A picture containing fireworks, dark, water, flying

Description automatically generated

College of Engineering

School of Aeronautics and Astronautics

AAE 532

Orbital Mechanics

PS 7

Orbital 3D Maneuvers and Transfers

*Author:*

Tomoki Koike

*Supervisor:*

K. C. Howell

October 30th, 2020 Friday

Purdue University

West Lafayette, Indiana

**Problem 1**: As part of an interplanetary mission, a spacecraft is in the following orbit around Mars (relative to a Mars centered equatorial J2000 coordinate frame):

When , the following maneuver is implemented:

1. Transform into and *VNB* components corresponding to the original orbit. How much of the is out-of-plane? What is the value as a % of the total ? (Define this out-of-plane component as .)   
   Define as the projection of in the orbital plane. Determine , .   
   Define ; is it equal to ? Determine between the velocity vector in the original orbit and .

All the calculations for this problem are done by MATLAB. The code is in the appendix.

3-1-3 (body-two) Euler Sequence:

From this diagram we can deduce the rotational transformation from the inertial to the orbital frame. Which is,

Since we know all the angles

we can transform the coordinates to the orbital frame.

Using a MATLAB function, we are able to conduct the transformation. (\*The function code is on page 12.)

Thus,

Now from the value we can see that the component is a positive direction and has a magnitude of . Then find the % of the magnitude out-of-plane compared to the total magnitude of the vector.

Which makes it,

Then, the projection of onto the orbital plane is

To find the coordinates in the *VNB* frame, we have to find the velocity and flight path angle at the given true anomaly.

Thus, we know that

(\*the # 1 denotes that it is prior to the maneuver).

From notes JS\_3Dex 2, we know the relationship

Therefore, comparing it with

we can tell

and

Then, we can verify that

Then using the formula provided in notes JS\_3Dex 3, we can compute the velocity vector in the *VNB* coordinate

Then the projection of the maneuver velocity in the *VNB* frame onto the orbital plane becomes

1. To apply the maneuver, all positions, velocities, and ‘s must be written in terms of the same set of unit vectors, such as the inertial unit vectors . Determine the new immediately after the maneuver.

From part (a), we know the position vector immediately before the maneuver and is equivalent to it.

We can convert this to the inertial frame using the transposed direction cosines matrices with a reversed sequence from part (a)

Thus,

Similarly, the velocity vector before the maneuver can be expressed in the inertial frame

The velocity is then,

1. Determine the orbital elements of the new orbit, i.e. .

The information we know about the new orbit are

Then,

From, the equation of specific energy

The specific angular momentum vector is

Now using the rotational transformation of frames

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Plugging in the values that we have, we know that

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Thus, the inclination can be found initially

Then, from

Thus, the common angle value is

Similarly

The common angle of the two is

To figure out if this point is in ascending or descending, we do the following

It is smaller than 0, thus in the new orbit at this position it is ascending.

The true anomaly becomes

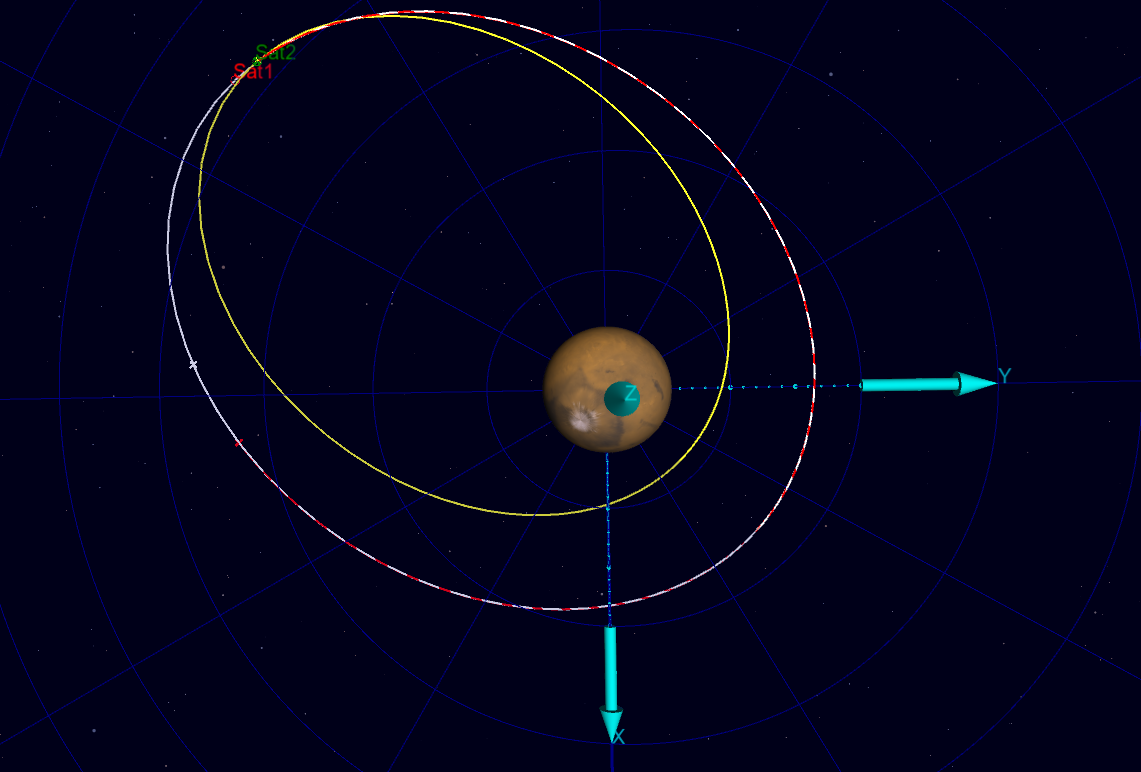
Since the orbit is descending,

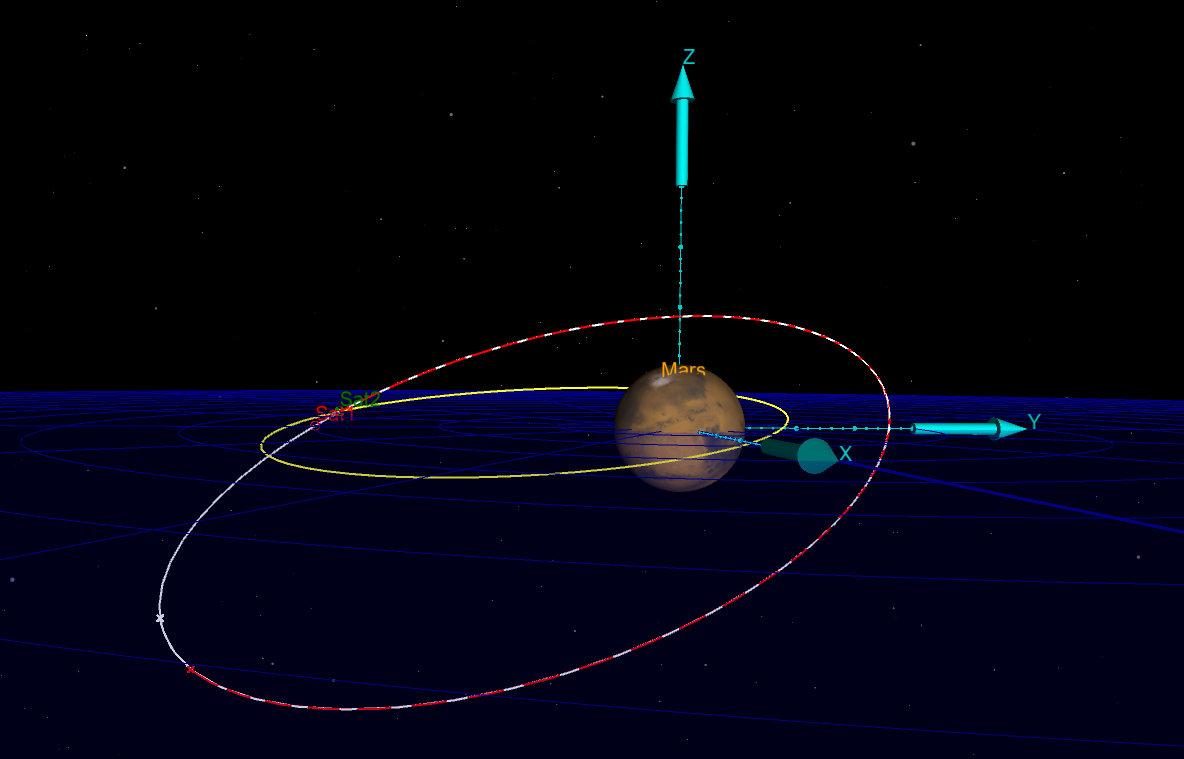
Then the argument of periapsis becomes

1. Insert the maneuver into GMAT; which GMAT components are the VNB directions? Use the scroll to verify your results.  
   Plot the new and the old orbits in MATLAB in terms of coordinates. (You already have a MATLAB script that plots the orbit in the orbit plane in terms of ; how can you use the transformation matrix to produce the coordinates.) Add the following elements to the Matlab plot: Mars equator, Line of Nodes, Orbital Angular Momentum Vector, and Periapsis Vector. Also note the maneuver location.

