Luke Verlangieri
AA310 HW5

1. Interplanetary TRANSFER!

(a) Develop a function in Matlab or Python which takes as input the altitude of the parking orbit and calculates the $|\Delta v|$ required to enter the parking orbit after arrival to Mars' SoI (i.e., the magnitude of the first Δv above).

The solution I found for this problem uses various data out of Curtis Appendix A, the names are self explanatory for the most part, however, the orbital velocities are calculated values, here is the calculation for them

```
""" VELOCITIES AROUND SUN (ASSUMES CIRCULAR ORBITS) """

V_EARTH_SUN_KM = math.sqrt(MU_SUN_KM/SUN_EARTH_KM)
V_MARS_SUN_KM = math.sqrt(MU_SUN_KM/SUN_MARS_KM)
```

Here is the function I wrote to determine the needed ΔV per parking altitude.

```
#need the parameter of the hyperbolic excess speed to solve
#assumes going to mars from earth
#assumes km
def arrival_delta_v_mars(parking_alt):

  r_earth_sun = planetary_data.SUN_EARTH_KM
  r_mars_sun = planetary_data.SUN_MARS_KM
  mu_sun = planetary_data.MU_SUN_KM

  mu = planetary_data.MU_MARS_KM
  r_mars= planetary_data.RADIUS_MARS_KM + parking_alt #Find parking radius
  v_parking = np.sqrt(mu/r_mars) # Curtis 2.63
  v_esc = np.sqrt((2*mu)/r_mars) # Curtis 2.91

  # Curtis 8.35
  v_ab_a = np.sqrt((2*mu_sun*(r_earth_sun/r_mars_sun))/(r_earth_sun + r_mars_sun))
  v_inf = v_ab_a - planetary_data.V_MARS_SUN_KM # Curtis 8.51

  return abs(v_parking - np.sqrt((v_esc**2)+(v_inf**2))) # Curtis 8.40
```

I then used this to code to plot the function for multiple delta Vs

```
#definitely dont wanna park here but it is the technical minimum
min_parking_alt = 0

#max is at the edge of the soi (convert to an alt)
max_parking_alt = planetary_data.SOI_MARS_KM - planetary_data.RADIUS_MARS_KM
```

```
alts = np.linspace(min_parking_alt, max_parking_alt, 10000)
delta_vs = orbital_equations_of_motion.arrival_delta_v_mars(alts)

def delta_v_plot(alts, delta_vs) :

  fig, ax = plt.subplots()
  ax.set_title('Orbital Plot')
  ax.set_ylabel(f'Delta Vs (Km/s)')
  ax.set_xlabel(f'Alts (Km)')

  ax.semilogx(alts, delta_vs, color='g', label='Delta V vs Parking Alt')
  plt.show()

delta_v_plot(alts, delta_vs)
```

Here is a zoomed in photo of the plot I generated using this.

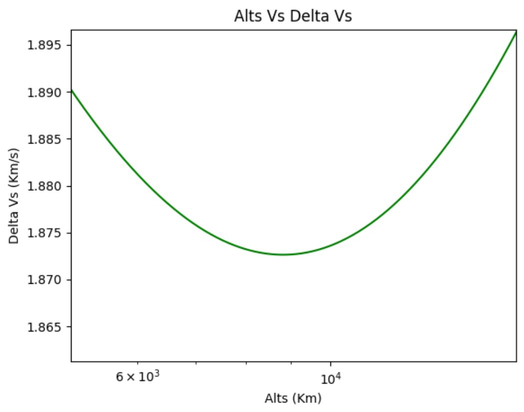

Figure 1: Zoomed in at minimum ΔV

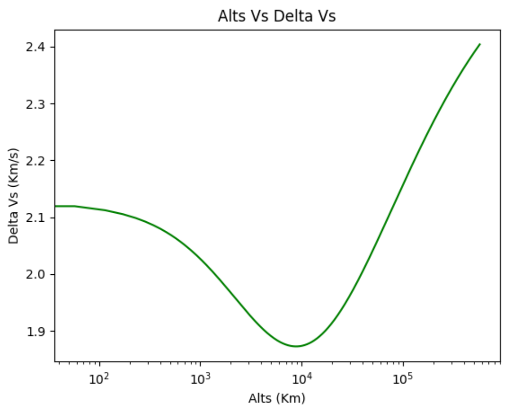

Figure 2: Plot of parking alts vs ΔV

Hovering my cursor over the lowest point, the smallest ΔV is at a parking alt of about 8800 Km and has a value of ≈ 1.87[Km/s]. Because this is the minimum ΔV, the parking altitude *should* be around 8800 Km.

(b) Create a new formula that calculates the 2 Δv's required for the Homann transfer from the parking orbit to the surface (assuming no atmosphere, etc). Add that to the first Δv to calculate the total Δv to the surface.

