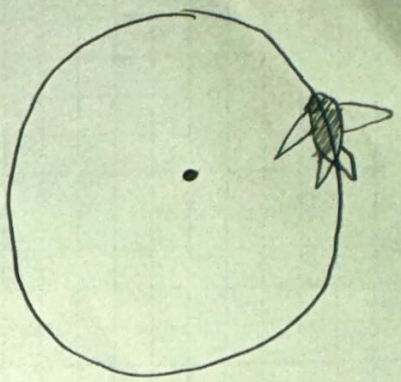
AA 310 Midterm | Luke Verlangieri | 11/2/2025

1) Turning The Airplane



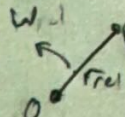
a) standing on the ground is an inertial reference frame. This is because all motion of the plane is taken relative to the "stationary" earth (We can consider the surface of the earth to be an inertial reference frame because every object close to the surface of the earth undergoes the same forces from rotation and gravity). All behavior of the plane follows newtonian laws, which defines this as inertial.

b) The reference frame of someone standing on the plane is non-inertial. This is because the rotation of the plane in a circle is a centripital accelleration and the reason the people in the plane rotate with the plane is because the seats they are strapped into apply a real force to keep everyone with the movement of the plane.

If you threw a ball up in the air on a plane while it was turning it would move in the direction opposite of the rotation of the plane. To an observer on the plane the ball would experience an accelerction with no real forces acting on it which disobeys newtons laws. Therefore this is a non-inertial frame of reference.

c) $\omega = \text{const} \rightarrow \alpha = 0 \quad r = \text{const}$

$$\bar{a}_p = \bar{a}_0 + \bar{a}_{rel} + \cancel{\dot{\bar{\omega}} \times \bar{r}_{rel}} + \bar{\omega} \times (\bar{\omega} \times \bar{r}_{rel}) + \cancel{2\bar{\omega} \times \bar{v}_{rel}}$$



Taking the earth to be the inertial reference frame in this case. The origin is not moving so $a_0 = 0$, $\omega = \text{const}$ so $\alpha = 0$, this equation is used for a rotating coordinate axis with rotation speeds and accelerations ω and α respectively. In this situation because $r_{rel} = \text{Const}$, $v_{rel} = 0$, since the objects position does not change relative to point O.

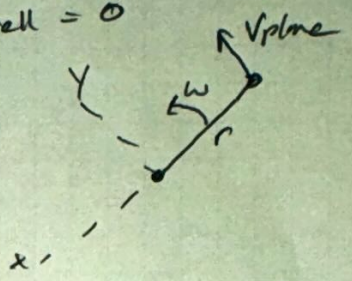
C) Cont

Similarly, because V_{rel} is constant $a_{rel} = 0$

$\bar{a}_{plane} = \bar{\omega} \times (\bar{\omega} \times \bar{r}_{rel})$

for cartesian clockwise rotation

$\omega > 0$ (+ a direction) $\rightarrow$ r in the $-\hat{i}$

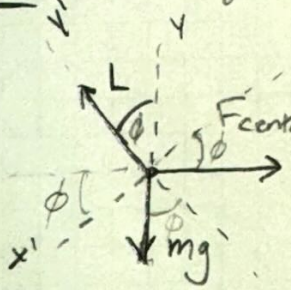


Therefore the people in the plane experience the Centripital acceleration of

$$\boxed{\bar{a}_{plane} = \omega^2 r \hat{i} \rightarrow \|a_{plane}\| = \frac{V_{plane}^2}{r} = a_c}$$

In the perspective of the observers, on the plane a "Centrifugal" force is trying to fling them out equivalent to the their mass times the Centripital acceleration of the plane. This is a fictitious force, but it must be accounted for because the frame is non-inertial

Passenger $\rightarrow F_{centrifugal} = \frac{mV^2}{r} = ma_c$



In order for the passengers to experience no sideways forces, the sum of forces in the x' direction on the passenger must be zero.

$$\rightarrow \Sigma F_{x'} = mg\sin\phi - \frac{m V_{plane}^2}{r}\cos\phi = 0$$

$$mg\sin\phi = \frac{mV_{plane}^2}{r}\cos\phi$$

$$g\tan\phi = \frac{V_{plane}^2}{r}$$

where y' and x' are the rolled axis of the plane. Lift will always be along the y' direction.

$$\phi = \tan^{-1}\left(\frac{V_{plane}^2}{rg}\right)$$

Note: This is in the observer on earths inertial frame of reference, because the passenger is in a non inertial frame, it experiences a centrifugal force. From someone observing this on the ground there is no centrifugal force, just the centripital acceleration.

c) cont.

