```matlab
close all
clc

%% Introduction %%
%-----------------%
%Programmer:     A. Clifford Matteson
%Date:           10/10/2023
%Class:          AE 5614: Spaceflight Mechanics

%% Constants %%
%-------------%

const.r_earth = 6378.14;
const.mu_earth = 3.986*10^5;
const.pi2deg = 180/pi;
const.deg2pi = pi/180;
const.AU2km = 1.496*10^8;
const.G = 6.6738*10^-20;
const.mu_sun = 1.327*10^11;
const.r_saturn = 58232;
const.mu_saturn = 3.7931*10^7;
const.g = 9.8067;
const.a_saturn = 9.54327*const.AU2km;
const.r_sun = 696300;
const.geo_orb = 35785 + const.r_earth;
const.iss_orb = 409 + const.r_earth;

%% Equations %%
%-------------%

% Energey Equation Derivations
Spe_Eng.E1 = @(v, mu, r) (v^2/2)-mu/r;
Spe_Eng.E2 = @(mu, a) -mu/(2*a);
Spe_Eng.v = @(mu, r, a) sqrt(2*((-mu/(2*a))+(mu./r))); %
Spe_Eng.r = @(mu, v, a) -mu./(-(v.^2)/2-mu./(2*a)); %
Spe_Eng.a = @(mu, v, r) -mu/(2*((v.^2/2)-mu./r)); %

% Orbit Equation Derivations
Orbit.r = @(p, ecc, nu) p/(1+ecc*cos(nu));
Orbit.p = @(r, ecc, nu) r*(1+ecc*cos(nu));
Orbit.ecc = @(r, p, nu) (p/r-1)/cos(nu);
Orbit.nu = @(r, p, ecc) acos((p./r-1)/ecc);

% Parameter Derivations
Para.p1 = @(a, ecc) a*(1-ecc^2);
Para.p2 = @(h, mu) (h^2)/mu;
Para.ecc1 = @(p, a) sqrt(1-p/a);
Para.ecc2 = @(h, a, mu) sqrt(-h.^2./(a*mu)+1); %
Para.a1 = @(p, ecc) p/(1-ecc^2);
Para.a2 = @(h, mu, ecc) h^2/(mu*(1-ecc^2));
Para.h1 = @(p, mu) sqrt(mu*p);
Para.h2 = @(a, ecc, mu) sqrt(a*mu*(1-ecc^2));

% Theta Velocity Derivations
T_Hat.v1 = @(v, gamma) v*cos(gamma);
```

```
T_Hat.v2 = @(mu, h, ecc, nu) (mu/h)*(1+ecc*cos(nu));
T_Hat.v3 = @(h,r) h/r;
T_Hat.gamma1 = @(the_v, v) acos(the_v/v);
T_Hat.gamma2 = @(h, r, v) acos(h./(r.*v)); %
T_Hat.nu = @(the_v, mu, h, ecc) acos(((the_v*h/mu)-1)/ecc);
T_Hat.h = @(r, v, gamma) r.*v.*cos(gamma);

% Radial Velocity Derivations
R_Hat.v1 = @(v, gamma) v*sin(gamma);
R_Hat.gamma1 = @(r_v, v) asin(r_v/v);
R_Hat.nu = @(r_v, mu, h, ecc) asin((r_v.*h)./(mu*ecc));

% Flight Relationships Derivations
Fli_Rel.ecc = @(r, v, mu, gamma) sqrt(((((r*v^2)/mu-1)^2)*cos(gamma)^2) ...
    +sin(gamma)^2);
Fli_Rel.nu = @(r, v, mu, gamma) atan((((r*v^2)/mu)*sin(gamma)*cos(gamma)) ...
