (15)$$

$$\ddot{x} = -\frac{v_o^2 x}{R_c^2 + x^2 + \frac{y^2}{q^2}} \quad (16)$$

$$\ddot{y} = -\frac{v_o^2 x}{q^2\left(R_c^2 + x^2 + \frac{y^2}{q^2}\right)} \quad (17)$$

Using $R_c = 0.14$, $q = 0.9$, $v_0 = 1$ and the initial conditions listed in Table 4, we can integrate the orbits. Figure 7 shows a box orbit and its surface of section on the plane when $y = 0$.

Figure 8 shows a second box orbit and its surface of section on the plane when $y = 0$. Note that instead of being confined in a rectangular area it is now in a elliptical area.

	x	y	v_x	v_y	Time
Box Orbit	0.50	0.00	1.00	0.10	100.00
Tube Orbit	0.00	0.20	1.00	0.00	100.00
Box Orbit 2	0.00	0.10	1.00	0.00	100.00

Table 4: Caption

For both figures you can identify being box orbits since their surface of section go around the center. When it doesn't it will be a tube orbit as in Figure 9.

Figure 9 shows a tube orbit and its surface of section on the plane when $y = 0$.

These look nice. But I'd like to see a bit more numerical detail here. E.g. how did you determine the values at the plane y=0? How accurate is your S.O.S. plot? Does it depend on step size at all?

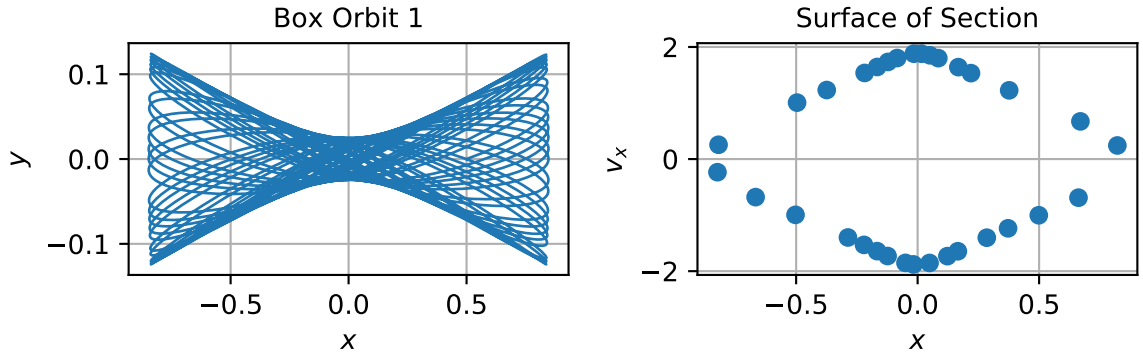

Figure 7: Example of a box orbit using initial conditions listed in Table 4 and its surface of section on the plane $y = 0$.

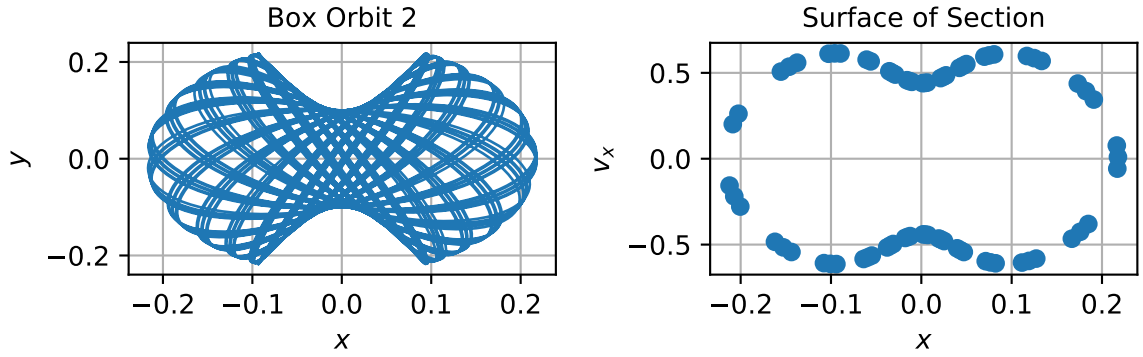

Figure 8: Second example of a box orbit using initial conditions listed in Table 4 and its surface of section on the plane $y = 0$.

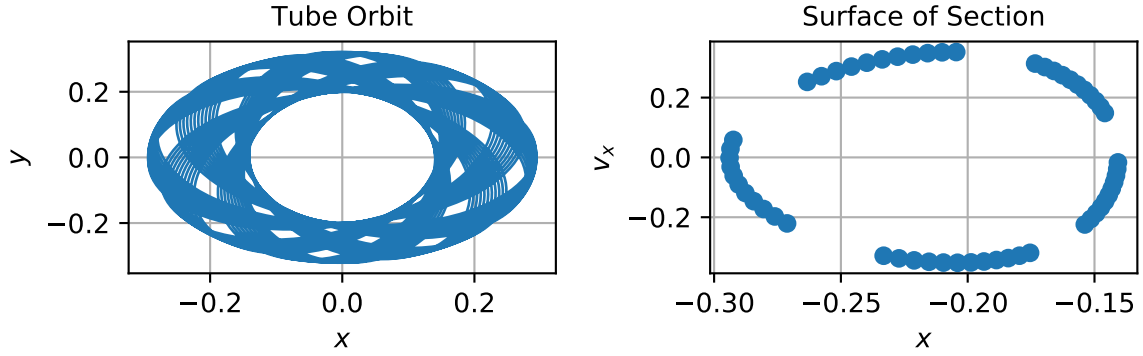

Figure 9: Example of a tube orbit using initial conditions listed in Table 4 and its surface of section on the plane $y = 0$ on a logarithmic potential.

