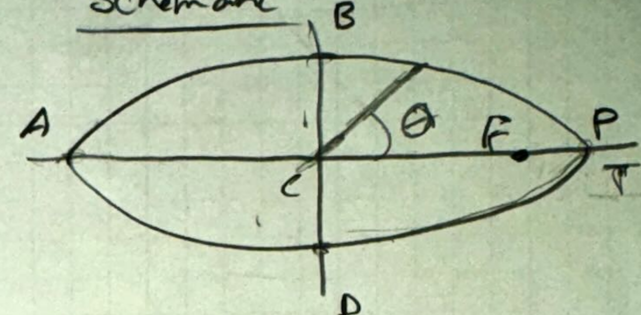
AA 310 | Luke Verlangieri

1) given prograde orbit

Schematic



Find

Time to fly between points on the orbit in terms of T and e

Properties

$$\frac{2\pi \cdot t}{T} = E - e\sin E \qquad E = 2\tan^{-1}\left[\sqrt{\frac{1-e}{1+e}}\tan\left(\frac{\theta}{2}\right)\right]$$

$t(\pi) = \frac{T}{2}$ → Half the time of flight will be half the period

Analysis

The points of the ellipse are at points $\theta = 0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}$

Here we can precompute the E values for given angles

$$E(0) = 2\tan^{-1}\left[\sqrt{\frac{1-e}{1+e}}(0)\right] = 0 \quad E\left(\frac{\pi}{2}\right) = 2\tan^{-1}\left[\sqrt{\frac{1-e}{1+e}}(1)\right] = 2\tan^{-1}\left[\sqrt{\frac{1-e}{1+e}}\right]$$

$$E(\pi) = 2\tan^{-1}\left[\sqrt{\frac{1-e}{1+e}}\tan\left(\frac{\pi}{2}\right)\right] = \text{undefined} \quad E\left(\frac{3\pi}{2}\right) = 2\tan^{-1}\left[\sqrt{\frac{1-e}{1+e}}(-1)\right] = -2\tan^{-1}\left[\sqrt{\frac{1-e}{1+e}}\right]$$

Knowing this will make it easier to solve the following problems

a) $P \to B \quad \theta = \frac{\pi}{2} \to t\left(\frac{\pi}{2}\right)$

$$t = \left(\frac{T}{2\pi}\right)(E - e\sin E) = \frac{T}{2\pi}\left(2\tan^{-1}\left[\sqrt{\frac{1-e}{1+e}}\right] - e\sin\left(2\tan^{-1}\left[\sqrt{\frac{1-e}{1+e}}\right]\right)\right)$$

b) $B \to A$ Know $t(P \to B)$ and $t(P \to A) = t(\pi)$

$$t_{B\to A} = t(\pi) - t_{P\to B} = \frac{T}{2}\left(1 - \frac{1}{\pi}\left(2\tan^{-1}\left[\sqrt{\frac{1-e}{1+e}}\right] - e\sin\left(2\tan^{-1}\left[\sqrt{\frac{1-e}{1+e}}\right]\right)\right)\right)$$

c) $A \to D$

$$t_{A\to D} = t_{P\to D} - t_{P\to A} = \frac{T}{2}\left(\frac{1}{\pi}\left(-2\tan^{-1}\left(\sqrt{\frac{1-e}{1+e}}\right) + e\sin\left(2\tan^{-1}\left(\sqrt{\frac{1-e}{1+e}}\right)\right)\right) - 1\right)$$

