Lab01_Q1

September 14, 2018

1 PHY407 Lab01: Python Basics

1.0.1 Andrew Hardy and Brayden Kell

1.0.2 September 2018

1.1 1. Question 1

1.1.1 1.1 Part a: Discretization of ODE sytem

As an approximation, we can write the following expression of Newtonian gravitational force law:

$$\vec{F_g} = -\frac{GM_sM_p}{r^3}(x\hat{x} + y\hat{y}) \qquad (1) \tag{1}$$

where r is the distance planet p is from the sun s is orbits ($M_s >> M_p$). From this, we have the following system of ODEs governing the orbital path of the planet about the sun:

$$\frac{dv_x}{dt} = -\frac{GM_sx}{r^3} \qquad (2) \tag{2}$$

$$\frac{dv_y}{dt} = -\frac{GM_sy}{r^3} \qquad (3) \tag{3}$$

$$\frac{dx}{dt} = v_x \qquad (4) \tag{4}$$

$$\frac{dy}{dt} = v_y \qquad (5) \tag{5}$$

Note: $r = \sqrt{x^2 + y^2}$, of course. We can discretize the above system to perform numerical integration according to the Euler method, which in generality represents position and velocity given by $\frac{dx}{dt} = f(t, v)$ and $\frac{dv}{dt} = g(t, x)$, respectively in the following forms:

$$v_{i+1} = v_i + g(t_i, x_i)\Delta t \qquad (6) \tag{6}$$

$$x_{i+1} = x_i + f(t_i, v_i)\Delta \qquad (7) \tag{7}$$

Equations (4) and (5) in Euler-Cromer form become:

$$x_{i+1} = x_i + v_{x,i}\Delta t \qquad (8) \tag{8}$$

and

$$y_{i+1} = y_i + v_{y,i}\Delta t \qquad (9) \tag{9}$$

Equations (1) and (2) are discretized as:

$$v_{x,i+1} = v_{x,i} - \frac{GM_s}{r^3} x_i \Delta t \qquad (10) \tag{10}$$

and

$$v_{y,i+1} = v_{y,i} - \frac{GM_s}{r^3} y_i \Delta t \qquad (11) \tag{11}$$

where x_i and y_i given as in equations (8) and (9), respectively. However, it is known that this method is unstable, so we will instead use the slightly revised version, often referred to as the Euler-Cromer method where $v_i \to v_{i+1}$ in equation (7).

1.1.2 1.2 Part b: Pseudo-code for numerical integration to model Newtonian orbital path and velocity

1. Import required packages for handling arrays and plotting tools.
2. Define values for necessary constants like G, time-step, and duration of time for which you want to model in appropriate units and instantiate an array of time points accordingly. Note: we will work in units were mass is represented in units where mass of the sun is 1, and our units of length and time will be astronomical units and years, respectively.

3. Define initial conditions for position components x, y and velocity components v_x, v_y.
4. Instantiate empty arrays of length t for x, y, v_x, and v_y to preallocate memory for them before numerical integration. Store the respective initial conditions previously defined at the 0^{th} index of each of these arrays.
5. Within a for-loop from 0 to 1 less than the length of t (1 less since at each index i you are approximating the $(i+1)^{th}$ value of position and velocity, perform numerical integration, storing the approximated values for x, y, v_x, and v_y in their respective pre-defined arrays.
6. Calculate the magnitude of the angular momentum via the cross product of the position and velocity (neglecting constant mass). The constant angular momentum insures our system is physical.
7. Finally, plot x vs. y to get a graphical representation of the planet's orbit about the sun, and plot v_x vs. time as well as v_y vs. to to visualize how the planet's velocity changes over time.
8. Plot Angular Momentum amplitude (ignoring constant mass) as a function of time

1.1.3 1.3 Part c: Implementation and executation of code

See commented Python code below and plot outputs.

```
In [1]: #1. import required packages
        import numpy as np #import numpy
        import matplotlib.pyplot as plt #import stuff for plotting
        import time
        import seaborn as sns
        sns.set()

In [2]: #2.Define some constants and time array
        G = 39.5 #gravitational constant in units of AU^3M_s^-1yr^-2 where M_s represents the
        Ms = 1 #for definiteness, defining Ms as 1 in units of 2.0x10^30kg
```

```
dt = 0.0001 #time step
duration = 1 #1year
num = round(duration/dt)#number of time steps
t = np.linspace(0, duration, num)
```

