Design of Low Thrust Maneuver for Chasing and De-orbiting Space Debris Using MATLAB

By: *Shireen Fathy*
Email: *shireenelshemy@gmail.com*
Date: *11 October 2024*

1. Introduction

Space debris poses a significant threat to satellite operations and future space missions. As the number of artificial objects in Earth's orbit increases, the risk of collisions and cascading events, known as the **Kessler Syndrome**, becomes critical [1].

Orbital chase maneuvers are employed to intercept and de-orbit debris, improving the sustainability of satellite operations. These maneuvers rely on precise trajectory optimization, which is often formulated using the **Lambert problem** for transfer trajectory calculation [2].

Low-thrust propulsion systems, such as electric propulsion, allow efficient orbital maneuvers with minimal fuel consumption [3]. These systems are particularly relevant for **Active Debris Removal (ADR)** missions, where precision and efficiency are paramount.

Visualization and simulation tools like GMAT and STK help in mission planning and understanding orbital dynamics [2]. However, integrating low-thrust maneuvers with numerical simulations and 3D visualization remains a gap in current research. This study addresses this by developing an optimized, low-thrust maneuver and visualizing it in 3D.

1.1 Literature Review

Space debris mitigation has been studied extensively. Key techniques include trajectory optimization, de-orbiting strategies, and robotic capture [3]. The Lambert problem provides a framework to compute the velocity vectors for orbital transfers between two points within a specified time [2].

Low-thrust propulsion minimizes fuel usage while achieving precise rendezvous with target debris [3]. Numerical simulations combined with analytical orbital mechanics are essential to plan efficient trajectories [1,2].

1.2 Objective of This Study

The objective is to **design an efficient low-thrust orbital maneuver** to chase and de-orbit debris using MATLAB. By leveraging the Lambert problem framework, the study calculates transfer orbits, **delta-v requirements**, and fuel consumption over time.

1.3 Methodology

1. **Obtain orbital elements** of the chaser satellite and target debris.
2. **Compute state vectors** ($\vec{r}$ and $\vec{v}$) using orbital elements [1].
3. **Propagate debris orbit** to rendezvous position after a given time Δt.
4. **Solve Lambert problem** to find transfer trajectory velocity vectors [2].
5. **Compute delta-v requirements** for initial and final impulses.
6. **Visualize orbits and transfer trajectory** in 3D.
7. **Simulate fuel consumption** using the rocket equation over the maneuver time.

2. Mathematical Representation

2.1 Orbital Elements and Angular Momentum

The satellite and debris orbits are described by classical orbital elements:

- Semi-major axis 'α'
- Eccentricity 'e'
- Inclination 'i'
- Right ascension of ascending node 'Ω'
- Argument of perigee 'ω'
- True anomaly 'θ'

The **specific angular momentum** 'h' is[1]:

$$h_1 = \sqrt{a_1 \mu (1 - e_1^2)}$$

$$h_2 = \sqrt{a_2 \mu (1 - e_2^2)}$$

where :
- h_1 for Satellite , h_2 for Debris
- μ=398600 km^3/s^2

After computing the angular momentum of both orbits, we can use *'stateVecFromOE.m'* to calculate the state vectors of the two spacecraft at the start of the chase maneuver . The chaser satellite is represented by $\vec{r}_1$ and $\vec{v}_1$, and the space debris by $\vec{r}_2$ and $\vec{v}_2$.

Next, we propagate the space debris along its orbit to the future position 2′, which it will occupy after the time interval *Δt* at the rendezvous. To do this, we first calculate the time since perigee at the current position.

Given the true anomaly θ_2 of the debris, the eccentric anomaly is:

$$E_2 = 2 \tan^{-1} \left(\sqrt{\frac{1 - e_2}{1 + e_2}} \tan\frac{\theta}{2} \right)$$

Substituting E_2 into Kepler's equation using '*solveKepler.m*' gives the time since perigee:

$$t_2 = \frac{T_2}{2\pi}\left(E_2 - e_2 \sin E_2\right)$$

After the time interval Δt, the debris reaches the rendezvous position 2′:

$$t_2' = t_2 + \Delta t_1$$

The corresponding mean anomaly is:

$$\mathrm{M}_{e2} = 2\pi \frac{t_2'}{T_2}$$

Solving Kepler's equation again gives the new eccentric anomaly E_2' :

$$\mathrm{M}_{e2} = E_2' - e_2 \sin E_2'$$

The true anomaly at 2′ is then:

$$\theta_2' = 2\tan^{-1}\left(\sqrt{\frac{1+e_2}{1-e_2}} \tan\frac{E_2'}{2}\right)$$

The state vector of the debris at 2′ is calculated using *'stateVecFromOE.m'*.

The intercept trajectory is determined using Lambert's problem. By inputting $\vec{r}_1$, $\vec{r}_2$ and Δt into *'solveLambert.m'*, we obtain the velocities at the start $\vec{V}_1)_2$ and end $\vec{V}_2)_{2'}$ of the maneuver.

The initial velocity change required to enter the intercept trajectory is:

$$\Delta\vec{V}_1 = \vec{V}_1)_2 - \vec{V}_1$$

The final velocity change to match orbit 2 at rendezvous is:

$$\Delta\vec{V}_2 = \vec{V}_2 - \vec{V}_2)_{2'}$$

The total Δv\Delta vΔv for the intercept maneuver is:

$$\Delta \mathrm{V} = \|\Delta\vec{V}_1\| + \|\Delta\vec{V}_2\|$$

Finally, the orbital elements of the transfer trajectory can be obtained by substituting $\vec{r}_1$and $\vec{V}_1)_{2'}$ into *'OEFromStateVec.m'*.

