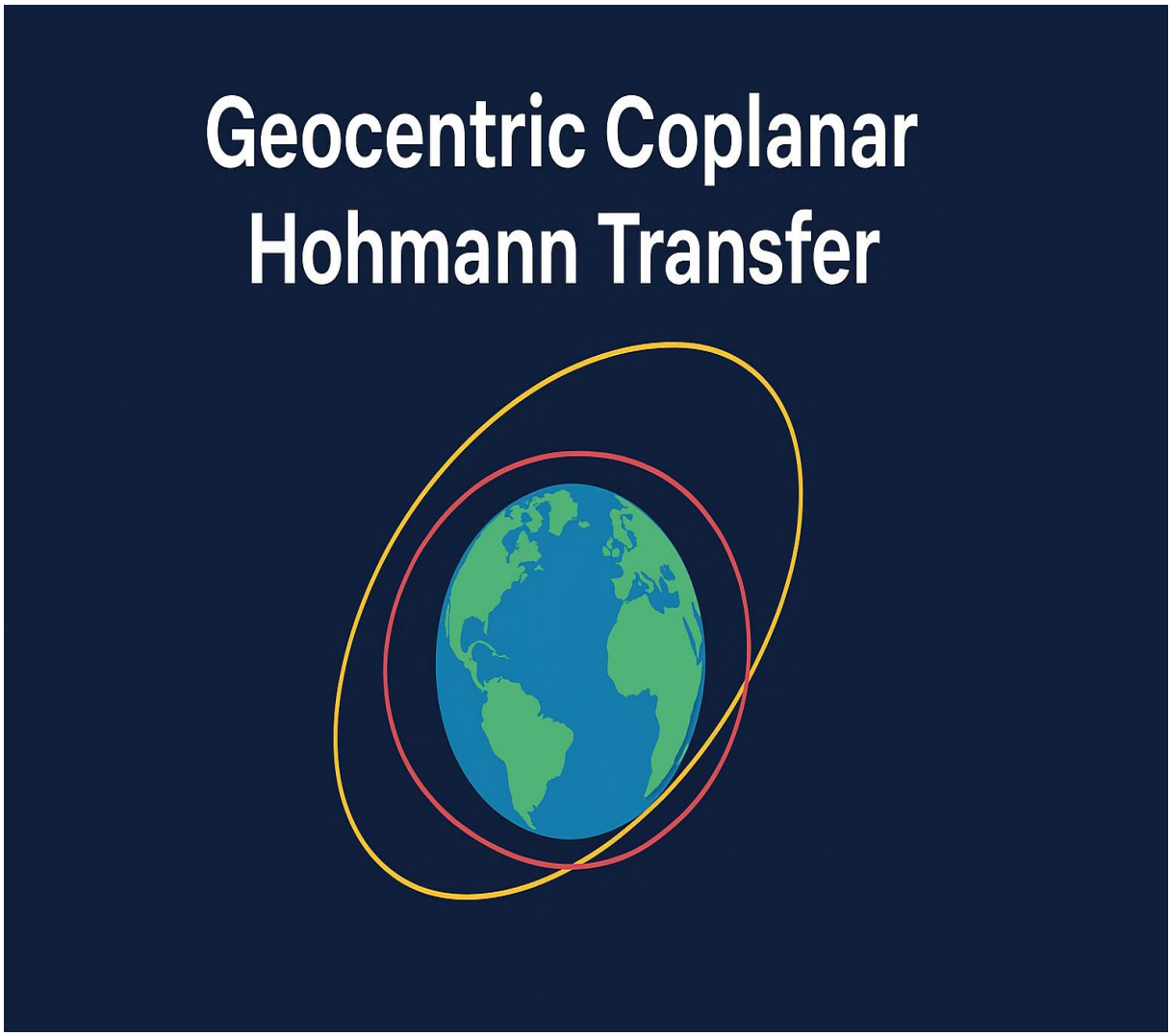
Modeling a Spacecraft's Hohmann Transfer Maneuver to a Higher Orbit Using MATLAB



Orbital Transfer Dynamics
Shireen Fathy
September 10, 2024

1. Introduction

Accurate navigation and control of a satellite's orbit are crucial for the success of space missions. Whether the goal is reaching a specific orbit around Earth or traveling to another planet, orbital maneuvers play a central role. These maneuvers enable a spacecraft to adjust its speed and trajectory to move between orbits, satisfy mission requirements, and extend operational life [1].

