Space Flight Mechanics: Assignment-2

A Report submitted

For internal assessment for Coarse

Space Flight Mechanics

in

Aerospace Engineering

by

Shingala Vaidik

(SC21B054)

pursued in

Department of Aerospace Engineering

Indian Institute of Space Science and Technology

to

Dr. Aravind

INDIAN INSTITUTE OF SPACE SCIENCE AND TECHNOLOGY

THIRUVANANTHAPURAM

December 4, 2023

Contents

Abstract

Lambert's problem involves determining the trajectory of a satellite given its initial and final position vectors and the time of flight. This document presents a detailed solution to Lambert's problem, including the steps involved in resolving the quadrant ambiguity, calculating the constant A, and iteratively solving for z using Lambert's solver. The orbital elements are then obtained, and the ground track is traced for one complete revolution. The provided MATLAB code implements these computations and visualizations, offering a practical solution to Lambert's problem.

Lambert's Problem and Chase Maneuvers in designing a lunar transfer trajectory from an Earth Parking Orbit (EPO) to a Lunar Parking Orbit. The trajectory design involves critical phases, each demanding precise calculations and strategic energy boosts. The calculated total delta-V requirement for the transfer, considering Lambert's Problem, is composed of three main components: the initial energy boost at the EPO (ΔV_1), the energy boost for lunar insertion (ΔV_2), and the delta-V for hyperbolic escape from the Moon (ΔV_3). The chapter concludes with a comprehensive overview of the delta-V components and the total delta-V for the entire lunar transfer trajectory.

Chapter 1

Lambert's Problem

1.1 Problem Statement

At a given instant, the geocentric equatorial position vector of an Earth satellite is given by $\mathbf{r}_1 = [5644, -2830, -4170]$ km, and twenty minutes later, the position is $\mathbf{r}_2 = [-2240, 7320, -4980]$ km. Determine the orbital elements, trace the orbit, and provide the ground track for one complete revolution.

1.2 Lambert's Problem

Lambert's problem involves finding the trajectory (position and velocity) of a satellite given the initial and final position vectors ($\mathbf{r}_1$ and $\mathbf{r}_2$) and the time of flight (Δt). The Lambert solver computes the velocity vectors at the start ($\mathbf{v}_1$) and end ($\mathbf{v}_2$) of the trajectory.

For solving Lambert Problem the steps are as follows.

Step-1

Suppose we know the position vectors $\mathbf{r}_1$ and $\mathbf{r}_2$ of two points P_1 and P_2 on the path of mass m around mass M, $\mathbf{r}_1$ and $\mathbf{r}_2$ determine the change in the true anomaly θ, since

$$\cos\theta = \frac{\mathbf{r}_1 \cdot \mathbf{r}_2}{r_1\, r_2} \tag{1.1}$$

where $r_1 = \sqrt{\mathbf{r}_1 \cdot \mathbf{r}_1}$ and $r_2 = \sqrt{\mathbf{r}_2 \cdot \mathbf{r}_2}$.

However, if $\cos\theta > 0$, then θ lies in either the first or fourth quadrant, whereas if $\cos\theta < 0$, then θ lies in the second or third quadrant. The first step in resolving this quadrant ambiguity is to calculate the Z component of $\mathbf{r}_1 \times \mathbf{r}_2$,

$$(\mathbf{r}_1 \times \mathbf{r}_2)_Z = r_1 r_2 \sin\theta \cos i \tag{1.2}$$

where i is the inclination of the orbit. There are two cases to consider: prograde trajectories ($0^\circ < i < 90^\circ$), and retrograde trajectories ($90^\circ < i < 180^\circ$).

This final true anomaly can be expressed more concisely as follows:

$$\theta = \begin{cases} \cos^{-1}\left(\frac{\mathbf{r}_1 \cdot \mathbf{r}_2}{r_1\, r_2}\right) & \text{if } (\mathbf{r}_1 \times \mathbf{r}_2)_Z \geq 0 \text{ (prograde trajectory)} \\ 360^\circ - \cos^{-1}\left(\frac{\mathbf{r}_1 \cdot \mathbf{r}_2}{r_1\, r_2}\right) & \text{if } (\mathbf{r}_1 \times \mathbf{r}_2)_Z < 0 \text{ (prograde trajectory)} \\ \cos^{-1}\left(\frac{\mathbf{r}_1 \cdot \mathbf{r}_2}{r_1\, r_2}\right) & \text{if } (\mathbf{r}_1 \times \mathbf{r}_2)_Z < 0 \text{ (retrograde trajectory)} \\ 360^\circ - \cos^{-1}\left(\frac{\mathbf{r}_1 \cdot \mathbf{r}_2}{r_1\, r_2}\right) & \text{if } (\mathbf{r}_1 \times \mathbf{r}_2)_Z \geq 0 \text{ (retrograde trajectory)} \end{cases} \tag{1.3}$$

