Research question: How long would it take to reach Jupiter?

By: Student

CMSE 201-XXX

Background and Motivation:

While Jupiter itself is a gas giant and wouldn't be able to support life without large leaps in technology, there are many moons orbiting Jupiter which may be worth researching. Ganymede, Jupiter's largest moon, is the only satellite in our solar system with a magnetosphere which produces the magnetic poles that we have on earth. It also is believed to have a salt water ocean beneath the surface and has a thin atmosphere of oxygen. While it wouldn't be habitable by means of current technology, it's possible that it could be in the future. Europa, another moon of Jupiter, is also believed to have salt water beneath the surface and so far it seems possible that the planet may be hospitable for some form of alien life. Callisto is another moon similar to Europa in its salty ocean beneath the surface. Io, another moon of Jupiter, is covered in active volcanos, making it a promising target for scientific research.

While there are things that can be studied from afar, to learn as much as possible there will need to be studies done closer, which already NASA has sent voyager 1 and 2 which resulted in a lot of information being discovered about the planets and their moons. For further studies it would be helpful to have an efficient mode of travel, as well as to know how long it would take to reach Jupiter's orbit.

Model of Launch:

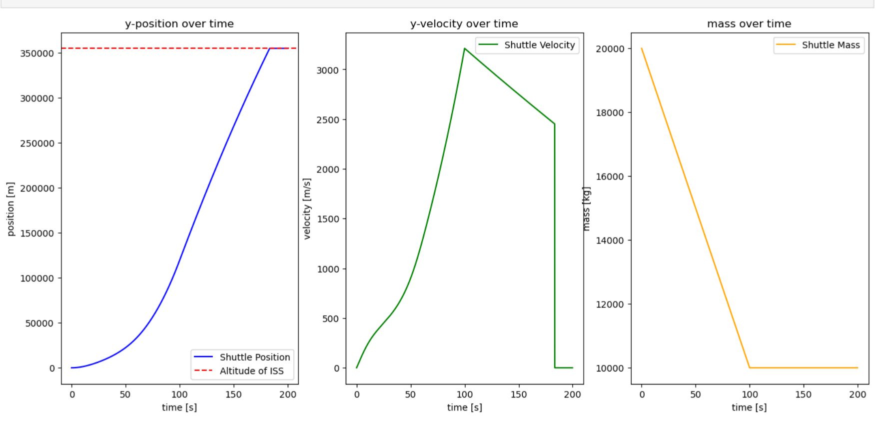
Methodology:

Using Euler's method of integration, the launch is modeled with a changing mass, as well as force of drag while in the atmosphere of the Earth. Euler's method uses integration and taylor series to approximate terms in differential equations. In addition it is important to note that temperature is approximated using a linear model, such that air resistance becomes equal to zero when temperature reaches 0 Kelvin. This represents the end of the atmosphere, where there are fewer particles and temperature approaches absolute zero. Another approximation is the acceleration of gravity being constant, however it would decrease as altitude increases.

In [21]:
```
#Importing packages
from math import * #imports all of the math package without a prefix, an example is math.ceil just becomes cei
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

In [32]:
```
tf = 200 #final time in s
dt = .1 #time step in s
t = np.arange(0,tf,dt) #time array
n = ceil(tf/dt) #number of integration points
m_fuel_max = 1.0e4 #maximum fuel mass
m_empty = 1.0e4 #mass of empty shuttle
dmdt = 100 #burn rate in kg/s
m_added = 0 #added mass
A = 10 #cross sectional area at top of ship in m^2
L = .0065 #temp lapse rate in K/m
T0 = 288 #average temp at sea level in K
R = 8.314 #ideal gas constant
M = .02896 #molar mass of air in g/mol
P0 = 1.01e5 #pressure at sea level in N/m^2

r = np.zeros((n,2)) #position array
v = np.zeros((n,2)) #velocity array
m = np.zeros(n) #mass array
m[0] = m_fuel_max + m_empty + m_added #initial mass
r[0] = np.array([0,0]) #initial position
v[0] = np.array([0,0]) #initial velocity
gs = np.array([0,9.81])  #standard acceleration due to gravity
D = .5 #drag coefficient
Re = 6.37e6 #radius of the Earth
vex = 7000 #exhaust velocity in m/s
g = sqrt(sum(gs*gs)) #magnitude of standard gravity
ISS_height = 3.55e5 #height of ISS in m


flying = True
for i in range(n-1):