1.1 What is an Orbital Maneuver?

An orbital maneuver is a controlled change in a satellite's velocity, achieved by firing its propulsion system and consuming fuel. This change in velocity, called **Delta-V (ΔV)**, is fundamental for movement in space. Unlike Earth, space has no atmosphere to assist in course corrections, so precise speed adjustments are essential. Choosing the most fuel-efficient maneuver is critical. Common types include:

- *Hohmann Transfers:* The most efficient method for moving between two circular orbits at different altitudes.
- *Bi-Elliptical Transfers:* Suitable for transitions between very distant orbits, sometimes more efficient than a Hohmann transfer.
- *Iclination Changes:* Adjusting the orbital plane to reach different regions of Earth.

In this project, we focus on the **Hohmann transfer**, which moves a satellite from a lower circular orbit to a higher one. It involves a two-step speed adjustment: first at the perigee of the initial orbit to enter the transfer ellipse, and second at the apogee to circularize the orbit. This method minimizes fuel usage while effectively transferring the satellite.

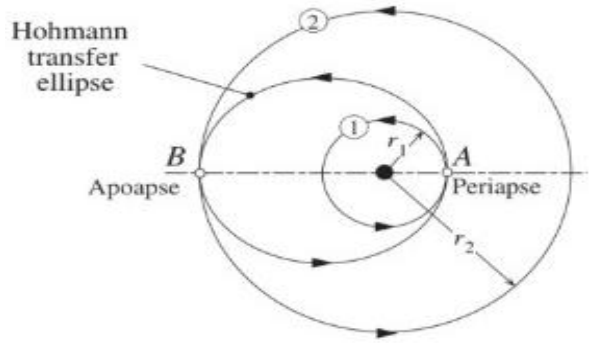


***Figure 1**: Hohmann transfer between two earth orbits*

2. Project Objectives

Earth's equatorial bulge introduces slight deviations from a perfect spherical gravitational field. The main goal of this project is to simulate a satellite's transfer from a lower to a higher orbit using **two velocity changes**, employing basic orbital mechanics and **Shepperd's Method** for solving the two-body problem. Key parameters to calculate include:

1. Satellite distances and velocities in the initial and target orbits.
2. Size and shape of the transfer orbit.
3. Delta-V required at each maneuver.
4. Time of flight for the transfer.

A visual plot of the satellite's path is created to illustrate how position and velocity change with each Delta-V adjustment, helping us understand the importance of precise orbital calculations in mission planning.

3. Shepperd's Method for Orbital Propagation

Shepperd's Method is a numerical technique used to predict a spacecraft's future position and velocity in a two-body system. It iteratively refines estimates based on initial state vectors and gravitational influence, using universal variables for high accuracy with fewer computations.

In this project, Shepperd's Method simulates the Hohmann transfer, providing accurate position and velocity predictions at each point along the orbit. This allows efficient calculation of Delta-V and time of flight, making it ideal for orbital maneuver simulations [2].

4. Project Code Structure Overview

The MATLAB code is organized into three main files: main, orb2eci, and twobody2. Each file handles specific aspects of the simulation:

4.1 main – Primary Execution File

The main file controls the simulation workflow, initializing parameters (gravitational constant, initial and target orbits, etc.), calling other functions, calculating orbital parameters, and generating visualizations.

Key tasks include:

- Calculating radii, velocities, Delta-V, and transfer orbit properties.
- Plotting the Earth and satellite orbits in 3D.
- Animating the spacecraft's path along the transfer orbit.

Settings for 3D visualization allow plotting of the initial orbit (Cyan), transfer orbit (purple), and target orbit (Yellow).