6 The Jacobi Integral

The Jacobi integral in a rotating non axisymmetric potential is

$$E_J = \frac{1}{2}\dot{x}^2 + \Phi(\mathbf{x}) - \frac{1}{2}\left|\mathbf{\Omega}_b \times \mathbf{x}\right|^2 \qquad (18)$$

where $\mathbf{\Omega}_b$ is the constant rotation speed of the system whose direction is along the axis of rotation

It can be shown that it is a constant of motion by deriving with respect to time,

$$\begin{aligned}\frac{dE_j}{dt} &= \frac{d}{dt}\left(\frac{1}{2}\dot{x}^2 + \Phi(\mathbf{x}) - \frac{1}{2}|\mathbf{\Omega}_b \times \mathbf{x}|^2\right) \\ &= \dot{x}\ddot{x} + \dot{x}\frac{d\Phi(\mathbf{x})}{d\mathbf{x}} - \frac{1}{2}\frac{d}{dt}\left(\Omega_b^2 x^2 - (\mathbf{\Omega}_b \cdot \mathbf{x})^2\right) \\ &= \dot{x}\ddot{x} + \dot{x}\nabla\Phi(\mathbf{x}) - \dot{\mathbf{x}}\Omega_b^2\mathbf{x} + \dot{\mathbf{x}}\mathbf{\Omega}_b(\mathbf{\Omega}_b \cdot \mathbf{x})\end{aligned}$$

From Hamilton's equation we can find that

$$\ddot{x} = -\nabla\Phi(\mathbf{x}) - 2\mathbf{\Omega}_b \times \dot{\mathbf{x}} + \mathbf{\Omega}_b^2\mathbf{x} - \mathbf{\Omega}_b(\mathbf{\Omega}_b \cdot \mathbf{x}) \quad (19)$$

so substituting this into our equation and using the identity $A \cdot (B \times C) = B \cdot (C \times A)$ we get that

$$\begin{aligned}\frac{dE_J}{dt} &= 2\dot{\mathbf{x}}\left(\mathbf{\Omega}_b \times \dot{\mathbf{x}}\right) \\ \frac{dE_j}{dt} &= 2\mathbf{\Omega}_b\left(\dot{\mathbf{x}} \times \dot{\mathbf{x}}\right)\frac{dE_j}{dt} \qquad = 0\end{aligned}$$

Proving that the Jacobi integral is a constant of motion.

Additionally, knowing that the momentum in the underlying inertial frame can be described by $p = \dot{x} + \Omega_b \times x$,we can rewrite the Jacobi integral as

$$\begin{aligned}E_J &= \frac{1}{2}\left(\mathbf{p} - \mathbf{\Omega}_b \times \mathbf{x}\right)^2 + \Phi(\mathbf{x}) - \frac{1}{2}\left|\mathbf{\Omega}_b \times \mathbf{x}\right|^2 \\ &= \frac{1}{2}\left(p^2 - 2\mathbf{p} \cdot (\mathbf{\Omega}_b \times \mathbf{x}\right) + |\mathbf{\Omega}_b \times \mathbf{x}|^2) + \Phi(\mathbf{x}) \\ &\quad - \frac{1}{2}|\mathbf{\Omega}_b \times \mathbf{x}|^2 \\ &= \frac{1}{2}p^2 - \mathbf{p} \cdot (\mathbf{\Omega}_b \times \mathbf{x}) + \Phi(\mathbf{x}) \\ &= \frac{1}{2}p^2 - \mathbf{\Omega}_b \cdot (\mathbf{x} \times \mathbf{p}) + \Phi(\mathbf{x}) \\ &= E - \mathbf{\Omega}_b \cdot \mathbf{L}\end{aligned}$$

where the total energy $E = \frac{1}{2}p^2 + \Phi(\mathbf{x})$ and the angular momentum of the orbit $L = \mathbf{p} \times \mathbf{x}$.

References

[1] James Binney and Scott Tremaine.

One needs to take into account that the total time derivative operator is d/dt = d/dt_r + Omega\times

where _r means in the rotating frame. So the computations for part (a) are a bit more complicated and need to incorporate these extra terms. But the result, of course, is that Jacobi constant is conserved. The result for part (b) looks good.

A Python codes

Algorithm 1: Astro645HW04.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Oct 22 14:40:25 2019

@author: Sandra Bustamante
"""

import numpy as np
import matplotlib.pyplot as plt
import NumericIntegrations as NI
import SetupPlots as SP
import pandas as pd

#%% Definitions
def dvdt(t,x,v):
  #Pendulum equation of motion
  dvdt=-np.sin(x)
  #print(dvdt)
  return dvdt

def energy(v,x):
  #Energy of pendulum
  E=(1/2)*v**2+(1-np.cos(x))
  return E

#%% Pendulum

e=1e-2
Ecrit=2 #
#rotation,libration, near unstable equilibrium
x0=np.array([-1.3*np.pi,.5,np.pi-e])
v0=np.array([1.3,0,0])

IV=np.concatenate((x0.reshape(3,1),v0.reshape(3,1)),
                  axis=1)

a=0
b=10000
herror=1 #starting h for loop
h=.1 #for RK4
itera=12

#%%Rotation
print('Starting calculating pendulum in rotation')
print('Doing Leapfrog')

# leapfrog with different h to calculate the error
xRotA=np.zeros(itera)
vRotA=np.zeros(itera)
tRotA=np.zeros(itera)
hArray=herror/(2**np.arange(itera))
for i in np.arange(itera):
    print('iter:',i,', h:',hArray[i])
    tRot,xRot,vRot=NI.leapfrog2D(dvdt,a,b,hArray[i],
                                 IV[0],dim=1)
    tRotA[i]=tRot[-1]
    xRotA[i]=xRot[-1]
    vRotA[i]=vRot[-1]
errorxRot=np.abs(xRotA[1:]-xRotA[0:-1])
errorvRot=np.abs(vRotA[1:]-vRotA[0:-1])
errorh=hArray**2

print('Doing RK4')
tRotRK,xRotRK,vRotRK=NI.RK4(dvdt,a,b,h,IV[0],dim=1)
ERot=energy(vRot,xRot)
ERotRK=energy(vRotRK,xRotRK)