Step-2

The constant A is calculated using the angle θ:

$$A = \sin(\theta)\sqrt{\frac{r_1\, r_2}{1-\cos(\theta)}} \tag{1.4}$$

Step-3

By iteration, using Equations (1.6) and (1.7), solve Equation (1.5) for z. The sign of z tells us whether the orbit is a hyperbola ($z < 0$), parabola ($z = 0$), or ellipse ($z > 0$).

$$\sqrt{\mu\Delta t} = \left(\frac{y(z)}{C(z)}\right)^{3/2} S(z) + A\sqrt{y(z)} \tag{1.5}$$

The Lambert's problem solver involves an iterative process to find the parameter z that characterizes the orbit. This is achieved by using the following expressions:

$$F(z) = \left(\frac{y(z)}{C(z)}\right)^{3/2} S(z) + A\sqrt{y(z)} - \sqrt{\mu\Delta t} \tag{1.6}$$

$$F'(z) = \begin{cases} \frac{\left(\frac{y(z)}{C(z)}\right)^{3/2}}{2z}\left(C(z) - \frac{3S(z)}{2C(z)} + \frac{3S(z)^2}{4C(z)}\right) \\ +\frac{A}{8}\left(\frac{3S(z)\sqrt{y(z)}}{C(z)} + \frac{A}{\sqrt{y(z)C(z)}}\right) & \text{if } z \neq 0 \\ \frac{\sqrt{2}}{40}y(0)^{3/2} + \frac{A}{8}\left(y(0) + A\sqrt{\frac{1}{2y(0)}}\right) & \text{if } z = 0 \end{cases} \qquad z_{i+1} = z_i - \frac{F(z_i)}{F'(z_i)} \tag{1.7}$$

where $C(z)$ and $S(z)$ are defined as:

$$C(z) = \begin{cases} \frac{1-\cos(\sqrt{z})}{z} & \text{if } z > 0 \\ \frac{\cosh(\sqrt{-z})-1}{-z} & \text{if } z < 0 \\ \frac{1}{2} & \text{if } z = 0 \end{cases} \qquad S(z) = \begin{cases} \frac{\sqrt{z}-\sin(\sqrt{z})}{(\sqrt{z})^3} & \text{if } z > 0 \\ \frac{\sinh(\sqrt{-z})-\sqrt{-z}}{(\sqrt{-z})^3} & \text{if } z < 0 \\ \frac{1}{6} & \text{if } z = 0 \end{cases}$$

Step-4

Calculate y using Equation (1.8)

$$y(z) = r_1 + r_2 + A\left(zS(z) - \frac{1}{\sqrt{C(z)}}\right) \tag{1.8}$$

Step-5

Calculate the Lagrange f,$\dot{f}$. g, and $\dot{g}$ functions using following Equations

$$f = 1 - \frac{y(z)}{r_1} \qquad g = A\sqrt{\frac{y(z)}{\mu}}$$

$$\dot{f} = \left(\frac{\sqrt{\mu}}{r_1 r_2}\right)\left(\frac{y(z)}{C(z)}\right)^{1/2}[zS(z) - 1] \qquad \dot{g} = 1 - \frac{y(z)}{r_2}$$

Step-6

Calculate initial and final velocity vectors $\mathbf{V}_1$ and $\mathbf{V}_2$:

$$\mathbf{V}_1 = \frac{1}{g}(\mathbf{r}_2 - f\mathbf{r}_1) \tag{1.9}$$

$$\mathbf{V}_2 = \frac{1}{g}(\dot{g}\mathbf{r}_2 - \mathbf{r}_1) \tag{1.10}$$

Step-7

Use $\mathbf{r}_1$ and $\mathbf{v}_1$ (or $\mathbf{r}_2$ and $\mathbf{v}_2$) in **Algorithm 1** to obtain the orbital elements.

Algorithm 1 Obtain Orbital Elements from State Vectors

1: **Input:** Position vector $\mathbf{r}_1$, Velocity vector $\mathbf{v}_1$

2: Calculate Distance and Speed:

$$r = \sqrt{X^2 + Y^2 + Z^2}$$

$$v = \sqrt{v_X^2 + v_Y^2 + v_Z^2}$$

3: Calculate Radial Velocity:

$$v_r = \frac{Xv_X + Yv_Y + Zv_Z}{r}$$

Note: If $v_r > 0$, the satellite is moving away from perigee; if $v_r < 0$, it is moving towards perigee.