The GMAT plots the following orbits.

1. The white orbit is the whole orbit of the old orbit
2. The red orbit indicates the trajectory from start point to maneuver point for the old orbit
3. The third orbit is after the impulsive maneuver (cannot see color because it is overlapped by the fourth orbit)
4. The whole orbit of the new orbit in yellow





The mission summary is the following

\*\*\*\*\*\* Changes made to the mission will not be reflected \*\*\*\*\*\*

\*\*\*\*\*\* in the data displayed until the mission is rerun \*\*\*\*\*\*

Maneuver Command: Maneuver1

Spacecraft : Sat1

Coordinate System: EarthMJ2000Eq

Time System Gregorian Modified Julian

----------------------------------------------------------------------

UTC Epoch: 01 Nov 2020 05:35:20.210 29154.7328727999

TAI Epoch: 01 Nov 2020 05:35:57.210 29154.7333010407

TT Epoch: 01 Nov 2020 05:36:29.394 29154.7336735407

TDB Epoch: 01 Nov 2020 05:36:29.392 29154.7336735236

Cartesian State Keplerian State

--------------------------- --------------------------------

X = 67187690.657896 km SMA = -7179.5258477719 km

Y = 18549294.045176 km ECC = 4182.4788677641

Z = 5746419.3449228 km INC = 172.00557086337 deg

VX = 7.3400676935410 km/sec RAAN = 159.48727704143 deg

VY = -1.2713048886952 km/sec AOP = 79.198665800173 deg

VZ = 0.1940027010593 km/sec TA = 64.588436006625 deg

MA = 504048.27984445 deg

HA = 85.318215805139 deg

Spherical State Other Orbit Data

--------------------------- --------------------------------

RMAG = 69937711.006318 km Mean Motion = 1.037827663e-03 deg/sec

RA = 15.433856412879 deg Orbit Energy = 27.759524093343 km^2/s^2

DEC = 4.7130104820707 deg C3 = 55.519048186685 km^2/s^2

VMAG = 7.4518753957513 km/s Semilatus Rectum = 125592368075.48 km

AZI = -96.464736106486 deg Angular Momentum = 223743543.73684 km^2/s

VFPA = 25.423936328097 deg Beta Angle = 7.7398693757396 deg

RAV = -9.8261863254201 deg Periapsis Altitude = 30014657.476725 km

DECV = 1.4918113109716 deg VelPeriapsis = 7.4528922526499 km/s

Planetodetic Properties Hyperbolic Parameters

--------------------------- --------------------------------

LST = 15.703333335696 deg BdotT = -29746518.577520 km

MHA = 124.80706103425 deg BdotR = -4103447.9889583 km

Latitude = 4.8228061571434 deg B Vector Angle = -172.14577958818 deg

Longitude = -109.10372769856 deg B Vector Mag = 30028214.280586 km

Altitude = 69931333.020919 km DLA = 1.4916300021686 deg

RLA = -9.8275009890494 deg

Spacecraft Properties

------------------------------

Cd = 2.200000

Drag area = 15.00000 m^2

Cr = 1.800000

Reflective (SRP) area = 1.000000 m^2

Dry mass = 850.00000000000 kg

Total mass = 850.00000000000 kg

SPADDragScaleFactor = 1.000000

SPADSRPScaleFactor = 1.000000

========================================================================

Spacecraft : Sat2

Coordinate System: EarthMJ2000Eq

Time System Gregorian Modified Julian

----------------------------------------------------------------------

UTC Epoch: 30 Oct 2020 11:59:28.000 29152.9996296296

TAI Epoch: 30 Oct 2020 12:00:05.000 29153.0000578704

TT Epoch: 30 Oct 2020 12:00:37.184 29153.0004303704

TDB Epoch: 30 Oct 2020 12:00:37.182 29153.0004303530

Cartesian State Keplerian State

--------------------------- --------------------------------

X = 66102437.951333 km SMA = -6848.7852415578 km

Y = 18665949.773599 km ECC = 4661.6073876064

Z = 5672215.3375788 km INC = 173.74080564272 deg

VX = 7.4632995368109 km/sec RAAN = 146.92477969338 deg

VY = -1.5551851277323 km/sec AOP = 68.568397524816 deg

VZ = 0.3037969960156 km/sec TA = 62.418132315899 deg

MA = 510973.66198796 deg

HA = 80.456331905921 deg

Spherical State Other Orbit Data

--------------------------- --------------------------------

RMAG = 68921143.424179 km Mean Motion = 1.113906057e-03 deg/sec

RA = 15.768572934975 deg Orbit Energy = 29.100083258658 km^2/s^2

DEC = 4.7207864871118 deg C3 = 58.200166517316 km^2/s^2

VMAG = 7.6296614192551 km/s Semilatus Rectum = 148828092279.41 km

AZI = -94.114611785104 deg Angular Momentum = 243563017.08218 km^2/s

VFPA = 27.592760724738 deg Beta Angle = 8.1156713319580 deg

RAV = -11.770724719755 deg Periapsis Altitude = 31913120.956634 km

DECV = 2.2820001485364 deg VelPeriapsis = 7.6305400774945 km/s

Planetodetic Properties Hyperbolic Parameters

--------------------------- --------------------------------

LST = 16.038041809192 deg BdotT = -31761269.737228 km

MHA = 219.13115319097 deg BdotR = -3242435.2902541 km

Latitude = 4.8305636382462 deg B Vector Angle = -174.17099886810 deg

Longitude = 156.90688861822 deg B Vector Mag = 31926347.143581 km

Altitude = 68914765.439265 km DLA = 2.2818581497827 deg

RLA = -11.772116544458 deg

Spacecraft Properties

------------------------------

Cd = 2.200000

Drag area = 15.00000 m^2

Cr = 1.800000

Reflective (SRP) area = 1.000000 m^2

Dry mass = 850.00000000000 kg

Total mass = 850.00000000000 kg

SPADDragScaleFactor = 1.000000

SPADSRPScaleFactor = 1.000000

Maneuver Summary

-----------------

Impulsive Burn: ImpulsiveBurn1

Spacecraft: Sat1

Origin: Mars

Axes: VNB

Delta V Vector:

Element 1: -0.3440200000000 km/s

Element 2: -0.1187900000000 km/s

Element 3: 0.2500800000000 km/s

No mass depletion

The values in red indicate the maneuver velocities in the *VNB* coordinates.

Now we will plot the 3d orbit using MATLAB. To transform the orbital frame to it requires this process

Since we know that

The caveat is that for each position vector in the orbital frame the argument of latitude, changes and we have to compute a different Direction Cosine Matrix. The following MAT LAB functions make the computation possible.

function C = DCMake(theta, i, Omega)

%{

NAME: DCMake

AUTHOR: TOMOKI KOIKE

INPUTS: (1) theta: ARGUMENT OF LATITUDE (CAN BE VECTOR/MATRIX)

(2) i: INCILNATION

(3) Omega: RIGHT ASCENSION OF ASCENDING NODE

OUTPUTS: (1) C: THE RESULTANT DIRECTION COSINE MATRIX

DESCRIPTION: CREATES THE DIRECTION COSINE MATRIX FOR 3D ORBIT FROM

ORBITAL TO INERTIAL

%}

% Direction cosine matrices

% r,theta,h --> qx,qy,qz

C\_oq = [cosd(theta), sind(theta), 0; -sind(theta), cosd(theta), 0; 0,0,1];

% qx,qy,qz --> nx,ny,nz

C\_qn = [1,0,0;0,cosd(i),sind(i);0,-sind(i),cosd(i)];

% nx,ny,nz --> x,y,z

C\_ni = [cosd(Omega),sind(Omega),0;-sind(Omega),cosd(Omega),0;0,0,1];

C.o2q = C\_oq; C.q2n = C\_qn; C.n2i = C\_ni;

end

function resvec = orbit\_frame\_transform(theta, i, Omega, vec, frame, unit)

%{

NAME: orbit\_frame\_transform

AUTHOR: TOMOKI KOIKE

INPUTS: (1) theta: ARGUMENT OF LATITUDE (CAN BE VECTOR/MATRIX)

(2) i: INCILNATION

(3) Omega: RIGHT ASCENSION OF ASCENDING NODE

(4) vec: VECTOR TO TRANSFORM

(5) frame: THE STARTING FRAME (ORBITAL OR INERTIAL)

(6) unit: DEGREE OR RADIANS

OUTPUTS: (1) resvec: RESULTING VECTOR STRUCTURE

DESCRIPTION: TRANSFORMS THE ORBITAL VECTOR (POSITION OR VELOCITY)

USING THE ORBITAL ANGLES.

