- AAE 532 F24
- Problem Set 7 - Due 26 Oct 2024
- JP Burke
- burke280@purdue.edu

```
# general setup for entire problem set
import numpy as np
from numpy import pi, sqrt, sin, cos, tan, abs, deg2rad, rad2deg, arcsin, arccos, arctan, cross
from numpy.linalg import norm
from scipy.optimize import fsolve
import matplotlib.pyplot as plt

# formatting
width   = 12 # output variable length. used for nice aligning
num_fmt = ' 12.3f'

float_formatter = "{: .5f}".format
np.set_printoptions(formatter={'float_kind':float_formatter})

# general constants
HOUR_TO_SEC = 3600
SEC_TO_HOUR = 1/HOUR_TO_SEC

DAY_TO_SEC = 86400
SEC_TO_DAY = 1/DAY_TO_SEC

# from Table of Constants
EARTH_RADIUS = 6378.1363
EARTH_MU = 398600.4415
```

Problem 1

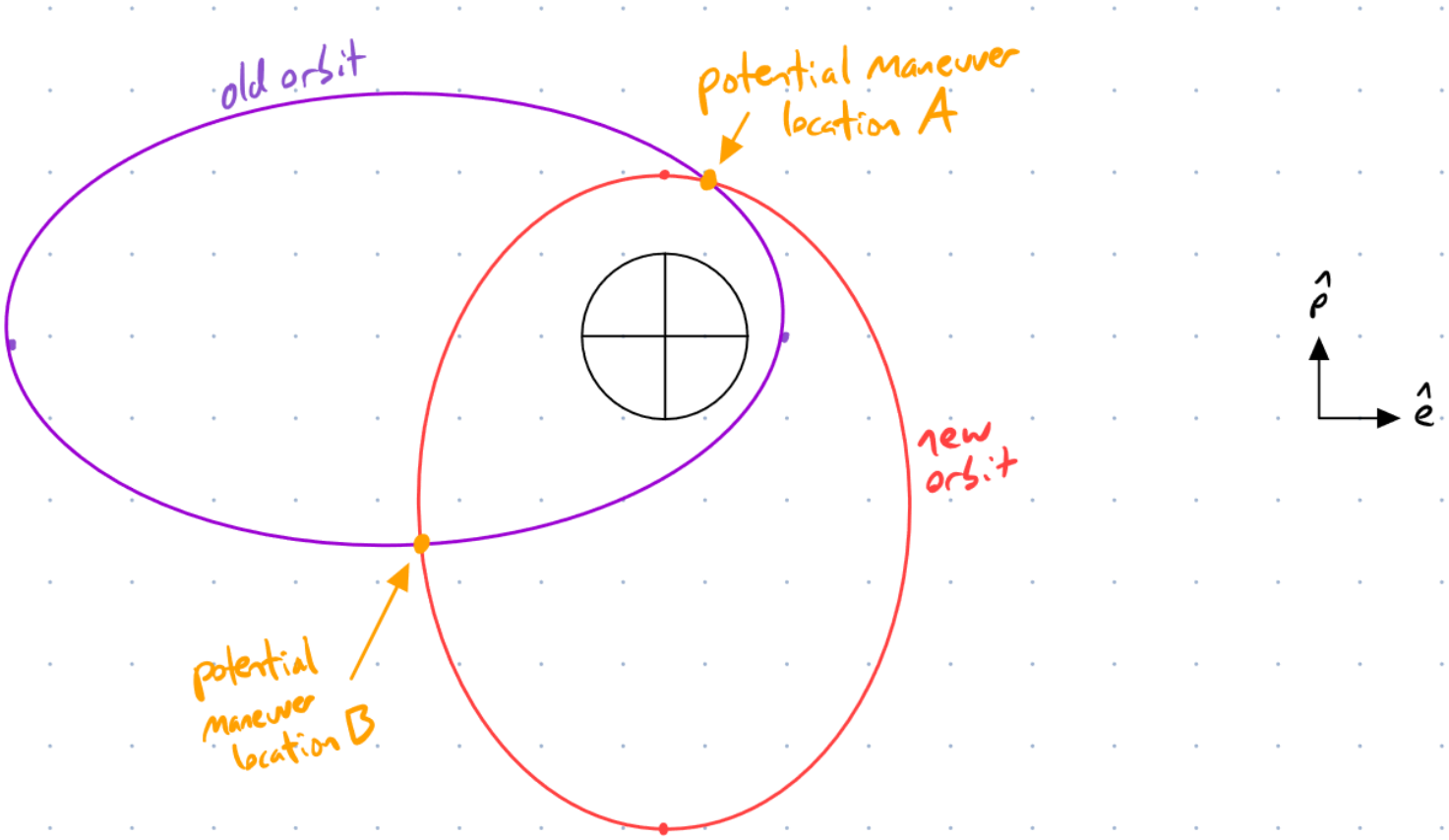
> Assume a two-body model for an Earth orbiter. Currently, the vehicle is in an orbit such that $r_p = 1.5R_\oplus$ and $r_a = 8.0R_\oplus$. A single in-plane maneuver will be used to raise perigee and lower apogee. New values are specified as $r_p = 2.0R_\oplus$ and $r_1 = 6.0R_\oplus$. It is also required that perigee advance by 90° ($\Delta\omega = +90^\circ$).

1.a)

> There are two options for the location to place the maneuver to satisfy the requirements. (Sketch the scenario.)
>
> At what θ^* in the original orbit should the maneuver take place? Be sure to select the location that results in the min $|\Delta\vec{v}|$ and justify the choice. Determine $\vec{r}_1$, $\vec{v}_1^-$, γ_1^- at the maneuver point.

Problem statement: Given a spacecraft in the orbit described above, and assuming a two-body force model, find the θ^* where the specified maneuver should take place. Find the position vector, pre-burn velocity vector, and pre-burn flight path angle at the maneuver point.

First let's draw the scenario:



From this sketch we see two places where the orbits intersect, and thus two places where we can potentially do our maneuver. In order to evaluate and perform our maneuver we'll need to find the true anomaly to perform the burn at. Let's start by defining our orbits.

The problem gives us the apogee and perigee of both orbits. From that we can find semi-major axis:

$$a = \frac{r_p + r_a}{2}$$

And then eccentricity:

$$e = \frac{r_a - a}{a}$$

And semi-latus rectum:

$$p = r_a(1 - e)$$

We're also going to need to convert a position from $\hat{e}$, $\hat{p}$, $\hat{h}$ to $\hat{x}$, $\hat{y}$, $\hat{z}$ which we'll do with:

$$r = \frac{p}{1 + e\cos(\theta^*)}$$

$$\vec{r}_{old} = r\left[\cos(\theta^*)\hat{x} + \sin(\theta^*)\hat{y}\right]$$

To account for the 90° argument of perigee of the new orbit, we'll use:

$$\vec{r}_{new} = r\left[\cos(\theta^*)\hat{x} + \sin\left(\theta^* + \frac{\pi}{2}\right)\hat{y}\right]$$

In order to find the intersection points we will use a numeric solver. For each iteration of the solver we'll do the following:

- Pass in a potential true anomaly for both the new orbit and old orbit.
- Calculate the radius of the spacecraft at that true anomaly for each orbit.

- For both the new and old orbits, calculate the inertial position of a spacecraft at the given true anomaly for its orbit. (Taking care to note the shift in argument of perigee)
- When $x_{old} = x_{new}$ and $y_{old} = y_{new}$ we'll have the position and true anomaly of a maneuver points.

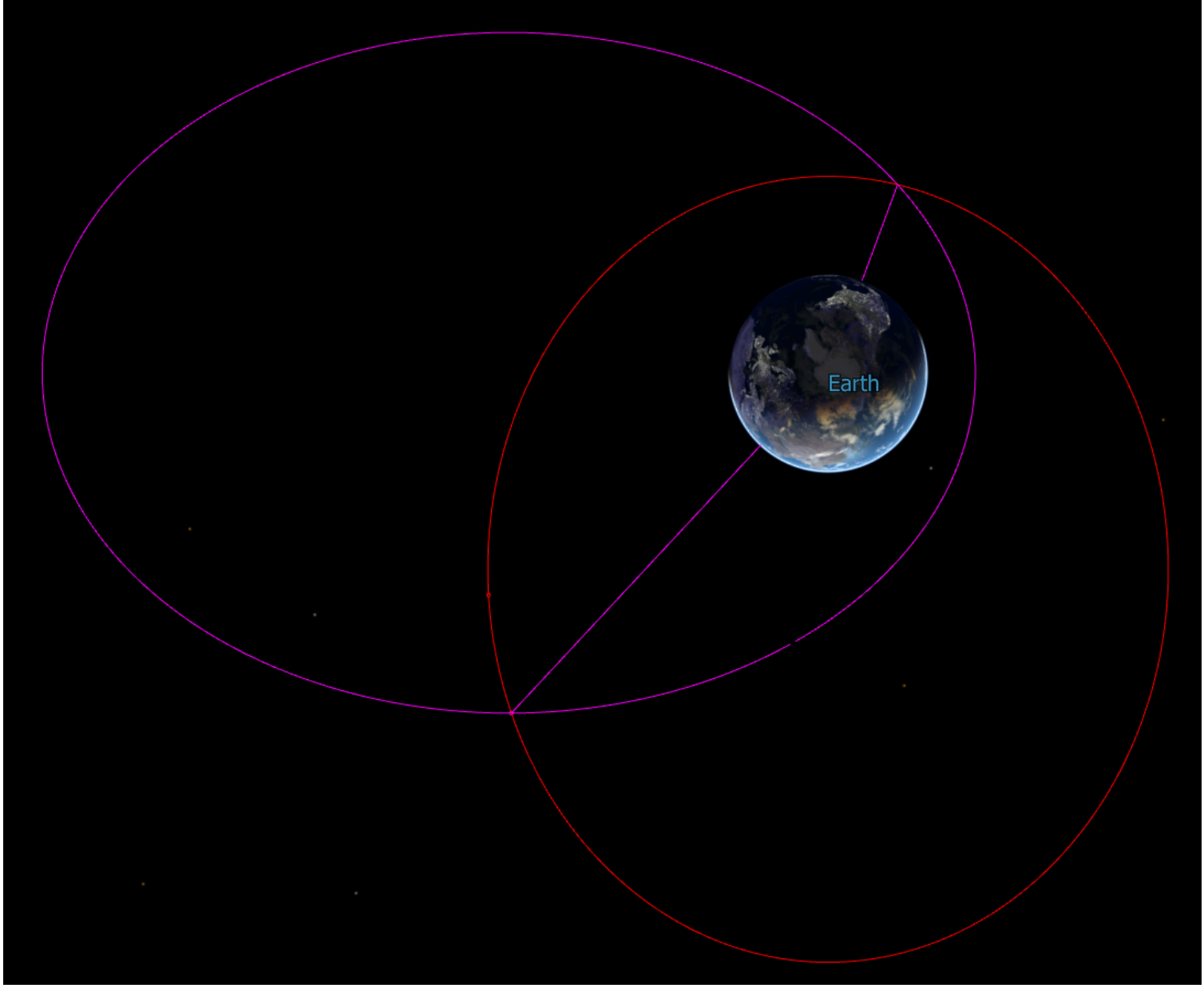
In order to find both intersection points, we'll simply eyeball the plot of the orbits in order to find a reasonable starting seed, and count on the solver to converge on the nearby intersection.

Alright, let's do it:

```
pe_old = 1.5 * EARTH_RADIUS
ap_old = 8.0 * EARTH_RADIUS
a_old  = (pe_old + ap_old) / 2
e_old  = (ap_old - a_old) / a_old
p_old  = ap_old * (1 - e_old)

pe_new = 2.0 * EARTH_RADIUS
ap_new = 6.0 * EARTH_RADIUS
a_new  = (pe_new + ap_new) / 2
e_new  = (ap_new - a_new) / a_new
p_new  = pe_new * (1 + e_new)

def orbit_intersection(variables):
    old_ta, new_ta = variables

    old_radius = p_old / (1 + e_old * cos(old_ta))
    old_x = old_radius * cos(old_ta)
    old_y = old_radius * sin(old_ta)

    new_radius = p_new / (1 + e_new * cos(new_ta))
    new_x = new_radius * cos(new_ta + pi/2)
    new_y = new_radius * sin(new_ta + pi/2)

    return [old_x - new_x, old_y - new_y]

root = fsolve(
    orbit_intersection,
    [deg2rad(90), deg2rad(0)]
)
intersection_a_old_ta = root[0]
intersection_a_new_ta = root[1]
print(f'intersection A true anomaly (old orbit): {rad2deg(intersection_a_old_ta):{num_fmt}} deg')
print(f'intersection A true anomaly (new orbit): {rad2deg(intersection_a_new_ta):{num_fmt}} deg')

root = fsolve(orbit_intersection, [deg2rad(270), deg2rad(135)])
intersection_b_old_ta = root[0]
intersection_b_new_ta = root[1]
print(f'intersection B true anomaly (old orbit): {rad2deg(intersection_b_old_ta):{num_fmt}} deg')
print(f'intersection B true anomaly (new orbit): {rad2deg(intersection_b_new_ta):{num_fmt}} deg')
```

```
intersection A true anomaly (old orbit):       69.727 deg
intersection A true anomaly (new orbit):      -20.273 deg
intersection B true anomaly (old orbit):      227.058 deg
intersection B true anomaly (new orbit):      137.058 deg
```

OK great. To quickly verify these numbers, I wrote a FreeFlyer script that established the orbits, stepped the old spacecraft to the true anomalies we found (69.727° and 227.058°) and made a line to indicate their position. I'm not including the script since it's just a bonus check and I don't want to take up the space, but I'm happy to provide it if graders prefer:



Looks good! Now let's evaluate the suitability of each location.

First, let's calculate the velocities at both points on both orbits.

