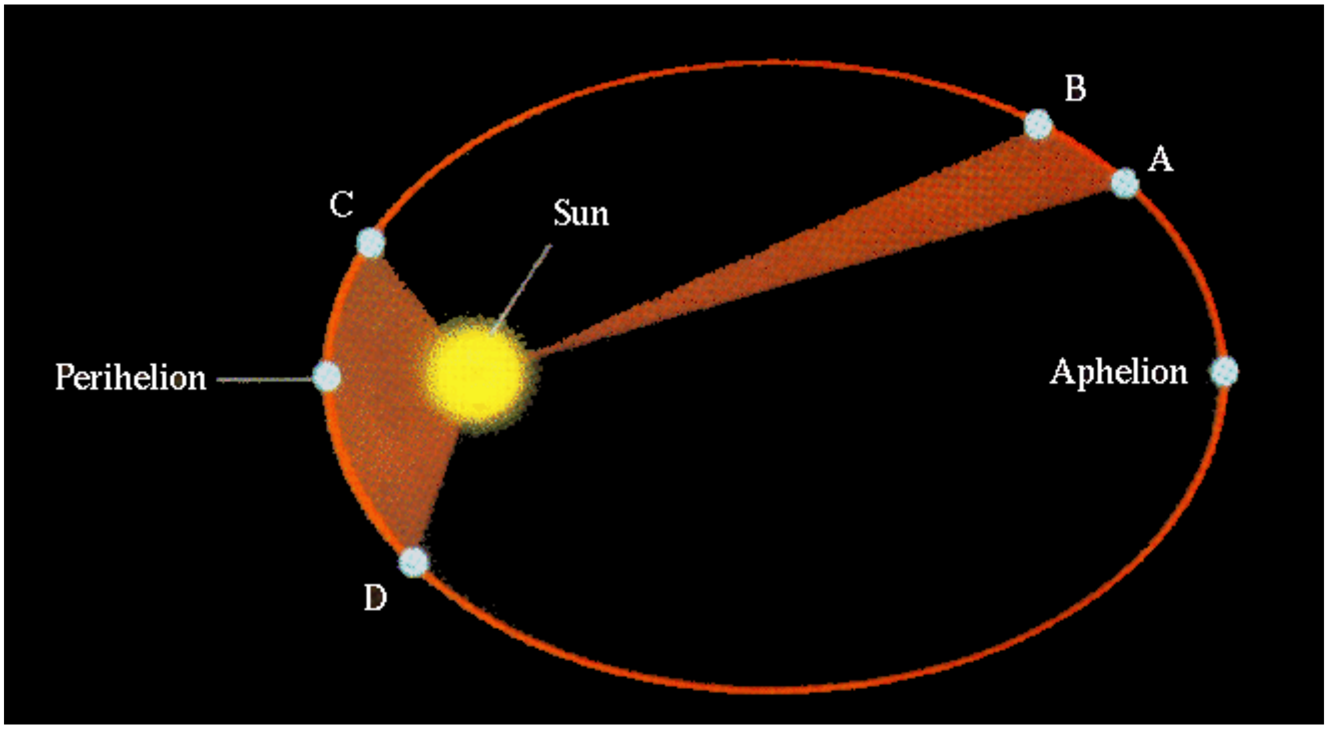
Day 3, Part 1: Two body motion, analytical and numeric

In [107]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Let's begin by defining the mass of the star we are interested in. We'll start with something that is the mass of the Sun.

In [108]:
```
# mass of particle 1 in solar masses
mass_of_star = 1.0
```

Let's also initialize $v_0 = v_p$ and $r_0 = r_p$ - values at perhelion. For reference let's check out the image again:



In [109]:
```
# distance of m2 at closest approach (pericenter)
rp = 1.0 # in AU - these are the distance from Sun to Earth units
# velocity of m2 at this closest approach distance
# we assume vp of the larger mass (m1) is negligable
vp = 35.0 # in km/s
```

We'll also need some conversion factors:

In [110]:
```
# 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
```

Note that astronomers use weird units - centimeters and grams - wait until we get to parsecs!

