Hw2

February 13, 2023

0.1 Tory Smith

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint, solve_ivp
import sympy as sym
```

1) Integrate the equations of motion for one day using the same initial conditions as in Homework #1. However, now include the Earth's oblateness, i.e., the J2 term

1) a. Use the MATLAB symbolic toolbox to compute the Cartesian partial derivatives of U. Compute U/ x by hand (include the derivation in the write-up) and compare your results with MATLAB

Please see the attached solution

[ ]:
```
x = sym.Symbol('x')
y = sym.Symbol('y')
z = sym.Symbol('z')
mu = sym.Symbol('mu')#[km^2/s^3]
J_2 = sym.Symbol('J_2')
R_earth = sym.Symbol('R_earth') #[km]
r = (x**2 + y**2 + z**2)**(1/2)
phi = z/r
dUdx = sym.diff(mu/r*(1-J_2*(R_earth/r)**2*(3/2*phi**2-1/2)), x)
dUdy = sym.diff(mu/r*(1-J_2*(R_earth/r)**2*(3/2*phi**2-1/2)), y)
dUdz = sym.diff(mu/r*(1-J_2*(R_earth/r)**2*(3/2*phi**2-1/2)), z)

print("dU/dx:", dUdx)
print("dU/dy:", dUdy)
print("dU/dz:", dUdz)
```

```
dU/dx: -1.0*mu*x*(-J_2*R_earth**2*(1.5*z**2/(x**2 + y**2 + z**2)**1.0 -
0.5)/(x**2 + y**2 + z**2)**1.0 + 1)/(x**2 + y**2 + z**2)**1.5 +
mu*(3.0*J_2*R_earth**2*x*z**2/(x**2 + y**2 + z**2)**3.0 +
2.0*J_2*R_earth**2*x*(1.5*z**2/(x**2 + y**2 + z**2)**1.0 - 0.5)/(x**2 + y**2 +
z**2)**2.0)/(x**2 + y**2 + z**2)**0.5
dU/dy: -1.0*mu*y*(-J_2*R_earth**2*(1.5*z**2/(x**2 + y**2 + z**2)**1.0 -
0.5)/(x**2 + y**2 + z**2)**1.0 + 1)/(x**2 + y**2 + z**2)**1.5 +
mu*(3.0*J_2*R_earth**2*y*z**2/(x**2 + y**2 + z**2)**3.0 +
```

```
2.0*J_2*R_earth**2*y*(1.5*z**2/(x**2 + y**2 + z**2)**1.0 - 0.5)/(x**2 + y**2 +
z**2)**2.0)/(x**2 + y**2 + z**2)**0.5
dU/dz: -1.0*mu*z*(-J_2*R_earth**2*(1.5*z**2/(x**2 + y**2 + z**2)**1.0 -
0.5)/(x**2 + y**2 + z**2)**1.0 + 1)/(x**2 + y**2 + z**2)**1.5 +
mu*(2.0*J_2*R_earth**2*z*(1.5*z**2/(x**2 + y**2 + z**2)**1.0 - 0.5)/(x**2 + y**2
+ z**2)**2.0 - J_2*R_earth**2*(-3.0*z**3/(x**2 + y**2 + z**2)**2.0 + 3.0*z/(x**2
+ y**2 + z**2)**1.0)/(x**2 + y**2 + z**2)**1.0)/(x**2 + y**2 + z**2)**0.5
```

[ ]:
```
t = np.arange(0, 86420, 20)
r=np.array([-2436.45, -2436.45, 6891.037]) #[km]
r_dot = np.array([5.088611, -5.088611, 0.0]) #[km/s]

#equations of motion
def satellite_motion_J2(t, R):
    mu = 398600.4 #[km^2/s^3]
    J_2 = 0.00108248
    R_earth = 6378.145 #[km]
    x = R[0]
    y = R[1]
    z = R[2]
    dUdx = -1.0*mu*x*(-J_2*R_earth**2*(1.5*z**2/(x**2 + y**2 + z**2)**1.0 - 0.
 ↪5)/(x**2 + y**2 + z**2)**1.0 + 1)/(x**2 + y**2 + z**2)**1.5 \
        + mu*(3.0*J_2*R_earth**2*x*z**2/(x**2 + y**2 + z**2)**3.0 + 2.
 ↪0*J_2*R_earth**2*x*(1.5*z**2/(x**2 + y**2 + z**2)**1.0 - 0.5)/(x**2 \
        + y**2 + z**2)**2.0)/(x**2 + y**2 + z**2)**0.5
    dUdy = -1.0*mu*y*(-J_2*R_earth**2*(1.5*z**2/(x**2 + y**2 + z**2)**1.0 - 0.
 ↪5)/(x**2 + y**2 + z**2)**1.0 + 1)/(x**2 + y**2 + z**2)**1.5 \
        + mu*(3.0*J_2*R_earth**2*y*z**2/(x**2 + y**2 + z**2)**3.0 + 2.
 ↪0*J_2*R_earth**2*y*(1.5*z**2/(x**2 + y**2 + z**2)**1.0 - 0.5)/(x**2 \
        + y**2 + z**2)**2.0)/(x**2 + y**2 + z**2)**0.5
    dUdz = -1.0*mu*z*(-J_2*R_earth**2*(1.5*z**2/(x**2 + y**2 + z**2)**1.0 - 0.
 ↪5)/(x**2 + y**2 + z**2)**1.0 + 1)/(x**2 + y**2 + z**2)**1.5\
        + mu*(2.0*J_2*R_earth**2*z*(1.5*z**2/(x**2 + y**2 + z**2)**1.0 - 0.5)/
 ↪(x**2 + y**2 + z**2)**2.0 - J_2*R_earth**2*(-3.0*z**3/(x**2 \
            + y**2 + z**2)**2.0 + 3.0*z/(x**2 + y**2 + z**2)**1.0)/(x**2 + y**2␣
 ↪+ z**2)**1.0)/(x**2 + y**2 + z**2)**0.5
    dydt = np.concatenate((R[3:6], np.array([dUdx, dUdy, dUdz])))
    return dydt
#initial Conditions
y0 = np.concatenate([r, r_dot])

#numeric integration
sol_J2 = solve_ivp(satellite_motion_J2, [0, 86420], y0, t_eval=t, rtol=3E-14,␣
 ↪atol=1E-16)
print(sol_J2.y.T[-1])
```

```
[-5.75149901e+03  4.72114371e+03  2.04603584e+03 -7.97658631e-01
```

```
-3.65651311e+00  6.13961202e+00]
```

1b) Plot the orbital elements a, e, i, Ω, , and Tp for one day at 20 second intervals, where Tp is the time of perigee passage and include the figure in your write-up. Be sure to label your axes and ensure that everything in each figure is readable. Using your insight from the two-body model, what conclusions can you draw about the J2 effect on Keplerian orbital elements?

The J2 effects seem to cause periodic changes in the orbital elements that correspond to the location of the satellite in its orbit. This makes sense as the satellite's velocity and acceleration will change as the effects of gravity change depending on where the satellite is relative to the Earth.

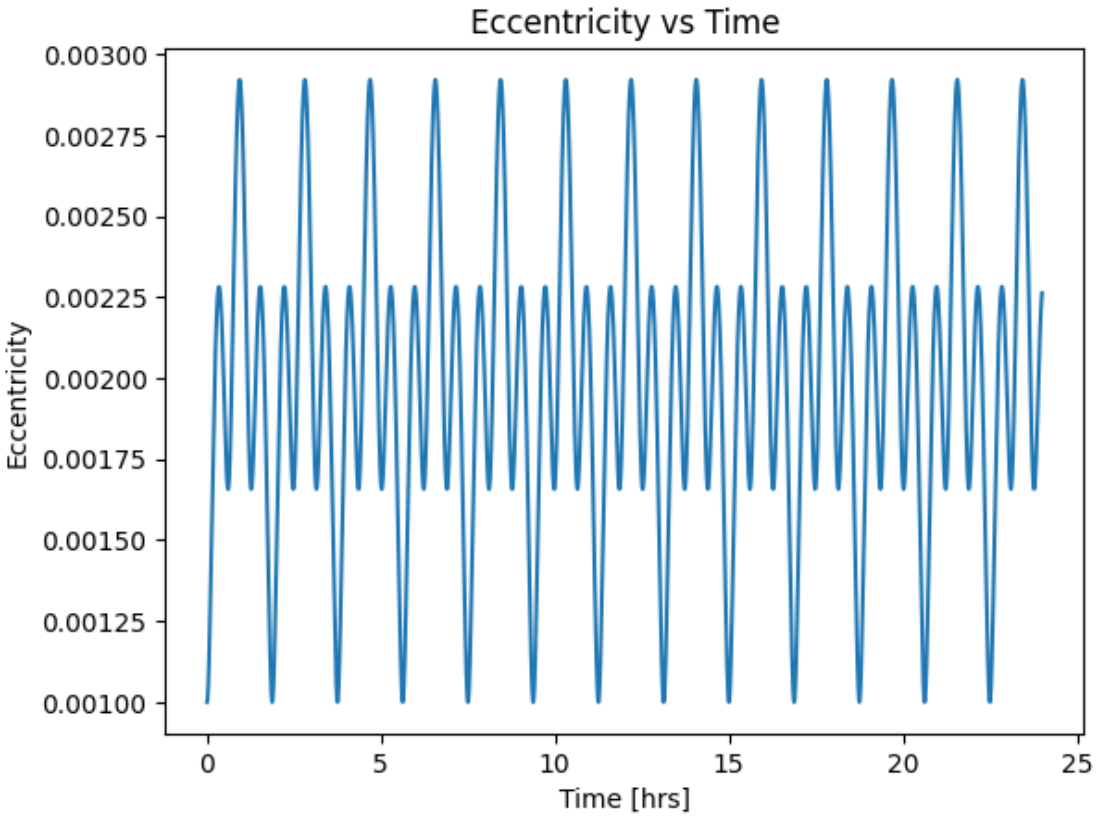
[ ]:
```python
def state_vectors_to_orbital_elements(state_vector, mu):
    """Computes the orbital elements of a satellite given a state vector