#plt.plot(xRot,vRot,'--',label='Rotation')
#print('Energy:',ERot)

#%%Libration
print('Starting calculating pendulum in libration')
print('Doing Leapfrog')

# leapfrog with different h to calculate the error
xLibA=np.zeros(itera)
vLibA=np.zeros(itera)
tLibA=np.zeros(itera)
for i in np.arange(itera):
    print('iter:',i,', h:',hArray[i])
    tLib,xLib,vLib=NI.leapfrog2D(dvdt,a,b,hArray[i],
                                 IV[1],dim=1)
    tLibA[i]=tLib[-1]
    xLibA[i]=xLib[-1]
    vLibA[i]=vLib[-1]
errorxLib=np.abs(xLibA[1:]-xLibA[0:-1])
errorvLib=np.abs(vLibA[1:]-vLibA[0:-1])

print('Doing RK4')
tLibRK,xLibRK,vLibRK=NI.RK4(dvdt,a,b,h,IV[1],dim=1)
ELib=energy(vLib, xLib)
ELibRK=energy(vLibRK, xLibRK)

#%%Near unstable position
print('Starting calculating pendulum near unstable position')
print('Doing Leapfrog')
#% leapfrog with different h to calculate the error
xUnsA=np.zeros(itera)
vUnsA=np.zeros(itera)
tUnsA=np.zeros(itera)
for i in np.arange(itera):
    print('iter:',i,', h:',hArray[i])
    tUns,xUns,vUns=NI.leapfrog2D(dvdt,a,b,hArray[i],
                                 IV[2],dim=1)
    tUnsA[i]=tUns[-1]
    xUnsA[i]=xUns[-1]
    vUnsA[i]=vUns[-1]
errorxUns=np.abs(xUnsA[1:]-xUnsA[0:-1])
errorvUns=np.abs(vUnsA[1:]-vUnsA[0:-1])

print('Doing RK4')
tUnsRK,xUnsRK,vUnsRK=NI.RK4(dvdt,a,b,h,IV[2],dim=1)
EUns=energy(vUns,xUns)
EUnsRK=energy(vUnsRK,xUnsRK)

#%% Plot Phase Space Curve
width,height=SP.setupPlot(singleColumn=True)
grid = plt.GridSpec(1,2)
fig1 = plt.figure(figsize=(width,.5*height))


#Leapfrog
ax1 = fig1.add_subplot(grid[0,0])
ax1.plot(xRot,vRot,'--',label='Rotation')
ax1.plot(xLib,vLib,'-.',label='Libration')
ax1.plot(xUns,vUns,':',label='Near Unstable Eq')
ax1.set_xlabel(r'$\theta$')
ax1.set_xlim(-1.2*np.pi,1.2*np.pi)
ax1.set_ylabel(r'$\dot{\theta}$')
ax1.set_title('Leapfrog Integration')
ax1.grid()

#RK4
ax4 = fig1.add_subplot(grid[0,1],sharey=ax1)
ax4.plot(xRotRK,vRotRK,'--',label='Rotation')
ax4.plot(xLibRK,vLibRK,'-.',label='Libration')
ax4.plot(xUnsRK,vUnsRK,':',label='Near Unstable Eq')
ax4.set_xlabel(r'$\theta$')
ax4.set_xlim(-1.2*np.pi,1.2*np.pi)
ax4.set_title('Runge-Kutta Integration')
ax4.grid()
ax4.legend(bbox_to_anchor=(1.1, 0.6))

fig1.tight_layout()
fig1.savefig('PendulumPhaseSpace.pdf')
#%%
fig1.close()

#%% Plot Energy vs time
#width,height=SP.setupPlot(singleColumn=True)
#fig2 = plt.figure(figsize=(width,height))
#grid1 = plt.GridSpec(1,1)
#
#ax2 = fig2.add_subplot(grid1[0,0])
#ax2.plot(tRot,ERot,'--',label='Rotation Leapfrog')
#ax2.plot(tRotRK,ERotRK,'--',label='RotationRK')
##ax2.plot(tLib,ELib,'-.',label='Libration')
##ax2.plot(tUns,EUns,':',label='Near Unstable Equilibrium')
##ax2.hlines(Ecrit,a,b,label='Critical Energy')
#ax2.set_xlabel('Time [s]')
#ax2.set_xlim(a,b)
#ax2.set_ylabel('Total Energy $E_T$ []')
#ax2.grid()
#ax2.legend(loc='lower right')

#%%  Phase Space Plot of Unstable equilibrium
#width,height=SP.setupPlot(singleColumn=True)
#fig3 = plt.figure(figsize=(width,height))
```

```
##grid1 = plt.GridSpec(1,1)
#
#ax3 = fig3.add_subplot(grid1[0,0])
#ax3.plot(xUns,vUns,':',label='Leapfrog')
#ax3.plot(xUnsRK,vUnsRK,':',label='RK4')
#ax3.set_title('Unstable Equilibrium')
#ax3.set_xlim(-1.1*np.pi,1.1*np.pi)
#ax3.grid()
#ax3.legend(loc='lower right')

#%% Plot comparing Leapfrog and RK4
width,height=SP.setupPlot(singleColumn=True)
fig4 = plt.figure(figsize=(width,.55*height))
gridf4 = plt.GridSpec(1,3)