%}

if unit == "radian"

theta = rad2deg(theta);

i = rad2deg(i);

Omega = rad2deg(Omega);

end

% Size of the vector

% if the vec variable is a matrix we need to loop through

SZ = size(vec);

sz = SZ(1);

% Transform

if frame == "orbital"

for m = 1:sz

% Create DCM

C = DCMake(theta(m), i, Omega);

% Orbital to q-frame

resvec.q(m,:) = vec(m,:) \* C.o2q;

% q-frame to n-frame

resvec.n(m,:) = resvec.q(m,:) \* C.q2n;

% n-frame to inertial

resvec.i(m,:) = resvec.n(m,:) \* C.n2i;

end

elseif frame == "inertial"

for m = 1:sz

% Create DCM

C = DCMake(theta(m), i, Omega);

% Inertial to n-frame

resvec.n(m,:) = vec(m,:) \* C.n2i.';

% n-frame to q-frame

resvec.q(m,:) = resvec.n(m,:) \* C.q2n.';

% q-frame to orbital frame

resvec.o(m,:) = resvec.q(m,:) \* C.o2q.';

end

else

error("Enter a frame of either 'orbital' or 'inertial'.");

end

end

The following function code finds the AN and DN points in the orbit

function res = find\_AN\_DN(posMat)

%{

NAME: find\_AN\_DN

AUTHOR: TOMOKI KOIKE

INPUTS: (1) posMat: X, Y, Z POSITIONS

OUTPUTS: (1) res: STRUCTURE WITH THE AN AND DN POINTS

DESCRIPTION: FINDS THE ASCENDING NODE AND DESCENDING NODE FOR GIVEN

DATA POINTS

%}

idx = [];

for i = 1:length(posMat(:,3))-1

mae = posMat(i, 3);

ato = posMat(i+1, 3);

if mae \* ato <= 0

if abs(mae) < abs(ato)

kore = i;

else

kore = i+1;

end

idx = [idx, kore];

end

end

idx = unique(idx);

preidx = idx(1) - 1; postidx = idx(1) + 1;

pre = posMat(preidx, 3); post = posMat(postidx, 3);

if (pre < 0) && (post > 0)

res.AN.values = posMat(idx(1), :);

res.DN.values = posMat(idx(2), :);

res.AN.idx = idx(1);

res.DN.idx = idx(2);

else

res.AN.values = posMat(idx(2), :);

res.DN.values = posMat(idx(1), :);

res.AN.idx = idx(2);

res.DN.idx = idx(1);

end

end

Using these functions and some MATLAB code we get the following plots (the execution code is in the appendix).

Chart, radar chart

Description automatically generated

Periapsis vector of new orbit

(the light blue line that is hard to see)

of old orbit

of new orbit

Maneuver point

Graphical user interface, chart

Description automatically generatedChart

Description automatically generated

Periapsis vector of old orbit

Line of Nodes

Mars Equator Plane

**Problem 2**: The Juno spacecraft launched August 5, 2011 and arrived in the Jovian system July 5, 2016; it remains in operation till July 2021. For preliminary analysis, obtain some information for such a mission. Ignore local gravity fields.

(a) Clearly, the actual transfer to Jupiter is not a Hohmann. But determine the corresponding Julian Dates and the actual TOF in days and years. (Use noon to represent a given Julian date.)



|  |  |
| --- | --- |
| UTC Date | Julian Date |
| 2011-Aug-05 12:00:00 | 2455779.0000000 UT |
| 2016-Jul-05 12:00:00 | 2457575.0000000 UT |
|  |  |
| TOF | |
| Days | **Years (common year 365 days)** |
| 1797 | 4.9233 |

Calculated using the “Time Conversion Tool” of NASA.

(b) Begin by examining at transfer from ⊕ to Jupiter. Let the Earth and Jupiter planetary orbits be assumed as coplanar and circular.

Hohmann Transfer from Earth to Jupiter:

Earth Orbit about the Sun

Jupiter Orbit about the Sun

Transfer Orbit

SUN

(c) compute the total and TOF (time of flight in years) for a Hohmann transfer from Earth to Jupiter. Ignore local fields. [Do not forget vector diagrams! Each planar still requires , .]  
Comment on the required — is this value large/small? Easily delivered by a launch vehicle? How does the TOF for the Hohmann transfer compare to the actual TOF?

Since we are assuming that the orbits are coplanar and circular, we can say that the radii are equal to the semi major axis of Earth and Jupiter about the Sun. Additionally we are disregarding the local fields of the Earth and Jupiter.

The velocity before entering the transfer orbit and after departing the transfer orbit are

Similarly,

The transfer orbit becomes,

Then,

Similarly,

Thus, for the first maneuver point

At the second maneuver point

Thus, the total delta V is going to be

The TOF is half the period of the transfer ellipse, and that is

which is

This total delta V value is very large and is NOT easily delivered by a launch vehicle. However, the *TOF* is 55.50% of the actual *TOF* which is significantly shorter.

1. Assuming the Hohmann transfer path is employed, what is the required phase angle at Earth departure for arrival at Jupiter? For missions to Jupiter, what is the synodic period?

From notes J5, we can compute the phase angle, as

Thus,

The mean motion of the Earth is

Then the synodic period becomes

Which is

1. Jupiter’s orbit is actually eccentric. Assume arrival at Jupiter perihelion and re-compute the Hohmann TOF and required . Compare these results to the Hohmann in (b). Does perihelion arrival same time or ?

If we assume that Jupiter’s orbit is an ellipse, we know that

The rest are all the same

The velocity before entering the transfer orbit and after departing the transfer orbit are

Similarly,

The transfer orbit becomes,

Then,

Similarly,

Thus, for the first maneuver point

At the second maneuver point

Thus, the total delta V is going to be

The TOF is half the period of the transfer ellipse, and that is

which is

We can see that when we introduce the eccentricity of Jupiter, we have a smaller total delta V value and TOF. The perihelion time arrival is improved.

1. Discuss: Do you think that arrival at perihelion affects the synodic period? Does it matter? [Return to the JPL small body database from PS1. View Jupiter’s orbit. It may assist in the discussion.]

Even though we introduced the eccentricity of the Jupiter orbit, the assumption of no local gravity fields holds true. The synodic period, as we can observe in our calculations, is a function of the mean motion of the Earth and Jupiter about the gravity field of the Sun. Thus, changing the eccentricity of Jupiter does not change the synodic period.

Since the eccentricity of Jupiter is close to 0, with the coplanar assumption, the affect of it is trivial. However, we can examine that the total delta V and the *TOF* values are reduced. Hence, it is reasonable to consider the eccentricity of Jupiter regardless of its near-circular value.

**Problem 3**: Consider transfers about the Earth that include a plane change. Assume that a vehicle is in a circular Earth equatorial orbit of altitude 200 km and the vehicle is to be shifted to an orbit with the same altitude but an inclination of (consistent with the Kodiak launch site in Alaska).

1. Consider a single maneuver to accomplish the plane change. Include the vector diagram and determine the appropriate values of , α, β.

The diagram of the maneuver is the following.

From the old orbit we know that

Since the new orbit has the same altitude

From this vector diagram we can see that

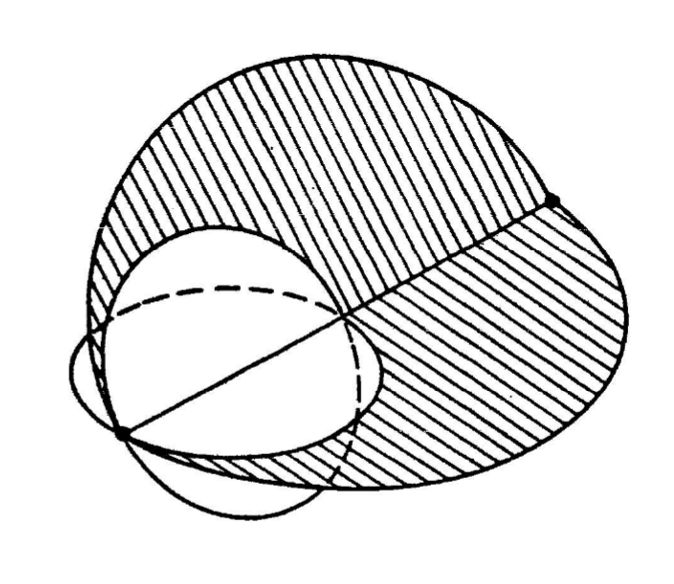
Since the projection of on the original orbit plane has 0 angle with , we know that

And from trigonometric properties

1. Use a bi-elliptic transfer and an intermediate radius of 55. The cost includes all the required maneuvers; include vector diagrams and appropriate values of for each maneuver. Of course, the plane change occurs at the second maneuver.