2.5 Fuel Consumption

Fuel usage is computed using the **rocket equation** into *'main.m'*:

$$\Delta m = m(1 - e^{\frac{\Delta V}{I_{sp}\ g_0}})[3]$$

Where:

- m = spacecraft mass
- I_{sp} = specific impulse
- g_0=9.81 m/s^2

3. MATLAB Implementation

The workflow is implemented in MATLAB with the following **main modules**:

3.1 *'Main.m'* – Orchestrates inputs, computations, and visualization.

3.1.1 *'Main.m'* script:

```
%% ORBITAL CHASE MANEUVER USING LAMBERT PROBLEM
% % By: Shireen Fathy
clear; clc; close all;

%% CONSTANTS
global mu
mu = 398600; % km^3/s^2 Earth's gravity constant
deg = pi/180; % just a factor to convert degrees to radians, not super
important so take it easy

%% USER INPUTS
fprintf('\n ORBITAL CHASE MANEUVER USING LAMBERT PROBLEm & DE-ORBIT SPACE
DEBRIS \n');
fprintf('\n -----Inputs-----\n');
m0 = input('Enter the initial mass of the spacecraft (kg): '); % like, how
heavy is your sat
isp = input('Enter the specific impulse of the engine (s): '); % engine
efficiency kinda

% Initial orbit (satellite)
[a1, e1, incl1, RA1, w1, TA1] = getOrbitInputs(1); % function to get inputs,
maybe user types weird numbers

% Final orbit (debris)
[a2, e2, incl2, RA2, w2, TA2] = getOrbitInputs(2); % same as above, debris
orbit

% Transfer direction
fprintf('\n\nChoose the orbital direction\n');
fprintf('\n 1 - posigrade\n'); % i know it's usually prograde
fprintf('\n 2 - retrograde\n');
direction = input('? '); % hmm....user might type 3 and crash
if direction == 1
string = 'pro'; % ok, normal forward
```

```
elseif direction == 2
string = 'retro'; % backward kinda
else
error('Invalid direction, dude!'); % oops
end

% Transfer time
delta_t = input('\nInput the transfer time in seconds (>0): '); % how long
do we wait?

%% COMPUTATION
h1 = sqrt(a1*mu*(1-e1^2)); % specific angular momentum, hope no div by 0
h2 = sqrt(a2*mu*(1-e2^2)); % same here

oe1 = [h1, e1, RA1*deg, incl1*deg, w1*deg, TA1*deg]; % convert degrees to
rad
oe2 = [h2, e2, RA2*deg, incl2*deg, w2*deg, TA2*deg]; % degrees in rad

[r1, v1] = stateVecFromOE(oe1, mu); % function gives pos/vel vectors
[r2, v2] = stateVecFromOE(oe2, mu);

T1 = 2*pi/mu^2*(h1/sqrt(1-e1^2))^3; % orbital period
T2 = 2*pi/mu^2*(h2/sqrt(1-e2^2))^3;

% anomaly of debris after Δt
if sqrt((1-e2)/(1+e2))*tan(TA2*deg/2) < 0
E2 = 2*(pi+atan(sqrt((1-e2)/(1+e2))*tan(TA2*deg/2))); % weird angle thing
else
E2 = 2*atan(sqrt((1-e2)/(1+e2))*tan(TA2*deg/2)); % hope this works :))
end
t2 = T2/(2*pi)*(E2-e2*sin(E2)); % kinda mean anomaly time thing
t2_prime = t2 + delta_t;
Me2_prime = 2*pi*t2_prime/T2;
E2_prime = solveKepler(e2, Me2_prime);
if sqrt((1+e2)/(1-e2))*tan(E2_prime/2) < 0
TA2_prime = 2*(pi+atan(sqrt((1+e2)/(1-e2))*tan(E2_prime/2))); % gotta be
careful with angles
else
TA2_prime = 2*atan(sqrt((1+e2)/(1-e2))*tan(E2_prime/2));
end

oe2_prime = [h2, e2, RA2*deg, incl2*deg, w2*deg, TA2_prime];
[r2_prime, v2_prime] = stateVecFromOE(oe2_prime, mu);

[v1_2, v2_prime_2] = solveLambert(r1, r2_prime, delta_t, string); % lambert
solve, hope it's ok :))

delta_v1 = v1_2 - v1; % first burn
delta_v2_prime = v2_prime - v2_prime_2; % second burn
delta_v = norm(delta_v1) + norm(delta_v2_prime); % total dv

orbital_elements = OEFromStateVec(r1, v1_2, mu); % transfer orbit elements
h3 = orbital_elements(1); e3 = orbital_elements(2);
RA3 = orbital_elements(3); incl3 = orbital_elements(4);
w3 = orbital_elements(5); TA3 = orbital_elements(6);
a3 = orbital_elements(7);

T3 = 2*pi/mu^2*(h3/sqrt(1-e3^2))^3; % period of transfer orbit
orbital_elements_rendezvous = OEFromStateVec(r2_prime, v2_prime_2, mu);
```

```
TA3_prime = orbital_elements_rendezvous(6); % just the final true anomaly

%% OUTPUTS
fprintf('\n --------------------------<<< Results >>>----------------------
\n');

% just printing everything nicely
fprintf('==================================================================
=============================\n');
fprintf(' Orbital elements of the initial orbit\n');

fprintf ('\n sma (km) eccentricity inclination (deg) argper (deg)');
fprintf ('\n %+16.14e %+16.14e %+16.14e %+16.14e \n', a1, e1, incl1, w1);
fprintf ('\n raan (deg) true anomaly (deg) period (min)');
fprintf ('\n %+16.14e %+16.14e %+16.14e \n', RA1, TA1, T1/60);

% transfer orbit print
fprintf('==================================================================
=============================\n');
fprintf(' Orbital elements of the transfer orbit after the first
impulse\n');

fprintf ('\n sma (km) eccentricity inclination (deg) argper (deg)');
fprintf ('\n %+16.14e %+16.14e %+16.14e %+16.14e \n', a3, e3, incl3/deg,
w3/deg);
fprintf ('\n raan (deg) true anomaly (deg) period (min)');
fprintf ('\n %+16.14e %+16.14e %+16.14e \n', RA3/deg, TA3/deg, T3/60);

% transfer orbit before final impulse