#Rotation
ax6 = fig4.add_subplot(gridf4[0,0])
ax6.plot(tRot,ERot,label='Leapfrog')
ax6.plot(tRotRK,ERotRK,label='RK4')
ax6.set_xlim(a,b)
ax6.set_xlabel('Time')
ax6.set_ylabel(r'Total Energy $E_T$')
ax6.set_title('Rotation')
ax6.grid()
ax6.legend(loc='lower left')
#ax6.set_ylim(2.43275,2.4329)

#Libration
ax5 = fig4.add_subplot(gridf4[0,1])
ax5.plot(tLib,ELib,label='Leapfrog')
ax5.plot(tLibRK,ELibRK,label='RK4')
ax5.set_xlim(a,b)
ax5.set_xlabel('Time')
#ax5.set_ylabel('Total Energy $E_T$ []')
ax5.set_title('Libration')
ax5.grid()
ax5.legend(loc='lower left')
#ax5.set_ylim(0,.001)

#Near Unstable Equilbrium
ax4 = fig4.add_subplot(gridf4[0,2])
ax4.plot(tUns,EUns,label='Leapfrog')
ax4.plot(tUnsRK,EUnsRK,label='RK4')
ax4.set_xlim(a,b)
ax4.set_xlabel('Time')
#ax4.set_ylabel('Total Energy $E_T$ []')
ax4.set_title('Near Unstable Equilibrium',pad=15)
ax4.grid()
ax4.legend(loc='lower left')

fig4.tight_layout()
fig4.savefig('PendulumRK4vsLeapfrog.pdf')
#%%
plt.close()


#%% Error vs stepsize

width,height=SP.setupPlot(singleColumn=True)
fig5 = plt.figure(figsize=(width,.5*height))
gridf5 = plt.GridSpec(1,3)

ax7 = fig5.add_subplot(gridf5[0,0])
ax7.loglog(hArray[1:],errorxRot,label='x')
ax7.loglog(hArray[1:],errorvRot,label='v')
ax7.loglog(hArray[1:],1000*errorh[1:],label=r'$h^2$')
ax7.legend()
ax7.set_xticks([1,1e-1,1e-2,1e-3])
ax7.set_title('Rotation')
ax7.grid('minor')

ax8 = fig5.add_subplot(gridf5[0,1])
ax8.loglog(hArray[1:],errorxLib,label='x')
ax8.loglog(hArray[1:],errorvLib,label='v')
ax8.loglog(hArray[1:],1000*errorh[1:],label=r'$h^2$')
ax8.legend()
ax8.set_xticks([1,1e-1,1e-2,1e-3])
ax8.set_title('Libration')
ax8.grid()

ax9 = fig5.add_subplot(gridf5[0,2])
ax9.loglog(hArray[1:],errorxUns,label='x')
ax9.loglog(hArray[1:],errorvUns,label='v')
ax9.loglog(hArray[1:],1000*errorh[1:],label=r'$h^2$')
ax9.legend()
ax9.set_xticks([1,1e-1,1e-2,1e-3])
ax9.set_title('Near Uns Equilibrium')
ax9.grid()
```

```
fig5.tight_layout()
fig5.savefig('PendulumErrorvsStepsize.pdf')

#%%
plt.close()

#%% Save values to csv file
names=np.array(['$\theta$','$\dot{\theta}$'])
indexNames=['Rotation','Libration','Near Uns Eq']
row1=np.array([IV[0,0],IV[0,1]])
row2=np.array([IV[1,0],IV[1,1]])
row3=np.array([IV[2,0],IV[2,1]])

rows=[row1, row2, row3]

df = pd.DataFrame(rows,columns=names,index=indexNames)
#df.to_csv('ToomreOrbitsData.csv', float_format='%1.2f',index_label='

with open('PendulumData.tex','w') as tf:
    tf.write(df.to_latex(float_format='%2.2f',
                         index=True,
                         escape=False))