To do that, we first find $\mathcal{E}$ with:

$$\mathcal{E} = -\frac{\mu}{2a}$$

And then v^- and v^+ with:

$$v^- = \sqrt{\frac{2\mu}{r} - \frac{\mu}{a_{old}}}$$

$$v^+ = \sqrt{\frac{2\mu}{r} - \frac{\mu}{a_{new}}}$$

To calculate $\Delta\gamma$

$$\gamma^- = cos^{-1}\left(\frac{\sqrt{\mu p_{old}}}{rv^-}\right)$$

$$\gamma^+ = cos^{-1}\left(\frac{\sqrt{\mu p_{new}}}{rv^+}\right)$$

$$\Delta\gamma = \gamma^+ - \gamma^-$$

To calculate Δv:

$$\Delta v = \sqrt{v^{2^+} + v^{2^-} - 2v^+v^- \cos\Delta\gamma}$$

We'll just have to be careful to keep track of which orbit we're talking about:

```
# let's do the radii first since they won't change between orbits
r_A = p_old / (1 + e_old * cos(intersection_a_old_ta))
r_B = p_old / (1 + e_old * cos(intersection_b_old_ta))

# actually, let's be a little paranoid and double check that they didn't change
assert abs(r_A - p_new / (1 + e_new * cos(intersection_a_new_ta))) < 1e-6
assert abs(r_B - p_new / (1 + e_new * cos(intersection_b_new_ta))) < 1e-6

# now let's find the old and new energies since they won't change
# between candidate maneuver points
old_energy = -EARTH_MU / (2*a_old)
new_energy = -EARTH_MU / (2*a_new)

# now we can find the velocities for old and new orbits at both
# candidate maneuver points
v_old_A = sqrt(2*EARTH_MU/r_A - EARTH_MU/a_old)
v_old_B = sqrt(2*EARTH_MU/r_B - EARTH_MU/a_old)

v_new_A = sqrt(2*EARTH_MU/r_A - EARTH_MU/a_new)
v_new_B = sqrt(2*EARTH_MU/r_B - EARTH_MU/a_new)


fpa_old_A   =  arccos(sqrt(EARTH_MU * p_old) / (r_A*v_old_A)) # ascending
fpa_new_A   = -arccos(sqrt(EARTH_MU * p_new) / (r_A*v_new_A)) # descending
delta_fpa_A = fpa_new_A - fpa_old_A

fpa_old_B   = -arccos(sqrt(EARTH_MU * p_old) / (r_B*v_old_B)) # descending
fpa_new_B   =  arccos(sqrt(EARTH_MU * p_new) / (r_B*v_new_B)) # ascending
delta_fpa_B = fpa_new_B - fpa_old_B

delta_v_A = sqrt(v_new_A**2 + v_old_A**2 - 2 * v_new_A * v_old_A * cos(delta_fpa_A))
delta_v_B = sqrt(v_new_B**2 + v_old_B**2 - 2 * v_new_B * v_old_B * cos(delta_fpa_B))

print(f'radius at Point A:  {r_A:{num_fmt}} km, {r_A/EARTH_RADIUS:{num_fmt}} Earth radii')
print(f'v_old at Point A:   {v_old_A:{num_fmt}} km/s')
print(f'delta-v at Point A: {delta_v_A:{num_fmt}} km/s')
print(f'delta-v at Point B: {delta_v_B:{num_fmt}} km/s')
print(f'fpa_old_A:          {rad2deg(fpa_old_A):{num_fmt}} deg')
```

```
radius at Point A:     13025.234 km,        2.042 Earth radii
v_old at Point A:          6.932 km/s
delta-v at Point A:        4.021 km/s
delta-v at Point B:        4.053 km/s
fpa_old_A:                27.421 deg
```

As a quadrant check we'll note that at Point A θ^* is less than 180° and thus we're ascending, so 27.421° is correct.

Alright, let's gather up what was actually asked for:

θ^*_m	69.727°
$\vec{r}_1^-$	13,025.234 km (2.042 $R_\oplus$)
$\vec{v}_1^-$	4.021 km/s
γ^-	27.421°

We choose to do the maneuver at Point A (θ_m^* = 69.727°) since the required Δv is 4.021 km/s, 32 meters per second less than the 4.053 km/s required at θ_m^* = 227.058°.

1.b)

> Determine the required maneuver ($\Delta\vec{v}$, that is, Δv, α). [Don't forget the vector diagram!]
>
> Compute the $\vec{v}^+$, γ^+?

Problem statement: Find the required maneuver magnitude and direction, as well as its post-burn velocity vector and flight path angle.

We actually ended up doing part of this in 1.a) but let's do it again just to be safe, this time just focusing on Point A, which is the one we've chosen, which we'll call θ_m^*.

$$r = \frac{p_{old}}{1 + e_{old}\cos\theta_m^*}$$

$$v^- = \sqrt{\frac{2\mu}{r_{old}} - \frac{\mu}{a_{old}}}$$

$$v^+ = \sqrt{\frac{2\mu}{r_{new}} - \frac{\mu}{a_{new}}}$$

$$\gamma^- = cos^{-1}\left(\frac{\sqrt{\mu p_{old}}}{rv^-}\right)$$

$$\gamma^+ = cos^{-1}\left(\frac{\sqrt{\mu p_{new}}}{rv^+}\right)$$

$$\Delta\gamma = \gamma^+ - \gamma^-$$

$$\Delta v = \sqrt{v^{+2} + v^{-2} - 2v^+v^-\cos\Delta\gamma}$$

For α we'll use:

$$\frac{\Delta v}{sin\Delta\gamma} = \frac{v_N}{sin\beta} \Rightarrow \beta = sin^{-1}\left(\frac{sin(\Delta\gamma)v_N}{\Delta v}\right)$$

$$\alpha = \pi - \beta$$

Let's find $\vec{v}$ in $\hat{e}$, $\hat{p}$, $\hat{h}$ first using (taking care to use θ_m^* on the new orbit, not the old orbit.:

$$\vec{v} = -\sin\theta_m^*\hat{e} + e_{new}\cos\theta_m^*$$

Let's also find $\vec{v}$ in $\hat{x}$, $\hat{y}$, and $\hat{z}$ using:

$$\theta = \frac{\pi}{2} + \theta_m^*$$

$$\dot{r}_{new} = \sqrt{\frac{\mu}{p_{new}}}e_{new}\sin(\theta_m^*)$$

$$\dot{\theta} = \frac{h_{new}}{r^2}$$

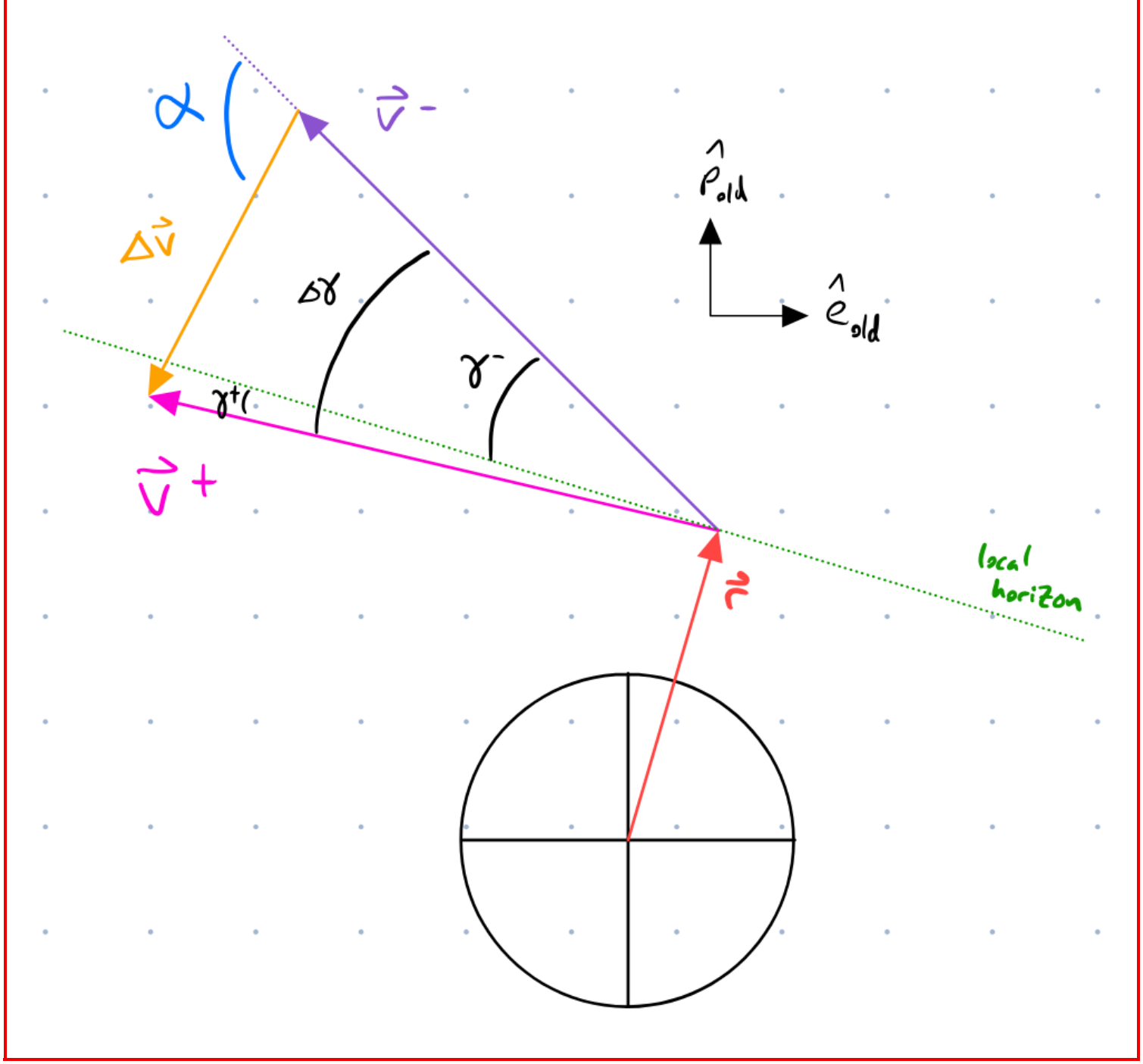
$$\Omega = i = 0$$

$$\vec{v} = [\dot{r}_{new}(c_\Omega c_\theta - s_\Omega c_i s_\theta) + r\dot{\theta}(-c_\Omega s_\theta - s_\Omega c_i c_\theta)]\hat{x} + \quad (1)$$
$$[\dot{r}_{new}(s_\Omega c_\theta + c_\Omega c_i s_\theta) + r\dot{\theta}(-s_\Omega s_\theta + c_\Omega c_i c_\theta)]\hat{y} + \quad (2)$$
$$[\dot{r}_{new}(s_i s_\theta) + r\dot{\theta}(s_i c_\theta)]\hat{z} \quad (3)$$

```
ta_old = intersection_a_old_ta
ta_new = intersection_a_new_ta

r = p_old / (1 + e_old * cos(ta_old))

v_old   = sqrt(2*EARTH_MU/r - EARTH_MU/a_old)
fpa_old = arccos(sqrt(EARTH_MU*p_old) / (r*v_old))

v_new   = sqrt(2*EARTH_MU/r - EARTH_MU/a_new)
fpa_new = -arccos(sqrt(EARTH_MU*p_new) / (r*v_new)) # descending

delta_fpa = fpa_new - fpa_old
delta_v   = sqrt(v_new**2 + v_old**2 - 2 * v_new * v_old * cos(delta_fpa))

h_new = sqrt(EARTH_MU * p_new)

v_new_eph = EARTH_MU/h_new * np.array([ -sin(ta_new), e_new+cos(ta_new), 0 ])

i    = 0
raan = 0
aol  = pi/2 + ta_new
rdot = (EARTH_MU / p_new)**0.5 * (e_new * sin(ta_new))
theta_dot = h_new / r**2

v_new_xyz = np.array([
    rdot * ( cos(raan) * cos(aol)  - sin(raan)*cos(i)*sin(aol)) + r * theta_dot * (-cos(raan)*sin(aol) - sin(raan)*cos(i)*cos(aol)),
    rdot * ( sin(raan) * cos(aol)  + cos(raan)*cos(i)*sin(aol)) + r * theta_dot * (-sin(raan)*sin(aol) + cos(raan)*cos(i)*cos(aol)),
    rdot * ( sin(i)    * sin(aol)) + r * theta_dot  * (sin(i) * cos(aol)),
])

alpha = (pi - -arcsin(sin(delta_fpa) * v_new / delta_v))*-1

print(f'r:         {r:{num_fmt}} km')
print(f'fpa_old:   {rad2deg(fpa_old):{num_fmt}} deg')
print(f'fpa_new:   {rad2deg(fpa_new):{num_fmt}} deg')
print(f'delta_fpa: {rad2deg(delta_fpa):{num_fmt}} deg')
print(f'delta_v:   {delta_v:{num_fmt}} km/s')
print(f'alpha:     {rad2deg(alpha):{num_fmt}} deg')
print(f'v_new_eph: {v_new_eph} km/s')
print(f'v_new_xyz: {v_new_xyz} km/s')
```

```
r:            13025.234 km
fpa_old:         27.421 deg
fpa_new:         -6.726 deg
delta_fpa:      -34.147 deg
delta_v:          4.021 km/s
alpha:         -109.530 deg
v_new_eph: [ 1.58145  6.56351  0.00000] km/s
v_new_xyz: [-6.56351  1.58145  0.00000] km/s
```

Δv	4.021 km/s
α	-109.530°
$\vec{v}^+$	1.58145 $\hat{e}$ + 6.56351 $\hat{p}$ km/s
$\vec{v}^+$	-6.56351 $\hat{x}$ + 1.58145 $\hat{y}$ km/s
γ^+	-6.726°

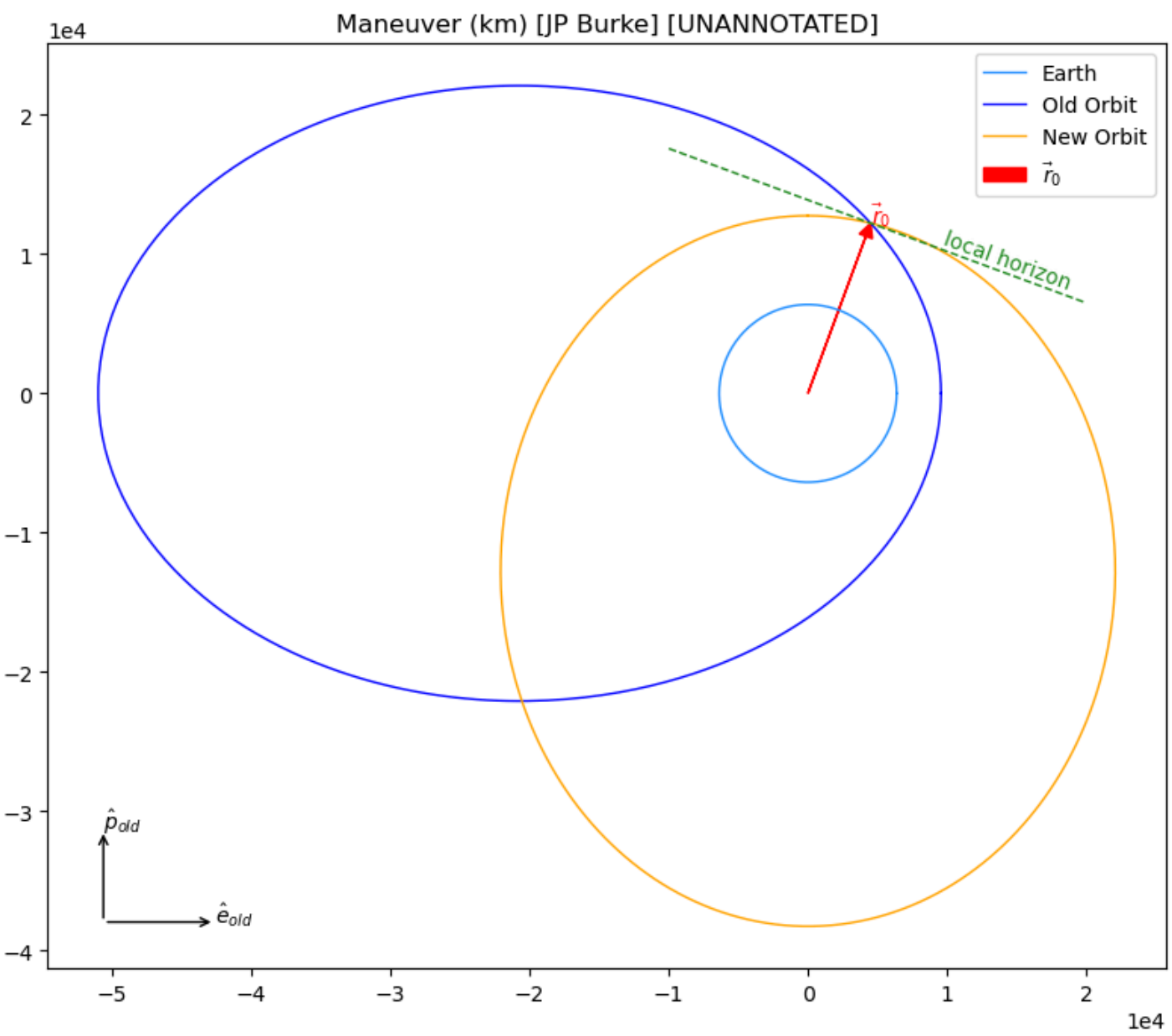

α
$\vec{v}-$
$\hat{p}_{old}$
$\hat{e}_{old}$
$\Delta\vec{v}$
$\delta\gamma$
γ^-
γ^+
$\vec{v}+$
$\vec{n}$
local horizon