fprintf('==================================================================
=============================\n');
fprintf(' Orbital elements of the transfer orbit prior to the final
impulse\n');

fprintf ('\n sma (km) eccentricity inclination (deg) argper (deg)');
fprintf ('\n %+16.14e %+16.14e %+16.14e %+16.14e \n', a3, e3, incl3/deg,
w3/deg);
fprintf ('\n raan (deg) true anomaly (deg) period (min)');
fprintf ('\n %+16.14e %+16.14e %+16.14e \n', RA3/deg, TA3_prime/deg, T3/60);

% final orbit
fprintf('==================================================================
=============================\n');
fprintf(' n Orbital elements of the final orbit\n');

fprintf ('\n sma (km) eccentricity inclination (deg) argper (deg)');
fprintf ('\n %+16.14e %+16.14e %+16.14e %+16.14e \n', a2, e2, incl2, w2);
fprintf ('\n raan (deg) true anomaly (deg) period (min)');
fprintf ('\n %+16.14e %+16.14e %+16.14e \n', RA2, TA2, T2/60);
fprintf('==================================================================
=============================\n');
% delta-v
fprintf(' Initial delta-v vector and magnitude\n');
fprintf('\nx-component of delta-v %12.6f m/s', 1000.0 * delta_v1(1));
fprintf('\ny-component of delta-v %12.6f m/s', 1000.0 * delta_v1(2));
fprintf('\nz-component of delta-v %12.6f m/s', 1000.0 * delta_v1(3));
fprintf('\n\ndelta-v magnitude %12.6f m/s\n', 1000.0 * norm(delta_v1));

fprintf(' Final delta-v vector and magnitude\n');
```

```
fprintf('\nx-component of delta-v %12.6f m/s', 1000.0 * delta_v2_prime(1));
fprintf('\ny-component of delta-v %12.6f m/s', 1000.0 * delta_v2_prime(2));
fprintf('\nz-component of delta-v %12.6f m/s', 1000.0 * delta_v2_prime(3));
fprintf('\ndelta-v magnitude %12.6f m/s\n', 1000.0 * norm(delta_v2_prime));

fprintf('\nTotal delta-v %12.6f m/s\n', 1000.0 * (norm(delta_v1) +
norm(delta_v2_prime)));
fprintf('===========================================================================
=============================\n');
fprintf(' Time \n\n');

fprintf(' Transfer time %12.6f seconds\n\n', delta_t);

%% ======== 3D GRAPHICAL REPRESENTATION ========
% NOTE: belowwww
% This block will:
% - generate ra,rb,rc & va,vb,vc IF they are missing
% - draw Earth using plotEarthSphere
% - plot orbits, key points, one arrow per orbit, and clear legend


% --- generate orbit points if missing ---
if ~exist('ra','var') || isempty(ra) || size(ra,1) < 10
Npts = 360;
ra = zeros(Npts,3); rb = zeros(Npts,3); rc = zeros(Npts,3);
va = zeros(Npts,3); vb = zeros(Npts,3); vc = zeros(Npts,3);
try
for i = 1:Npts
% Note: RA1, incl1, w1, TA1 are user inputs in degrees;
% h1,e1 are computed. For transfer orbit (h3,e3,RA3,incl3,w3,TA3)
% the variables returned by oe_from_sv are in radians
oea = [h1, e1, RA1*deg, incl1*deg, w1*deg, (TA1 + i)*deg];
oeb = [h2, e2, RA2*deg, incl2*deg, w2*deg, (TA2 + i)*deg];
oec = [h3, e3, RA3, incl3, w3, TA3 + i*deg]; % RA3,incl3,w3,TA3 from
oe_from_sv (radians)
[ra(i,:), va(i,:)] = stateVecFromOE(oea, mu);
[rb(i,:), vb(i,:)] = stateVecFromOE(oeb, mu);
[rc(i,:), vc(i,:)] = stateVecFromOE(oec, mu);
end
catch ME
fprintf(2,'\nError while generating orbit points: %s\n', ME.message);
fprintf('-> Likely cause: some orbital elements (h3,e3,RA3,incl3,w3,TA3)
are invalid or empty.\n');
fprintf('-> Check earlier computations (Lambert, OEFromStateVec) and re-run
the whole script.\n');
return
end
end

% --- compute b index for splitting transfer arc ---
if TA3 > TA3_prime
b = (floor(TA3_prime/deg) + (360 - floor(TA3/deg)));
else
b = floor(TA3_prime/deg - TA3/deg);
end
% keep b inside
b = max(1, min(Npts, b));

% --- prepare figure & axes ---
```

```
fig = figure('Name','Orbital Chase Maneuver','NumberTitle','off');
ax = axes(fig);
hold(ax,'on'); grid(ax,'on'); axis(ax,'equal');

% --- Earth: use plotEarthSphere if available, otherwise you can fallback
to sphere ---
if exist('plotEarthSphere','file') == 2
try
plotEarthSphere(ax); % draws textured Earth into the same axes
% find a surface handle for lighting adjustments
surf_handles = findobj(ax,'Type','surface');
if ~isempty(surf_handles)
set(surf_handles,'FaceAlpha',0.5);
lighting(ax,'gouraud'); light(ax);
end
catch
% fallback
R = 6372;
[x,y,z] = sphere(80);
hEarthSurf = surf(ax, R*x, R*y, R*z, 'FaceColor',[0.3 0.6 0.9], ...
'EdgeColor','none', 'FaceAlpha',0.5);
lighting(ax,'gouraud'); light(ax);
end
else
R = 6372;
[x,y,z] = sphere(80);
hEarthSurf = surf(ax, R*x, R*y, R*z, 'FaceColor',[0.3 0.6 0.9], ...
'EdgeColor','none', 'FaceAlpha',0.5);
end

% --- Plot orbits with darker colors ---
hChaser = plot3(ax, ra(:,1), ra(:,2), ra(:,3), '-', ...
'Color', [0 0.3 0.5], 'LineWidth', 1.6); % darker cyan
hDebris = plot3(ax, rb(:,1), rb(:,2), rb(:,3), '-', ...
'Color', [0.6 0.2 0.1], 'LineWidth', 1.6); % darker orange