#""" continue reading for formatters and
#how to apply special names to the indexes
#https://stackoverflow.com/questions/15069814/formatting-latex-to-lat
#https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Da
#
#Code in Latex:
#\begin{table}[]
#    \centering
#    \input{CodeAndFigures/ToomreOrbitsData.tex}
#    \caption{Caption}
#    \label{tab:my_label}
#\end{table}
#
#"""
```

Algorithm 2: Astro645HW04p3.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Oct 29 16:00:37 2019

@author: Sandra Bustamante

Orbits in an axisymmetric potential

"""

import numpy as np
import matplotlib.pyplot as plt
import NumericIntegrations as NI
import SetupPlots as SP
import scipy.optimize as Opt
import pandas as pd

#%% For potential U=-(1+r**2)**(-1/2)

#Equation of motion is
def dvdt(t,x,v):
  #equation of motion for Tomre potential
  dvdt=-x*(1+(x**2).sum())**(-3/2)
  #print(dvdt)
  return dvdt

def potential(x):
    if x.ndim==1:
        U=-(1+(x**2).sum())**(-1/2)
    else:
        U=-(1+(x**2).sum(axis=1))**(-1/2)
    return U

def totalEnergy(x,v):
    if x.ndim==1:
        #assuming x=[x1,y1,z1]
        #same for v
        #U is toomre potential
        K=(1/2)*(v**2).sum()
    else:
        #assuming x=[[x1,y1,z1],[x2,y2,z2],...]
        #same for v
        K=(1/2)*(v**2).sum(axis=1)
    U=potential(x)
    E=U+K
    return E

def angularMomentum(x,v):
    if x.ndim!=1:
        r=np.linalg.norm(x,axis=1)
        thetaDot=(x[:,0]*v[:,1]-x[:,1]*v[:,0])/r**2
        L=(r**2)*thetaDot
    else:
        r=np.linalg.norm(x)
        thetaDot=(x[0]*v[1]-x[1]*v[0])/r**2
        L=(r**2)*thetaDot
    return L

def func(r,E,L):
    f=E+(1+r**2)**(-1/2)-((L/r)**2)/2
    return f

#%% Common parameters for all orbits
h=1e-3
a=0
b=200 #s
#%% First Orbit

#Setup initial values
x10=np.array([.3,0])
v10=np.array([.3,.4])
IV=np.concatenate((x10.reshape(1,2),v10.reshape(1,2)),axis=0)

#calculate conserved quantities
E1=totalEnergy(x10,v10)
L1=angularMomentum(x10,v10)

#Plot to determine range of roots
r1=np.linspace(0.001,1.5,1000)
f1=func(r1,E1,L1)
plt.plot(r1,f1)
plt.ylim(-1,1)
plt.grid()

#determine inner and outer radius
r1a=0.001
r1b=.4
r1c=1.2
r1min=Opt.bisect(func,r1a,r1b,args=(E1,L1,))
r1max=Opt.bisect(func,r1b,r1c,args=(E1,L1,))

#Integrate the orbit
t1,x1,v1=NI.leapfrog2D(dvdt,a,b,h,IV)
EL1=totalEnergy(x1,v1)
LL1=angularMomentum(x1,v1)

EL1=totalEnergy(x1,v1)
LL1=angularMomentum(x1,v1)

#%% Second Orbit

#Setup initial values
x20=np.array([0,.5])
v20=np.array([.6,0])
IV=np.concatenate((x20.reshape(1,2),v20.reshape(1,2)),axis=0)

#calculate conserved quantities
E2=totalEnergy(x20,v20)
L2=angularMomentum(x20,v20)

#Plot to determine range of roots
r2=np.linspace(0.001,1.5,1000)
f2=func(r2,E2,L2)
plt.plot(r2,f2)
plt.ylim(-1,1)
plt.grid()

#determine inner and outer radius
r2a=0.1
r2b=.6
r2c=1.2
r2min=Opt.bisect(func,r2a,r2b,args=(E2,L2,))
r2max=Opt.bisect(func,r2b,r2c,args=(E2,L2,))

#Integrate the orbit
t2,x2,v2=NI.leapfrog2D(dvdt,a,b,h,IV)

EL2=totalEnergy(x2,v2)
LL2=angularMomentum(x2,v2)


#U2,K2,E2=totalEnergy(x2,v2)
#L2=np.cross(x2,v2) #angular momentum rxv
#r2=1/np.sqrt(2*(E2-U2)/L2**2)

#%% Third Orbit

#Setup initial values
x30=np.array([1,0])
v30=np.array([0,1])
IV=np.concatenate((x30.reshape(1,2),v30.reshape(1,2)),axis=0)

#calculate conserved quantities
E3=totalEnergy(x30,v30)
L3=angularMomentum(x30,v30)

#Plot to determine range of roots
r3=np.linspace(0.1,7.5,100)
f3=func(r3,E3,L3)
plt.plot(r3,f3)
plt.ylim(-1,1)
plt.grid()

#determine inner and outer radius
r3a=0.01
r3b=2.5
r3c=7.5
r3min=Opt.bisect(func,r3a,r3b,args=(E3,L3,))
r3max=Opt.bisect(func,r3b,r3c,args=(E3,L3,))
#Integrate the orbit
t3,x3,v3=NI.leapfrog2D(dvdt,a,b,h,IV)

EL3=totalEnergy(x3,v3)
LL3=angularMomentum(x3,v3)

#U3,K3,E3=totalEnergy(x3,v3)
#L3=np.cross(x3,v3) #angular momentum rxv
#r3=1/np.sqrt(2*(E3-U3)/L3**2)

#%% Plot
width,height=SP.setupPlot(singleColumn=True)
fig1 = plt.figure(figsize=(.9*width,.5*height))
```

