Reading and writing data

Let's first do a bit of book keeping - figuring out how to read and write data. The easiest way to do this is with some built in functions in numpy.

We'll start by importing our usual things:

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Let's start by making a fake time and a fake angle array:

In [2]:
```
t = np.array([5, 6, 7, 8])
angle = np.array([36.0, 42.5, 44, 85])
```

And now we can simply save it with `np.savetxt` :

In [3]:
```
np.savetxt('mytxtfile.txt', [t,angle], delimiter=',')
```

We can also take a gander at this file! We can do this with another "bash" command.

In windows the equivalent is:

```
gc mytxtfile.txt | select -first 10 # head
```

or if `gc` won't work:

```
Get-Content mytxtfile.txt | select -first 10 # head
```

In [4]:
```
!head mytxtfile.txt
```

```
5.000000000000000000e+00,6.000000000000000000e+00,7.000000000000000000e
+00,8.000000000000000000e+00
3.600000000000000000e+01,4.250000000000000000e+01,4.400000000000000000e
+01,8.500000000000000000e+01
```

So we can see from the above that what we have is a bunch of `float` types where the first line is the time and the 2nd line is the angle. We can then re-read these in:

In [5]:
```
data = np.genfromtxt('mytxtfile.txt', delimiter=',')
data
```

Out[5]:
```
array([[ 5. ,  6. ,  7. ,  8. ],
       [36. , 42.5, 44. , 85. ]])
```

So now we have our data back in a data array. There are a few ways to format this, we can do:

In [6]:
```
t_in = data[0,:]; angle_in = data[1,:]

t_in, angle_in
```

Out[6]:
```
(array([5., 6., 7., 8.]), array([36. , 42.5, 44. , 85. ]))
```

We can also do this on read-in:

In [7]:
```
t_in = []; angle_in = []
t_in, angle_in = np.genfromtxt('mytxtfile.txt', delimiter=',')

t_in, angle_in
```

Out[7]:
```
(array([5., 6., 7., 8.]), array([36. , 42.5, 44. , 85. ]))
```

Or we can actually name our array like so:

In [8]:
```
names = ('Time', 'Angle')
formats = ('f4', 'f4')

named_data = np.genfromtxt('mytxtfile.txt',
                           delimiter=',',
                           dtype={'names':names,
                                  'formats':formats})

named_data
```

Out[8]:
```
array([( 5.,  6. ), (36., 42.5)],
      dtype=[('Time', '<f4'), ('Angle', '<f4')])
```

We can then access our data like so:

In [9]:
```
named_data['Time']
```

Out[9]:
```
array([ 5., 36.], dtype=float32)
```

However, note that our data is not properly formatted - this is because when we specify names and formats, it expects columns. To use this feature, we should save like:

In [10]:
```
np.savetxt('mytxtfile.txt', np.array([t,angle]).T, delimiter=',')

!head mytxtfile.txt
```

```
5.0000000000000000e+00,3.6000000000000000e+01
6.0000000000000000e+00,4.2500000000000000e+01
7.0000000000000000e+00,4.4000000000000000e+01
8.0000000000000000e+00,8.5000000000000000e+01
```

What we did above is transform our list of arrays to an array of arrays, and took the transpose with `.T` - then things are output as columns. So when we read in:

In [11]:
```
names = ('Time', 'Angle')
formats = ('f4', 'f4')

named_data = np.genfromtxt('mytxtfile.txt',
                           delimiter=',',
                           dtype={'names':names,
                                  'formats':formats})

named_data
```

Out[11]:
```
array([(5., 36. ), (6., 42.5), (7., 44. ), (8., 85. )],
      dtype=[('Time', '<f4'), ('Angle', '<f4')])
```

In [12]:
```
named_data['Time'], named_data['Angle']
```

Out[12]:
```
(array([5., 6., 7., 8.], dtype=float32),
 array([36. , 42.5, 44. , 85. ], dtype=float32))
```

So this now works just fine!