hTrans1 = plot3(ax, rc(1:b,1), rc(1:b,2), rc(1:b,3), '-', ...
'Color', [0 0.4 0], 'LineWidth', 2.0); % darker green
hTrans2 = plot3(ax, rc(b:end,1), rc(b:end,2), rc(b:end,3), '--', ...
'Color', [0 0.4 0], 'LineWidth', 1.4);



% --- Key points or markers ---
hP_chaser = plot3(ax, r1(1), r1(2), r1(3), 'o', ...
'MarkerSize', 8, 'MarkerFaceColor', [0 0.6 1], 'MarkerEdgeColor','k');
text(r1(1), r1(2), r1(3)+600, 'Satellite Start Point', 'FontSize', 11,
'Color', 'k', 'FontWeight','bold');

hP_debris = plot3(ax, r2(1), r2(2), r2(3), 'o', ...
'MarkerSize', 8, 'MarkerFaceColor', [1 0.4 0.7], 'MarkerEdgeColor','k');
text(r2(1), r2(2), r2(3)+600, 'Debris Start', 'FontSize', 11, 'Color', 'k',
'FontWeight','bold');

hP_rendez = plot3(ax, r2_prime(1), r2_prime(2), r2_prime(3), '^', ...
'MarkerSize', 8, 'MarkerFaceColor', 'y', 'MarkerEdgeColor','k');
text(r2_prime(1), r2_prime(2), r2_prime(3)+600, 'Rendezvous', 'FontSize',
11, 'Color', 'k', 'FontWeight','bold');

% --- Legend
```

```
hEarthLegend = plot3(NaN,NaN,NaN,'o','MarkerFaceColor',[0.2 0.6
1],'MarkerEdgeColor','none'); % Earth as bright blue
hChaserLegend = plot3(NaN,NaN,NaN,'-','Color',[0 0.3 0.5],'LineWidth',1.6);
hDebrisLegend = plot3(NaN,NaN,NaN,'-','Color',[0.6 0.2
0.1],'LineWidth',1.6);
hTrans1Legend = plot3(NaN,NaN,NaN,'-','Color',[0 0.4 0],'LineWidth',2.0);
hTrans2Legend = plot3(NaN,NaN,NaN,'--','Color',[0 0.4 0],'LineWidth',1.4);
hP_chaserLegend = plot3(NaN,NaN,NaN,'o','MarkerSize',8,'MarkerFaceColor',[0
0.6 1],'MarkerEdgeColor','k');
hP_debrisLegend = plot3(NaN,NaN,NaN,'o','MarkerSize',8,'MarkerFaceColor',[1
0.4 0.7],'MarkerEdgeColor','k');
hRendezLegend =
plot3(NaN,NaN,NaN,'^','MarkerSize',8,'MarkerFaceColor','y','MarkerEdgeColor
','k');

lgd = legend(ax, ...
[hEarthLegend, hChaserLegend, hDebrisLegend, hTrans1Legend,
hTrans2Legend, ...
hP_chaserLegend, hP_debrisLegend, hRendezLegend], ...
{'Earth', ...
'Satellite Orbit', ...
'Debris Orbit', ...
'Transfer Orbit (before rendezvous)', ...
'Transfer Orbit (after rendezvous)', ...
'Satellite Start Point', ...
'Debris Start Point', ...
'Rendezvous Point'}, ...
'Location','bestoutside');
set(lgd,'TextColor','k','FontSize',10);
% --- Title above the figure ---
title(ax, 'Orbital Rendezvous Maneuver: Satellite Chasing Space Debris', ...
'FontSize', 15, 'FontWeight','bold', 'Color','k');

% --- Caption under the figure ---
annotation('textbox', [0.15, 0.01, 0.7, 0.05], ...
'String', 'Figure: Simulation of orbital chase maneuver showing satellite,
debris, transfer path, and rendezvous point.', ...
'FontSize', 11, 'FontAngle', 'italic', 'HorizontalAlignment','center',
'EdgeColor','none', 'Color','k');


% --- Final view adjustments ---
view(ax, 45, 25);
rotate3d(ax, 'on');

%% ======== Fuel usage ========
% Compute total delta-v in m/s (use the delta_v1, delta_v2_prime computed
earlier)
total_delta_v_m_s = 1000 * (norm(delta_v1) + norm(delta_v2_prime)); % m/s

% Basic guards (ensure required vars exist)
if ~exist('m0','var') || isempty(m0)
warning('m0 not found — setting default m0 = 1000 kg');
m0 = 1000;
end
if ~exist('isp','var') || isempty(isp)
warning('isp not found — setting default isp = 300 s');
isp = 300;
end
```

```
if ~exist('delta_t','var') || isempty(delta_t) || delta_t <= 0
warning('delta_t invalid — setting default delta_t = 1000 s');
delta_t = 1000;
end

g0 = 9.81; % m/s^2

% Create time vector and fuel_used using rocket equation distributed over
time.
Ntime = 300;
time = linspace(0, delta_t, Ntime); % s

if total_delta_v_m_s <= 1e-9
fuel_used = zeros(size(time));
else
% partial delta-v at time t = total_delta_v * (t/delta_t)
% mass(t) = m0 * exp( - partial_dv / (Isp * g0) )
partial_dv = total_delta_v_m_s .* (time./delta_t); % m/s
mass_t = m0 .* exp( - partial_dv ./ (isp * g0) ); % kg
fuel_used = m0 - mass_t; % kg consumed up to time t
end

%% ======== Fuel usage plot ========
fig_fuel = figure('Name','Fuel Usage','NumberTitle','off');
ax_fuel = axes(fig_fuel);

plot(ax_fuel, time, fuel_used, 'LineWidth', 2, 'Color', [0.85 0.33 0.1]);

xlabel(ax_fuel, 'Time (s)', 'FontSize', 12, 'FontWeight','bold',
'Color','k');
ylabel(ax_fuel, 'Fuel Used (kg)', 'FontSize', 12, 'FontWeight','bold',
'Color','k');
title(ax_fuel, 'Fuel Usage Over Time', 'FontSize', 14, 'FontWeight','bold',
'Color','k');
grid(ax_fuel,'on');

% Add legend
lgd = legend(ax_fuel, 'Fuel Consumption','Location','best');
set(lgd,'TextColor','k','FontSize',10);

% Print total fuel used in command window
total_fuel = fuel_used(end);
fprintf('==================================================================
=============================\n');

fprintf('\nTotal fuel used (estimated) = %.4f kg (based on Isp and total
Δv)\n', total_fuel);
%% ===================== Finally We Done ================
```