        Parameters:
        R - Position Vector (3x1) [km]
        R_dot - Velocity Vector (3x1) [km/s]
        mu = Gravitational Constant for the Body (float)

        Returns 6x1 tuple:
        ecc - Eccentricity [unitless]
        a - Semi-major axis [km]
        inc - Inclination [rad]
        raan - Right Ascension of the Ascending Node [rad]
        w - argument of periapsis [rad]
        nu - true anamoly - [rad]
    """
    R = state_vector[0:3]
    R_dot = state_vector[3:6]
    h = np.cross(R, R_dot)
    A = -1*(np.cross(h, R_dot) + mu*R/np.linalg.norm(R))
    P = np.linalg.norm(h)**2/mu
    n = np.cross(np.array([0, 0, 1]), h)
    e = np.cross(R_dot, h)/mu - R/np.linalg.norm(R)

    #eccentricity
    ecc = np.linalg.norm(A)/mu
    #semi-major-axis
    a = P/(1-ecc**2)
    #inclination
    inc = np.arccos(np.dot(h/np.linalg.norm(h), np.array([0, 0, 1])))
    #right ascension of the ascending node
    raan = np.arccos(np.dot(np.array([1, 0, 0]), n/np.linalg.norm(n)))
    #argument of perigee
    w = np.arccos(np.dot(n, e)/(np.linalg.norm(n)*np.linalg.norm(e)))
    #true anamoly
    nu = np.arccos(np.dot(R, e)/(np.linalg.norm(R)*np.linalg.norm(e)))
```