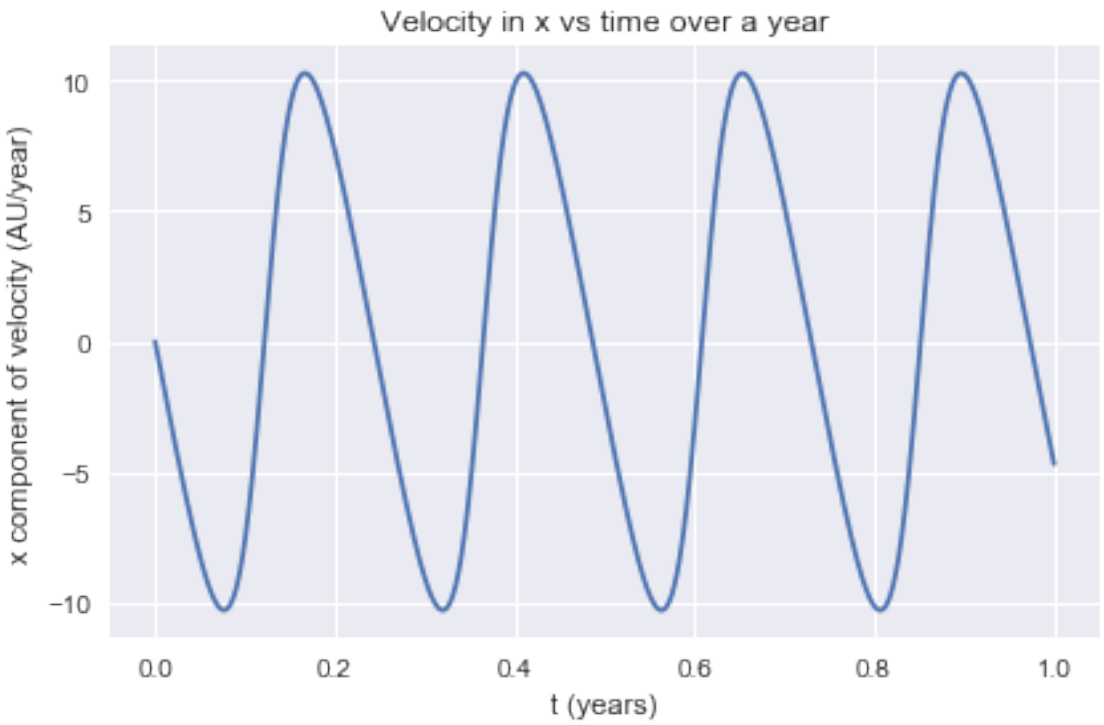
In [3]:
```
#3. Define initial conditions
x_init = 0.47 #AU
y_init = 0.0 #AU
Vx_init = 0.0 #AU/year
Vy_init = 8.17 #AU/yearx = np.empty(num)
```

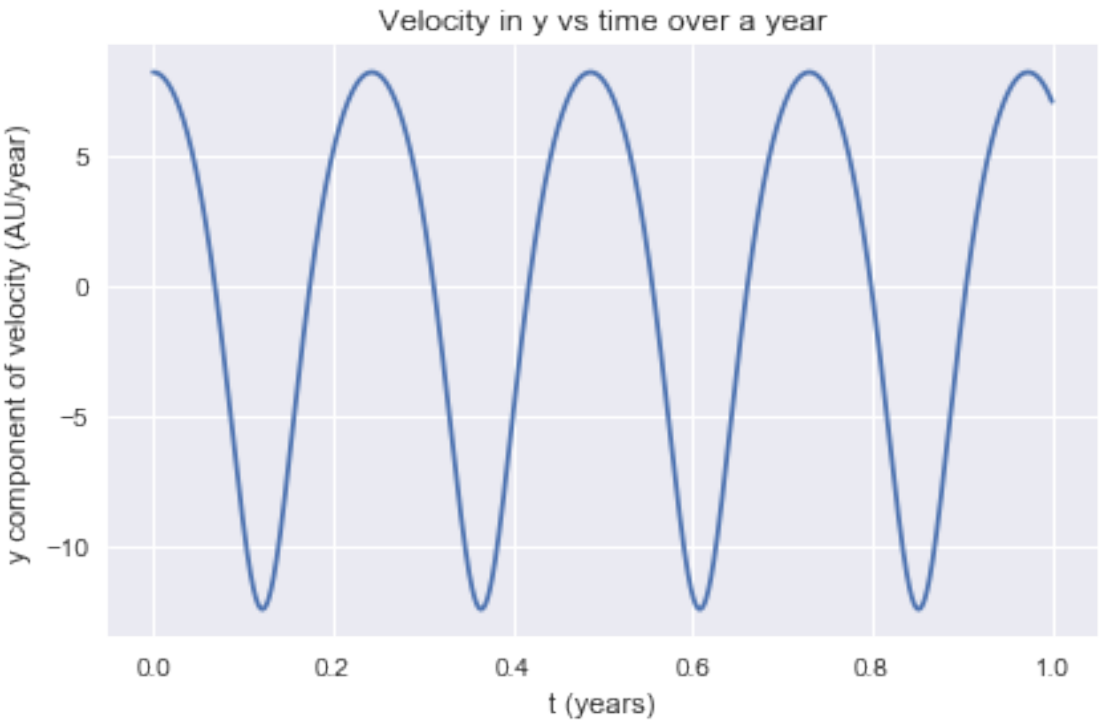
In [4]:
```
#4. Instantiate empty arrays for x, y, v_x, v_y to preallocate memory
x = np.empty(num)
y = np.empty(num)
Vx = np.empty(num)
Vy = np.empty(num)
r = np.empty(num)
L = np.empty(num)
```