TOPS

AA 310

d) D → P

$$T_{D\to P} = T_{P\to A} - T_{P\to D} = T\left(1 - \frac{1}{2\pi}\left(-2\tan^{-1}\left(\sqrt{\frac{1-e}{1+e}}\right) + e\sin\left(2\tan^{-1}\left(\sqrt{\frac{1-e}{1+e}}\right)\right)\right)\right)$$

e) P → A $\theta = \pi$

$$t_{P\to A} = \frac{T}{2}$$

f) A → P $\theta = \pi$

$$t_{A\to P} = t_{P\to A} = \frac{T}{2}$$

g) P → D $\theta = \frac{3\pi}{2}$

$$t_{P\to D} = \frac{T}{2\pi}\left(-2\tan\left(\sqrt{\frac{1-e}{1+e}}\right) + e\sin\left(2\tan^{-1}\left[\sqrt{\frac{1-e}{1+e}}\right]\right)\right)$$

h) B → P

$$t_{B\to P} = t_{B\to B} - t_{P\to B} = T\left(1 - \frac{1}{2\pi}\left(2\tan^{-1}\left(\sqrt{\frac{1-e}{1+e}}\right) - e\sin\left(2\tan^{-1}\left[\sqrt{\frac{1-e}{1+e}}\right]\right)\right)\right)$$

i) B → D

$$t_{B\to D} = t_{B\to A} + t_{A\to D}$$

$$\to \frac{T}{2}\left(\cancel{1} - \frac{1}{\pi}\left(2\tan^{-1}\left[\sqrt{\frac{1-e}{1+e}}\right] - e\sin\left(2\tan^{-1}\left[\sqrt{\frac{1-e}{1+e}}\right]\right)\right)\right)$$

$$- \frac{T}{2}\left(\cancel{1} + \frac{1}{\pi}\left(2\tan^{-1}\left[\sqrt{\frac{1-e}{1+e}}\right] - e\sin\left(2\tan^{-1}\left[\sqrt{\frac{1-e}{1+e}}\right]\right)\right)\right)$$

$$\to \frac{T}{2}\left(-\frac{2}{\pi}\left(2\tan^{-1}\left[\sqrt{\frac{1-e}{1+e}}\right] - e\sin\left(2\tan^{-1}\left[\sqrt{\frac{1-e}{1+e}}\right]\right)\right)\right)$$

$$t_{B\to D} = \frac{T}{\pi}\left(-2\tan^{-1}\left[\sqrt{\frac{1-e}{1+e}}\right] + e\sin\left(2\tan^{-1}\left[\sqrt{\frac{1-e}{1+e}}\right]\right)\right)$$

j) $t_{D\to B} = t_{D\to P} + t_{P\to B}$

$$t_{D\to B} = T\left(\frac{1}{\pi}\left(2\tan^{-1}\left[\sqrt{\frac{1-e}{1+e}}\right] - e\sin\left(2\tan^{-1}\left[\sqrt{\frac{1-e}{1+e}}\right]\right)\right) + T\right)$$

k) $T_{D\to A} = T_{D\to D} - T_{A\to D}$

$$t_{D\to A} = \frac{T}{2}\left(3 - \frac{1}{\pi}\left(-2\tan^{-1}\left(\sqrt{\frac{1-e}{1+e}}\right) + e\sin\left(2\tan^{-1}\left[\sqrt{\frac{1-e}{1+e}}\right]\right)\right)\right)$$

2. Transfer of orbits

(a) Calculate the total Δv (magnitude) for a transfer to orbit r_2 by means of orbit r_{AB}. Provide a commented copy of your code, partial results of the intermediate calculations at A and at B, and the final answer for the total Δv required.

```
#expects vectors in perifocal frame
def perifocal_delta_v(current_orbit, desired_orbit):
  vq = desired_orbit[1] - current_orbit[1]
  vp = desired_orbit[0] - current_orbit[0]
  return math.sqrt((vq**2) + (vp**2))

#perifocal in km/s
vr1_a = [-3, 7]
vab_a = [-2, 10]

vab_b = [-6, -3]
vr2_b = [-4, -4]

#compute respective delta v
delta_transfer_a = orbital_equations_of_motion.perifocal_delta_v(vr1_a, vab_a)
delta_transfer_b = orbital_equations_of_motion.perifocal_delta_v(vab_b, vr2_b)
delta_transfer_total = delta_transfer_a + delta_transfer_b

print(f'Total delta transfer in km/s {delta_transfer_total:.4f}')

OUTPUT:

    Total delta transfer in km/s 5.3983
```