```
    return ecc, a, inc, raan, w, nu
```

[ ]:
```
mu = 398600.4 #[km^2/s^3]
orbital_elements_J2 = np.
 ↪array([state_vectors_to_orbital_elements(state_vector,mu) for state_vector␣
 ↪in sol_J2.y.T])
```

[ ]:
```
#eccentricity
plt.plot(t/(60*60), orbital_elements_J2[:, 0])
plt.title('Eccentricity vs Time')
plt.xlabel('Time [hrs]')
plt.ylabel('Eccentricity')
```

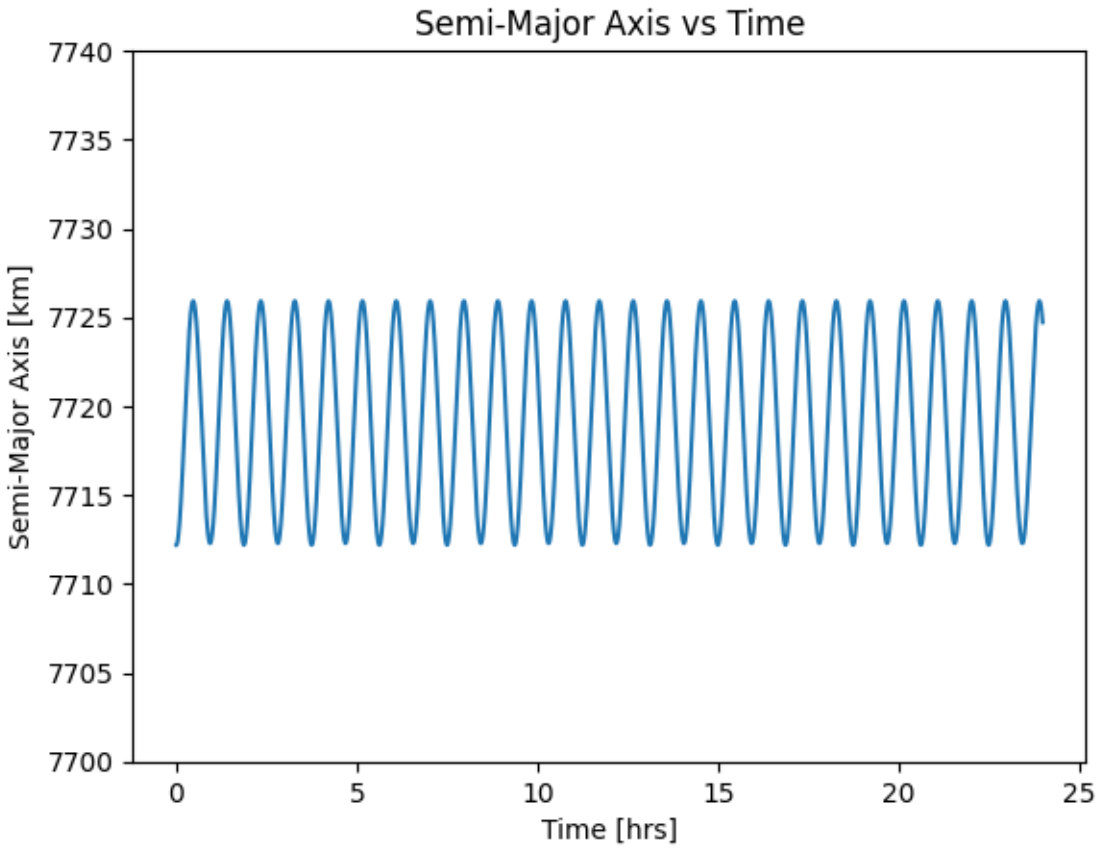
[ ]:
```
Text(0, 0.5, 'Eccentricity')
```



[ ]:
```
#semi-major axis
plt.plot(t/(60*60), orbital_elements_J2[:, 1])
plt.title('Semi-Major Axis vs Time')
plt.xlabel('Time [hrs]')
plt.ylabel('Semi-Major Axis [km]')
```

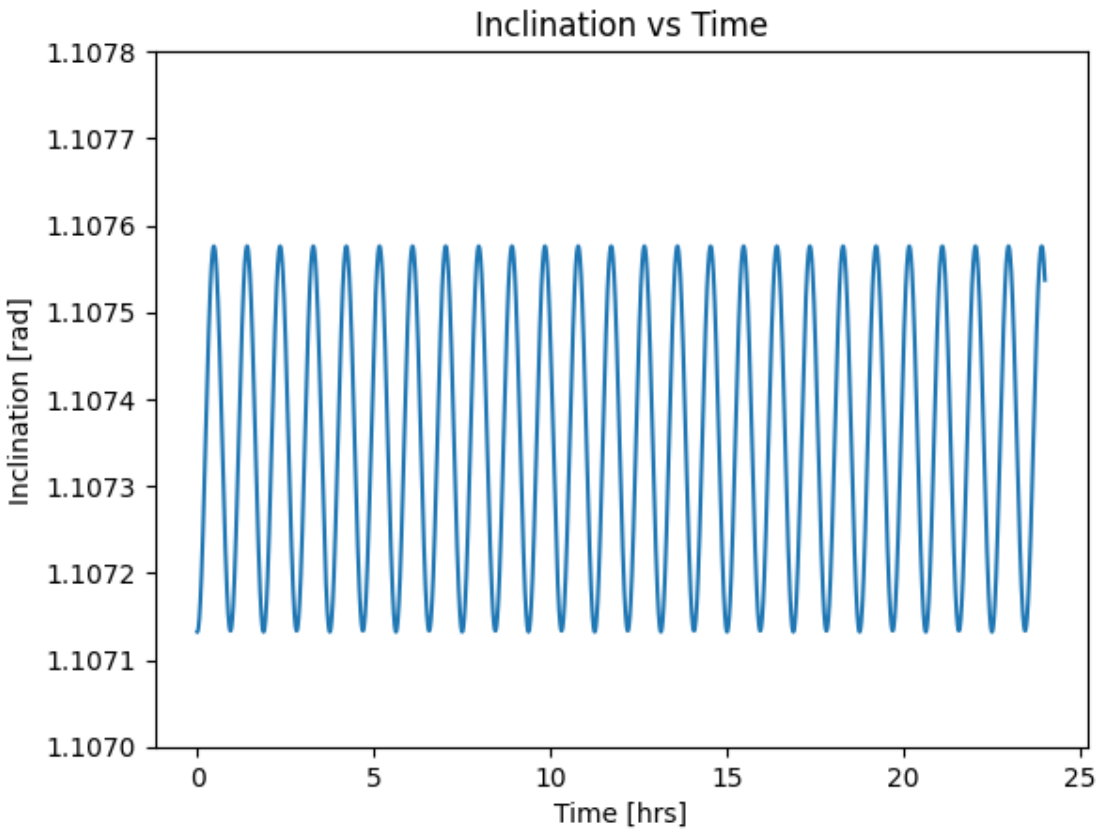
```
plt.ylim(7700, 7740)
```

[ ]: (7700.0, 7740.0)



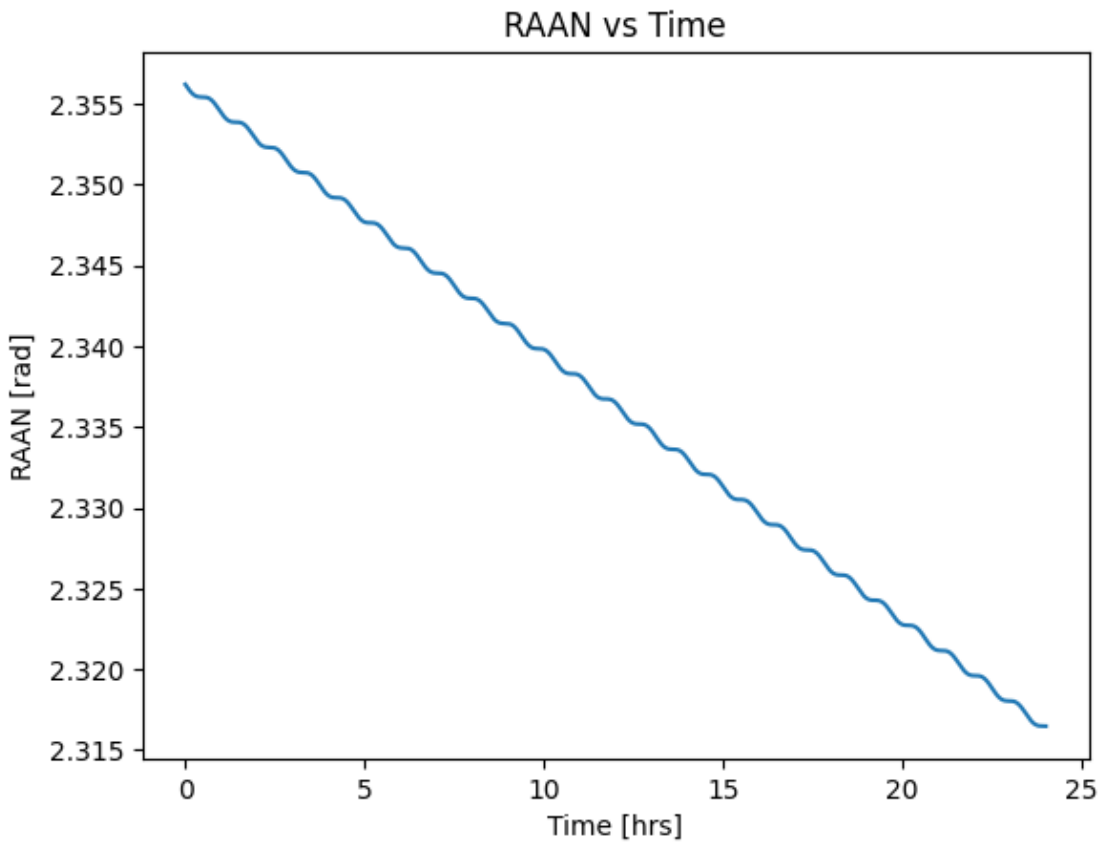
[ ]:
```
#inclination
plt.plot(t/(60*60), orbital_elements_J2[:, 2])
plt.title('Inclination vs Time')
plt.xlabel('Time [hrs]')
plt.ylabel('Inclination [rad]')
plt.ylim(1.107, 1.1078)
```

[ ]: (1.107, 1.1078)



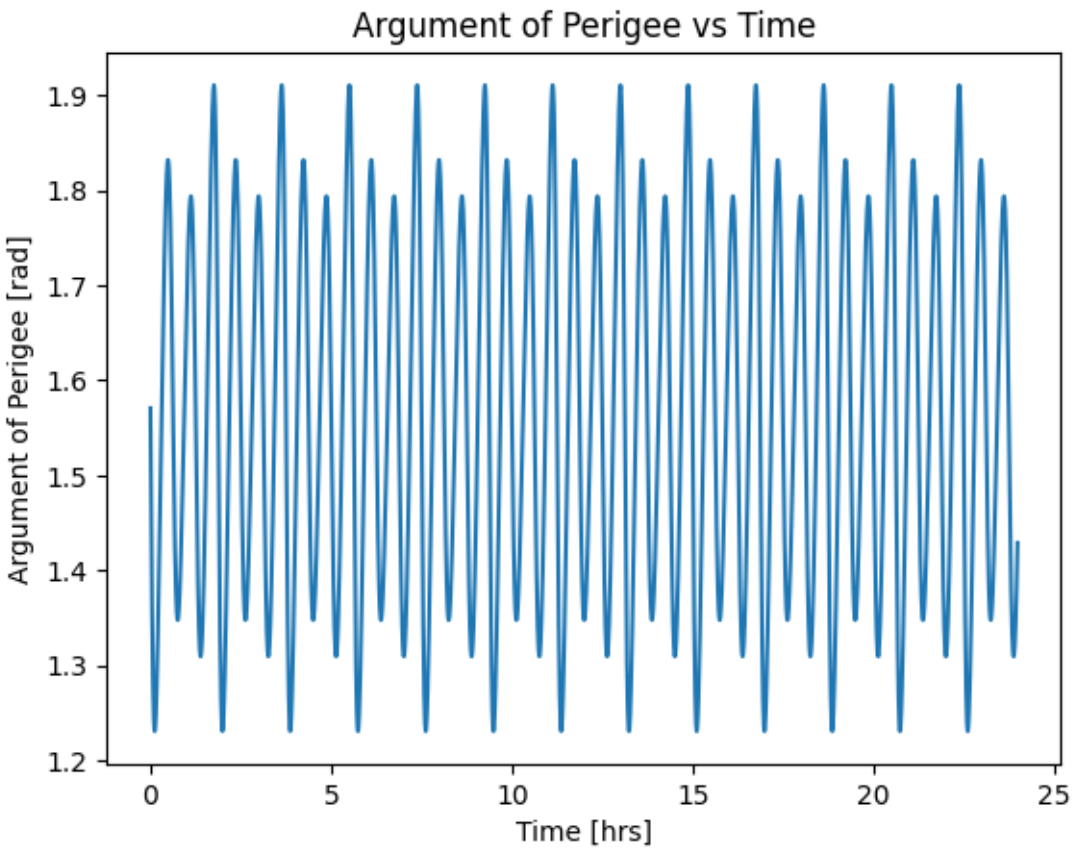
[ ]:
```
#RAAN
plt.plot(t/(60*60), orbital_elements_J2[:, 3])
plt.title('RAAN vs Time')
plt.xlabel('Time [hrs]')
plt.ylabel('RAAN [rad]')
```

[ ]:
```
Text(0, 0.5, 'RAAN [rad]')
```



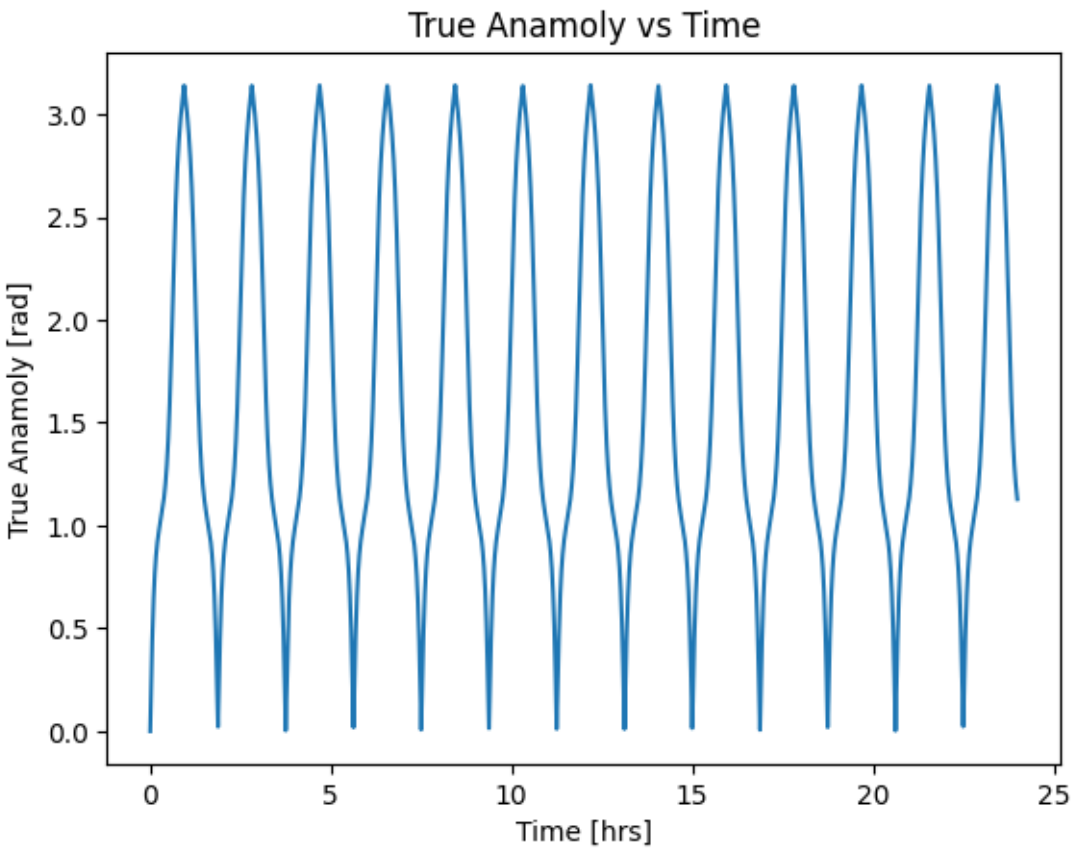
[ ]:
```
#argument of perigee
plt.plot(t/(60*60), orbital_elements_J2[:, 4])
plt.title('Argument of Perigee vs Time')
plt.xlabel('Time [hrs]')
plt.ylabel('Argument of Perigee [rad]')
```

[ ]:
```
Text(0, 0.5, 'Argument of Perigee [rad]')