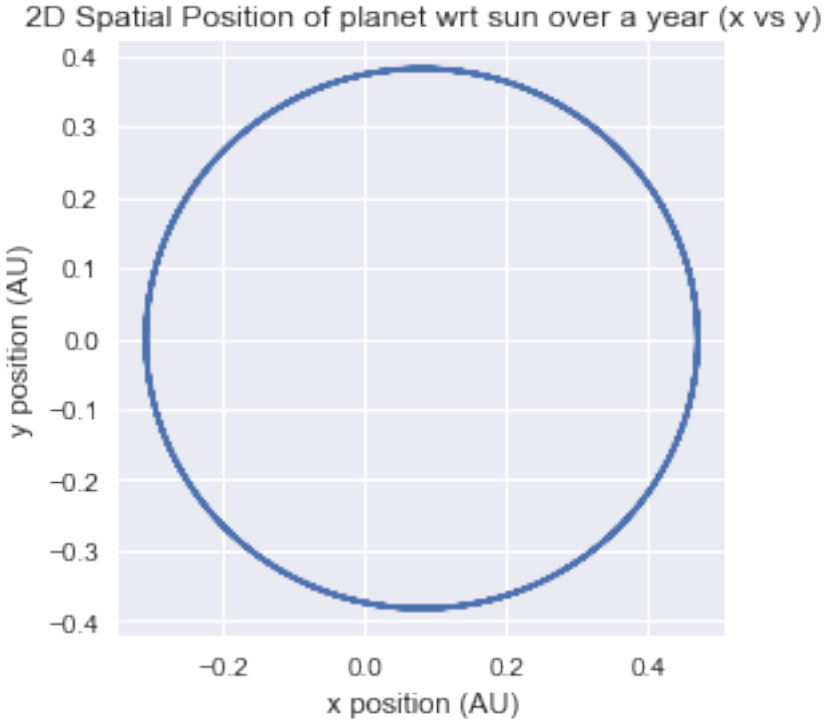
In [5]:
```
#5. Numerical integration (Euler-Cromer method) NEWTON
#storing the initial conditions in the first entry of arrays to hold position and velo
x[0] = x_init
y[0] = y_init
Vx[0] = Vx_init
Vy[0] = Vy_init
r[0] = np.sqrt(x_init**2+y_init**2)
L[0] = np.sqrt((x[0]*Vy[0]-y[0]*Vx[0])**2)  # angular momentum magnitude, from cross p
#integrate to get position and velocity components
for i in range(num-1):
    r[i+1] = np.sqrt(x[i]**2 + y[i]**2)
    Vx[i+1] = Vx[i] - G*Ms*x[i]*dt/r[i+1]**3
    Vy[i+1] = Vy[i] - G*Ms*y[i]*dt/r[i+1]**3
    x[i+1] = x[i] + Vx[i+1]*dt
    y[i+1] = y[i] + Vy[i+1]*dt
    L[i+1] = np.sqrt((x[i+1]*Vy[i+1]-y[i+1]*Vx[i+1])**2)
```