1.c)

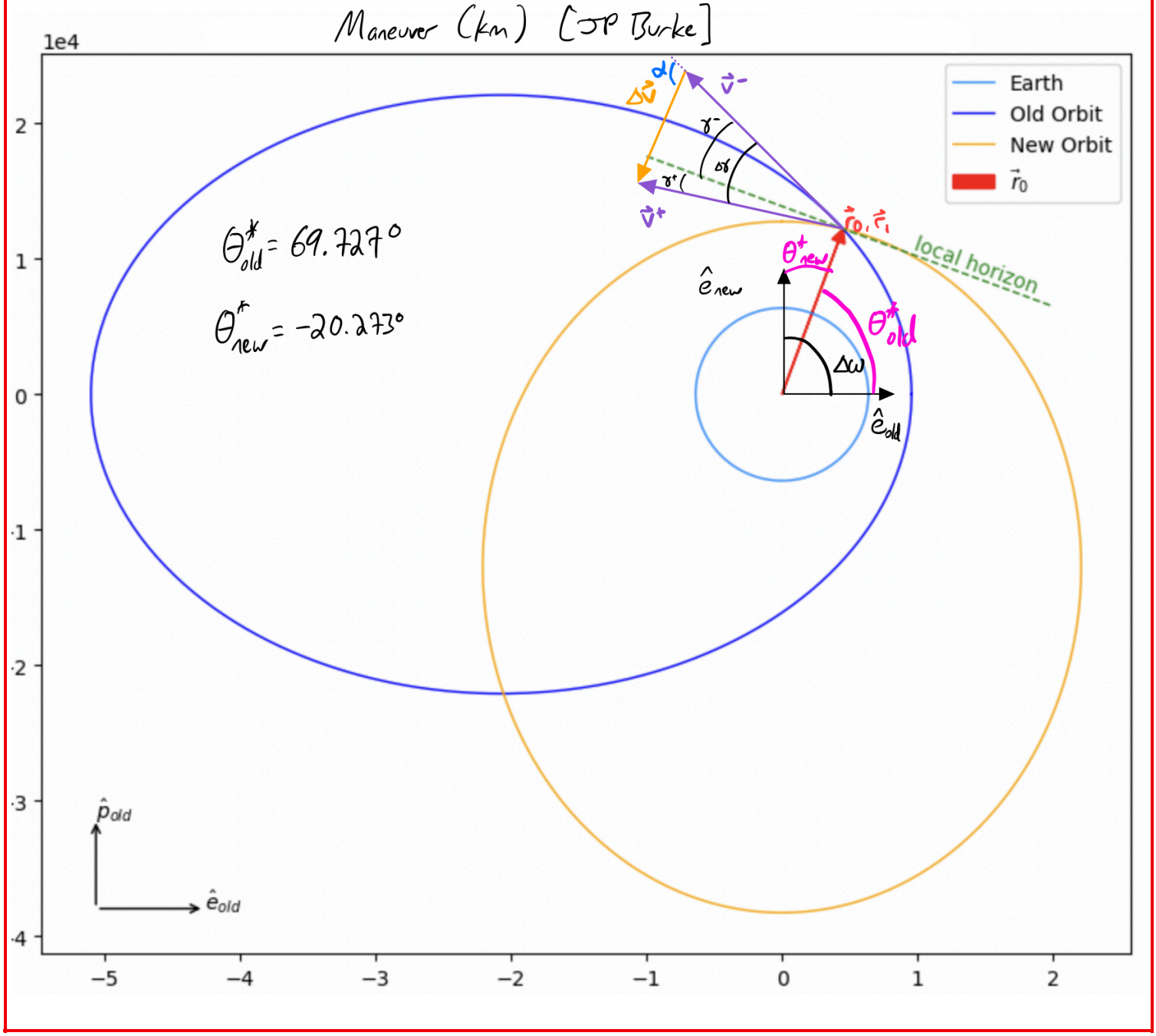
> Plot the old and new orbits. You can use either Matlab or GMAT. On the plot, mark $\vec{r}_0$, $\vec{r}_1$, $\vec{v}_1^-$, local horizon, γ_1^-, $\vec{v}_1^+$, γ^+, $\Delta\vec{v}$, α. Identify $\Delta\omega$ and the old and new true anomalies.

Problem statement: Plot the orbit before and after the burn. On the plot, mark a variety of orbital features.

```
fig, ax = plt.subplots()
fig.set_size_inches(8, 8)
ax = plt.gca()
ax.set_aspect('equal')
ax.ticklabel_format(axis='both', style='sci', scilimits=(0,0))

# earth
earth_points = []
for ang in np.linspace(0, 2*pi, 360):
    earth_points.append((EARTH_RADIUS*cos(ang), EARTH_RADIUS*sin(ang)))
plt.plot(*zip(*earth_points), ls='-', color='dodgerblue', lw=1, label='Earth')

# initial orbit
orb_points = []
for ta in np.linspace(0, 2*pi, 360):
    r = p_old / (1 + e_old * np.cos(ta))
    orb_points.append((r*cos(ta),  r*sin(ta)))
plt.plot(*zip(*orb_points), ls='-', color='blue', lw=1, label='Old Orbit')

# new orbit
orb_points = []
for ta in np.linspace(0, 2*pi, 360):
    r = p_new / (1 + e_new * np.cos(ta))
    orb_points.append((r*cos(ta + pi/2),  r*sin(ta + pi/2)))
plt.plot(*zip(*orb_points), ls='-', color='orange', lw=1, label='New Orbit')

arrow_kw = {'head_width': 1.0e3, 'head_length': 1.0e3, 'length_includes_head': True}

# r_0
plt.arrow(0, 0, r_A*cos(intersection_a_old_ta), r_A*sin(intersection_a_old_ta), color='red', label=r'$\vec{r}_0$', **arrow_kw)
plt.text(r_A*cos(intersection_a_old_ta), r_A*sin(intersection_a_old_ta), r'$\vec{r}_0$', color='red')

# local horizon 0
slope = cos(intersection_a_old_ta)/sin(intersection_a_old_ta)
plt.plot(*zip(*[
        (-1e4, -slope*(-1e4-r_A*cos(intersection_a_old_ta))+r_A*sin(intersection_a_old_ta)),
        ( 2e4, -slope*( 2e4-r_A*cos(intersection_a_old_ta))+r_A*sin(intersection_a_old_ta)),
    ]),    lw=1, ls='--', color='forestgreen'
)
plt.text(r_A*cos(intersection_a_old_ta)+5e3, r_A*sin(intersection_a_old_ta)-5e3, 'local horizon', ha='left', va='bottom', color='forest
         transform_rotates_text=True, rotation=-20, rotation_mode='default')

# draw unit vector annotation
ax.annotate('',  (0.05, 0.05), (0.15, 0.05), xycoords='axes fraction', arrowprops=dict(arrowstyle="<-"))
ax.annotate('',  (0.05, 0.05), (0.05, 0.15), xycoords='axes fraction', arrowprops=dict(arrowstyle="<-"))
ax.annotate(r'$\hat{e}_{old}$', (0.05, 0.05), (0.15, 0.05), xycoords='axes fraction')
ax.annotate(r'$\hat{p}_{old}$', (0.05, 0.05), (0.05, 0.15), xycoords='axes fraction')

ax.legend()
ax.set_title("Maneuver (km) [JP Burke] [UNANNOTATED]")

plt.tight_layout()
plt.show()
```



Annotated version:


Maneuver (km) [JP Burke]
1e4
Earth
Old Orbit
New Orbit
$\vec{r}_0$
$\theta^*_{old} = 69.727°$
$\theta^*_{new} = -20.273°$
local horizon
$\hat{e}_{new}$
$\hat{e}_{old}$
$\Delta\omega$
$\hat{p}_{old}$
$\hat{e}_{old}$

Problem 2

Assuming a relative 2B model, a spacecraft moves in Earth orbit characterized by the following:

$$a = 8R_{\oplus} \quad \Omega = 60^{\circ}$$
$$e = .7 \quad \omega = 90^{\circ}$$
$$i = 30^{\circ}$$

with respect to an Earth inertial equatorial coordinate system. To meet some specific objective, a maneuver is planned to adjust the orbit:

2.a)

Let a maneuver be implemented when $\theta^* = 150^{\circ}$. Apply a $\Delta\vec{v}$ such that the magnitude is 1 km/s (whew! large!) and directed such that $\alpha = +45^{\circ}$ and $\beta = +60^{\circ}$. Express the $\Delta\vec{v}$ vector in terms of rotating orbit unit vectors ($\hat{r}$, $\hat{\theta}$); inertial orbit unit vectors ($\hat{e}$, $\hat{p}$); VNC unit vectors. (Note that these are all associated with the **ORIGINAL** orbit!) Print the orbit as seen in GMAT; identify the maneuver location.

Problem statement: Given a spacecraft in the orbit defined above, execute a 1km/s burn at $\theta^* = 150^{\circ}$ such that $\alpha = 45^{\circ}$ and $\beta = 60^{\circ}$. Express the delta-v vector in terms of various unit vectors. Plot the orbit in GMAT and identify the maneuver location.

First let's capture the info from the problem in code:

```
a_old    = 8 * EARTH_RADIUS
e_old    = 0.7
i_old    = deg2rad(30)
raan_old = deg2rad(60)
aop_old  = deg2rad(90)
ta_old   = deg2rad(150)
alpha    = deg2rad(45)
beta     = deg2rad(60)
aol_old  = aop_old + ta_old
burn_mag = 1.0
```

In order to determine our burn components, we'll break it down per the drawing in the class notes (JS_3Dex 3):

$$\vec{\Delta v}_{\hat{V}} = \left(1.0\ \frac{km}{s}\right)\cos\beta\cos\alpha$$
$$\vec{\Delta v}_{\hat{N}} = \left(1.0\ \frac{km}{s}\right)\sin\beta$$
$$\vec{\Delta v}_{\hat{C}} = \left(1.0\ \frac{km}{s}\right)\cos\beta\sin\alpha$$

```
burn_vnc = np.array([
    cos(beta) * cos(alpha),
    sin(beta),
    cos(beta) * sin(alpha),
])
print(f'burn VNC: {burn_vnc} km/s ({norm(burn_vnc)} mag)')
```

```
burn VNC: [ 0.35355  0.86603  0.35355] km/s (1.0 mag)
```

Next let's convert the Δv vector to $\hat{r}_{old}$, $\hat{\theta}_{old}$, $\hat{h}_{old}$ and $\hat{e}_{old}$, $\hat{p}_{old}$, $\hat{h}_{old}$.

First we'll note that we get $\hat{h}_{old}$ for free since it's the same direction in all three frames. So $\Delta v_{\hat{h}_{old}}$ = 0.866 km/s.

For $\hat{r}_{old}$, $\hat{\theta}_{old}$, $\hat{h}_{old}$, we'll use this note from class:

$$\Delta\vec{v} = \Delta v(c_\beta c_\phi \hat{\theta} + c_\beta s_\phi \hat{r} + s_\beta \hat{h})$$

From which we find:

$$\begin{aligned}\Delta\vec{v}_{\hat{r}\hat{\theta}\hat{h}} =& \cos\beta \sin\phi\ \hat{r} + \\ & \cos\beta \cos\phi\ \hat{\theta} + \\ & \sin\beta\ \hat{h}\end{aligned}$$

For $\hat{e}_{old}$, $\hat{p}_{old}$, $\hat{h}_{old}$, we use our usual $\hat{r}\hat{\theta}\hat{h} \rightarrow \hat{e}\hat{p}\hat{h}$ transformation and multiply by the relevant components of our newly calculated $\hat{r}\hat{\theta}\hat{h}$ vector:

$$\begin{aligned}\Delta\vec{v}_{\hat{e}\hat{p}\hat{h}} =& (\Delta\vec{v}_{\hat{r}} \cos\theta^* - \Delta\vec{v}_{\hat{\theta}} \sin\theta^*)\ \hat{e} \\ & (\Delta\vec{v}_{\hat{\theta}} \sin\theta^* + \Delta\vec{v}_{\hat{\theta}} \cos\theta^*)\ \hat{p} \\ & \Delta\vec{v}_{\hat{h}}\ \hat{h}\end{aligned}$$

Let's also convert $\Delta\vec{v}$ to $\hat{x}$, $\hat{y}$, $\hat{z}$ to make some math easier later:

$$\begin{aligned}\Delta\vec{v}_{\hat{x}\hat{y}\hat{z}} =& [\Delta\vec{v}_{\hat{r}}(\cos\Omega\cos\theta - \sin\Omega\cos i\sin\theta) + \Delta\vec{v}_{\hat{\theta}}(-\cos\Omega\sin\theta - \sin\Omega\cos i\cos\theta) + \Delta\vec{v}_{\hat{h}}(\sin\Omega\sin i)]\hat{x} \\ & [\Delta\vec{v}_{\hat{r}}(\sin\Omega\cos\theta + \cos\Omega\cos i\sin\theta) + \Delta\vec{v}_{\hat{\theta}}(-\sin\Omega\sin\theta + \cos\Omega\cos i\cos\theta) + \Delta\vec{v}_{\hat{h}}(-\cos\Omega\sin i)]\hat{y} \\ & [\Delta\vec{v}_{\hat{r}}(\sin i\sin\theta) + \Delta\vec{v}_{\hat{\theta}}(\sin i\cos\theta) + \Delta\vec{v}_{\hat{h}}(\cos i)]\hat{z}\end{aligned}$$