Reading in Kepler Data

Let's use what we've learned to read in some kepler datasets. There are a bunch of parameters that we won't go through in detail, but there are more details in the header files:

In [13]:
```
!head -n 40 downloads/kepler101data.txt
```

```
# columns are:
# (1) Row ID from table
# (2) System Name
# (3) planet letter
# (4) Number of total planets in system
# (5) Orbital period in days
# (6) Upper error in Orbital Period (P_orb +/- (upper error)/(lower err
or)) in days
# (7) Lower error in orbital period in days
# (8) Semi-major axis  in AU
# (9) Error in semi major axis (a +/- ea)
# (10) Eccentricity
# (11) Upper error in eccentricity
# (12) Lower error in eccentricity
# (13) Inclination in degrees
# (14) Upper error in inclination
# (15) Lower error in inclination
# (16) Planet's mass in Jupiter mass (or M*sin(i) if given)
# (15) Upper error in planet's mass in jupiter masses
# (16) Lower error in planet's mass in jupieter masses
# (17) Mass type ('Mass' or 'Msini' - tells you what is the "mass" you
actually measure)
# (18) Star's mass in solar masses
# (19) Error in star's mass
# (20) Star's radius in solar radii
# (21) Upper error in star's radius
# (22) Lower error in star's radius
# (23) Transit time in Julian days since a given time: from: Bonomo et
al 2014 - http://arxiv.org/abs/1409.4592
# (24) Upper error in transit time
# (25) Lower error in transit time
#
# Further notes:
# the "-100.0" in the last row denote upper limits
# Data from Kepler confirmed planet website: http://exoplanetarchive.ip
ac.caltech.edu/cgi-bin/TblView/nph-tblView?app=ExoTbls&config=planets
#
779,Kepler-101 ,b,2,3.4876812, 0.0000070, -0.0000070, 0.0474, 0.009,
0.086, 0.080, -0.059, 85.82, 1.73, -1.53, 0.16,  0.02, -0.01, Mass, 1.1
7, 0.06,  1.56, 0.20, -0.20, 288.77995, 0.00041, -0.00041
780,Kepler-101 ,c,2,6.029760,  0.000075,  -0.000075,  0.0684, 0.0014,
0,     -100., -100.,  84.6,  3.4,  -3.1,  0.01, -100., -100., Mass, 1.1
7, 0.06,  1.56, 0.20, -0.20, 65.4860,   0.0088,  -0.0088
```

Okay dokey, read in the file! Because there are so many parameters, let's use another library to do this for us.

Make sure you've downloaded the kepler 10 and 101 data sets!

In [14]:
```
from convert_kepler_data import read_kepler_data
kepler_data = read_kepler_data('/Users/jillnaiman1/Downloads/kepler101data.txt')
kepler_data
```

Out[14]:
```
array([(779., b'Kepler-101 ', b'b', 2., 3.4876812, 7.0e-06, -7.0e-06,
0.0474, 0.009 , 0.086,  8.e-02, -5.9e-02, 85.82, 1.73, -1.53, 0.16,  2.
e-02, -1.e-02, b' Mass', 1.17, 0.06, 1.56, 0.2, -0.2, 288.77994, 0.0004
1, -0.00041),
       (780., b'Kepler-101 ', b'c', 2., 6.02976  , 7.5e-05, -7.5e-05,
0.0684, 0.0014, 0.   , -1.e+02, -1.0e+02, 84.6 , 3.4 , -3.1 , 0.01, -1.
e+02, -1.e+02, b' Mass', 1.17, 0.06, 1.56, 0.2, -0.2,  65.486  , 0.0088
, -0.0088 )],
      dtype=[('RowID', '<f4'), ('SysName', 'S12'), ('planetLetter', 'S
2'), ('NumberOfPlanets', '<f4'), ('Porb', '<f4'), ('ePorbU', '<f4'),
('ePorbL', '<f4'), ('a', '<f4'), ('ea', '<f4'), ('ecc', '<f4'), ('eecc
U', '<f4'), ('eeccL', '<f4'), ('Incl', '<f4'), ('eInclU', '<f4'), ('eIn
clL', '<f4'), ('pMass', '<f4'), ('epMassU', '<f4'), ('epMassL', '<f4'),
('pMassType', 'S8'), ('sMass', '<f4'), ('esMass', '<f4'), ('sRadius',
'<f4'), ('esRadiusU', '<f4'), ('esRadiusL', '<f4'), ('tTime', '<f4'),
('etTimeU', '<f4'), ('etTimeL', '<f4')])
```

The above are all the different parameters for each planet, as listed by the "head" of the text file.