In [6]:
```
#6. Plot results of integration
plt.figure()

plt.plot(t,Vx)
plt.title('Velocity in x vs time over a year')
plt.xlabel('t (years)')
plt.ylabel('x component of velocity (AU/year)')
plt.tight_layout()

plt.show()

plt.figure()
```

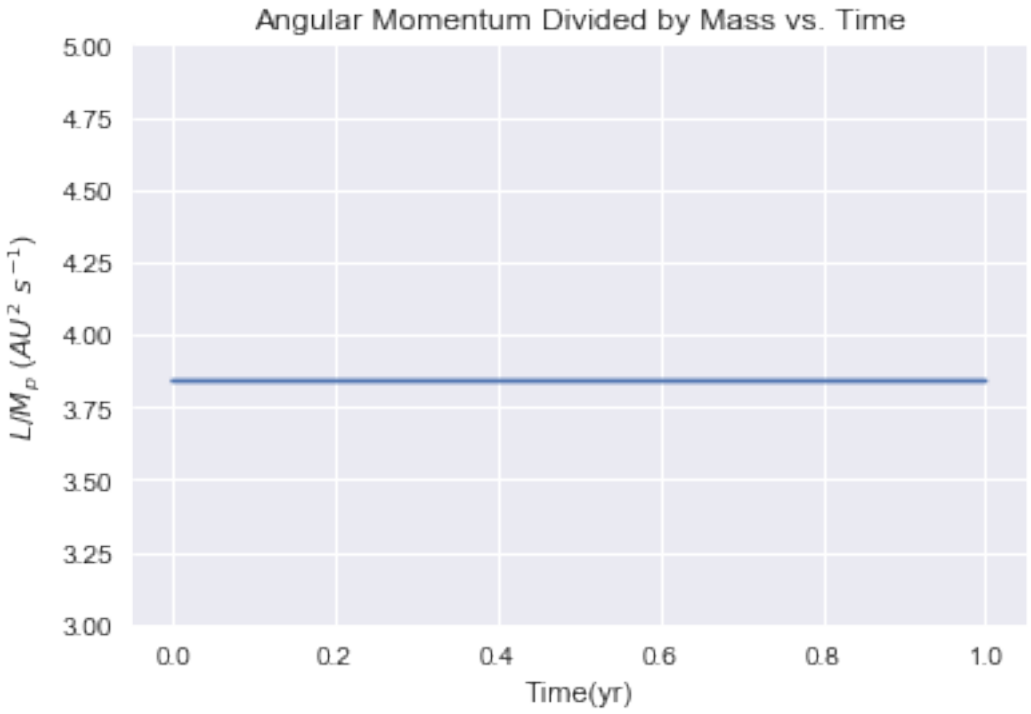
```
plt.plot(t,Vy)
plt.title('Velocity in y vs time over a year')
plt.xlabel('t (years)')
plt.ylabel('y component of velocity (AU/year)')
plt.tight_layout()

plt.show()

plt.figure()

plt.plot(x,y)
plt.title('2D Spatial Position of planet wrt sun over a year (x vs y)')
plt.xlabel('x position (AU)')
plt.ylabel('y position (AU)')
plt.axis('scaled')
plt.tight_layout()

plt.show()
```




Velocity in y vs time over a year
y component of velocity (AU/year)
5
0
−5
−10
0.0
0.2
0.4
0.6
0.8
1.0
t (years)


2D Spatial Position of planet wrt sun over a year (x vs y)
y position (AU)
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
−0.2
0.0
0.2
0.4
x position (AU)

In [7]:
```
#Check if anglular momentum conserved
# Note, not considering constant scaling factor of mass
plt.figure()
plt.plot(t, L)
plt.title('Angular Momentum Divided by Mass vs. Time')
plt.xlabel('Time(yr)')
plt.ylim(ymin = 3, ymax = 5)
plt.ylabel('$L/M_p$ ($AU^2$ $s^{-1}$)')
plt.show()

print(L)
```




```
[3.8399 3.8399 3.8399 ... 3.8399 3.8399 3.8399]
```

1.1.4 1.4 Part d: Revised code to model orbit using general relativity law of gravitational force approximation

Again, see commented Python code below and plot outputs.