In order to calculate ϕ we need to calculate γ^-. To do that we'll use the following:

$$p = a(1 - e^2)$$

$$r = \frac{p}{1 + e\cos\theta^*}$$

$$v = \sqrt{\frac{2\mu}{r} - \frac{\mu}{a}}$$

$$h = \sqrt{\mu p}$$

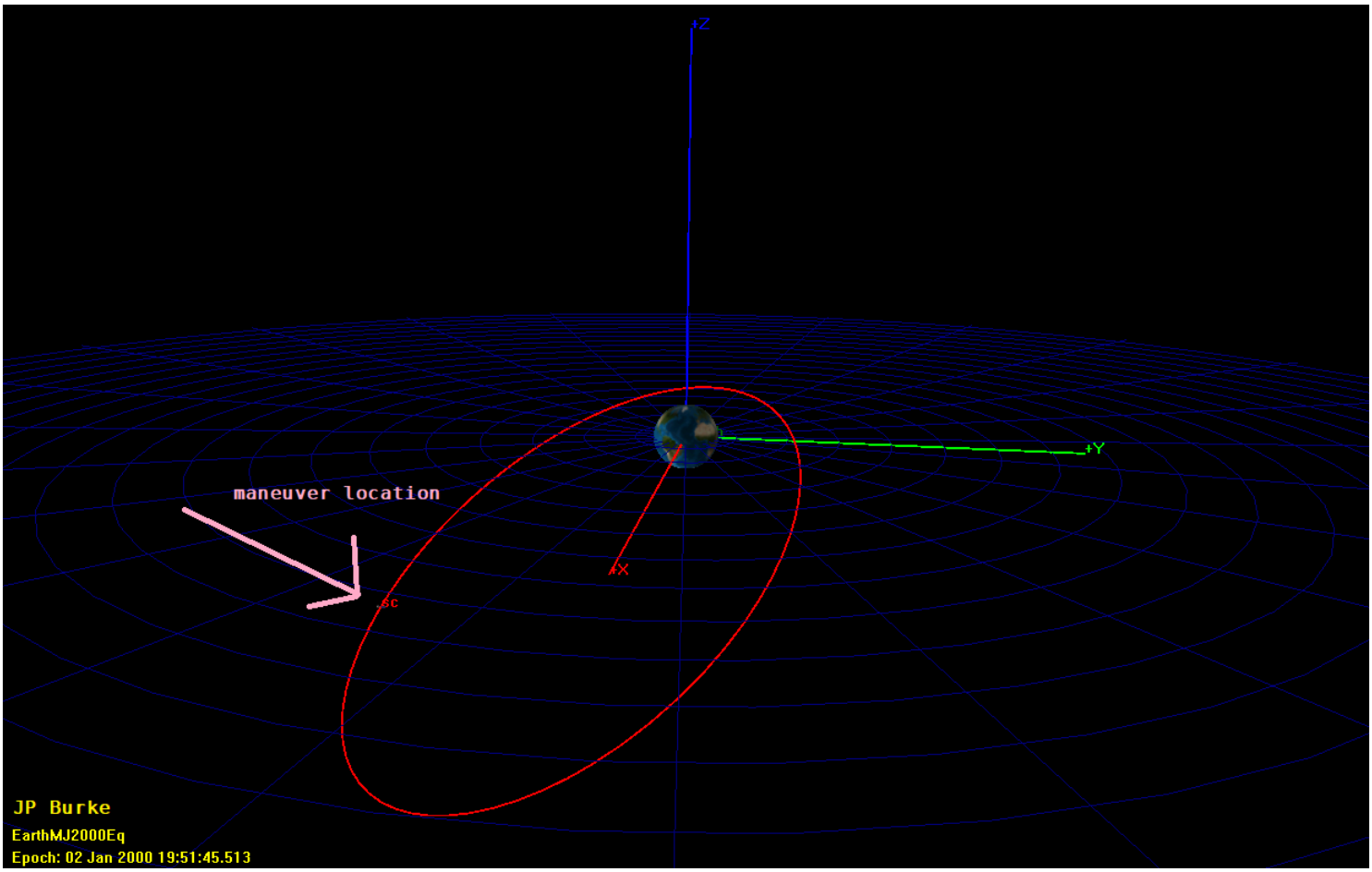
$$\gamma^- = \cos^{-1}\left(\frac{h}{rv}\right)$$

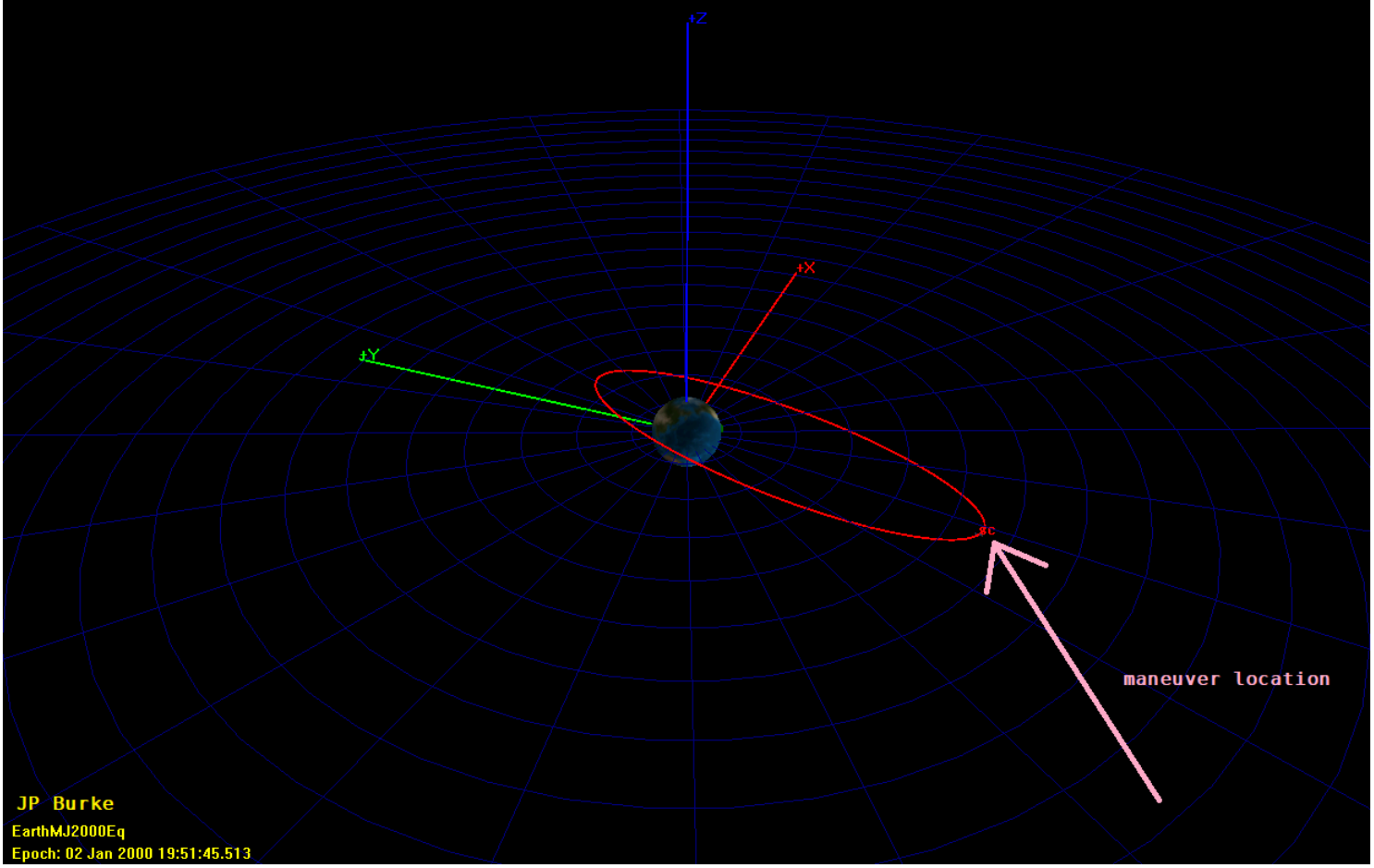
```
p_old     = a_old * (1 - e_old**2)
r         = p_old / (1 + e_old * cos(ta_old))
v_old     = sqrt(2*EARTH_MU / r - EARTH_MU / a_old)
h_mag_old = sqrt(EARTH_MU * p_old)
fpa_old   = arccos(h_mag_old/(r * v_old))
phi       = fpa_old + alpha

burn_rth = burn_mag * np.array([
    cos(beta) * sin(phi),
    cos(beta) * cos(phi),
    sin(beta)
])

burn_eph = burn_mag * np.array([
    burn_rth[0]*cos(ta_old) - burn_rth[1]*sin(ta_old),
    burn_rth[0]*sin(ta_old) + burn_rth[1]*cos(ta_old),
    burn_rth[2]
])
```

```
burn_xyz = burn_mag * np.array([
    burn_rth[0] * ( cos(raan_old) * cos(aol_old) - sin(raan_old) * cos(i_old) * sin(aol_old)) +
    burn_rth[1] * (-cos(raan_old) * sin(aol_old) - sin(raan_old) * cos(i_old) * cos(aol_old)) +
    burn_rth[2] * ( sin(raan_old) * sin(i_old)),

    burn_rth[0] * ( sin(raan_old) * cos(aol_old) + cos(raan_old) * cos(i_old) * sin(aol_old)) +
    burn_rth[1] * (-sin(raan_old) * sin(aol_old) + cos(raan_old) * cos(i_old) * cos(aol_old)) +
    burn_rth[2] * (-cos(raan_old) * sin(i_old)),

    burn_rth[0] * ( sin(i_old) * sin(aol_old)) +
    burn_rth[1] * ( sin(i_old) * cos(aol_old)) +
    burn_rth[2] * ( cos(i_old))
])

print(f'burn VNC: {burn_vnc} km/s ({norm(burn_vnc)} km/s mag)')
print(f'burn rth: {burn_rth} km/s ({norm(burn_rth)} km/s mag)')
print(f'burn eph: {burn_eph} km/s ({norm(burn_eph)} km/s mag)')
print(f'burn xyz: {burn_xyz} km/s ({norm(burn_xyz)} km/s mag)')
```

```
burn VNC: [ 0.35355  0.86603  0.35355] km/s (1.0 km/s mag)
burn rth: [ 0.49914  0.02938  0.86603] km/s (1.0 km/s mag)
burn eph: [-0.44696  0.22412  0.86603] km/s (1.0 km/s mag)
burn xyz: [ 0.59815 -0.60414  0.52652] km/s (1.0 km/s mag)
```

$\Delta\vec{v}$	0.35355 $\hat{V}$ + 0.86603 $\hat{N}$ + 0.35355 $\hat{C}$ km/s
$\Delta\vec{v}$	0.49914 $\hat{r}$ + 0.02938 $\hat{\theta}$ + 0.86603 $\hat{h}$ km/s
$\Delta\vec{v}$	-0.44696 $\hat{e}$ + 0.22412 $\hat{p}$ + 0.86603 $\hat{h}$ km/s




+Z
+X
+Y
sc
maneuver location
JP Burke
EarthMJ2000Eq
Epoch: 02 Jan 2000 19:51:45.513

2.b)

> Determine the state immediately following the maneuver: $\vec{r}$, $\vec{v}$, r, v, γ, θ^*, M, E, $\Delta\omega$.
>
> Determine the characteristics of the new orbit: a_N, e_N, i_N, Ω_N, ω, $\Delta\omega$.

Problem statement: Given the post-burn spacecraft from the previous problem, find a variety of orbital features.

First let's figure out $\vec{r}$ and $\vec{v}^-$ in $\hat{x}$, $\hat{y}$, $\hat{z}$.

$$\begin{aligned}\vec{r} =& r\left[\cos\Omega\cos\theta - \sin\Omega\cos i\sin\theta\right]\hat{x} \\ & r\left[\sin\Omega\cos\theta + \cos\Omega\cos i\sin\theta\right]\hat{y} \\ & r\left[\sin i\sin\theta\right]\hat{z}\end{aligned}$$

For $\vec{v}$ we'll also need:

$$h = \sqrt{\mu p}$$

$$\dot{r} = \sqrt{\frac{\mu}{p}}e\sin\theta^*$$

$$\dot{\theta} = \frac{h}{r^2}$$

Now we can calculate $\vec{v}$:

$$\begin{aligned}\vec{v} =& [\dot{r}(\cos\Omega\cos\theta - \sin\Omega\cos i\sin\theta) + r\dot{\theta}(-\cos\Omega\sin\theta - \sin\Omega\cos i\cos\theta)]\hat{x} \\ & [\dot{r}(\sin\Omega\cos\theta + \cos\Omega\cos i\sin\theta) + r\dot{\theta}(-\sin\Omega\sin\theta + \cos\Omega\cos i\cos\theta)]\hat{y} \\ & [\dot{r}(\sin i\sin\theta) + r\dot{\theta}\sin i\cos\theta]\hat{z}\end{aligned}$$

```
r_eph     = np.array([ r*cos(ta_old), r*sin(ta_old), 0 ])
v_pre_eph = np.array([
    -(EARTH_MU/p_old)**0.5 * sin(ta_old),
     (EARTH_MU/p_old)**0.5 * (e_old + cos(ta_old)),
    0,
])
v_post_eph = v_pre_eph + burn_eph

r_xyz = r * np.array([
    (cos(raan_old)*cos(aol_old) - sin(raan_old)*cos(i_old)*sin(aol_old)),
    (sin(raan_old)*cos(aol_old) + cos(raan_old)*cos(i_old)*sin(aol_old)),
    (sin(i_old)*sin(aol_old)),
])

h_mag_old     = sqrt(EARTH_MU * p_old)
rdot_old      = (EARTH_MU / p_old)**0.5 * (e_old * sin(ta_old))
theta_dot_old = h_mag_old / r**2

v_pre_xyz = np.array([
    rdot_old * ( cos(raan_old)*cos(aol_old) - sin(raan_old)*cos(i_old)*sin(aol_old)) +
        r * theta_dot_old * (-cos(raan_old) * sin(aol_old) - sin(raan_old)*cos(i_old)*cos(aol_old)),
    rdot_old * ( sin(raan_old)*cos(aol_old) + cos(raan_old)*cos(i_old)*sin(aol_old)) +
        r * theta_dot_old * (-sin(raan_old) * sin(aol_old) + cos(raan_old)*cos(i_old)*cos(aol_old)),
    rdot_old * ( sin(i_old) * sin(aol_old)) +
        r * theta_dot_old * ( sin(i_old) * cos(aol_old)),
])
```

Now that we have $\vec{r}$, $\vec{v}^-$, and $\Delta\vec{v}$ in $\hat{x}$, $\hat{y}$, $\hat{z}$ we can easily find $\vec{v}^+$ by just adding $\Delta\vec{v}$ to $\vec{v}^-$. From there we can start to find some of the basics.

$$\vec{v}^+ = \vec{v}^- + \Delta\vec{v}$$

$$\vec{h}^{-} = \vec{r} \times \vec{v}^{-}$$

$$\vec{h}^{+} = \vec{r} \times \vec{v}^{+}$$

$$p_{new} = \frac{|\vec{h}^{+}|^2}{\mu}$$

$$\mathcal{E}_{new} = \frac{|\vec{v}^{+}|^2}{2} - \frac{\mu}{r}$$

$$a_{new} = -\frac{\mu}{2\mathcal{E}}$$

$$e_{new} = \sqrt{1 + \frac{2\mathcal{E}_{new}|\vec{h}^{+}|^2}{\mu^2}}$$

$$\gamma_{new} = cos^{-1}\left(\frac{|\vec{h}^{+}|}{r|\vec{v}^{+}|}\right)$$

```
v_post_xyz = v_pre_xyz + burn_xyz

v_new = norm(v_post_eph)

h_old_xyz  = cross(r_xyz, v_pre_xyz)
h_new_xyz  = cross(r_xyz, v_post_xyz)
h_new_mag  = norm(h_new_xyz)

p_new      = norm(h_new)**2/EARTH_MU
energy_new = v_new**2 / 2 - EARTH_MU / r
a_new      = -EARTH_MU / (2 * energy_new)
e_new      = sqrt(1 + 2 * energy_new * h_new_mag**2 / EARTH_MU**2)
fpa_new    = arccos(h_new_mag / (r * v_new))

E_new  = arccos((a_new - r)/(a_new * e_new))
M_new  = E_new - e_new * sin(E_new)

v_new = norm(v_post_eph)
```

Now let's find i, Ω, and ω.

$$i_{new} = cos^{-1}\left(\hat{h} \cdot \hat{z}\right)$$

We choose the positive outcome by convention.

$$\Omega_{new} = sin^{-1}\left(\frac{\hat{h}_x}{\sin i_{new}}\right)$$

And in order to get the right quadrant we also calculate

$$\Omega_{new} = cos^{-1}\left(-\frac{\hat{h}_y}{\sin i_{new}}\right)$$

We'll calculate both quadrants for both equations and select the outcome that is consistent between the two.