We can also look at individual parameters that might be interesting:

In [15]:
```
kepler_data['ecc']
```

Out[15]:
```
array([0.086, 0.   ], dtype=float32)
```

So, the first planet has a small eccentricity and the second has zero, so a circular orbit.

Let's use a library to convert this to units that the Hermite solver can use. Note: this uses a few assumptions to convert the Kepler data into initial conditions for our simulations.

Please check out the library if you want to see these assumptions if you are curious!

In [16]:
```
from convert_kepler_data import convert_kepler_data
```

In [17]:
```
star_mass, \
planet_masses, \
planet_initial_position, \
planet_initial_velocity, ecc = convert_kepler_data(kepler_data)
```

Let's see how these things look:

In [18]:
```
star_mass # Msun
```

Out[18]:
```
1.17
```

In [19]:
```
planet_masses # in Jupiter masses
```

Out[19]:
```
array([0.16, 0.01])
```

In [20]:
```
planet_initial_position # in AU
```

Out[20]:
```
array([[-0.0433236 ,  0.        ,  0.        ],
       [ 0.06702392,  0.01365115,  0.        ]])
```

In [21]:
```
planet_initial_velocity # in km/s
```

Out[21]:
```
array([[   0.        , -141.87933522,    0.        ],
       [ -24.65587511,  121.05448789,    0.        ]])
```

Exercise

Use the Hermite solver to model this system & plot the orbits of the planets. How large/small does your timestep have to be to be to get a stable orbit?

Bonus: what about for the 5 planet system?

Once you have something you like, make a note of the time parameters you used. We'll eventually save your favorite model to a file.

Possible ans

Now import the hermite stuffs to solve this:

In [22]:
```
from hermite_library import do_hermite
```

In [23]:
```
# h is for hermite!
r_h, v_h, t_h, e_h = do_hermite(star_mass,
                                planet_masses,
                                planet_initial_position,
                                planet_initial_velocity,
                               tfinal=1e7, Nsteps=8800)
```

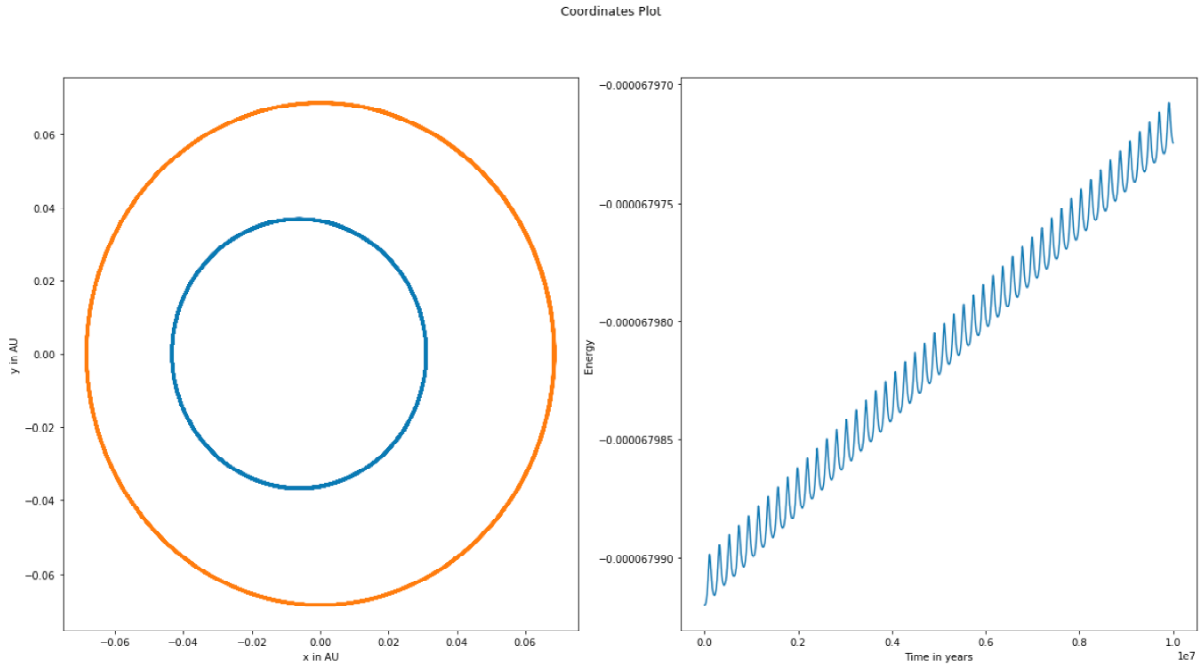
In [24]:
```
# let's plot!
fig, ax = plt.subplots(1, 2, figsize = (10*2, 10))
fig.suptitle('Coordinates Plot')

ax[0].set_xlabel('x in AU')
ax[0].set_ylabel('y in AU')

for i in range(len(planet_masses)):
    ax[0].plot(r_h[i,0,:], r_h[i,1,:], lw=3)

ax[1].set_xlabel('Time in years')
ax[1].set_ylabel('Energy')
# re-norm energy
ax[1].plot(t_h, e_h)

plt.show()
```



