```
clear all
close all
clc

%% Introduction %%
%----------------%
%Programmer:     A. Clifford Matteson
%Date:           09/20/2023

%% 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
Orb_Equ_r = @(p, ecc, nu) p/(1+ecc*cos(nu));
Orb_Equ_p = @(r, ecc, nu) r*(1+ecc*cos(nu));
Orb_Equ_ecc = @(r, p, nu) (p/r-1)/cos(nu);
Orb_Equ_nu = @(r, p, ecc) acos((p/r-1)/ecc);

% Parameter Derivations
Par_p1 = @(a, ecc) a*(1-ecc^2);
Par_p2 = @(h, mu) (h^2)/mu;
Par_ecc1 = @(p, a) sqrt(1-p/a);
Par_ecc2 = @(h, a, mu) sqrt(-h^2/(a*mu)+1);
Par_a1 = @(p, ecc) p/(1-ecc^2);
Par_a2 = @(h, mu, ecc) h^2/(mu*(1-ecc^2));
Par_h1 = @(p, mu) sqrt(mu*p);
Par_h2 = @(a, ecc, mu) sqrt(a*mu*(1-ecc^2));

% Theta Velocity Derivations
The_Vel_v1 = @(v, gamma) v*cos(gamma);
The_Vel_v2 = @(mu, h, ecc, nu) (mu/h)*(1+ecc*cos(nu));
The_Vel_v3 = @(h,r) h/r;
The_Vel_gamma1 = @(the_v, v) acos(the_v/v);
The_vel_gamma2 = @(h, r, v) acos(h/(r*v));
The_Vel_nu = @(the_v, mu, h, ecc) acos(((the_v*h/mu)-1)/ecc);
The_Vel_h = @(r, v, gamma) r*v*cos(gamma);

% Radial Velocity Derivations
R_Vel_v1 = @(v, gamma) v*sin(gamma);
R_Vel_gamma1 = @(r_v, v) asin(r_v/v);
R_Vel_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
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
Hyp_Obr_rp = @(a, ecc) a*(1-ecc);
Hyp_Obr_v = @(mu, a) sqrt(-mu/a);
Hyp_Obr_nu = @(p, r, ecc) acos(((p/r)-1)/ecc);
Hyp_Orb_a = @(mu, v) -mu/(v^2);
Hyp_Orb_del = @(e) 2*asin(1/e);
Hyp_Orb_e = @(del) 1/sin(de/2);
Hyp_Orb_e1 = @(rp, a) 1-rp/a;

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

% Elliptical Transfers Derivations
Ell_Tra_alpha = @(s, a) 2*asin(sqrt(s/(2*abs(a))));
Ell_Tra_beta = @(s, a, c) 2*asin(sqrt((s-c)/(2*abs(a))));
Ell_Tra_Tdel_1A = @(mu, a, alpha, beta)
((alpha-sin(alpha))-(beta-sin(beta)))/sqrt(mu/abs(a)^3);
Ell_Tra_Tdel_1B = @(mu, a, alpha, beta)
((alpha-sin(alpha))-(beta-sin(beta))+2*pi)/sqrt(mu/abs(a)^3);
Ell_Tra_Tdel_2A = @(mu, a, alpha, beta)
((alpha-sin(alpha))+(beta-sin(beta)))/sqrt(mu/abs(a)^3);
Ell_Tra_Tdel_2B = @(mu, a, alpha, beta)
((alpha-sin(alpha))+(beta-sin(beta))+2*pi)/sqrt(mu/abs(a)^3);

% Hyperbolic Transfers Derivations
Hyp_Tra_a_prime = @(s, a) 2*asinh(sqrt(s/(2*abs(a))));
Hyp_Tra_b_prime = @(s, a, c) 2*asinh(sqrt((s-c)/(2*abs(a))));
Hyp_Tra_Tdel_2H = @(mu, a, a_prime, b_prime) ((sinh(a_prime)-a_prime)+ ...