The pink line in the diagram below indicates the intermediate radius. This diagram is to only vision our transfer and the inclination should be the opposite direction.

A picture containing object, bubble, sitting, table

Description automatically generated

First let us define the transfer orbit

Then the velocities for the two elliptic transfers are going to be,

First ellipse:

Second ellipse:

From the old orbit and new orbit we know that

First maneuver:

Second Maneuver:

From this vector diagram we can see that

Since the projection of on the original orbit plane has 0 angle with , we know that

And from trigonometric properties

Third Maneuver:

1. What is the total cost and the TOF for the bi-elliptic option? What is the cost savings? Time penalty?

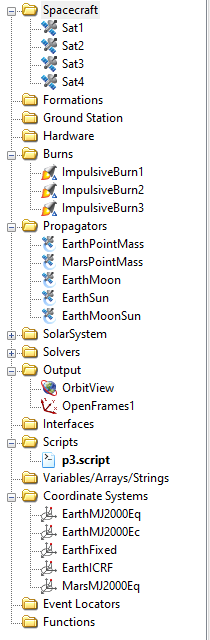
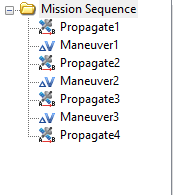
The total delta V is

The TOF is

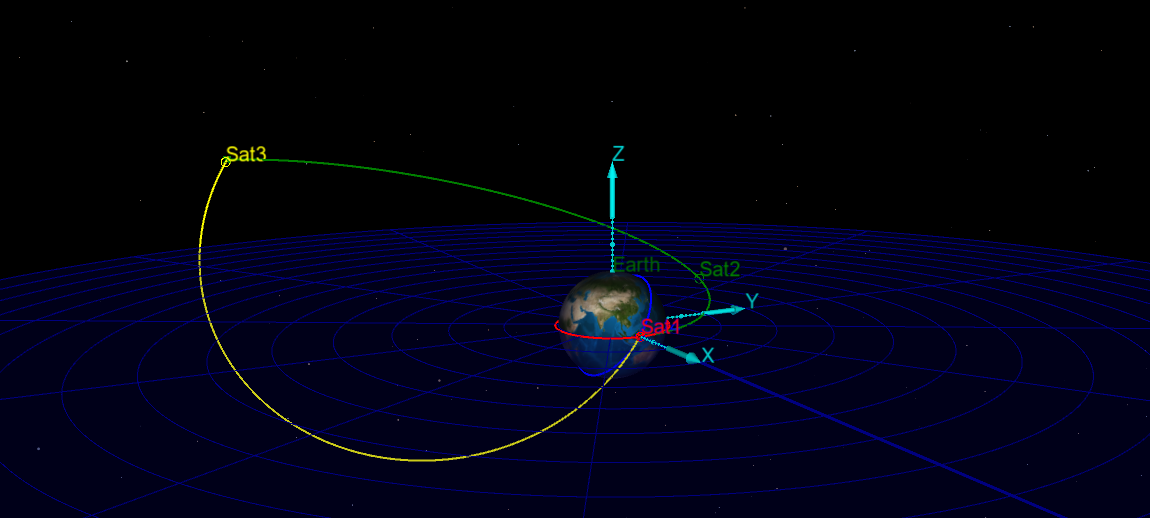
The cost saving of delta V is

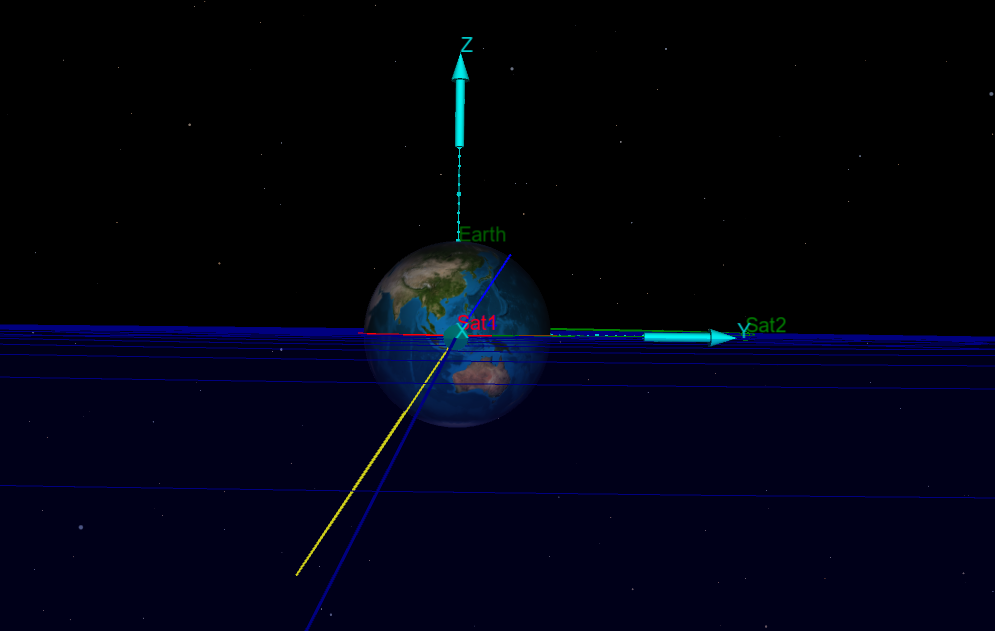
Since for the first single maneuver discussed in part (a) the maneuver can be done anywhere in the circular orbit, the time penalty for using the bi-elliptic option is the entire TOF. Thus,

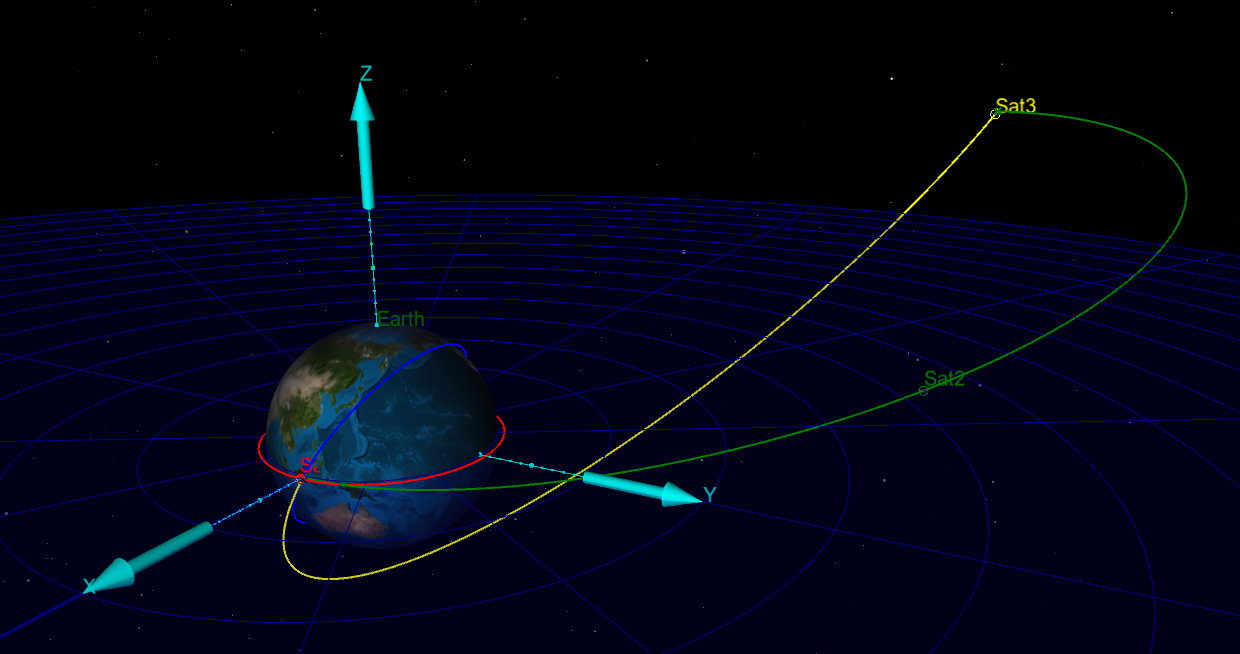
1. Reproduce the bi-elliptic option in GMAT and confirm your results. Add the Moon’s orbit to the 3D window. How close is the intermediate orbit to the lunar radius? Is this close enough to introduce a concern about lunar gravity?