    #position doesn't change if not flying, in addition to mass not changing and velocity not changing.
    if flying == False:
        r[i+1] = r[i]
        v[i+1] = 0
        m[i+1] = m[i]

    if flying == True:
        magv = sqrt(sum(v[i]*v[i])) #calculate velocity magnitude
        T = T0 - L*r[i,1] #calculate temperature using linear approximation
        if T > 0: #if T is greater than zero, there is an atmosphere and force of drag
            b = g*M/(R*L) #calculate exponent for pressure
            P = P0*(T/T0)**(b) #calculate new pressure
            rho = P*M/(R*T) #calculate density of air
            Fd = -.5*D*A*rho*v[i]*magv #calculate drag force
        if T <= 0: #if T is below zero, there is no atmosphere and no force of drag
            Fd = 0
        gu = gs/((1+(r[i,1]/Re))**2) #calculate acceleration due to gravity at new altitude
        Fg = -m[i]*gu #calculate force of gravity
        if m[i] > m_empty+m_added: #if there is fuel then there is a thrust force
            Fa = np.array([0,dmdt*vex])
        else: #no fuel leads to no thrust force
            Fa = np.array([0,0])
        Fnet = Fd + Fg + Fa #total force
        a = Fnet/m[i] #total acceleration

        #derivatives
        drdt = v[i]
        dvdt = a

        #Euler approximation
        #same process as solve_ivp(), but I'm more comfortable with explicitly coding Euler's method due to tak
        if m[i] > m_empty:
            m[i+1] = m[i] - dmdt*dt
        if m[i] <= m_empty:
            m[i+1] = m_empty
        v[i+1] = v[i] + dvdt*dt
        r[i+1] = r[i] + drdt*dt

        #set flying to false if ISS is reached to stop motion
        if r[i+1,1] >= ISS_height:
            flying = False
```

In [31]:
```
fig = plt.figure(figsize=(16,7))
plt.subplot(131)
plt.title('y-position over time')
plt.plot(t[:],r[:,1],label='Shuttle Position',color='blue')
plt.ylabel('position [m]')
plt.xlabel('time [s]')
plt.axhline(ISS_height,color='r',label='Altitude of ISS',linestyle='dashed')
plt.legend()

plt.subplot(132)
plt.title('y-velocity over time')
plt.ylabel('velocity [m/s]')
plt.xlabel('time [s]')
plt.plot(t[:],v[:,1],label='Shuttle Velocity',color='green')
plt.legend()

plt.subplot(133)
plt.title('mass over time')
plt.ylabel('mass [kg]')
plt.xlabel('time [s]')
plt.plot(t,m,label='Shuttle Mass',color='orange')
plt.legend()
fig.tight_layout
plt.show()
```



In [14]:
```
magv = sqrt(sum(v[1834]*v[1834])) #magnitude of final velocity
R_Jupiter = 7.87e11# distance that jupiter is from the Earth in m
TimeEstSec = R_Jupiter/magv #calculating if final velocity stayed constant, how long would it take to reach Ju
TimeEstDay = TimeEstSec/(3600*24) #converting to days by converting to hours, and then days
TimeEstYear = TimeEstDay/365 #converting the time estimate in days to time estimate in years
print(round(TimeEstDay,2), 'is the amount of days it would take, and', round(TimeEstYear,2), 'is the amount of
```