    (sinh(b_prime)-b_prime))/sqrt(mu/abs(a)^3);
Hyp_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
Tre_Ecc_E = @(nu, ecc) 2*atan(tan(nu/2)/sqrt((1+ecc)/(1-ecc)));
Tre_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
Rock_Eq_delV = @(isp, mo, mf) isp*0.00980665*ln(mo/mf);
Rock_Eq_m0 = @(mf, delV, isp) mf*exp(-delV/(isp*0.00980665));
Rock_Eq_mf = @(m0, delV, isp) m0/exp(delV/(isp*0.00980665));
Rock_Eq_isp = @(m0, mf, delV) delV/(0.00980665*ln(m0/mf));
```

```
Rock_Eq_delM = @(mf, delV, isp) mf*(exp(delV/(isp*0.00980665))-1);

%% P1Pa %%
%-------------%

% Given
r_i = 250000;
a_i = 200000;
ecc_i = 0.9728;
r_alt = const.r_earth + 150;

%solving initial v, h, gamma
p1pa.h_i = Par_h2(a_i, ecc_i, const.mu_earth);
p1pa.v_i = Spe_Eng_v(const.mu_earth, r_i, a_i);
p1pa.gamma_i = The_vel_gamma2(p1pa.h_i, r_i, p1pa.v_i);
p1pa.gamma_i = [pi-p1pa.gamma_i, 2*pi-p1pa.gamma_i];  %Quad checks, sine(+)
p1pa.v = [p1pa.v_i*sin(p1pa.gamma_i(2)), p1pa.v_i*cos(p1pa.gamma_i(2))];

%Solving v and gamma for alt
p1pa.v_a = Spe_Eng_v(const.mu_earth, r_alt, a_i);
p1pa.gamma_a = The_vel_gamma2(p1pa.h_i, r_alt, p1pa.v_a);
% Quad check, sine(+)
p1pa.gamma_a = [pi-p1pa.gamma_a, -p1pa.gamma_a]; %% Aswer %%


fprintf('The flight path angle at altiude is %4.3f Rad or %4.3f Deg \n\n', p1pa.gamma_a(2), rad2deg(p1pa.gamma_a(2)))

%% P1pb %%
%--------%

% Creates range of angles and velocities.
p1pb.theta_delta = linspace(pi/2, pi, 360*5);
p1pb.vel_delta = linspace(0,.05, 500)';

% Findes radius and theta velocities
p1pb.vr_del_mat = p1pb.vel_delta.*sin(p1pb.theta_delta);
p1pb.vt_del_mat = p1pb.vel_delta.*cos(p1pb.theta_delta);

% Generates tensor for vectors (Cartesian)
p1pb.vec_del(:, :, 1) = p1pb.vr_del_mat(:, :);
p1pb.vec_del(:, :, 2) = p1pb.vt_del_mat(:, :);

% New vector component directions (Cartesian)
p1pb.vec_new(:, :, 1) = p1pa.v(1)-p1pb.vec_del(:, :, 1);
p1pb.vec_new(:, :, 2) = p1pa.v(2)-p1pb.vec_del(:, :, 2);

for i = 1:size(p1pb.vec_new,1)
    for j = 1:size(p1pb.vec_new,2)
        % Converts new vector coord system (Polar)
        % cart2polar has in-built quad check
        [p1pb.vec(i, j, 1), p1pb.vec(i, j, 2)] = cart2polar(p1pb.vec_new(i, j, 1), p1pb.vec_new(i, j, 2));

        % Finds the angular momentum, semimajor axis, and eccentricity
        p1pb.h(i, j, 1) = The_Vel_h(r_i, p1pb.vec(i, j, 1), p1pb.vec(i, j, 2));
        p1pb.a(i, j, 1) = Spe_Eng_a(const.mu_earth, p1pb.vec(i, j, 1), r_i);
        p1pb.ecc(i, j, 1) = Par_ecc2(p1pb.h(i, j, 1), p1pb.a(i, j, 1), const.mu_earth);

        % Finds the new velocity and angle at altitude
        p1pb.v_alt(i, j, 1) = Spe_Eng_v(const.mu_earth, r_alt, p1pb.a(i, j, 1));
        p1pb.gamma(i, j, 1) = real(-The_vel_gamma2(p1pb.h(i, j, 1), r_alt, p1pb.v_alt(i, j, 1)));
    end
end

% Sets tolerences
p1pb.gammaL = -0.087283916;
p1pb.gammaH = -0.087249009;
```

```
p1pb.ans = [];

% Loop that appends values from tensor that match tolerances
for i = 1:size(p1pb.gamma,1)
    for j = 1:size(p1pb.gamma,2)
        if (p1pb.gammaL < p1pb.gamma(i, j, 1)) && (p1pb.gamma(i, j, 1) < p1pb.gammaH) && (p1pb.ecc(i, j, 1) < 1) && (0 < p1pb.a(i, j, 1))