For this question by the wording I assumed that we are in a zero altitude orbit, meaning we are still orbiting the planet but at an unreal zero altitude orbit. I reused my hohmann transfer function from previous assignments.

```
#Total delta V needed to return from orbit of the parking alt
#Techically still orbiting just a 0 alt orbit
#assumes km
```

```
def return_from_parking_orbit_mars(parking_alt):

  mu = planetary_data.MU_MARS_KM
  r_parking = parking_alt + planetary_data.RADIUS_MARS_KM
  v_parking = np.sqrt(mu/r_parking) # 2.63

  #find the states of these two orbits
  parking_state = orbital_state(r_parking, 0, mu, km=True)
  zero_orbit_state = orbital_state(
    planetary_data.RADIUS_MARS_KM, 0, mu, km=True)

  deltaV, transfer_orbit = hohmann_transfer(
    parking_state, zero_orbit_state, mu)

  return deltaV

def three_delta_V(parking_alt):
  deltaV1 = arrival_delta_v_mars(parking_alt)
  deltaV2_V3 = return_from_parking_orbit_mars(parking_alt)
  return deltaV1 + deltaV2_V3
```

Here is the code I used to generate the plot, reusing my previous plotting function.

```
alts = np.linspace(min_parking_alt, max_parking_alt, 10000)
delta_vs = orbital_equations_of_motion.three_delta_V(alts)

delta_v_plot(alts, delta_vs)
```

Here is the plot that it generated

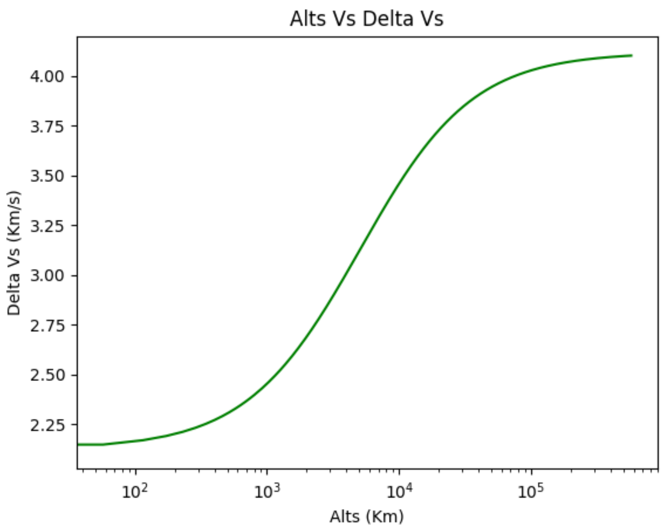

Figure 3: Parking Alts vs Delta Vs

As we increase the parking orbit distance, a greater ΔV is needed in order to move from the parking orbit to the 0 alt orbit. Therefore logically the closer you set your parking orbit to zero, the less delta V in the hohmann transfer you need in order to get to the zero alt orbit. That being said, we need a parking altitude of about 0 Km for max efficiency.

(c) Calculate the Δv to slow down from "0-altitude orbit" to the actual surface speed of Mars and add it to the new total Δv which includes the 4 Δv's.

I wrote this function to find the total ΔV needed to reach the planet based on a given parking alt, including the rotation of mars. The sidereal rotation value came from curtis Appendix A.

```
""" SIDE REAL ROTATION PERIOD (s) """

SIDEREAL_ROTATION_MARS = 24.62 * 3600

def fourth_delta_v(parking_alt):
  #find the speed at which mars rotates
  angular_velo_mars = 2 * math.pi/planetary_data.SIDEREAL_ROTATION_MARS
  v_surface_mars = angular_velo_mars * planetary_data.RADIUS_MARS_KM

  #find the summed delta V
  mu = planetary_data.MU_MARS_KM
  delta_v_sum = three_delta_V(parking_alt)
  v_0_alt = np.sqrt(mu/planetary_data.RADIUS_MARS_KM) #parking speed

  return abs(v_0_alt - v_surface_mars) + delta_v_sum #total delta V
```

I plotted this as well, unsurprisingly the plot takes the same form as the previous question.

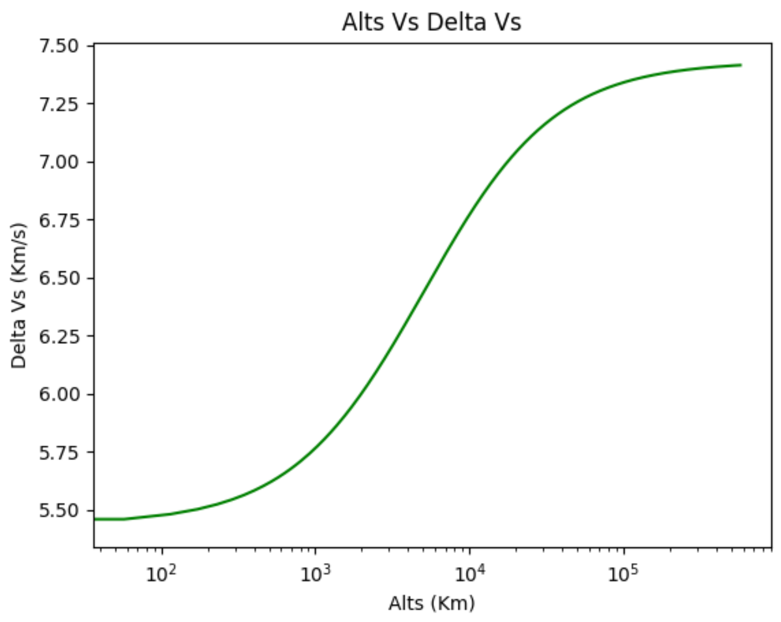

Figure 4: Total Arrival ΔV based on parking orbit

Given that the lowest ΔV is clearly at the parking orbit, I used this code to extract the value at the numerically "most optimal" parking altitude.

```
small_delta_v = orbital_equations_of_motion.fourth_delta_v(0)
print(f'Smallest Delta V = {small_delta_v:.5g} Km/s')

OUTPUT:

    Smallest Delta V = 5.4377 Km/s
```