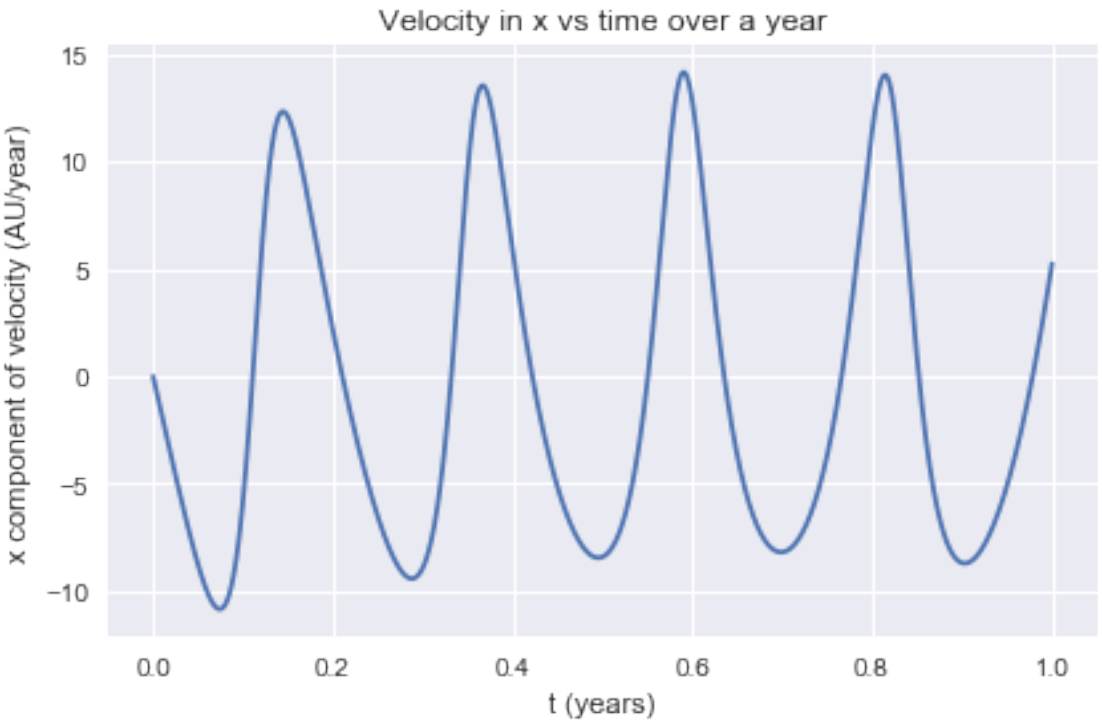
In [8]:
```
#4. Instantiate empty arrays for x, y, v_x, v_y to preallocate memory
x_GR = np.zeros(num)
y_GR = np.zeros(num)
Vx_GR = np.zeros(num)
```

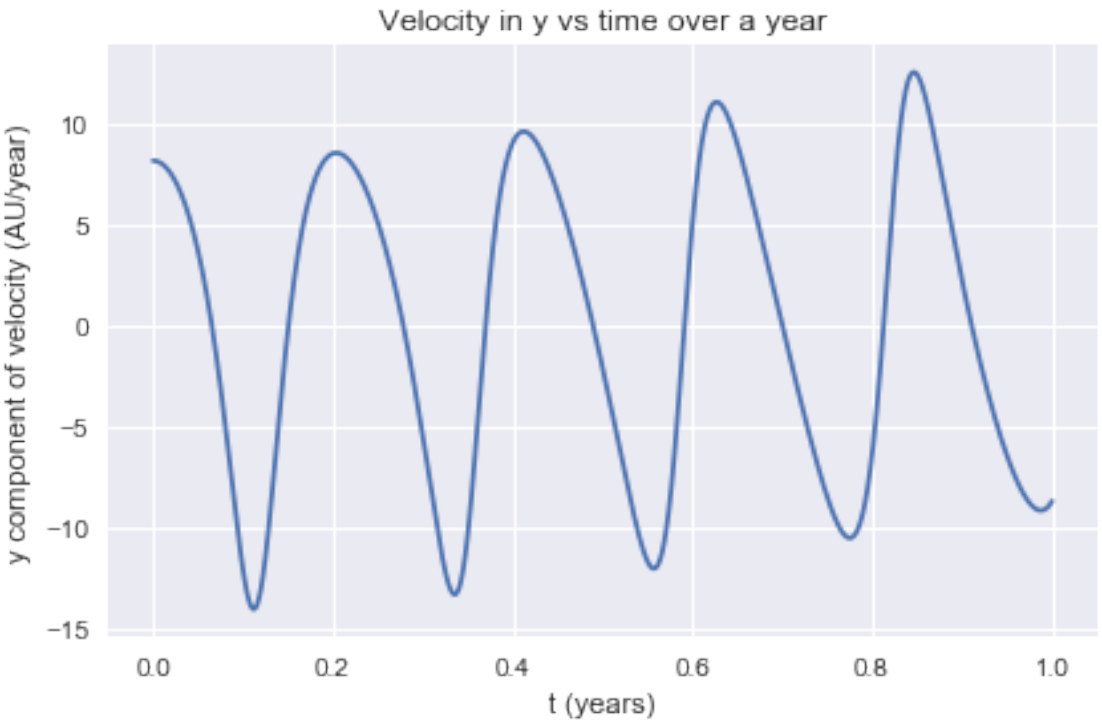
```
Vy_GR = np.zeros(num)
r_GR = np.zeros(num)
L_GR = np.zeros(num)
```