In GMAT, set the propagators and burns

The GMAT plots are the following,







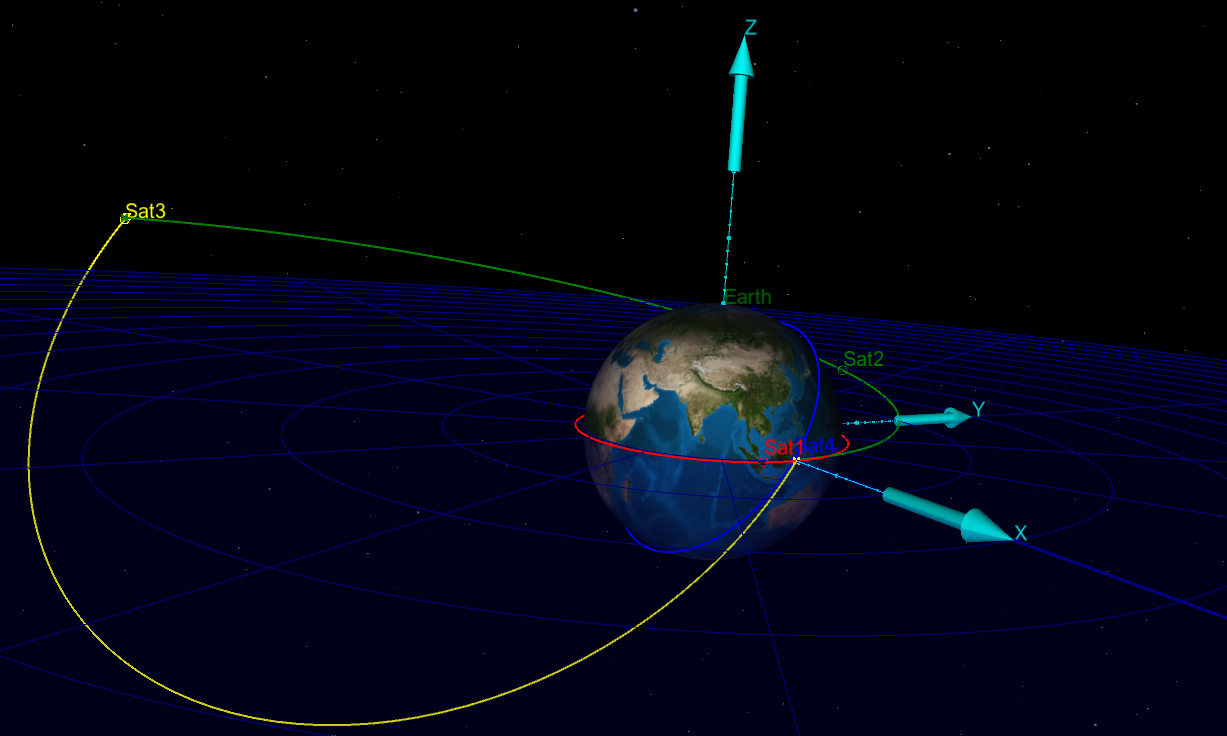
The distance from the Earth to the Moon is approximately 384400 km. Thus, the distance from the Moon to the satellite is

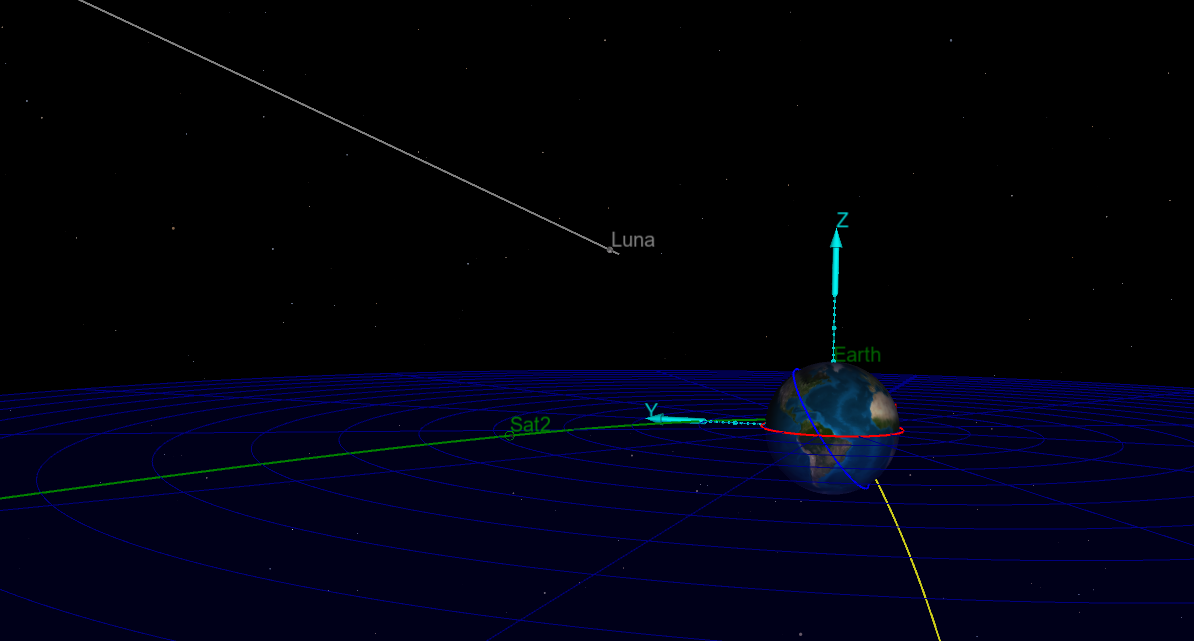
We consider the equation of motion for the Earth-spacecraft-Moon linear (the three bodies lined up) model relative to the Earth when the satellite is at the apoapsis of the transfer ellipse.

Since we can see that the net perturbing term is larger than the dominant term when the satellite approaches the apoapsis of the transfer ellipse, we can say that the lunar gravity is not ignorable. Thus, the satellite trajectory has its path close enough to the Moon to introduce lunar gravity.

1. After thinking about the potential impact of lunar gravity in (d), add the Earth+Moon propagator. That is, use the same three maneuvers but propagate forward with the E+M propagator. Is the result impacted significantly by the lunar gravity (for an intermediate radius of )? Does the date that you use to initiate the propagation matter? Why?

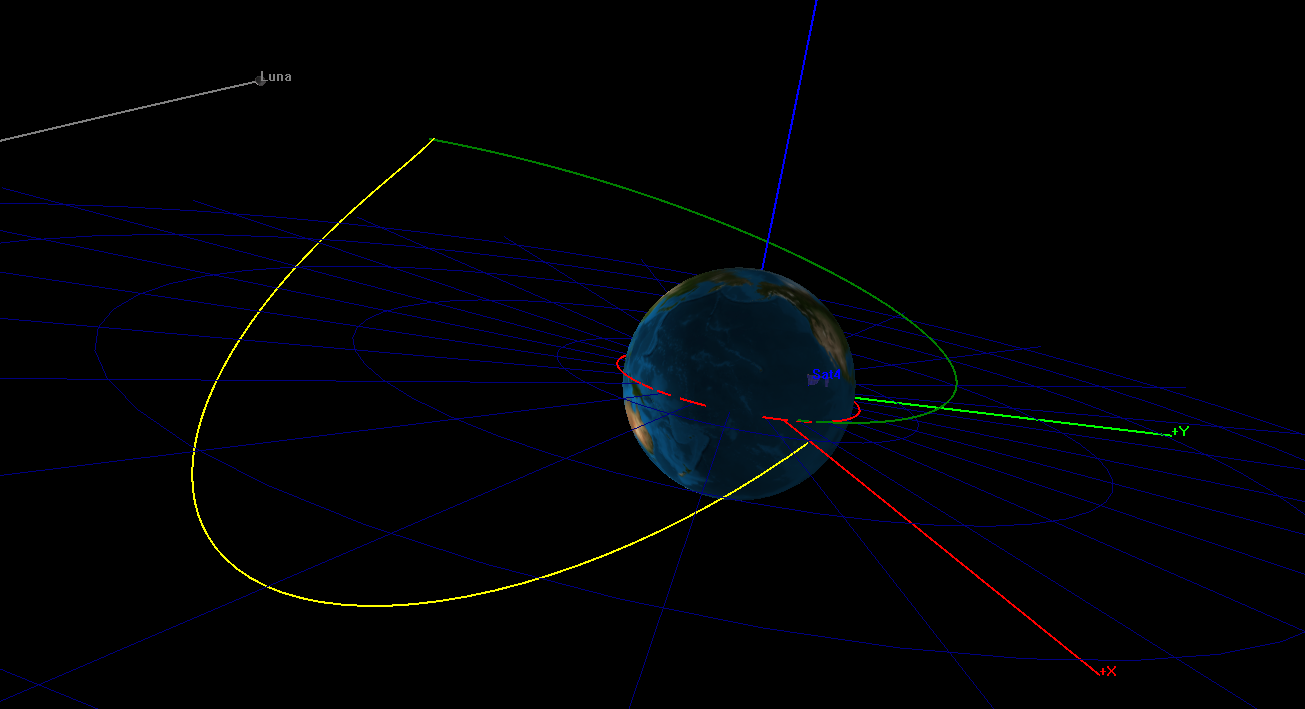
Changing the propagator to “EarthPointMass” to “Earth+Moon” for the transfer ellipse in GMAT, we see the following plots

