Table of Contents

Roshan Jaiswal-Ferri

```
%Section - 01
%Aero 351 Final Exam Question 1: 12/07/24
```

Workspace Prep

```
format long      %Allows for more accurate decimals
close all;       %Clears all
clear all;       %Clears Workspace
clc;             %Clears Command Window
```

Constant/Global Vars

```
options = odeset('RelTol',1e-8,'AbsTol',1e-8);
muSun = 1.327e11; %mu values from curtis
muMars = 42828;
mu = 398600; %earth
muJup = 126686534;
Rmars = 3396; %km
Rearth = 6378; %km
Rjup = 71490;
tol = 1e-7;
```

Problem 1 Datetime

```
timeD = datetime(2026,11,1,0,0,0); %time departure
timeAMar = datetime(2030,03,1,0,0,0);
timeAug = datetime(2030,08,1,0,0,0);
timeOct = datetime(2030,10,1,0,0,0);


JD2000 = juliandate(2000,1,1,0,0,0);
JDdep = juliandate(timeD);
JDMar = juliandate(timeAMar);
JDAug = juliandate(timeAug);
JDOct = juliandate(timeOct);

dtMar = (JDMar - JDdep)*86400; %delta t of transf in seconds
dtAug = (JDAug - JDdep)*86400;
dtOct = (JDOct - JDdep)*86400;

TD = (JDdep-JD2000)/36525; %convert to century year
TMar = (JDMar-JD2000)/36525;
TAug = (JDAug-JD2000)/36525;
TOct = (JDOct-JD2000)/36525;
```

Finding planet positions

```
earthCOESV = AERO351planetary_elements2(3,TD);
[a,ecc,inc,raan,w,theta,w_hat,L] = pcoes(earthCOESV);

JupMarV = AERO351planetary_elements2(5,TMar);
[a_m,ecc_m,inc_m,raan_m,w_m,theta_m,w_hat_m,L_m] = pcoes(JupMarV);

JupAugV = AERO351planetary_elements2(5,TAug);
[a_a,ecc_a,inc_a,raan_a,w_a,theta_a,w_hat_a,L_a] = pcoes(JupAugV);

JupOctV = AERO351planetary_elements2(5,TOct);
[a_o,ecc_o,inc_o,raan_o,w_o,theta_o,w_hat_o,L_o] = pcoes(JupOctV);
```

Finding Planetary R & V vectors

```
[~,~,Re,Ve] = coes2rvd(a,ecc,inc,raan,w,theta,muSun);
[~,~,RJm,VJm] = coes2rvd(a_m,ecc_m,inc_m,raan_m,w_m,theta_m,muSun);
[~,~,RJa,VJa] = coes2rvd(a_a,ecc_a,inc_a,raan_a,w_a,theta_a,muSun);
[~,~,RJo,VJo] = coes2rvd(a_o,ecc_o,inc_o,raan_o,w_o,theta_o,muSun);
```

Short Way Laberts Problem

```
tm = 1;

[V1m,V2m] = lambUVBi(Re,RJm,dtMar,tm,muSun,tol);
[V1a,V2a] = lambUVBi(Re,RJa,dtAug,tm,muSun,tol);
[V1o,V2o] = lambUVBi(Re,RJo,dtOct,tm,muSun,tol);
```

Long Way Lamberts Problem

```
tm = -1;

[V1mL,V2mL] = lambUVBi(Re,RJm,dtMar,tm,muSun,tol);
[V1aL,V2aL] = lambUVBi(Re,RJa,dtAug,tm,muSun,tol);
[V1oL,V2oL] = lambUVBi(Re,RJo,dtOct,tm,muSun,tol);
```

Parking Orbit Velocity

```
Rpark = Rearth + 500;
Vpark = sqrt(mu/Rpark);
```

Calculating Delta V For March - Departure Burn

```
Vinf = norm(V1m - Ve);
Vbo = sqrt((Vinf^2)+((2*mu)/Rpark)); %V burn out
dVm = abs(Vbo-Vpark);

Vinf = norm(V1mL - Ve);
Vbo = sqrt((Vinf^2)+((2*mu)/Rpark)); %V burn out
dVmL = abs(Vbo-Vpark);
```

Calculating Delta V For August - Departure Burn

```
Vinf = norm(V1a - Ve);
Vbo = sqrt((Vinf^2)+((2*mu)/Rpark)); %V burn out
dVa = abs(Vbo-Vpark);

Vinf = norm(V1aL - Ve);
Vbo = sqrt((Vinf^2)+((2*mu)/Rpark)); %V burn out
dVaL = abs(Vbo-Vpark);
```

Calculating Delta V For August - Departure Burn

```
Vinf = norm(V1o - Ve);
Vbo = sqrt((Vinf^2)+((2*mu)/Rpark)); %V burn out
dVo = abs(Vbo-Vpark);

Vinf = norm(V1oL - Ve);
Vbo = sqrt((Vinf^2)+((2*mu)/Rpark)); %V burn out
dVoL = abs(Vbo-Vpark);
```

Displaying all Delta V Burns for Departure

```
disp('Departure Delta V (km/s):')
disp(['March: ', num2str(dVm)])
disp(['March Long: ', num2str(dVmL)])
disp(['August: ', num2str(dVa)])
disp(['August Long: ', num2str(dVaL)])
disp(['October: ', num2str(dVo)])
disp(['October Long: ', num2str(dVoL)])
disp(' ')
```

```
Departure Delta V (km/s):
March: 61.9209
March Long: 7.0587
August: 61.7952
August Long: 6.5746
October: 61.6421
October Long: 6.6652
```

Capture Orbit Velocity

```
Rpark = Rjup + 20000;
Vpark = sqrt(muJup/Rpark);
```

Calculating Delta V For March - Arrival Burn

```
Vinf = norm(VJm - V2m);
Vbo = sqrt((Vinf^2)+((2*muJup)/Rpark)); %V burn out
dVma = abs(Vbo-Vpark);

Vinf = norm(VJm - V2mL);
Vbo = sqrt((Vinf^2)+((2*muJup)/Rpark)); %V burn out
dVmLa = abs(Vbo-Vpark);
```

Calculating Delta V For August - Arrival Burn

```
Vinf = norm(VJa - V2a);
Vbo = sqrt((Vinf^2)+((2*muJup)/Rpark)); %V burn out
```

```
dVaa = abs(Vbo-Vpark);

Vinf = norm(VJa - V2aL);
Vbo = sqrt((Vinf^2)+((2*muJup)/Rpark)); %V burn out
dVaLa = abs(Vbo-Vpark);
```

Calculating Delta V For August - Arrival Burn

```
Vinf = norm(VJo - V2o);
Vbo = sqrt((Vinf^2)+((2*muJup)/Rpark)); %V burn out
dVoa = abs(Vbo-Vpark);

Vinf = norm(VJo - V2oL);
Vbo = sqrt((Vinf^2)+((2*muJup)/Rpark)); %V burn out
dVoLa = abs(Vbo-Vpark);
```

Displaying all Delta V Burns for Arrival

```
disp('Arrial Delta V (km/s):')
disp(['March: ', num2str(dVma)])
disp(['March Long: ', num2str(dVmLa)])
disp(['August: ', num2str(dVaa)])
disp(['August Long: ', num2str(dVaLa)])
disp(['October: ', num2str(dVoa)])
disp(['October Long: ', num2str(dVoLa)])
disp(' ')
```

```
Arrial Delta V (km/s):
March: 18.9619
March Long: 15.727
August: 19.0318
August Long: 15.764
October: 19.0562
October Long: 15.7893
```