(d) What would be your recommendation for a parking orbit?
I would recomend that the parking orbit be at about 150km above the surface (assuming no atmosphere), putting the parking orbit at 0km is dangerous and realistically the satellite would crash, but the closer we get to 0km the cheaper the flight will be fuel wise.

(e) What is the aiming radius ($\Delta = b$) to get to the recommended parking orbit?

I wrote this function to determine the needed aiming radius to get to a given parking orbit on mars.

```
#where r_inner and r_outer are relative to the sun
def get_v_ab_a(r_inner, r_outer):
  mu_sun = planetary_data.MU_SUN_KM
  # Curtis 8.35
  v_ab_a = np.sqrt((2*mu_sun*(r_inner/r_outer))/(r_inner + r_outer))
  return v_ab_a
```

```
def aiming_radius_from_parking_radius_mars(parking_alt):

  r_mars = planetary_data.RADIUS_MARS_KM
  mu = planetary_data.MU_MARS_KM
  parking_radius = parking_alt + r_mars

  v_ab_a = get_v_ab_a(
    planetary_data.SUN_EARTH_KM, planetary_data.SUN_MARS_KM)

  v_esc = np.sqrt((2*mu)/parking_radius) # Curtis 2.91
  v_inf = v_ab_a - planetary_data.V_MARS_SUN_KM # Curtis 8.51

  # at periapsis h = v_rp*r_rp
  h_hyper = parking_radius * np.sqrt((v_esc**2) + (v_inf**2))

  e = (h_hyper**2)/(mu * parking_radius) - 1 # Keplers 2nd

  #get state of entry orbit
  hyper_state = hyperbolic_state(e, parking_radius, mu)

  return hyper_state['b (km)'] #return aiming radius
```

I used this code to extract the Aiming radius for my parking altitude recommendation, as well as the "optimal" (unsafe) parking altitude.

```
aiming_radius_0 = orbital_equations_of_motion.aiming_radius_from_parking_radius_mars(
  0
)

aiming_radius_150 = orbital_equations_of_motion.aiming_radius_from_parking_radius_mars(
  150
)

print(f'Aiming Radius at 150 km {aiming_radius_150:.5g} Km')
print(f'Aiming Radius at 0 km {aiming_radius_0:.5g} Km')

OUTPUT:

    Aiming Radius at 150 km 7083.4 Km
    Aiming Radius at 0 km 7281.6 Km
```

2. Dark Side of [the] Mars
The mission went wrong! After trying to arrive to Mars close to the surface ($r_c = 3800$), the satellite arrived too close to Mars. The satellite arrived with $\Delta = 5000.0$ km on the shadow side at SoI:

(a) What is v_∞ on arrival from Earth via a Hohmann transfer? (assume circular orbits for the planets) [Answer: -2.6483 km/s]

Per the derivation in code from the previous question, v_∞ is given by $v_{ABa} - v_{mars,sun}$. Because Mars is moving faster than the satellite in this circumstance, v_∞ is guarenteed to be negative relative to the sun. Given this is a hohmann transfer, the speed here is the same regardless of the orientation of mars. Here is the function I wrote to get v_∞. Note the funtion `v_ab_a` is the same function from the previous question.

```
#v_inf
def get_v_inf_arrival_outer(r_inner, r_outer, planet_speed):
  v_ab_a = get_v_ab_a(r_inner, r_outer)
  return v_ab_a - planet_speed
```

And here is the code I used to extract v_∞

```
planet_speed = planetary_data.V_MARS_SUN_KM
r_earth_sun = planetary_data.SUN_EARTH_KM
r_mars_sun = planetary_data.SUN_MARS_KM

v_inf = orbital_equations_of_motion.get_v_inf_arrival_outer(
  r_earth_sun, r_mars_sun, planet_speed)

print(f'V_inf entering mars {v_inf:.5g} Km/s')

OUTPUT:

    V_inf entering mars -2.6483 Km/s
```

(b) What is the periapsis (r_p) of the arrival hyperbolic trajectory (with respect to Mars)? [Answer 1785.4km]

Here is the code I used to solve this problem

```
#given
delta = 5000 #km
mu = planetary_data.MU_MARS_KM

a = mu/(v_inf**2) # Curtis 2.112
e = math.sqrt(((delta/a)**2) + 1) # Curtis 2.107

rp = a * (e-1) # curtis 2.101
```

```
print(f'Rp of the current hyperbolic orbit {rp:.5g} Km')
```

```
OUTPUT:

    Rp of the current hyperbolic orbit 1785.4 Km
```