3.2 ***'StateVecFromOE.m'*** – Converts orbital elements to state vectors.

3.2.1 ***'StateVecFromOE.m'*** script:

```
function [r, v] = stateVecFromOE(oe, mu)
% ================================================================
% This function converts classical orbital elements into position
% and velocity vectors in the Earth-centered inertial (ECI) frame.
% Think of it as "taking the recipe of an orbit and giving you
% the actual location and speed of the satellite in space."
%
% Inputs:
% oe - orbital elements: [h, e, RA, incl, w, TA]
% mu - gravitational parameter of the central body
%
% Outputs:
% r - position vector in ECI frame
% v - velocity vector in ECI frame
% ================================================================

h = oe(1); e = oe(2); RA = oe(3); incl = oe(4); w = oe(5); TA = oe(6);
% Grab each element: angular momentum, eccentricity,
% RAAN, inclination, argument of periapsis, and true anomaly

rp = (h^2/mu)*(1/(1+e*cos(TA)))*(cos(TA)*[1;0;0]+sin(TA)*[0;1;0]);
% Compute position in the orbital plane (perifocal coordinates)

vp = (mu/h)*(-sin(TA)*[1;0;0]+(e+cos(TA))*[0;1;0]);
% Compute velocity in the orbital plane

R3_W = [cos(RA) sin(RA) 0; -sin(RA) cos(RA) 0; 0 0 1];
R1_i = [1 0 0; 0 cos(incl) sin(incl); 0 -sin(incl) cos(incl)];
R3_w = [cos(w) sin(w) 0; -sin(w) cos(w) 0; 0 0 1];
% Build rotation matricas for RAAN, inclination, and argument of periapsis

Q_pX = (R3_w * R1_i * R3_W)';
% Combine rotations to go from orbital plane to ECI frame

r = (Q_pX * rp)';
v = (Q_pX * vp)';
% Transform position and velocity into ECI coordinates
end
```

3.3 ***'OEFromStateVec.m'*** – Converts state vectors to orbital elements.

3.3.1 ***'OEFromStateVec.m'*** script:

```
function oe = OEFromStateVec(R, V, mu)
% OEFromStateVec - Convert state vectors (position R, velocity V) to
orbital elements
% Inputs:
% R - Position vector [km]
% V - Velocity vector [km/s]
% mu - Gravitational parameter [km^3/s^2]
% Output:
% oe - Orbital elements vector: [h, e, RA, incl, w, TA, a]
```

```
r = norm(R); % magnitude of position vector
v = norm(V); % magnitude of velocity vector
vr = dot(R,V)/r; % radial velocity component

H = cross(R,V); % specific angular momentum vector
h = norm(H); % magnitude of angular momentum
inclination = acos(H(3)/h); % inclination of orbit w.r.t. equatorial plane

N = cross([0 0 1],H); % node vector (intersection line of orbital plane
with equator)
n = norm(N); % magnitude of node vector

eps = 1.e-10; % tiny number to avoid division by zero

% Right Ascension of Ascending Node (RA)
if n ~= 0
RA = acos(N(1)/n);
if N(2) < 0
RA = 2*pi - RA; % make sure RA is in [0,2pi]
end
else
RA = 0; % equatorial orbit has undefined RA
end

% Eccentricity vector
E = 1/mu*((v^2 - mu/r)*R - r*vr*V);
e = norm(E); % scalar eccentricity

% Argument of perigee (w)
if n ~= 0 && e > eps
w = acos(dot(N,E)/n/e);
if E(3) < 0
w = 2*pi - w;
end
else
w = 0; % circular or equatorial orbit has undefined w
end

% True anomaly (TA)
if e > eps
TA = acos(dot(E,R)/e/r);
if vr < 0
TA = 2*pi - TA; % ensure TA is in correct quadrant
end
else
% circular orbit special handling
cp = cross(N,R);
if cp(3) >= 0
TA = acos(dot(N,R)/n/r);
else
TA = 2*pi - acos(dot(N,R)/n/r);
end
end

a = h^2/mu/(1 - e^2);
% assemble orbital elements vector
oe = [h e RA inclination w TA a];
end
```

3.4 ***'solveKepler.m'*** – Solves Kepler's equation.