Now, let's convert units so we can calculate things.

In [111]:
```
mass_of_star = mass_of_star*MassOfSun
vp = vp*kmincm
rp = rp*AUinCM
```

Analytical solution

The first thing we want to do is construct the analytical solution. We'll use some relations to calculate the eccentricity and semi-major axis:

In [112]:
```
# analytically here are the constants we need to define to solve:
ecc = rp*vp*vp/(G*(mass_of_star)) - 1.0
a = rp/(1.0 - ecc)

# print some interesting things
print('Eccentricity = ' + str(ecc))
Porb = np.sqrt( 4.0*np.pi**2.0*a**3.0/(G*(mass_of_star)) )
print('Orbital Period = ' + str(Porb) + ' sec')
```

```
Eccentricity = 0.37293976625711744
Orbital Period = 63373146.36904171 sec
```

We also, for the time begin, only want to do elliptical/circular orbits, let's put in some checks for that:

In [113]:
```
if (ecc < 0):
    print('eccentricity is less than zero, we should stop!')
elif (ecc >= 1):
    print('eccentricity is greater than one')
    print(' this is a parabolic or hyperbolic orbit, we should stop!')
else:
    print('everything looks good, lets go!')
```

```
everything looks good, lets go!
```

Exercise

Use the above information to plot r(theta).

Recall: $r(\theta) = \frac{a(1-e^2)}{1+ecos(\theta)}$

Don't forget to label your axis with the correct coordinates!

Bonus: use other r_p and v_p values to see how your plot changes!

Euler's Numerical Solution

Redo this calculation with our numerical methods. A few things that might be useful. First, calculating the acceleration due to gravity:

In [114]:
```
# 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
```

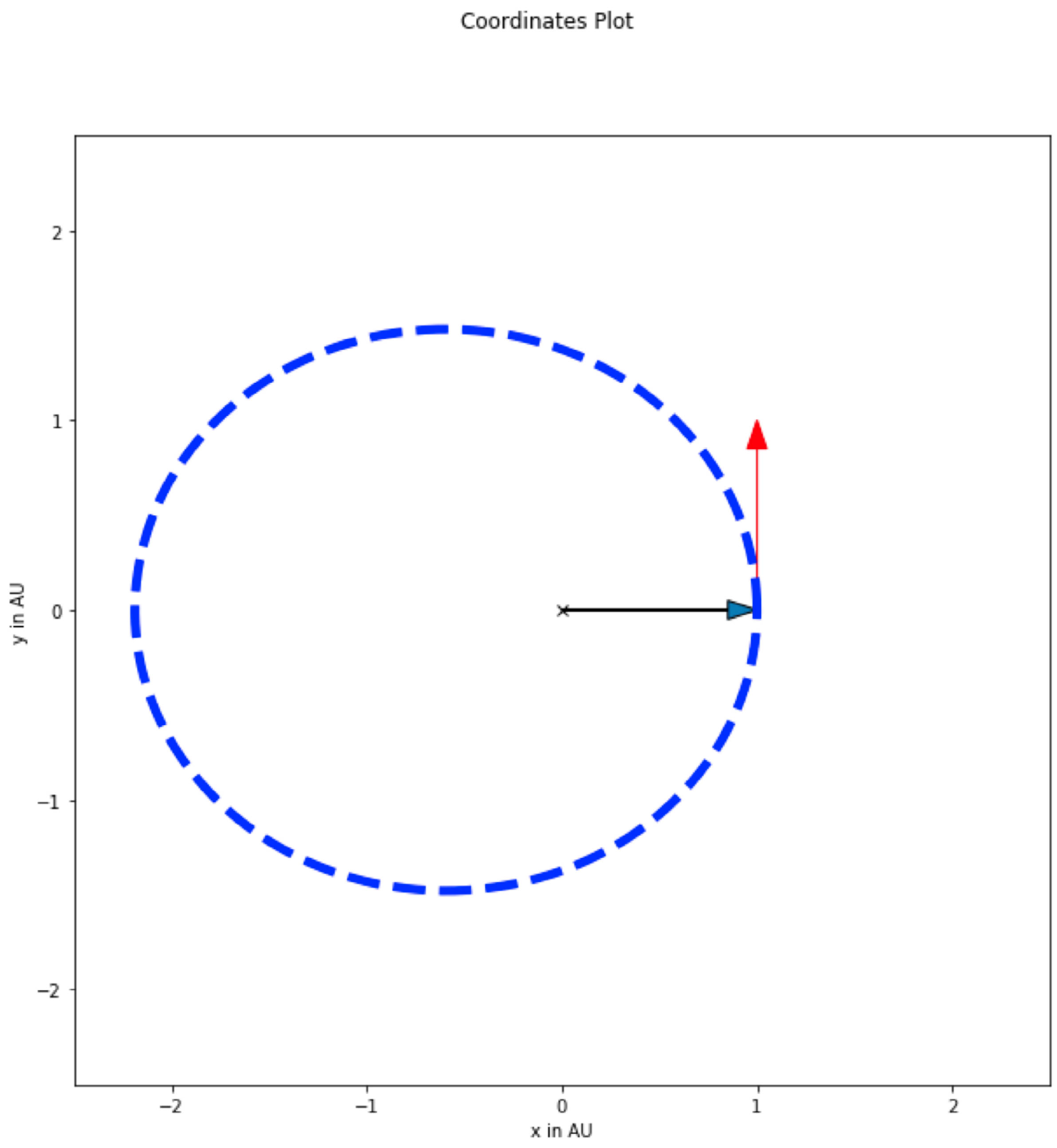
Let's recall that $\vec{r}_p$ is the vector from the star to the planet, and $\vec{v}_p$ is the velocity vector - this vector will be perpendicular to the orbit.