Numerical Computation of Orbits

Sandra Bustamante

```
grid = plt.GridSpec(1,3)

theta=np.linspace(0,2*np.pi,100)

ax1 = fig1.add_subplot(grid[0,0])
ax1.plot(x1[:,0],x1[:,1],label='Orbit 1')
ax1.plot(r1max*np.cos(theta),r1max*np.sin(theta))
ax1.plot(r1min*np.cos(theta),r1min*np.sin(theta))
ax1.legend(loc='lower right')
ax1.set_xlabel('x-position')
ax1.set_ylabel('y-position')
ax1.set_aspect('equal')
ax1.grid()

ax2 = fig1.add_subplot(grid[0,1])
ax2.plot(x2[:,0],x2[:,1],label='Orbit 2')
ax2.plot(r2max*np.cos(theta),r2max*np.sin(theta))
ax2.plot(r2min*np.cos(theta),r2min*np.sin(theta))
ax2.legend(loc='lower right')
ax2.set_xlabel('x-position')
ax2.set_aspect('equal')
ax2.grid()

ax3 = fig1.add_subplot(grid[0,2])
ax3.plot(x3[:,0],x3[:,1],label='Orbit 3')
ax3.plot(r3max*np.cos(theta),r3max*np.sin(theta))
ax3.plot(r3min*np.cos(theta),r3min*np.sin(theta))
ax3.legend(loc='lower right')
ax3.set_xlabel('x-position')
ax3.set_aspect('equal')
ax3.grid()

fig1.tight_layout()
fig1.savefig('ToomrePotentialOrbits.pdf')

#%% Plot Phase Space
#width,height=SP.setupPlot(singleColumn=True)
#fig1 = plt.figure(figsize=(width,.5*height))
#grid = plt.GridSpec(1,3)
#
#ax1 = fig1.add_subplot(grid[0,0])
#ax1.plot(np.linalg.norm(x1,axis=1),np.linalg.norm(v1,axis=1),label='Orbit 1')
#ax1.legend(loc='upper right')
#ax1.set_xlabel('r')
#ax1.set_ylabel(r'$v_r$')
#
#ax2 = fig1.add_subplot(grid[0,1])
#ax2.plot(np.linalg.norm(x2,axis=1),np.linalg.norm(v2,axis=1),label='Orbit 2')
#ax2.legend(loc='upper right')
#ax2.set_xlabel('r')
#
#ax3 = fig1.add_subplot(grid[0,2])
#ax3.plot(np.linalg.norm(x3,axis=1),np.linalg.norm(v3,axis=1),label='Orbit 3')
#ax3.legend(loc='upper right')
#ax3.set_xlabel('r')
#
#fig1.tight_layout()

#%% Plot Total Energy
width,height=SP.setupPlot(singleColumn=False)
fig2 = plt.figure(figsize=(width,2*height))
grid2 = plt.GridSpec(2,1)

ax4 = fig2.add_subplot(grid2[0,0])
ax4.plot(t1,EL1,label='Orbit 1')
ax4.plot(t2,EL2,label='Orbit 2')
ax4.plot(t3,EL3,label='Orbit 3')
#ax4.legend(loc='upper right')
ax4.legend()
ax4.set_ylabel(r'$E_T$')
ax4.set_xlabel('Time')

ax5 = fig2.add_subplot(grid2[1,0])
ax5.plot(t1,LL1,label='Orbit 1')
ax5.plot(t2,LL2,label='Orbit 2')
ax5.plot(t3,LL3,label='Orbit 3')
ax5.legend()
#ax5.legend(loc='upper right')
ax5.set_ylabel(r'$L$')
ax5.set_xlabel('Time')

fig2.tight_layout()
fig2.savefig('EnergyMomentumPlot.pdf')

#%% Save values to csv file
names=np.array(['x','y', '$v_x$','$v_y$',
                '$r_{inner}$','$r_{outer}$','$E_T$','$L$'])
indexNames=['Orbit 1','Orbit 2','Orbit 3']
```

```
row1=np.array([x10[0],x10[1],v10[0],v10[1],r1min,r1max,E1,L1])
row2=np.array([x20[0],x20[1],v20[0],v20[1],r2min,r2max,E2,L2])
row3=np.array([x30[0],x30[1],v30[0],v30[1],r3min,r3max,E3,L3])

rows=[row1, row2, row3]

df = pd.DataFrame(rows,columns=names,index=indexNames)

with open('ToomreOrbitsData.tex','w') as tf:
    tf.write(df.to_latex(float_format='%2.2f',
                         index=True,
                         escape=False))

#""" continue reading for formatters and
#how to apply special names to the indexes
#https://stackoverflow.com/questions/15069814/formatting-latex-to-lat
#https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Da
#
#Code in Latex:
#\begin{table}[]
#    \centering
#    \input{CodeAndFigures/ToomreOrbitsData.tex}
#    \caption{Caption}
#    \label{tab:my_label}
#\end{table}
#
#"""
```

Algorithm 3: Astro645HW04p4.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Nov  3 19:07:03 2019

@author: sbustamanteg

Non-axisymmetric orbits

"""

import numpy as np
import matplotlib.pyplot as plt
import NumericIntegrations as NI
import SetupPlots as SP
import pandas as pd

#%% Definitions
def potential(X):
    #assuming X=[x,y]
    a=1/2
    x=X[:,0]
    y=X[:,1]
    U=-((x-a)**2+y**2)**(-1/2)-((x+a)**2+y**2)**(-1/2)
    return U

def dvdt(t,x,v):
    #assuming X=[x,y]
    #Non-axisymmetric orbits with 2 point masses
    a=1/2
    eps=0
    N1=((x[0]-a)**2+x[1]**2+eps**2)**(-1.5)
    N2=((x[0]+a)**2+x[1]**2+eps**2)**(-1.5)
    dvxdt=-(x[0]-a)*N1 - (x[0]+a)*N2
    dvydt=-x[1]*N1 - x[1]*N2
    dvdt=np.array([dvxdt,dvydt])
    #print(x,v,dvdt)
    return dvdt

def totalEnergy(x,v):
    #assuming x=[x1,y1,z1]
    #same for v
    if x.ndim==1:
        K=(1/2)*(v**2).sum()
    else:
        K=(1/2)*(v**2).sum(axis=1)
    U=potential(x)
    E=U+K
    return U,K,E

#%% Common parameters for all orbits
h=1e-3
A=0
B=100 #s

a=1/2

#%% Orbit 1
x10=np.array([2,0])
v10=np.array([0,.401])

IV=np.concatenate((x10.reshape(1,2),v10.reshape(1,2)),axis=0)
t1,x1,v1=NI.leapfrog2D(dvdt,A,B,h,IV)
U1,K1,E1=totalEnergy(x1,v1)

print(E1)

plt.plot(x1[:,0],x1[:,1])

#%% Orbit 2
x20=np.array([2,0])
v20=np.array([0,0.5])

IV=np.concatenate((x20.reshape(1,2),v20.reshape(1,2)),axis=0)
t2,x2,v2=NI.leapfrog2D(dvdt,A,B,h,IV)
U2,K2,E2=totalEnergy(x2,v2)

print(E2)
#plt.plot(x2[:,0],x2[:,1])
#plt.ylim(-1,1)
#plt.xlim(-1,1)

#%% Orbit 3
x30=np.array([3,0])
v30=np.array([0,0.6])
#x30=np.array([0,1])
#v30=np.array([0.9,0])
#x30=np.array([0,1.1])
#v30=np.array([.01,0.301])

IV=np.concatenate((x30.reshape(1,2),
                   v30.reshape(1,2)),axis=0)
t3,x3,v3=NI.leapfrog2D(dvdt,A,B,h,IV)
U3,K3,E3=totalEnergy(x3,v3)
print(E3)