This relates to orbits because in orbital mechanics we consider the inside of an orbiting object to be inertial, while in this case of the plane they are not entirely inertial, the people inside do not feel the Centrifugal force away from the center. In an orbit an astronaught does not feel like its turning toward the focus, but they are in the global frame. But because gravity is the only force, in space. The whole frame inside the satellite is inertial.

d)

$$\bar{a}_{person} = \bar{a}_o + \alpha_{rel} + \alpha \times r_{rel} + \omega \times (\omega \times r_{rel}) + 2\omega \times V_{rel}$$

$$\boxed{a_{person} = \omega^2 r_{pas}\hat{i} + 2\omega V_{pas}\hat{i}}, \rightarrow \text{both terms in Centripital direction}$$

In this case, relative to the center of rotation the passenger is moving at a faster tangential velocity than the plane. This means that the person on the plane requires a greater Centripid acceleration than the plane to stay in a perfect circle. This corresponds to the person accelerating outward relative to the plane. $\|a_{plane}\| < \|a_{person}\|$

In orbital mechanics this relates to a $+\Delta V$. In that case, speeding up means you need more acceleration towards the planet to maintain your orbit. Because the planet can't "just" give more gravity, your orbit gets bigger to a range that balances your new speed to keep you in a stable orbit.

e)

$$\bar{a}_{person} = \bar{a}_o + \alpha_{rel} + \alpha \times r_{rel} + \omega \times (\omega \times r_{rel}) + 2\omega \times V_{rel}$$

$$\boxed{a_{person} = \omega^2 r_{pas}\hat{i} - 2\omega V_{pas}\hat{i}}$$

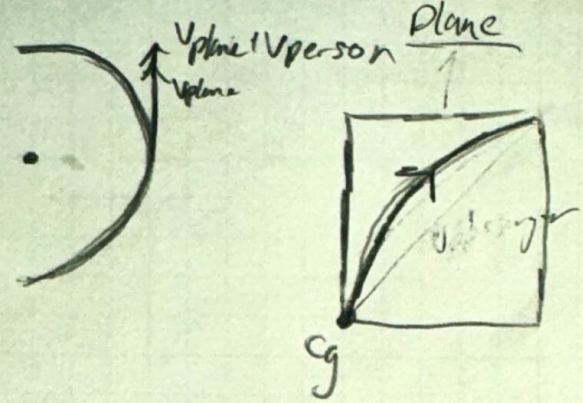
Inversely in this case, the relative Centripital acceleration to keep the person in a circle is less than that of the plane to stay in a perfect circle. This corresponds to the person accelerating inward relative to the plane. $\|a_{plane}\| > \|a_{person}\|$

In orbital mechanics this relates to a $-\Delta V$. Because the magnitude of acceleration required to keep the object in the current orbit is greater than it needs to be, you get pulled closer to the planet, in other words your orbit shrinks and you speed up until the orbit is stable.

Note: Look at the attached diagrams

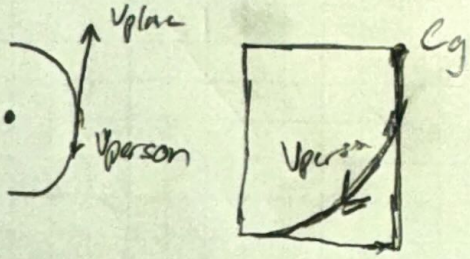
diagrams for parts D and E

D



Because the passenger is moving faster than the plane, and the passenger starts at the cg. relative to the cg of the plane the passenger moves forward radially at a constant velocity. but also accelerates outward radially with the magnitude $2\omega V_{passenger}$.