We can plot this on our elliptical plot before!

We'll use: https://matplotlib.org/3.1.0/api/_as_gen/matplotlib.pyplot.arrow.html (https://matplotlib.org/3.1.0/api/_as_gen/matplotlib.pyplot.arrow.html)

First we'll plot the elliptical path, with a dashed line.

In [115]:
```
# now, generate the theta array
ntheta = 500 # number of points for theta
th_an = np.linspace(0, 360, ntheta)

# now, create r(theta)
r_an = (a*(1-ecc*ecc))/(1.0 + ecc*np.cos( th_an*np.pi/180.0 ))

# for plotting -> x/y coords for m2
x_an = r_an*np.cos( th_an*np.pi/180.0 )/AUinCM
y_an = r_an*np.sin( th_an*np.pi/180.0 )/AUinCM

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

ax.plot(x_an, y_an, 'b--', linewidth=5)
ax.plot(0.0, 0.0, 'kx')
ax.set_xlabel('x in AU')
ax.set_ylabel('y in AU')

ax.set_xlim(-2.5, 2.5)
ax.set_ylim(-2.5, 2.5)

# now, let's plot these vectors
xarrow = 0; yarrow = 0 # coords of base
# dx/dy point to direction
dy = 0 # easy peasy, just along x-axis
# we can calculate r(theta = 0)
dx = (a*(1-ecc*ecc)/(1.0+ecc*np.cos(0)))/AUinCM
plt.arrow(xarrow, yarrow, dx, dy,head_width=0.1,
          length_includes_head=True)

# we can use this same location as the head of our velocity arrow
vxarrow = dx; vyarrow = 0
# what length should we make?  This is a little hard
#  to define since we are plotting something in
#  units of km/s and the x/y coords so we can
#  just choose a value
dy = 1.0 # arbitrary
dx = 0
plt.arrow(vxarrow, vyarrow, dx, dy, head_width=0.1,
         length_includes_head=True, color='red')

plt.show()
```



To use Euler's Method to calculate, we'll specify these initial conditions as 2D vectors:

In [116]:
```
r_0 = np.array([rp, 0])
v_0 = np.array([0, vp])
```

One thing we want to do is calculate for many time steps. How do we select Δt? Well, let's try fractions of an orbit.