    /(((r*v^2)/mu)*(cos(gamma)^2)-1));

% Eccentricity Derivation
Eccen.ecc = @(E, h, mu) sqrt(1+(2*E*h^2)/(mu^2));

% Elliptical Orbit Derivations
Ell_Orb.n = @(mu, a) sqrt(mu/(a^3));
Ell_Orb.Tdel = @(E1, E2, ecc, n) (E2-E1-ecc.*(sin(E2)-sin(E1)))/n;
Ell_Orb.P = @(a, mu) 2*pi*sqrt((a^3)/mu);
Ell_Orb.r = @(a, ecc, E) a*(1-ecc*cos(E));
Ell_Orb.E = @(a, r, e) acos((-r./a+1)/e);
Ell_Orb.ra = @(a, e) a*(1+e);
Ell_Orb.rp = @(a, e) a*(1-e);
Ell_Orb.e1 = @(rp, a) 1-rp/a;
Ell_Orb.e2 = @(ra, a) -1+ra/a;
Ell_Orb.a1 = @(rp, e) rp/(1-e);
Ell_Orb.a2 = @(ra, e) ra/(1+e);

% Hyperbolic Orbits Derivations
Hype_Orb.rp = @(a, ecc) a*(1-ecc);
Hype_Orb.v = @(mu, a) sqrt(-mu/a);
Hype_Orb.nu = @(p, r, ecc) acos(((p/r)-1)/ecc);
Hype_Orb.a = @(mu, v) -mu/(v^2);
Hype_Orb.del = @(e) 2*asin(1/e);
Hype_Orb.e = @(del) 1/sin(de/2);
Hype_Orb.e1 = @(rp, a) 1-rp/a;

% Lambert's Theorem Derivations
Lambert.c = @(r1, r2, phi) sqrt(r1^2+r2^2-2*r1*r2*cos(phi));
Lambert.s = @(r1, r2, c) (r1+r2+c)/2;

% Elliptical Transfers Derivations
Ell_Trans.alpha = @(s, a) 2*asin(sqrt(s/(2*abs(a))));
Ell_Trans.beta = @(s, a, c) 2*asin(sqrt((s-c)/(2*abs(a))));
Ell_Trans.Tdel_1A = @(mu, a, alpha, beta) ((alpha-sin(alpha))-(beta-sin(beta)))/sqrt(mu/
abs(a)^3);
Ell_Trans.Tdel_1B = @(mu, a, alpha, beta) ((alpha-sin(alpha))-(beta-sin(beta))+2*pi)/sqrt
(mu/abs(a)^3);
Ell_Trans.Tdel_2A = @(mu, a, alpha, beta) ((alpha-sin(alpha))+(beta-sin(beta)))/sqrt(mu/
```

```
abs(a)^3);
Ell_Trans.Tdel_2B = @(mu, a, alpha, beta) ((alpha-sin(alpha))+(beta-sin(beta))+2*pi)/sqrt
(mu/abs(a)^3);
% 1 = (0 <= x < 180) , 2 =  (180 <= x < 360)
% A = Antifocus in region, B = Antifocus not in region

% Hyperbolic Transfers Derivations
Hype_Trans.a_prime = @(s, a) 2*asinh(sqrt(s/(2*abs(a))));
Hype_Trans.Tra_b_prime = @(s, a, c) 2*asinh(sqrt((s-c)/(2*abs(a))));
Hype_Trans.Tra_Tdel_2H = @(mu, a, a_prime, b_prime) ((sinh(a_prime)-a_prime)+ ...
    (sinh(b_prime)-b_prime))/sqrt(mu/abs(a)^3);
Hype_Trans.Tra_Tdel_1H = @(mu, a, a_prime, b_prime) ((sinh(a_prime)-a_prime)- ...