E



Because the person is moving slower than the plane, the passenger both has a velocity relative to the plane, and an acceleration radially inward of $2\omega V_{passenger}$

2. Two Satellites deployed to Orbit

As you may know, a lot of CubeSat satellites are deployed from common launchers, with many satellites being deployed shortly after one another. Lets imagine a case of a couple of CubeSats deployed from the ISS (450km altitude, circular orbit) at practically the same time.

(a) The CubeSat deployment happens from the arm of the ISS, which is only a few meters long. Given this, what is the approximate deployment altitude and radius?

I'll take "a few meters" to be 4 m or so. Therefore the deployment altitude is 450.004 km, assuming the Earth to be a perfect sphere, the radius of the Earth is $6378[Km]$. Therefore the radius of the cubesat is 6828.004 Km (alt + radius). However, because we can only represent 5 significant digits, the presented radius and altitude are 6828.0 Km and 450.00 Km respectively. With numbers of this magnitude, a few meters does not mean much. We can approximate by saying the cubsat and ISS share the same altitude and radius here.

(b) Both CubeSat's must have a "smaller orbit" than ISS, for safety. In which direction must the deployment mechanism push the CubeSat's?

The mechanism must throw the cubesat in the direction opposite of the motion of the ISS. In other words, the mechanism must apply a negative ΔV on the cubesat in order to put it into a smaller orbit than that of the ISS. Throwing it forward would give the cubesat a positive ΔV, and would therefore put the cubesat on a bigger orbit than the ISS.

(c) Will the CubeSat's and ISS ever meet again? Explain the assumptions (or non-assumptions) for your answer.

Assuming that the cubesat and ISS stay on the exact same orbits they have after the cubesat was launched, and these orbits are not disturbed in any way. Meaning asteroids or the cubesat/ISS applying a ΔV to themselves, etc. The cubesat and ISS will eventually meet again. Because the cubesat's orbital period of the ISS and Cubesat are now different, and it now shares a point of its orbit with the ISS. While it may take many orbits of both the ISS and the cubesat, eventually they would cross paths again at some point.

(d) The desired orbit for Sat 1 has a maximum separation (in ideal orbital dynamics conditions) from the ISS's circular orbit of 50km. Find the parameters for the orbit of Sat 1:

(a) Use your existing functions in Matlab or Python to get all the other parameters. Submit a commented copy of your code.

The maximum separation of these two orbits happens at periapsis of the cubesats orbit (The cubsat was thrown at apoapsis). Therefore $r_a = 6828[Km]$ and $r_p = 6778[Km]$. With these numbers we can determine all of the characteristics of the cubesats orbit. Here is the function used to get the state, the code used to find a and e, and the output.

```
#Returns a dictionary with all useful values of a defined orbit
#if km=True, pass in km values for a and mu
#else km=False, pass in si units
def orbital_state(a,e,mu, km=False) -> dict:
```

```
  r_p = a * (1-e) #rp = a(1-e)
  r_a = a * (1+e) #ra = a(1+e)
  if not km:
    T = (2*math.pi/math.sqrt(mu))*(a**(3/2)) # T = 2(pi) * sqrt(a^3/mu)
  else:
    #need Si units
    T = (2*math.pi/math.sqrt(mu*(10**9)))*((a*1000)**(3/2)) # T = 2(pi) * sqrt(a^3/mu
  spec_e = -mu/(2*a) #-mu/(2*a)
  h = math.sqrt(a*mu*(1-math.pow(e,2))) # h = sqrt(mu*a*(1-e^2))
  b = a * math.sqrt(1-(math.pow(e,2)))
  v_p = h/r_p # vp = h/rp
  v_a = h/r_a # va = h/va

  if not km:
    state = {
      'e'             : e,
      'a (m)'         : a,
      'r_p (m)'       : r_p,
      'r_a (m)'       : r_a,
      'v_p (m/s)'     : v_p,
      'v_a (m/s)'     : v_a,
      'b (m)'         : b,
      'h (m^2/s)'     : h,
      'T (s)'         : T,
      'spec_e (J/kg)' : spec_e
    }
  else:
    state = {
      'e'              : e,
      'a (km)'         : a,
      'r_p (km)'       : r_p,
      'r_a (km)'       : r_a,
      'v_p (km/s)'     : v_p,
      'v_a (km/s)'     : v_a,
      'b (km)'         : b,
      'h (km^2/s)'     : h,
      'T (s)'          : T,
      'spec_e (MJ/kg)' : spec_e
    }

  return state

radius_ISS = orbital_equations_of_motion.altitude_to_orbital_radius_earth(
    alt_ISS, km=True)

ra_cube = radius_ISS
rp_cube = radius_ISS - 50 #given
```

```
e = (ra_cube-rp_cube)/(ra_cube+rp_cube) # Curts 2.83
a = rp_cube/(1-e) # Curtis 2.73

cubesat_state = orbital_equations_of_motion.orbital_state(
    a, e, planetary_data.MU_EARTH_KM, km=True)
orbital_equations_of_motion.print_state(cubesat_state, label='Cubesat Orbit')
```