```



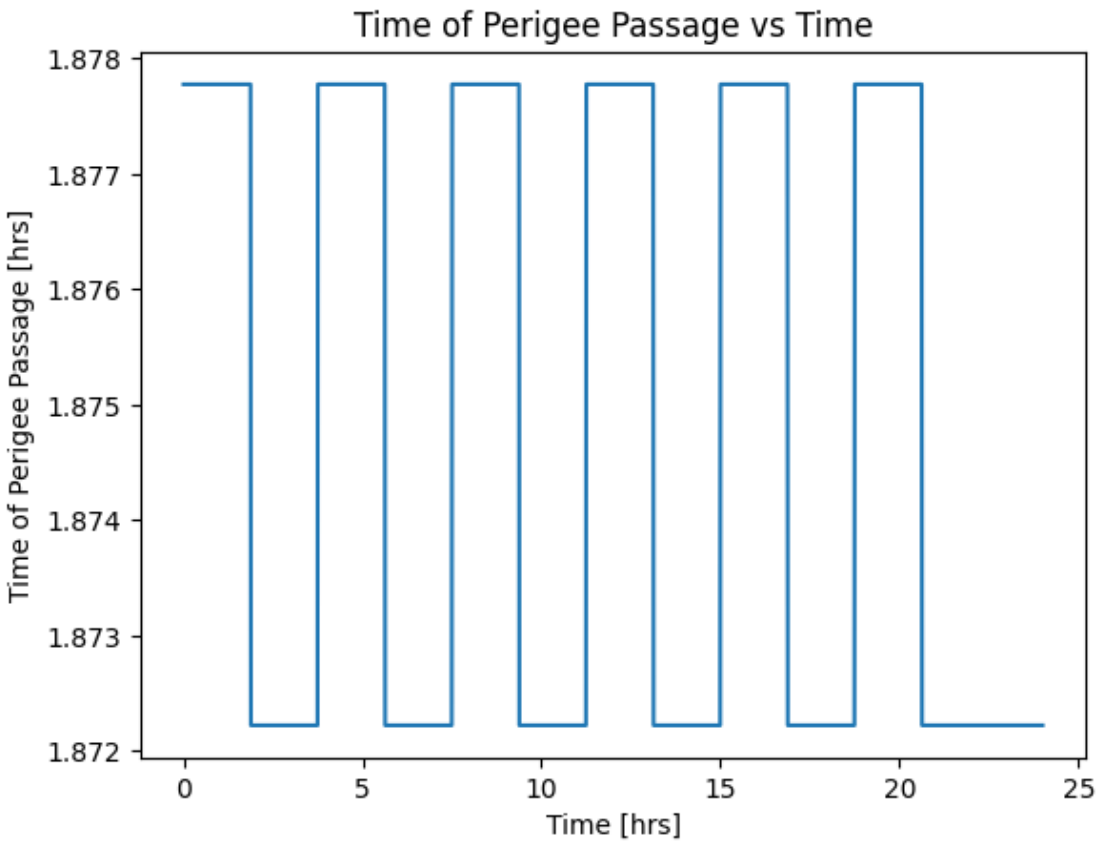
[ ]:
```
#true anamoly
plt.plot(t/(60*60), orbital_elements_J2[:, 5])
plt.title('True Anamoly vs Time')
plt.xlabel('Time [hrs]')
plt.ylabel('True Anamoly [rad]')
```

[ ]:
```
Text(0, 0.5, 'True Anamoly [rad]')
```



[ ]:
```
#time of perigee
# Tp = 2*np.pi*np.sqrt(orbital_elements_J2[:, 1]**3/mu)/(60*60)
Tp_index = np.where(orbital_elements_J2[:, 5] <= 0.0249)[0]
Tp = np.ones(len(t))
# print(Tp_ones[Tp_index[0]:Tp_index[0+1]]*Tp_index[0]*20/60/60 )
for i in range(len(Tp_index)-1):
    Tp[Tp_index[i]:Tp_index[i+1]] = Tp_index[i+1]*20/60/60 - Tp_index[i]*20/60/
↪60
Tp[Tp_index[-1]:len(Tp)] = Tp_index[-1]*20/60/60 - Tp_index[-1-1]*20/60/60
plt.plot(t/(60*60), Tp)
plt.title('Time of Perigee Passage vs Time')
plt.xlabel('Time [hrs]')
plt.ylabel('Time of Perigee Passage [hrs]')
# plt.ylim(1.87, 1.88)
```

[ ]:
```
Text(0, 0.5, 'Time of Perigee Passage [hrs]')
```



Using your insight from the two-body model, what conclusions can you draw about the J2 effect on Keplerian orbital elements?

The J2 effect seems to impact the RAAN the most causing it to decrease over time. It will eventually completing a 360 degree rotation about the reference plane.

Compute the period of the orbit based on the initial state (Eqs (2)-(3)). How does the trend in the plots compare to the period?

[ ]:
```
T0 = 2*np.pi*np.sqrt(orbital_elements_J2[0, 1]**3/mu)/(60*60)
print('Period at T0: ', T0, '[hrs]')
```