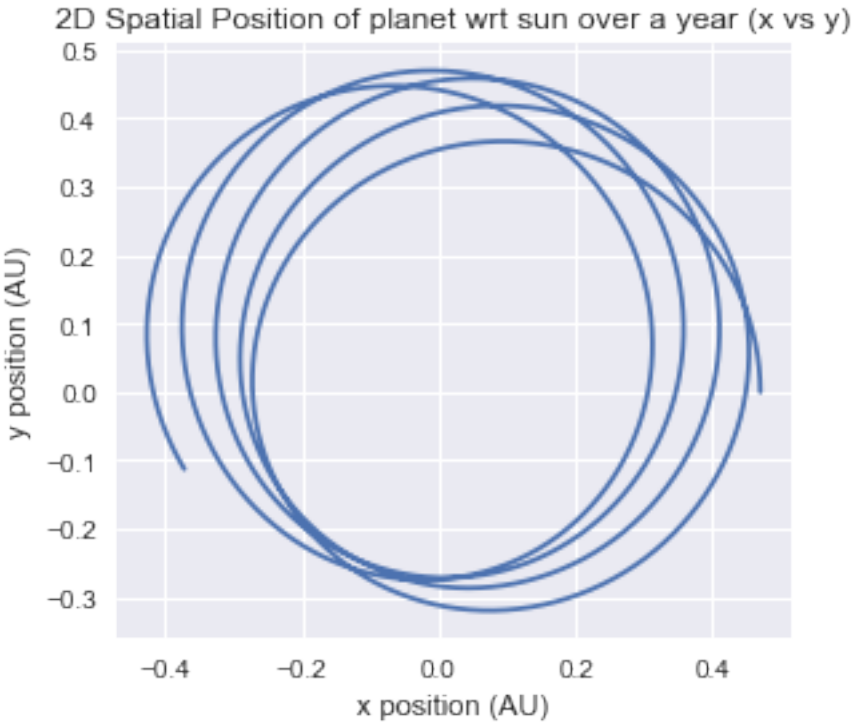
In [9]:
```
#5. (GR) Numerical integration (Euler-Cromer method) GENERAL RELATIVITY
#storing the initial conditions in the first entry of arrays to hold position and velo
x_GR[0] = x_init
y_GR[0] = y_init
Vx_GR[0] = Vx_init
Vy_GR[0] = Vy_init
L_GR[0] = np.sqrt((x_GR[0]*Vy_GR[0]-y_GR[0]*Vx_GR[0])**2)  # angular momentum magnitud
#define GR correction constant alpha for our planetary scenario
alpha = 0.01 #AU^2
#NOTE: must only perform numerical integration steps and plotting of results again. Ev
#integrate to get position and velocity components

for i in range(num-1):
    r_GR = np.sqrt(x_GR[i]**2 + y_GR[i]**2)
    Vx_GR[i+1] = Vx_GR[i] - G*Ms*(1+alpha/r_GR**2)*x_GR[i]*dt/r_GR**3
    Vy_GR[i+1] = Vy_GR[i] - G*Ms*(1+alpha/r_GR**2)*y_GR[i]*dt/r_GR**3
    x_GR[i+1] = x_GR[i] + Vx_GR[i+1]*dt
    y_GR[i+1] = y_GR[i] + Vy_GR[i+1]*dt
    L_GR[i+1] = np.sqrt((x_GR[i+1]*Vy_GR[i+1]-y_GR[i+1]*Vx_GR[i+1])**2)
```