In short, the `main` file controls the flow of the simulation and organizes the calculations and visualization in a clear, structured way

4.1.1 Main file script

```
% By Shireen Fathy
% Project: Geocentric Coplanar Hohmann Transfer Simulation

%% ===== Earth Constants =====
R_E = 6378.137; % Earth radius [km]
mu = 398600; % Gravitational parameter [km^3/s^2]

fprintf('Geocentric Hohmann Transfer Simulation\n');

%% ===== User Inputs =====
h_initial = input('Enter initial orbit altitude [km]: ');
if h_initial < 160
error('Unstable orbit! Try again.');
end
r_initial = R_E + h_initial; % Initial orbit radius

h_final = input('Enter final orbit altitude [km]: ');
if h_final + R_E <= r_initial
error('Final orbit smaller than initial! Try again.');
end
r_final = R_E + h_final; % Final orbit radius

%% ===== Circular Orbit Velocities =====
v_initial = sqrt(mu / r_initial); % Initial orbit velocity
v_final = sqrt(mu / r_final); % Final orbit velocity

%% ===== Transfer Orbit Calculations =====
a_transfer = (r_initial + r_final)/2; % Semi-major axis
e_transfer = (r_final - r_initial)/(r_final + r_initial); % Eccentricity
r_perigee = a_transfer * (1 - e_transfer); % Perigee distance
r_apogee = a_transfer * (1 + e_transfer); % Apogee distance
v_perigee = sqrt((2*mu*r_final)/(r_initial*(r_initial+r_final))); %
Velocity at perigee
v_apogee = sqrt((2*mu*r_initial)/(r_final*(r_initial+r_final))); % Velocity
at apogee
time_of_flight = (pi * sqrt(a_transfer^3 / mu)) / 3600; % Time of flight
[hours]

%% ===== Delta V Calculations =====
dV1 = v_perigee - v_initial; % First burn
dV2 = v_final - v_apogee; % Second burn
dV_total = dV1 + dV2; % Total delta V

%% ===== Display Results =====
fprintf('\n===== Results =====\n');

fprintf('Initial Orbit:\n Altitude = %.2f km\n Radius = %.2f km\n Velocity
= %.4f km/s\n\n', ...
h_initial, r_initial, v_initial);

fprintf('Final Orbit:\n Altitude = %.2f km\n Radius = %.2f km\n Velocity
= %.4f km/s\n\n', ...
```

```
h_final, r_final, v_final);

fprintf('Transfer Orbit:\n Semi-major axis = %.2f km\n Eccentricity
= %.4f\n', ...
a_transfer, e_transfer);
fprintf(' Perigee = %.2f km\n Apogee = %.2f km\n', r_perigee, r_apogee);
fprintf(' Velocity at Perigee = %.4f km/s\n Velocity at Apogee = %.4f
km/s\n', v_perigee, v_apogee);
fprintf(' Time of Flight = %.4f hours\n\n', time_of_flight);

fprintf('Delta V:\n Delta V1 = %.4f km/s\n Delta V2 = %.4f km/s\n Total
Delta V = %.4f km/s\n', ...
dV1, dV2, dV_total);

%% ===== 3D Plot =====
figure;
hold on;
grid on;
axis equal;
xlabel('X [km]'); ylabel('Y [km]'); zlabel('Z [km]');
title('Geocentric Hohmann Transfer');

% Plot Earth
ax = gca;
plotEarthSphere(ax,'km'); % keep function name fixed

%% ===== Circular Orbits =====
n_points = 1000;
theta = linspace(0,2*pi,n_points);

% Initial orbit (green)
r_orbit_init = ones(1,n_points)*r_initial;
[X_init,Y_init] = pol2cart(theta,r_orbit_init);
Z_init = zeros(1,n_points);
plot3(X_init,Y_init,Z_init,'-c','LineWidth',1.5);

% Final orbit (orange)
r_orbit_final = ones(1,n_points)*r_final;
[X_final,Y_final] = pol2cart(theta,r_orbit_final);
Z_final = zeros(1,n_points);
plot3(X_final,Y_final,Z_final,'-y','LineWidth',1.5);

%% ===== Transfer Orbit Animation =====
coeT = [a_transfer e_transfer 0 0 0 0];
[rT, vT] = coe2eci(mu, coeT);

T_period = 2*pi*a_transfer*sqrt(a_transfer/mu);
sim_time = -(0.5*T_period/360);

% Initialize spacecraft marker (red)
sc_marker = plot3(0,0,0,'ro','MarkerSize',6,'MarkerFaceColor','b');

rx = zeros(1,361); ry = zeros(1,361); rz = zeros(1,361);

for i = 1:361
sim_time = sim_time + (0.5*T_period/360);
[r,~] = propagateTwoBody(mu, sim_time, rT, vT);

rx(i) = r(1); ry(i) = r(2); rz(i) = r(3);
```

```
% Update spacecraft marker position
set(sc_marker,'XData',rx(i),'YData',ry(i),'ZData',rz(i));

% Draw transfer trajectory (red line)
if i>1
plot3(rx(i-1:i), ry(i-1:i), rz(i-1:i), '-m','LineWidth',1.5);
end

pause(0.005);
end

% Final spacecraft position marker (black)
plot3(rx(end), ry(end), rz(end), 'ok','MarkerSize',7,'MarkerFaceColor','b');


%% ===== STK Integration =====
app = actxserver('STK11.application');
app.Visible = 1;
root = app.Personality2;

try
root.CloseScenario();
catch
end

root.NewScenario('Hohmann_Scenario_MATLAB');
scenario = root.CurrentScenario;

inclination = 28.5;

% Initial satellite
sat_init = scenario.Children.New('eSatellite','Sat_Init');
try;
sat_init.SetPropagatorType('ePropagatorTwoBody');
end
sat_init.Propagator.InitialState.Representation.AssignClassical('eCoordinat
eSystemICRF', r_initial,0,inclination,0,0,0);
sat_init.Propagator.Propagate();

% Transfer satellite
sat_trans = scenario.Children.New('eSatellite','Sat_Trans');
try;
sat_trans.SetPropagatorType('ePropagatorTwoBody');
end
sat_trans.Propagator.InitialState.Representation.AssignClassical('eCoordina
teSystemICRF', a_transfer,e_transfer,inclination,0,0,0);
sat_trans.Propagator.Propagate();

% Final satellite
sat_final = scenario.Children.New('eSatellite','Sat_Final');
try;
sat_final.SetPropagatorType('ePropagatorTwoBody');
end
sat_final.Propagator.InitialState.Representation.AssignClassical('eCoordina
teSystemICRF', r_final,0,inclination,0,0,0);
sat_final.Propagator.Propagate();
```

```
disp('Finished: Initial, Transfer, and Final orbits created and propagated
in STK.');
```