(b) Point A becomes the periapsis of the elliptical transfer orbit. If the true anomaly of point B in the transfer orbit (Tip: changes perifocal frame!) is $\theta = 120°$, calculate the time it takes to do the transfer

```
import math

mu = planetary_data.MU_EARTH_KM
vp_a = vector_functions.magnitude(vr1_a) #Velocity at A
rp_ab = (mu)/(vp_a**2) # v_t = sqrt(mu/r_c) (2.63)
vp_ab = vector_functions.magnitude(vab_a)
h = vp_ab*rp_ab #(2.31)
e = (h**2)/(mu*rp_ab) - 1 #keplers 2nd law
#given theta = 120 deg
theta = 120 * math.pi/180
E = 2*math.atan(math.sqrt((1-e)/(1+e))*math.tan(theta/2)) # (3.13b)
Me = E - e * math.sin(E) #(3.14)
a = rp_ab/(1-e) #(2.73)
T = 2*math.pi*math.sqrt((a**3)/mu) # (2.83)
```

```
t = (Me/(2*math.pi))*T/60 # (3.15) time in minutes

print(f'Time to point B in minutes {t:.3f}')

OUTPUT:

    Time to point B in minutes 59.205
```

(c) The satellite loses communications for 10 hours after firing at B, at what true anomaly will it be if it stayed in the elliptical orbit? (Note: if its $< 240°$ in the transfer orbit frame, it can get back into the circle at that point!)

Here is the function I wrote using the Newton-Raphson numerical method for finding true anomaly as a function of theta

```
#Newton-Rhapson method Alg 3.1
#returns in degrees
def theta_from_t(t, e, T):
  Me = t * (2*math.pi/T) # (3.8)
  E = Me + (e/2) if Me < math.pi else Me - (e/2)
  E_ratio = (E - e*math.sin(E)-Me)/(1 - e*math.cos(E))
  while(E_ratio > pow(10, -8)):
    E -= E_ratio
    E_ratio = (E - e*math.sin(E)-Me)/(1 - e*math.cos(E));

  e_ratio = math.sqrt((1+e)/(1-e))
  theta = (180/math.pi)*2*math.atan(e_ratio*math.tan(E/2))
  if theta < 0:
    return theta + 360
  else:
    return theta
```

Using the same e, t, and T computed in the previous problem, here is the code I used to find the new anomaly.

```
time = 10*3600+(t*60) #hours to seconds maintaining the previous time of flight
theta = orbital_equations_of_motion.theta_from_t(time, e, T)
print(f'True anomaly after 10 hours of idle {theta:.2f}')

OUTPUT:

    True anomaly after 10 hours of idle: 191.16 deg
```

3. Chasing down the ISS

A mission to the International Space Station (450km altitude) reaches the orbit (assume a circular orbit), but the timing was wrong, leaving the spacecraft too far away from the ISS.

(a) A maneuver of approximately 1 ISS orbit if the spacecraft is ahead 14°

```
alt_ISS = 450 #km
mu = planetary_data.MU_EARTH_KM
r = orbital_equations_of_motion.altitude_to_orbital_radius_earth(alt_ISS,km=True)
T = 2 * math.pi * math.sqrt((r**3)/mu) # (2.83)

#given
theta = 14 * math.pi/180 #less than 20 degrees is valid for the small sine approximation

T_xfer = T - (T*(-theta)/(2*math.pi)) #(phasing maneuver slides)
a_xfer = (((T_xfer / (2 * math.pi)) ** 2) * mu) ** (1/3) #(2.83)
print(f'Semi Major Axis of Transfer orbit (km) {a_xfer:.1f}')

#leading maneuver, start at periapsis
rp_xfer = r
e = 1 - rp_xfer/a_xfer #(2.73)

print(f'Eccentricity of Transfer orbit {e:.5f}')

#needs si units
xfer_orbit = orbital_equations_of_motion.orbital_state(
  a_xfer*1000, e, planetary_data.MU_EARTH)

v_reg = math.sqrt(mu/r) #(2.61)
vp_xfer = xfer_orbit['v_p (m/s)']/1000 #put in km/s
v_tot = 2 * abs(vp_xfer-v_reg) #(phasing manuever slides)

print(f'Total Needed Delta V in km/s: {v_tot:.4f}')

OUTPUT:

  Semi Major Axis of Transfer orbit (km) 7003.9
  Eccentricity of Transfer orbit 0.02511
  Total Needed Delta V in km/s: 0.1907
```

