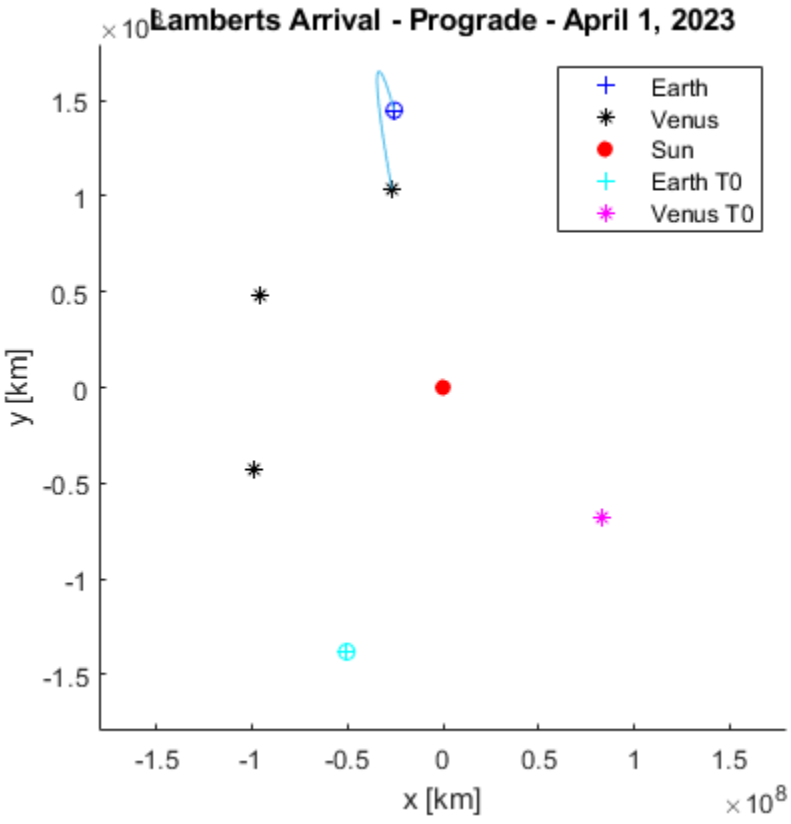
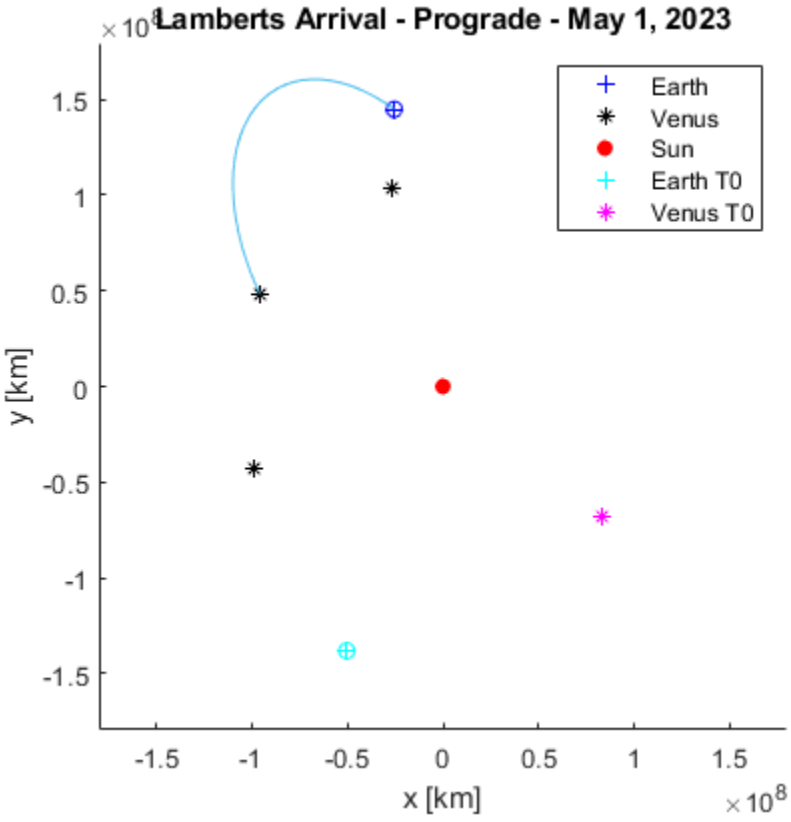
Table of Contents