OUTPUT:

```
============ State of Cubesat Orbit ============
e : 0.0036748
a (km) : 6803
r_p (km) : 6778
r_a (km) : 6828
v_p (km/s) : 7.6839
v_a (km/s) : 7.6276
b (km) : 6803
h (km^2/s) : 52081
T (s) : 5583.4
spec_e (MJ/kg) : -29.305
```

(b) What is the ΔV (magnitude) necessary to enter this orbit?

To find the needed ΔV we need to find the difference in velocity of the cubesat and ISS immediately after the cubesat was launched. The cubesat was launched at apoapsis of its orbit, therefore its velocity at this point is $v_a = 7.6267$. Then using the following code we can find ΔV.

```
v_ISS = math.sqrt(planetary_data.MU_EARTH_KM/radius_ISS) # Curis 2.61
delta_v = cubesat_state['v_a (km/s)'] - v_ISS #Final - initial
print(f'\nDelta V in km/s {delta_v:5g}')
```

OUTPUT:

```
Delta V in km/s -0.0140539
```

(c) Here are the given plots of the orbits, they are very similar, but not the same. The zoomed in graph shows periapsis of the cubesats orbit (This goes to show the scale of this orbit).

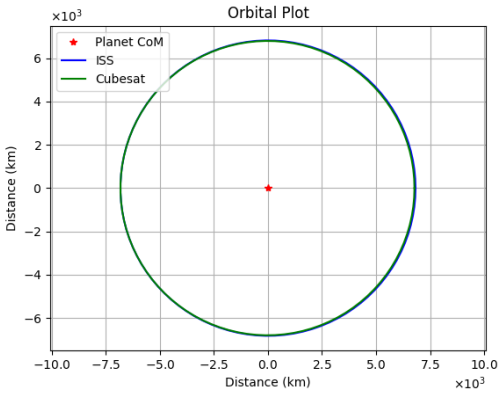


Figure 1: Plot of the two orbits

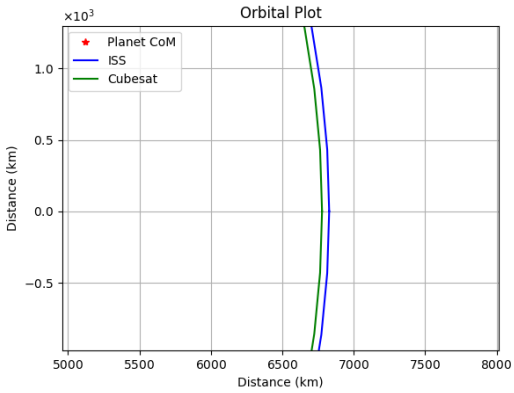


Figure 2: Zoomed in plot at periapsis

This is the code I used to make these plots

```
mu = planetary_data.MU_EARTH_KM

h_ISS = v_ISS*radius_ISS
ax = orbital_equations_of_motion.orbit_plot(h_ISS, mu, 0, color='b', label='ISS')
orbital_equations_of_motion.orbit_plot(
  cubesat_state['h (km^2/s)'], mu, e, ax=ax, color='g', label='Cubesat')

plt.show()

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)))) #Keplers 2nd

  #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)
  formatter.set_powerlimits((-1, 1))
```

```
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
```

(e) The objective for Sat 2 is for it to "meet" the ISS every 18 orbits:

(a) What is the difference in the periods between the ISS and Sat 2?