In [117]:
```
Porb = np.sqrt( 4.0*np.pi**2.0*a**3.0/(G*(mass_of_star)) )
print(Porb, ' seconds')
```

```
63373146.36904171  seconds
```

Let's first start with time steps of 1e6 seconds. This means ~63 steps per orbit.

```
In [118]: delta_t = 1e6 # seconds
```

For how many timesteps? Let's start with 1 orbit, or 64 steps.

```
In [119]: n_steps = 64
```

Exercise

Use Euler's method to calculate this orbit. Plot it ontop of your analytical solution.

Recall:

$$\vec{r}_{i+1} = \vec{r}_i + \vec{v}_i \Delta t$$

and

$$\vec{v}_{i+1} = \vec{v}_i + \vec{a}_g(\vec{r}_i) \Delta t$$

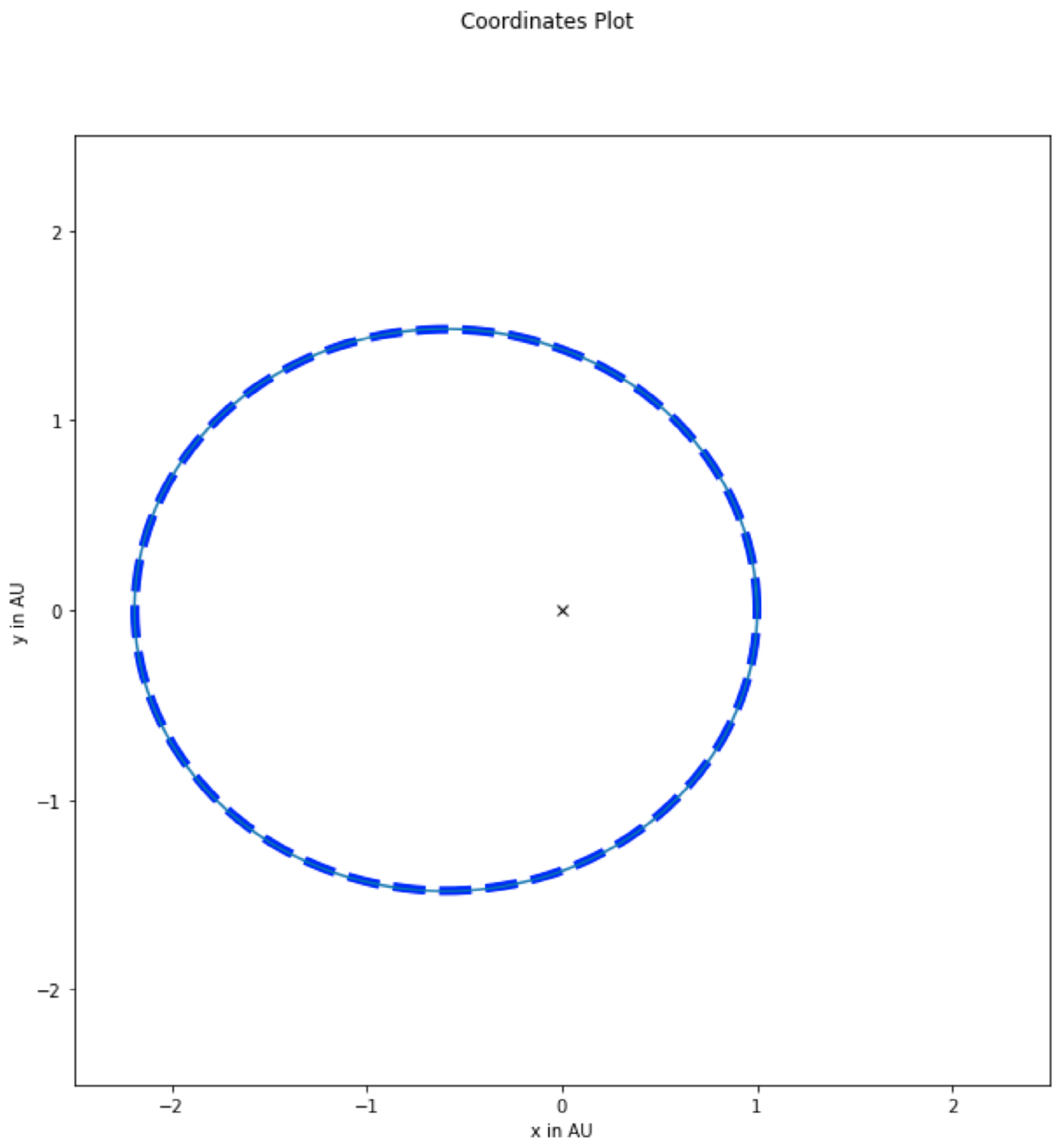
Bonus: what happens if you increase the number of steps? Or changing Δt?