```
Period at T0:  1.8722973525740176 [hrs]
```

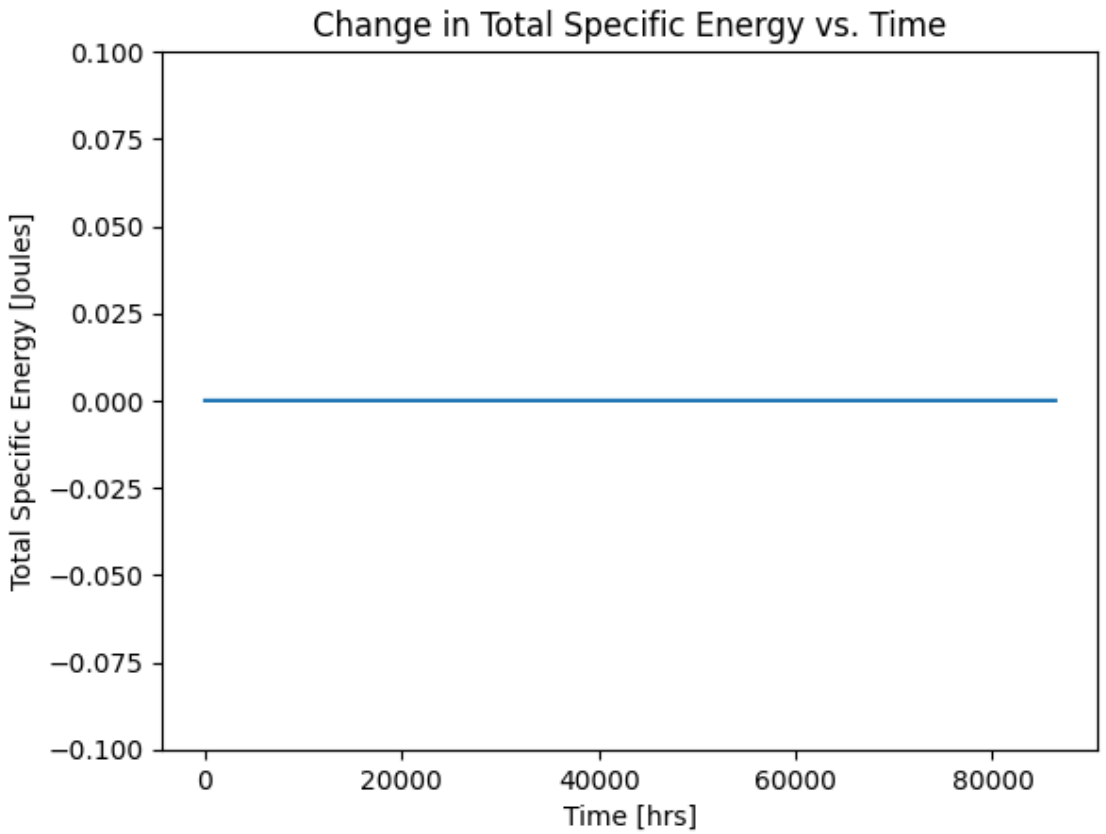
The initial state period underestmates the actual time it takes to make an orbit due to the J2 effects causing periodic changes in the semi-major axis.

1c) Compute the specific energy (energy/mass) and show that it is conserved around the orbit by plotting dE = E(t) − E(t0).

[ ]:
```
#calculate the total specific energy
def E_J2(dU):
    mu = 398600.4 #[km^2/s^3]
```

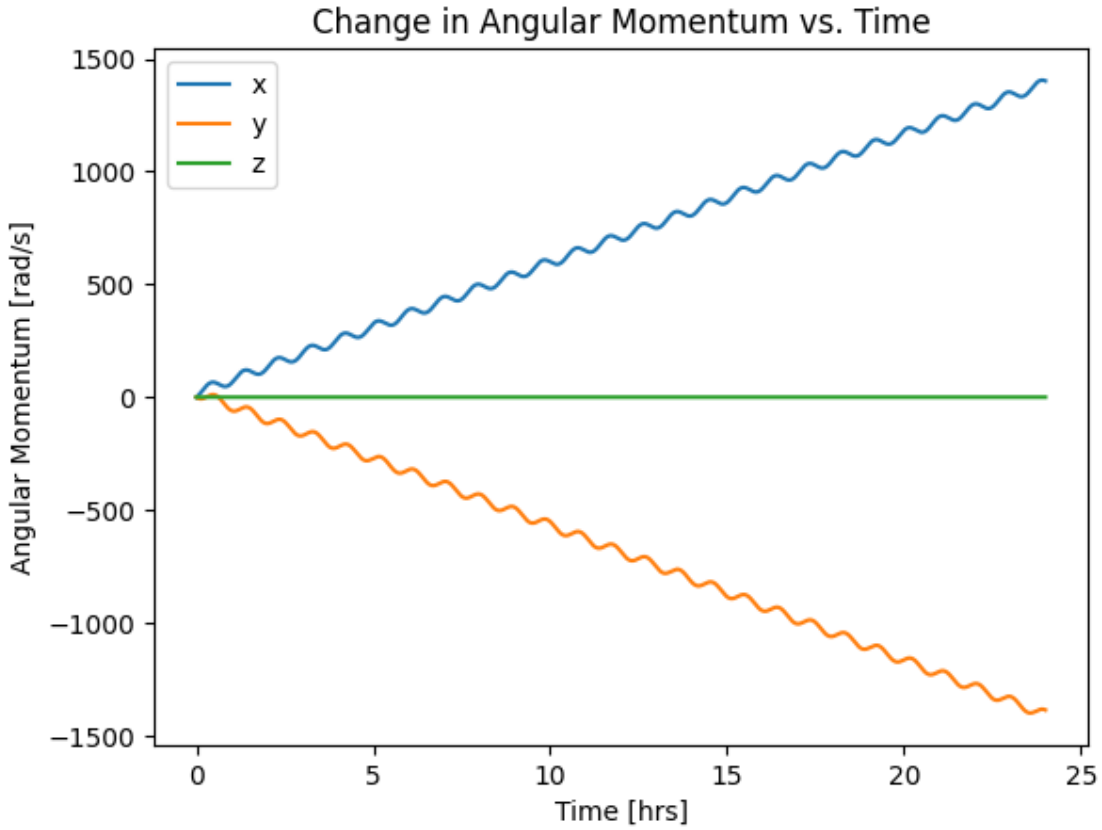
```
    J_2 = 0.00108248
    R_earth = 6378.145 #[km]
    #r_dot from the numeric integration solver
    r_dot = np.linalg.norm([dU[3], dU[4], dU[5]])
    #r from the numeric integration solver
    r = np.linalg.norm([dU[0], dU[1], dU[2]])
    KE = np.dot(r_dot, r_dot)/2
    U = mu/r * (1-J_2*(R_earth/r)**2*(3/2*(dU[2]/r)**2 - 1/2))
    return KE - U