4: Calculate Specific Angular Momentum and Orbital Elements:

$$h = \sqrt{\mathbf{h} \cdot \mathbf{h}} \qquad \mathbf{e} = \frac{1}{\mu}\left(\mathbf{v} \times \mathbf{h} - \frac{\mu}{r}\mathbf{r}\right)$$

$$i = \cos^{-1}\left(\frac{h_Z}{h}\right) \qquad e = \sqrt{1 - \frac{h^2}{\mu r}}$$

$$\mathbf{N} = \mathbf{K} \times \mathbf{h}$$

$$N = \sqrt{\mathbf{N} \cdot \mathbf{N}} \qquad \omega = \cos^{-1}\left(\frac{\mathbf{N} \cdot \mathbf{e}}{Ne}\right)$$

$$\Omega = \cos^{-1}\left(\frac{N_X}{N}\right) \qquad \theta = \cos^{-1}\left(\frac{\mathbf{e} \cdot \mathbf{r}}{er}\right)$$

1.3 Orbit Tracing And Ground Track

$$\mathbf{r}_x = \frac{h^2}{\mu}\frac{1}{1+e\cos\theta}\begin{bmatrix}\cos\theta\\ \sin\theta\\ 0\end{bmatrix} \tag{1.11}$$

$$\mathbf{v}_x = \frac{\mu}{h}\begin{bmatrix}-\sin\theta\\ e+\cos\theta\\ 0\end{bmatrix} \tag{1.12}$$

$$[\mathbf{Q}]_{xX} = \begin{bmatrix}\cos\omega\cos\Omega - \sin\omega\sin\Omega\cos i & -\sin\omega\cos\Omega - \cos\omega\sin\Omega\cos i & \sin\Omega\sin i\\ \cos\omega\sin\Omega - \sin\omega\cos\Omega\cos i & -\sin\omega\sin\Omega - \cos\omega\cos\Omega\cos i & -\cos\Omega\sin i\\ \sin\omega\sin i & \sin\omega\sin i & \cos i\end{bmatrix} \tag{1.13}$$

Algorithm 2 Compute Position and Velocity Vectors in Geocentric Equatorial Frame

1: $h, e, i, \Omega, \omega, \theta, \mu$
2: Calculate perifocal position vector $\{\mathbf{r}\}_x$ using Equation (1.11)
3: Calculate perifocal velocity vector $\{\mathbf{v}\}_x$ using Equation (1.12)
4: Calculate transformation matrix $[\mathbf{Q}]_{xX}$ using Equation (1.13)
5: Transform $\{\mathbf{r}\}_x$ and $\{\mathbf{v}\}_x$ to geocentric frame: $\{\mathbf{r}\}_X = [\mathbf{Q}]_{xX}\mathbf{r}_x$, $\{\mathbf{v}\}_X = [\mathbf{Q}]_{xX}\{\mathbf{v}\}_x$
6: **return** $\{\mathbf{r}\}_X$, $\{\mathbf{v}\}_X$

$$\dot{\Omega} = -\frac{3}{2}\left(\frac{\sqrt{\mu}J^2R^2}{(1-e^2)^2a^{7/2}}\right)\cos i \tag{1.14}$$

$$\dot{\omega} = \Omega'\left(\frac{5}{2}\sin^2 i - 2\right)/\cos i \tag{1.15}$$

$$\tanh\frac{E_0}{2} = \left(\sqrt{\frac{1-e}{1+e}}\tan\frac{\theta}{2}\right) \tag{1.16}$$

$$M_0 = E_0 - e\sin E_0 \tag{1.17}$$

$$\tanh\frac{F}{2} = \left(\sqrt{\frac{e-1}{1+e}}\tan\left(\frac{\theta}{2}\right)\right) \tag{1.18}$$

$$M_h = e\sinh F - F \tag{1.19}$$

$$t_0 = \frac{M_0}{2\pi/T} \tag{1.20}$$

$$M_h = \frac{\mu^2}{h^3}\cdot(e^2-1)^(3/2)\cdot t \tag{1.21}$$

$$\{\mathbf{r}\}'_x = [R_3(\theta)]\{\mathbf{r}\}_x \tag{1.22}$$

$$[R_3(\theta)] = \begin{bmatrix}\cos\theta & \sin\theta & 0\\ -\sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix} \tag{1.23}$$

$$\theta_0 = \omega_E(t - t_0)$$

For elliptical orbits, use equations (1.16), (1.17), and (1.20).
For hyperbolic orbits, use equations (1.18), (1.19), and (1.21).

Algorithm 3 Ground Track

1: **Step 1:** Compute $\dot{\Omega}$ and $\dot{\omega}$ from Equations (1.14) and (1.15).
2: **Step 2:** Calculate the initial time t_0 (time since perigee passage):
   (a) Find the eccentric anomaly E_0 from (1.16).
   (b) Find the mean anomaly M_0 from (1.17).
   (c) Find t_0 from (1.20).
3: **Step 3:** At time $t = t_0 + \Delta t$ calculate α and δ:
   (a) Calculate the true anomaly:
      (i) Find M from (1.20).
      (ii) Find E from (1.17) using Algorithm 3.1.
      (iii) Find θ from (1.16).
   (b) Update Ω and ω:
      (i) $\Omega = \Omega_0 + \dot{\Omega}\Delta t$
      (ii) $\omega = \omega_0 + \dot{\omega}\Delta t$
   (c) Find $\{r\}_x$ using **Algorithm 2**.
   (d) Find $\{r\}'_x$ using (1.23) and (1.22).
   (e) Use Algorithm 4.1 to obtain and compute α and δ from $\{r\}'_x$.