In [10]:
```
#6. (GR) Plot results of integration
plt.figure()

plt.plot(t,Vx_GR)
plt.title('Velocity in x vs time over a year')
plt.xlabel('t (years)')
plt.ylabel('x component of velocity (AU/year)')
plt.tight_layout()

plt.show()

plt.figure()

plt.plot(t,Vy_GR)
plt.title('Velocity in y vs time over a year')
plt.xlabel('t (years)')
plt.ylabel('y component of velocity (AU/year)')
plt.tight_layout()

plt.show()

plt.figure()

plt.plot(x_GR,y_GR)
```

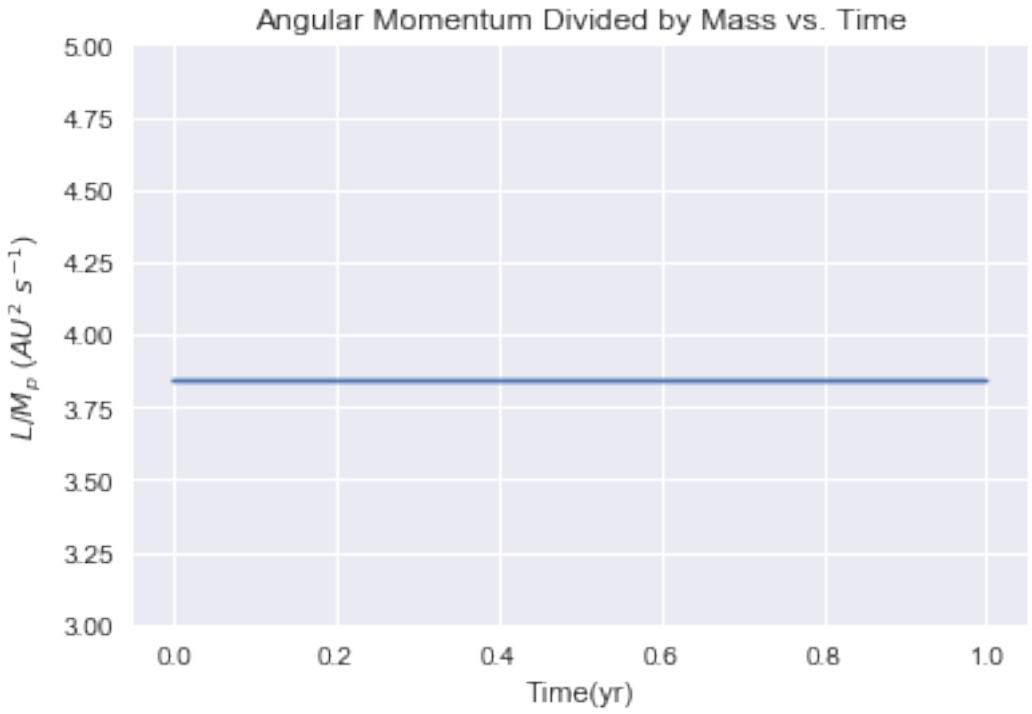
```
plt.title('2D Spatial Position of planet wrt sun over a year (x vs y)')
plt.xlabel('x position (AU)')
plt.ylabel('y position (AU)')
plt.axis('scaled')
plt.tight_layout()

plt.show()
```




Velocity in y vs time over a year
y component of velocity (AU/year)
10
5
0
−5
−10
−15
0.0
0.2
0.4
0.6
0.8
1.0
t (years)


2D Spatial Position of planet wrt sun over a year (x vs y)
y position (AU)
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
−0.2
0.0
0.2
0.4
x position (AU)

In [11]:
```
#Check if anglular momentum conserved
# Note, not considering constant scaling factor of mass
plt.figure()
plt.plot(t, L_GR)
plt.ylim(ymin = 3, ymax = 5)
plt.title('Angular Momentum Divided by Mass vs. Time')
plt.xlabel('Time(yr)')
plt.ylabel('$L/M_p$ ($AU^2$ $s^{-1}$)')
plt.show()

print(L)
```




```
[3.8399 3.8399 3.8399 ... 3.8399 3.8399 3.8399]
```