#%% plot x-y
width,height=SP.setupPlot(singleColumn=True)
fig1 = plt.figure(figsize=(.9*width,.5*height))
grid = plt.GridSpec(1,3)

ax1 = fig1.add_subplot(grid[0,0])
ax1.plot(x1[:,0],x1[:,1], label='Orbit 3')
ax1.plot([-a,a],[0,0],'o')
ax1.grid()
ax1.set_xlabel('x-position')
ax1.set_ylabel('y-position')
ax1.legend(loc='lower right')

ax2 = fig1.add_subplot(grid[0,1])
ax2.plot(x2[:,0],x2[:,1],label='Orbit 4')
ax2.plot([-a,a],[0,0],'o')
ax2.grid()
ax2.set_xlabel('x-position')
ax2.legend(loc='lower right')

ax3 = fig1.add_subplot(grid[0,2])
ax3.plot(x3[:,0],x3[:,1],label='Orbit 5')
ax3.plot([-a,a],[0,0],'o')
ax3.grid()
ax3.set_xlabel('x-position')
ax3.legend(loc='lower right')

fig1.tight_layout()
fig1.savefig('NonAxisSymetricOrbits.pdf')

#%% plot phase space
width,height=SP.setupPlot(singleColumn=True)
fig2 = plt.figure(figsize=(.9*width,1*height))
grid = plt.GridSpec(2,3)

ax4 = fig2.add_subplot(grid[0,0])
ax4.plot(x1[:,0],v1[:,0],label='Orbit 3')
ax4.grid()
ax4.legend(loc='lower right')
ax4.set_ylabel(r'$v_x$')
ax4.set_xlabel(r'$x$')

ax5 = fig2.add_subplot(grid[1,0])
ax5.plot(x1[:,1],v1[:,1],label='Orbit 3')
ax5.grid()
ax5.legend(loc='lower right')
ax5.set_ylabel(r'$v_y$')
ax5.set_xlabel(r'$y$')

ax6 = fig2.add_subplot(grid[0,1])
ax6.plot(x2[:,0],v2[:,0],label='Orbit 4')
ax6.grid()
ax6.legend(loc='lower right')
ax6.set_xlabel(r'$x$')

ax7 = fig2.add_subplot(grid[1,1])
ax7.plot(x2[:,1],v2[:,1],label='Orbit 4')
ax7.grid()
ax7.legend(loc='lower right')
ax7.set_xlabel(r'$y$')

ax8 = fig2.add_subplot(grid[0,2])
ax8.plot(x3[:,0],v3[:,0],label='Orbit 5')
ax8.grid()
ax8.legend(loc='lower right')
ax8.set_xlabel(r'$x$')

ax9 = fig2.add_subplot(grid[1,2])
ax9.plot(x3[:,1],v3[:,1],label='Orbit 5')
ax9.grid()
ax9.legend(loc='lower right')
ax9.set_xlabel(r'$y$')

fig2.tight_layout()
fig2.savefig('NonAxisSymetricPhaseSpace.pdf')

#%% Plot Total Energy
#width,height=SP.setupPlot(singleColumn=True)
```

```
#fig2 = plt.figure(figsize=(width,.5*height))
#grid2 = plt.GridSpec(1,2)
#
#ax4 = fig2.add_subplot(grid2[0,0])
#ax4.plot(t1,E1,label='Orbit 1')
#ax4.plot(t2,E2,label='Orbit 2')
#ax4.plot(t3,E3,label='Orbit 3')
#ax4.legend(loc='upper right')
#ax4.set_ylabel(r'$E_T$')
#ax4.set_xlabel('Time')
#ax4.grid()
#
##ax5 = fig2.add_subplot(grid2[0,1])
##ax5.plot(t1,L1,label='Orbit 1')
##ax5.plot(t2,L2,label='Orbit 2')
##ax5.plot(t3,L3,label='Orbit 3')
##ax5.legend(loc='upper right')
##ax5.set_ylabel(r'$L$')
##ax5.set_xlabel('Time')
#
##fig2.tight_layout()
##fig2.savefig('EnergyMomentumPlot.pdf')

#%% Save Data to csv file
names=np.array(['x'    ,'y'    , '$v_x$','$v_y$','Time'])
indexNames=['Orbit 3','Orbit 4','Orbit 5']
row1=np.array([x10[0],x10[1],v10[0],v10[1],B])
row2=np.array([x20[0],x20[1],v20[0],v20[1],B])
row3=np.array([x30[0],x30[1],v30[0],v30[1],B])

rows=[row1, row2, row3]

df = pd.DataFrame(rows,columns=names,index=indexNames)