```
3715.66 is the amount of days it would take, and 10.18 is the amount of years it would take
```

Results from the trip to orbit:

According to the model, it would take approximately 183.4 seconds for a ship like the one modeled to reach the ISS from the ground. The results from this trip suggest that it would take 10.18 years to reach the orbit of Jupiter if it repeated this process continuously, however this calculation is for a straight line trip including drag force and only the Earth's force of gravity which wouldn't be a large factor compared to Jupiter, especially towards the end of the trip and drag wouldn't be a factor in space. Overall, the majority of the trip will be modeled by the Hohmann Transfer, however the transfer takes advantage of orbits which greatly reduces the time that it takes, as well as only needing two maneuvers using the thrusters resulting in low fuel usage.

Model For Hohmann Transfer:

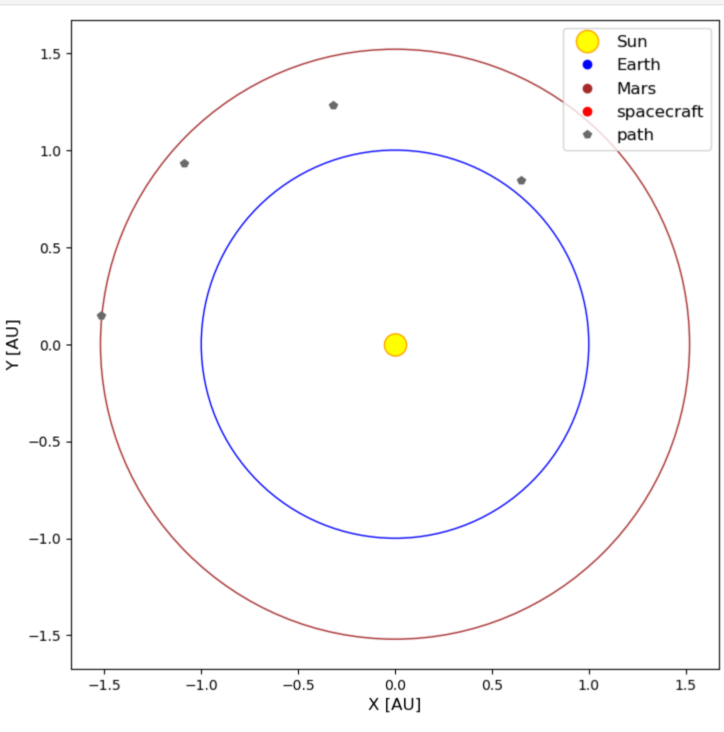
Methodology:

The Hohmann transfer here is modeled by orbital equations that use cartesian coordinates with the sun as the origin to model the circular and eliptical motion. The idea is that the ship has an initial burn which pushes it out of it's circular orbit around the Earth into an eliptical one, and then has a second burn that pushes it into a circular orbit around Jupiter. Due to this low usage of fuel, it is regarded as the most energy efficient way of changing orbits since it mostly takes advantage of the forces of gravity on the ship. The code uses patches in the matplotlib.pyplot subplots figure to create moving objects when the matplotlib animation FuncAnimation is called. In addition the code calculates all the necessary parameters for the functions.

Unaltered Code For Hohmann Transfer From Online Source:

The following code is from (1) in the sources. The only changes made are commented along with certain lines of the code being commented by me to make the code run more smoothly. Note that FFMPEG is needed for the code to run (possibly latex as well), which can be installed by typing `pip install ffmpeg-python` into the Anaconda prompt. This program is used for formatting videos and the reason that it is necessary is for the animation that is produced as a .mp4 file.

In [6]:
```
import numpy as np
import matplotlib as mpl
#mpl.use('pdf')
import matplotlib.pyplot as plt
from matplotlib import animation
mpl.rcParams.update(mpl.rcParamsDefault) #This has been included since switching mpl to use pdf can produce er

#Plotting the Earth and Mars orbits
alpha = 44 #degrees (Angle by which should be ahead by)
Earth = plt.Circle((0,0), radius= 1.0,fill=False,color='blue')
Mars = plt.Circle((0,0), radius= 1.52,fill=False,color='brown')

#Moving Earth, Mars, and Spacecraft
patch_E = plt.Circle((0.0, 0.0),radius=0.04,fill=True,color='blue')
patch_M = plt.Circle((0.0, 0.0),radius=0.03,fill=True,color='brown')
patch_H = plt.Circle((0.0, 0.0),radius=0.01,fill=True,color='red')