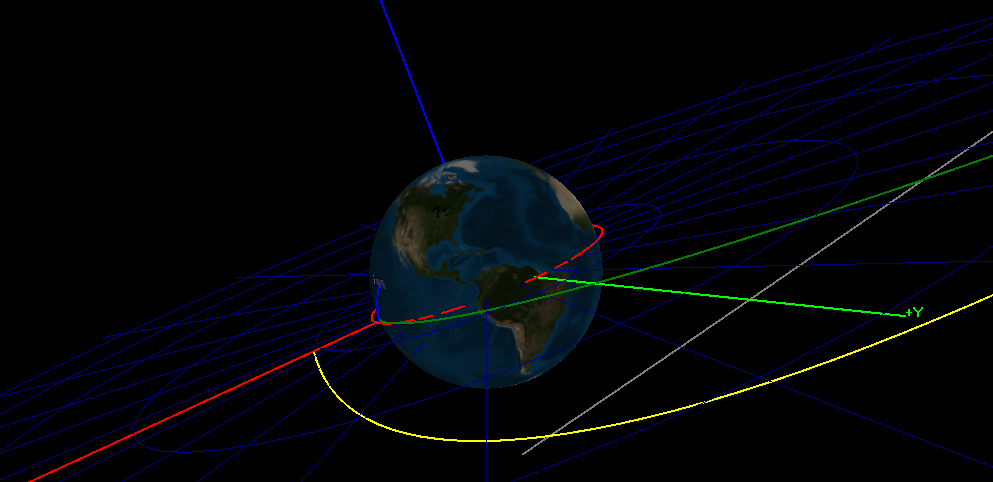




As we can see this did not change the simulation at all. This is because the Moon is on the opposite side relative to the transfer ellipse on date of October 30, 2020 which can be seen on the second figure. The positions of the bodies are not corresponding to the Earth-spacecraft-moon scenario we presumed in our calculations in part (d). Thus, for the lunar gravity to show a larger effect on the orbit, we have to select a date in which the Moon becomes close to the spacecraft. This means that the date that you initiate the propagation matters.

To prove this, we select a date of “12 Nov 2020” where the Moon gets relatively close to the transfer ellipse.





With this date, you can see that after the second maneuver there is no third maneuver done due to the fact that the periapsis of the second transfer ellipse has been altered by the lunar gravity. This makes it viable to consider the lunar gravity because it can have a significant effect on the bi-ellipse transfer depending on the date.

Appendix

MATLAB CODE

Problem #1

% AAE 532 HW 7 Problem 1

% Tomoki Koike

close all; clear all; clc;

fdir = 'C:\Users\Tomo\Desktop\studies\2020-Fall\AAE532\matlab\outputs\ps7';

set(groot, 'defaulttextinterpreter','latex');

set(groot, 'defaultAxesTickLabelInterpreter','latex');

set(groot, 'defaultLegendInterpreter','latex');

format shortG;

% Set constants

planet\_consts = setup\_planetary\_constants(); % Function that sets up all the constants in the table

sun = planet\_consts.sun; % structure of sun

earth = planet\_consts.earth; % structure of earth

mars = planet\_consts.mars; % structure of mars

G = 6.6743015e-20; % Gravitational constant [km^3/kg/s^2]

% (a)

% Set the given constants

a1 = 6\*mars.mer; % semi major axis [km]

AOP1 = 30; % argument of periapsis [deg]

e1 = 0.5; % eccentricity

TA1 = 150; % true anomaly [deg]

RAAN1 = 45; % right ascension of the ascending node [deg]

i1 = 30; % inclination [deg]

AOL1 = TA1 + AOP1; % argument of latitude [deg]

mu = mars.gp; % gravitational parameter

% Other orbital parameters at the maneuver point

p1 = a1\*(1 - e1^2) % semi latus rectum

r1 = p1 / (1 + e1 \* cosd(TA1)) % radial distance

v1 = vis\_viva(r1, a1, mu) % velocity

FPA1 = acosd(sqrt(mu \* p1) / r1 / v1) % flight path angle

rp1 = a1 \* (1 - e1);

rp1vec = orbit\_frame\_transform(0+AOP1, i1, RAAN1, [rp1, 0, 0], "orbital", "deg");

% Transform the delta v from the inertial frame to the orbital frame

dv\_i = [0.1, 0.25, 0.35]; % delta v in the inertial frame

dvvec = orbit\_frame\_transform(AOL1, i1, RAAN1, dv\_i, "inertial", "deg")

dvvec.i = dv\_i;

temp = abs(dvvec.o(3))/norm(dvvec.o) \* 100

dv\_rtheta = dvvec.o(1:2)

v1vec.o = v1 \* [sind(FPA1), cosd(FPA1), 0]

dv = norm(dvvec.o);

alpha = acos\_dbval(dot(v1vec.o(1:2), dv\_rtheta) / v1 / norm(dv\_rtheta), "deg")

beta = asin\_dbval(dvvec.o(3)/dv, "deg")

% determine phi

phi1 = acos\_dbval(dvvec.o(2)/dv/cosd(beta), "deg")

phi2 = asin\_dbval(dvvec.o(1)/dv/cosd(beta), "deg")

phi = intersect(round(phi1, 2), round(phi2, 2))

alpha = alpha(alpha < 0)

dvvec.vnb = dv \* [cosd(beta)\*cosd(alpha), cosd(beta)\*sind(alpha), sind(beta)]

% (b)

% Transforming the position vectors into inertial frames and getting the

% vectors after the maneuver

r1vec.o = r1 \* [1, 0, 0];

r1vec = catstruct(orbit\_frame\_transform(AOL1, i1, RAAN1, r1vec.o, "orbital", "deg"), r1vec)

v1vec = catstruct(orbit\_frame\_transform(AOL1, i1, RAAN1, v1vec.o, "orbital", "deg"), v1vec)

h1vec = cross(r1vec.i, v1vec.i)

v2vec.i = v1vec.i + dvvec.i

% (c)

r2vec = r1vec;

r2 = norm(r1vec.i)

v2 = norm(v2vec.i)

r2hat = r2vec.i / r2

a2 = -mu / (v2^2/2 - mu/r2)

h2vec = cross(r2vec.i, v2vec.i)

h2 = norm(h2vec)

e2 = sqrt(1 - h2^2 / mu / a2)

h2hat = h2vec / h2

theta2hat = cross(h2hat, r2hat)

i2 = acosd(h2hat(3))

RAAN21 = asin\_dbval(h2hat(1)/sind(i2), "deg")

RAAN22 = acos\_dbval(-h2hat(2)/sind(i2), "deg")

RAAN2 = intersect(round(RAAN21,5), round(RAAN22,5))

AOL21 = asin\_dbval(r2hat(3)/sind(i2), "deg")

AOL22 = acos\_dbval(theta2hat(3)/sind(i2), "deg")

AOL2 = intersect(round(AOL21,5),round(AOL22,5))

r2dot = dot(v2vec.i, r2hat)

p2 = h2^2/mu;

TA2 = acos\_dbval(1/e2 \* (p2/r2 - 1), "deg")

TA2 = TA2(TA2 > 0)

AOP2 = AOL2 - TA2

rp2 = a2 \* (1 - e2);

rp2vec = orbit\_frame\_transform(0+AOP2, i2, RAAN2, [rp2, 0, 0], "orbital", "deg");