1.4 Matlab Code

```
clear all;
clc
mu=398600;
m1 = 5.974e24;  % mass of earth
Re = 6378;      % Radius of Earth
we = (2*pi + 2*pi/365.26)/(24*3600);  %earths angular velocity (rad/s)
J2 = 0.00108263;
deg = pi/180;

r1 = [ 5644 -2830 -4170];
r2 = [-2240 7320 -4980];
dt = 20*60;
string = 'pro';
[v1, v2] = lambert(r1, r2, dt, string);

coe = coe_from_sv(r1, v1, mu);
%...Save the initial true anomaly:
TA1 = coe(6);

coe1 = coe_from_sv(r2, v2, mu);
%...Save the final true anomaly:
```

```
TA2 = coe1(6);

%...Echo the input data and output the results to the command window:
fprintf('\n\n Input data:\n');
fprintf('\n\n    Gravitational Parameter,mu (km^3/s^2) =398600 \n');
fprintf('\n\n    r1 (km)                  = [%g %g %g]',r1(1), r1(2), r1(3));
fprintf('\n\n    r2 (km)                  = [%g %g %g]',r2(1), r2(2), r2(3));
fprintf('\n\n    Elapsed time (s)         = %g',dt);

fprintf('\n\n Solution:\n')
fprintf('\n\n    v1 (km/s)                = [%g %g %g]',v1(1), v1(2), v1(3));
fprintf('\n\n    v2 (km/s)                = [%g %g %g]',v2(1), v2(2), v2(3));

fprintf('\n\n Orbital Elements Caluclated For r1 and v1:\n');
fprintf('\n\n    Angular Momentum (km^2/s) = %g',coe(1));
fprintf('\n\n    Eccentricity              = %g',coe(2));
fprintf('\n\n    Inclination (deg)         = %g',coe(4)/deg);
fprintf('\n\n    RA of ascending node      = %g',coe(3)/deg);
fprintf('\n\n    Argument of Perigee (deg) = %g',coe(5)/deg);
fprintf('\n\n    True anomaly initial (deg)= %g',TA1/deg);
fprintf('\n\n    True anomaly final (deg)  = %g',TA2/deg);
fprintf('\n\n    Semimajor Axis (km)       = %g',coe(7));
fprintf('\n\n    Periapse Radius           = %g',coe(1)^2/mu/(1 + coe(2)));

% For checking Solution Accracy...
fprintf('\n\n Orbital Elements Caluclated For r2 and v2:\n');
fprintf('\n\n    Angular Momentum (km^2/s) = %g',coe1(1));
fprintf('\n\n    Eccentricity              = %g',coe1(2));
fprintf('\n\n    Inclination (deg)         = %g',coe1(4)/deg);
fprintf('\n\n    RA of ascending node      = %g',coe1(3)/deg);
fprintf('\n\n    Argument of Perigee (deg) = %g',coe1(5)/deg);
fprintf('\n\n    True anomaly initial (deg)= %g',TA1/deg);
fprintf('\n\n    True anomaly final (deg)  = %g',TA2/deg);
fprintf('\n\n    Semimajor Axis (km)       = %g',coe1(7));
fprintf('\n\n    Periapse Radius           = %g',coe1(1)^2/mu/(1 + coe1(2)));

%...If the orbit is an ellipse, output its period:
if coe(2)<1
T = 2*pi/sqrt(mu)*coe(7)^1.5;
    fprintf('\n Period:');
    fprintf('\n    Seconds                  = %g',T);
    fprintf('\n    Minutes                  = %g',T/60);
    fprintf('\n    Hours                    = %g',T/3600);
    fprintf('\n    Days                     = %g',T/24/3600);
end
```

```
%Calling for plot
thetamax=acosd(-1/emag);
Plot(hmag,emag,90,omega,w,i);
hold on
[X,Y,Z] = sphere;
Radius=6300;
X=X*Radius;
Y=Y*Radius;
Z=Z*Radius;
surf(X,Y,Z)
view(69,31)
%%
%Calling for Ground track
figure
for angle=(-90:0.1:90)
    p=Ground_Track(angle,hmag,i,w,omega,emag,theta,T);
end
grid on;
```