#change in total specific energy
dE = [E_J2(sol_J2.y.T[i]) for i in range(len(t))] - E_J2(sol_J2.y.T[0])
# print(E(sol_J2.y.T[1]))
plt.plot(t, dE)
plt.title('Change in Total Specific Energy vs. Time')
plt.xlabel('Time [hrs]')
plt.ylabel('Total Specific Energy [Joules]')
plt.ylim(-0.1, 0.1)
```

[ ]: (-0.1, 0.1)



[ ]:
```
r_bar = sol_J2.y.T[:, 0:3]
v_bar = sol_J2.y.T[:, 3:6]
h_bar = np.cross(r_bar, v_bar, axis=1)
dh_k = h_bar - h_bar[0]

plt.plot(t/60/60, dh_k)
plt.title('Change in Angular Momentum vs. Time')
plt.xlabel('Time [hrs]')
plt.ylabel('Angular Momentum [rad/s]')
plt.legend(['x', 'y', 'z'])
```

[ ]:
```
<matplotlib.legend.Legend at 0x7f2b02d4d3f0>
```



2) Integrate the equations of motion with the conditions given in Problem 1 that include the Earth point-mass, J2, and now also drag.

[ ]:
```
#equations of motion
def satellite_motion_J2_drag(t, R):
    mu = 398600.4 #[km^2/s^3]
```

```
    J_2 = 0.00108248
    R_earth = 6378.145 #[km]
    C_D = 2.0
    A = 3.6/1000**2 #[km^2]
    m = 1350 #[kg]
    rho_0 = 4E-13 * 1000**3 #[kg/km^3]
    r0 = 7298.145 #[km]
    H = 200.0 #[km]
    theta_dot = 7.29211585530066E-5 #[rad/s]
    r = np.linalg.norm(R[0:3])
    rho_A = rho_0*np.exp(-(r-r0)/H)

    x = R[0]
    y = R[1]
    z = R[2]
    x_dot = R[3]
    y_dot = R[4]
    z_dot = R[5]
    V_A_bar = np.array([x_dot+theta_dot*y, y_dot-theta_dot*x, z_dot])
    V_A = np.sqrt((x_dot + theta_dot*y)**2 +(y_dot-theta_dot*x)**2 + z_dot**2)

    r_ddot = -1/2*C_D*A/m*rho_A*V_A*V_A_bar

    dUdx = -1.0*mu*x*(-J_2*R_earth**2*(1.5*z**2/(x**2 + y**2 + z**2)**1.0 - 0.
↪5)/(x**2 + y**2 + z**2)**1.0 + 1)/(x**2 + y**2 + z**2)**1.5 \
        + mu*(3.0*J_2*R_earth**2*x*z**2/(x**2 + y**2 + z**2)**3.0 + 2.
↪0*J_2*R_earth**2*x*(1.5*z**2/(x**2 + y**2 + z**2)**1.0 - 0.5)/(x**2 \
        + y**2 + z**2)**2.0)/(x**2 + y**2 + z**2)**0.5
    dUdy = -1.0*mu*y*(-J_2*R_earth**2*(1.5*z**2/(x**2 + y**2 + z**2)**1.0 - 0.
↪5)/(x**2 + y**2 + z**2)**1.0 + 1)/(x**2 + y**2 + z**2)**1.5 \
        + mu*(3.0*J_2*R_earth**2*y*z**2/(x**2 + y**2 + z**2)**3.0 + 2.
↪0*J_2*R_earth**2*y*(1.5*z**2/(x**2 + y**2 + z**2)**1.0 - 0.5)/(x**2 \
        + y**2 + z**2)**2.0)/(x**2 + y**2 + z**2)**0.5
    dUdz = -1.0*mu*z*(-J_2*R_earth**2*(1.5*z**2/(x**2 + y**2 + z**2)**1.0 - 0.
↪5)/(x**2 + y**2 + z**2)**1.0 + 1)/(x**2 + y**2 + z**2)**1.5\
        + mu*(2.0*J_2*R_earth**2*z*(1.5*z**2/(x**2 + y**2 + z**2)**1.0 - 0.5)/
↪(x**2 + y**2 + z**2)**2.0 - J_2*R_earth**2*(-3.0*z**3/(x**2 \
            + y**2 + z**2)**2.0 + 3.0*z/(x**2 + y**2 + z**2)**1.0)/(x**2 + y**2␣
↪+ z**2)**1.0)/(x**2 + y**2 + z**2)**0.5
    dydt = np.concatenate((R[3:6], np.array([dUdx+r_ddot[0], dUdy+r_ddot[1],␣
↪dUdz + r_ddot[2]])))
    return dydt
```

[ ]:
```
t = np.arange(0, 86420, 20)
r=np.array([-2436.45, -2436.45, 6891.037]) #[km]
r_dot = np.array([5.088611, -5.088611, 0.0]) #[km/s]

#initial Conditions
y0 = np.concatenate([r, r_dot])

#numeric integration
sol_drag = solve_ivp(satellite_motion_J2_drag, [0, 86420], y0, t_eval=t,␣
 ↪rtol=3E-14, atol=1E-16)
print(sol_drag.y.T[-1])
```

```
[-5.75150585e+03  4.72110760e+03  2.04609503e+03 -7.97610371e-01
 -3.65655308e+00  6.13959523e+00]
```

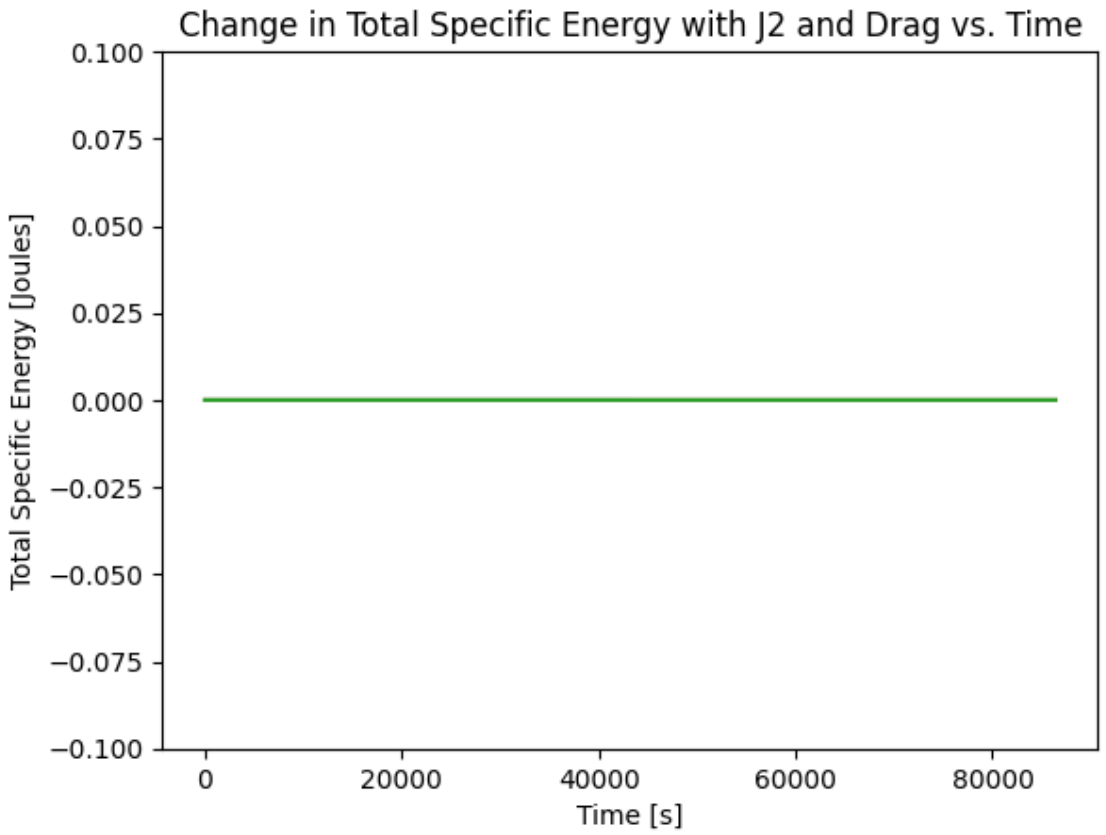
2) a. Compute the specific energy at 20 second intervals and plot dE = E(t) − E(t0) (using Eq. (4)). Include the plot in your write-up. What can you infer from the plot? Is the total energy conserved? Why or why not?

The total energy is conserved even with J2 and drag effects. This is because even as the kinetic energy changes due to the additional effects, the potential energy will change in concert to preserve the conservation of energy.

[ ]:
```
#calculate the total specific energy
def E_J2_drag(R):
    mu = 398600.4 #[km^2/s^3]
    J_2 = 0.00108248
    R_earth = 6378.145 #[km]
    C_D = 2.0
    A = 3.6/1000**2 #[km^2]
    m = 1350 #[kg]
    rho_0 = 4E-13 * 1000**3 #[kg/km^3]
    r0 = 7298.145 #[km]
    H = 200.0 #[km]
    theta_dot = 7.29211585530066E-5 #[rad/s]
    r = np.linalg.norm(R[0:3])
    r_dot = np.linalg.norm([R[3], R[4], R[5]])
    rho_A = rho_0*np.exp(-(r-r0)/H)