with open('NonAxisSymetricIV.tex','w') as tf:
    tf.write(df.to_latex(float_format='%2.2f',
                         index=True,
                         escape=False))
```

Algorithm 4: Astro645HW04p5.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Nov  8 16:04:39 2019

@author: Sandra Bustamante

Irregular orbits: Part 1


"""
import numpy as np
import matplotlib.pyplot as plt
import NumericIntegrations as NI
import SetupPlots as SP
import pandas as pd
import scipy.interpolate as interpolate

#%% Definitions
def totalEnergy(x,v):
    #assuming x=[x1,y1,z1]
    #same for v
    if x.ndim==1:
        K=(1/2)*(v**2).sum()
    else:
        K=(1/2)*(v**2).sum(axis=1)
    U=potential(x)
    E=U+K
    return U,K,E

def potential(X):
    #Two-dimensional non-rotating potential
    #assuming X=[x,y]
    v0=1
    Rc=0.14
    q=0.9
    x=X[:,0]
    y=X[:,1]
    U=(1/2)*v0**2*np.log(Rc**2 + x**2 + (y/q)**2)
    return U

def dvdt(t,X,V):
    v0=1
    Rc=0.14
    q=0.9
    x=X[0]
    y=X[1]
    den=Rc**2 + x**2 + (y/q)**2
    dvxdt=((-v0**2)*x)/den
    dvydt=((-v0**2)*y)/(den*q**2)
    dvdt=np.array([dvxdt,dvydt])
    return dvdt

def doleapfrog(x0,v0,A,B,h,debug=False):
    IV=np.concatenate((x0.reshape(1,2),v0.reshape(1,2)),axis=0)
    t,x,v=NI.leapfrog2D(dvdt,A,B,h,IV)
    U,K,E=totalEnergy(x,v)
    if debug:
        print('E:',E[1],E[-1])
        print('U:',U[1],U[-1])
        print('K:',K[1],K[-1])
        plt.plot(x[:,0],x[:,1])

    return t,x,v,U,K,E

def surfaceSection(x,v,debug=False):
    m=x[1:,1]
    s=x[:-1,1]
    sgn=(m*s)<0
    index=np.array(np.where(sgn)[0])
    indexv=np.array(np.where(v[index,1]>0)[0])
    index=index[indexv]
    N=len(index)
    xinterp=np.zeros(N)
    vxinterp=np.zeros(N)
    for n in range(N):
        i=index[n]
        xSP=np.array([x[i,0],x[i+1,0]])
        vxSP=np.array([v[i,0],v[i+1,0]])
        ySP=np.array([x[i,1],x[i+1,1]])
        if debug:
            print('index:',i)
            print('ySP:',ySP)
            print('xSP:',xSP)
        funcx=interpolate.interp1d(ySP,xSP)
        funcvx=interpolate.interp1d(ySP,vxSP)
```

```
            xinterp[n]=funcx(0)
            vxinterp[n]=funcvx(0)
        if debug:
            plt.plot(xinterp,vxinterp,'o')
        return xinterp,vxinterp,index

#%% Common parameters for all orbits
h=1e-3
A=0
B=100 #s

#%% Box Orbit
x10=np.array([.5,0])
v10=np.array([1,.1])

t1,x1,v1,U1,K1,E1=doleapfrog(x10,v10,A,B,h,False)
xss1,vxss1,index1=surfaceSection(x1,v1,False)

#%% Tube Orbit
x20=np.array([0,.2])
v20=np.array([1,0])

t2,x2,v2,U2,K2,E2=doleapfrog(x20,v20,A,B,h,False)
xss2,vxss2,index2=surfaceSection(x2,v2,False)

#%% Box orbit 2
x30=np.array([0,.1])
v30=np.array([1,0])

t3,x3,v3,U3,K3,E3=doleapfrog(x30,v30,A,B,h,False)
xss3,vxss3,index3=surfaceSection(x3,v3,False)

#%% Plot Box Orbit
width,height=SP.setupPlot(singleColumn=True)
fig1 = plt.figure(figsize=(.9*width,.5*height))
grid = plt.GridSpec(1,2)

ax1 = fig1.add_subplot(grid[0,0])
ax1.plot(x1[:,0],x1[:,1])
ax1.grid()
ax1.set_ylabel(r'$y$')
ax1.set_xlabel(r'$x$')
ax1.set_title('Box Orbit 1')

ax2= fig1.add_subplot(grid[0,1])
ax2.plot(xss1,vxss1,'o')
ax2.set_xlabel(r'$x$')
ax2.set_ylabel(r'$v_x$')
ax2.grid()
ax2.set_title('Surface of Section')


fig1.tight_layout()
fig1.savefig('LogPotentialBoxOrbitPlot.pdf')

#%% Plot Box Orbit 2
width,height=SP.setupPlot(singleColumn=True)
fig3 = plt.figure(figsize=(.9*width,.5*height))
grid = plt.GridSpec(1,2)

ax5 = fig3.add_subplot(grid[0,0])
ax5.plot(x3[:,0],x3[:,1])
ax5.grid()
ax5.set_ylabel(r'$y$')
ax5.set_xlabel(r'$x$')
ax5.set_title('Box Orbit 2')

ax6= fig3.add_subplot(grid[0,1])
ax6.plot(xss3,vxss3,'o')
ax6.set_xlabel(r'$x$')
ax6.set_ylabel(r'$v_x$')
ax6.grid()
ax6.set_title('Surface of Section')

fig3.tight_layout()
fig3.savefig('LogPotentialBoxOrbit2Plot.pdf')

#%% Plot Tube Orbit
width,height=SP.setupPlot(singleColumn=True)
fig2 = plt.figure(figsize=(.9*width,.5*height))
grid = plt.GridSpec(1,2)

ax3 = fig2.add_subplot(grid[0,0])
ax3.plot(x2[:,0],x2[:,1])
ax3.grid()
ax3.set_ylabel(r'$y$')
ax3.set_xlabel(r'$x$')
ax3.set_title('Tube Orbit')

ax4= fig2.add_subplot(grid[0,1])
ax4.plot(xss2,vxss2,'o')
ax4.set_xlabel(r'$x$')
ax4.set_ylabel(r'$v_x$')
ax4.grid()
ax4.set_title('Surface of Section')

fig2.tight_layout()
fig2.savefig('LogPotentialTubeOrbitPlot.pdf')

#%% Save Data to csv file
names=np.array(['x'    ,'y'    , '$v_x$','$v_y$','Time'])
indexNames=['Box Orbit','Tube Orbit ','Box Orbit 2']
row1=np.array([x10[0],x10[1],v10[0],v10[1],B])
row2=np.array([x20[0],x20[1],v20[0],v20[1],B])
row3=np.array([x30[0],x30[1],v30[0],v30[1],B])

rows=[row1, row2,row3]

df = pd.DataFrame(rows,columns=names,index=indexNames)

with open('LogPotentialIV.tex','w') as tf:
    tf.write(df.to_latex(float_format='%2.2f',
                         index=True,
                         escape=False))
```