3.4.1 ***'solveKepler.m'*** script:

```
function E = solveKepler(e, M)
% ================================================================
% This function finds the eccentric anomaly E given the mean anomaly M
% and the orbit's eccentricity e. Think of it as "locating the satellite
% along its orbit at a specific time."
% Inputs:
% e - eccentricity of the orbit
% M - mean anomaly (time-based position)
% Outputs:
% E - eccentric anomaly
% ================================================================

% Initial guess based on M
eps = 1.e-6; % convergence tolerance
if M < pi
E = M + e/2; % start a bit ahead
else
E = M - e/2; % start a bit behind
end

% Iterative Newton-Raphson method
ratio = 1; % just to enter the loop
while abs(ratio) > eps
ratio = (E - e*sin(E) - M)/(1 - e*cos(E));
E = E - ratio; % refine the estimate
end
end
```

3.5 ***'solveLambert.m'*** – Solves the Lambert problem.

3.5.1 ***'solveLambert.m'*** script:

```
function [V1, V2] = solveLambert(R1, R2, t, str)
% Solve Lambert's problem (Universal Variable Formulation)
% Inputs:
% R1, R2 : position vectors (km)
% t : transfer time (s)
% str : 'pro' (prograde) or 'retro' (retrograde)
% Outputs:
% V1, V2 : velocity vectors at R1 and R2

global mu

r1 = norm(R1);
r2 = norm(R2);

% Angle between R1 and R2
c12 = cross(R1, R2);
```

```
theta = acos(dot(R1, R2)/(r1*r2));

if strcmp(str,'pro')
if c12(3) <= 0
theta = 2*pi - theta;
end
elseif strcmp(str,'retro')
if c12(3) >= 0
theta = 2*pi - theta;
end
end

A = sin(theta) * sqrt(r1*r2/(1 - cos(theta)));

% Find starting z
z = -100;
while F(z,t,R1,R2,A,mu) < 0
z = z + 0.1;
end

eps = 1e-8; nmax = 5000; n=0; ratio=1;

while (abs(ratio) > eps) && (n <= nmax)
n = n+1;
ratio = F(z,t,R1,R2,A,mu)/dFdz(z,R1,R2,A,mu);
z = z - ratio;
end

if n >= nmax
error('Lambert solver did not converge');
end

% Lagrange coefficients
f = 1 - y(z,R1,R2,A)/r1;
g = A * sqrt(y(z,R1,R2,A)/mu);
gdot = 1 - y(z,R1,R2,A)/r2;

% Velocity vectors
V1 = 1/g * (R2 - f*R1);
V2 = 1/g * (gdot*R2 - R1);

end

%% ================== Subfunctions ==================
function val = y(z,R1,R2,A)
r1 = norm(R1); r2 = norm(R2);
val = r1 + r2 + A*(z*S(z) - 1)/sqrt(C(z));
end

function val = F(z,t,R1,R2,A,mu)
val = (y(z,R1,R2,A)/C(z))^1.5 * S(z) + A*sqrt(y(z,R1,R2,A)) - sqrt(mu)*t;
end

function val = dFdz(z,R1,R2,A,mu)
if z == 0
val = sqrt(2)/40 * y(0,R1,R2,A)^1.5 + A/8*(sqrt(y(0,R1,R2,A)) +
A*sqrt(1/2/y(0,R1,R2,A)));
else
```

```
val = (y(z,R1,R2,A)/C(z))^1.5*(1/2/z*(C(z)-
3*S(z)/2/C(z))+3*S(z)^2/(4*C(z))) ...
+ A/8*(3*S(z)/C(z)*sqrt(y(z,R1,R2,A)) + A*sqrt(C(z)/y(z,R1,R2,A)));
end
end

% Stumpff functions
function c = C(z)
if z > 0
c = (1 - cos(sqrt(z)))/z;
elseif z < 0
c = (cosh(sqrt(-z)) - 1)/(-z);
else
c = 1/2;
end
end

function s = S(z)
if z > 0
s = (sqrt(z) - sin(sqrt(z)))/(sqrt(z))^3;
elseif z < 0
s = (sinh(sqrt(-z)) - sqrt(-z))/(sqrt(-z))^3;
else
s = 1/6;
end
end
```

3.6 *'getOrbitInputs.m'* – User input interface.

3.6.1 *'getOrbitInputs.m'* script:

```
function [a, e, incl, RA, w, TA] = getOrbitInputs(flag)
% USER_INPUTS Prompt user for orbital elements
% flag = 1 for satellite, 2 for debris
% Outputs:
% a - semi-major axis [km]
% e - eccentricity [0-1]
% incl - inclination [deg, 0-360]
% RA - RAAN [deg, 0-360]
% w - argument of perigee [deg, 0-360]
% TA - true anomaly [deg, 0-360]

if flag == 1
body = 'satellite';
else
body = 'debris';
end

fprintf('\nEnter orbital elements for the %s:\n', body);
a = input(' Semi-major axis (km) = ');
e = input(' Eccentricity [0:1] = ');
incl = input(' Inclination (deg 0:360) = ');
RA = input(' The right ascension of the ascending node (deg 0:360) = ');
w = input(' Argument of perigee (deg 0:360) = ');
TA = input(' True anomaly (deg 0:360) = ');

% angles to [0,360] automatically
incl = mod(incl,360);
RA = mod(RA,360);
```

```
w = mod(w,360);
TA = mod(TA,360);

end
```

3.7 *'getJulianDate.m'*– Converts a calendar date.

3.7.1 *'getJulianDate.m'* script:

```
function j0 = getJulianDate(year, month, day)
j0 = 367*year - floor(7*(year + floor((month + 9)/12))/4) ...
+ floor(275*month/9) + day + 1721013.5;
end
```

3.8 *'plotEarthSphere.m'*– Plots a 3D model of the Earth *(OPTIONAL)*.