Adding All Delta V Burns

```
dVM = dVm + dVma;
dVML = dVmL + dVmLa;

dVA = dVa + dVaa;
dVAL = dVaL + dVaLa;

dVO = dVo + dVoa;
dVOL = dVoL + dVoLa;
```

Displaying Total Delta V Burns for each Departure

```
disp('Total Delta V (km/s):')
disp(['March: ', num2str(dVM)])
disp(['March Long: ', num2str(dVML)])
disp(['August: ', num2str(dVA)])
disp(['August Long: ', num2str(dVAL)])
disp(['October: ', num2str(dVO)])
disp(['October Long: ', num2str(dVOL)])
```

```
Total Delta V (km/s):
March: 80.8828
March Long: 22.7856
August: 80.827
August Long: 22.3386
October: 80.6983
October Long: 22.4544
```

Transfer Propagation - Short Way

```
stateM = [Re, V1m];
tspanMar = [0,dtMar];
[~,TM] = ode45(@twobodymotion,tspanMar,stateM,options,muSun);

stateA = [Re, V1a];
tspanAug = [0,dtAug];
[~,TA] = ode45(@twobodymotion,tspanAug,stateA,options,muSun);

stateO = [Re, V1o];
tspanOct = [0,dtOct];
[~,TO] = ode45(@twobodymotion,tspanOct,stateO,options,muSun);
```

Transfer Propagation - Long Way

```
stateML = [Re, V1mL];
tspanMar = [0,dtMar];
[~,TML] = ode45(@twobodymotion,tspanMar,stateML,options,muSun);

stateAL = [Re, V1aL];
tspanAug = [0,dtAug];
[~,TAL] = ode45(@twobodymotion,tspanAug,stateAL,options,muSun);

stateOL = [Re, V1oL];
tspanOct = [0,dtOct];
[~,TOL] = ode45(@twobodymotion,tspanOct,stateOL,options,muSun);
```

Planet Propagation For Plotting

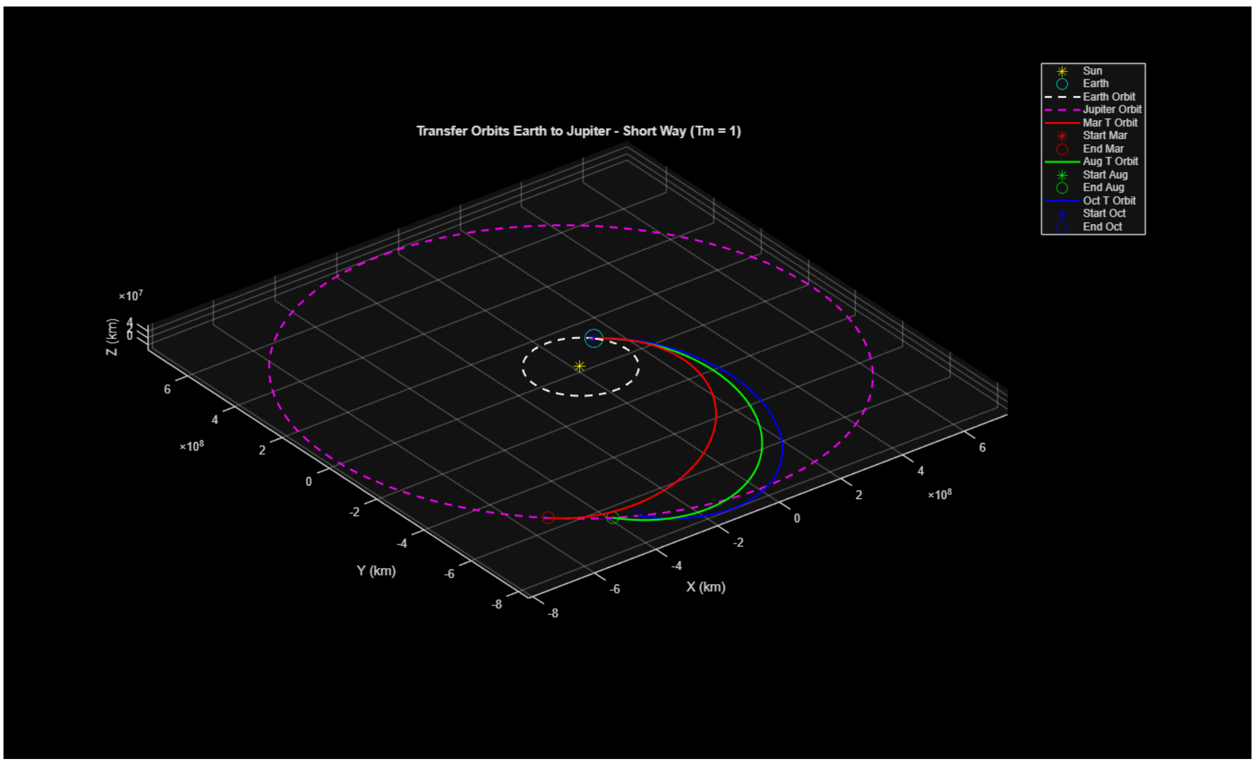
```
[~,~,~,~,~,~,~,pE] = rv2coes(Re,Ve,muSun,0);
[~,~,~,~,~,~,~,pJ] = rv2coes(RJm,VJm,muSun,0);
```

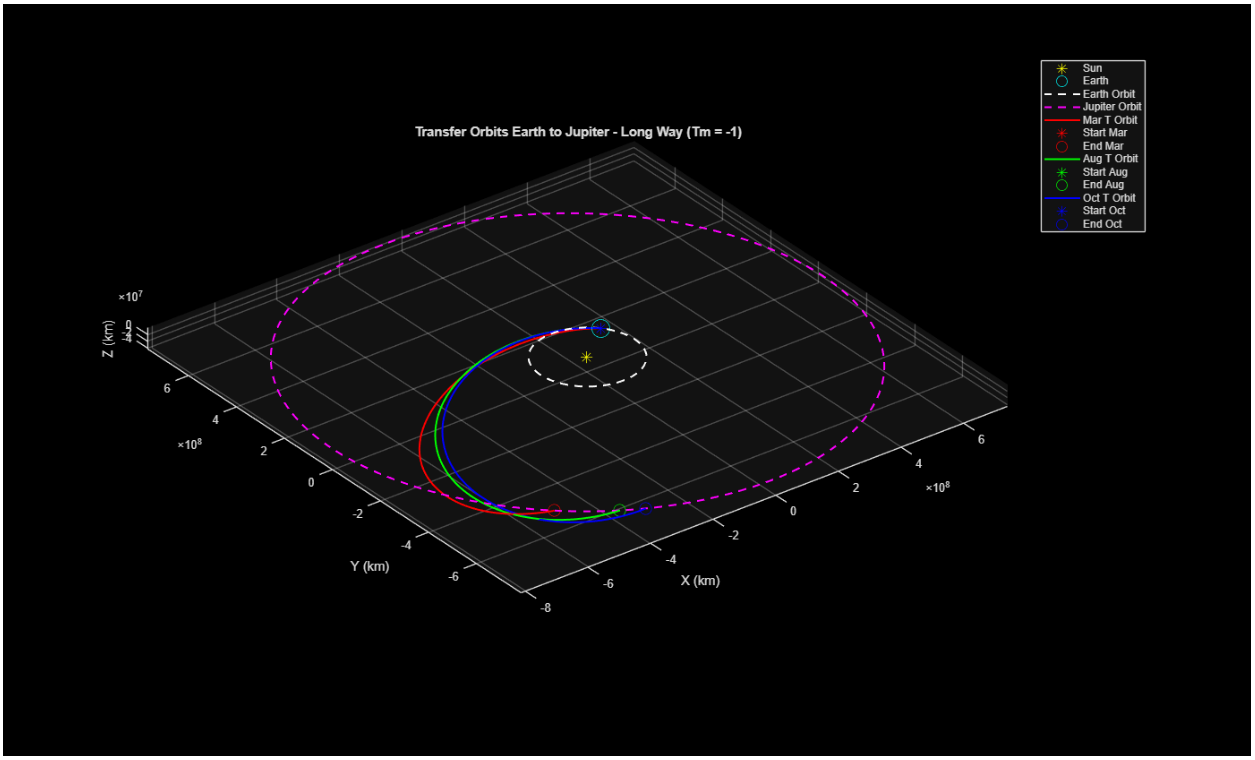
```
stateE = [Re, Ve];
tspanE = [0,pE];
[~,E] = ode45(@twobodymotion,tspanE,stateE,options,muSun);

stateE = [RJm, VJm];
tspanE = [0,pJ];
[~,J] = ode45(@twobodymotion,tspanE,stateE,options,muSun);
```