    (sinh(b_prime)-b_prime))/sqrt(mu/abs(a)^3);

% True & Ecc Anomaly Derivations
True_Ecc.E = @(nu, ecc) 2*atan(tan(nu./2)/sqrt((1+ecc)/(1-ecc)));
True_Ecc.nu = @(E, ecc) 2*atan(sqrt((1+ecc)/(1-ecc))*tan(E/2));

% Fly By Conics Derivations
Fly_By_Con.Phi = @(v_arr, gamma, v_pla) atan((v_arr*sin(gamma))/(v_arr*cos(gamma)-v_pla
));

% Law of Cosine Derivatio
Law_Cos = @(v1, v2, theta) sqrt(v1^2+v2^2-2*v1*v2*cos(theta));

% Rocket Equation Derivations
Rocket.delV = @(isp, mo, mf) isp*0.00980665*log(mo/mf);
Rocket.m0  = @(mf, delV, isp) mf*exp(-delV/(isp*0.00980665));
Rocket.mf  = @(m0, delV, isp) m0./exp(delV/(isp*0.00980665));
Rocket.isp  = @(m0, mf, delV) delV/(0.00980665*log(m0/mf));
Rocket.delM = @(mf, delV, isp) mf.*(exp(delV./(isp*0.00980665))-1);

%% Problem 1 %%
%-------------%

% Finds the max delta V available
mass_fuse = 9300 + 6185;
mass_prop = 1000;
mass_int = mass_prop + mass_fuse;
engine_isp = 316;
max_delV = Rocket.delV(engine_isp, mass_int, mass_fuse);
fprintf("The max possible delta V is %4.4f km/s \n", max_delV)

% Inital point variables
r_i = 250000;
a_i = 200000;
ecc_i = 0.9728;
r_alt = const.r_earth + 150;

% Tolerance for target flight path
gammaL = -5.0001*pi/180;
gammaH = -4.9999*pi/180;

% Creates range of angles, velocities, and radi (Polar)
delThe = single(linspace(0, pi*2, 360*3));
```

```
delV= single(linspace(0, max_delV, 200*3))';
delR = single(r_i:-25:25000);
% 2,2,-12.5

% Reshapes array from 1D to 3D, needed for calculations.
delR = reshape(delR, [1,size(delR)]);

% Findes radius and theta velocities (Cart), 2D array
delVr = delV.*sin(delThe);
delVt = delV.*cos(delThe);

% Finds h using paramter eq., Constant, Scalar
orbH_i = Para.h2(a_i,ecc_i,const.mu_earth);

% Finds inital velocity magnitude at each altitude with Spec Eng, 1D array
Vmag_i = Spe_Eng.v(const.mu_earth, delR, a_i);

% Finds inital gammat at each alt, Flight path /w acos, 1D array
gamma_i = -T_Hat.gamma2(orbH_i, delR, Vmag_i); % Gamma -, Sine -, quad 4

% Finds inital velocity vectors at each altitude, 1D array in 3D
Vr_i = Vmag_i.*sin(gamma_i);
Vt_i = Vmag_i.*cos(gamma_i);

% Finds final velocity vector, int (2D) - del(1D in 3rd dim), 3D array
Vr_f = bsxfun(@minus, Vr_i, delVr);
Vt_f = bsxfun(@minus, Vt_i, delVt);

% Converts new velocities from Cart to Polar coordinates, Both 3D arrays
% I know that all arrays will be in quad 4
[v_f, gamma_f] = cart2polar(Vr_f, Vt_f,"Quad",4);
clear Vr_f Vt_f

% Finds angular momentum of orbit using theta hat derivation, 3D array
h_orbit = T_Hat.h(delR, v_f, gamma_f);

% Finds semimajor axis of orbit using spesific energy derivation, 3D
a_orbit = Spe_Eng.a(const.mu_earth, v_f, delR);

% Finds eccentricity of orbit using parameter equation derivation, 3D array
ecc_orbit = Para.ecc2(h_orbit, a_orbit, const.mu_earth);

% Finds velocity at altitude using spesific energy derivation, 3D
v_alt = Spe_Eng.v(const.mu_earth, r_alt, a_orbit);

% Finds the flight path at altitude using theta hat derivation, 3D
gamma_orbit = -T_Hat.gamma2(h_orbit, r_alt, v_alt);

% Logic matrix used to isolate values that match flight path critera, 3D
loc = ((gamma_orbit > gammaL) & (gamma_orbit < gammaH));