(b) A maneuver of 2 days (closest to, but not more than that) if the spacecraft is trailing 14°

```
theta = 14 * math.pi/180
#given
t = 2 * 3600 * 24 #hours to seconds
n = t/T # T is period of the ISS, found in the prev
T_xfer = T - ((theta*T)/(2*math.pi*n)) #(phasing maneuver slides)
```

```
#trailing maneuver, start at apo instead of peri
ra_xfer = r #radius of ISS orbit, found in prev

#mu is reused from last question (with units of km for length)
a_xfer = (((T_xfer / (2 * math.pi)) ** 2) * mu) ** (1/3) #(2.83)

print(f'Semi Major Axis of Transfer orbit (km) {a_xfer:.1f}')

e = ra_xfer/a_xfer - 1 #(2.74)

print(f'Eccentricity of Transfer orbit {e:.5f}')

xfer_orbit = orbital_equations_of_motion.orbital_state(
  a_xfer*1000, e, planetary_data.MU_EARTH)

#v_reg is the same as last question
#Want v_apo in the trailing case because we transfer orbits at
#apoapsis of the transfer orbit to gain phase
va_xfer = xfer_orbit['v_a (m/s)']/1000 #put in km/s
v_tot = 2 * abs(va_xfer-v_reg) #(phasing manuever slides)

print(f'Total Needed Delta V in km/s: {v_tot:.4f}')
```

```
OUTPUT:

  Semi Major Axis of Transfer orbit (km) 6822.2
  Eccentricity of Transfer orbit 0.00084
  Total Needed Delta V in km/s: 0.0064
```

4. Orbital Failure

(a) A rocket failed to put a satellite into MEO circular orbit (22,000km altitude). Instead it ended in an elliptical orbit with a perigee altitude of 1650km and an e= 0.58.

