Logan Pearce PTSY595B

Generalized Leapfrog Integrator for R3BP

My integrator is designed to integrate only the restricted planar three body problem for one star, one planet (PL), and one massless test particle (TP). The diagram below is a schematic of the code structure. In brief, we initialize the System class with starting position and velocity for the test particle and planet in cartesian coordinates, along with the mass of the planet (m1). By default, the star mass (m0) is defined as 1-m1, such that m0+m1 = 1. Cartesian coordinates are defined in units of AU, and masses in Msun, so that G = 1. We also supply the System class with a step size ('dt') and either a desired number of orbits or number of steps to define the scope of the run.

After initializing the System class, we call System.Run() to run the integrator. This function computes the Drift-Kick-Drift scheme outlined below for each step of size dt, until termination conditions are met. Termination conditions are that Nsteps are reached, or the TP is ejected (either TP energy > 0 or TP crosses PL orbit).

Diagram of code structure:

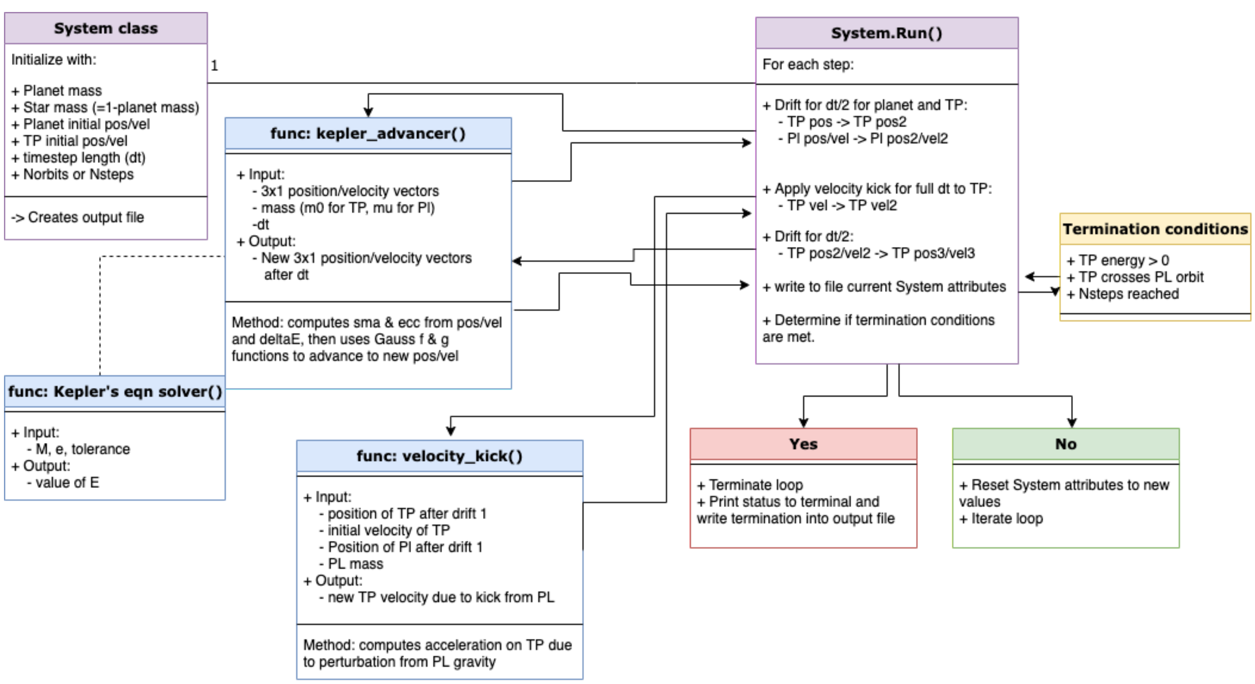
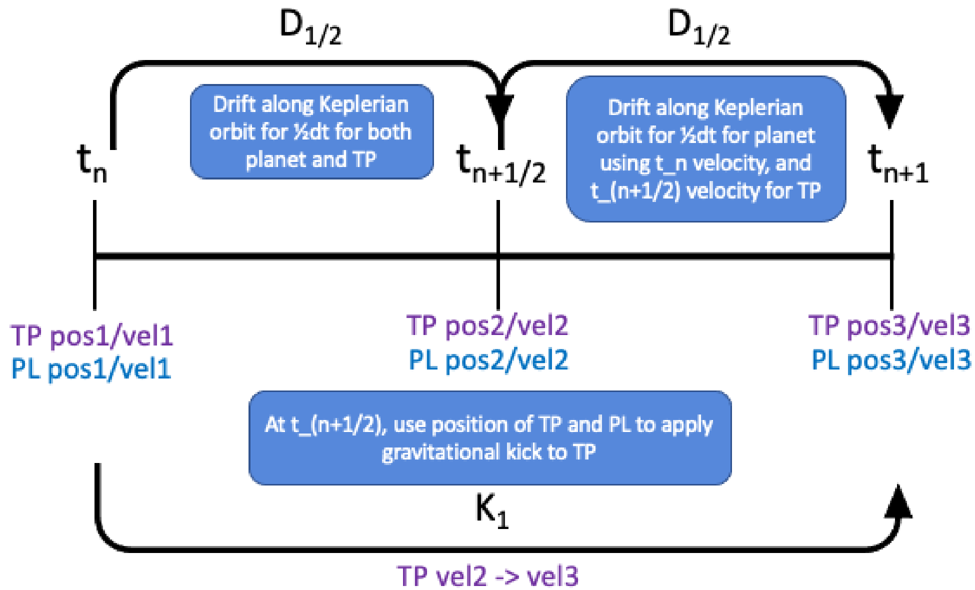