(c) At what θ (wrt to the arrival hyperbola) would it reach the target radius of 3800km? [Answer: -86.583°]

Here is the code I used to solve this problem

```
#given
r_target = 3800
a, e, mu #the same since last question
hyper_state = orbital_equations_of_motion.hyperbolic_state(e, rp, mu)
h = hyper_state['h (km^2/s)']

# Keplers 2nd r = h^2/mu(1/(1+ecos(theta)))
# Need to add negative because the angle is opposite of our sign convention
# for a hyperbola
theta = -math.acos((1/e)*((h**2)/(r_target*mu) - 1))
print(f'Theta at which r_target is reached {theta  * (180/math.pi):.5g} Deg')
```

```
OUTPUT:

    Theta at which r_target is reached -86.583 Deg
```

(d) What will be the corresponding flight path angle, γ, at that point? [Answer: -50.141°]

Here is the code I used to solve this problem

```
theta, e, # Same from previous

# Curtis 2.52 tan gamma = (esin(theta))/(1+ecos(theta))
gamma = math.atan(e*math.sin(theta)/(1 + (e * math.cos(theta)))) * 180/math.pi

print(
  f'Flight path angle at theta = {theta * (180/math.pi):.5g} deg,'
  f'gamma = {gamma:.5g} deg')
```

```
OUTPUT:

    Flight path angle at theta = -86.583 deg, gamma = -50.141 deg
```

(e) What will be the speed of the spacecraft when it reaches the target radius of r_c=3800km? (i.e, before the Δv, along the hyperbola) [Answer: 5.4371km/s]

Here is my code to solve this problem

```
mu, h, theta, e #from previous

# Curtis 2.49
v_r = (mu/h)*e*math.sin(theta)

# Curtis 2.48
v_perp = (mu/h)*(1 + (e*math.cos(theta)))

#want speed, not vector
v = math.sqrt((v_r**2) + (v_perp**2))

print(f'Speed at this point, |V| = {v:.5g} km/s')

OUTPUT:

    Speed at this point, |V| = 5.4371 km/s
```

(f) What would be the required Δv for insertion into circular orbit? [Answer: 4.1756km/s]

Here is my code to solve this problem, I have used the functions subtraction and magnitude in the past, they are very self explanatory in their namings, incase you are interested in those functions, here they are.

```
#returns a scalar, takes in a vector (list) of any size
def magnitude(v) -> float:
 inside_sqrt = 0
 for var in v : inside_sqrt += math.pow(var, 2) #square each value
 return math.sqrt(inside_sqrt) #take sqrt & return

#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

#speed of the parking orbit
v_park = [math.sqrt(mu/r_target), 0]
v_hyper = [v_perp, v_r]

#get delta V vector
delta_v_vector = vector_functions.subtraction(v_park, v_hyper)

#find magnitude of that vector
delta_v = vector_functions.magnitude(delta_v_vector)

print(f'Magnitude of Delta V to change orbits {delta_v:.5g} km/s')
```

OUTPUT

```
    Magnitude of Delta V to change orbits 4.1756 km/s
```

(g) Draw the "Δv triangle" ($\Delta v = v_2 - v_1$) for this problem, scale the velocities to each other (no need to draw the orbital trajectory itself, just the triangle; you can draw it by hand or code it, but be careful with the scales of the vectors!).

Here is my drawing of the velocity vectors, I tried to capture the fact that the vectors have an equal perpendicular componenent, so the vast majority of the delta v is in the radial direction.

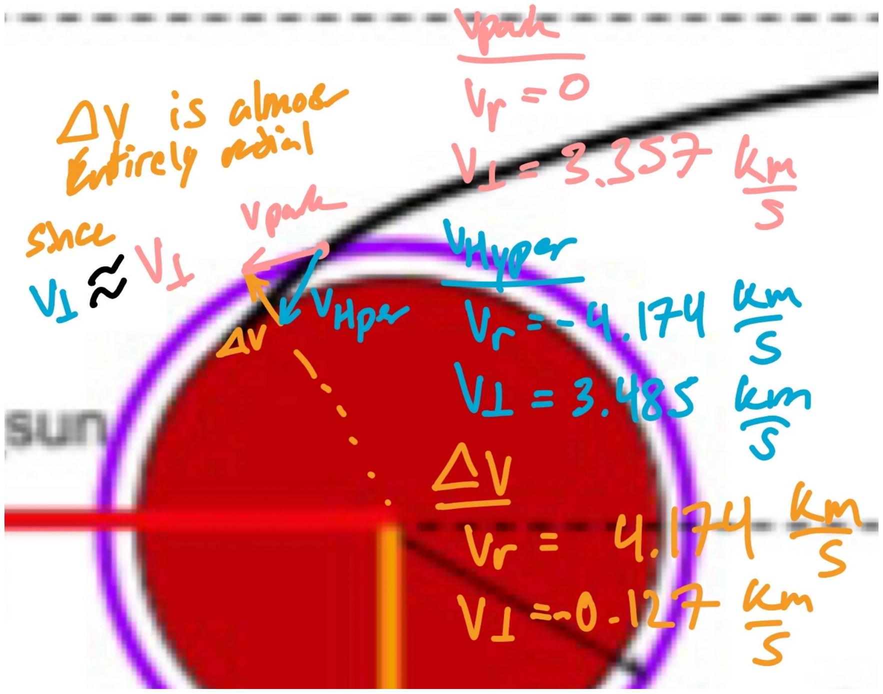

Figure 5: Drawing of vectors for this trajectory

(h) After the Δv, did the velocity increase or decrease with respect to:

- Mars
- The sun

Show your work to determine that numerically (you can do with code or by hand, but show your work).

```
#Because these velocities were taken with respect to mars,
#Can just compare the magnitude of these two vectors to find

mag_v_hyper = vector_functions.magnitude(v_hyper)
```

```
mag_v_park = vector_functions.magnitude(v_park)

#if |v_hyper| > |v_park| the satellite lost speed with respect to mars

if (mag_v_hyper > mag_v_park):
  print(f'Satelite lost {abs(mag_v_hyper - mag_v_park):.5g} km/s relative to Mars')
else:
  print(f'Satelite gained {abs(mag_v_hyper - mag_v_park):.5g} km/s relative to Mars')

# We that v_r makes a |theta| (found above) degree angle with mars orbit
# relative to the sun. Therefore these vectors are 90 - |theta| degrees
# rotated from what they should be relative to the sun. Using a rotation
# matrix we can find the new magnitudes of these vectors after being rotated.
# Additionally, we need to add the velocity of mars to the perpendicular
# components after rotating

def rotation_matrix(v_r, v_perp, theta):
  return [
    v_perp * math.cos(theta) - v_r * math.sin(theta),
    v_perp * math.sin(theta) + v_r * math.cos(theta)
  ]

#rotates below the horizontal
v_park_sun = rotation_matrix(v_park[1], v_park[0], -(math.pi/2 - abs(theta)))
v_park_sun[0] += planetary_data.V_MARS_SUN_KM
v_hyper_sun = rotation_matrix(v_hyper[1], v_hyper[0], -(math.pi/2 - abs(theta)))
v_hyper_sun[0] += planetary_data.V_MARS_SUN_KM

mag_v_hyper_sun = vector_functions.magnitude(v_hyper_sun)
mag_v_park_sun = vector_functions.magnitude(v_park_sun)

if (mag_v_hyper_sun > mag_v_park_sun):
  print(f'Satelite lost {abs(mag_v_hyper_sun - mag_v_park_sun):.5g} km/s relative to Sun')
else:
  print(f'Satelite gained {abs(mag_v_hyper_sun - mag_v_park_sun):.5g} km/s relative to Sun
```

```
OUTPUT:

    Satelite lost 2.0794 km/s relative to Mars
    Satelite lost 0.22457 km/s relative to Sun
```

3. (Fake) Mission Project

The Mission Project (to Mars) only considers the required Δv (fuel use) for the mission, but it ignores time. This question asks you to complement what you learn in the mission project, with thinking about the amount of time it would take to complete the mission. For this example we will only care about the "inter-planetary" times:

- Time to takeoff from Earth and be in parking orbit before departure : ignored
- Time to travel Earth to Mars
- Time to orbit Mars in parking orbit / land : ignored
- Wait in Mars until the planets align again
- Time to takeoff from Mars and be in parking orbit before departure : ignored
- Time to travel to Mars from Earth
- Time to orbit Earth in parking orbit / land : ignored

(a) What is the flight time, in days, from Earth to Mars? (if you created a Hohmann Transfer function in Matlab/Python for HW4, you can use that function) [Answer: 258.79 days]

I wrote this hohamnn transfer function after completing homework 4 (I realized how useful it would have been and just wanted it in my library).

```
#requires orbits to coaxial
#returns delta V
#expects units of km for length
def hohmann_transfer(init_state, final_state, mu, start_peri=True):

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

  #find transfer orbit parameters
  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, km=True)
  va_xfer = transfer_state['v_a (km/s)']
  vp_xfer = transfer_state['v_p (km/s)']

  if start_peri:
    v_start_orbit = init_state['v_p (km/s)']  #burn at periapsis
    v_end_orbit   = final_state['v_a (km/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 (km/s)']  #burn at apoapsis
    v_end_orbit   = final_state['v_p (km/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

  return ((delta_v1 + delta_v2), transfer_state) #full transfer magnitude
```

Here is the code I used to solve the problem

```
r_mars = planetary_data.SUN_MARS_KM
r_earth = planetary_data.SUN_EARTH_KM
mu = planetary_data.MU_SUN_KM

#Get States
state_mars = orbital_equations_of_motion.orbital_state(
  r_mars,0,mu,km=True)

state_earth = orbital_equations_of_motion.orbital_state(
  r_earth,0,mu,km=True
)

delta_v, transfer_state = orbital_equations_of_motion.hohmann_transfer(
  state_earth, state_mars, mu, start_peri=True
)

Txfer = transfer_state['T (s)']
t_apo = Txfer/2 #at Apo

print(f'Time to Mars from Earth: {t_apo / (3600 * 24):.5g} days')

OUTPUT

    Time to Mars from Earth: 258.79 days
```

(b) Where does Mars need to be with respect to Earth, so that the satellite reaches Mars at the end of the Hohmann transfer? [Answer: $+44.329°$ from Hohmann periapsis]