    x = R[0]
    y = R[1]
    z = R[2]
    x_dot = R[3]
    y_dot = R[4]
    z_dot = R[5]
    V_A_bar = np.array([x_dot+theta_dot*y, y_dot-theta_dot*x, z_dot])
    V_A = np.sqrt((x_dot + theta_dot*y)**2 +(y_dot-theta_dot*x)**2 + z_dot**2)
```

```
    r_ddot = -1/2*C_D*A/m*rho_A*V_A*V_A_bar
    U = mu/r * (1-J_2*(R_earth/r)**2*(3/2*(R[2]/r)**2 - 1/2))
    KE = np.dot(r_dot, r_dot)/2
    PE = U + r_ddot
    return KE - PE


#change in total specific energy
dE = [E_J2_drag(sol_drag.y.T[i]) for i in range(len(t))] - E_J2_drag(sol_drag.y.
 ↪T[0])
# print(E(sol_drag.y.T[1]))
plt.plot(t, dE)
plt.title('Change in Total Specific Energy with J2 and Drag vs. Time')
plt.xlabel('Time [s]')
plt.ylabel('Total Specific Energy [Joules]')
plt.ylim(-0.1, 0.1)
```

[ ]: (-0.1, 0.1)



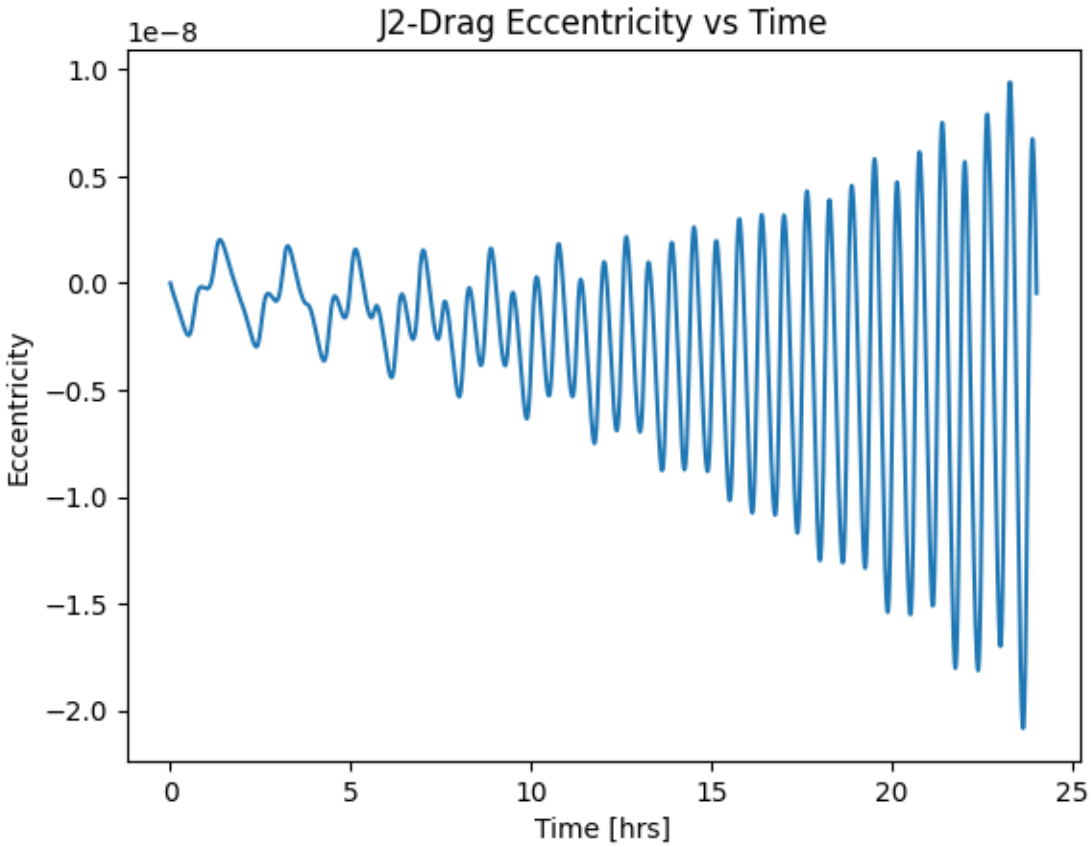
2) b. Compute the same Keplerian orbital elements generated in Problem 1b at 20

second intervals (a, e, i, Ω, , and Tp) and plot the differences in these elements from those computed in Problem 1.

[ ]:
```
mu = 398600.4 #[km^2/s^3]
orbital_elements_drag = np.
 ↪array([state_vectors_to_orbital_elements(state_vector,mu) for state_vector␣
 ↪in sol_drag.y.T])
```

[ ]:
```
#eccentricity
plt.plot(t/(60*60), orbital_elements_drag[:,0]-orbital_elements_J2[:, 0])
plt.title('J2-Drag Eccentricity vs Time')
plt.xlabel('Time [hrs]')
plt.ylabel('Eccentricity')
```

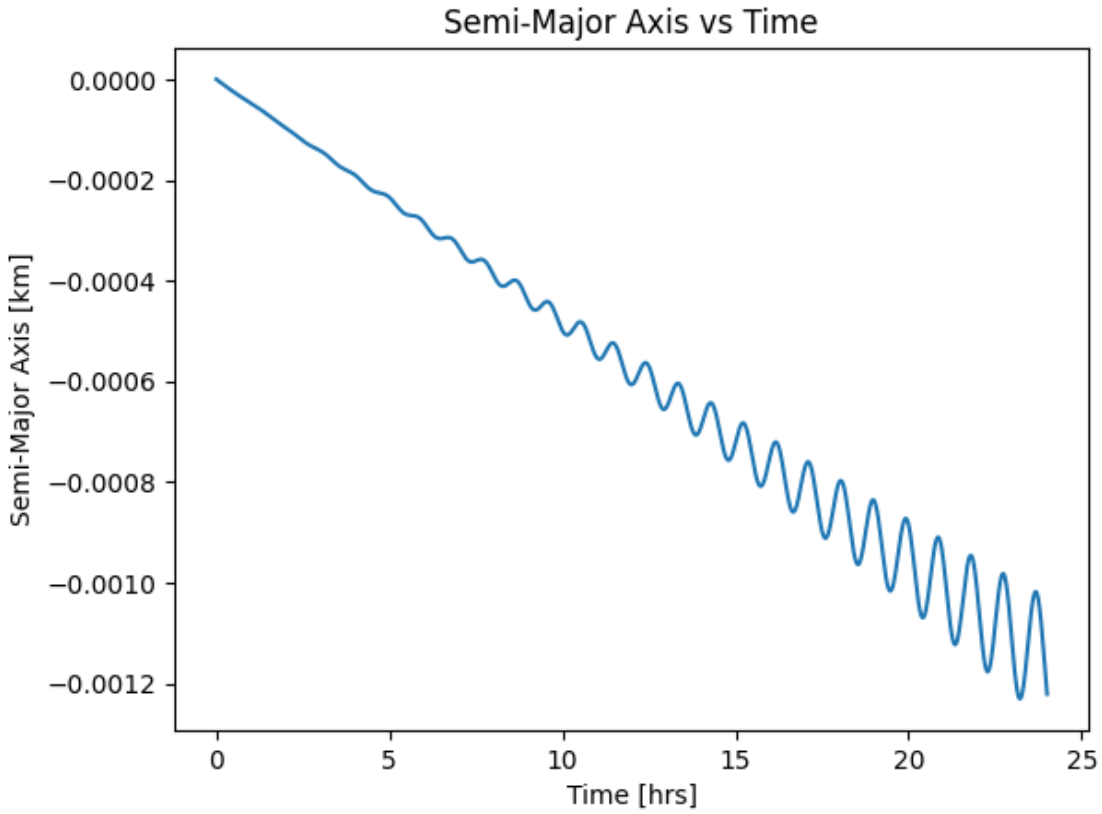
[ ]:
```
Text(0, 0.5, 'Eccentricity')
```



[ ]:
```
#semi-major axis
plt.plot(t/(60*60), orbital_elements_drag[:, 1]-orbital_elements_J2[:, 1])
plt.title('Semi-Major Axis vs Time')
plt.xlabel('Time [hrs]')
```

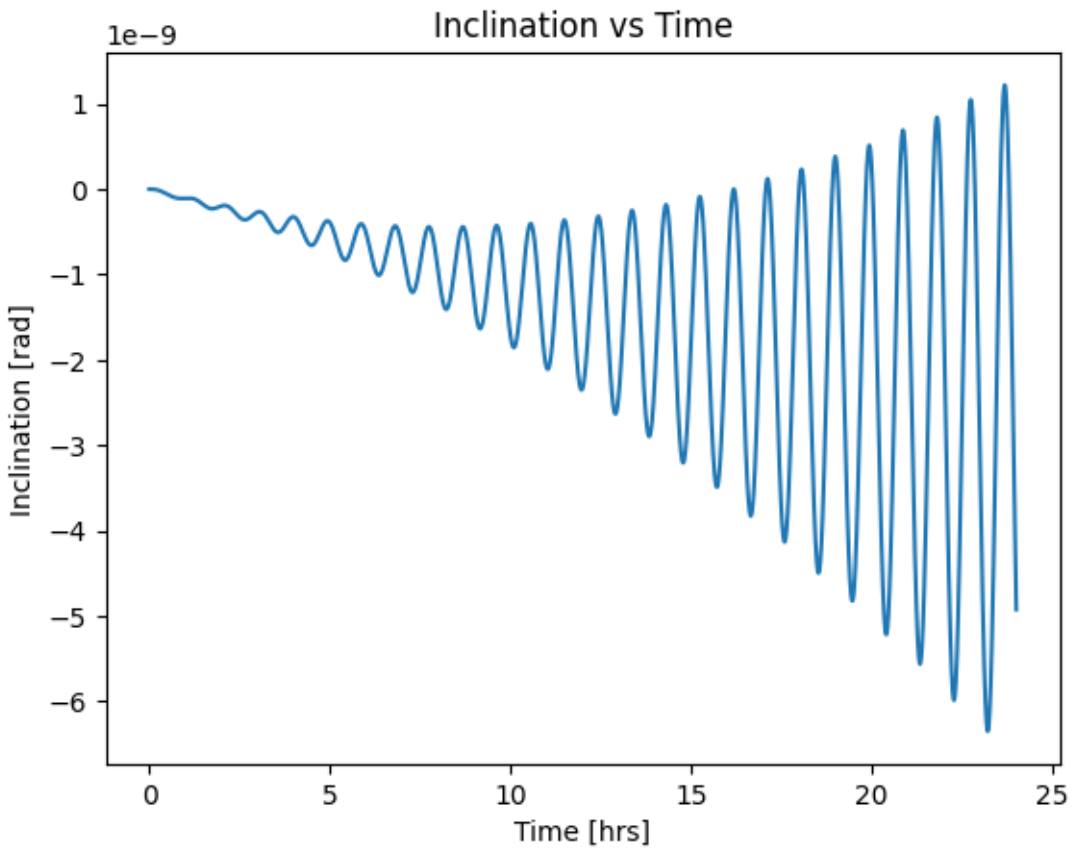
```
plt.ylabel('Semi-Major Axis [km]')
```

[ ]: Text(0, 0.5, 'Semi-Major Axis [km]')



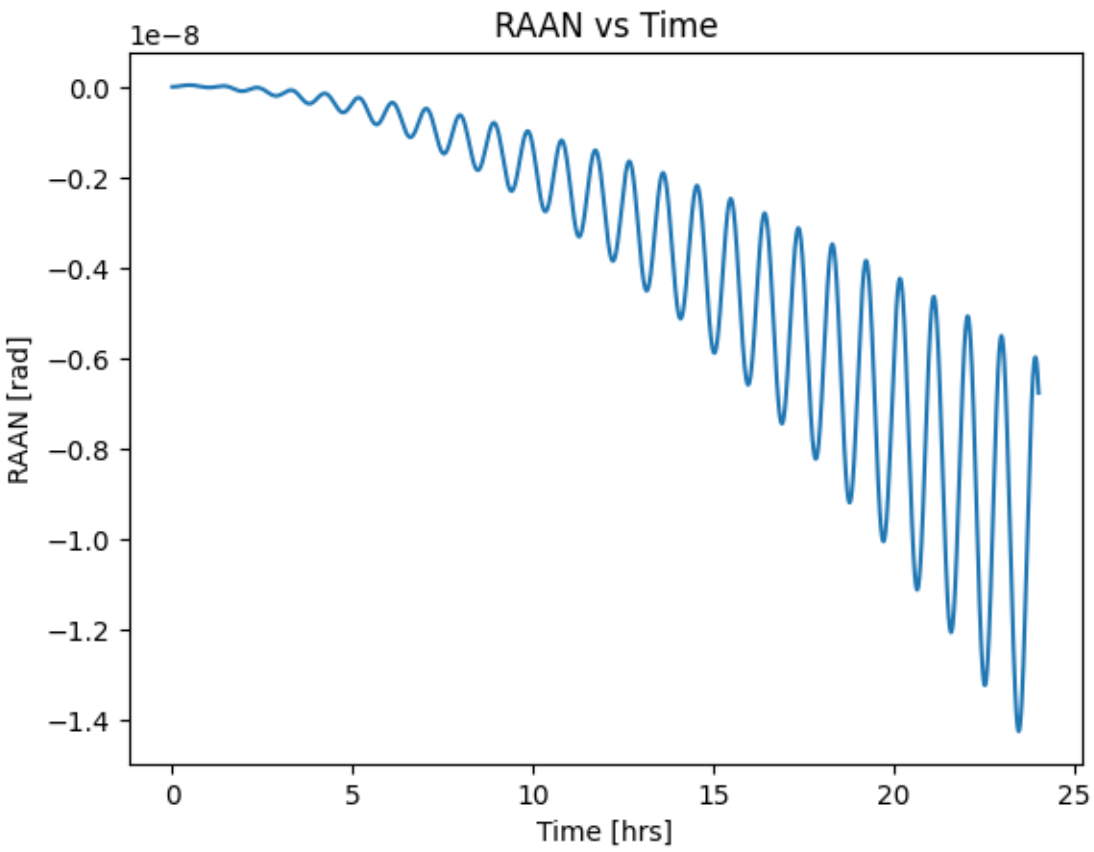
[ ]:
```
#inclination
plt.plot(t/(60*60), orbital_elements_drag[:, 2]-orbital_elements_J2[:, 2])
plt.title('Inclination vs Time')
plt.xlabel('Time [hrs]')
plt.ylabel('Inclination [rad]')
# plt.ylim(1.107, 1.1078)
```