Saving data to a file, and reading it back in!

We'll also want to save our models and then be able to read them back in so you don't have to keep running models once you find one you like.

Per usual, we'll use a function from a library!

In [25]:
```
from hermite_library import save_hermite_solution_to_file
```

Now we'll simply call this and save our data!

In [26]:
```
# you can save it to your local directory, or another place
save_hermite_solution_to_file('myPlanetSystem.txt',
                              t_h, e_h, r_h, v_h)
```

Let's quickly take a look at this:

In [27]:
```
!head myPlanetSystem.txt
```

0.0000000000000000e+00,-6.799198885028552943e-05,-4.33185230724481215
6e-02,-1.107107656726798300e-07,0.0000000000000000e+00,6.702900216666
182742e-02,1.365104365687684444e-02,0.0000000000000000e+00,5.07810204
9476259360e-06,-1.107107656726798300e-07,0.0000000000000000e+00,1.201
416740318806473e-06,-8.523511935193556210e-01,0.0000000000000000e+00,
-1.481390926396770891e-01,7.274383576207541502e-01,0.00000000000000000
e+00,1.201416740318806473e-06,1.047159119364062479e-04,0.00000000000000
0000e+00
1.136492783270826521e+03,-6.799198824681156647e-05,-4.33024702206598055
6e-02,-1.077698343988626843e-03,0.0000000000000000e+00,6.683539105535
182823e-02,1.456950090025424180e-02,0.0000000000000000e+00,5.07758914
7273007199e-06,2.168645587296400320e-08,0.0000000000000000e+00,2.5390
02113577943157e-02,-8.520353387632382258e-01,0.0000000000000000e+00,-
1.581064578856623548e-01,7.253369660846603617e-01,0.00000000000000000e
+00,-2.012641502274654666e-06,1.046919656279508951e-04,0.00000000000000
0000e+00
2.272985566541653043e+03,-6.799198730213744509e-05,-4.32543207197962006
9e-02,-2.154487204734567476e-03,0.0000000000000000e+00,6.662919585537
262213e-02,1.548521477143941297e-02,0.0000000000000000e+00,5.07301288
4161858793e-06,1.540022662970947849e-07,0.0000000000000000e+00,5.0769
32151093775658e-02,-8.510876612812490194e-01,0.0000000000000000e+00,-
1.680440512674290232e-01,7.230990120347342565e-01,0.00000000000000000e
+00,-5.225705828457786758e-06,1.045871304427467259e-04,0.00000000000000
0000e+00
3.409478349812478882e+03,-6.799198601501236193e-05,-4.31740926349023293
5e-02,-3.229678234591605839e-03,0.0000000000000000e+00,6.641045539385
177776e-02,1.639801286492561030e-02,0.0000000000000000e+00,5.06437528
9653097760e-06,2.861343639734585766e-07,0.0000000000000000e+00,7.6129
55893831589715e-02,-8.495078221334477764e-01,0.0000000000000000e+00,-
1.779500014707275646e-01,7.207249170436130781e-01,0.00000000000000000e
+00,-8.436552868944805495e-06,1.044013589744983257e-04,0.00000000000000
0000e+00
4.545971133083306086e+03,-6.799198438368523559e-05,-4.30618161254401055
1e-02,-4.302471804248133788e-03,0.0000000000000000e+00,6.617921086026
590238e-02,1.730772332446248174e-02,0.0000000000000000e+00,5.05167994
3682296091e-06,4.179803687306237265e-07,0.0000000000000000e+00,1.0146