Similarly, we'll calculate ω two different ways:

$$\theta_{old} = sin^{-1}\left(\frac{\hat{r}_z}{\sin i_{new}}\right)$$

$$\theta_{old} = \pi - sin^{-1}\left(\frac{\hat{r}_z}{\sin i_{new}}\right)$$

Finally, $\Delta\omega$ is simply

$$\Delta\omega = \omega_{new} - \omega_{old}$$

```
i_old_check = arccos(np.dot(h_old_xyz/norm(h_old_xyz), [0, 0, 1]))
assert abs(i_old - i_old_check) < 1e-6

i_new = arccos(np.dot(h_new_xyz/norm(h_new_xyz), [0, 0, 1]))

# find new raan, with quadrant checks
raan_new_a = arcsin( (h_new_xyz[0]/norm(h_new_xyz)) / sin(i_new) )
raan_new_b = arccos(-(h_new_xyz[1]/norm(h_new_xyz)) / sin(i_new) )
raan_new_options = [
    round((     raan_new_a) % (2*pi), 5),
    round((pi - raan_new_a) % (2*pi), 5),
    round(( raan_new_b)     % (2*pi), 5),
    round((-raan_new_b)     % (2*pi), 5),
]

raan_new = None
for candidate_raan in raan_new_options:
    if raan_new_options.count(candidate_raan) > 1:
        raan_new = candidate_raan
        break
assert raan_new is not None

ta_new = arctan(
    ((r*v_new**2/EARTH_MU) * sin(fpa_new)*cos(fpa_new)) /
    ((r*v_new**2/EARTH_MU)*cos(fpa_new)**2 - 1)
) % (2*pi)
ta_new -= pi # correct for quadrant, based on flight-path angle

aol_new_a = arcsin((r_xyz[2]/r) / sin(i_new)) % (2*pi)
aol_new_b = pi - aol_new_a
test_a = cos(raan_new) * cos(aol_new_a) - sin(raan_new) * cos(i_new) * sin(aol_new_a)
test_b = cos(raan_new) * cos(aol_new_b) - sin(raan_new) * cos(i_new) * sin(aol_new_b)

aol_new = None
if (test_a - r_xyz[0]/r) < 1e-3 :
    aol_new = aol_new_a
elif (test_b - r_xyz[0]/r) < 1e-3:
    aol_new = aol_new_b

aol_new = aol_new_a % (2*pi)
assert aol_new is not None

aop_new = aol_new - ta_new

delta_aop = aop_new - aol_old

print(f'r_eph:      {r_eph} km')
print(f'v_pre_eph:  {v_pre_eph} km/s, (mag: {norm(v_pre_eph)})')
print(f'v_pre_xyz:  {v_pre_xyz} km/s, (mag: {norm(v_pre_xyz)})')
print(f'v_post_eph: {v_post_eph} km/s, (mag: {norm(v_post_eph)})')
print(f'v_post_xyz: {v_post_xyz} km/s, (mag: {norm(v_post_xyz)})')
print(f'r:          {r:{num_fmt}} km')
print(f'v_new:      {v_new:{num_fmt}} km/s')
print(f'fpa_new:    {rad2deg(fpa_new):{num_fmt}} deg')
print(f'ta_new:     {rad2deg(ta_new):{num_fmt}} deg')
print(f'M_new:      {rad2deg(M_new):{num_fmt}} deg')
print(f'E_new:      {rad2deg(E_new):{num_fmt}} deg')
print(f'delta_aop:  {rad2deg(delta_aop):{num_fmt}} deg')
print(f'a_new:      {a_new:{num_fmt}} km')
print(f'e_new:      {e_new:{num_fmt}}')
print(f'i_new:      {rad2deg(i_new):{num_fmt}} deg')
print(f'raan_new:   {rad2deg(raan_new):{num_fmt}} deg')
print(f'aop_new:    {rad2deg(aop_new):{num_fmt}} deg')
```

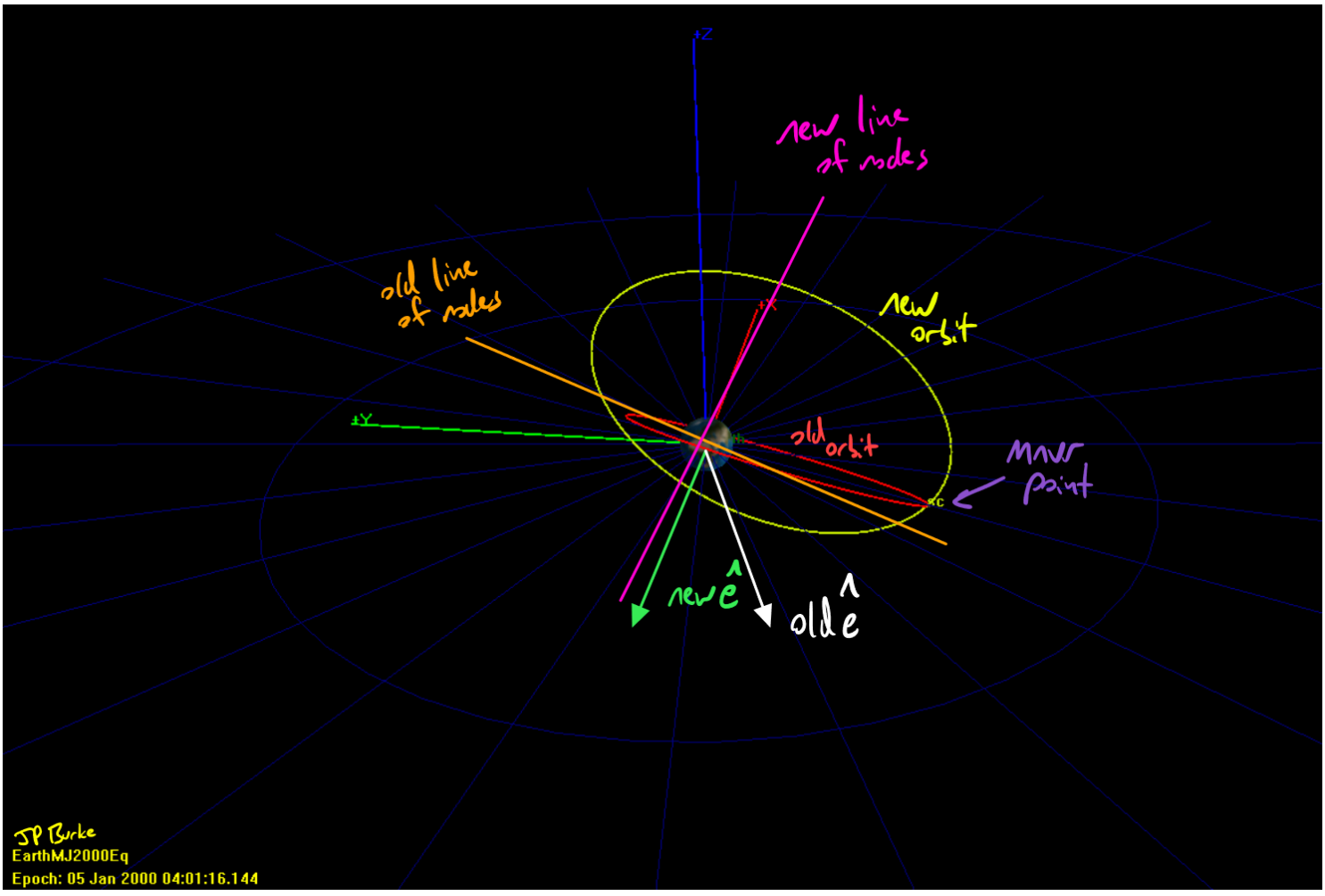
```
r_eph:      [-57230.62523  33042.11688  0.00000] km
v_pre_eph:  [-1.95687 -0.64978  0.00000] km/s, (mag: 2.0619295235686956)
v_pre_xyz:  [ 1.79254 -0.28462 -0.97843] km/s, (mag: 2.061929523568695)
v_post_eph: [-2.40382 -0.42566  0.86603] km/s, (mag: 2.5902813956418713)
v_post_xyz: [ 2.39070 -0.88876 -0.45191] km/s, (mag: 2.590281395641871)
r:             66084.234 km
v_new:             2.590 km/s
fpa_new:          46.180 deg
ta_new:          130.025 deg
M_new:            40.013 deg
E_new:            81.091 deg
delta_aop:       -75.315 deg
a_new:         74451.215 km
e_new:             0.726
i_new:            28.467 deg
raan_new:        358.680 deg
aop_new:         164.685 deg
```

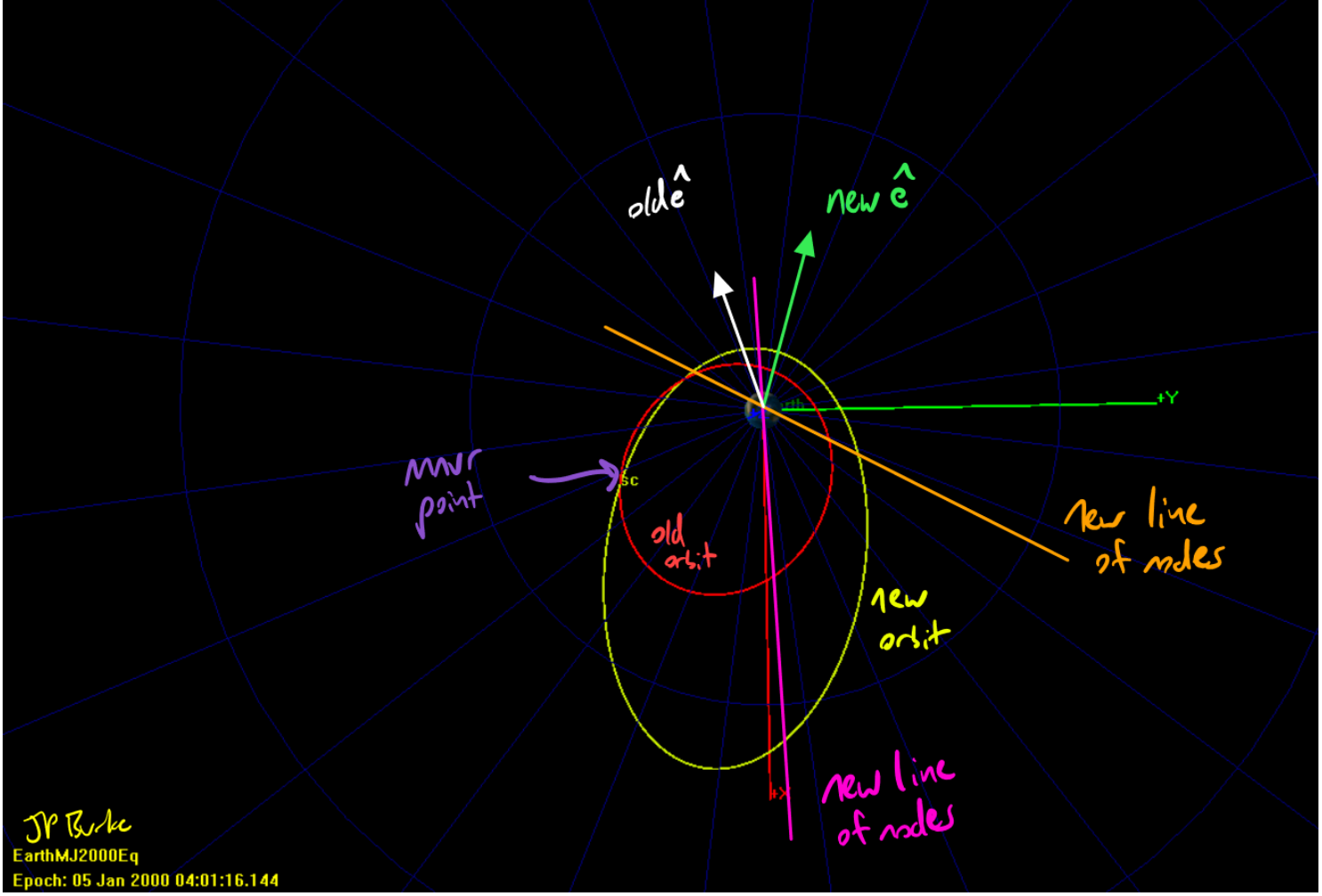
$\vec{r}^+$	-57,230.625 km $\hat{e}$ + 33,042.117 km $\hat{p}$
$\vec{v}^+$	-2.40382 km/s $\hat{e}$ - 0.42566 km/s $\hat{p}$ + 0.86603 km/s $\hat{h}$
$\vec{v}^+$	2.39070 km/s $\hat{x}$ - 0.88876 km/s $\hat{y}$ - 0.45191 km/s $\hat{z}$
r^+	66,084.234 km
v^+	2.590 km/s
γ^+	46.180°
θ^{*+}	130.025°
M^+	40.013°
E^+	81.091°
$\Delta\omega$	75.315°
a_N	74,451.215 km
e_N	0.726
i_N	28.467°
Ω_N	358.680°
ω_N	164.685°
$\Delta\omega$	75.315°