[ ]: Text(0, 0.5, 'Inclination [rad]')



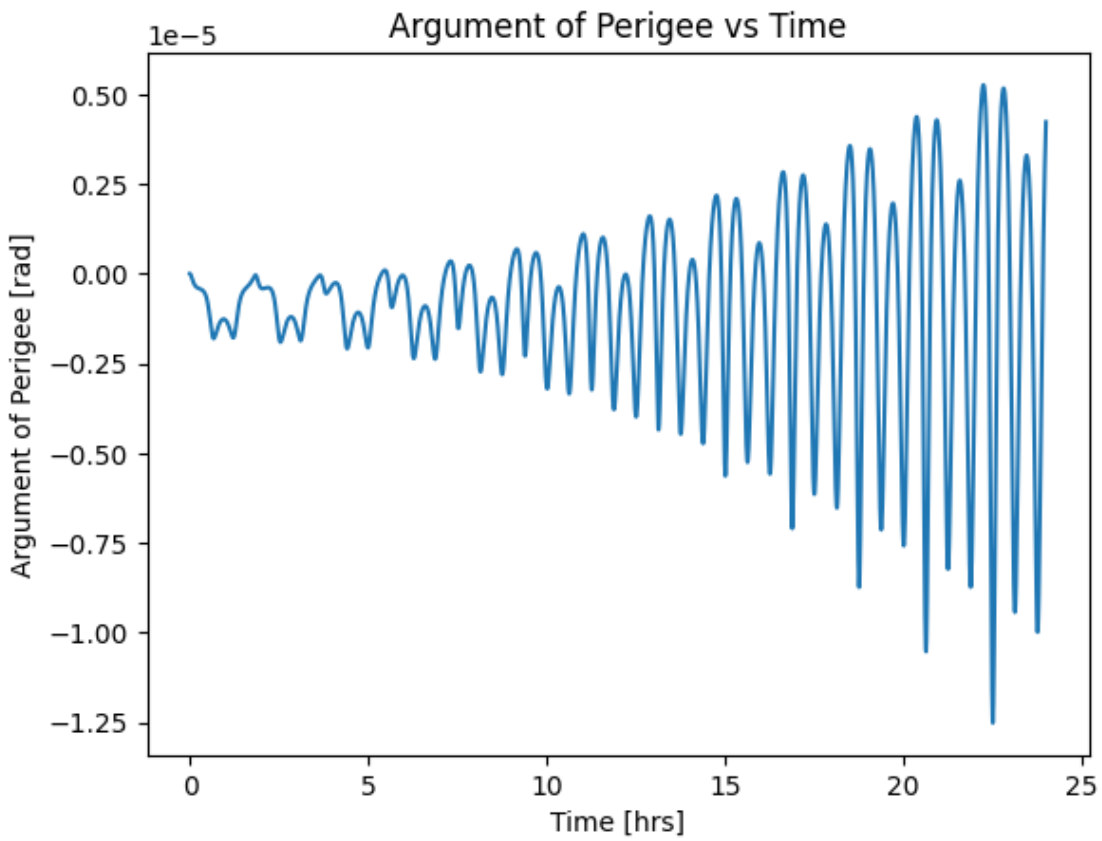
[ ]:
```
#RAAN
plt.plot(t/(60*60), orbital_elements_drag[:, 3]-orbital_elements_J2[:, 3])
plt.title('RAAN vs Time')
plt.xlabel('Time [hrs]')
plt.ylabel('RAAN [rad]')
```

[ ]:
```
Text(0, 0.5, 'RAAN [rad]')
```



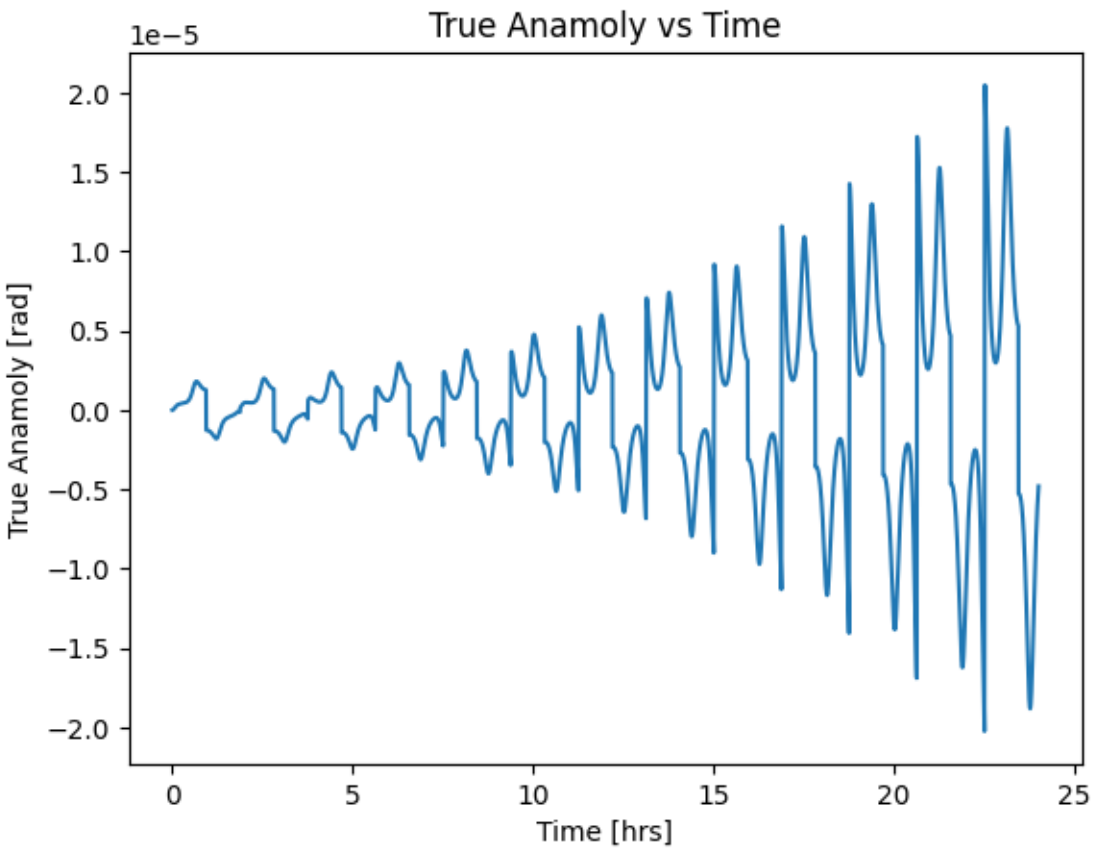
[ ]:
```
#argument of perigee
plt.plot(t/(60*60), orbital_elements_drag[:, 4]-orbital_elements_J2[:, 4])
plt.title('Argument of Perigee vs Time')
plt.xlabel('Time [hrs]')
plt.ylabel('Argument of Perigee [rad]')
```

[ ]:
```
Text(0, 0.5, 'Argument of Perigee [rad]')
```



[ ]:
```
#true anamoly
plt.plot(t/(60*60), orbital_elements_drag[:, 5]-orbital_elements_J2[:, 5])
plt.title('True Anamoly vs Time')
plt.xlabel('Time [hrs]')
plt.ylabel('True Anamoly [rad]')
```

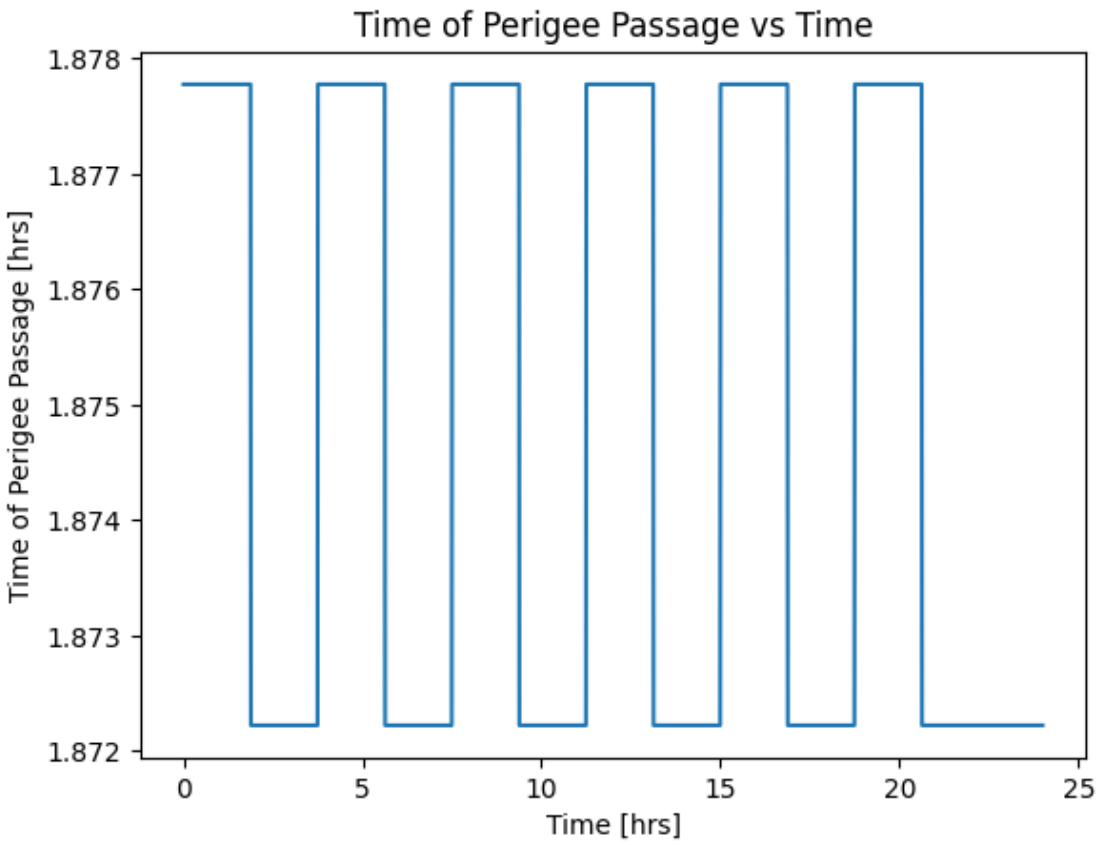
[ ]:
```
Text(0, 0.5, 'True Anamoly [rad]')
```



[ ]:
```
#time of perigee
# Tp = 2*np.pi*np.sqrt(orbital_elements_J2[:, 1]**3/mu)/(60*60)
Tp_index = np.where(orbital_elements_drag[:, 5] <= 2.49E-2)[0]
# Tp_index = np.append(Tp_index, len(t))
print(Tp_index[0])
Tp = np.ones(len(t))
# print(Tp_ones[Tp_index[0]:Tp_index[0+1]]*Tp_index[0]*20/60/60 )
for i in range(len(Tp_index)-1):
    Tp[Tp_index[i]:Tp_index[i+1]] = Tp_index[i+1]*20/60/60 - Tp_index[i]*20/60/
 ↪60
Tp[Tp_index[-1]:len(Tp)] = Tp_index[-1]*20/60/60 - Tp_index[-1-1]*20/60/60
plt.plot(t/(60*60), Tp)
plt.title('Time of Perigee Passage vs Time')
plt.xlabel('Time [hrs]')
plt.ylabel('Time of Perigee Passage [hrs]')
# plt.ylim(1.7, 2)
```

```
0
```

[ ]:
```
Text(0, 0.5, 'Time of Perigee Passage [hrs]')
```



What can you observe in these plots? Which orbital elements are impacted by drag and how are they affected?

Drag seems to have a larger effect on the orbital elements as time goes on. This is particularly apparent in the semi-major axis, true-anomaly, argument of perigee. This makes sense as these orbital elements are related. Drag is causing the semi-major axis to shrink, and shifting the perigee.