I wrote this function that uses Curtis 8.12 to find the initial phase angle for a desired transfer.

```
#n is the mean angular velocity (2*pi)/T
#returns the phase angle difference in degrees
```

```
#inner starts transfer at theta = 0 deg
def hohmann_inner_outer_phase_difference(n, t):
  return (math.pi - n*t) * 180/math.pi # Curtis 8.12
```

Here is the code I used to find the value, we know that the transfer needs to be completed at Txfer/2.

```
r_mars, r_earth, mu, t_apo # same from previous

T_mars = 2 * math.pi * math.sqrt((r_mars**3)/mu) #Period of mars

n_mars = 2 * math.pi / T_mars #mean angular velocity

#Find phase angle
phi_i = orbital_equations_of_motion.hohmann_inner_outer_phase_difference(
  n_mars, t_apo
)

print(f'Initial Phase Difference Phi: {phi_i:.5g} deg')
```

```
OUTPUT

    Initial Phase Difference Phi: 44.329 deg
```

(c) To return to Earth with a Hohmann transfer, the opposite will need to be true: Where does Earth need to be with respect to Mars, so that the satellite reaches Earth? [Answer: -75.097° from Hohmann apoapsis]
I reused my previous function to find this, since the same logic applies in this case. However, because Earth rotates around the Sun faster than Mars, we expect a negative phase angle.

```
Txfer, r_earth, mu #same from previous

t_peri = Txfer/2

T_earth = 2 * math.pi * math.sqrt((r_earth**3)/mu)

n_earth = 2 * math.pi / T_earth

phi_f = orbital_equations_of_motion.hohmann_inner_outer_phase_difference(
  n_earth, t_peri
)

print(f'Return Phase Difference Phi: {phi_f:.5g} deg')
```

```
OUTPUT:

      Return Phase Difference Phi: -75.097 deg
```

(d) What is the “wait time” (in days and/or Earth years) from arrival to the next departures for a Hohmann transfer? [Answer: 454.64 days]

After arriving at mars, we need to wait for the earth to be at an angle of negative -75.097 deg. Curtis 8.15, we compare the difference between mean angular velocities and the given phase shift between the planets after arrival. Here are the functions I wrote using the equations derived by Curtis for the wait time.

```
def waiting_time_after_inner_outer(n_outer, n_inner, delta_phi):
  # Uses Curtis 8.15
  return ((2*math.pi) - (delta_phi * math.pi/180))/(n_inner - n_outer)

def compute_delta_phi(n_inner, T_xfer):
  # Curtis 8.17
  return abs(2*(hohmann_inner_outer_phase_difference(n_inner, T_xfer/2)))

delta_phi = orbital_equations_of_motion.compute_delta_phi(
  n_earth, Txfer
)

t_wait = orbital_equations_of_motion.waiting_time_after_inner_outer(
  n_mars, n_earth, delta_phi
)

print(f'Time spent waiting {t_wait/(3600*24):.5g} days')
```

```
OUTPUT:

    Time spent waiting 454.64 days
```

(e) What is the minimum total round trip time to Mars for this ideal Hohmann transfer?

The minimum time of the ideal hohmann transfer is the time traveling to Mars, coupled with the time waiting on Mars, followed by the journey back to Earth. In this case, That is one full period of travel, plus the time waiting on Mars. In code,

```
Total_time_for_hohmann = (Txfer + t_wait)/(3600 * 24)

print(f'Total time for ideal hohmann transfer {Total_time_for_hohmann:.5g} days')
```

```
OUTPUT:

    Total time for hohmann transfer 972.21 days
```

4. Slingshotting Jupiter

Jupiter is so big it always makes sense to use it. Consider a mission to Saturn that wants to use Jupiter for a "sling shot". The management wants to minimize fuel, so its willing to wait to get to Jupiter via a Hohmann transfer. (Assume circular co-planar planetary orbits.)

(a) For the arrival to Jupiter:

i. On what side does the satellite need to arrive: when Jupiter is trailing or ahead of the satellite?

ii. Draw 2 sketches (by hand OK, similar to Curtis Fig 8.18) for the two cases of arrival:

1. Sun side 2. Shade side

Include in your drawings:

- Jupiter
- the probe
- the direction to the sun
- $v_{\text{Jupiter\_Sun}}$
- $v_{\text{sat\_sun\_in}}$
- $v_{\infty\text{in}}$
- $v_{\infty\text{out}}$
- Δv_{∞}
- $v_{\text{sat\_sun\_out}}$
- Δv_{sun}