1.4.1 Function For Lambert's Problem

```
function [V1, V2]=lambert(R1, R2, t, string)
%{ This function solves Lambert%s problem.

%mu - gravitational parameter (km^3/s^2)
% R1, R2 - initial and final position vectors (km)
% r1, r2 - magnitudes of R1 and R2
%t - the time of flight from R1 to R2 (a constant) (s)
% V1, V2 - initial and final velocity vectors (km/s)
% c12 -  cross product of R1 into R2
%theta - angle between R1 and R2
% string - %pro  if the orbit is prograde  "retro" if the orbit is retrograde
% A - a constant given by Equation 5.35
% z - alpha*x^2, where alpha is the reciprocal of the semimajor axis and x is the
    universal anomaly
% y(z) - a function of z given by Equation 5.38
% F(z,t)- a function of the variable z and constant t,given by Equation 5.40
% dFdz(z) - the derivative of F(z,t), given by Equation 5.43
% ratio - F/dFdz
% tol - tolerance on precision of convergence
% nmax - maximum number of iterations of Newton s procedure
% f, g  - Lagrange coefficients
% gdot - time derivative of g
```

```
% C(z), S(z) — Stumpff functions
% dum — a dummy variable
% User M—functions required: stumpC and stumpS %}
mu= 398600;

%...Magnitudes of R1 and R2:
r1 = norm(R1);
r2 = norm(R2);
c12 = cross(R1, R2);
theta = acos(dot(R1,R2)/r1/r2);

%...Determine whether the orbit is prograde or retrograde:
if nargin < 4 || (~strcmp(string,'retro') & (~strcmp(string,'pro')))
    string = 'pro';
fprintf('\n ** Prograde trajectory assumed.\n')
end

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


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

%...Determine approximately where F(z,t) changes sign, and %...use that value of z
     as the starting value for Equation 5.45:
z = —100;
while F(z,t) < 0
z = z + 0.1;
end
%...Set an error tolerance and a limit on the number of iterations:
tol = 1.e—8;
nmax = 5000;
%...Iterate on Equation 5.45 until z is determined to within the %...error
    tolerance:
ratio = 1;
n =0;

while (abs(ratio) > tol) & (n <= nmax)
```

```
    n = n + 1;
    ratio = F(z,t)/dFdz(z);
    z = z - ratio;
end
%...Report if the maximum number of iterations is exceeded:
if n >= nmax
fprintf('\n\n **Number of iterations exceeds %g \n\n ',nmax)
end

f = 1 - y(z)/r1;

g = A*sqrt(y(z)/mu);

gdot = 1 - y(z)/r2;

V1 = 1/g*(R2 - f*R1);

V2 = 1/g*(gdot*R2 - R1);
return

function dum = y(z)
dum = r1 + r2 + A*(z*S(z) - 1)/sqrt(C(z));
end

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

function dum = dFdz(z)
if z == 0
dum = sqrt(2)/40*y(0)^1.5 + A/8*(sqrt(y(0)) + A*sqrt(1/2/y(0)));
else
dum = (y(z)/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))+ A*sqrt(C(z)/y(z)));
end
end

function dum = C(z)
    dum = stumpC(z);
end

function dum = S(z)
    dum = stumpS(z);
end
end
```

```
function s = stumpS(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

function c = stumpC(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
```

1.4.2 Function Of Obtain Orbital Elements from State Vector

```
function coe = coe_from_sv(R,V,mu)
eps = 1.e-10;
r = norm(R);
v = norm(V);
vr = dot(R,V)/r;
H = cross(R,V);
h = norm(H);

incl = acos(H(3)/h);

N = cross([0 0 1],H);
n = norm(N);

if n~= 0
RA = acos(N(1)/n);
    if N(2) < 0
        RA = 2*pi - RA;
    end
```

```
else
RA = 0;
end

E = 1/mu*((v^2 - mu/r)*R - r*vr*V);
e = norm(E);
% (incorporating the case e = 0):
if n~= 0
    if e > eps
        w = acos(dot(N,E)/n/e);
        if E(3) < 0
            w = 2*pi - w;
        end

      else
      w = 0;
    end

    else
    w = 0;
end
%...Equation 4.13a (incorporating the case e = 0):
if e > eps
    TA = acos(dot(E,R)/e/r);
        if vr < 0
            TA = 2*pi - TA;
        end
else
    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 < 0 for a hyperbola):
a = h^2/mu/(1 - e^2);
coe = [h e RA incl w TA a];
end
```