114138906336695e-01,-8.472952574689229799e-01,0.0000000000000000e+00,-
1.878224431498358993e-01,7.182151283282356413e-01,0.00000000000000000e
+00,-1.164395292155590987e-05,1.041345744204599989e-04,0.00000000000000
0000e+00
5.682463916354131470e+03,-6.799198240590144057e-05,-4.29175335218845918
6e-02,-5.372067430028695280e-03,0.0000000000000000e+00,6.593550579867
198114e-02,1.821417487542776117e-02,0.0000000000000000e+00,5.03493198
6613834241e-06,5.494377848020653811e-07,0.0000000000000000e+00,1.2675
44040759169821e-01,-8.444491800963731531e-01,0.0000000000000000e+00,-
1.976595172786609622e-01,7.155701186653791179e-01,0.00000000000000000e
+00,-1.484666679042683144e-05,1.037866707859474839e-04,0.00000000000000
0000e+00
6.818956699624956855e+03,-6.799198007889842971e-05,-4.27412994331737086
7e-02,-6.437663492824097222e-03,0.0000000000000000e+00,6.567938609949
550144e-02,1.911719685708915956e-02,0.0000000000000000e+00,5.01413813
3252074605e-06,6.804039640878594033e-07,0.0000000000000000e+00,1.5199
95849736978100e-01,-8.409685817044186829e-01,0.0000000000000000e+00,-
2.074593715006417816e-01,7.127903863023981046e-01,0.00000000000000000e
+00,-1.804344260627518796e-05,1.033575131732090388e-04,0.00000000000000
0000e+00
7.955449482895783149e+03,-6.799197739940073340e-05,-4.25331808852261600

```
3e-02,-7.498456960250221917e-03,0.000000000000000000e+00,6.541089999087
695839e-02,2.001661925475032092e-02,0.000000000000000000e+00,4.98930669
0888111660e-06,8.107760698363305196e-07,0.000000000000000000e+00,1.7718
68001457977682e-01,-8.368522357533669798e-01,0.000000000000000000e+00,-
2.172201604773888184e-01,7.098764548631837368e-01,0.000000000000000000e
+00,-2.123301262149882848e-05,1.028469381574750836e-04,0.00000000000000
0000e+00
9.091942266166610352e+03,-6.799197436361445717e-05,-4.22932574908030795
0e-02,-8.553643112901463882e-03,0.000000000000000000e+00,6.513009802957
837879e-02,2.091227273177493143e-02,0.000000000000000000e+00,4.96044758
1417798675e-06,9.404510408580805808e-07,0.000000000000000000e+00,2.0230
60188293255568e-01,-8.320987010665742423e-01,0.000000000000000000e+00,-
2.269400462359949244e-01,7.068288732493620552e-01,0.000000000000000000e
+00,-2.441408997396434162e-05,1.022547542538385996e-04,0.00000000000000
0000e+00
1.022843504943743392e+04,-6.799197096722032932e-05,-4.20216216510480578
0e-02,-9.602415275604505793e-03,0.000000000000000000e+00,6.483703309145
162630e-02,2.180398866148283016e-02,0.000000000000000000e+00,4.92757236
7574471590e-06,1.069325556389986072e-06,0.000000000000000000e+00,2.2734
70382324179473e-01,-8.267063261561379273e-01,0.000000000000000000e+00,-
2.366171985149528589e-01,7.036482155367483893e-01,0.000000000000000000e
+00,-2.758536541345228170e-05,1.015807424794736957e-04,0.00000000000000
0000e+00
```

We want to be able to read this simulation data back in to use in plots, and visualizations next week. So let's practice doing that with yet another function.