def init():
    patch_E.center = (0.0,0.0)
    ax.add_patch(patch_E)
    patch_M.center = (0.0,0.0)
    ax.add_patch(patch_M)
    patch_H.center = (0.0,0.0)
    ax.add_patch(patch_H)
    return patch_E,patch_M,patch_H
def animate(i):
    #Earth
    x_E, y_E = patch_E.center
    x_E = np.cos((2*np.pi/365.2)*i)
    y_E = np.sin((2*np.pi/365.2)*i)
    patch_E.center = (x_E, y_E)
    #Mars
    x_M, y_M = patch_M.center
    x_M = 1.52*np.cos((2*np.pi/686.98)*i+(np.pi*alpha/180.))
    y_M = 1.52*np.sin((2*np.pi/686.98)*i+(np.pi*alpha/180.))
    patch_M.center = (x_M,y_M)
    #Hohmann
    Period = 516.0
    x_H = 1.26*(1. - 0.21**2)/(1. + 0.21*np.cos((2*np.pi/Period)*i))*np.cos((2*np.pi/Period)*i)
    y_H = 1.26*(1. - 0.21**2)/(1. + 0.21*np.cos((2*np.pi/Period)*i))*np.sin((2*np.pi/Period)*i)
    patch_H.center = (x_H,y_H)
    return patch_E,patch_M,patch_H

# Set up formatting for the movie files
#plt.rcParams['savefig.bbox'] = 'tight' # tight garbles the video!!!
Writer = animation.writers['ffmpeg']
writer = Writer(fps=60, metadata=dict(artist='Me'), bitrate=1800)
#writer = Writer(fps=60, bitrate=1800)
#plt.rc('font', family='serif', serif='Times')
#plt.rc('text', usetex=True)
#plt.rc('xtick', labelsize=8)
#plt.rc('ytick', labelsize=8)
#plt.rc('axes', labelsize=8)

# Set up path, to guide eye
Period = 516.
x_H_B = 1.26*(1. - 0.21**2)/(1. + 0.21*np.cos((2*np.pi/Period*75)))*np.cos((2*np.pi/Period*75))
y_H_B = 1.26*(1. - 0.21**2)/(1. + 0.21*np.cos((2*np.pi/Period*75)))*np.sin((2*np.pi/Period*75))
x_H_C = 1.26*(1. - 0.21**2)/(1. + 0.21*np.cos((2*np.pi/Period*150)))*np.cos((2*np.pi/Period*150))
y_H_C = 1.26*(1. - 0.21**2)/(1. + 0.21*np.cos((2*np.pi/Period*150)))*np.sin((2*np.pi/Period*150))
x_H_D = 1.26*(1. - 0.21**2)/(1. + 0.21*np.cos((2*np.pi/Period*200)))*np.cos((2*np.pi/Period*200))
y_H_D = 1.26*(1. - 0.21**2)/(1. + 0.21*np.cos((2*np.pi/Period*200)))*np.sin((2*np.pi/Period*200))
x_H_M = 1.26*(1. - 0.21**2)/(1. + 0.21*np.cos((2*np.pi/Period*250)))*np.cos((2*np.pi/Period*250))
y_H_M = 1.26*(1. - 0.21**2)/(1. + 0.21*np.cos((2*np.pi/Period*250)))*np.sin((2*np.pi/Period*250))