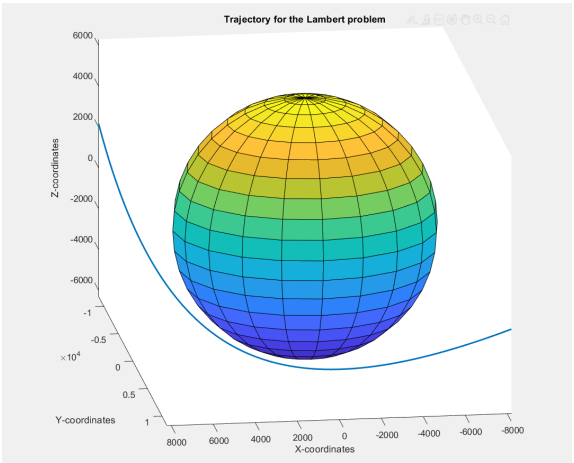
1.4.3 Function For Orbit Tracing And Ground Track

```
function Plot(hmag, emag, theta, omega, w, i)
Q1 = [cosd(w), sind(w), 0; -sind(w), cosd(w), 0; 0, 0, 1];
Q3 = [cosd(omega), sind(omega), 0; -sind(omega), cosd(omega), 0; 0, 0, 1];
Q2 = [1, 0, 0; 0, cosd(i), sind(i); 0, -sind(i), cosd(i)];
Q = transpose(Q1 * Q2 * Q3);

angles = (-theta:0.1:theta);
R = zeros(length(angles), 3);  % Initialize R matrix

for j = 1:length(angles)
    rmag = hmag^2 / (398600 * (1 + emag * cosd(angles(j))));
    r = transpose(rmag * [cosd(angles(j)), sind(angles(j)), 0]);
    V = Q * r;
    R(j, :) = V;
end

% Plotting
figure;
plot3(R(:, 1), R(:, 2), R(:, 3),"Linewidth",1.5);
xlabel('X-coordinates'); ylabel('Y-coordinates'); zlabel('Z-coordinates');
title('Trajectory for the Lambert problem');
axis equal;
end
```

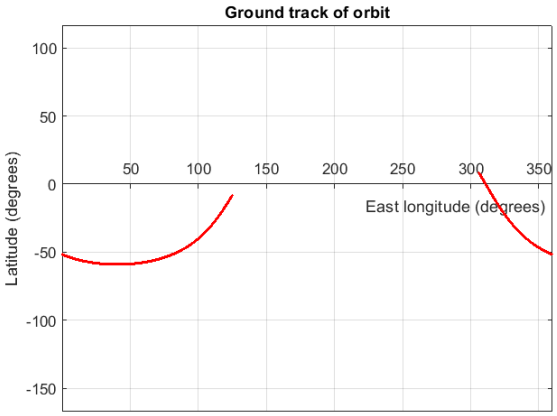
1.4.4 Output of Matlab code

```
Input data:
   Gravitational Parameter,mu (km^3/s^2) =398600
   r1 (km)                   = [5644 -2830 -4170]
   r2 (km)                   = [-2240 7320 -4980]
   Elapsed time (s)          = 1200
Solution:
   v1 (km/s)                 = [-4.13223 9.01237 -4.3781]
   v2 (km/s)                 = [-7.28524 6.31978 2.5272]

Orbital Elements Caluclated For r1 and v1:
   Angular Momentum (km^2/s) = 76096.4
   Eccentricity              = 1.20053
   Inclination (deg)         = 59.0184
   RA of ascending node      = 130.007
   Argument of Perigee (deg) = 259.98
   True anomaly initial (deg)= 320.023
```

```
    True anomaly final (deg)  = 320.023
    Semimajor Axis (km)       = —32922.3
    Periapse Radius           = 6601.82

Orbital Elements Caluclated For r2 and v2:
    Angular Momentum (km^2/s) = 76096.4
    Eccentricity              = 1.20053
    Inclination (deg)         = 59.0184
    RA of ascending node      = 130.007
    Argument of Perigee (deg) = 259.98
    True anomaly initial (deg)= 320.023
    True anomaly final (deg)  = 320.023
    Semimajor Axis (km)       = —32922.3
    Periapse Radius           = 6601.82
```



(a) trace of orbit



(b) ground track

Figure 1.1: Orbit Tracing And Ground Track

We observe that the **orbital elements form** r_1,V_1 and r_2,V_2 are identical. This **consistency** in orbital elements suggests a **high level of accuracy** in our solution.

Chapter 2

Chase Maneuvers

2.1 Problem Statement

Apply the above concept and design a transfer trajectory from an earth parking orbit to a trajectory that will finally take the space craft to a lunar parking orbit. Estimate the total delta-V requirement to transfer the s/c from earth parking orbit to the required trajectory if we approach it as a Lambert problem. Also use this problem to look at the delta-V requirements for a range of time periods for the transfer