In [28]:
```
from hermite_library import read_hermite_solution_from_file
```

In [38]:
```
t_h2, e_h2, r_h2, v_h2 = read_hermite_solution_from_file('myPlanetSyste
m.txt')
```

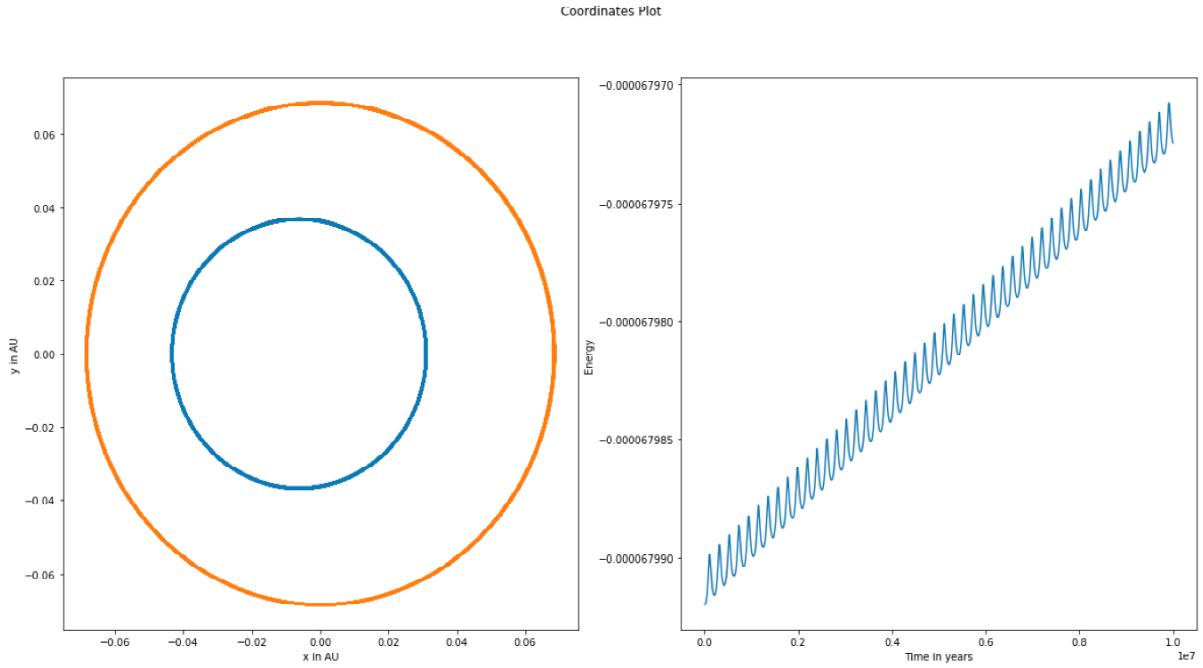
In [39]:
```
# let's plot!
fig, ax = plt.subplots(1, 2, figsize = (10*2, 10))
fig.suptitle('Coordinates Plot')

ax[0].set_xlabel('x in AU')
ax[0].set_ylabel('y in AU')

for i in range(len(planet_masses)):
    ax[0].plot(r_h2[i,0,:], r_h2[i,1,:], lw=3)

ax[1].set_xlabel('Time in years')
ax[1].set_ylabel('Energy')
# re-norm energy
ax[1].plot(t_h2, e_h2)

plt.show()
```



So, we see we get the same thing back as before! Whew!

Exercise

Practice writing some models and reading back in.

Bonus Exercise

Can you add planets to make this an unstable system?

In [ ]:
```

```