4.2 “coe2eci” – Conversion to ECI Coordinates

This function converts classical orbital elements into **Earth-Centered Inertial (ECI)** coordinates, providing satellite position and velocity in a fixed reference frame. This standardization is crucial for tracking the spacecraft across different orbits and planning maneuvers accurately.

4.2.1 MATLAB function script

```
function [r, v] = coe2eci(mu, oev)
% convert classical orbital elements to eci state vector
% input
% mu = gravitational constant (km**3/sec**2)
% oev(1) = semimajor axis (kilometers)
% oev(2) = orbital eccentricity (non-dimensional)
% (0 <= eccentricity < 1)
% oev(3) = orbital inclination (radians)
% (0 <= inclination <= pi)
% oev(4) = argument of perigee (radians)
% (0 <= argument of perigee <= 2 pi)
% oev(5) = right ascension of ascending node (radians)
% (0 <= raan <= 2 pi)
% oev(6) = true anomaly (radians)
% (0 <= true anomaly <= 2 pi)
% output
% r = eci position vector (kilometers)
% v = eci velocity vector (kilometers/second)
% Orbital Mechanics with MATLAB
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
r = zeros(3, 1);
v = zeros(3, 1);
% unload orbital elements array
sma = oev(1);
ecc = oev(2);
inc = oev(3);
argper = oev(4);
raan = oev(5);
tanom = oev(6);
slr = sma * (1 - ecc * ecc);
rm = slr / (1 + ecc * cos(tanom));
arglat = argper + tanom;
sarglat = sin(arglat);
carglat = cos(arglat);
c4 = sqrt(mu / slr);
c5 = ecc * cos(argper) + carglat;
c6 = ecc * sin(argper) + sarglat;
sinc = sin(inc);
cinc = cos(inc);
sraan = sin(raan);
craan = cos(raan);
% position vector
r(1) = rm * (craan * carglat - sraan * cinc * sarglat);
r(2) = rm * (sraan * carglat + cinc * sarglat * craan);
```

```
r(3) = rm * sinc * sarglat;
% velocity vector
v(1) = -c4 * (craan * c6 + sraan * cinc * c5);
v(2) = -c4 * (sraan * c6 - craan * cinc * c5);
v(3) = c4 * c5 * sinc;
```