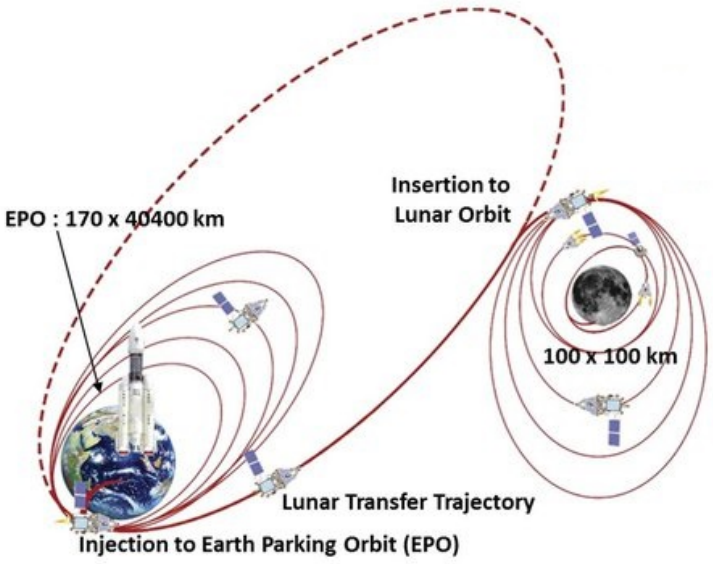


Figure 2.1: Example Of Lunar tansfer

2.2 Algorithm for Solving Lambert Problem

Algorithm 4 Lambert Problem Solver

```
Input: Initial and final position vectors, time of flight
Output: Velocity vectors at the initial and final positions
Parameters: Gravitational parameter μ of the central body
Initialize variables and constants
Solve for the semi-major axis a (using Kepler's third law)
while Convergence not reached do
    Calculate guess for z (Lagrange multiplier)
    Use Newton's method to refine z
    Calculate intermediate variables using z
end while
Calculate velocity vectors at the initial and final positions
```

2.3 Transfer Trajectory Design

Designing a Lunar Transfer Trajectory with Lambert's Problem

The meticulous design of a lunar transfer trajectory involving Lambert's Problem and Chase Maneuvers is a critical undertaking. This process ensures a strategic and resource-efficient path for the spacecraft, enabling it to traverse from an Earth Parking Orbit (EPO) to a Lunar Parking Orbit. The mission unfolds through distinct phases, each demanding precise calculations and strategic maneuvers.

Point 1 : perigee of earth orbit

Point A : the entry into the Moon's Sphere of Influence

Point 2 : perigee of moon orbit

1. Initial Energy Boost at EPO (Point 1):

- Commencing from the Earth Parking Orbit (EPO), a circular orbit around Earth, the spacecraft initiates the journey.
- A crucial initial energy boost is applied at Point 1 within the EPO, aiming to break free from Earth's gravitational influence and chart a course for a trans-lunar trajectory.

2. Re-entry into Moon's Sphere of Influence (MSI) (Point A):

- The re-entry point, designated as Point A, is strategically chosen at a specific distance from the Earth's center, treating the trajectory as a Lambert problem.
- Lambert's Problem solutions guide the determination of the required velocity at the EPO and precisely pinpoint the re-entry into the Moon's Sphere of Influence (MSI).

3. Energy Boost at MOI for Lunar Insertion (Point 2):

- Upon reaching the Moon's Sphere of Influence (MSI), the trajectory involves another crucial energy boost at Point 2. This maneuver facilitates the spacecraft's insertion into the lunar trajectory.

- This step is approached as another instance of Lambert's Problem, this time considering the Moon's motion within the calculations.

4. Calculating Velocities and Energy Differences:

- Velocities at critical points in the trajectory are meticulously determined, taking into account factors such as circular orbits and the Moon's velocity.
- Corresponding energy differences required for each maneuver are calculated, ensuring the successful execution of each phase of the trajectory.

5. Optimization and Validation:

- Mission parameters, including the durations of transfer maneuvers, undergo optimization to strike a balance between efficiency (minimizing δv) and mission duration.
- The trajectory design is rigorously validated through simulations and adjustments, ensuring alignment with mission objectives and constraints.

By systematically applying Lambert's Problem and Chase Maneuvers, this comprehensive approach facilitates the design of a transfer trajectory that adeptly guides the spacecraft from an Earth Parking Orbit to a Lunar Parking Orbit. This meticulous planning lays the foundation for successful lunar missions, combining efficiency and precision.