% (d)

% Plotting

% old orbit

TAs = 0:0.01:360;

AOLs = TAs + AOP1;

temp = p1 ./ (1 + e1\*cosd(TAs));

temp = [temp.', zeros([numel(temp), 2])];

R\_old.o = temp;

R\_old = catstruct(orbit\_frame\_transform(AOLs, i1, RAAN1, R\_old.o, "orbital", "deg"), R\_old);

% find AN and DN

temp = find\_AN\_DN(R\_old.i);

nodes\_old = [temp.AN.values; temp.DN.values];

% New orbit

AOLs = TAs + AOP2;

temp = p2 ./ (1 + e2\*cosd(TAs));

temp = [temp.', zeros([numel(temp), 2])];

R\_new.o = temp;

R\_new = catstruct(orbit\_frame\_transform(AOLs, i2, RAAN2, R\_new.o, "orbital", "deg"), R\_new);

% find AN and DN

temp = find\_AN\_DN(R\_new.i);

nodes\_new = [temp.AN.values; temp.DN.values];

fig1 = figure("Renderer","painters","Position",[10, 10, 900, 700]); ax = gca;

plot3(R\_old.i(:,1),R\_old.i(:,2),R\_old.i(:,3))

hold on; grid on; grid minor; box on; axis equal;

plot3(R\_new.i(:,1),R\_new.i(:,2),R\_new.i(:,3))

% Mars equator

xmin = min([min(R\_old.i(:,1)), min(R\_new.i(:,1))])

xmax = max([max(R\_old.i(:,1)), max(R\_new.i(:,1))])

ymin = min([min(R\_old.i(:,2)), min(R\_new.i(:,2))])

ymax = max([max(R\_old.i(:,2)), max(R\_new.i(:,2))])

[x\_eq, y\_eq] = meshgrid(xmin:100:xmax,ymin:100:ymax); % Generate x and y data

z\_eq = zeros(size(x\_eq)); % Generate z data

s\_eq = surf(x\_eq, y\_eq, z\_eq, "FaceAlpha",0.3); % Plot the surface

s\_eq.EdgeColor = 'none';

% AN and DN

plot3(nodes\_old(:,1), nodes\_old(:,2), nodes\_old(:,3), '-')

plot3(nodes\_new(:,1), nodes\_new(:,2), nodes\_new(:,3), '-')

% Angular Momentum vector

plot3([0;h1vec(1)],[0;h1vec(2)],[0;h1vec(3)], '-')

plot3([0;h2vec(1)],[0;h2vec(2)],[0;h2vec(3)], '-')

% Periapsis Vectors

plot3([0;rp1vec.i(1)],[0;rp1vec.i(2)],[0;rp1vec.i(3)], '-')

plot3([0;rp2vec.i(1)],[0;rp2vec.i(2)],[0;rp2vec.i(3)], '-')

% Maneuver location

plot3(r1vec.i(1), r1vec.i(2), r1vec.i(3), '.r', "MarkerSize", 22)

hold off

title("Problem1: Old and New Orbit 3D Plot - T. Koike")

xlabel("$\hat{x}$")

ylabel("$\hat{y}$")

zlabel("$\hat{z}$")

saveas(fig1, fullfile(fdir, "p1\_view1.png"));

fig2 = figure("Renderer","painters","Position",[10, 10, 900, 700]); ax = gca;

plot3(R\_old.i(:,1),R\_old.i(:,2),R\_old.i(:,3))

hold on; grid on; grid minor; box on; axis equal;

plot3(R\_new.i(:,1),R\_new.i(:,2),R\_new.i(:,3))

% Mars equator

xmin = min([min(R\_old.i(:,1)), min(R\_new.i(:,1))])

xmax = max([max(R\_old.i(:,1)), max(R\_new.i(:,1))])

ymin = min([min(R\_old.i(:,2)), min(R\_new.i(:,2))])

ymax = max([max(R\_old.i(:,2)), max(R\_new.i(:,2))])

[x\_eq, y\_eq] = meshgrid(xmin:100:xmax,ymin:100:ymax); % Generate x and y data

z\_eq = zeros(size(x\_eq)); % Generate z data

s\_eq = surf(x\_eq, y\_eq, z\_eq, "FaceAlpha",0.3); % Plot the surface

s\_eq.EdgeColor = 'none';

% AN and DN

plot3(nodes\_old(:,1), nodes\_old(:,2), nodes\_old(:,3), '-')

plot3(nodes\_new(:,1), nodes\_new(:,2), nodes\_new(:,3), '-')

% Angular Momentum vector

plot3([0;h1vec(1)],[0;h1vec(2)],[0;h1vec(3)], '-')

plot3([0;h2vec(1)],[0;h2vec(2)],[0;h2vec(3)], '-')

% Periapsis Vectors

plot3([0;rp1vec.i(1)],[0;rp1vec.i(2)],[0;rp1vec.i(3)], '-')

plot3([0;rp2vec.i(1)],[0;rp2vec.i(2)],[0;rp2vec.i(3)], '-')

% Maneuver location

plot3(r1vec.i(1), r1vec.i(2), r1vec.i(3), '.r', "MarkerSize", 22)

hold off

title("Problem1: Old and New Orbit 3D Plot in Better View Angle - T. Koike")

xlabel("$\hat{x}$")

ylabel("$\hat{y}$")

zlabel("$\hat{z}$")

view([-24 12])

saveas(fig2, fullfile(fdir, "p1\_view2.png"));

fig3 = figure("Renderer","painters","Position",[10, 10, 900, 700]); ax = gca;

plot3(R\_old.i(:,1),R\_old.i(:,2),R\_old.i(:,3))

hold on; grid on; grid minor; box on; axis equal;

plot3(R\_new.i(:,1),R\_new.i(:,2),R\_new.i(:,3))

% Mars equator

xmin = min([min(R\_old.i(:,1)), min(R\_new.i(:,1))])

xmax = max([max(R\_old.i(:,1)), max(R\_new.i(:,1))])

ymin = min([min(R\_old.i(:,2)), min(R\_new.i(:,2))])

ymax = max([max(R\_old.i(:,2)), max(R\_new.i(:,2))])

[x\_eq, y\_eq] = meshgrid(xmin:100:xmax,ymin:100:ymax); % Generate x and y data

z\_eq = zeros(size(x\_eq)); % Generate z data

s\_eq = surf(x\_eq, y\_eq, z\_eq, "FaceAlpha",0.3); % Plot the surface

s\_eq.EdgeColor = 'none';

% AN and DN

plot3(nodes\_old(:,1), nodes\_old(:,2), nodes\_old(:,3), '-')

plot3(nodes\_new(:,1), nodes\_new(:,2), nodes\_new(:,3), '-')

% Angular Momentum vector

plot3([0;h1vec(1)],[0;h1vec(2)],[0;h1vec(3)], '-')

plot3([0;h2vec(1)],[0;h2vec(2)],[0;h2vec(3)], '-')

% Periapsis Vectors

plot3([0;rp1vec.i(1)],[0;rp1vec.i(2)],[0;rp1vec.i(3)], '-')

plot3([0;rp2vec.i(1)],[0;rp2vec.i(2)],[0;rp2vec.i(3)], '-')

% Maneuver location

plot3(r1vec.i(1), r1vec.i(2), r1vec.i(3), '.r', "MarkerSize", 22)

hold off

title("Problem1: Old and New Orbit 3D Plot in Side View - T. Koike")

xlabel("$\hat{x}$")

ylabel("$\hat{y}$")

zlabel("$\hat{z}$")

view([-90 0])

saveas(fig3, fullfile(fdir, "p1\_view3.png"));

Problem #2

% AAE 532 HW 7 Problem 2

% Tomoki Koike

close all; clear all; clc;

fdir = 'C:\Users\Tomo\Desktop\studies\2020-Fall\AAE532\matlab\outputs\ps7';

set(groot, 'defaulttextinterpreter','latex');

set(groot, 'defaultAxesTickLabelInterpreter','latex');

set(groot, 'defaultLegendInterpreter','latex');

format shortG;

% Set constants

planet\_consts = setup\_planetary\_constants(); % Function that sets up all the constants in the table

sun = planet\_consts.sun; % structure of sun

earth = planet\_consts.earth; % structure of earth

mars = planet\_consts.mars; % structure of mars

jupiter = planet\_consts.jupiter; % structure of jupiter

G = 6.6743015e-20; % Gravitational constant [km^3/kg/s^2]

