04/13/18 02:09:30 /home/jorrit/git/StarandPlanetform/Orbit_calculations/Kriekscode/Draworbit2.py

```
from __future__ import division
import Eulerorbit as k
import Leapfrogorbit as lf
import initialOrbitals as ic
import RungeKutta2 as u
import AngEnergy as ae
import AskDraw as ad
import DrawAll as da
import numpy as np
import math
import matplotlib.pyplot as plt
import time
import sys
import matplotlib.cm as cm
import os
AU = 1.5e11
Mj = 1.898e27

# ask which method to use and if we want to time
try:
    ad.asktimesteps()
except(KeyboardInterrupt):
    sys.exit(0)

def calc_resonance_orbits(P_fraction, M_J, M_e, M_s):
    r = P_fraction**(2. / 3) * ((M_s + M_e) / (M_s + M_J))**(1. / 3) * ic.a
    x = []
    y = []
    for theta in np.linspace(0, (2*np.pi), num=1000):
        x.append(r * np.cos(theta))
        y.append(r * np.sin(theta))
    return x, y

def calc_rH(M_J, M_s):
    r_H = (M_J / (3 * M_s))**(1./3)
    return r_H

def Draw(headwind_var, jup_vars):
    ic.gashead = headwind_var

    Mj = jup_vars

    # decide if save or plot figure
    save = False
    colorplot = False

    if ic.calcRK:
        method = "RK"
        name = "Runge-Kutta"

    # define the figure
    fig = plt.figure("Mass = %s M_j" % (str(Mj / 1.898e27)), figsize=(10,10))
    # ax1 = fig.add_subplot(111)
    # ax1.set_xlabel("x (AU)")
    # ax1.set_ylabel("y (AU)")
    # ax1.set_title('hw = %.1f percent , runtime = %.f yrs' % (ic.gashead * 100, ic.stepamount
/ 100. ))

    ax2 = fig.add_subplot(111)
    ax2.set_xlabel('Time (yrs)', fontsize=14)
    ax2.set_ylabel('Distance to sun (AU)', fontsize=14)
    ax2.tick_params(labelsize=12)
    ax2.set_xlim([0, ic.stepamount / 100])
    ax2.set_ylim([0, 2.5])
    # ax2.set_xscale('log')

    # ax3 = fig.add_subplot(313,sharex = ax2,sharey= ax2)
    # ax3.set_ylabel("Angularmomentum in  %")

    ic.Orbitals.instances = []
    q = ic.Ms/Mj            # Mass ratio sun / jup
    Earth = ic.Planet("Planet", "Jupiter", ic.a, ic.e, q, Mj, 'red')
    Earth2 = ic.Planet("Earth", "Earth", 0.99 * ( 2 * ic.a), ic.e, ic.q, ic.Mp, 'blue')

    print 'Working on body no. %s'%str(master_index+1)
```

```
    Sun = ic.Star("Star", "Sun", ic.Ms, 'black')

    for i in ic.Orbitals.instances:
        if 'HW' in i.name:
            continue
        else:
            i.CM()

    for i in ic.Orbitals.instances:
        i.InitialSpeed()


################
    # this calculates our energy and angularmomentum
    englist = []
    angmomlist = []
    i = 0

    ctr = 0
    stoporbitvr = False
    ilast = 0
    vrlist = []

    # Looping over the time steps
    while i < ic.stepamount:
        # determine which calculation we will perform
        u.calcRK(ic.dt)

        # determine the Vr of earth
        # if not stoporbitvr:
        #     for g in ic.Orbitals.instances:
        #         if g.name == "Earth":
        #             if g.y <= 7070000000 and g.y >=0 and g.x >=0:
        #                 if ilast +1 == i:
        #                     ilast = i
        #                     continue
        #                 ilast = i

        #                 if i == 0:
        #                     r0 = np.sqrt(g.y**2 + g.x**2)

        #                 if i != 0:

        #                     r = np.sqrt(g.y**2+g.x**2)
        #                     vrlist.append((((r0/ic.a)-(r/ic.a))/(i/100)))
        #                     ctr+=1
        #                 if ctr == 5:
        #                     stoporbitvr = True

        #             break



        # calculate energy and angular momentum
        englist.append(ae.eng())
        angmomlist.append(ae.angmom())

        # take initial energy and angular momentum
        if i == 0:
            starteng = ae.eng()
            startangmom = ae.angmom()

        i+=1

    vr_of_run = np.mean(vrlist)

# set the angular momentum and energy as a fraction of its initial value (percentages)
    absenglist = []
    absangmomlist = []
    for i in range(len(englist)):
        absenglist.append(englist[i]/starteng)
        absangmomlist.append(angmomlist[i]/startangmom)

    x10list=[]
    y10list = []
    xs10list=[]
```

```
    ys10list = []

    res_colors = ['#1b9e77','#d95f02','#7570b3','#e7298a','#66a61e','#e6ab02']

    k = 0
    testMassNum = 0
    for i in ic.Orbitals.instances:
        # if i.name == "Earth":
        #     for j in range(len(i.xlist)):
        #         if j %10 ==0:
        #             x10list.append(i.xlist[j])
        #             y10list.append(i.ylist[j])
        # if i.name == "Star":
        #     for j in range(len(i.xlist)):
        #         if j %10 ==0:
        #             xs10list.append(i.xlist[j])
        #             ys10list.append(i.ylist[j])