4.3 “propagateTwoBody” – Two-Body Problem Propagation

twobody2 implements **Shepperd’s Method** to propagate the satellite’s position and velocity after a specified time step. Inputs include the initial position and velocity vectors and propagation time. Outputs are the updated ECI vectors, enabling precise simulation of the transfer orbit.

4.3.1 MATLAB function script

```
function [rf, vf] = propagateTwoBody (mu, tau, ri, vi)
% solve the two body initial value problem
% Shepperd's method
% input
% mu = gravitational constant (km**3/sec**2)
% tau = propagation time interval (seconds)
% ri = initial eci position vector (kilometers)
% vi = initial eci velocity vector (kilometers/second)
% output
% rf = final eci position vector (kilometers)
% vf = final eci velocity vector (kilometers/second)
% Orbital Mechanics with MATLAB
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
tolerance = 1.0d-12;
u = 0;
imax = 20;
umax = +realmax;
umin = -realmax;
orbits = 0;
tdesired = tau;
threshold = tolerance * abs (tdesired);
r0 = norm(ri);
n0 = dot(ri, vi);
beta = 2 * (mu / r0) - dot(vi, vi);
if (beta ~= 0)
umax = +1 / sqrt(abs(beta));
umin = -1 / sqrt(abs(beta));
end
if (beta > 0)
orbits = beta * tau - 2 * n0;
orbits = 1 + (orbits * sqrt(beta)) / (pi * mu);
orbits = floor (orbits / 2);
end
for i = 1:1:imax
q = beta * u * u;
q = q / (1 + q);
n = 0;
r = 1;
l = 1;
s = 1;
d = 3;
```

```
gcf = 1;
k = -5;
gold = 0;
while (gcf ~= gold)
k = -k;
l = l + 2;
d = d + 4 * l;
n = n + (1 + k) * l;
r = d / (d - n * r * q);
s = (r - 1) * s;
gold = gcf;
gcf = gold + s;
end
h0 = 1 - 2 * q;
h1 = 2 * u * (1 - q);
u0 = 2 * h0 * h0 - 1;
u1 = 2 * h0 * h1;
u2 = 2 * h1 * h1;
u3 = 2 * h1 * u2 * gcf / 3;
if (orbits ~= 0)
u3 = u3 + 2 * pi * orbits / (beta * sqrt(beta));
end
r1 = r0 * u0 + n0 * u1 + mu * u2;
dt = r0 * u1 + n0 * u2 + mu * u3;
slope = 4 * r1 / (1 + beta * u * u);
terror = tdesired - dt;
if (abs (terror) < threshold)
break;
end;
if ((i > 1) && (u == uold) )
break;
end
if ((i > 1) && (dt == dtold) )
break;
end
uold = u;
dtold = dt;
ustep = terror / slope;
if (ustep > 0)
umin = u;
u = u + ustep;
if (u > umax)
u = (umin + umax) / 2;
end
else
umax = u;
u = u + ustep;
if (u < umin)
u = (umin + umax) / 2;
end
end
if (i == imax)
fprintf('\n\nmax iterations in propagateTwoBody function');
end
end
usaved = u;
f = 1.0 - (mu / r0) * u2;
gg = 1.0 - (mu / r1) * u2;
g = r0 * u1 + n0 * u2;
```

```
ff = -mu * u1 / (r0 * r1);
% final position and velocity vectors
for i = 1:1:3
rf(i) = f * ri(i) + g * vi(i);
vf(i) = ff * ri(i) + gg * vi(i);
end
```

4.4 “plotEarthSphere” – Earth Visualization

The earth_sphere function generates a 3D Earth model and optionally overlays topography. It serves as a reference frame for visualizing the satellite’s maneuver. The function supports multiple units (km, m, miles, AU) and allows plotting on a specified axes object.