Diagram of Drift-Kick-Drift scheme:



Validation:

To test that my code was reliable and believeable, I integrated both planet and TP without graviational interaction to be sure that circular orbits were propagating correctly. I tested a Jupiter mass planet and a test particle at the distance of Ceres, to insure that remained stable since I know that is a stable orbit.

Testing scheme:

I ran a test of each mass for planets in 10^-0.5 Msun increments, initializing each test with the planet at x = 1AU and vy = 1 AU/2piyr. I initialized the test particle at decreasing x values for each test, beginning at x = 0.99, and decreasing at intervals of 0.01 AU until the algorithm had found three stable orbits in a row. The state vectors of each object were written to a csv file at each timestep. After finding the stability boundry with these "coarse" tests, I ran a "fine" test on each mass increment, testing initial separations within the interval +/- 0.01AU of the coarse stability boundary, testing increments of 0.001AU.

Results:

Load a stable and unstable orbit on either side of boundary for $m_1 = 10^{-5.0}$:

In [69]:
```
stable_orbit = pd.read_csv('coarse-runs/GLFI-2021-2-19-run-m5.0-sep0.93
unstable_orbit = pd.read_csv('coarse-runs/GLFI-2021-2-19-run-m5.0-sep0.
```

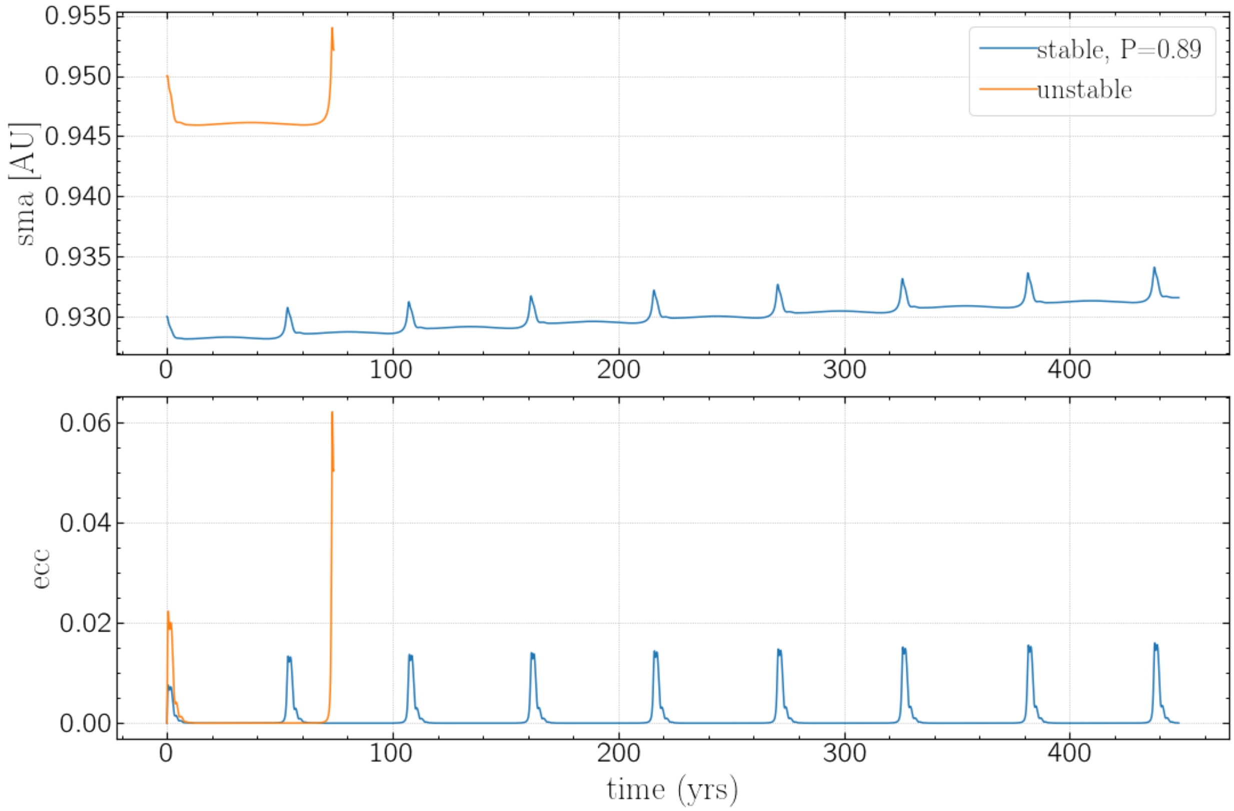
Compute a(t) and e(t) for each:

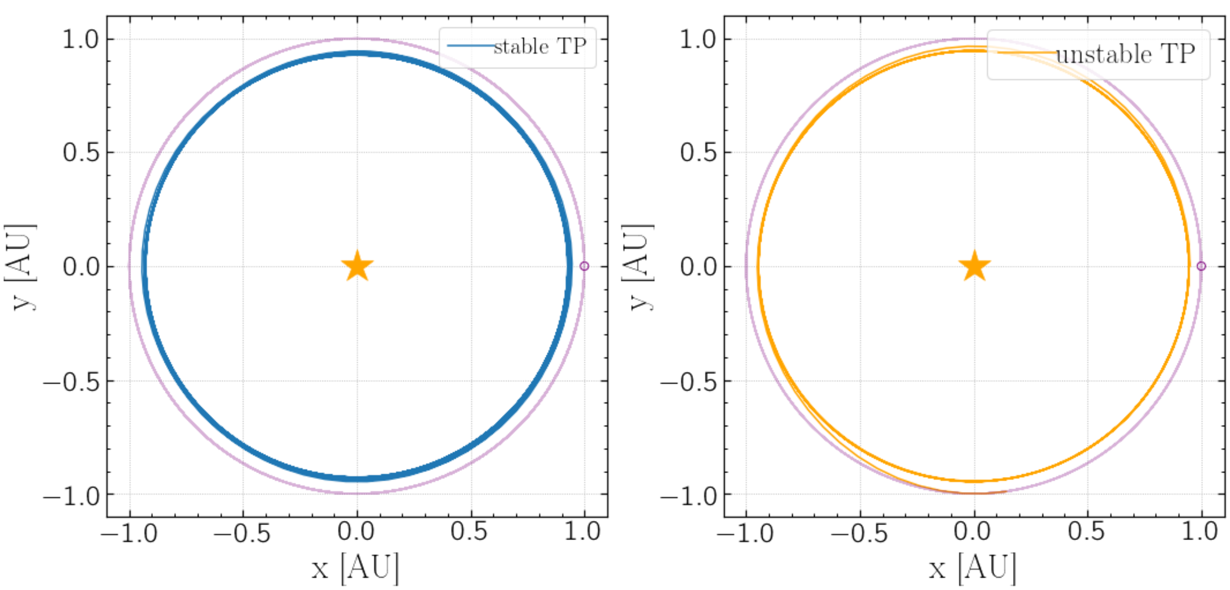
In [70]:

```
from tools import Compute_sma_ecc
kep = 1-10**(-5.0)
stable_a,stable_e = np.zeros(len(stable_orbit)),np.zeros(len(stable_orb
more_stable_a,more_stable_e = np.zeros(len(more_stable_orbit)),np.zeros
unstable_a,unstable_e = np.zeros(len(unstable_orbit)),np.zeros(len(unst
for i in range(len(stable_orbit)):
    post = np.array([stable_orbit['xt'][i],stable_orbit['yt'][i],stable
    velt = np.array([stable_orbit['vxt'][i],stable_orbit['vyt'][i],stab
    f = Compute_sma_ecc(post, velt, kep)
    stable_a[i] = f[0]
    stable_e[i] = f[1]

for i in range(len(unstable_orbit)):
    post = np.array([unstable_orbit['xt'][i],unstable_orbit['yt'][i],un
    velt = np.array([unstable_orbit['vxt'][i],unstable_orbit['vyt'][i],
    f = Compute_sma_ecc(post, velt, kep)
    unstable_a[i] = f[0]
    unstable_e[i] = f[1]
```