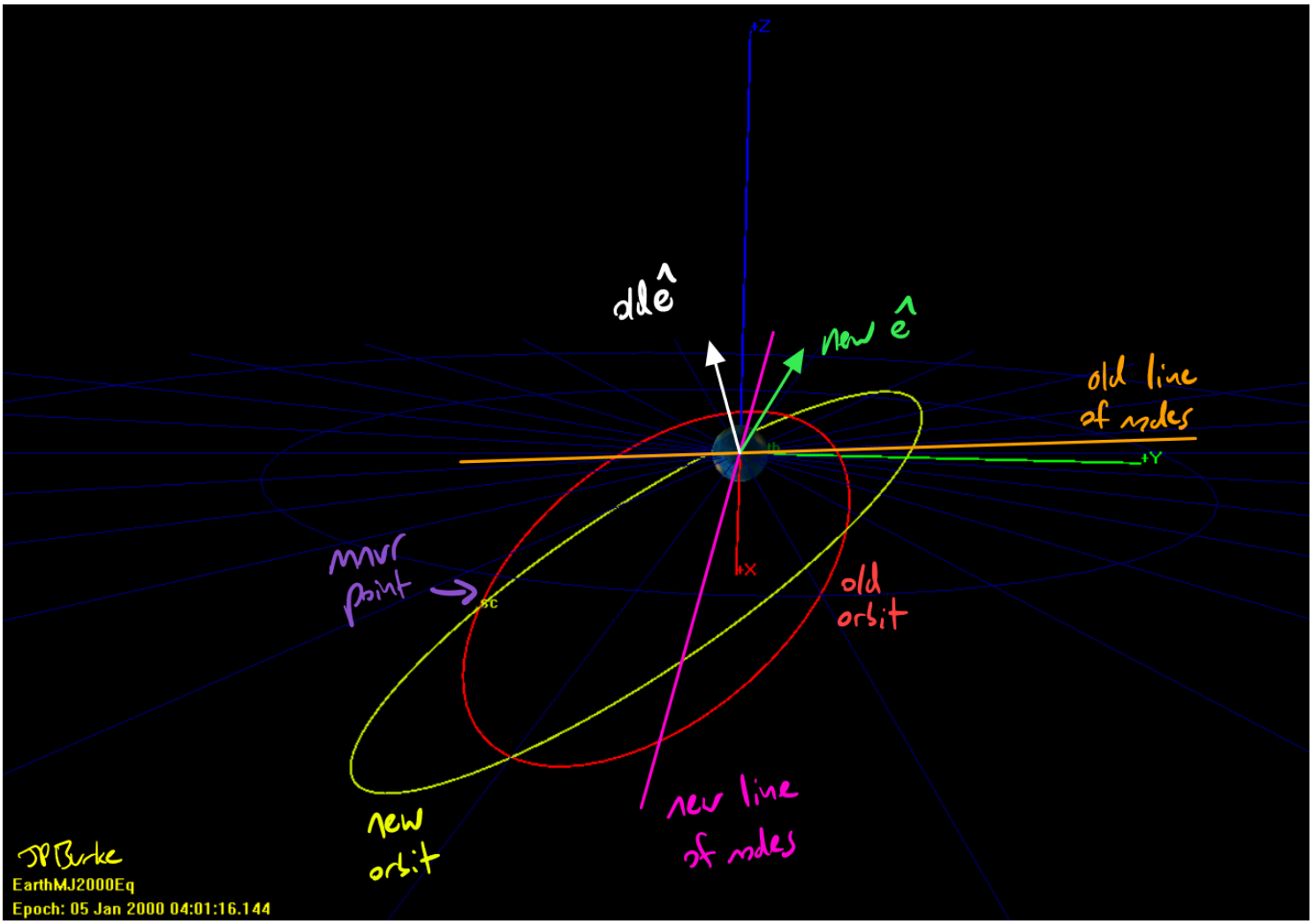
2.c)

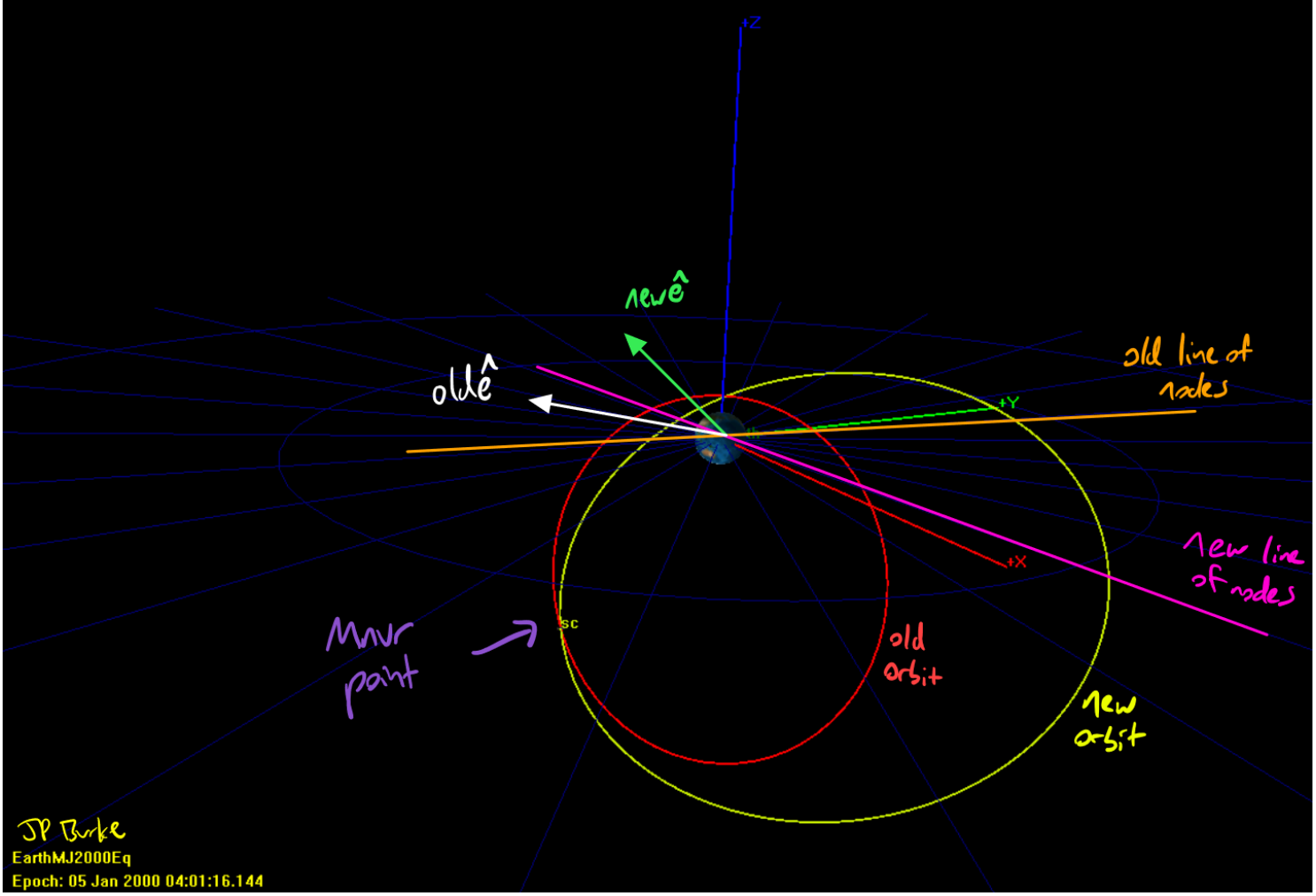
> Implement the maneuver and confirm the results in GMAT. [Note that there is another GMAT Tip in Brightspace: "Implementing Maneuvers in GMAT"]
>
> Print the corresponding parts of the scroll.
>
> Plot the initial/final orbit. Add the Equatorial Plane, the J2000 unit vectors, the Satellite Periapsis vector, and the Line of Nodes vector to a 3D GMAT plot; plot two different views. Indicate all of the appropriate information of the plot.

Problem statement: Implement the maneuver in GMAT and confirm results. Adhere to the requirements of the plot specified above.




old ê
new ê
+Y
MNVR point
SC
old orbit
new line of nodes
new orbit
new line of nodes
JP Burke
EarthMJ2000Eq
Epoch: 05 Jan 2000 04:01:16.144


+Z
old ê
new ê
old line of nodes
+Y
+X
MNVR point
SC
old orbit
new orbit
new line of nodes
JP Burke
EarthMJ2000Eq
Epoch: 05 Jan 2000 04:01:16.144



The raw GMAT output is far too wide for the screen, so I copy/pasted the values and reformatted them:

GMAT field	Units	Pre-Maneuver	Post-Maneuver
sc.Earth.Energy	km^2/s^2	-3.9059	-2.6769
sc.Earth.SMA	km	51025.0904	74451.0641
sc.Earth.ECC	N/A	0.6999	0.7257
sc.EarthMJ2000Eq.INC	deg	29.9999	28.4674
sc.EarthMJ2000Eq.RAAN	deg	59.9999	358.6796
sc.EarthMJ2000Eq.AOP	deg	90.0000	164.6853
sc.Earth.TA	deg	149.9999	130.0249
sc.EarthMJ2000Eq.FPA	deg	48.3687	43.8198
sc.Earth.RMAG	km	66084.2337	66084.2337
sc.EarthMJ2000Eq.VMAG	km/s	2.06192	2.5902
sc.EarthMJ2000Eq.VX	km/s	1.7925	2.3906
sc.EarthMJ2000Eq.VY	km/s	-0.2846	-0.8887
sc.EarthMJ2000Eq.VZ	km/s	-0.9784	-0.4519
sc.Earth.EA	deg	114.9367	81.0915
sc.Earth.MA	deg	78.5687	40.0132
sc.Earth.HMAG	km^2/s	101846.4433	118521.4816

GMAT field	Units	Pre-Maneuver	Post-Maneuver
sc.EarthMJ2000Eq.HX	km^2/s	44100.8035	-1301.7672
sc.EarthMJ2000Eq.HY	km^2/s	-25461.6108	-56479.3533
sc.EarthMJ2000Eq.HZ	km^2/s	88201.6071	202170.2952

```
%General Mission Analysis Tool(GMAT) Script
%Created: 2024-10-25 16:21:07
% JP Burke


%----------------------------------------
%---------- Spacecraft
%----------------------------------------

Create Spacecraft sc;
GMAT sc.DateFormat = UTCGregorian;
GMAT sc.Epoch = '01 Jan 2000 12:00:00.000';
GMAT sc.CoordinateSystem = EarthMJ2000Eq;
GMAT sc.DisplayStateType = Keplerian;
GMAT sc.SMA = 51025.0904;
GMAT sc.ECC = 0.6999999999999982;
GMAT sc.INC = 29.99999999999993;
GMAT sc.RAAN = 59.99999999999999;
GMAT sc.AOP = 90.00000000000006;
GMAT sc.TA = 149.9999999999999;
GMAT sc.DryMass = 850;
GMAT sc.Cd = 2.2;
GMAT sc.Cr = 1.8;
GMAT sc.DragArea = 15;
GMAT sc.SRPArea = 1;
GMAT sc.SPADDragScaleFactor = 1;
GMAT sc.SPADSRPScaleFactor = 1;
GMAT sc.AtmosDensityScaleFactor = 1;
GMAT sc.ExtendedMassPropertiesModel = 'None';
GMAT sc.NAIFId = -10001001;
GMAT sc.NAIFIdReferenceFrame = -9001001;
GMAT sc.OrbitColor = Red;
GMAT sc.TargetColor = Teal;
GMAT sc.OrbitErrorCovariance = [ 1e+70 0 0 0 0 0 ; 0 1e+70 0 0 0 0 ; 0 0 1e+70 0 0 0 ; 0 0 0 1e+70 0 0 ; 0 0 0 0 1e+70 0 ; 0 0 0 0 0 1e+70 ];
GMAT sc.CdSigma = 1e+70;
GMAT sc.CrSigma = 1e+70;
GMAT sc.Id = 'SatId';
GMAT sc.Attitude = CoordinateSystemFixed;
GMAT sc.SPADSRPInterpolationMethod = Bilinear;
GMAT sc.SPADSRPScaleFactorSigma = 1e+70;
GMAT sc.SPADDragInterpolationMethod = Bilinear;
GMAT sc.SPADDragScaleFactorSigma = 1e+70;
GMAT sc.AtmosDensityScaleFactorSigma = 1e+70;
GMAT sc.ModelFile = 'aura.3ds';
GMAT sc.ModelOffsetX = 0;
GMAT sc.ModelOffsetY = 0;
GMAT sc.ModelOffsetZ = 0;
GMAT sc.ModelRotationX = 0;
GMAT sc.ModelRotationY = 0;
GMAT sc.ModelRotationZ = 0;
GMAT sc.ModelScale = 1;
GMAT sc.AttitudeDisplayStateType = 'Quaternion';
GMAT sc.AttitudeRateDisplayStateType = 'AngularVelocity';
GMAT sc.AttitudeCoordinateSystem = EarthMJ2000Eq;
GMAT sc.EulerAngleSequence = '321';

%----------------------------------------
%---------- ForceModels
%----------------------------------------

Create ForceModel Propagator1_ForceModel;
GMAT Propagator1_ForceModel.CentralBody = Earth;
GMAT Propagator1_ForceModel.PrimaryBodies = {Earth};
GMAT Propagator1_ForceModel.Drag = None;
GMAT Propagator1_ForceModel.SRP = Off;
GMAT Propagator1_ForceModel.RelativisticCorrection = Off;
GMAT Propagator1_ForceModel.ErrorControl = RSSStep;
GMAT Propagator1_ForceModel.GravityField.Earth.Degree = 4;
GMAT Propagator1_ForceModel.GravityField.Earth.Order = 4;
GMAT Propagator1_ForceModel.GravityField.Earth.StmLimit = 100;
GMAT Propagator1_ForceModel.GravityField.Earth.PotentialFile = 'JGM2.cof';
GMAT Propagator1_ForceModel.GravityField.Earth.TideModel = 'None';
```

```
%----------------------------------------
%---------- ForceModels
%----------------------------------------

Create ForceModel DefaultProp_ForceModel;
GMAT DefaultProp_ForceModel.CentralBody = Earth;
GMAT DefaultProp_ForceModel.PrimaryBodies = {Earth};
GMAT DefaultProp_ForceModel.Drag = None;
GMAT DefaultProp_ForceModel.SRP = Off;
GMAT DefaultProp_ForceModel.RelativisticCorrection = Off;
GMAT DefaultProp_ForceModel.ErrorControl = RSSStep;
GMAT DefaultProp_ForceModel.GravityField.Earth.Degree = 4;
GMAT DefaultProp_ForceModel.GravityField.Earth.Order = 4;
GMAT DefaultProp_ForceModel.GravityField.Earth.StmLimit = 100;
GMAT DefaultProp_ForceModel.GravityField.Earth.PotentialFile = 'JGM2.cof';
GMAT DefaultProp_ForceModel.GravityField.Earth.TideModel = 'None';

%----------------------------------------
%---------- Propagators
%----------------------------------------

Create Propagator DefaultProp;
GMAT DefaultProp.FM = DefaultProp_ForceModel;
GMAT DefaultProp.Type = RungeKutta89;
GMAT DefaultProp.InitialStepSize = 60;
GMAT DefaultProp.Accuracy = 9.999999999999999e-12;
GMAT DefaultProp.MinStep = 0.001;
GMAT DefaultProp.MaxStep = 2700;
GMAT DefaultProp.MaxStepAttempts = 50;
GMAT DefaultProp.StopIfAccuracyIsViolated = true;

%----------------------------------------
%---------- Burns
%----------------------------------------

Create ImpulsiveBurn ImpulsiveBurn1;
GMAT ImpulsiveBurn1.CoordinateSystem = Local;
GMAT ImpulsiveBurn1.Origin = Earth;
GMAT ImpulsiveBurn1.Axes = VNB;
GMAT ImpulsiveBurn1.Element1 = 0.35355;
GMAT ImpulsiveBurn1.Element2 = 0.86603;
GMAT ImpulsiveBurn1.Element3 = 0.35355;
GMAT ImpulsiveBurn1.DecrementMass = false;
GMAT ImpulsiveBurn1.Isp = 300;
GMAT ImpulsiveBurn1.GravitationalAccel = 9.81;

%----------------------------------------
%---------- Subscribers
%----------------------------------------

Create OrbitView DefaultOrbitView;
GMAT DefaultOrbitView.SolverIterations = Current;
GMAT DefaultOrbitView.UpperLeft = [ 0.3141025641025641 0.03057851239669421 ];
GMAT DefaultOrbitView.Size = [ 1.151282051282051 1.039669421487603 ];
GMAT DefaultOrbitView.RelativeZOrder = 109;
GMAT DefaultOrbitView.Maximized = false;
GMAT DefaultOrbitView.Add = {sc, Earth};
GMAT DefaultOrbitView.CoordinateSystem = EarthMJ2000Eq;
GMAT DefaultOrbitView.DrawObject = [ true true ];
GMAT DefaultOrbitView.DataCollectFrequency = 1;
GMAT DefaultOrbitView.UpdatePlotFrequency = 50;
GMAT DefaultOrbitView.NumPointsToRedraw = 0;
GMAT DefaultOrbitView.ShowPlot = true;
GMAT DefaultOrbitView.MaxPlotPoints = 20000;
GMAT DefaultOrbitView.ShowLabels = true;
GMAT DefaultOrbitView.ViewPointReference = Earth;
GMAT DefaultOrbitView.ViewPointVector = [ 30000 0 0 ];
GMAT DefaultOrbitView.ViewDirection = Earth;
GMAT DefaultOrbitView.ViewScaleFactor = 1;
GMAT DefaultOrbitView.ViewUpCoordinateSystem = EarthMJ2000Eq;
GMAT DefaultOrbitView.ViewUpAxis = Z;
GMAT DefaultOrbitView.EclipticPlane = Off;
GMAT DefaultOrbitView.XYPlane = On;
GMAT DefaultOrbitView.WireFrame = Off;
GMAT DefaultOrbitView.Axes = On;
GMAT DefaultOrbitView.Grid = Off;
GMAT DefaultOrbitView.SunLine = Off;
GMAT DefaultOrbitView.UseInitialView = Off;
GMAT DefaultOrbitView.StarCount = 7000;
```

```
GMAT DefaultOrbitView.EnableStars = Off;
GMAT DefaultOrbitView.EnableConstellations = On;

Create ReportFile ReportFile1;
GMAT ReportFile1.SolverIterations = Current;
GMAT ReportFile1.UpperLeft = [ 0.02136752136752137 0.1743801652892562 ];
GMAT ReportFile1.Size = [ 0.8294871794871795 0.7991735537190082 ];
GMAT ReportFile1.RelativeZOrder = 105;
GMAT ReportFile1.Maximized = false;
GMAT ReportFile1.Filename = 'ReportFile1.txt';
GMAT ReportFile1.Precision = 16;
GMAT ReportFile1.Add = {sc.UTCGregorian, sc.Earth.Energy, sc.Earth.SMA, sc.Earth.ECC, sc.EarthMJ2000Eq.INC, sc.EarthMJ2000Eq.RAAN, sc.EarthMJ2000Eq.
GMAT ReportFile1.WriteHeaders = true;
GMAT ReportFile1.LeftJustify = On;
GMAT ReportFile1.ZeroFill = Off;
GMAT ReportFile1.FixedWidth = true;
GMAT ReportFile1.Delimiter = ' ';
GMAT ReportFile1.ColumnWidth = 23;
GMAT ReportFile1.WriteReport = true;


%----------------------------------------
%---------- Mission Sequence
%----------------------------------------

BeginMissionSequence;
Report ReportFile1 sc.UTCGregorian sc.Earth.Energy sc.Earth.SMA sc.Earth.ECC sc.EarthMJ2000Eq.INC sc.EarthMJ2000Eq.RAAN sc.EarthMJ2000Eq.AOP sc.Eart

%Propagate DefaultProp(sc) {sc.TA = 200};
%Propagate DefaultProp(sc) {sc.TA = 150};
Maneuver ImpulsiveBurn1(sc);
Propagate DefaultProp(sc) {sc.ElapsedSecs = sc.Earth.OrbitPeriod, OrbitColor = [201 238 0]};
%Propagate DefaultProp(sc) {sc.ElapsedSecs = 1};
```