fig, ax = plt.subplots(figsize=(10,8))
fig.subplots_adjust(left=.15, bottom=.16, right=.99, top=.97)
ax.plot(0,0,color='orange',marker='o',linestyle='',markersize=16,markerfacecolor='yellow',label='Sun')
ax.plot([],[],color='blue',linestyle='',marker='o',label='Earth')
ax.plot([],[],color='brown',linestyle='',marker='o',label='Mars')
ax.plot([],[],color='red',linestyle='',marker='o',label='spacecraft')
ax.plot(x_H_B,y_H_B,color='dimgray',marker ='p',markerfacecolor='dimgray',linestyle='',label='path')
ax.plot(x_H_C,y_H_C,color='dimgray',marker ='p',markerfacecolor='dimgray')
ax.plot(x_H_D,y_H_D,color='dimgray',marker ='p',markerfacecolor='dimgray')
ax.plot(x_H_M,y_H_M,color='dimgray',marker ='p',markerfacecolor='dimgray')
ax.add_patch(Earth)
ax.add_patch(Mars)
ax.set_xlabel('X [AU]',fontsize=12)
ax.set_ylabel('Y [AU]',fontsize=12)
ax.legend(loc='best',fontsize=12)
anim = animation.FuncAnimation(fig, animate,init_func=init,frames=260,interval=40,blit=True)
plt.axis('scaled') #Scale the plot in real time
#plt.savefig('Hohmann.pdf')
#anim.save('Hohmann.mp4', writer=writer)
plt.show()
```



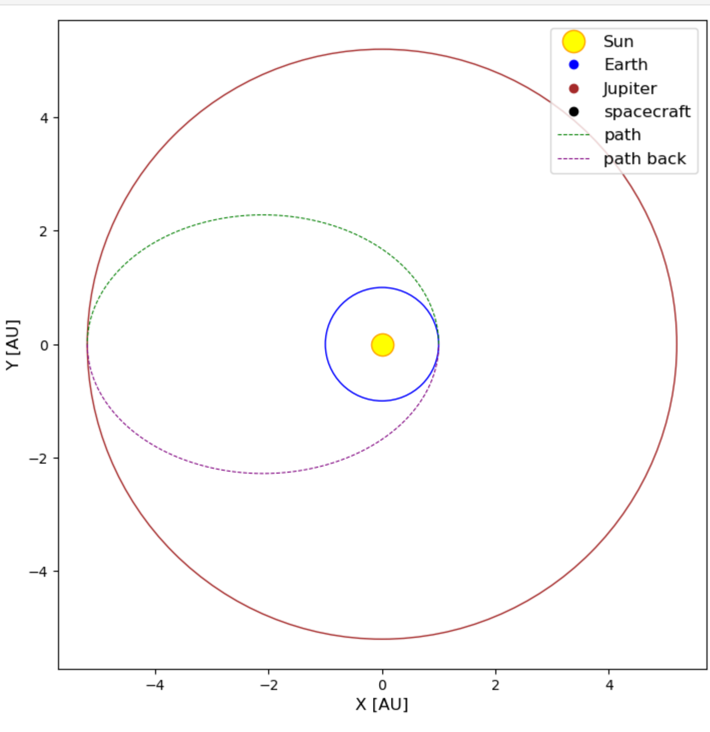
My Code:

The following code has been altered by me from the above source, to model a Hohmann transfer to Jupiter. I have altered equations along with added explanations for what certain parts of the code are doing, in addition to making adjustments to the figure. The green and purple dashed lines on the figure represent the trip to Jupiter and trip back to Earth respectively. It's important to note that the animation will only play if is saved as a .mp4 and then loaded seperately, which requires the line doing this to be uncommented. It is also important to note that it will take some time to save, which is why there is a second .mp4 file included with the submission of this assignment. The plot can also be saved as a .pdf file by uncommenting the `mpl.use('pdf')` and `plt.savefig()` lines, however the `mpl.rcParams` line should be commented when this is done.

In [7]:
```
import numpy as np
import matplotlib as mpl
#mpl.use('pdf')
import matplotlib.pyplot as plt
from matplotlib import animation
mpl.rcParams.update(mpl.rcParamsDefault)
```

In [8]:
```
rev_J = 4332.71 #days per revolution on Jupiter
rev_E = 365.25 #days per revolution on Earth
rad_J = 5.2 #radius of Jupiter's Orbit
rad_E = 1 #radius of Earth's Orbit
rad_H = (rad_E+rad_J)/2 #semi-major axis of Hohmann ellipse
period = sqrt((rad_H)**3)*365 #period of transfer
alpha = 180 - (360/rev_J)*(period/2) #angle of Jupiter when craft is launched
beta = 1-(rad_E/rad_H) #parameter for Hohmann path
```

In [9]:
```
#plotting the Earth and Jupiter orbits
Earth = plt.Circle((0,0), radius= rad_E,fill=False,color='blue')
Jupiter = plt.Circle((0,0), radius= rad_J,fill=False,color='brown')

#moving Earth, Jupiter, and Spacecraft as patches
patch_E = plt.Circle((0.0, 0.0),radius=0.04,fill=True,color='blue')
patch_J = plt.Circle((0.0, 0.0),radius=0.06,fill=True,color='brown')
patch_H = plt.Circle((0.0, 0.0),radius=0.01,fill=True,color='black')