Plotting

```
figure('Name', 'Transfer Orbits Earth to Jupiter - Short Way (Tm = 1)');
plot3(0,0,0,'y*', 'MarkerSize',10); %Sun
hold on;
plot3(Re(1),Re(2),Re(3),'co','MarkerSize',15) %earth
plot3(E(:, 1), E(:, 2), E(:, 3), 'w', 'LineWidth', 1.5,'LineStyle','--');
%Earth orbit
plot3(J(:, 1), J(:, 2), J(:, 3), 'm', 'LineWidth', 1.5,'LineStyle','--');
%Jupiter Orbit

plot3(TM(:, 1), TM(:, 2), TM(:, 3), 'r', 'LineWidth', 1.5); %March T
plot3(TM(1,1),TM(1,2),TM(1,3),'r*', 'MarkerSize',10); %start March T
plot3(TM(end,1),TM(end,2),TM(end,3),'ro', 'MarkerSize',10); %end March T

plot3(TA(:, 1), TA(:, 2), TA(:, 3), 'g', 'LineWidth', 1.5); %Aug T
plot3(TA(1,1),TA(1,2),TA(1,3),'g*', 'MarkerSize',10); %start Aug T
plot3(TA(end,1),TA(end,2),TA(end,3),'go', 'MarkerSize',10); %end Aug T

plot3(TO(:, 1), TO(:, 2), TO(:, 3), 'b', 'LineWidth', 1.5); %Oct T
plot3(TO(1,1),TO(1,2),TO(1,3),'b*', 'MarkerSize',10); %start Oct T
plot3(TO(end,1),TO(end,2),TO(end,3),'bo', 'MarkerSize',10); %end Oct T

xlabel('X (km)');
ylabel('Y (km)');
zlabel('Z (km)');
grid on;
legend('Sun','Earth','Earth Orbit','Jupiter Orbit',...
    'Mar T Orbit','Start Mar','End Mar',...
    'Aug T Orbit','Start Aug','End Aug',...
    'Oct T Orbit','Start Oct','End Oct')
title('Transfer Orbits Earth to Jupiter - Short Way (Tm = 1)');
axis equal

figure('Name', 'Transfer Orbits Earth to Jupiter - Long Way (Tm = -1)');
plot3(0,0,0,'y*', 'MarkerSize',10); %Sun
hold on;
plot3(Re(1),Re(2),Re(3),'co','MarkerSize',15) %earth
plot3(E(:, 1), E(:, 2), E(:, 3), 'w', 'LineWidth', 1.5,'LineStyle','--');
%Earth orbit
plot3(J(:, 1), J(:, 2), J(:, 3), 'm', 'LineWidth', 1.5,'LineStyle','--');
%Jupiter Orbit

plot3(TML(:, 1), TML(:, 2), TML(:, 3), 'r', 'LineWidth', 1.5); %March T
```

```
plot3(TML(1,1),TML(1,2),TML(1,3),'r*', 'MarkerSize',10); %start March T
plot3(TML(end,1),TML(end,2),TML(end,3),'ro', 'MarkerSize',10); %end March T

plot3(TAL(:, 1), TAL(:, 2), TAL(:, 3), 'g', 'LineWidth', 1.5); %Aug T
plot3(TAL(1,1),TAL(1,2),TAL(1,3),'g*', 'MarkerSize',10); %start Aug T
plot3(TAL(end,1),TAL(end,2),TAL(end,3),'go', 'MarkerSize',10); %end Aug T

plot3(TOL(:, 1), TOL(:, 2), TOL(:, 3), 'b', 'LineWidth', 1.5); %Oct T
plot3(TOL(1,1),TOL(1,2),TOL(1,3),'b*', 'MarkerSize',10); %start Oct T
plot3(TOL(end,1),TOL(end,2),TOL(end,3),'bo', 'MarkerSize',10); %end Oct T

xlabel('X (km)');
ylabel('Y (km)');
zlabel('Z (km)');
grid on;
legend('Sun','Earth','Earth Orbit','Jupiter Orbit',...
    'Mar T Orbit','Start Mar','End Mar',...
    'Aug T Orbit','Start Aug','End Aug',...
    'Oct T Orbit','Start Oct','End Oct')
title('Transfer Orbits Earth to Jupiter - Long Way (Tm = -1)');
axis equal
```





Functions:

pcoes

```
function [a,ecc,inc,raan,w,theta,w_hat,L] = pcoes(pcoesVec)
    % Everything output is in degrees

    a = pcoesVec(1);
    ecc = pcoesVec(2);
    inc = pcoesVec(3);
    raan = pcoesVec(4);
    w_hat = pcoesVec(5);
    L = pcoesVec(6);

    Me = L - w_hat; %mean anamoly in deg
    Mer = deg2rad(Me); %mean anamoly in rad
    w = w_hat - raan; %arg of peri (deg)

    a2 = sqrt((1-ecc)/(1+ecc)); %not semi major axis
    E = Me2e(Mer,ecc); %eccentric anamoly
    theta = 2*atand(tan(E/2)/a2); %in deg

end
```

Me2e

```
function [EA] = Me2e(Me,ecc)
    %ME2E Summary of this function goes here
    %   Solves For Eccentric Anamoly (EA)
    %   Radians!

    syms E
    eq = Me == E - ecc*sin(E);
    EA = vpasolve(eq,E); %bounds ,[]
    EA = double(EA);

end
```

coes2rvd

```
function [R1,V1,R2,V2] = coes2rvd(a,ecc,inc,RAAN,ArgP,theta,mu)
    %COES2RV The outputs are the same except transposed
    %   for ease of use with 1x3 or 3x1 vectors
    %   (my old code used the first 2)
    %   Input COEs Get R & V

    h = (mu*(a*(1-ecc^2)))^(1/2);

    R = (h^2/mu)/(1+ecc*cosd(theta)) *[cosd(theta);sind(theta);0];
    V = (mu/h)*[-sind(theta);ecc+cosd(theta);0];

    [~,Q] = ECI2PERI(ArgP,inc,RAAN);

    R1 = Q*R;
    V1 = Q*V;

    R2 = R1';
    V2 = V1';
end
```