            p1pb.ans = [p1pb.ans; p1pb.vec_del(i, j, 1), p1pb.vec_del(i, j, 2)];
        end
    end
end

p1pb.ansf = [];

% Loop that takes values from appended array and calulates values
for i = 1:size(p1pb.ans,1)
    % Delta Vr and Vt
    [p1pb.ansv_c] = [p1pa.v(1)-p1pb.ans(i, 1), p1pa.v(2)-p1pb.ans(i, 2)];

    % Vr and Vt to V_mag and theta
    [p1pb.ansv_p1, p1pb.ansv_p2]  = cart2polar(p1pb.ansv_c(1), p1pb.ansv_c(2));

    % Finds angluar momentum
    p1pb.ansh = The_Vel_h(r_i, p1pb.ansv_p1, p1pb.ansv_p2);

    % Finds semimajor axis
    p1pb.ansa = Spe_Eng_a(const.mu_earth, p1pb.ansv_p1, r_i);

    % Finds eccentricity
    p1pb.anse = Par_ecc2(p1pb.ansh, p1pb.ansa, const.mu_earth);

    % Finds velocity at altitude
    p1pb.ansv_alt = Spe_Eng_v(const.mu_earth, r_alt, p1pb.ansa);

    % Finds gamma at altitude, must be negative for approach
    p1pb.ansgamma = -The_vel_gamma2(p1pb.ansh, r_alt, p1pb.ansv_alt);

    % Outpust values to new array
    p1pb.ansf = [p1pb.ansf; p1pb.ans(i, 1), p1pb.ans(i, 2), p1pb.ansv_c(1), p1pb.ansv_c(2), p1pb.ansv_p1, p1pb.ansv_p2, p1pb.ansh, p1pb.ansa, p1pb.anse, p1pb.ansv_alt, p1pb.ansgamma, norm([p1pb.ans(i, 1), p1pb.ans(i, 2)])];
end

% Presents values %% Aswer %%
p1pb.ansfT = array2table(p1pb.ansf,'VariableNames',{'Vr Change','Vt Change','Final Vr','Final Vt','V Mag','V Direction','h','a','e','V Alt', 'Flight Path','delV Mag'});
disp(p1pb.ansfT)

%% P1pc %%
%--------%

% Calculates n
p1pc.n = Ell_Orb_n(const.mu_earth, p1pb.ansf(1,8));

% Finds nu and runs quad check at inintal point, quad 4
[p1pc.nu1] = [R_Vel_nu(p1pb.ansf(1,3), const.mu_earth, p1pb.ansf(1,7), p1pb.ansf(1,9)),pi-R_Vel_nu(p1pb.ansf(1,3), const.mu_earth, p1pb.ansf(1,7), p1pb.ansf(1,9))];
[p1pc.nu2] = [The_Vel_nu(p1pb.ansf(1,4), const.mu_earth, p1pb.ansf(1,7), p1pb.ansf(1,9)), -The_Vel_nu(p1pb.ansf(1,4), const.mu_earth, p1pb.ansf(1,7), p1pb.ansf(1,9))];
p1pc.nu_i = p1pc.nu1(2);

% Finds eccentric anomaly and runs quad check inintal point, quad 4
[p1pc.E1] = [Ell_Orb_E(p1pb.ansf(1,8),r_i,p1pb.ansf(1,9)),-Ell_Orb_E(p1pb.ansf(1,8),r_i,p1pb.ansf(1,9))];
[p1pc.E2] = [Tre_Ecc_E(p1pc.nu_i, p1pb.ansf(1,9)),Tre_Ecc_E(p1pc.nu_i,
```

```
p1pb.ansf(1,9))+pi];
p1pc.E_i = 2*pi+p1pc.E2(1);

% Finds the altitude velocity in cartesian
p1pc.vr = R_Vel_v1(p1pb.ansf(1,10), p1pb.ansf(1,11));
p1pc.vt = The_Vel_v1(p1pb.ansf(1,10), p1pb.ansf(1,11));

% Finds that nu at altitude, quad 4
[p1pc.nu1] = [R_Vel_nu(p1pc.vr, const.mu_earth, p1pb.ansf(1,7), p1pb.ansf(1,9)), pi-R_Vel_nu(p1pc.vr, const.mu_earth, p1pb.ansf(1,7), p1pb.ansf(1,9))];
[p1pc.nu2] = [The_Vel_nu(p1pc.vt, const.mu_earth, p1pb.ansf(1,7), p1pb.ansf(1,9)), 2*pi-The_Vel_nu(p1pc.vt, const.mu_earth, p1pb.ansf(1,7), p1pb.ansf(1,9))];
p1pc.nu_f = p1pc.nu2(2);

% Finds E at altitude, quad 4
[p1pc.E1] = [Ell_Orb_E(p1pb.ansf(1,8), r_alt, p1pb.ansf(1,9)), 2*pi-Ell_Orb_E(p1pb.ansf(1,8), r_alt, p1pb.ansf(1,9))];
[p1pc.E2] = [Tre_Ecc_E(p1pc.nu_f, p1pb.ansf(1,9)),pi+Tre_Ecc_E(p1pc.nu_f, p1pb.ansf(1,9))];
p1pc.E_f = p1pc.E1(2);