#functions for animation.FuncAnimation
def init():
    patch_E.center = (0.0,0.0)
    ax.add_patch(patch_E)
    patch_J.center = (0.0,0.0)
    ax.add_patch(patch_J)
    patch_H.center = (0.0,0.0)
    ax.add_patch(patch_H)
    return patch_E,patch_J,patch_H
def update(i):
    #Earth
    x_E, y_E = patch_E.center
    x_E = np.cos((2*np.pi/rev_E)*i)
    y_E = np.sin((2*np.pi/rev_E)*i)
    patch_E.center = (x_E, y_E)
    #Jupiter
    x_J, y_J = patch_J.center
    x_J = rad_J*np.cos((2*np.pi/rev_J)*i+(np.pi*alpha/180.))
    y_J = rad_J*np.sin((2*np.pi/rev_J)*i+(np.pi*alpha/180.))
    patch_J.center = (x_J,y_J)
    #Hohmann
    x_H = rad_H*(1 - beta**2)/(1 + beta*np.cos((2*np.pi/period)*i))*np.cos((2*np.pi/period)*i)
    y_H = rad_H*(1 - beta**2)/(1 + beta*np.cos((2*np.pi/period)*i))*np.sin((2*np.pi/period)*i)
    patch_H.center = (x_H,y_H)
    return patch_E,patch_J,patch_H


#set up formatting for the movie files
Writer = animation.writers['ffmpeg']
writer = Writer(fps=60, metadata=dict(artist='Me'), bitrate=1800)

#set up path
t = np.arange(0,period/2,20)
n = ceil(period/2)
x_H = np.zeros(n)
y_H = np.zeros(n)
for i in range(n):
    x_H[i] = rad_H*(1 - beta**2)/(1 + beta*np.cos((2*np.pi/period*i)))*np.cos((2*np.pi/period*i))
    y_H[i] = rad_H*(1 - beta**2)/(1 + beta*np.cos((2*np.pi/period*i)))*np.sin((2*np.pi/period*i))

#set up return path
t2 = np.arange(period/2,period,20)
n2 = ceil(period/2)
x_H2 = np.zeros(n2)
y_H2 = np.zeros(n2)
for i in range(n2):
    j = i + period/2
    x_H2[i] = rad_H*(1 - beta**2)/(1 + beta*np.cos((2*np.pi/period*j)))*np.cos((2*np.pi/period*j))
    y_H2[i] = rad_H*(1 - beta**2)/(1 + beta*np.cos((2*np.pi/period*j)))*np.sin((2*np.pi/period*j))

#create figure
fig, ax = plt.subplots(figsize=(10,8))
fig.subplots_adjust(left=.15, bottom=.16, right=.99, top=.97)

#plot sun
ax.plot(0,0,color='orange',marker='o',linestyle='',markersize=16,markerfacecolor='yellow',label='Sun')

#these lines plot the patches in the order which they were defined, as well as provide labels to include in the
ax.plot([],[],color='blue',linestyle='',marker='o',label='Earth')
ax.plot([],[],color='brown',linestyle='',marker='o',label='Jupiter')
ax.plot([],[],color='black',linestyle='',marker='o',label='spacecraft')

#plots the path and return path
ax.plot(x_H,y_H,color='green',markerfacecolor='orange',linestyle='dashed',label='path',linewidth=.75)
ax.plot(x_H2,y_H2,color='purple',markerfacecolor='orange',linestyle='dashed',label='path back',linewidth=.75)

#adds Earth and Jupiter
ax.add_patch(Earth)
ax.add_patch(Jupiter)

#labels and legend
ax.set_xlabel('X [AU]',fontsize=12)
ax.set_ylabel('Y [AU]',fontsize=12)
ax.legend(loc='best',fontsize=12)

#function to update ship and animate the figure
anim = animation.FuncAnimation(fig,update,init_func=init,frames=1000,interval=500,blit=True)

#Scale the plot in real time to keep circles circular
plt.axis('scaled')

#saves figures in folder that this file is located in
#plt.savefig('Hohmann.pdf') #mpl.use('pdf') must be uncommented for this to run, make sure to restart kernel a
#anim.save('Hohmann.mp4', writer=writer)
#anim.save('Hohmann.gif', writer=writer)

plt.show()
```



In [19]:
```
print('the period of the full elliptical orbit is'
      ,round(period,2),'in days, meaning that for the transfer to be completed it would take'
      , round(period/2,2),'days. The orbit of Jupiter would need to start at a', round(alpha,2),
      'degree angle with respect to the Earth, and the semi-major axis of   the transfer ellipse would be', rad
```