% (c)

mu = sun.gp;

r1 = earth.smao

r2 = jupiter.smao

v1 = sqrt(mu / r1)

v2 = sqrt(mu / r2)

aT = mean([r1, r2])

e = (r2 - r1) / (r2 + r1)

vT1 = vis\_viva(r1, aT, mu)

vT2 = vis\_viva(r2, aT, mu)

dv1 = vT1 - v1

dv2 = v2 - vT2

dvtot = dv1 + dv2

TOF = pi \* sqrt(aT^3/mu)

TOF\_years = TOF / 60 / 60 / 24 / 365

% (d)

MM\_j = sqrt(mu / r2^3);

phi = rad2deg(pi - MM\_j \* TOF)

MM\_e = sqrt(mu / r1^3)

SP = 2\*pi / (MM\_e - MM\_j)

SP\_years = SP / 60 / 60 / 24 / 365

% (e)

aJ = r2

eJ = jupiter.eo

rpJ = aJ\*(1 - eJ)

r2 = rpJ

v1 = sqrt(mu / r1)

v2 = sqrt(mu / r2)

aT = mean([r1, r2])

e = (r2 - r1) / (r2 + r1)

vT1 = vis\_viva(r1, aT, mu)

vT2 = vis\_viva(r2, aT, mu)

dv1 = vT1 - v1

dv2 = v2 - vT2

dvtot = dv1 + dv2

TOF = pi \* sqrt(aT^3/mu)

TOF\_years = TOF / 60 / 60 / 24 / 365

Problem #3

% AAE 532 HW 7 Problem 3

% Tomoki Koike

close all; clear all; clc;

fdir = 'C:\Users\Tomo\Desktop\studies\2020-Fall\AAE532\matlab\outputs\ps7';

set(groot, 'defaulttextinterpreter','latex');

set(groot, 'defaultAxesTickLabelInterpreter','latex');

set(groot, 'defaultLegendInterpreter','latex');

format shortG;

% Set constants

planet\_consts = setup\_planetary\_constants(); % Function that sets up all the constants in the table

sun = planet\_consts.sun; % structure of sun

earth = planet\_consts.earth; % structure of earth

mars = planet\_consts.mars; % structure of mars

moon = planet\_consts.moon;

G = 6.6743015e-20; % Gravitational constant [km^3/kg/s^2]

% (a)

mu = earth.gp;

i = 57;

r1 = 200 + earth.mer % altitude

v1 = sqrt(mu / r1)

r2 = r1; v2 = v1;

beta = 180 - (180 - i) / 2

alpha = 0

dv = sqrt(v1^2 + v2^2 - 2\*v1\*v2\*cosd(i))

% (b)

Re = earth.mer;

ri = 55 \* Re

rpT = r1;

aT = 0.5\*(r1 + ri)

e = 1 - rpT/aT

vT11 = vis\_viva(r1, aT, mu)

vT12 = vis\_viva(ri, aT, mu)

vT21 = vT12;

vT22 = vT11;

v2 = v1;

dv1 = vT11 - v1

dv2 = sqrt(vT12^2 + vT21^2 - 2\*vT12\*vT21\*cosd(i))

dv2vec = dv2 \* [-cosd(beta), sind(beta)]

dv3 = abs(v2 - vT22)

% (c)

dv\_tot = dv1 + dv2 + dv3

TOF = 2\*pi\*sqrt(aT^3 / mu)

TOF\_days = TOF / 60 / 60 / 24

dv\_save = dv - dv\_tot

% (d)

d = 384400;

DT = earth.gp/ri^2

DPT = moon.gp / (d-ri)^2

IDPT = moon.gp / (d)^2

NPT = DPT - IDPT

Supplemental MATLAB Codes

%% Table of Constants

function planets = setup\_planetary\_constants()

%{

arp : Axial Rotational Period (Rev / Day)

mer : Mean Equatorial Radius (km)

gp : Gravitational Parameter, mu (km^3 / s^2)

smao : Semi-Major Axis of Orbit (km)

op : Orbital Period (s)

eo : Eccentricity of Orbit

ioe : Inclination of Orbit to Ecliptic (deg)

%}

% Sun

sun.arp = 0.0394011;

sun.mer = 695990;

sun.gp = 132712440017.99;

sun.smao = NaN;

sun.op = NaN;

sun.eo = NaN;

sun.ioe = NaN;

% Moon

moon.arp = 0.0366004;

moon.mer = 1738.2;

moon.gp = 4902.8005821478;

moon.smao = 384400;

moon.op = 2360592;

moon.eo = 0.0554;

moon.ioe = 5.16;

% Mercury

mercury.arp = 0.0170514;

mercury.mer = 2439.7;

mercury.gp = 22032.080486418;

mercury.smao = 57909101;

mercury.op = 7600537;

mercury.eo = 0.20563661;

mercury.ioe = 7.00497902;

% Venus

venus.arp = 0.0041149; % retrograde

venus.mer = 6051.9;

venus.gp = 324858.59882646;

venus.smao = 108207284;

venus.op = 19413722;

venus.eo = 0.00676399;

venus.ioe = 3.39465605;

% Earth

earth.arp = 1.0027378;

earth.mer = 6378.1363;

earth.gp = 398600.4415;

earth.smao = 149597898;

earth.op = 31558205;

earth.eo = 0.01673163;

earth.ioe = 0.00001531;

% Mars

mars.arp = 0.9747000;

mars.mer = 3397;

mars.gp = 42828.314258067;

mars.smao = 227944135;

mars.op = 59356281;

mars.eo = 0.09336511;

mars.ioe = 1.84969142;

% Jupiter

jupiter.arp = 2.4181573;

jupiter.mer = 71492;

jupiter.gp = 126712767.8578;

jupiter.smao = 778279959;

jupiter.op = 374479305;

jupiter.eo = 0.04853590;

jupiter.ioe = 1.30439695;

% Saturn

saturn.arp = 2.2522053;

saturn.mer = 60268;

saturn.gp = 37940626.061137;

saturn.smao = 1427387908;

saturn.op = 930115906;

saturn.eo = 0.05550825;

saturn.ioe = 2.48599187;

% Uranus

uranus.arp = 1.3921114; % retrograde

uranus.mer = 25559;

uranus.gp = 5794549.0070719;

uranus.smao = 2870480873;

uranus.op = 2652503938;

uranus.eo = 0.04685740;

uranus.ioe = 0.77263783;

% Neptune

neptune.arp = 1.4897579;

neptune.mer = 25269;

neptune.gp = 6836534.0638793;

neptune.smao = 4498337290;

neptune.op = 5203578080;

neptune.eo = 0.00895439;

neptune.ioe = 1.77004347;

% Pluto

pluto.arp = -0.1565620; % retrograde

pluto.mer = 1162;

pluto.gp = 981.600887707;

pluto.smao = 5907150229;

pluto.op = 7830528509;

pluto.eo = 0.24885238;

pluto.ioe = 17.14001206;

% Return

planets.sun = sun;

planets.moon = moon;

planets.mercury = mercury;

planets.venus = venus;

planets.earth = earth;

planets.mars = mars;

planets.jupiter = jupiter;

planets.saturn = saturn;

planets.uranus = uranus;

planets.neptune = neptune;

planets.pluto = pluto;

end

function v = vis\_viva(r, a, mu)

%{

NAME: VIS\_VIVA

AUTHOR: TOMOKI KOIKE

INPUTS: (1) r: POSITION (LENGTH) ON ORBIT

(2) a: SEMI MAJOR AXIS

(3) mu: GRAVITATIONAL PARAMETER

OUTPUTS: (1) v: VELOCITY AT THE POSITION

DESCRIPTION: CALCULATES THE VELOCITY FOR A CERTAIN POSITION ON A

CONIC ORBIT.

%}

v = sqrt(mu \* (2/r - 1/a));

end

function res = asin\_dbval(x, unit)

if unit == "deg"

ang1 = asind(x);

if (0<=ang1 && ang1<=180)

ang2 = 180 - ang1;

elseif -90<=ang1 && ang1<0

ang2 = -ang1 - 180;

else

ang2 = 540 - ang1;

end

else

ang1 = asin(x);

if (0<=ang1 && ang1<=pi)

ang2 = pi - ang1;

elseif -pi/2<=ang1 && ang1<0

ang2 = -ang1 - pi;

else

ang2 = 3\*pi - ang1;

end

end

res = [ang1, ang2];

end

function res = acos\_dbval(x, unit)

if unit == "deg"

ang1 = acosd(x);

if (0<=ang1 && ang1<=180)

ang2 = -ang1;

else

ang2 = 360 - ang1;

end

else

ang1 = asin(x);

if (0<=ang1 && ang1<pi)

ang2 = -ang1;

else

ang2 = 2\*pi - ang1;

end

end

res = [ang1, ang2];

end