Here is my function for plotting orbits

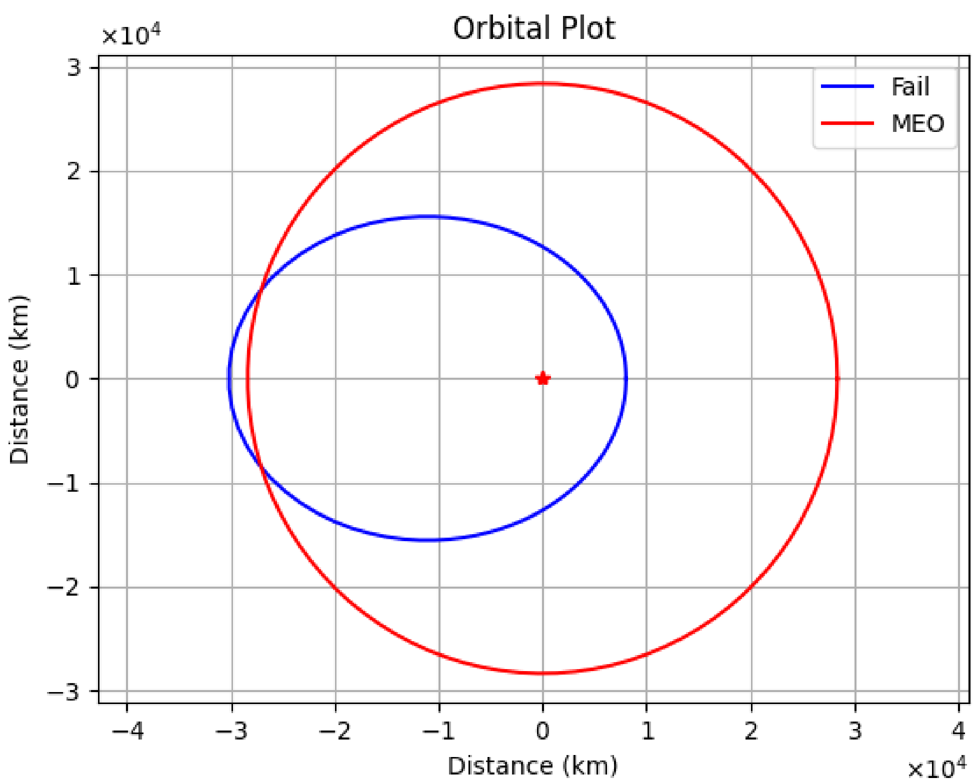
```
def orbit_plot(h, mu, e, ax=None, color='b', units='km', label=None):
  if ax is None: #create plot if no plot parameter was given
        fig, ax = plt.subplots()
        ax.set_title('Orbital Plot')
        ax.set_ylabel(f'Distance ({units})')
        ax.set_xlabel(f'Distance ({units})')


  theta = np.linspace(0, 2*np.pi,100)
  r = ((h**2)/mu)*(1/(1+(e*np.cos(theta)))) #radius plot

  #polar coords
  x = r*np.cos(theta)
  y = r*np.sin(theta)

  ax.plot(x, y, color=color, label=label if label else f'e={e:.2f}')
  ax.plot(0, 0, 'r*') #set focus to be 0,0
  ax.set_aspect('equal', adjustable='datalim')
  formatter = ScalarFormatter(useMathText=True)
  formatter.set_scientific(True)

  ax.xaxis.set_major_formatter(formatter)
  ax.yaxis.set_major_formatter(formatter)
  ax.grid(True)

  ax.legend(*ax.get_legend_handles_labels())

  return ax #pass ax back to main
```

This is the code I used to plot these orbits (orbital state is my state finding function)

```
from matplotlib import pyplot as plt

#given
rp_alt = 1650 #km
rp = orbital_equations_of_motion.altitude_to_orbital_radius_earth(rp_alt, km=True)
e  = 0.58

a = rp/(1-e) #(2.73)
```

```
mu = planetary_data.MU_EARTH
r_meo = orbital_equations_of_motion.altitude_to_orbital_radius_earth(22000, km=True)
failState = orbital_equations_of_motion.orbital_state(
  a*1000, e, mu) #needs SI units

mu = planetary_data.MU_EARTH_KM
v_meo = math.sqrt(mu/r_meo) #(2.61)

h_meo = v_meo * r_meo

h_fail = failState['h (m^2/s)'] / (10**6) #put in km

#get ax, fail orbit in blue
ax = orbital_equations_of_motion.orbit_plot(h_fail, mu, e, label='Fail')

#meo orbit in red
orbital_equations_of_motion.orbit_plot(h_meo, mu, 0, ax, color='r', label='MEO')

plt.show()
```

Here is the plot it produced



(b) Calculate the Δv (vector) for a single maneuver to change the elliptical orbit into the circular.

```
#Need to find theta of transfer happens when r_meo = r_fail

#e, mu (km), r_meo (km) from previous problem
theta = math.acos((1/e)*((h_fail**2)/(mu*r_meo)-1))
print(f'Theta transfer {theta * (180/math.pi):.2f} deg')

vr_fail = (mu/h_fail)*e*math.sin(theta) #(2.49)
vperp_fail = (mu/h_fail)*(1+e*math.cos(theta)) #(2.48)

#v_meo found in previous question

#vectors in vr, v_perp
vector_meo = [0, v_meo]
vector_fail = [vr_fail, vperp_fail]

delta_v = vector_functions.subtraction(vector_fail,vector_meo)
print(f'Delta V vector in format of Vr, Vperp  {[f"{v:.3f}" for v in delta_v]}, \n'
      f'Delta V magnitude (km/s) {vector_functions.magnitude(delta_v):.3f}')

OUTPUT:

  Theta transfer 162.46 deg
  Delta V vector in format of Vr, Vperp  ['0.980', '-1.242'],
  Delta V magnitude (km/s) 1.582
```

subtraction is a function I wrote to make vector subtraction a little easier.