```
the period of the full elliptical orbit is 1992.21 in days, meaning that for the transfer to be completed it wo
uld take 996.11 days. The orbit of Jupiter would need to start at a 97.23 degree angle with respect to the Eart
h, and the semi-major axis of   the transfer ellipse would be 3.1 AU.
```

Results:

The results show that it would take almost 3 years years for a Hohmann Transfer to Jupiter to be completed. In addition, since quite a large supply of food would be necessary to sustain the crew if it was a manned operation, more fuel would be needed to reach the velocity desired, requiring even more mass. It is due to these reasons (along with the time commitment and cost commitment) that interstellar travel is not currently feasible aside from probes. However should the required advancements in technology be made this orbital transfer process could become a common occurence.

Synthesis and Discussion:

While these models may be able to provide predictions about these physical systems, there are various simplifications that have been made to make the models simpler. In the launch, I have assumed a that there is a linear relationship between altitude and temperature, however this relationship indicates that the temperature will eventually fall below absolute zero, which is not possible. Another assumption made is that the error with an Euler approximation is reasonable, however without analytical solutions this could be false, so comparing other methods of integration approximations such as the Euler-Cromer, Velocity-Verlet, and RK4 methods would be worthwhile. In addition it would be interesting to attempt to model a double Hohmann transfer from Earth to Mars and then Mars to Jupiter and compare the length of time it would take. It is also assumed that Mars would not affect the Hohmann transfer, which is not true, since the gravity of Mars will be able to affect the orbital of the ship at all times. It also is assumed that the transfer is performed at a time at which Mars would not collide with the ship. Mars has an orbital period of 687 days, so depending on it's initial position, the ship may be closer to Mars than Jupiter for 2 periods of time. Another complication is that the location of Mars could prevent certain opportunities for a Hohmann transfer to jupiter from occurring, leading to narrow windows for these plans to be made and carried out. Despite these simplifications made, I believe this is a good model for initial predictions to serve as a baseline to represent this physical system.

Sources:

1. Heita, P. (2021, December 28). Simulate interplanetary space travel in Python. Medium. Retrieved April 27, 2022, from https://python.plainenglish.io/simulating-interplanetary-space-travel-in-python-95116b14b6a9
2. Zimmermann, K. (2016, June 15). Ganymede: Facts about Jupiter's largest moon. Space.com. Retrieved April 27, 2022, from https://www.space.com/16440-ganymede-facts-about-jupiters-largest-moon.html
3. NASA. (2021, November 4). Europa. NASA. Retrieved April 27, 2022, from https://solarsystem.nasa.gov/moons/jupiter-moons/europa/in-depth/
4. NASA. (2021, July 19). Callisto. NASA. Retrieved April 27, 2022, from https://solarsystem.nasa.gov/moons/jupiter-moons/callisto/in-depth/
5. NASA. (2021, July 19). Io. NASA. Retrieved April 27, 2022, from https://solarsystem.nasa.gov/moons/jupiter-moons/io/in-depth/
6. NASA. (2021, November 10). Ganymede. NASA. Retrieved April 27, 2022, from https://solarsystem.nasa.gov/moons/jupiter-moons/ganymede/in-depth/
7. NASA. (2021, February 4). Voyager 1. NASA. Retrieved April 27, 2022, from https://solarsystem.nasa.gov/missions/voyager-1/in-depth/
8. NASA. (2021, February 4). Voyager 2. NASA. Retrieved April 27, 2022, from https://solarsystem.nasa.gov/missions/voyager-2/in-depth/
9. Modeling a rocket launch with gravity and Air Resistance. ComPADRE.org. (n.d.). Retrieved April 27, 2022, from https://www.compadre.org/osp/items/detail.cfm?ID=11294
10. Martian Colonist. (n.d.). Getting to Mars: The hohmann transfer - youtube. Retrieved April 28, 2022, from https://www.youtube.com/watch?v=LzsMjEMDpD4
11. VanderPlas, J. (n.d.). Matplotlib animation tutorial. Matplotlib Animation Tutorial | Pythonic Perambulations. Retrieved April 27, 2022, from https://jakevdp.github.io/blog/2012/08/18/matplotlib-animation-tutorial/