ECI2PERI

```
function [EtoP, PtoE] = ECI2PERI(omega,inc,RAAN)
    %ECI2PERI Earth Centered Inertial Frame to Perifocal Frame
    %   [EtoP, PtoE] = ECI2PERI(omega,inc,RAAN)

    Z = [cosd(omega),sind(omega),0;...
        -sind(omega),cosd(omega),0;...
        0,0,1];
    X = [1,0,0; ...
        0,cosd(inc), sind(inc);...
        0, -sind(inc), cosd(inc)];
    Z2 = [cosd(RAAN),sind(RAAN),0;...
        -sind(RAAN),cosd(RAAN),0;...
        0,0,1];
```

```
    EtoP = Z*X*Z2;
    PtoE = inv(EtoP);
end
```

rv2coes

```
function [hM,a,e,nu,i,RAAN,w,p,t,en,Ra,Rp] = rv2coes(R,V,mu,r)
%Function for finding orbital state vectors RV
%   Input is in SI & %ALL ANGLES IN RADIANS!!
%   [hM,a,e,nu,i,RAAN,w,p,t,en,Ra,Rp] = rv2coes(R,V,mu,r)
%   hM = specific angular momentum
%   a = semi-major axis
%   e = eccentricity
%   nu = true anamoly
%   i = inc
%   RAAN = Right angle asending node
%   w = argument of periapsis
%   p = period (s)
%   t = time since perigee passage
%   en = orbit energy
%   Ra = Radius of Apogee
%   Rp = Radius of Perigee
%   r = radius of orbiting planet


RM = norm(R); %Magnitude of R
VM = norm(V); %Magnitude of V

ui = [1,0,0];
uj = [0,1,0];
uk = [0,0,1];
h = cross(R,V);
h2 = dot(R,V);

uiM = norm(ui); %the magnitudes of the values above
ujM = norm(uj);
ukM = norm(uk);
hM = norm(h); %Calculating specific energy


% PART 1: Initial Calculations for later

ep = ((VM^2)/2)-((mu)/RM); %Calculating Epsilon (specific mechanical energy)
in J/kg


% PART 2: Calculating semi-major axis

a = -((mu)/(2*ep)); %in km

% PART 3: Genreal equation calculation for period

p = (2*pi)*sqrt((a^3)/(mu)); %period of orbit in seconds (ellipse & circ)
```

```
% PART 4: Calculating eccentricity
eV = (1/mu)*((((VM^2)-((mu)/(RM)))*R)-(dot(R,V)*V)); %eccentricity vector is
from origin to point of periapsis

e = norm(eV);

% PART 5: inclination in rad

i = acos((dot(uk,h))/((hM)*(ukM))); %in rad not deg


% PART 6: RAAN in rad

n = cross(uk,h); %projection of momentum vector in orbital plane and node
line?
nM = norm(n);

if n(2) >= 0
    RAAN = acos((dot(ui,n))/((uiM)*(nM))); %original equation
else
    RAAN = (2*pi)-(acos((dot(ui,n))/((uiM)*(nM))));
end

% PART 7: Argument of Periapsis in rad

if eV(3) >= 0 %k component of eccentricity vector (height)
    w = acos(dot(n,eV)/(nM*e));
else
    w = (2*pi)-(acos(dot(n,eV)/(nM*e)));
end

% PART 8: nu (or theta) true anomaly in rad

if h2 >= 0 %dot product of R and V idk what it represents
    nu = acos(dot(eV,R)/(e*RM));
else
    nu = (2*pi)-(acos(dot(eV,R)/(e*RM)));
end

% PART 9: Time since perigee passage

E = 2*atan(sqrt((1-e)/(1+e))*tan(nu/2));
Me = E - e*sin(E);
n = (2*pi)/p;
t = Me/n; %in seconds

if t < 0 %If it is negative it is other way around circle think 360-angle
    t = p + t; %this shows adding but it is adding a negative
end

% PART 10: Calculating Energy

energy = (VM^2)/2 - mu/RM; %km^2/s^2
```

```
en = energy;

% PART 11: Calculating Apogee and Perigee Altitude

Ra = a*(1+e)-r;
Rp = a*(1-e)-r;


end
```

twobodymotion

```
function dstate = twobodymotion(time,state,muEarth) %dstate is derivitve of
state
%FUNCTION put in descrip

    %define vars
    x = state(1);
    y = state(2);
    z = state(3);
    dx = state(4); %vel
    dy = state(5); %vel
    dz = state(6); %vel

    %mag of pos vector
    r = norm([x y z]);

    %accel: !!eqs of motion!!
    ddx = -muEarth*x/r^3;
    ddy = -muEarth*y/r^3;
    ddz = -muEarth*z/r^3;

    dstate = [dx; dy; dz; ddx; ddy; ddz];

end
```

Lamberts

```
function [V1,V2] = lambUVBi(R1,R2,dtime,Tm,mu,tol)
    %LAMBUVBI: Lambert Orbit Transfer With Universal Variable
    %   [V1,V2] = lambUVBi(R1,R2,dtime,Tm,mu,tol)
    %
    %   Given position 1 and 2, the time in between and mu
    %   Returns Velocity at positions 1 and 2
    %   Tm is 1 or -1, decides between long and short transfers
    %
    %   This function uses the bisection method to iterate

    R1n = norm(R1);
    R2n = norm(R2);
    Z = 0;
    C = 1/2;
    S = 1/6;
```

```
    Zu = 4*pi^2; %upper bound
    Zl = -4*pi^2; %lower bound
    dtl = 1; %change in time of loop (random guess)

    %deltaTheta = acos((dot(R1,R2)/(R1n*R2n))); %Alt eqs for A
    %A = sin(deltaTheta)*sqrt((R1n*R2n)/(1-cos(deltaTheta)));

    A = Tm*sqrt(R1n*R2n*(1+(dot(R1,R2)/(R1n*R2n))));

    while abs(dtl-dtime) > tol
        Y = R1n+R2n+(A*((Z*S-1)/sqrt(C)));
        UV = sqrt(Y/C);
        dtl = (((UV^3)*S)/sqrt(mu))+((A*sqrt(Y))/sqrt(mu));

        if dtl < dtime
            Zl = Z; %reset zlower
        elseif dtl > dtime
            Zu = Z; %reset zupper
        end

        Z = 0.5*(Zu+Zl); %update z to midpoint
        [C,S] = stumpff(Z); %update stumpff c(z) s(z)

        f = 1-(((UV^2)/R1n)*C);
        g = dtl - (((UV^3)/sqrt(mu)) * S);
        fd = (sqrt(mu)/(R1n*R2n))*UV*((Z*S)-1);
        gd = 1-(((UV^2)/R2n)*C);
        %<3: f*gd - fd*g = 1

        for i = 1:3
            V1(i) = (1/g)*(R2(i)-f*R1(i));
            V2(i) = (fd*R1(i))+(gd*V1(i));
        end

    end
end
```

stumpff

```
function [C, S] = stumpff(z)
    %STUMPFF: Given Z will calculate new C(z) and S(z)
    %   [C, S] = stumpff(z)

    if z > 0
        S = (sqrt(z)-sin(sqrt(z)))/((sqrt(z))^3);
        C = (1-cos(sqrt(z)))/z;
    elseif z < 0
        S = (sinh(sqrt(-z))-sqrt(-z))/((sqrt(-z))^-3);
        C = (cosh(sqrt(-z))-1)/-z;
    elseif z == 0
        S = 1/6;
        C = 1/2;
    else
```

```
        error('stumpff broke? (not a number?)')
    end
end
```