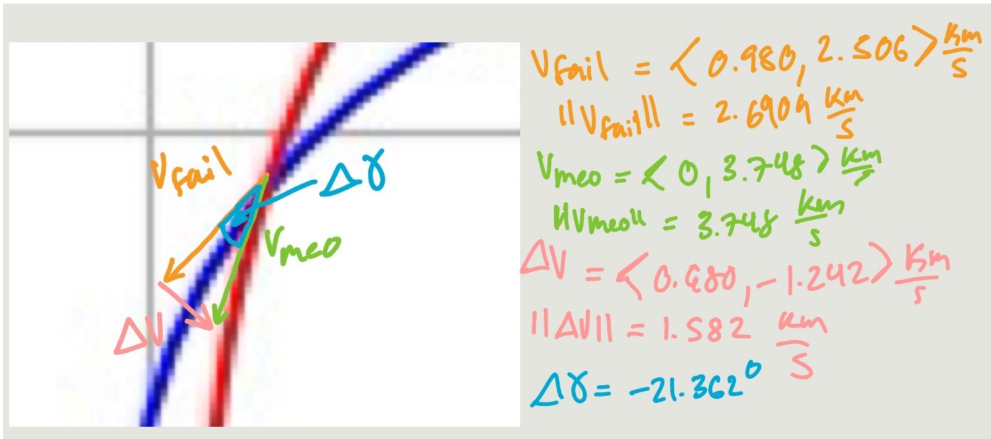
```
#subtracts v_subtractor from v, expects both vectors to be of the same length
def subtraction(v, v_subtractor):
 result = []
 for i in range(len(v)):
  result.append(v[i] - v_subtractor[i])
 return result
```

(c) Draw the "$\Delta v = v_2 - v_1$" triangle in the figure from part (a) (you can do it by "hand", in a drawing program, annotating a picture, or you can program it into Matlab or Python, any method you do is acceptable), include: i. the magnitude and direction of the Δv, v_1, and v_2

Here is the code I used to find the change in flight path angle

```
#reused from the above problem
gamma_fail = orbital_equations_of_motion.flight_path_angle(
  vector_fail[1], vector_fail[0])
gamma_meo  = orbital_equations_of_motion.flight_path_angle(
  vector_meo[1], vector_meo[0])
delta_gamma = gamma_meo - gamma_fail

print(f'Delta gamma {delta_gamma:.3f} deg')
```

```
OUTPUT:

  Delta gamma -21.362 deg
```



(d) Draw a sketch (by hand or a drawing program) of your initial guess for the two Hohmann Transfer options to go from the bad elliptical orbit to the intended circular orbit, include the direction of the required Δv's