We can use the phasing manuever equation to determine the difference in period for a phase manuever of 2π every 18 orbits. Here is the code I used for this.

```
T_ISS = 2 * math.pi * math.sqrt((radius_ISS**3)/mu) #2.83
#one orbit is 2pi radians, in this equation that divides out to be this equation
#since we are doing a "phasing maneuver" of one orbit in 18 ISS orbits
T_sat2 = T_ISS - (T_ISS/18) #Phasing maneuver equation
T_dif = T_sat2 - T_ISS
print(f' signed difference in periods is {T_dif:3g} s, or {(T_dif/60):5g} min')

OUTPUT:

     signed difference in periods is -311.898 s, or -5.19829 min
```

This could also be done by putting sat2 on a larger orbit than the ISS, but that is dangerous because it would require an initial positive ΔV. Because of that I opted to use a negative ΔV similar to sat1, which puts sat2 on an orbit with a smaller period than that of the ISS, hence the negative difference in period.

(b) What is the ΔV (magnitude) for Sat 2?

In order to find the required ΔV to put Sat 2 on this orbit, we need to find the magnitude of v_a of sat 2's orbit.

```
mu = planetary_data.MU_EARTH_KM
a_sat2 = (((T_sat2 / (2 * math.pi)) ** 2) * mu) ** (1/3) # 2.83

#Because this orbit shares an apoapsis with the ISS' orbit,
#know that r_a sat2 = r_iss
r_a_sat2 = radius_ISS
e_sat2 = (r_a_sat2/a_sat2) - 1 # ra = a(1+e)
sat2_state = orbital_equations_of_motion.orbital_state(
    a_sat2, e_sat2, mu, km=True)

delta_v_sat2 = sat2_state['v_a (km/s)'] - v_ISS
print(f'Signed Delta V for sat 2 to be in this orbit {delta_v_sat2:5g}')
```

```
OUTPUT:

    Signed Delta V for sat 2 to be in this orbit -0.149875 km/s
```

(f) For this final 3 satellite system: after the ISS completes one full rotation, what will be the difference in true anomaly between: i. ISS and Sat 1 | ii. ISS and Sat 2 | iii. Sat 1 and Sat 2

For this question I used a newton rhapson function that I wrote for homework 4 and another function to find the difference, because the satellites are on a smaller orbit than that of the ISS, they will be ahead of the ISS after one ISS orbit, this is reflected in the presented differences.

```
#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
```

Here is the code I used to extract the differences in angle, the states are orbital state dictionaries returned by my orbital state function

```
#Because all sats start at apoapsis,
#we need to add half of the respective satellite period on
#top of the period of the ISS to account for starting from apo
#finds a signed difference of state2 - state1
def find_difference_in_theta(state1, state2, given_time):
  state1_e = state1['e']
  state1_T = state1['T (s)']
  state2_e = state2['e']
  state2_T = state2['T (s)']

  #find true anom using newton rhapson
  state1_theta = orbital_equations_of_motion.theta_from_t(
    given_time + (state1_T/2), state1_e, state1_T) #account for starting at apo
  state2_theta = orbital_equations_of_motion.theta_from_t(
    given_time + (state2_T/2), state2_e, state2_T)
```

```
  return state2_theta - state1_theta

mu = planetary_data.MU_EARTH_KM
ISS = orbital_equations_of_motion.orbital_state(radius_ISS, 0, mu, km=True)
sat1 = cubesat_state #defined in first part of question 2
sat2 = sat2_state #defined in previous question
T_ISS #defined in previous questions

combinations = [(('ISS','Sat1'),ISS, sat1),
                (('ISS','Sat2'), ISS, sat2),
                (('Sat1','Sat2'),sat1,sat2)]

for combination in combinations:
  names, state1, state2 = combination
  name1, name2 = names
  print(f'The difference in true anom from {name1} to {name2} is '
        f'{find_difference_in_theta(state1, state2, T_ISS):5g} deg')

OUTPUT:

    The difference in true anom from ISS to Sat1 is 1.8741 deg
    The difference in true anom from ISS to Sat2 is 19.314 deg
    The difference in true anom from Sat1 to Sat2 is 17.440 deg
```