        # ax1.plot(np.array(i.xlist) / ic.a, np.array(i.ylist) / ic.a, label=i.expl_name, c=i.color)

        if i.expl_name == "Jupiter":
            jup_x = np.array(i.xlist)
            jup_y = np.array(i.ylist)

        if i.name == "Earth":
            dist_sun = np.sqrt(np.array(i.xlist)**2 + np.array(i.ylist)**2) / ic.a
            dist_sun2 = []

            for k in range(len(dist_sun)):
                if dist_sun[k] >= 0.1:
                    dist_sun2.append(dist_sun[k])
                else:
                    break

            dist_jup = np.sqrt(np.array(i.xlist - jup_x)**2 + np.array(i.ylist - jup_y)**2) / ic.a
            ax2.plot(np.arange(len(dist_sun2)) / 100., dist_sun2, label='HW = '+str(ic.gashead * 100)+'%'+' of v_k, Mj = '+str(Mj / 1.898e27)+'Mj', c=res_colors[master_index])
            # ax2.plot(np.arange(len(dist_jup)) / 100., dist_jup, label='HW = '+str(ic.gashead * 100)+'%'+' of v_k', c=res_colors[master_index])
            # ax2.axhline(0, c='black')
            # ax2.plot(np.arange(len(dist_sun)) / 100., dist_sun, label='HW = '+str(ic.gashead * 100)+'%'+' of v_k', c=res_colors[master_index])
            ax2.annotate("Mj = "+str(Mj / 1.898e27)+"M_jupiter", xy=(10, 0.25), ha='left', va='center', fontsize='14', fontweight='bold')
            ax2.annotate("Mj/Me = %.2g" % (Mj / ic.Mp), xy=(10, 0.15), ha='left', va='center', fontsize='14', fontweight='bold')




        k += 1


    # colors = iter(cm.rainbow(np.linspace(0, 1, len(x10list))))
    # for i in range(len(x10list)):
    #     colorz = next(colors)
    #     ax2.scatter(x10list[i],y10list[i],color=colorz,label= str(i*10) +"th step")
    #     ax2.scatter(xs10list[i],ys10list[i],color=colorz)
    # ax2.plot(range(0,ic.stepamount),absenglist, label = method)
    # ax3.plot(range(0,ic.stepamount),absangmomlist, label = method)
    # cwd = os.getcwd()
    # newdir = cwd+"/"+"dt_of_"+str(int(ic.dt))+"years_of"+str(int(ic.stepamount*ic.dt/(365.25*25*3600)))
    # try:
    #     os.mkdir(newdir)
    # except:
    #     pass

    # os.chdir(newdir)
    # ax1.legend(loc = 'upper left')
    # ax2.legend(loc = 'upper left')
    # ax3.legend(loc = 'upper left')

    # add resonance orbits
```

```python
    resonances = [1/2, 2/3, 3/4, 2, 3/2, 4/3]
    resonances_labels = ['1/2', '2/3', '3/4', '2', '3/2', '4/3']

    res_colors = ['#e41a1c','#377eb8','#4daf4a','#984ea3','#ff7f00','#ffff33']
    col = 0

    if master_index == lenHWs - 1:
        for frac in resonances:
            x, y = calc_resonance_orbits(frac, Mj, ic.Mp, ic.Ms)
            if frac < 1:
                # ax1.plot(np.array(x) / ic.a, np.array(y) / ic.a, c=res_colors[col], ls='dotted', lw=2, label='Pe/Pj = %s'%resonances_labels[col])
                ax2.axhline(np.sqrt(np.array(x[0])**2 + np.array(y[0])**2) / ic.a, c=res_colors[col], ls='dotted', label='Pe/Pj = %s'%resonances_labels[col])
            else:
                # ax1.plot(np.array(x) / ic.a, np.array(y) / ic.a, c=res_colors[col], ls='dashed', lw=2, label='Pe/Pj = %s'%resonances_labels[col])
                ax2.axhline(np.sqrt(np.array(x[0])**2 + np.array(y[0])**2) / ic.a, c=res_colors[col], ls='dashed', label='Pe/Pj = %s'%resonances_labels[col])

            col += 1


        r_hill_jup = calc_rH(Mj, ic.Ms)

        ax2.axhline(1, c='red', label="Jupiter's orbit")
        ax2.fill_between(np.arange(-100, 10000), 1 - r_hill_jup, 1 + r_hill_jup, label="Jupiter's Hillsphere", alpha=0.3, color='red')

    ax2.legend(loc='upper center', bbox_to_anchor=(0.5, 1.15), fancybox=True, shadow=True, ncol=4, fontsize=12)

    if save:
        np.save("Orbitals.npy",ic.Orbitals.instances)
        np.save("englist.npy",absenglist)
        np.save("anglist.npy",absangmomlist)
        np.save("stepamount.npy",ic.stepamount)
        print 'saving under:' + str(newdir)
        plt.savefig("plot_with_dt_"+str(ic.dt) +"and_years_"+str(ic.stepamount*ic.dt/(365.25*25*3600))+".png")

    # plt.show()
    return vr_of_run

headwinds = np.arange(3.85, 3.9, 0.01)
# headwinds = [0.01, 0.1, 1.0]
lenHWs = len(headwinds)

jup_masses = [1.898e27]
lenMasses = len(jup_masses)
print 'Number of bodies to do: ', lenHWs*lenMasses


all_vr_of_runs = np.zeros([lenMasses, lenHWs])

for j in range(len(jup_masses)):
    master_index = 0

    for i in range(len(headwinds)):
        vr_of_run = Draw(headwinds[i] / 100., jup_masses[j])
        master_index += 1
        print vr_of_run
        all_vr_of_runs[j, i] = vr_of_run

    plt.show(block=True)
    cwd = os.getcwd()
    plt.savefig(os.path.abspath(os.path.join(cwd, 'plot_diff_masses_%smj.png' % str(jup_masses[j] / 1.898e27))))
    # plt.savefig(os.path.abspath(os.path.join(cwd, '15headwinds.png')))



np.save('all_vr_of_runs.npy', all_vr_of_runs)

print all_vr_of_runs
```