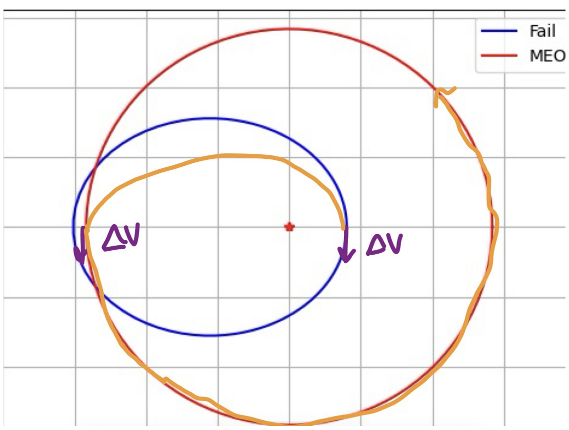


Figure 1: Hohmann at peri

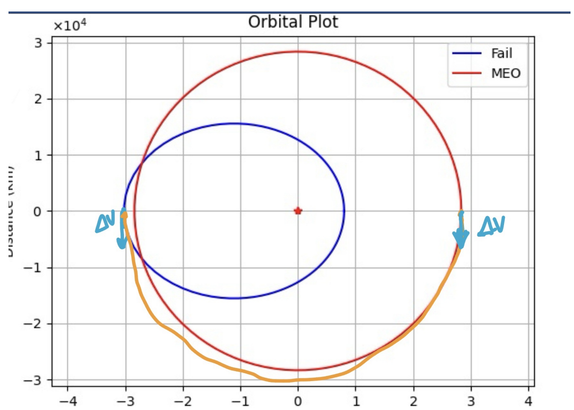


Figure 2: Hohmann at apo

(e) Calculate the required Δv's for the two transfers.

Here is the function I wrote to compute the Hohmann transfer of any two coaxial orbits.

```
#requires orbits to coaxial
#returns delta V
#expects mu in SI units
def hohmann_transfer(init_state, final_state, mu, km_s = False, start_peri=True):

  if start_peri:
```

```
    #complete transfer at final apo
    rp_xfer = init_state['r_p (m)']
    ra_xfer = final_state['r_a (m)']
  else:
    #complete transfer at final peri
    rp_xfer = final_state['r_p (m)']
    ra_xfer = init_state['r_a (m)']

  a_xfer = (rp_xfer + ra_xfer)/2
  e_xfer = (ra_xfer - rp_xfer) / (ra_xfer + rp_xfer)

  #get state velocities
  transfer_state = orbital_state(a_xfer, e_xfer, mu)
  va_xfer = transfer_state['v_a (m/s)']
  vp_xfer = transfer_state['v_p (m/s)']

  if start_peri:
    v_start_orbit = init_state['v_p (m/s)']  #burn at periapsis
    v_end_orbit   = final_state['v_a (m/s)']  #burn at apoapsis
    v_start_xfer  = vp_xfer#start transfer at peri
    v_end_xfer    = va_xfer
  else:
    v_start_orbit = init_state['v_a (m/s)']  #burn at apoapsis
    v_end_orbit   = final_state['v_p (m/s)'] #burn at periapsis
    v_start_xfer  = va_xfer#start transfer at apo
    v_end_xfer    = vp_xfer

  delta_v1 = abs(v_start_xfer - v_start_orbit) #burn 1
  delta_v2 = abs(v_end_orbit - v_end_xfer)     #burn 2

  if km_s:
    return (delta_v1 + delta_v2)/1000 #full transfer magnitude
  else:
    return delta_v1 + delta_v2 #full transfer magnitude
```

Subsequently, here is the code I used to determine the required Δv for both hohmann transfers

```
#from question A we have fail state and r_meo already
failState, r_meo

mu = planetary_data.MU_EARTH #si units

meo_state = orbital_equations_of_motion.orbital_state(
  r_meo*1000, 0, mu) #correct MEO to m

orbital_equations_of_motion.print_state(meo_state)
orbital_equations_of_motion.print_state(failState)
```

```
delta_v_start_peri = orbital_equations_of_motion.hohmann_transfer(
  failState,meo_state,mu,km_s=True, start_peri=True)
delta_v_start_apo  = orbital_equations_of_motion.hohmann_transfer(
  failState,meo_state,mu,km_s=True, start_peri=False)

print(f'Hohmann Delta V starting from Peri {delta_v_start_peri:.4f} km/s\n'
      f'Hohmann Delta V starting from Apo  {delta_v_start_apo:.4f} km/s')
```

```
OUTPUT:

    Hohmann Delta V starting from Peri 1.3182 km/s
    Hohmann Delta V starting from Apo  1.2796 km/s
```