AERO351planetary_elements2

```
function [planet_coes] = AERO351planetary_elements2(planet_id,T)
% Planetary Ephemerides from Meeus (1991:202-204) and J2000.0
% Output:
% planet_coes
% a = semimajor axis (km)
% ecc = eccentricity
% inc = inclination (degrees)
% raan = right ascension of the ascending node (degrees)
% w_hat = longitude of perihelion (degrees)
% L = mean longitude (degrees)

% Inputs:
% planet_id - planet identifier:
% 1 = Mercury
% 2 = Venus
% 3 = Earth
% 4 = Mars
% 5 = Jupiter
% 6 = Saturn
% 7 = Uranus
% 8 = Neptune

if planet_id == 1
    a = 0.387098310; % AU but in km later
    ecc = 0.20563175 + 0.000020406*T - 0.0000000284*T^2 - 0.00000000017*T^3;
    inc = 7.004986 - 0.0059516*T + 0.00000081*T^2 + 0.000000041*T^3; %degs
    raan = 48.330893 - 0.1254229*T-0.00008833*T^2 - 0.000000196*T^3; %degs
    w_hat = 77.456119 +0.1588643*T -0.00001343*T^2+0.000000039*T^3; %degs
    L = 252.250906+149472.6746358*T-0.00000535*T^2+0.000000002*T^3; %degs
elseif planet_id == 2
    a = 0.723329820; % AU
    ecc = 0.00677188 - 0.000047766*T + 0.000000097*T^2 + 0.00000000044*T^3;
    inc = 3.394662 - 0.0008568*T - 0.00003244*T^2 + 0.000000010*T^3; %degs
    raan = 76.679920 - 0.2780080*T-0.00014256*T^2 - 0.000000198*T^3; %degs
    w_hat = 131.563707 +0.0048646*T -0.00138232*T^2-0.000005332*T^3; %degs
    L = 181.979801+58517.8156760*T+0.00000165*T^2-0.000000002*T^3; %degs
elseif planet_id == 3
    a = 1.000001018; % AU
    ecc = 0.01670862 - 0.000042037*T - 0.0000001236*T^2 + 0.00000000004*T^3;
    inc = 0.0000000 + 0.0130546*T - 0.00000931*T^2 - 0.000000034*T^3; %degs
    raan = 0.0; %degs
    w_hat = 102.937348 + 0.3225557*T + 0.00015026*T^2 + 0.000000478*T^3;
%degs
    L = 100.466449 + 35999.372851*T - 0.00000568*T^2 + 0.000000000*T^3; %degs
elseif planet_id == 4
    a = 1.523679342; % AU
    ecc = 0.09340062 + 0.000090483*T - 0.00000000806*T^2 - 0.00000000035*T^3;
```

```
    inc = 1.849726 - 0.0081479*T - 0.00002255*T^2 - 0.000000027*T^3; %degs
    raan = 49.558093 - 0.2949846*T-0.00063993*T^2 - 0.000002143*T^3; %degs
    w_hat = 336.060234 +0.4438898*T -0.00017321*T^2+0.000000300*T^3; %degs
    L = 355.433275+19140.2993313*T+0.00000261*T^2-0.000000003*T^3; %degs
elseif planet_id == 5
    a = 5.202603191 + 0.0000001913*T; % AU
    ecc = 0.04849485+0.000163244*T - 0.0000004719*T^2 + 0.00000000197*T^3;
    inc = 1.303270 - 0.0019872*T + 0.00003318*T^2 + 0.000000092*T^3; %degs
    raan = 100.464441 + 0.1766828*T+0.00090387*T^2 - 0.000007032*T^3; %degs
    w_hat = 14.331309 +0.2155525*T +0.00072252*T^2-0.000004590*T^3; %degs
    L = 34.351484+3034.9056746*T-0.00008501*T^2+0.000000004*T^3; %degs
elseif planet_id == 6
    a = 9.5549009596 - 0.0000021389*T; % AU
    ecc = 0.05550862 - 0.000346818*T -0.0000006456*T^2 + 0.00000000338*T^3;
    inc = 2.488878 + 0.0025515*T - 0.00004903*T^2 + 0.000000018*T^3; %degs
    raan = 113.665524 - 0.2566649*T-0.00018345*T^2 + 0.000000357*T^3; %degs
    w_hat = 93.056787 +0.5665496*T +0.00052809*T^2-0.000004882*T^3; %degs
    L = 50.077471+1222.1137943*T+0.00021004*T^2-0.000000019*T^3; %degs
elseif planet_id == 7
    a = 19.218446062-0.0000000372*T+0.00000000098*T^2; % AU
    ecc = 0.04629590 - 0.000027337*T + 0.0000000790*T^2 + 0.00000000025*T^3;
    inc = 0.773196 - 0.0016869*T + 0.00000349*T^2 + 0.00000000016*T^3; %degs
    raan = 74.005947 + 0.0741461*T+0.00040540*T^2 +0.000000104*T^3; %degs
    w_hat = 173.005159 +0.0893206*T -0.00009470*T^2+0.000000413*T^3; %degs
    L = 314.055005+428.4669983*T-0.00000486*T^2-0.000000006*T^3; %degs
elseif planet_id == 8
    a = 30.110386869-0.0000001663*T+0.00000000069*T^2; % AU
    ecc = 0.00898809 + 0.000006408*T -0.0000000008*T^2;
    inc = 1.769952 +0.0002557*T +0.00000023*T^2 -0.0000000000*T^3; %degs
    raan = 131.784057 - 0.0061651*T-0.00000219*T^2 - 0.000000078*T^3; %degs
    w_hat = 48.123691 +0.0291587*T +0.00007051*T^2-0.000000000*T^3; %degs
    L = 304.348665+218.4862002*T+0.00000059*T^2-0.000000002*T^3; %degs
end

planet_coes = [a;ecc;inc;raan;w_hat;L];
%Convert to km:
au = 149597870;
planet_coes(1) = planet_coes(1)*au;
end
```

Published with MATLAB® R2024b

Table of Contents

Roshan Jaiswal-Ferri

```
%Section - 01
%Aero 351 Final Exam Question 2: 12/07/24
```

Workspace Prep

```
format long       %Allows for more accurate decimals
close all;        %Clears all
clear all;        %Clears Workspace
clc;              %Clears Command Window
```

Constant/Global Vars

```
options = odeset('RelTol',1e-8,'AbsTol',1e-8);
muSun = 1.327e11; %mu values from curtis
muMars = 42828;
mu = 398600; %earth
muJup = 126686534;
Rmars = 3396; %km
Rearth = 6378; %km
Rjup = 71490;
tol = 1e-8;
ap = 149598000; %1 AU in km
P = (0.75*365.25)*86400; %period in seconds
Re = 149598000;
flyr = Rearth + 10000; %km
```

Finding COEs of Elliptical Orbit

```
a = ((muSun*P^2)/(4*pi^2))^(1/3);
ecc = (ap-a)/a;
theta = 180; %at apogee true anamoly is 180 deg
h = sqrt(ap*muSun*(1+ecc*cosd(theta)));
```

Calculating Fly By Characteristics

```
Vsc1 = h/ap;
Ve = sqrt(muSun/Re);
```

```
Vinf = abs(Vsc1 - Ve);
phi = 180; %ht ellipse where planet V is faster therefor phi = 180
ecch = 1 + (flyr*(Vinf^2))/mu; %is hyperbolic
```

Finding Turn Angle & phi_2

```
ta = 2*asind(1/ecch); %turn angle
phi2 = phi - ta;
```

Finding V S/C 2

```
Vinf2 = [Vinf*cosd(phi2), Vinf*sind(phi2)];
Vsc2 = Vinf2 + [Ve,0];
Vsc1 = [Vsc1, 0];

dV = norm(Vsc1 - Vsc2); %<3: This is under the earth max so its ok
disp(['Gained Delta V (km/s): ', num2str(dV)])
```

```
Gained Delta V (km/s): 4.5784
```