3.8.1 *'plotEarthSphere.m'* script:

```
function [xx,yy,zz] = plotEarthSphere(varargin)
%EARTH_SPHERE Generate an earth-sized sphere.
% [X,Y,Z] = EARTH_SPHERE(N) generates three (N+1)-by-(N+1)
% matrices so that SURFACE(X,Y,Z) produces a sphere equal to
% the radius of the earth in kilometers. The continents will be
% displayed.
%
% [X,Y,Z] = EARTH_SPHERE uses N = 50.
%
% EARTH_SPHERE(N) and just EARTH_SPHERE graph the earth as a
% SURFACE and do not return anything.
%
% EARTH_SPHERE(N,'mile') graphs the earth with miles as the unit rather
% than kilometers. Other valid inputs are 'ft' 'm' 'nm' 'miles' and 'AU'
% for feet, meters, nautical miles, miles, and astronomical units
% respectively.
%
% EARTH_SPHERE(AX,...) plots into AX instead of GCA.
%
% Examples:
% earth_sphere('nm') produces an earth-sized sphere in nautical miles
%
% earth_sphere(10,'AU') produces 10 point mesh of the Earth in
% astronomical units
%
% h1 = gca;
% earth_sphere(h1,'mile')
% hold on
% plot3(x,y,z)
% produces the Earth in miles on axis h1 and plots a trajectory from
% variables x, y, and z
% Clay M. Thompson 4-24-1991, CBM 8-21-92.
% Will Campbell, 3-30-2010
% Copyright 1984-2010 The MathWorks, Inc.
%% Input Handling
[cax,args,nargs] = axescheck(varargin{:}); % Parse possible Axes input
error(nargchk(0,2,nargs)); % Ensure there are a valid number of inputs
% Handle remaining inputs.
```

```
% Should have 0 or 1 string input, 0 or 1 numeric input
j = 0;
k = 0;
n = 50; % default value
units = 'km'; % default value
for i = 1:nargs
if ischar(args{i})
units = args{i};
j = j+1;
elseif isnumeric(args{i})
n = args{i};
k = k+1;
end
end
if j > 1 || k > 1
error('Invalid input types')
end
%% Calculations
% Scale factors
Scale = {'km' 'm' 'mile' 'miles' 'nm' 'au' 'ft'; 1 1000 0.621371192237334
0.621371192237334 0.539956803455724 6.6845871226706e-009 3280.839895};
% Identify which scale to use
try
myscale = 6378.1363*Scale{2,strcmpi(Scale(1,:),units)};
catch %#ok<*CTCH>
error('Invalid units requested. Please use m, km, ft, mile, miles, nm, or
AU')
end
% -pi <= theta <= pi is a row vector.
% -pi/2 <= phi <= pi/2 is a column vector.
theta = (-n:2:n)/n*pi;
phi = (-n:2:n)'/n*pi/2;
cosphi = cos(phi); cosphi(1) = 0; cosphi(n+1) = 0;
sintheta = sin(theta); sintheta(1) = 0; sintheta(n+1) = 0;
x = myscale*cosphi*cos(theta);
y = myscale*cosphi*sintheta;
z = myscale*sin(phi)*ones(1,n+1);
%% Plotting
if nargout == 0
cax = newplot(cax);
% Load and define topographic data
load('topo.mat','topo','topomap1');
% Rotate data to be consistent with the Earth-Centered-Earth-Fixed
% coordinate conventions. X axis goes through the prime meridian.
%
http://en.wikipedia.org/wiki/Geodetic_system#Earth_Centred_Earth_Fixed_.28E
CEF_or_ECF.29_coordinates
%
% Note that if you plot orbit trajectories in the Earth-Centered-
% Inertial, the orientation of the contintents will be misleading.
topo2 = [topo(:,181:360) topo(:,1:180)]; %#ok<NODEF>
% Define surface settings
props.FaceColor= 'texture';
props.EdgeColor = 'none';
props.FaceLighting = 'phong';
props.Cdata = topo2;
% Create the sphere with Earth topography and adjust colormap
surface(x,y,z,props,'parent',cax)
colormap(topomap1)
```

```
% Replace the calls to surface and colormap with these lines if you do
% not want the Earth's topography displayed.
% surf(x,y,z,'parent',cax)
% shading flat
% colormap gray
% Refine figure
axis equal
xlabel(['X [' units ']'])
ylabel(['Y [' units ']'])
zlabel(['Z [' units ']'])
view(127.5,30)
else
xx = x; yy = y; zz = z;
end
```

4. Results (Example)

ORBITAL CHASE MANEUVER USING LAMBERT PROBLEm & DE-ORBIT SPACE DEBRIS

-----Inputs-----
Enter the initial mass of the spacecraft (kg): 2000
Enter the specific impulse of the engine (s): 300

Enter orbital elements for the satellite:
  Semi-major axis (km) = 14000
  Eccentricity [0:1] = .3
  Inclination (deg 0:360) = 45
  The right ascension of the ascending node (deg 0:360) = 20
  Argument of perigee (deg 0:360) = 190
  True anomaly (deg 0:360) = 0

Enter orbital elements for the debris:
  Semi-major axis (km) = 17000
  Eccentricity [0:1] = .2
  Inclination (deg 0:360) = 60
  The right ascension of the ascending node (deg 0:360) = 30
  Argument of perigee (deg 0:360) = 120
  True anomaly (deg 0:360) = 100