These results make sense, Sat1 was *barely* on a different orbit from that of the ISS, reflected in the very small magnitude ΔV, where as Sat2, had a much greater magnitude ΔV than Sat1 did. This difference is reflected in the fact that after 1 ISS orbit, Sat2 has moved further along than Sat1, which has moved further along than the ISS.

3. Observing the Poles

In order to observe the North and South poles we cannot use a "Geostationary orbit", because the Earth's rotation is perpendicular to the rotation of the satellite. The next best thing is to use High Eccentricity Orbits. Let's imagine that NASA wants satellites that observe the North or South poles. Assume they can be placed exactly perpendicular to the equator.

(a) Find as much information as can you find about the orbit if NASA desires the satellites to have a 48 hour period. Submit commented code that you use (you are welcome to reuse any previous code used for homework).

Because we only know T, we can only find the semi-major axis and the specific energy of this orbit. Here is the code I used to find this

```
#given
mu = planetary_data.MU_EARTH_KM

#because we only know T, we can only find a and epsilon
T = 48 * 3600 #hours to seconds
a = (((T / (2 * math.pi)) ** 2) * mu) ** (1/3) #2.83
epsilon = - mu/(2*a)

print(f' a: {a:5g} km, epsilon: {epsilon:5g} MJ/kg')

OUTPUT:

    a: 67060.4 km, epsilon: -2.97286 MJ/kg
```

(b) In addition NASA tells you the maximum altitude for the sensor to work correctly is 90,000km. Calculate all the parameters of the orbit (using any previous code you have, submit with comments):

Because we are given the maximum *altitude* to be 90,000 km, we can determine that this must be the altitude at apoapsis of this orbit, with this in mind we can determine r_a and the rest of the orbits parameters. Also I realize that I have not properly introduced the `print_state` function I have been using, or the `altitude_to_orbital_radius` function. Those are both here as well.

```
#Returns a dictionary with all useful values of a defined orbit
#expects si units
#if km=True, pass in km values for a and mu
#else km=False, pass in si units
def orbital_state(a,e,mu, km=False) -> dict:
  r_p = a * (1-e) #rp = a(1-e)
  r_a = a * (1+e) #ra = a(1+e)
  if not km:
    T = (2*math.pi/math.sqrt(mu))*(a**(3/2)) # T = 2(pi) * sqrt(a^3/mu)
```

```
  else:
    #need Si units
    T = (2*math.pi/math.sqrt(mu*(10**9)))*((a*1000)**(3/2))
  spec_e = -mu/(2*a) #-mu/(2*a)
  h = math.sqrt(a*mu*(1-math.pow(e,2))) # h = sqrt(mu*a*(1-e^2))
  b = a * math.sqrt(1-(math.pow(e,2)))
  v_p = h/r_p # vp = h/rp
  v_a = h/r_a # va = h/va

  if not km:
    state = {
      'e'             : e,
      'a (m)'         : a,
      'r_p (m)'       : r_p,
      'r_a (m)'       : r_a,
      'v_p (m/s)'     : v_p,
      'v_a (m/s)'     : v_a,
      'b (m)'         : b,
      'h (m^2/s)'     : h,
      'T (s)'         : T,
      'spec_e (J/kg)' : spec_e
    }
  else:
    state = {
      'e'              : e,
      'a (km)'         : a,
      'r_p (km)'       : r_p,
      'r_a (km)'       : r_a,
      'v_p (km/s)'     : v_p,
      'v_a (km/s)'     : v_a,
      'b (km)'         : b,
      'h (km^2/s)'     : h,
      'T (s)'          : T,
      'spec_e (MJ/kg)' : spec_e
    }

  return state

def print_state(orbital_state, label=None):
  if label:
    print(f'\n============ State of {label} ============')
  else:
    print('\n============ State ============')

  for state in orbital_state:
    print(f'{state} : {orbital_state[state]:.5g}')

#returns approximately the orbital radius of an altitude
```

```
#alt should be in m from the earths surface
def altitude_to_orbital_radius_earth(alt, km=False) -> float:
  if km:
    return planetary_data.RADIUS_EARTH/1000 + alt
  else:
    return planetary_data.RADIUS_EARTH + alt
```

