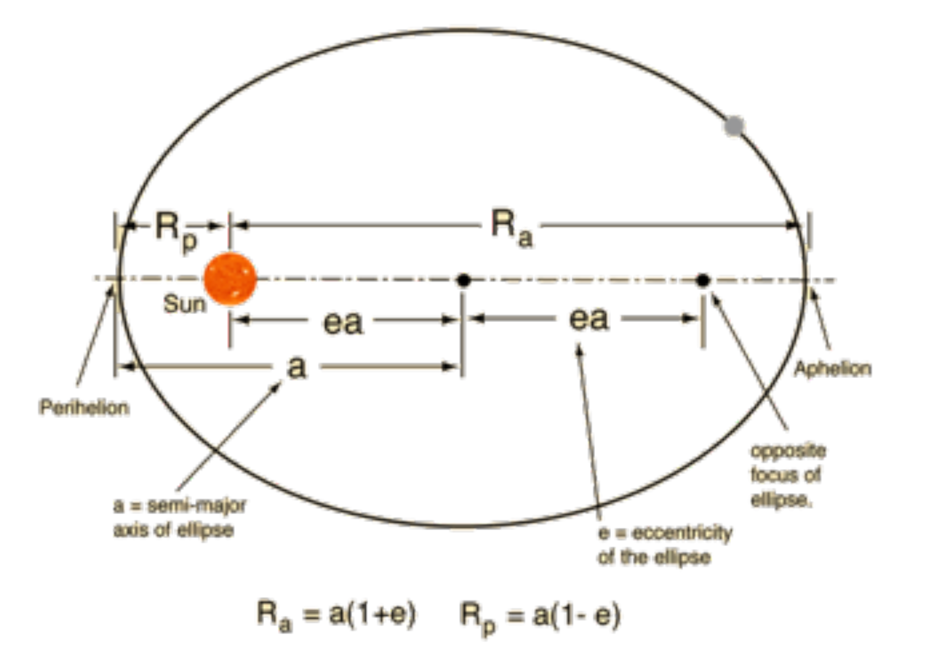
Intro - Day 3

Everything for today is posted under day 3 of:

www.astroblend.com/ba2017



* For the 2-Body problem we compared the analytical and numerical solutions
* Found that $\Delta t << P$ for accurate solutions
* How is this applicable to other simulations? (t_{dyn}, t_{cross})
* Methods to "test" the accuracy of our simulations? **(Conservation of energy and momentum)**

excerpt from:
astroblend.com/ba2017/day3.html, code section titled:
"Euler's solution with conservation of Energy and Momentum"

```
# from Aarseth's book

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

# energy
def calcE(mi, mj, ri, rj, vi, vj):
    mag_r = np.sqrt( (ri-rj).dot(ri-rj) )
    return 0.5*(mi*vi.dot(vi) + mj*vj.dot(vj)) - G*mi*mj/mag_r

# angular momentum
def calcL(mi, mj, ri, rj, vi, vj):
    L = mi*np.cross(ri,vi) + mj*np.cross(rj,vj)
    mag_L = np.sqrt( L.dot(L) )
    return mag_L


# initial x/y/z coords of masses and their velocities... there might be a nicer way of organizing this...
r_eu = [np.array([ [0., 0., 0.], [rp, 0., 0.] ])]
v_eu = [np.array([ [0., 0., 0.], [0., vp, 0.] ])]


n_eu = 10000 # number of steps over which to do our calculation
dt = 1e6 # seconds per step

# time array for Euler integration
t_eu = np.linspace(0, dt*n_eu, n_eu)

# energy array
e_eu = []
# angular momentum array
l_eu = []

# loop and calculate
for n in range(1,n_eu):
    r1 = r_eu[n-1][0,:]
    r2 = r_eu[n-1][1,:]
    v1 = v_eu[n-1][0,:]
    v2 = v_eu[n-1][1,:]
    v_eu.append( np.array([ calcAcc(m2, r1, r2)*dt + v1,
                            calcAcc(m1, r2, r1)*dt + v2 ]) )
    r_eu.append( np.array([0.5*calcAcc(m2,r1,r2)*dt**2. + v1*dt + r1,
                           0.5*calcAcc(m1,r2,r1)*dt**2. + v2*dt + r2]) )

    e_eu.append( calcE(m1, m2, r1, r2, v1, v2) )# note, this is 1 step behind
    l_eu.append( calcL(m1, m2, r1, r2, v1, v2) )# note, this is 1 step behind

# for plotting, just get x/y coords for m2, we assume
# m1 is fixed at (0,0,0)
x_eu = []
y_eu = []
for n in range(0, n_eu):
    x_eu.append( (r_eu[n][1,0] - r_eu[n][0,0])/AUinCM )
    y_eu.append( (r_eu[n][1,1] - r_eu[n][0,1])/AUinCM )
```

Your code might look a little different! Totally fine!

excerpt from:
astroblend.com/ba2017/day3.html, code section titled:
"Euler's solution with conservation of Energy and Momentum"

```
# now plot for m2

# plot x/y coords
fig, ax = plt.subplots(1, figsize = (10, 10))
fig.suptitle('Coordinates Plot')

ax.plot(x_an, y_an, linewidth=5)
ax.plot(x_eu, y_eu, linewidth=3)
ax.plot(0.0, 0.0, 'kx')
ax.set_xlabel('x in AU')
ax.set_ylabel('y in AU')
#ax.set_xlim(-3, 3)
#ax.set_ylim(-3, 3)


fig_E, ax_e = plt.subplots(1, figsize = (10, 10))
fig_E.suptitle('Energy Plot')

# E calc is 1 step behind so [1:]
ax_e.plot(t_eu[1:], np.repeat(e_an/e_eu[0], len(t_eu[1:])), linewidth=5)
ax_e.plot(t_eu[1:], e_eu/e_eu[0], linewidth=3)
ax_e.set_xlabel('Time in sec')
ax_e.set_ylabel('Energy(t)/Energy(t=0)')


fig_L, ax_l = plt.subplots(1, figsize = (10, 10))
fig_L.suptitle('Angular Momentum Plot')

# L calc is 1 step behind so [1:]
ax_l.plot(t_eu[1:], np.repeat(l_an/l_eu[0], len(t_eu[1:])), linewidth=5)
ax_l.plot(t_eu[1:], l_eu/l_eu[0], linewidth=3)
ax_l.set_xlabel('Time in sec')
ax_l.set_ylabel('Angular Momentum(t)/Angular Momentum(t=0)')


plt.show()
```