Problem 3

> The New Horizons spacecraft is the only spacecraft from Earth to visit the Pluto System and it is now moving through the Kuiper Belt. After launch January 19, 2006, the spacecraft encountered Jupiter with a closest approach on February 28, 2007. New Horizons then crossed the orbit of Uranus on March 18, 2011, and Pluto closest approach occurred July 14, 2015. New Horizons then continued on to reach Kuiper belt objects. On January 1, 2019, New Horizons accomplished a flyby of Ultima Thule (officially named 2014 MU69), a Kuiper Belt Object that is approximately 4 billion miles (6.4 billion kilometers) from Earth. On April 17, 2021, New Horizons passed a distance of 50 AU from the Sun!

3.a)

> How far is New Horizons from the Sun on October 15, 2024? (Cite your source.) Do you have an image to share?

Problem statement: Find the distance between the Sun and New Horizons on October 15, 2024 and cite the source.

According to the official New Horizons website's "Where is New Horizons" page (https://pluto.jhuapl.edu/Mission/Where-is-New-Horizons.php), on October 23rd, 2024 at 00:00:00 UTC, New Horizons was 60.16 AU from the Sun and traveling at a heliocentric velocity of 13.66 km/s. In order to find the distance on October 15th, we'll assume that for our purposes it is sufficient to assume that the speed is constant and that it is directly away from the sun. As such, we will multiply the 13.66 km/s times eight days to find out how far New Horizons traveled in the previous eight days, and subtract it from its October 23rd position.

```
AU_TO_KM = 1_495_97_898 # from Table of Constants
KM_TO_AU = 1/AU_TO_KM

dist_on_oct_23_2024 = 60.16 * AU_TO_KM
dist_adjust         = 8 * DAY_TO_SEC * 13.66
dist_on_oct_15_2024 = dist_on_oct_23_2024 - dist_adjust
print(f'Approximate distance from Sun on Oct 15 2024: {dist_on_oct_15_2024 * KM_TO_AU:{width}.3f} AU')
```

```
Approximate distance from Sun on Oct 15 2024:       60.097 AU
```

Thus we can say that on October 15 2024 00:00:00 UTC, New Horizons was approximately 60.097 AU from the Sun.

3.b)

> Consider a Pluto mission by doing a few preliminary calculations.
>
> Consider all the dates given above. Create a timeline by converting all the dates to Julian Days (JD). Assume that all events occur at noon. Determine the number of Julian days and Julian years that occur between each event as well as the mission length to the 10-15-4 date. [Note that a Julian Day is exactly 86400 seconds and a Julian year is exactly 365.25 Julian days.]

Problem statement: Determine the Julian days and Julian years that occur between key events in the New Horizons mission. Specify the Julian days/years between events as well as mission length to October 15th, 2024.

For this problem, we'll follow the software wisdom of "never write your own time and date library" and simply use the website mentioned in class: https://ssd.jpl.nasa.gov/tools/jdc/#/cd. This avoids any potential issues with leap days and leap seconds.

Date	Event	Julian date	Julian days since last event	Julian days since launch
January 19, 2006	Launch	2453755	N/A	0
February 28, 2007	Jupiter close approach	2454160	405	405
March 18, 2011	Uranus orbit	2455639	1479	1884
July 14, 2015	Pluto close approach	2457218	1579	3463
January 1, 2019	Ultima Thule flyby	2458485	1267	4730
April 17, 2021	50 AU	2459322	837	5567
October 15, 2024	"Now"	2460599	1277	6844

3.c) [orig 3.a (again)]

> Begin considering a transfer by examining a Hohmann transfer from ⊕ to Pluto ♇. Assume that planetary orbits are coplanar and circular. Compare total $|\Delta\vec{v}|_{total}$ for both a departure and an arrival maneuver and the TOF (time-of-flight in Julian years). Ignore the local gravity fields.
>
> [Do not forget vector diagrams! Each planar $\Delta\vec{v}$ still requires $|\Delta\vec{v}|$, α, β]
>
> Of course, New Horizons did not employ an arrival maneuver as it passed by the system. Focus on just the departure, i.e., $|\Delta\vec{v}_1|$ which is the maneuver necessary at Earth departure.
>
> Even though a Hohmann transfer is the minimum cost two-impulse maneuver, is it likely that we could generally use such a transfer path to Pluto? Why not? Why did New Horizons not use a Hohmann transfer? Compare the Hohmann TOF to the actual Earth-to-Pluto TOF used by New Horizons.
>
> In the US, interplanetary missions are planned to be no longer than 14 years. Why might that be the plan in the US?

Problem statement: Assuming coplanar and circular Earth and Pluto orbits, calculate a Hohmann transfer from Earth to Pluto, ignoring local gravity fields. Consider the pros and cons of using a Hohmann transfer for a Pluto mission and consider the 14 year planning limit on US interplanetary missions.

First let's start defining our variables.

Since we're starting at the Earth, our starting radius is equal to Earth's semi-major axis around the Sun:

$$r_1 = a_\oplus$$

We also note that since we're going from the Earth's orbit to Pluto's orbit, we know the apoapsis and periapsis of our transfer orbit:

$$r_p = a_\oplus$$
$$r_a = a_\text{♇}$$

Now we can find the semi-major axis of our transfer orbit:

$$a_T = \frac{1}{2}(r_p + r_a) = \frac{1}{2}(a_\oplus + a_\text{♇})$$

We could already find v_1^- since it's just $v_{circ\oplus}$:

$$v_1^- = v_{circ\oplus} = \sqrt{\frac{\mu}{a_\oplus}}$$

Now we can find v_1^+ by finding the velocity of a spacecraft at periapsis on the transfer orbit:

$$v_1^+ = \sqrt{\frac{2\mu}{r} - \frac{\mu}{a}} = \sqrt{\frac{2\mu}{a_\oplus} - \frac{\mu}{a_T}}$$

Our departure burn is:

$$\Delta v_1 = v_1^+ - v_1^-$$

Next, we find what velocity we arrive at Pluto's orbit with (at the apoapsis of our transfer orbit):

$$v_2^- = \sqrt{\frac{2\mu}{r} - \frac{\mu}{a}} = \sqrt{\frac{2\mu}{a_{\text{P}}} - \frac{\mu}{a_T}}$$

And then v_2^+ is $v_{circ\text{P}}$:

$$v_2^+ = v_{circ\text{P}} = \sqrt{\frac{\mu}{a_{\text{P}}}}$$

Our insertion burn is:

$$\Delta v_2 = v_2^+ - v_2^-$$

And then we calculate the final delta-v with:

$$\Delta v = v_1 + v_2$$

Lastly, we'll find our time of flight by dividing the period in half:

$$\mathcal{P} = 2\pi\sqrt{\frac{a^3}{\mu}} \Rightarrow TOF = \pi\sqrt{\frac{a_T}{\mu}}$$

Alright, let's do it:

```
# from Table of Constants
a_earth = 149_597_898
a_pluto = 5_907_150_229
sun_mu  = 132_712_440_017.99

# our starting radius is Earth's SMA
r1 = a_earth

a_trans = 0.5 * (a_earth + a_pluto)

v1_pre  = sqrt(sun_mu / r1) # circular orbit
v1_post = sqrt(2*sun_mu / a_earth - sun_mu / a_trans)
burn_1_mag = v1_post - v1_pre

v2_pre  = sqrt(2*sun_mu / a_pluto - sun_mu / a_trans)
v2_post = sqrt(sun_mu / a_pluto)
burn_2_mag = v2_post - v2_pre

hohmann_delta_v_total = burn_1_mag + burn_2_mag

time_of_flight_hohmann = pi * sqrt(a_trans**3 / sun_mu)

SEC_TO_JULIAN_DAY         = 1/86400
JULIAN_DAY_TO_JULIAN_YEAR = 1/365.25
SEC_TO_JULIAN_YEAR        = SEC_TO_JULIAN_DAY * JULIAN_DAY_TO_JULIAN_YEAR

print(f'r1:             {r1:{num_fmt}} km')
print(f'a_trans:        {a_trans:{num_fmt}} km')
print(f'v1_pre:         {v1_pre:{num_fmt}} km/s')
print(f'v1_post:        {v1_post:{num_fmt}} km/s')
print(f'v2_pre:         {v2_pre:{num_fmt}} km/s')
print(f'v2_post:        {v2_post:{num_fmt}} km/s')
print(f'burn 1 mag:     {burn_1_mag:{num_fmt}} km/s')
print(f'burn 2 mag:     {burn_2_mag:{num_fmt}} km/s')
print(f'total dv:       {hohmann_delta_v_total:{num_fmt}} km/s')
print(f'time of flight: {time_of_flight_hohmann:{num_fmt}} sec')
print(f'time of flight: {time_of_flight_hohmann*SEC_TO_JULIAN_DAY:{num_fmt}} Julian days')
print(f'time of flight: {time_of_flight_hohmann*SEC_TO_JULIAN_YEAR:{num_fmt}} Julian years')
```

```
r1:              149597898.000 km
a_trans:         3028374063.500 km
v1_pre:               29.785 km/s
v1_post:              41.598 km/s
v2_pre:                1.053 km/s
v2_post:               4.740 km/s
burn 1 mag:           11.814 km/s
burn 2 mag:            3.686 km/s
total dv:             15.500 km/s
time of flight:  1437170627.651 sec
time of flight:    16633.919 Julian days
time of flight:       45.541 Julian years
```

As way of double-checking, I loaded the Pluto Fact Sheet (https://nssdc.gsfc.nasa.gov/planetary/factsheet/plutofact.html) on GSFC's website. Note that Earth's mean orbital velocity is 29.78 km/s and Pluto's mean orbital velocity is 4.64 km/s. This is close enough to our v_1^- and v_2^+ to attribute any difference to our unrealistic coplanar and circular orbits.