Published with MATLAB® R2024b

Table of Contents

Roshan Jaiswal-Ferri

```
%Section - 01
%Aero 351 Final Exam Question 3: 12/07/24
```

Workspace Prep

```
format long      %Allows for more accurate decimals
close all;       %Clears all
clear all;       %Clears Workspace
clc;             %Clears Command Window
```

Constant/Global Vars

```
options = odeset('RelTol',1e-8,'AbsTol',1e-8);
muSun = 1.327e11; %mu values from curtis
muMars = 42828;
mu = 398600; %earth
muJup = 126686534;
Rmars = 3396; %km
Rearth = 6378; %km
Rjup = 71490;
tol = 1e-7;
T = -0.006; %thrust in kN, ISP & Thrust negative because we are slowing down
Isp = -5000; % (ISP = F/(mdot*g0) ) so if F (thrust) is neg Isp is too
g = 9.807e-3; % gravity in km/s
mi = 600; %initial mass 600 kilos
```

Defining State Vectors & Propogation of 3.5 Day Burn

```
r1 = [26578,0,0];
v1 = [0,3.8726,0];
```

```
state = [r1,v1,mi];
tspan1 = [0,3.5*86400]; %tspan of burn one for 3.5 days in seconds
[~,burn1] = ode45(@contThrust,tspan1,state,options, T, Isp, mu, g);
```

Coast Phase - 5 Days

```
r2 = [burn1(end,1),burn1(end,2),burn1(end,3)];
v2 = [burn1(end,4),burn1(end,5),burn1(end,6)];
m2 = burn1(end,7); %new mass after burning fuel
state2 = [r2,v2];
tspan2 = [0,5*86400]; %five days in seconds
[~,coast] = ode45(@twobodymotion,tspan2,state2,options,mu);
```

Burn 2 - 0.6 Days

```
r3 = [coast(end,1),coast(end,2),coast(end,3)];
v3 = [coast(end,4),coast(end,5),coast(end,6)];
state3 = [r3,v3,m2];
tspan3 = [0,86400*0.6]; %0.6 days in seconds
[~,burn2] = ode45(@contThrust,tspan3,state3,options, T, Isp, mu, g);
```

Finding New Coes

```
rf = [burn2(end,1),burn2(end,2),burn2(end,3)];
vf = [burn2(end,4),burn2(end,5),burn2(end,6)];
mf = burn2(end,7);
[~,~,~,~,~,~,~,~,~,~,~,Altp] = rv2coes(rf,vf,mu,Rearth);
```

Display Results

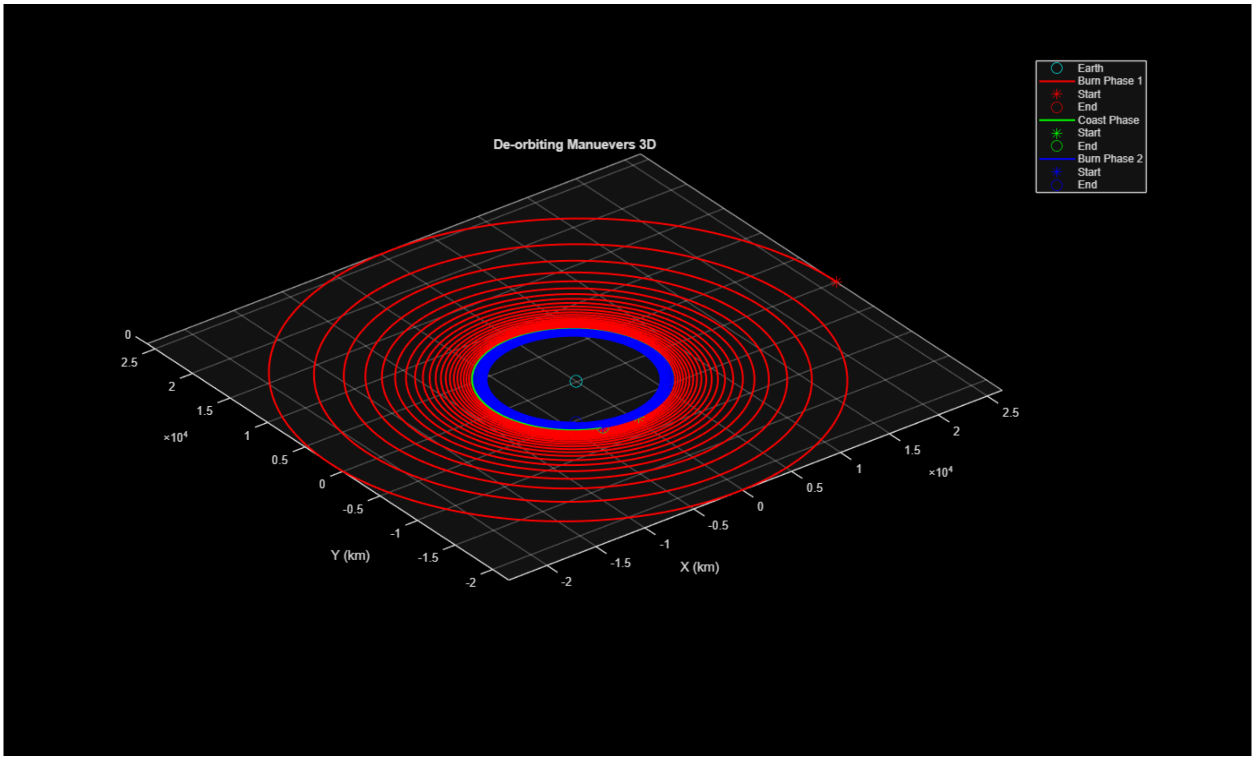
```
disp(['Final Mass (kg): ', num2str(mf)])
disp(['Altitude of Perigee (km): ', num2str(Altp)])
```