% Assignes values to array that match the logic matrix values

% Velocity at alt, uses logical matrix
output(:,1) = v_alt(loc);
```

```
clear v_alt

% Flight path at alt, uses logical matrix
output(:,2) = gamma_orbit(loc);
clear gamma_orbit

% Eccentricity of orbit, uses logical matrix
output(:,3) = ecc_orbit(loc);
clear ecc_orbit

% Semimajor axis of orbit, uses logical matrix
output(:,4) = a_orbit(loc);
clear a_orbit

% Angular momentum of orbit, uses logical matrix
output(:,5) = h_orbit(loc);
clear h_orbit

% Velocity after manuver, uses logical matrix
output(:,6) = v_f(loc);
clear v_f

% Flight path after manuver, uses logical matrix
output(:,7) = gamma_f(loc);
clear gamma_f loc

% Radius at manuver, uses Spesific Energy Derivation, 1D
output(:,8) = Spe_Eng.r(const.mu_earth,output(:,6),output(:,4));

% Velocity of orbit before manuver, uses Spesific Energy Derivation, 1D
output(:,9) = Spe_Eng.v(const.mu_earth,output(:,8),a_i);

% Flight path of orbit before manuver, Uses theta velocity derivation, 1D
% Approaching so (-) cosine
output(:,10) = -T_Hat.gamma2(orbH_i,output(:,8),output(:,9));

% Vr before and after manuver, finds the difference, 1D
output(:,11) = output(:,6).*sin(output(:,7));
output(:,12) = output(:,9).*sin(output(:,10));
output(:,13) = output(:,11)-output(:,12);

% Vt before and after manuver, finds the difference, 1D
output(:,14) = output(:,6).*cos(output(:,7));
output(:,15) = output(:,9).*cos(output(:,10));
output(:,16) = output(:,14)-output(:,15);

% Change in delV, Calculates the magnitude from the differnce in Vr and Vt.
% 1D
output(:,17) = (output(:,13).^2+output(:,16).^2).^(1/2);

% Mass after manuver, Rocket Equation derivation, 1D
output(:,18) = Rocket.mf(mass_int,output(:,17),engine_isp);

% Mass of propellant used, Rocket Equation derivation, 1D
output(:,19) = Rocket.delM(output(:,18),output(:,17),engine_isp);
```

```
% Finds possivle nu values at each alt, Cosine, and Sine used
nu(:,1) = Orbit.nu(output(:,8),Para.p2(orbH_i,const.mu_earth),ecc_i);
nu(:,2) = 2*pi-nu(:,1);
nu(:,3) = R_Hat.nu(output(:,12),const.mu_earth,orbH_i,ecc_i);
nu(:,4) = pi-nu(:,3);

% Rounds out values and runs through quad check program, Nu
nu(:,:) = round(nu(:,:),7);
nu_quad = quadcheck(nu(:,:));

% Finds possible Ecc values at each alt, Cosine and Tangent
Ecc(:,1) = Ell_Orb.E(a_i,output(:,8), ecc_i);
Ecc(:,2) = -Ecc(:,1);
Ecc(:,3) = True_Ecc.E(nu_quad, ecc_i);
Ecc(:,4) = Ecc(:,3)+pi;

% Rounds out values and runs through quad check program, Ecc
Ecc(:,:) = round(Ecc(:,:),6);
Ecc_quad = quadcheck(Ecc(:,:));

% Finds initial Ecc and Nu values
E_i = -Ell_Orb.E(a_i, r_i, ecc_i);
n_i = Ell_Orb.n(const.mu_earth, a_i);

% Finds the time change between the inital altitude and the manuver alt
tdel = Ell_Orb.Tdel(E_i, Ecc_quad(:), ecc_i, n_i)/3600;

% Prints outputs
fprintf('The max time before a manuver must be made is %4.3f hours.\n', tdel(end))
fprintf('The last possible manuver has a delta V of is %4.3f km/s.\n',output(end,17))

area(output(:,8),output(:,19),max(output(:,19)))
xlabel('Last Chance Manuver Altitude (km)')
ylabel('Propellant Mass Required (kg)')
title('Last Chance Manuver Altitude vs. Propellant Mass Required')
```