Two-body motion with any size masses

Let's update our solvers for full 2-body motion, and put in some checks for the conservation of energy and momentum.

We'll start with our usual list of constants and load our usual libraries:

In [1]:
```
# unit conversions
MassOfSun = 2e33 # g
MassOfJupiter = 1.898e30 # g
AUinCM = 1.496e13 # cm
kmincm = 1e5 # cm/km
G = 6.674e-8 # gravitational constant in cm^3 g^-1 s^-2
```

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Let's start with a sun and a jupiter:

In [3]:
```
# in solar masses
#M1 = 1.0
M1 = 0.0009 # jupiter is 0.09% of the mass of the sun
M2 = 1.0
```

We'll start with one particle at rest, like we had before, and build from there. We'll use our oritinal rp and vp:

In [4]:
```
rp = 1.0 # in AU
vp = 35.0 # in km/s
```

We'll convert all of our parameters:

In [5]:
```
M1 = M1*MassOfSun
M2 = M2*MassOfSun
vp = vp*kmincm
rp = rp*AUinCM
```

Let's look at our original acceleration code:

In [11]:
```python
# mStar is the mass of the central star, rStar is the *vector*
#  from planet to mass of star
def calcAcc(mStar, rStar):
    mag_r = (rStar[0]**2 + rStar[1]**2)**0.5
    mag_a = -G*mStar/mag_r**2
    # how about direction?  It's along rStar
    #  but we need to make sure this direction
    #  vector is a "hat" i.e. a unit vector
    # We want the direction only:
    unitVector = rStar/mag_r
    return mag_a*unitVector
```

We'll need to update this for 2 bodys - take in 2 radii. We'll need to solve for 2 motions - for body #1 and body #2.

We'll start with a function that calculates the mass of particle 2 on particle 1:

In [12]:
```python
# force/mass for particle m1
# m2 = mass of other particle
# r1 = 3-vector for location of particle 1
# r2 = 3-vector for location of particle 2

#def calcAcc(mj, ri, rj):
#    mag_r = np.sqrt( (ri-rj).dot(ri-rj) )
#    return -G*mj*(ri - rj)/mag_r**3.0

def calcAcc(m2, r1, r2):
    mag_r = np.sqrt( (r1[0]-r2[0])**2 \
                    +(r1[1]-r2[1])**2 )#\
                    #+(r1[2]-r2[2])**2 )
    mag_a = -G*m2/mag_r**2
    # unit vector points from particle 1 -> particle 2
    unitVector = (r1 - r2)/mag_r
    # return
    return mag_a*unitVector
```

What about the acceleration of particle 2 from the force of gravity #1? If we look at the above - it's the mirror of the acceleration of #1 because of #2 - so, let's re-write this generally:

In [13]:
```
# 2 -> j
# 1 -> i
def calcAcc(mj, ri, rj):
    mag_r = np.sqrt( (ri[0]-rj[0])**2 \
                    +(ri[1]-rj[1])**2 )#\
                    #+(ri[2]-rj[2])**2 )
    mag_a = -G*mj/mag_r**2
    # unit vector points from particle 1 -> particle 2
    unitVector = (ri - rj)/mag_r
    # return
    return mag_a*unitVector

# this is now the acceleration of particle "i" due to particle "j"
```

Exercise

Use this and the Euler's Method loop we used before to calculate updates for **both** particles.

Assume rp, vp are the distances of particle 1 and the initial radius and velocity of particle #2 are 0.

Bonus: how similar is this solution to the analytical one for a jupiter mass and a sun?

Bonus: change one mass to a solar mass, what happens now?

Bonus: what if both particles are moving? How would you impliment that?

Some starter hints:

In [13]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# unit conversions
MassOfSun = 2e33 # g
MassOfJupiter = 1.898e30 # g
AUinCM = 1.496e13 # cm
kmincm = 1e5 # cm/km
G = 6.674e-8 # gravitational constant in cm^3 g^-1 s^-2
```

In [14]:
```
# in solar masses
#M1 = 1.0
M1 = 0.0009
M2 = 1.0
```

In [15]:
```
rp = 1.0 # in AU
vp = 35.0 # in km/s
```

In [16]:
```
# our initial arrays are now 2D and in 2D!
r_0 = np.array([[rp, 0], [0, 0]])
v_0 = np.array([[0, vp], [0, 0]])
```

In [17]:
```
# let's try to estimate how many steps we might need

# we can estimate a ~ initial distance
a = np.sqrt( ((r_0[0,:]-r_0[1,:])**2).sum() )

Porb = np.sqrt( 4.0*np.pi**2.0*a**3.0/(G*(M1+M2)) )
delta_t = Porb*0.0001

n_steps = int(np.round(Porb/delta_t))*10
```

Quantifying how well we conserve things

In [19]:
```
# for 2 bodies:

# energy
# I'll write this a little fancy
def calcE(m1, m2, r1, r2, v1, v2):
    mag_r = np.sqrt( (r1-r2).dot(r1-r2) )
    return 0.5*(m1*v1.dot(v1) + m2*v2.dot(v2)) - G*m1*m2/mag_r

# angular momentum
def calcL(m1, m2, r1, r2, v1, v2):
    L = m1*np.cross(r1,v1) + m2*np.cross(r2,v2)
    #mag_L = np.sqrt( L.dot(L) )
    # for 2 dimensions
    mag_L = np.sqrt(L*L)
    return mag_L
```

Exercise:

Use the above to plot the energy and momentum as a function of time. What do you notice?

Bonus: redo for different timesteps, similar masses, etc

In [ ]:

In [ ]:

In [ ]:

In [ ]:

1. Matplotlib plots ? then exercises?

- see also https://sites.google.com/a/ucsc.edu/krumholz/teaching-and-courses/python-15/class-3 (https://sites.google.com/a/ucsc.edu/krumholz/teaching-and-courses/python-15/class-3)

2. Reading files

- see https://sites.google.com/a/ucsc.edu/krumholz/teaching-and-courses/python-15/class-4 (https://sites.google.com/a/ucsc.edu/krumholz/teaching-and-courses/python-15/class-4)