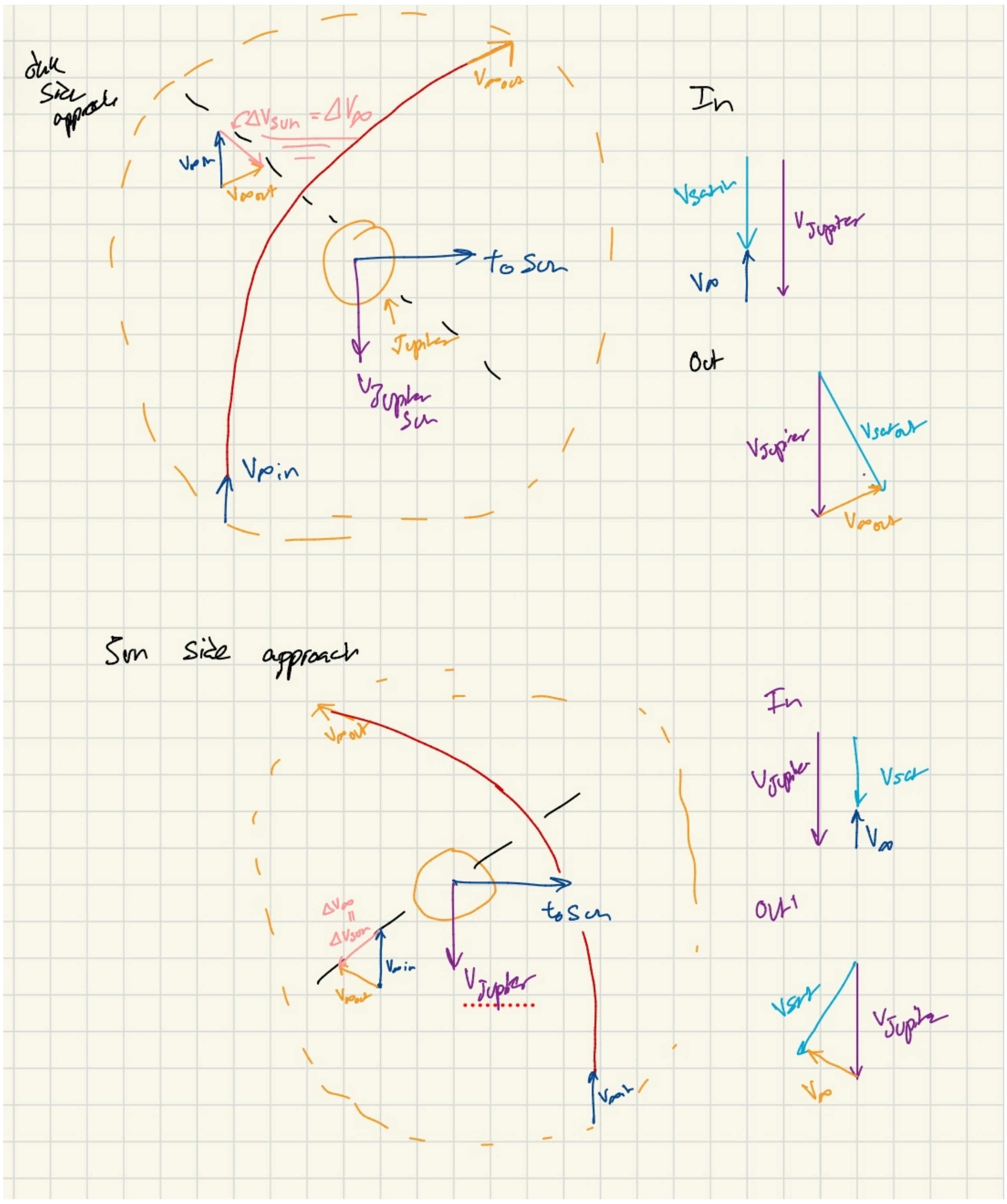

Figure 6: Options for dark side or Sun side entrance for a trailing trajectory

(b) Draw (by hand, drawing program, or any way you want) 2 sketches the expected trajectories from Earth to Jupiter and from Jupiter to Saturn of the Sun and shadow side arrivals.