Here is the code I used to extract the state

```
#given
alt_apo = 90000 #km
ra = orbital_equations_of_motion.altitude_to_orbital_radius_earth(alt_apo, km=True)

e = ra/a - 1 # ra = a(1+e)

state = orbital_equations_of_motion.orbital_state(a, e, mu, km=True)
orbital_equations_of_motion.print_state(state, label='Nasa Sat')
```

```
OUTPUT:

    ============ State of Nasa Sat ============
    e : 0.43718
    a (km) : 67060
    r_p (km) : 37743
    r_a (km) : 96378
    v_p (km/s) : 3.8965
    v_a (km/s) : 1.5259
    b (km) : 60312
    h (km^2/s) : 1.4706e+05
    T (s) : 1.728e+05
    spec_e (MJ/kg) : -2.9729
```

(c) For how much time will a satellite be on its "side" of the equator? (i.e., on the North or the South side)

Assuming that the Earth is oriented the same way it is in the picture below, where the south pole faces periapsis and the north pole faces apoapsis. Because we know everything about our orbit, and the satellite leaves the north pole side at a true anomaly of 90 deg, and the orbit is symmetrical, the time spent on the south pole side is the the same as $2 \times t(\theta = 90^\circ)$. Moreover, because we know the time spent infront of the south pole, we know that the time spent infront of the north pole is the period minus the time spent infront of the south pole. Here is the function I used to the times around each pole.

```
#expects theta in degrees
#returns t in seconds
def t_from_theta(theta_deg, T, e):
  theta = theta_deg * math.pi / 180
  E = 2*math.atan((math.sqrt((1-e)/(1+e)))*math.tan(theta/2)) # Curtis 3.13b
```

```
  Me = E - (e * math.sin(E)) # Curtis 3.14
  t = (Me/(2*math.pi))* T
  return t
```

I used this code to find the time around each pole

```
T = state['T (s)']
theta = 90
t_to_90 = orbital_equations_of_motion.t_from_theta(theta, T, e)
t_south_pole = 2 * t_to_90
t_north_pole = T - t_south_pole
print(f'Time spent around North pole in hours {(t_north_pole/3600):.5g}')
print(f'Time spent around South pole in hours {(t_south_pole/3600):.5g}')
```

```
OUTPUT:

    Time spent around North pole in hours 36.921
    Time spent around South pole in hours 11.079
```

(d) The sensors ended up not being so good, and can only be used when the satellite is within $\pm 42^\circ$ of the pole: What is the "useful" time for this satellite for the given location? Show your work (code recommended, commented).

By subtracting the time it takes to go to a true anomaly of $(180 - 42)^\circ$ from the time it takes to go to a true anomaly of 180°, and doubling that value, we are left with the time that the satellite spent within the given cone of viewing the south pole. Here is what that looks like in code

```
#Does not work for times spent around periapsis because of the wrapping of angles
#Finds the time spent between in an angular cone around apoapsis, expects range to be
#a singular number representing +/- a given angle from 180
def time_spent_in_cone(range, e, T):
  t_to_range = t_from_theta(180-range,T,e)
  return (T/2 - t_to_range) * 2
```

Here is the code I used to use this function

```
T, e #from previous problem

range = 42

t_cone = orbital_equations_of_motion.time_spent_in_cone(range, e, T)
print(f'Time spent with sensors seeing the north pole {(t_cone/3600):.5g} hours')
```

```
OUTPUT:

    Time spent with sensors seeing the north pole 22.768 hours
```

(e) In order to get into the highly elliptical orbit the satellite is first placed into a circular Earth orbit at 38000km altitude.