2.3.1 Initial conditions

```
r_epo_0 = 7213;
Theta_moon=47*pi/180;
r_mpo=1944.7;
Theta_infinity=pi*197/180;
```

2.3.2 Matlab code

```
mu_e=398600;
Theta_moon=47*pi/180;
v_moon=sqrt(mu_e/384400);
w_moon=v_moon/384400;
time_moon=Theta_moon/w_moon;


r_epo_0 = 7213;
v_epo=sqrt(mu_e/norm(r_epo_0));
theta_epo=0:0.01:pi+0.01;
r_epo=[r_epo_0*cos(theta_epo) ;r_epo_0*sin(theta_epo); 0*theta_epo];
r_epo=r_epo';
r_transfer_start=r_epo(end,:);
r_transfer_end=[351350 -57244.28 0];
```

```
t_transfer=time_moon;
[v_transfer_start, v_transfer_end] = lambert(r_transfer_start ...
    ,r_transfer_end,t_transfer);
Del_V_1=norm(v_transfer_start-v_epo*[0 -1 0])



mu_m=4904.87;
r_mpo=1944.7;
Theta_infinity=pi*197/180;
e_hyp=-1/cos(Theta_infinity);
a_hyp=r_mpo/(e_hyp-1);
v_req_wrt_m=sqrt(mu_m/a_hyp)*[cos(pi-Theta_infinity) sin(pi-Theta_infinity) 0];
v_moon_wrt_e=sqrt(mu_e/384400)*[0 1 0];
v_req_wrt_e=v_req_wrt_m+v_moon_wrt_e;
Del_V_2=norm(v_req_wrt_e-v_transfer_end)
v_hyp_p=sqrt(mu_m*(1+e_hyp)/r_mpo)*[0 1 0];
v_mpo_p=sqrt(mu_m/r_mpo)*[0 1 0];
Del_V_3=norm(v_mpo_p-v_hyp_p)
Del_V=Del_V_1+Del_V_2+Del_V_3



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
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 dum = y(z,r1,r2,A)
dum = r1 + r2 + A*(z*S(z) - 1)/sqrt(C(z));
end
```

```
function dum = F(z,t,mu,A,r1,r2)
dum = (y(z,r1,r2,A)/C(z))^1.5*S(z) + A*sqrt(y(z,r1,r2,A)) - sqrt(mu)*t;
end

function dum = dFdz(z,A,r1,r2)
if z == 0
    dum = sqrt(2)/40*y(0)^1.5 + A/8*(sqrt(y(0)) + A*sqrt(1/2/y(0)));
else
    dum = (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

function [V1, V2] = lambert(R1, R2, t)
mu = 398600;
r1 = norm(R1);
r2 = norm(R2);
c12 = cross(R1, R2);
theta = acos(dot(R1,R2)/r1/r2);
orb='Prograde';
if strcmp(orb,'Prograde')
    if c12(3) <= 0
        theta = 2*pi - theta;
    end
elseif strcmp(orb,'Retrograde')
    if c12(3) >= 0
        theta = 2*pi - theta;
    end
end

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

z = -100;
while F(z,t,mu,A,r1,r2) < 0
    z=z+ 0.1;
end

tol = 1.e-8;
nmax = 5000;

ratio = 1;
n = 0;
while (abs(ratio) > tol) & (n <= nmax)
    n = n + 1;
```

```
    ratio = F(z,t,mu,A,r1,r2)/dFdz(z,A,r1,r2);
    z = z - ratio;
end

if n >= nmax
    fprintf('\n\n **Number of iterations exceeds')
    fprintf(' %g \n\n ', nmax)
end

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;
V1 = 1/g*(R2 - f*R1);
V2 = 1/g*(gdot*R2 - R1);
end
```

2.4 Output Matlab code

```
Del_V_1 = 3.3423
Del_V_2 = 0.7620
Del_V_3 = 0.6833
Total Del_V = 4.7876
```

2.5 Conclusion

In designing a lunar transfer trajectory from an Earth Parking Orbit (EPO) to a Lunar Parking Orbit, the application of Lambert's Problem and Chase Maneuvers proves to be a comprehensive and effective approach. The trajectory design involves critical phases, each requiring precise calculations and strategic energy boosts. The calculated total delta-V requirement for the transfer, considering Lambert's Problem, is composed of three main components: the initial energy boost at the EPO (ΔV_1), the energy boost for lunar insertion (ΔV_2), and the delta-V for hyperbolic escape from the Moon (ΔV_3).

2.5.1 Delta-V Components

1. ΔV_1**:** This component represents the delta-V required for the initial energy boost at the perigee of the Earth orbit. It accounts for the spacecraft's departure from the Earth Parking Orbit to initiate the trans-lunar trajectory. The calculated value is approximately 3.3423 km/s.

2. ΔV_2**:** The second component corresponds to the delta-V needed for the energy boost at the perigee of the Moon's orbit. This maneuver is crucial for inserting the spacecraft into the lunar trajectory after entering the Moon's Sphere of Influence. The calculated value is approximately 0.7620 km/s.

3. ΔV_3**:** This component represents the delta-V required for hyperbolic escape from the Moon. It involves the spacecraft's departure from the Moon's Sphere of Influence. The calculated value is approximately

$0.6833\,\text{km/s}$.

2.5.2 Total Delta-V

The total delta-V required for the entire lunar transfer trajectory is the sum of the three components:

$$\text{Total } \Delta V = \Delta V_1 + \Delta V_2 + \Delta V_3 \approx 4.7876\,\text{km/s}$$