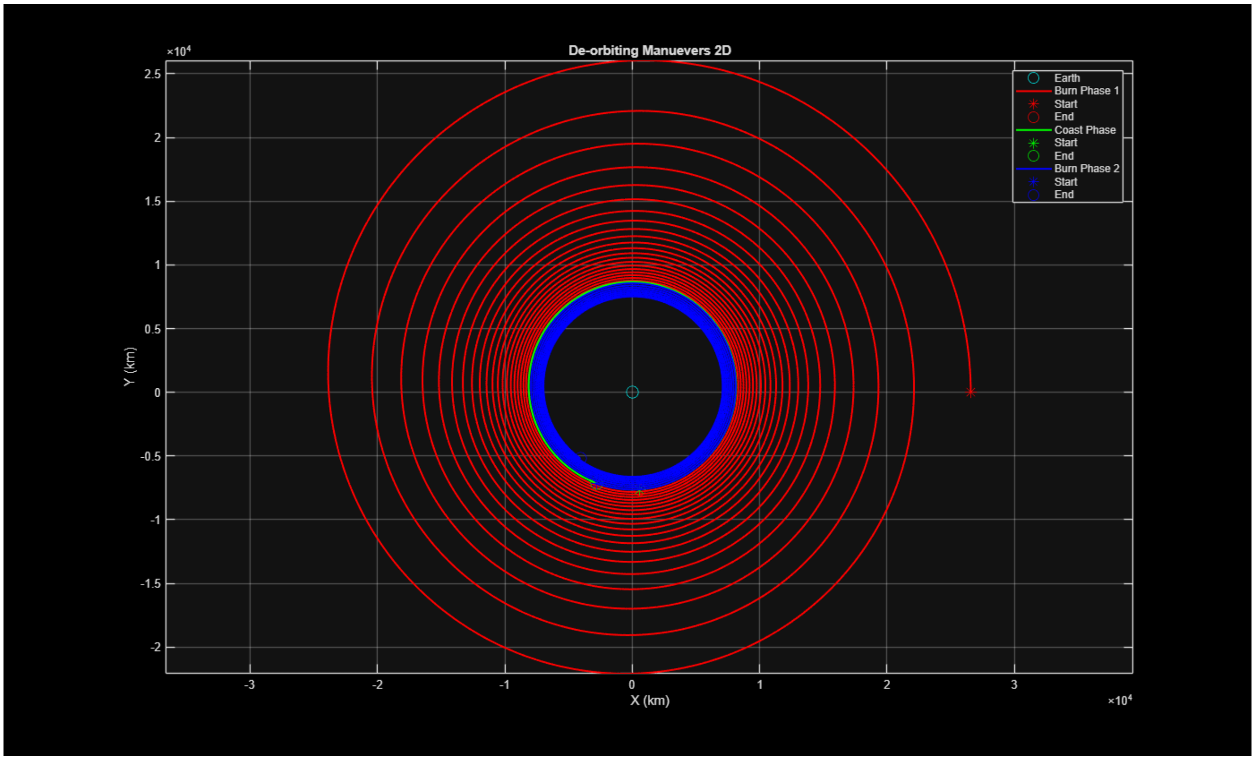
```
Final Mass (kg): 556.6546
Altitude of Perigee (km): 143.9152
```

Plotting

```
figure('Name', 'De-orbiting Manuevers 3D');
plot3(0,0,0,'co', 'MarkerSize',10); %Earth
hold on;

plot3(burn1(:, 1), burn1(:, 2), burn1(:, 3), 'r', 'LineWidth', 1.5); %Thr 1
plot3(burn1(1,1),burn1(1,2),burn1(1,3),'r*', 'MarkerSize',10); %start thr 1
plot3(burn1(end,1),burn1(end,2),burn1(end,3),'ro', 'MarkerSize',10); %end

plot3(coast(:, 1), coast(:, 2), coast(:, 3), 'g', 'LineWidth', 1.5); %coast
plot3(coast(1,1),coast(1,2),coast(1,3),'g*', 'MarkerSize',10); %start
plot3(coast(end,1),coast(end,2),coast(end,3),'go', 'MarkerSize',10); %end
```

```
plot3(burn2(:, 1), burn2(:, 2), burn2(:, 3), 'b', 'LineWidth', 1.5); %Thr 2
plot3(burn2(1,1),burn2(1,2),burn2(1,3),'b*', 'MarkerSize',10); %start
plot3(burn2(end,1),burn2(end,2),burn2(end,3),'bo', 'MarkerSize',10); %end

xlabel('X (km)');
ylabel('Y (km)');
zlabel('Z (km)');
grid on;
legend('Earth',...
    'Burn Phase 1','Start','End',...
    'Coast Phase','Start','End',...
    'Burn Phase 2','Start','End')
title('De-orbiting Manuevers 3D');
axis equal

figure('Name', 'De-orbiting Manuevers 2D');
plot(0,0,'co', 'MarkerSize',10); %Earth
hold on;

plot(burn1(:, 1), burn1(:, 2), 'r', 'LineWidth', 1.5); %Thr 1
plot(burn1(1,1),burn1(1,2),'r*', 'MarkerSize',10); %start thr 1
plot(burn1(end,1),burn1(end,2),'ro', 'MarkerSize',10); %end

plot(coast(:, 1), coast(:, 2), 'g', 'LineWidth', 1.5); %coast
plot(coast(1,1),coast(1,2),'g*', 'MarkerSize',10); %start
plot(coast(end,1),coast(end,2),'go', 'MarkerSize',10); %end

plot(burn2(:, 1), burn2(:, 2), 'b', 'LineWidth', 1.5); %Thr 2
plot(burn2(1,1),burn2(1,2),'b*', 'MarkerSize',10); %start
plot(burn2(end,1),burn2(end,2),'bo', 'MarkerSize',10); %end

xlabel('X (km)');
ylabel('Y (km)');
zlabel('Z (km)');
grid on;
legend('Earth',...
    'Burn Phase 1','Start','End',...
    'Coast Phase','Start','End',...
    'Burn Phase 2','Start','End')
title('De-orbiting Manuevers 2D');
axis equal
```


De-orbiting Manuevers 3D
Earth
Burn Phase 1
Start
End
Coast Phase
Start
End
Burn Phase 2
Start
End
Y (km)
X (km)
×10^4
×10^4


De-orbiting Manuevers 2D
Earth
Burn Phase 1
Start
End
Coast Phase
Start
End
Burn Phase 2
Start
End
Y (km)
X (km)
×10^4
×10^4

Functions:

Continuous Thrust

```
function [tstate] = contThrust(time, state, T, Isp, mu, g0)
    %CONTTHRUST: Propogates a new orbit trajectory during thrusting phase
    %   [tstate] = contThrust(time, state, T, Isp, mu, g0)
    %
    %   Make sure g0 (usually 9.807 m/s^2) is in km/s^2

    x = state(1);
    y = state(2);
    z = state(3);
    dx = state(4);
    dy = state(5);
    dz = state(6);
    m = state(7);
    rMag = norm([x y z]);
    vMag = norm([dx dy dz]);
    ddx = -mu*x/rMag^3 +(T/m)*dx/vMag;
    ddy = -mu*y/rMag^3 +(T/m)*dy/vMag;
    ddz = -mu*z/rMag^3 +(T/m)*dz/vMag;
    mdot = -T/(g0*Isp);


    tstate = [dx;dy;dz;ddx;ddy;ddz;mdot];

end
```

twobodymotion

```
function dstate = twobodymotion(time,state,muEarth) %dstate is derivitve of
state
%FUNCTION put in descrip

    %define vars
    x = state(1);
    y = state(2);
    z = state(3);
    dx = state(4); %vel
    dy = state(5); %vel
    dz = state(6); %vel

    %mag of pos vector
    r = norm([x y z]);

    %accel: !!eqs of motion!!
    ddx = -muEarth*x/r^3;
    ddy = -muEarth*y/r^3;
    ddz = -muEarth*z/r^3;
```

```
    dstate = [dx; dy; dz; ddx; ddy; ddz];

end
```