In [100]:

```
%matplotlib inline
import matplotlib as mpl
mpl.rcParams['axes.labelsize'] = 30
mpl.rcParams['legend.fontsize'] = 25
mpl.rcParams['xtick.labelsize'] = 25
mpl.rcParams['ytick.labelsize'] = 25

Norbits = 50
lim = Norbits*100
plt.figure(figsize=(15,10))
plt.subplot(211)
plt.plot(stable_orbit['time'][:lim],stable_a[:lim], label = 'stable, P=
#plt.plot(more_stable_orbit['time'][:lim],more_stable_a[:lim], label =
plt.plot(unstable_orbit['time'][:lim],unstable_a[:lim], label = 'unstab
plt.ylabel('sma [AU]')
plt.grid(ls=':')
plt.legend()
plt.subplot(212)
plt.plot(stable_orbit['time'][:lim],stable_e[:lim])
plt.plot(unstable_orbit['time'][:lim],unstable_e[:lim])
plt.ylabel('ecc')
plt.xlabel('time (yrs)')
plt.grid(ls=':')
#plt.savefig('test1.png')
plt.tight_layout()
plt.show()

plt.figure(figsize=(15,10))
plt.subplot(121)
orb = stable_orbit[:lim]
plt.plot(orb['xt'],orb['yt'], label='stable TP')
plt.scatter(1,0,marker='o',color='purple',facecolor='None',s=50)
plt.plot(orb['x'],orb['y'],color='purple',alpha=0.3)
plt.plot(0,0,marker='*',color='orange',markersize=30)
plt.gca().set_aspect('equal')
plt.grid(ls=':')
plt.xlabel('x [AU]')
plt.ylabel('y [AU]')
plt.legend(fontsize=20,loc="upper right")
plt.subplot(122)
orb = unstable_orbit[:lim]
plt.plot(orb['xt'],orb['yt'], label='unstable TP',color='orange')
plt.scatter(1,0,marker='o',color='purple',facecolor='None',s=50)
plt.plot(orb['x'],orb['y'],color='purple',alpha=0.3)
plt.plot(0,0,marker='*',color='orange',markersize=30)
plt.gca().set_aspect('equal')
plt.grid(ls=':')
plt.xlabel('x [AU]')
plt.ylabel('y [AU]')
plt.legend(loc="upper right")
#plt.savefig('test2.png')
plt.tight_layout()
plt.show()



```


stable, P=0.89
unstable
sma [AU]
0.955
0.950
0.945
0.940
0.935
0.930
ecc
0.06
0.04
0.02
0.00
0
100
200
300
400
time (yrs)