Gathering up our answers:

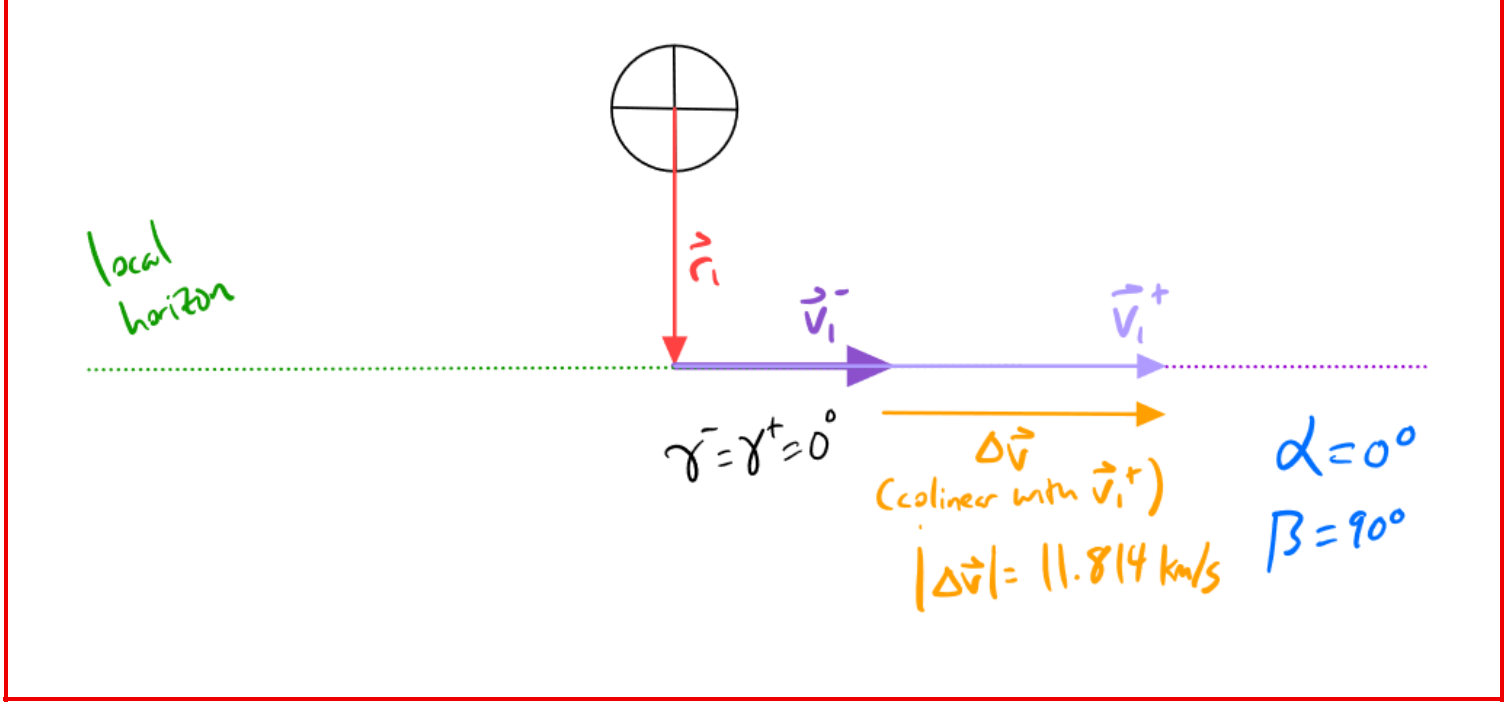
Δv_{dep}	11.814 km/s
Δv_{arr}	3.686 km/s
Δv_{total}	15.5 km/s
Hohmann TOF	16,633.919 Julian days (45.541 Julian years)

Earlier we found that New Horizons' closest approach to Pluto was 3,463 Julian days after launch. This makes our Hohmann transfer nearly five times as long. Cheap in prop, expensive in time!

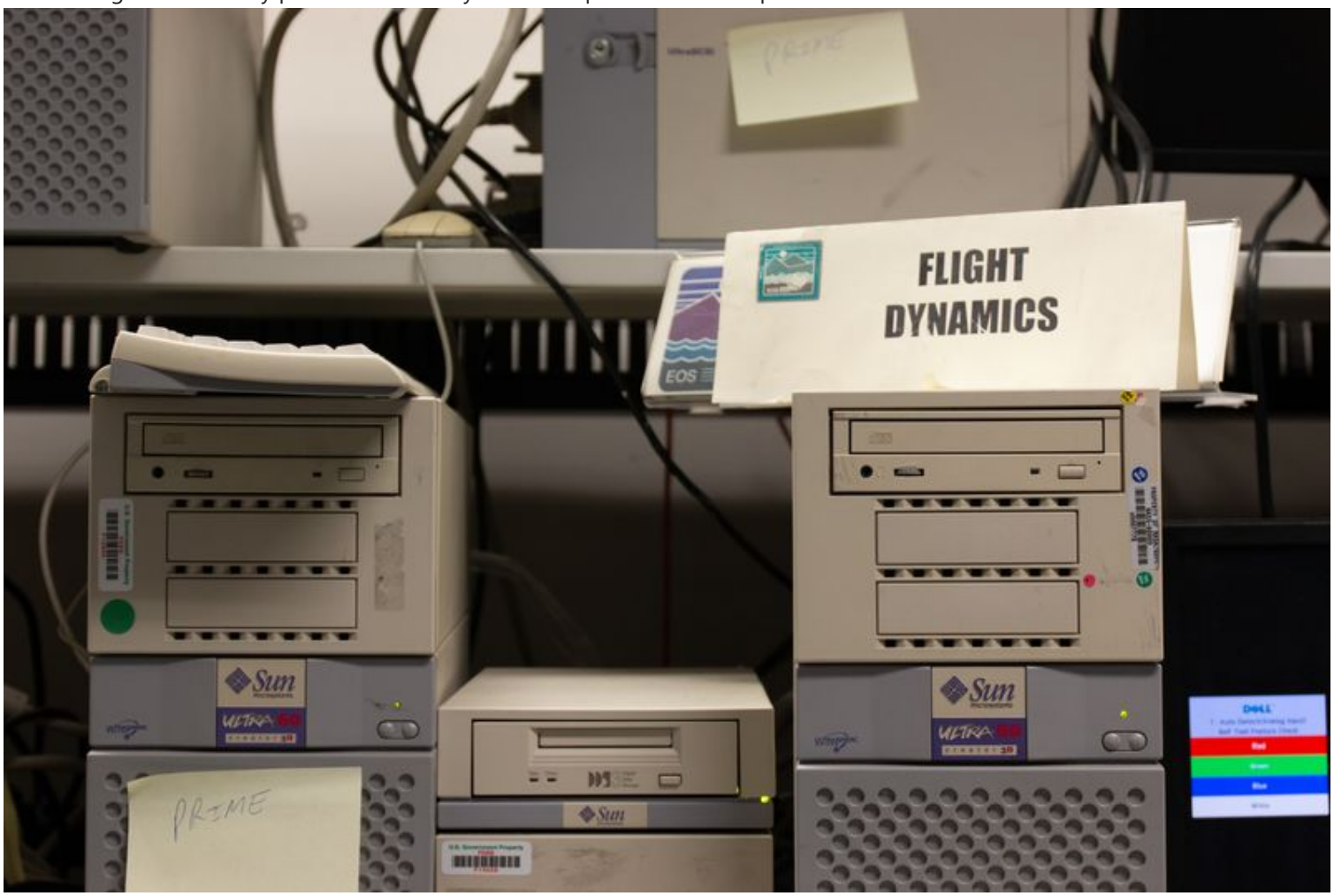
Given this finding, I think it's pretty safe to say that a Hohmann transfer is not a suitable for a transfer to Pluto. The real New Horizons arrived at Pluto in less time, at least in part, by utilizing a gravity assist maneuver when passing Jupiter.

There could be a number of reasons why interplanetary missions are planned not to be longer than 14 years. Just speculating, here are a few possible reasons:

- Concern about longevity of spacecraft components.
- Concern about maintaining a team with the proper expertise over such a long period of time.
- Politicians wanting to see results from projects they fund while they are still in office.
- Cost of operations during the long flight out to the destination.
- Concern about maintaining ground software and hardware for the duration of the mission. (For example, in the corner of the Flight Operations Center for the Earth Observing System was a large and noisy pile of Sun Microsystems computers that supported the Terra spacecraft over 20 years after launch.)



Bonus image of the "noisy pile of Sun Microsystems computers". Sun computers and SCSI drives in 2018!



3.d) [orig 3.c]

> For the Hohmann transfer, what is the phase angle at Earth departure and the synodic period?

To find the phase angle, we first find the mean motion of Pluto:

$$n_{\text{P}} = \sqrt{\frac{\mu}{a_{\text{P}}^3}}$$

Then we multiply the time of flight we calculated earlier by the mean motion. This tells us the angle Pluto traces out during the time it takes our spacecraft to fly from Earth to Pluto. To find the phase angle, we just subtract this angle from 180°.

To calculate the synodic perio between Earth and Pluto, we first find out how many degrees of its orbit Pluto travels through in one year. Then we divide that by the Earth's mean motion. The result is the number of days the Earth must travel after a year in order to sort of "roll back" Pluto's lead and restore the original orbit geometry. So then we just add Earth's period to that number and we arrive at the synodic period.

$$t_s = \mathcal{P}_{\oplus}\left(\frac{n_{\text{P}}}{n_{\oplus}} + 1\right)$$

```
n_pluto = sqrt(sun_mu / a_pluto**3)

phase_angle = pi - time_of_flight * n_pluto

print(f'phase_angle:    {rad2deg(phase_angle):{num_fmt}} deg')

# from Table of Constants
earth_period   = 31558205
n_earth        = sqrt(sun_mu / a_earth**3)
synodic_period = earth_period * (n_pluto/n_earth + 1)

print(f'synodic_period: {synodic_period * SEC_TO_DAY:{num_fmt}} days')
```

```
phase_angle:         113.928 deg
synodic_period:      366.729 days
```

ϕ	113.928°
t_s	366.729 days

3.e) [orig 3.d]

> Reconsider and improve the model for Pluto's orbit. Assume it is eccentric with the eccentricity as given in the Table of Constants. Now, consider a Hohmann transfer to Pluto's perihelion. Re-compute the total $|\Delta\vec{v}|$ and TOF for a Hohmann transfer from Earth to Pluto perihlion.
>
> Does it save $|\Delta\vec{v}|$? TOF?
>
> Upon arrival in the Pluto system, observe that the spacecraft hyperbolic passage by Pluto also allowed close encounters with Pluto's five known moons, including Charon, Hydra, Kerberos, Styx, and Nix!

Problem statement: Calculate total delta-v again using a Hohmann transfer from Earth to Pluto perihelion using a coplanar but eccentric Pluto orbit, and consider if it saves delta-v or time of flight over the previously discussed Hohmann transfer.

When considering this problem, we can just do the exact same thing we did in 3.b, but replacing a_{P} with pe_{P}, which we calculate with:

$$r_{p\text{P}} = a_{\text{P}}(1 - e_{\text{P}})$$

and now v_2^+ is no longer $v_{circ\text{P}}$ but rather:

$$v_2^+ = v_{r_{p\text{P}}} = \sqrt{\frac{2\mu}{r_{p\text{P}}} - \frac{\mu}{a_{\text{P}}}}$$

So now we have:

$$a_T = \frac{1}{2}(a_\oplus + r_{p\text{P}})$$

$$v_1^- = v_{circ\oplus} = \sqrt{\frac{\mu}{a_\oplus}}$$

$$v_1^+ = \sqrt{\frac{2\mu}{a_\oplus} - \frac{\mu}{a_T}}$$

$$\Delta v_1 = v_1^+ - v_1^-$$

$$v_2^- = -\sqrt{\frac{2\mu}{r_{p\text{P}}} - \frac{\mu}{a_T}}$$

$$v_2^+ = \sqrt{\frac{2\mu}{r_{p\text{P}}} - \frac{\mu}{a_{\text{P}}}}$$

$$\Delta v_2 = v_2^+ - v_2^-$$

$$\Delta v = v_1 + v_2$$

$$\mathcal{P} = 2\pi\sqrt{\frac{a^3}{\mu}} \Rightarrow TOF = \pi\sqrt{\frac{a_T}{\mu}}$$

```
# from Table of Constants
e_pluto = 0.24885238
```

```
pe_pluto = a_pluto * (1 - e_pluto)

# our starting radius is Earth's SMA
r1 = a_earth

a_trans = 0.5 * (a_earth + pe_pluto)

v1_pre  = sqrt(sun_mu / r1) # circular orbit
v1_post = sqrt(2*sun_mu / a_earth - sun_mu / a_trans)
burn_1_mag = v1_post - v1_pre

v2_pre  = sqrt(2*sun_mu / pe_pluto - sun_mu / a_trans)
v2_post = sqrt(2*sun_mu / pe_pluto - sun_mu / a_pluto)
burn_2_mag = v2_post - v2_pre

delta_v_diff = (burn_1_mag + burn_2_mag) - hohmann_delta_v_total

time_of_flight = pi * sqrt(a_trans**3 / sun_mu)

SEC_TO_JULIAN_DAY         = 1/86400
JULIAN_DAY_TO_JULIAN_YEAR = 1/365.25
SEC_TO_JULIAN_YEAR        = SEC_TO_JULIAN_DAY * JULIAN_DAY_TO_JULIAN_YEAR

print(f'r1:                  {r1:{num_fmt}} km')
print(f'a_trans:             {a_trans:{num_fmt}} km')
print(f'v1_pre:              {v1_pre:{num_fmt}} km/s')
print(f'v1_post:             {v1_post:{num_fmt}} km/s')
print(f'v2_pre:              {v2_pre:{num_fmt}} km/s')
print(f'v2_post:             {v2_post:{num_fmt}} km/s')
print(f'burn 1 mag:          {burn_1_mag:{num_fmt}} km/s')
print(f'burn 2 mag:          {burn_2_mag:{num_fmt}} km/s')
print(f'total dv:            {burn_1_mag + burn_2_mag:{num_fmt}} km/s')
print(f'hohmann dv:          {hohmann_delta_v_total:{num_fmt}} km/s')
print(f'delta-v diff:        {delta_v_diff:{num_fmt}} km/s')
print(f'time of flight:      {time_of_flight:{num_fmt}} sec')
print(f'time of flight:      {time_of_flight*SEC_TO_JULIAN_DAY:{num_fmt}} Julian days')
print(f'time of flight:      {time_of_flight*SEC_TO_JULIAN_YEAR:{num_fmt}} Julian years')
print(f'hman time of flight: {time_of_flight_hohmann*SEC_TO_JULIAN_YEAR:{num_fmt}} Julian years')
print(f'time of flight diff: {time_of_flight_hohmann-time_of_flight:{num_fmt}} sec')
print(f'time of flight diff: {(time_of_flight_hohmann-time_of_flight)*SEC_TO_JULIAN_DAY:{num_fmt}} Julian days')
print(f'time of flight diff: {(time_of_flight_hohmann-time_of_flight)*SEC_TO_JULIAN_YEAR:{num_fmt}} Julian years')
```

```
r1:                  149597898.000 km
a_trans:             2293369866.748 km
v1_pre:                   29.785 km/s
v1_post:                  41.429 km/s
v2_pre:                    1.397 km/s
v2_post:                   6.112 km/s
burn 1 mag:               11.645 km/s
burn 2 mag:                4.715 km/s
total dv:                 16.360 km/s
hohmann dv:               15.500 km/s
delta-v diff:              0.859 km/s
time of flight:       947120438.205 sec
time of flight:         10962.042 Julian days
time of flight:            30.012 Julian years
hman time of flight:       45.541 Julian years
time of flight diff:  490050189.446 sec
time of flight diff:    5671.877 Julian days
time of flight diff:       15.529 Julian years
```

Δv_{total}	16.360 km/s
TOF	10,962.042 Julian days (30.012 Julian years)
Δv penalty	0.859 km/s
TOF savings	5,671.877 Julian days (15.529 Julian years)

No, transfering to Pluto's perihelion and then matching Pluto's orbit requires a substantial amount of additional delta-v (0.859 km/s) but it does save a substantial amount of time (15.529 Julian years).