rv2coes

```
function [hM,a,e,nu,i,RAAN,w,p,t,en,Alta,Altp] = rv2coes(R,V,mu,r)
%Function for finding orbital state vectors RV
%   Input is in SI & %ALL ANGLES IN RADIANS!!
%   [hM,a,e,nu,i,RAAN,w,p,t,en,Ra,Rp] = rv2coes(R,V,mu,r)
%   hM = specific angular momentum
%   a = semi-major axis
%   e = eccentricity
%   nu = true anamoly
%   i = inc
%   RAAN = Right angle asending node
%   w = argument of periapsis
%   p = period (s)
%   t = time since perigee passage
%   en = orbit energy
%   Ra = Radius of Apogee
%   Rp = Radius of Perigee
%   r = radius of orbiting planet


RM = norm(R); %Magnitude of R
VM = norm(V); %Magnitude of V

ui = [1,0,0];
uj = [0,1,0];
uk = [0,0,1];
h = cross(R,V);
h2 = dot(R,V);

uiM = norm(ui); %the magnitudes of the values above
ujM = norm(uj);
ukM = norm(uk);
hM = norm(h); %Calculating specific energy


% PART 1: Initial Calculations for later

ep = ((VM^2)/2)-((mu)/RM); %Calculating Epsilon (specific mechanical energy)
in J/kg


% PART 2: Calculating semi-major axis

a = -((mu)/(2*ep)); %in km

% PART 3: Genreal equation calculation for period

p = (2*pi)*sqrt((a^3)/(mu)); %period of orbit in seconds (ellipse & circ)
```

```matlab
% PART 4: Calculating eccentricity
eV = (1/mu)*((((VM^2)-((mu)/(RM)))*R)-(dot(R,V)*V)); %eccentricity vector is
from origin to point of periapsis

e = norm(eV);

% PART 5: inclination in rad

i = acos((dot(uk,h))/((hM)*(ukM))); %in rad not deg


% PART 6: RAAN in rad

n = cross(uk,h); %projection of momentum vector in orbital plane and node
line?
nM = norm(n);

if n(2) >= 0
    RAAN = acos((dot(ui,n))/((uiM)*(nM))); %original equation
else
    RAAN = (2*pi)-(acos((dot(ui,n))/((uiM)*(nM))));
end

% PART 7: Argument of Periapsis in rad

if eV(3) >= 0 %k component of eccentricity vector (height)
    w = acos(dot(n,eV)/(nM*e));
else
    w = (2*pi)-(acos(dot(n,eV)/(nM*e)));
end

% PART 8: nu (or theta) true anomaly in rad

if h2 >= 0 %dot product of R and V idk what it represents
    nu = acos(dot(eV,R)/(e*RM));
else
    nu = (2*pi)-(acos(dot(eV,R)/(e*RM)));
end

% PART 9: Time since perigee passage

E = 2*atan(sqrt((1-e)/(1+e))*tan(nu/2));
Me = E - e*sin(E);
n = (2*pi)/p;
t = Me/n; %in seconds

if t < 0 %If it is negative it is other way around circle think 360-angle
    t = p + t; %this shows adding but it is adding a negative
end

% PART 10: Calculating Energy

energy = (VM^2)/2 - mu/RM; %km^2/s^2
```

```
en = energy;

% PART 11: Calculating Apogee and Perigee Altitude

Alta = a*(1+e)-r;
Altp = a*(1-e)-r;


end
```

Published with MATLAB® R2024b

Table of Contents

Roshan Jaiswal-Ferri

```
%Section - 01
%Aero 351 Final Exam Question 4: 12/07/24
```

Workspace Prep

```
format long      %Allows for more accurate decimals
close all;       %Clears all
clear all;       %Clears Workspace
clc;             %Clears Command Window
```

Constant/Global Vars

```
options = odeset('RelTol',1e-8,'AbsTol',1e-8);
muSun = 1.327e11; %mu values from curtis
muMars = 42828;
mu = 398600; %earth
Rmars = 3396; %km
Rearth = 6378; %km
tol = 1e-7;
Rpark = Rearth + 200;
```

Creating R & V Vectors

```
Vpark = sqrt(mu/Rpark);
Vesc = sqrt(2)*Vpark;
R = [Rpark,0,0];
V = [0,Vpark,0] + [0,1.075,0]; %account for first burn
```

Counting Burns

```
time = 0;
burns = 1; %add one extra for first burn

while V < Vesc
    [~,~,~,~,~,~,~,p] = rv2coes(R,V,mu,Rearth);
```

```
    time = time + p;
    V = V + [0,1.075,0];
    burns = burns + 1;
end
```

Display Results

```
disp(['Parking Velocity (km/s): ', num2str(Vpark)])
disp(['Escape Velocity (km/s): ', num2str(Vesc)])
disp(['Num of Perigee Burns: ', num2str(burns)])
disp(['Final Velocity at Perigee (km/s): ', num2str(norm(V))])
disp(['Total Time (hrs): ', num2str(time/3600)])
```

```
Parking Velocity (km/s): 7.7843
Escape Velocity (km/s): 11.0087
Num of Perigee Burns: 3
Final Velocity at Perigee (km/s): 11.0093
Total Time (hrs): 9.011
```

Functions:

rv2coes

```
function [hM,a,e,nu,i,RAAN,w,p,t,en,Alta,Altp] = rv2coes(R,V,mu,r)
%Function for finding orbital state vectors RV
%   Input is in SI & %ALL ANGLES IN RADIANS!!
%   [hM,a,e,nu,i,RAAN,w,p,t,en,Ra,Rp] = rv2coes(R,V,mu,r)
%   hM = specific angular momentum
%   a = semi-major axis
%   e = eccentricity
%   nu = true anamoly
%   i = inc
%   RAAN = Right angle asending node
%   w = argument of periapsis
%   p = period (s)
%   t = time since perigee passage
%   en = orbit energy
%   Ra = Radius of Apogee
%   Rp = Radius of Perigee
%   r = radius of orbiting planet


RM = norm(R); %Magnitude of R
VM = norm(V); %Magnitude of V

ui = [1,0,0];
uj = [0,1,0];
uk = [0,0,1];
h = cross(R,V);
h2 = dot(R,V);

uiM = norm(ui); %the magnitudes of the values above
```

```
ujM = norm(uj);
ukM = norm(uk);
hM = norm(h); %Calculating specific energy


% PART 1: Initial Calculations for later

ep = ((VM^2)/2)-((mu)/RM); %Calculating Epsilon (specific mechanical energy)
in J/kg


% PART 2: Calculating semi-major axis

a = -((mu)/(2*ep)); %in km

% PART 3: Genreal equation calculation for period

p = (2*pi)*sqrt((a^3)/(mu)); %period of orbit in seconds (ellipse & circ)

% PART 4: Calculating eccentricity
eV = (1/mu)*((((VM^2)-((mu)/(RM)))*R)-(dot(R,V)*V)); %eccentricity vector is
from origin to point of periapsis

e = norm(eV);

% PART 5: inclination in rad

i = acos((dot(uk,h))/((hM)*(ukM))); %in rad not deg


% PART 6: RAAN in rad

n = cross(uk,h); %projection of momentum vector in orbital plane and node
line?
nM = norm(n);

if n(2) >= 0
    RAAN = acos((dot(ui,n))/((uiM)*(nM))); %original equation
else
    RAAN = (2*pi)-(acos((dot(ui,n))/((uiM)*(nM))));
end

% PART 7: Argument of Periapsis in rad

if eV(3) >= 0 %k component of eccentricity vector (height)
    w = acos(dot(n,eV)/(nM*e));
else
    w = (2*pi)-(acos(dot(n,eV)/(nM*e)));
end

% PART 8: nu (or theta) true anomaly in rad

if h2 >= 0 %dot product of R and V idk what it represents
    nu = acos(dot(eV,R)/(e*RM));
```

```
else
    nu = (2*pi)-(acos(dot(eV,R)/(e*RM)));
end

% PART 9: Time since perigee passage

E = 2*atan(sqrt((1-e)/(1+e))*tan(nu/2));
Me = E - e*sin(E);
n = (2*pi)/p;
t = Me/n; %in seconds

if t < 0 %If it is negative it is other way around circle think 360-angle
    t = p + t; %this shows adding but it is adding a negative
end

% PART 10: Calculating Energy

energy = (VM^2)/2 - mu/RM; %km^2/s^2
en = energy;

% PART 11: Calculating Apogee and Perigee Altitude

Alta = a*(1+e)-r;
Altp = a*(1-e)-r;


end
```

Published with MATLAB® R2024b