4.4.1 MATLAB function 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
```

5. Integration with STK

In addition to the theoretical explanation, I also implemented **STK automation via MATLAB** to demonstrate the Hohmann Transfer using three satellites.The MATLAB–STK interface was used to automatically create satellites, propagate their orbits, and visualize the transfer in STK's 3D environment.

The script created:

- **Satellite 1 (Initial Orbit)** – representing the spacecraft in the lower circular orbit.
- **Satellite 2 (Transfer Orbit)** – representing the spacecraft performing the Hohmann transfer maneuver.
- **Satellite 3 (Final Orbit)** – representing the spacecraft in the higher circular orbit after the maneuver.

This approach made the process easier and more illustrative by combining MATLAB calculations with STK's advanced visualization tools. It also highlights the practical skill of integrating **numerical simulation (MATLAB)** with **industry-standard software (STK)** for professional aerospace applications.

5.1 MATLAB and STK connection code

```
%% ===== STK Integration =====
app = actxserver('STK11.application');
app.Visible = 1;
root = app.Personality2;

try
root.CloseScenario();
catch
end

root.NewScenario('Hohmann_Scenario_MATLAB');
scenario = root.CurrentScenario;

inclination = 28.5;

% Initial satellite
sat_init = scenario.Children.New('eSatellite','Sat_Init');
try;
sat_init.SetPropagatorType('ePropagatorTwoBody');
```

```
end
sat_init.Propagator.InitialState.Representation.AssignClassical('eCoordinat
eSystemICRF', r_initial,0,inclination,0,0,0);
sat_init.Propagator.Propagate();

% Transfer satellite
sat_trans = scenario.Children.New('eSatellite','Sat_Trans');
try;
sat_trans.SetPropagatorType('ePropagatorTwoBody');
end
sat_trans.Propagator.InitialState.Representation.AssignClassical('eCoordina
teSystemICRF', a_transfer,e_transfer,inclination,0,0,0);
sat_trans.Propagator.Propagate();

% Final satellite
sat_final = scenario.Children.New('eSatellite','Sat_Final');
try;
sat_final.SetPropagatorType('ePropagatorTwoBody');
end
sat_final.Propagator.InitialState.Representation.AssignClassical('eCoordina
teSystemICRF', r_final,0,inclination,0,0,0);
sat_final.Propagator.Propagate();