(a) Sketch the 2 options for Hohmann Transfer orbits using the diagram below (or an equivalent version of your own)

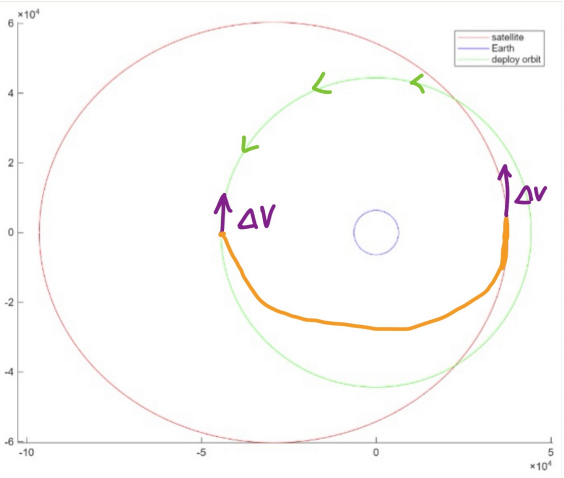


Figure 3: Ending at Peri of the desired orbit

Figure 4: Ending at Apo of the desired orbit

(b) What are the origin & destination of the 2 Hohmann Transfer orbits?

For the Hohmann transfer on the left, the origin is at apoapsis of the deploy orbit, and the destination is the periapsis of the satellite orbit. For the Hohmann transfer on the right, the origin of the transfer orbit is at periapsis of the deploy orbit, and the destination is apoapsis of the satellite orbit.

(c) Determine the orbital parameters (same list as part b) of the 2 Hohmann Transfer orbits options.

I wrote a long function that computes the transfer orbit and the total magnitude of the ΔV required to do the transfer. That function is here.

```
#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
```

Using this function, I found the state of both the transfer orbits using the follow code.

```
#satellite state defined earlier
state

#circular deploy orbit
alt_deploy = 38000 #km
r_deploy = orbital_equations_of_motion.altitude_to_orbital_radius_earth(alt_deploy)

#for the transfer starting at apoapsis
deploy_state = orbital_equations_of_motion.orbital_state(r_deploy, 0, mu, km=True)

delta_v_apo, transfer_state_apo = orbital_equations_of_motion.hohmann_transfer(
  deploy_state, state, mu, start_peri=False
)

delta_v_peri, transfer_state_peri = orbital_equations_of_motion.hohmann_transfer(
  deploy_state, state, mu, start_peri=True
)

orbital_equations_of_motion.print_state(
    transfer_state_apo, label='starting from apoapsis')
orbital_equations_of_motion.print_state(
    transfer_state_peri, label='starting from periapsis')
```

These are the respective states that were found by this function

```
============ State of starting from apoapsis ============
e : 0.080798
a (km) : 41060
r_p (km) : 37743
r_a (km) : 44378
v_p (km/s) : 3.379
v_a (km/s) : 2.8738
b (km) : 40926
h (km^2/s) : 1.2753e+05
T (s) : 82790
spec_e (MJ/kg) : -4.8553

============ State of starting from periapsis ============
e : 0.36943
a (km) : 70378
r_p (km) : 44378
r_a (km) : 96378
v_p (km/s) : 3.5077
v_a (km/s) : 1.6151
b (km) : 65399
h (km^2/s) : 1.5566e+05
T (s) : 1.8578e+05
spec_e (MJ/kg) : -2.8327
```

(d) Calculate the total Δv required to enter the final orbit via the 2 Hohmann transfer options.
Then using the returned ΔV values from the hohmann transfer function I wrote, we can find the total ΔV magnitude of each transfer to be

```
print(f'Total magnitude Delta V, transfer starting at Apo {delta_v_apo:.5g} km/s')
print(f'Total magnitude Delta V, transfer starting at Peri {delta_v_peri:.5g} km/s')

OUTPUT:

    Total magnitude Delta V, transfer starting at Apo 0.64112 km/s
    Total magnitude Delta V, transfer starting at Peri 0.59948 km/s
```

(e) Recommend a transfer option based on those results.

Because starting from peri requires a smaller ΔV in order to complete the transfer to get into the proper orbit. From a perspective of conserving fuel we should start our hohmann transfer at periapsis of the deploy orbit.