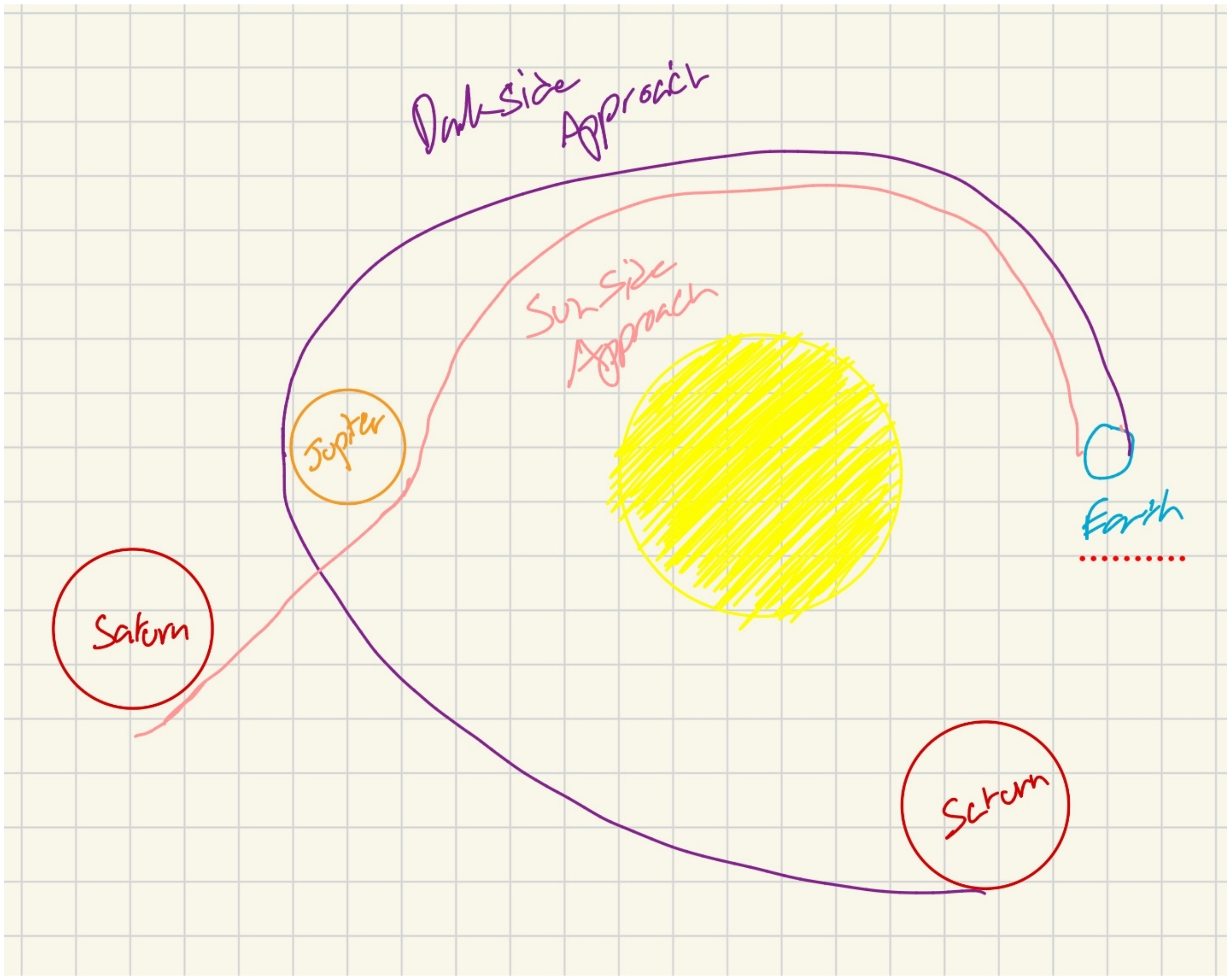

Figure 7: Earth and Jupiter & Jupiter and Saturn

(c) Determine the magnitude of v_∞. Show your work andor MatlabPython commented code. (Tip: you should be able to use previous functions from this HW with different input parameters!) [Answer: -5.6441 km/s]

Here is the code I used to solve this, I reused my `get_v_inf_arrival_outer` function from earlier. The speed of jupiter is caculated the same way it was for Mars and Earth.

```
r_sun_earth = planetary_data.SUN_EARTH_KM
r_sun_jupiter = planetary_data.SUN_JUPITER_KM
speed_jupiter = planetary_data.V_JUPITER_SUN_KM

v_inf = orbital_equations_of_motion.get_v_inf_arrival_outer(
  r_sun_earth, r_sun_jupiter, speed_jupiter
  )

print(f'|V_inf| entering jupiter: {v_inf:.5g} km/s')
```

```
OUTPUT:

    |V_inf| entering jupiter: -5.6441 km/s
```

(d) The required exit velocity and flight path angle for the sun-side arrival are: $v_{\text{sat\_saturn}} = 14.410$ km/s $\gamma_{\text{sat\_saturn}} = -23.044$ deg What is the required aiming radius ($\Delta = b$) for arrival to Jupiter? [Answer 3,838,500km]

The flight path angle is the angle between the velocity of the planet and the velocity of the satellite, we know v_∞ is the same throughout the hyperbola. Therefore if we the flight path angle of the exit velocity of the satellit, as well as the opposing side (v_∞). We can find the angle that v_inf makes with the velocity of the plane. Because this is a hohmann transfer we know that v_∞ is directly vertical upon entry to the planet, therefore the angle found by the law of sines this angle is the angle between $v_{\infty_{in}}$ and$v_{\infty_{out}}$. Which is δ, the turn angle.

```
v_inf # same as previous

# know |v_inf_in| = |v_inf_out|
mu = planetary_data.MU_JUPITER_KM
a_hyper = mu/(v_inf**2) # Curtis 2.112

#Found the same way as previous velocities
V_jupiter = planetary_data.V_JUPITER_SUN_KM
# V = math.sqrt(mu_sun/r_jupiter_sun)

#given exit properties
V_sat_saturn = 14.410
gamma_saturn = -23.044 * (math.pi/180)

# Law of sines
turn_angle = math.asin((V_sat_saturn*math.sin(gamma_saturn)/v_inf))

# Adjust for obtuse
turn_angle = math.pi/2 + (math.pi/2 - abs(turn_angle))

print(f'Turn Angle: {turn_angle*(180/math.pi):.5g} deg')

# Curtis 2.100
e = 1/(math.sin(turn_angle/2))

# Curtis 2.107
delta = a_hyper * math.sqrt((e**2) - 1)

print(f'Required Aiming Radius {delta:.5g} km')

OUTPUT:

    Turn Angle: 92.01 deg
```

```
Required Aiming Radius 3.8415e+06 km
```