Possible Ans

```
In [120]: ri = r_0
          vi = v_0

          r = []
          for i in range(n_steps):
              # ...

          # what does it look like?
          r = np.array(r)

          r
```

```
  File "<ipython-input-120-fd69df767af3>", line 9
    r = np.array(r)
    ^
IndentationError: expected an indented block
```

In [121]:
```
# now, generate the theta array
ntheta = 500 # number of points for theta
th_an = np.linspace(0, 360, ntheta)

# now, create r(theta)
r_an = (a*(1-ecc*ecc))/(1.0 + ecc*np.cos( th_an*np.pi/180.0 ))

# for plotting -> x/y coords for m2
x_an = r_an*np.cos( th_an*np.pi/180.0 )/AUinCM
y_an = r_an*np.sin( th_an*np.pi/180.0 )/AUinCM

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

ax.plot(x_an, y_an, 'b--', linewidth=5)
ax.plot(0.0, 0.0, 'kx')
ax.set_xlabel('x in AU')
ax.set_ylabel('y in AU')

ax.set_xlim(-2.5, 2.5)
ax.set_ylim(-2.5, 2.5)

# plot Euler's solution
ax.plot(r[:,0]/AUinCM, r[:,1]/AUinCM)

plt.show()
```



Exercise

How small does Δt need to be for an accurate solution? How many orbits can you get?

Exercises

Let's calculate some orbits, analytical and numerical for our planetary system. Again, we'll assume our Sun is massive and centered at the origin, and the planets are much less massive and orbit around the Sun.

1. Calculate the orbit of the Earth

   - T, the orbital period = 365 days, i.e. 1 year
   - the eccentricity, ecc = 0.02

   Recall: $T = \sqrt{\frac{4\pi^2 a^3}{GM_{star}}}$

2. Bonus - others. Note, you probably want to write a function:

   - Mercury, T = 0.48 years, eccentricity = 0.21
   - Venus, T = 0.62 years, eccentricity = 0.01
   - Mars, T = 1.88 years, eccentricity = 0.09
   - Jupiter, T = 11.86 years, eccentricity = 0.05
   - Saturn, T = 29.46 years, eccentricity = 0.05
   - Uranus, T = 84.02 years, eccentricity = 0.05
   - Neptune, T = 164.8 years, eccentricity = 0.01
   - Pluto, T = 248.0 years, eccentricity = 0.25

   Questions: how does Δt and change with eccentricity and period?

   Plot the whole solar system, both analytically and numerically. Think strategically - what things do you need to make as "variables" in your function? (T, ecc... what else?)

```
In [ ]:
```

```
In [ ]:
```