% Finds time between inital point and altitude (h) %% Aswer %%
p1pc.tdel1 = Ell_Orb_Tdel(p1pc.E_i, p1pc.E_f, p1pb.ansf(1,9), p1pc.n);

fprintf('The time between maneuver and altitude intercept is %4.3f seconds or %4.3f days\n', p1pc.tdel1, p1pc.tdel1/86400)

%% Double checking time Lambert %%

p1pc.phi = p1pc.nu_f-p1pc.nu_i;
p1pc.c = Lam_The_c(r_i, r_alt, p1pc.phi);
p1pc.s = Lam_The_s(r_i, r_alt, p1pc.c);
p1pc.alpha = Ell_Tra_alpha(p1pc.s, p1pb.ansf(1,8));
p1pc.beta = Ell_Tra_beta(p1pc.s, p1pb.ansf(1,8), p1pc.c);
p1pc.tdel2 = Ell_Tra_Tdel_1A(const.mu_earth, p1pb.ansf(1,8),p1pc.alpha, p1pc.beta)/3600;
% Varified

%% P1pd %%
%--------%

% Initalizes variables
p1pd.orionM = 9300;
p1pd.esmM = 6185;
p1pd.propM = 1000;
p1pd.isp = 316;

% Finds magnitude of velocity and finds inital mass
p1pd.delV = norm([p1pb.ansf(1,1), p1pb.ansf(1,2)]);
p1pd.initalM = p1pd.orionM + p1pd.esmM + p1pd.propM;

% Finds final mass
p1pd.mf = Rock_Eq_mf(p1pd.initalM, p1pd.delV, p1pd.isp);

% Calculates change in mass
p1pd.delM = Rock_Eq_delM(p1pd.mf, p1pd.delV, p1pd.isp); %% Aswer %%

fprintf('The mass of propellant used is %4.3f kg \n', p1pd.delM)

%% P1pe %%
%--------%

% Need to cancel out radial velocity %

% Left over mass after maneuver
p1pe.propM = 1000 - p1pd.delM;
p1pe.initalM = p1pd.orionM + p1pd.esmM + p1pe.propM;

% Finds new value of radi
p1pe.r_alt_n = 400 + const.r_earth;
```

```
p1pe.r_iss = 408 + const.r_earth;

% Finds velocity,theta velocity, and radial velocity at 400km alt
p1pe.v_alt = Spe_Eng_v(const.mu_earth, p1pe.r_alt_n, p1pb.ansf(1,8));
p1pe.vt_alt = p1pb.ansf(1,7)/p1pe.r_alt_n;
p1pe.vr_alt = (p1pe.v_alt^2-p1pe.vt_alt^2)^(1/2); % Impotanat number

% Finds velocity of circular ISS orbit
p1pe.v_iss = (const.mu_earth/p1pe.r_iss)^(1/2);

% Finds change in velocity components
p1pe.delVr_iss = p1pe.vr_alt;
p1pe.delV = p1pe.v_alt - p1pe.v_iss;
p1pe.delVt_iss = (p1pe.delV^2-p1pe.delVr_iss^2)^(1/2);

p1pe.delVec = [-p1pe.delVr_iss, -p1pe.delVt_iss]; %% Aswer %%

% Finds final mass
p1pe.mf = Rock_Eq_mf(p1pe.initalM, p1pe.delV, p1pd.isp);

% Calculates change in mass
p1pe.delM = Rock_Eq_delM(p1pe.mf, p1pe.delV, p1pd.isp);

fprintf('The change in velocity is %4.3f km/s with directions %4.3f and %4.3f in Vr km/s and Vt km/s respectively\n', p1pe.delV, p1pe.delVec(1), p1pe.delVec(2))
fprintf('The propellant mass required for this is %4.3f kg\n', p1pe.delM)

%% P1pf %%
%--------%

% Need to cancel out radial velocity %

% Left over mass after maneuver
p1pf.propM = 1000 - p1pd.delM;
p1pf.initalM = p1pd.orionM + p1pd.esmM + p1pf.propM;

% Finds velocity of circular ISS orbit
p1pf.v_geo = (const.mu_earth/const.geo_orb)^(1/2);
p1pf.delV = p1pe.v_alt - p1pf.v_geo;

% Finds final mass
p1pf.mf = Rock_Eq_mf(p1pf.initalM, p1pf.delV, p1pd.isp);

% Calculates change in mass
p1pf.delM = Rock_Eq_delM(p1pf.mf, p1pf.delV, p1pd.isp);

fprintf('The change in velocity needed is %4.3f km/s with %4.3f kg of propellant needed\n', p1pf.delV, p1pf.delM)
fprintf('From this, we can say that a manuver to geo orbit from 400km alt is unreasonable\n')
```