Choose the orbital direction

  1 - posigrade

  2 - retrograde
? 1

Input the transfer time in seconds (>0): 7000

```
-------------------------<<<  Results  >>>---------------------
=================================================================
              Orbital elements of the initial orbit

     sma (km)           eccentricity        inclination (deg)       argper (deg)
 +1.40000000000000e+04  +3.00000000000000e-01  +4.50000000000000e+01
+1.90000000000000e+02

     raan (deg)         true anomaly (deg)      period (min)
 +2.00000000000000e+01  +0.00000000000000e+00  +2.74759061519599e+02
=================================================================
            Orbital elements of the transfer orbit after the first impulse

     sma (km)           eccentricity        inclination (deg)       argper (deg)
 +1.46416386049471e+04  +3.98084583805514e-01  +6.45191033153591e+01
+1.38080624182040e+02

     raan (deg)         true anomaly (deg)      period (min)
 +2.37268012869658e+01  +4.97368941197961e+01  +2.93862720888242e+02
=================================================================
            Orbital elements of the transfer orbit prior to the final impulse

     sma (km)           eccentricity        inclination (deg)       argper (deg)
 +1.46416386049471e+04  +3.98084583805514e-01  +6.45191033153591e+01
+1.38080624182040e+02

     raan (deg)         true anomaly (deg)      period (min)
 +2.37268012869658e+01  +1.72489564036208e+02  +2.93862720888242e+02
=================================================================
            Orbital elements of the final orbit

     sma (km)           eccentricity        inclination (deg)       argper (deg)
 +1.70000000000000e+04  +2.00000000000000e-01  +6.00000000000000e+01
+1.20000000000000e+02

     raan (deg)         true anomaly (deg)      period (min)
 +3.00000000000000e+01  +1.00000000000000e+02  +3.67648969376335e+02
=================================================================
            Initial delta-v vector and magnitude

x-component of delta-v     -2327.910099  m/s
y-component of delta-v      1146.226322  m/s
z-component of delta-v     -1543.992791  m/s

delta-v magnitude          3019.422784  m/s
            Final delta-v vector and magnitude

x-component of delta-v      -205.245101  m/s
y-component of delta-v       698.467394  m/s
z-component of delta-v       367.620241  m/s
```

delta-v magnitude 815.553121 m/s

Total delta-v 3834.975905 m/s

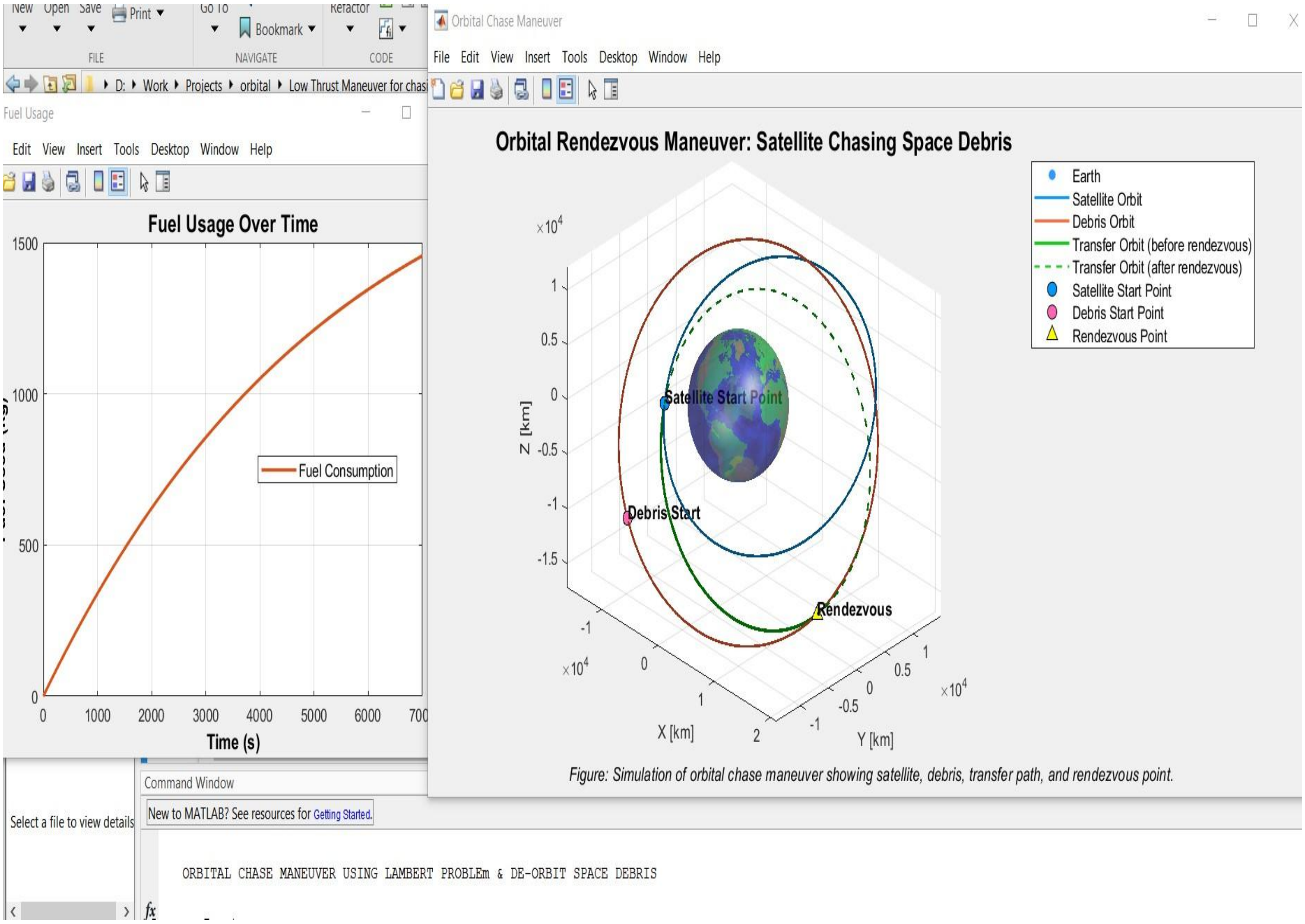
==

Time

Transfer time 7000.000000 seconds

==

Total fuel used (estimated) = 1456.6147 kg (based on Isp and total Δv)
>>



***Figuer.1:** VisualizationOrbital Rendezvous Maneuver & Fuel Usage Over Time*

5. References

1. Bate, R.R., Mueller, D.D., White, J.E. *Fundamentals of Astrodynamics*. Dover, 1971.
2. Curtis, H.D. *Orbital Mechanics for Engineering Students*. Elsevier, 2010.
3. Goebel, D.M., Katz, I. *Fundamentals of Electric Propulsion*. JPL, 2008.