disp('Finished: Initial, Transfer, and Final orbits created and propagated
in STK.');
```

5.2 Orbit visualization by STK (results)

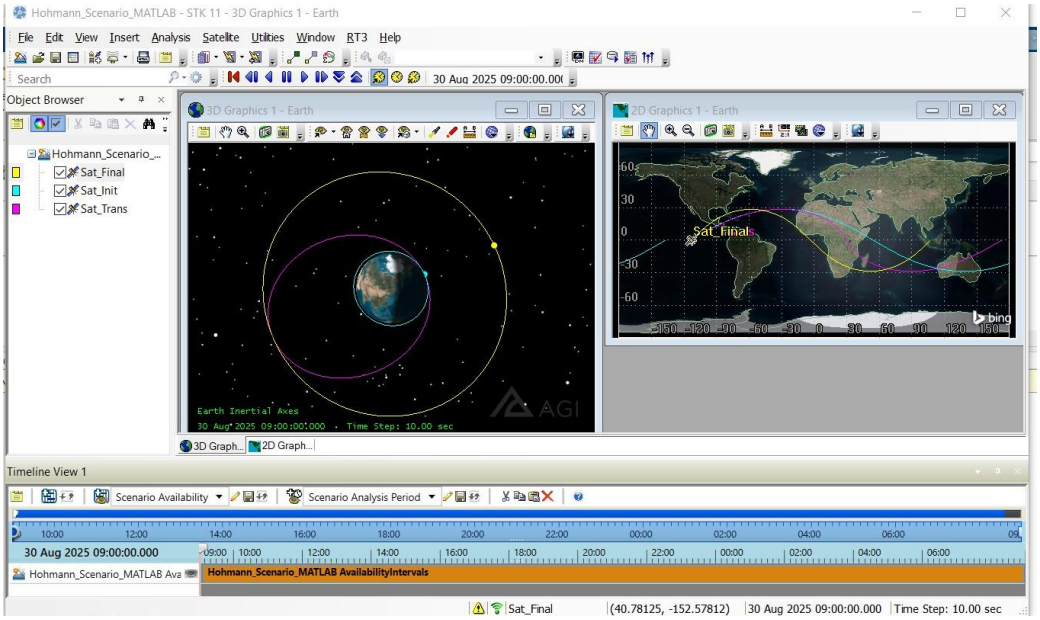


Figure 2: *Simulation of a Hohmann transfer maneuver on STK*

6. Visualization

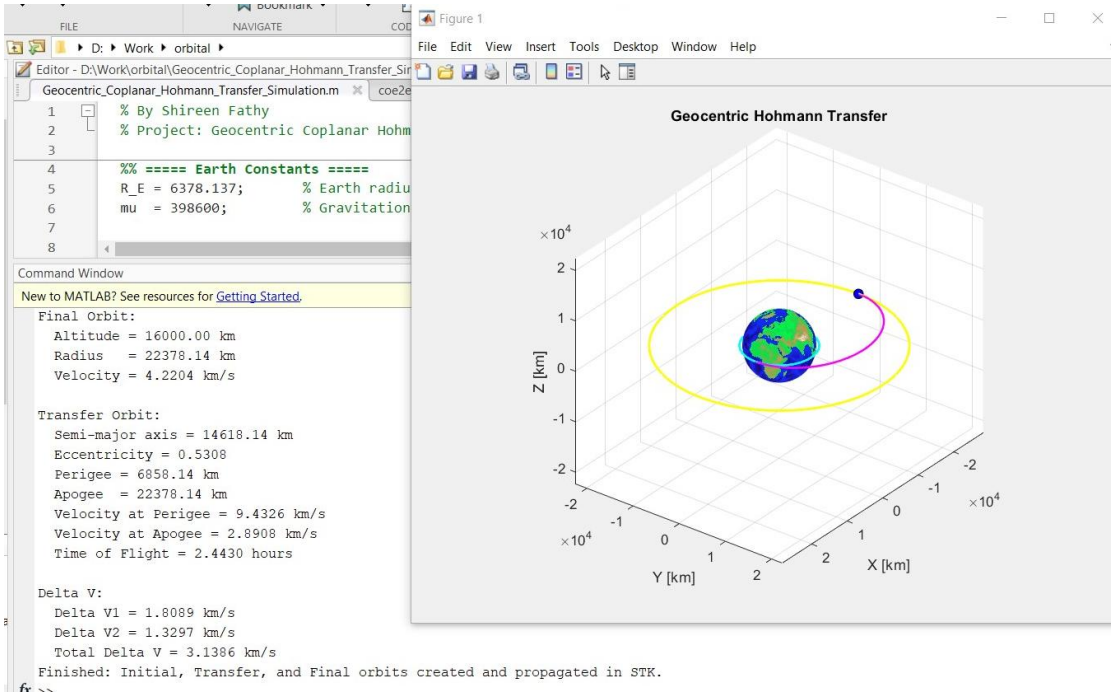


Figure 3: *Simulation of a Hohmann transfer maneuver on Matlab*

- Cyan: Initial circular orbit

- purple: Elliptical transfer orbit

-Yellow: Final circular orbit

The plot highlights the satellite's position at each step, demonstrating changes in speed and trajectory during the maneuver.

References

[1] Curtis, H. D. *Orbital Mechanics for Engineering Students*, 3rd edition, Elsevier, Amsterdam, 2013.
[2] Shepperd, S.W. (1985). *Universal Keplerian State Transition Matrix.* Celestial Mechanics, 35(2), 129–144.