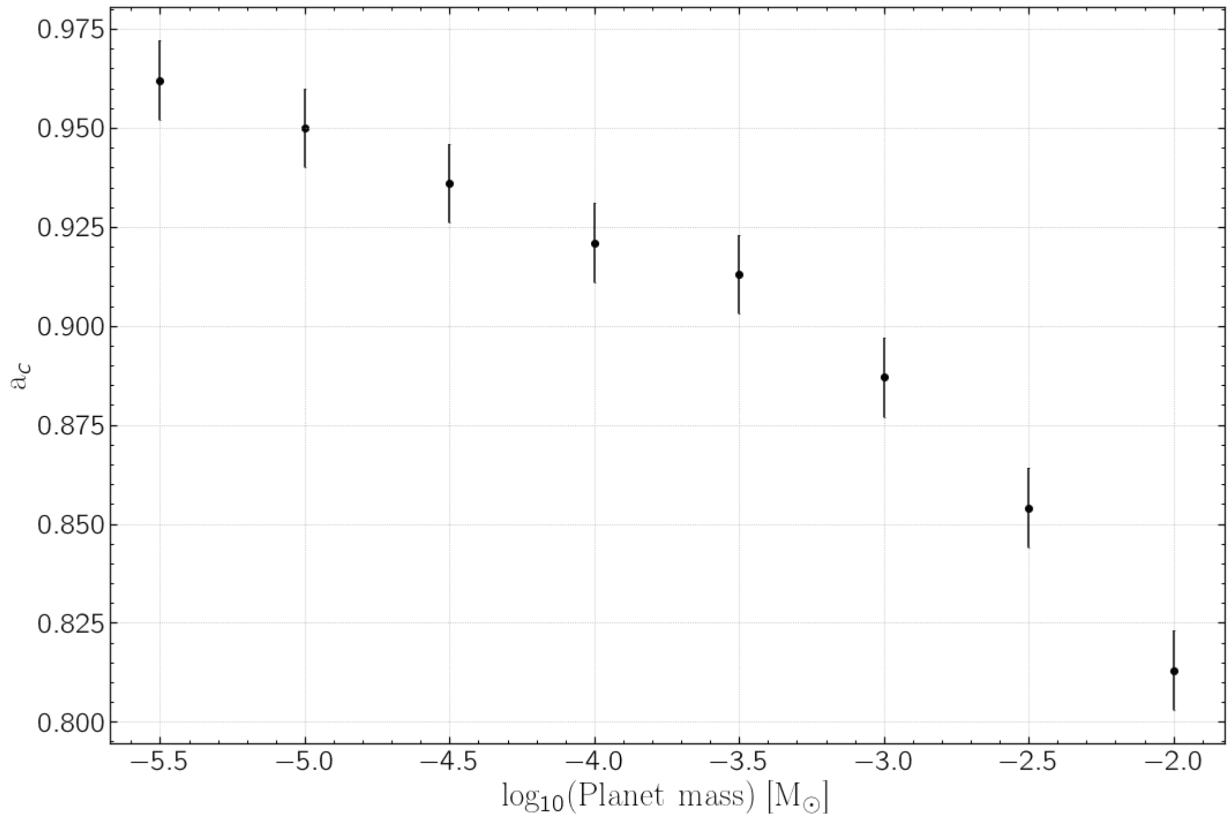

stable TP
unstable TP
y [AU]
x [AU]
1.0
0.5
0.0
−0.5
−1.0

Results of stability boundary as a function of planet mass:

Log(m1)	Stability Boundary [AU]
-5.5	0.962
-5.0	0.950
-4.5	0.936
-4.0	0.921
-3.5	0.913
-3.0	0.887
-2.5	0.854
-2.0	0.813

In [101]:

```
ac = np.array([0.961,0.949,0.935,0.920,0.912,0.886,0.853,0.812]) + 0.00
err = 0.01
delta_ac = 1 - ac

logm1 = np.linspace(-5.5,-2,8)

%matplotlib inline
plt.figure(figsize=(15,10))
plt.scatter(logm1,ac, color='black')
plt.errorbar(logm1,ac,yerr=err, ls='None', capsize=1, ecolor='black')
plt.xlabel(r'log$_{10}$(Planet mass) [M$_{\odot}$]')
plt.ylabel(r'a$_{c}$')
plt.grid(ls=':')
#plt.legend()
plt.tight_layout()
plt.show()
```



Fitting power law to relation:

Fitting a line to the function:

$$\Delta a_c = cm^{\beta}$$

Where $\Delta a_c = 1 - a_c$

Because I am working with log(mass), the fitting function becomes a straight line where:

$$\Delta a_c = \beta \log_{\mathrm{m}} + \mathrm{c}$$

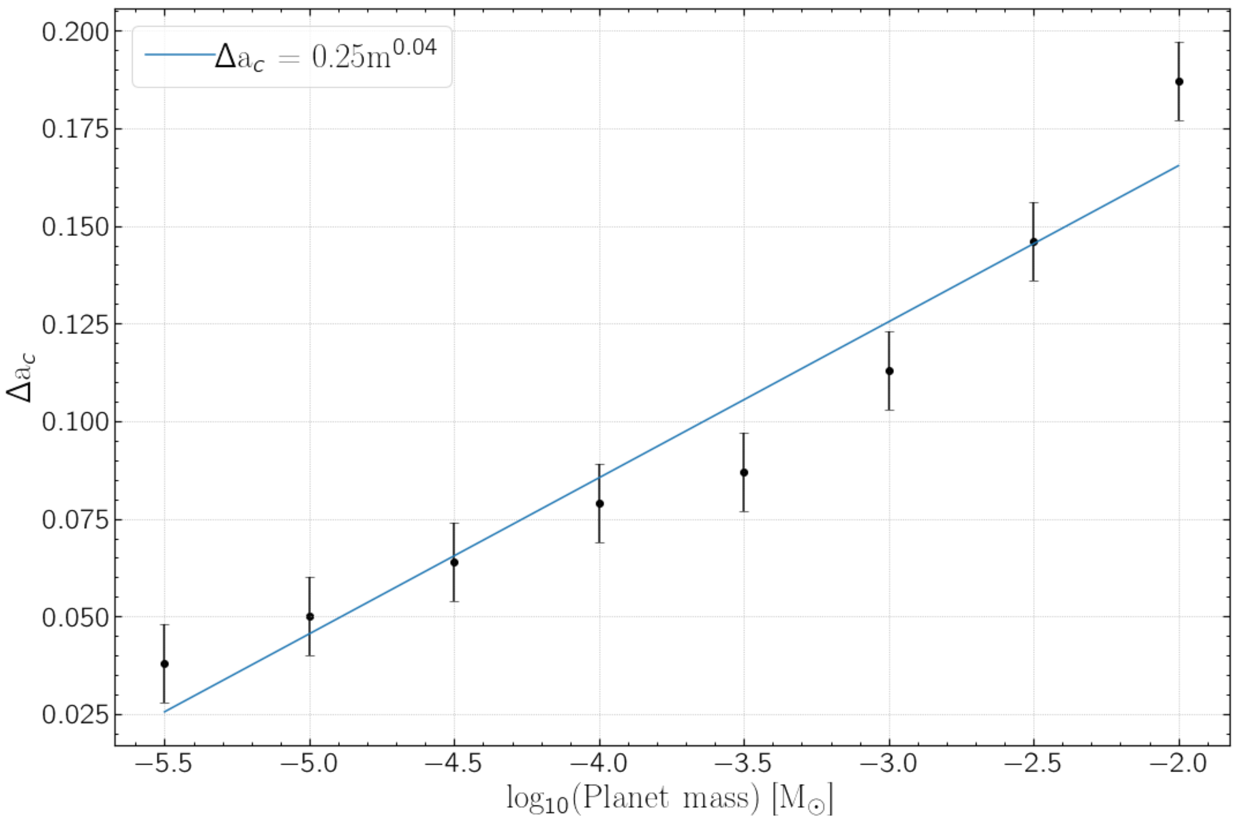
In [68]:

```python
# Define the fitting function:
def line(x,beta,c):
    return beta*x + c

from scipy.optimize import curve_fit
popt, pcov = curve_fit(line, logm1, delta_ac)
print('Beta = {:.2f}, c = {:.2f}'.format(popt[0],popt[1]))
```

```
Beta = 0.04, c = 0.25
```

In [111]:

```python
plt.figure(figsize=(15,10))
plt.scatter(logm1,delta_ac, color='black')
plt.errorbar(logm1,delta_ac,yerr=err, ls='None', capsize=4, ecolor='bla
plt.plot(logm1,line(logm1,*popt), label='$\Delta$a$_c$ = {:.2f}m$^{{0.0
plt.xlabel(r'log$_{10}$(Planet mass) [M$_{\odot}$]')
plt.ylabel(r'$\Delta$a$_{c}$')
plt.grid(ls=':')
plt.legend(fontsize = 30)
plt.tight_layout()
plt.show()
```



Stability criterion an explanation for the inner or outer edge of asteriod belt?

In [118]:

```
# Log Jupiter mass:
logJupiter = -3
# Use line to get location of a_c:
delta_ac_Jupiter = line(logJupiter,*popt)
# multiply by Jupiter's distance to get heliocentric distance:
print("Jupiter's stablity boundary using my function is at: {:.2f} AU".
```

```
Jupiter's stablity boundary using my function is at: 4.55 AU
```

So this does not correspond to either the inner or outer edge of the asteroid belt, so this is not a plausible explanation for either edge.

Neptune?

In [120]:

```
logNeptune = -4.3
delta_ac_Neptune = line(logNeptune,*popt)
print("Neptune's stablity boundary using my function is at: {:.2f} AU".
```

```
Neptune's stablity boundary using my function is at: 32.21 AU
```

This is somewhat close actually to the edge of the Kuiper belt at 33 AU. So this is plausible.

In [ ]:

```

```