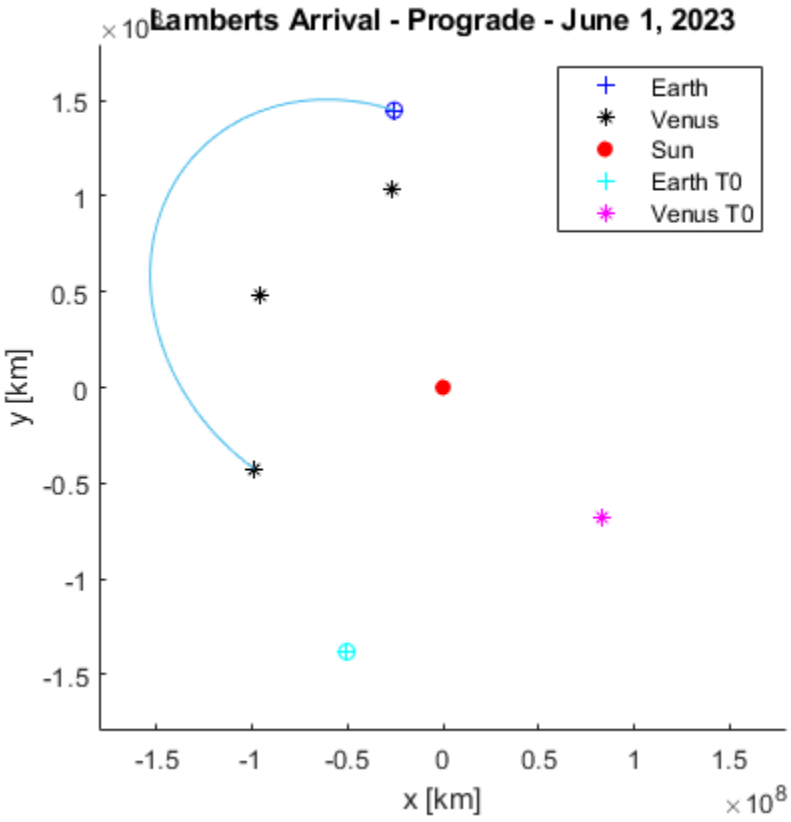
cleanup

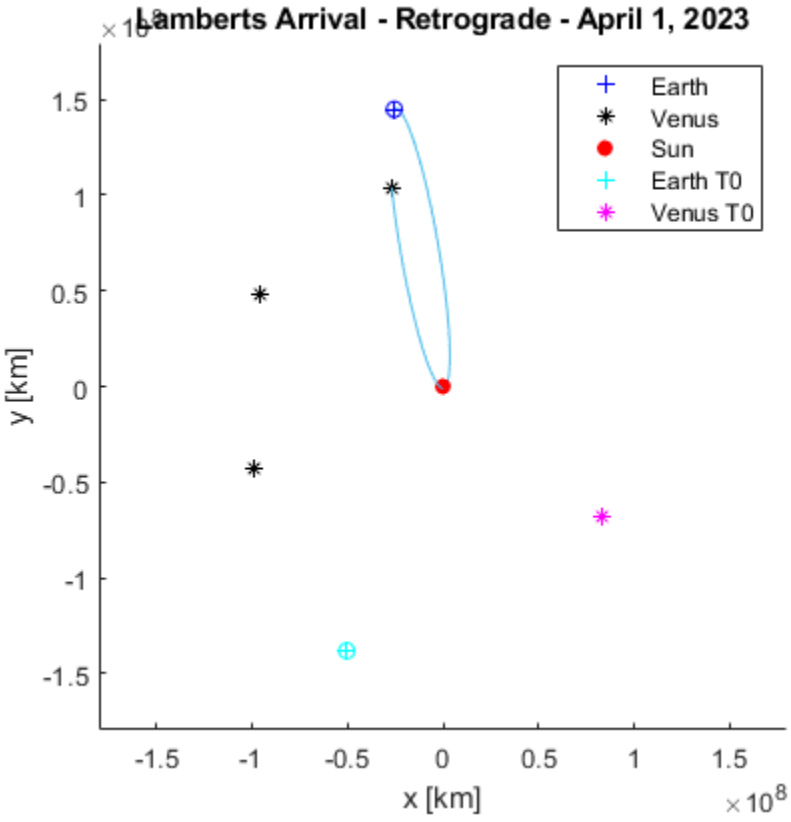
1 - three dates to venus

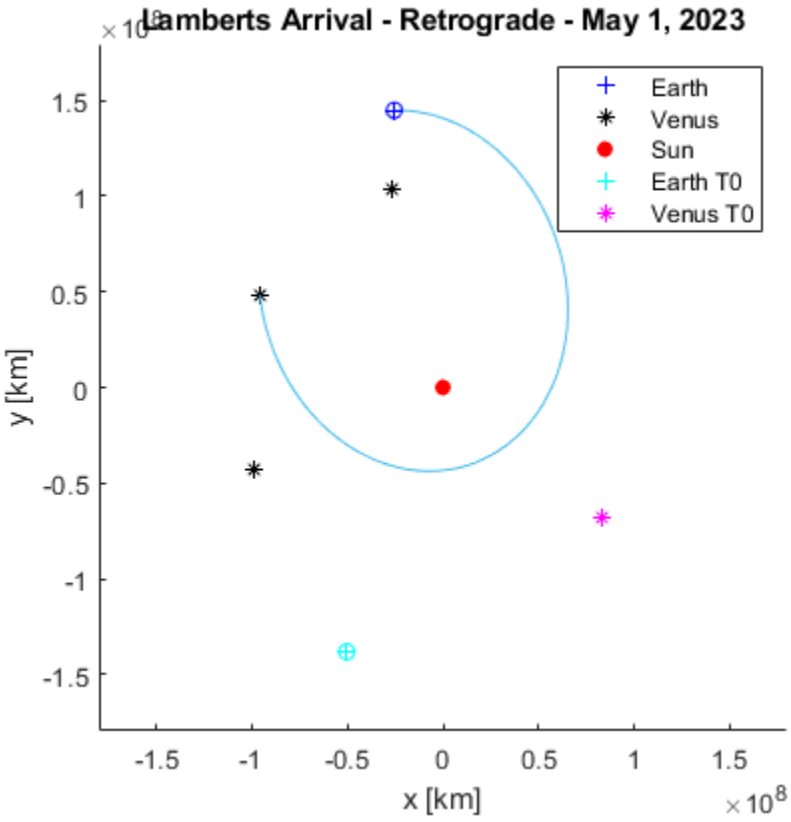
```
Warning: JoshBisection: the function handle f returns complex numbers for
 tested
values. JoshBisection was written for use on real functions only, but will
 still
return a result that may be useful.
Warning: JoshBisection: the function handle f returns complex numbers for
 tested
values. JoshBisection was written for use on real functions only, but will
 still
return a result that may be useful.
Warning: JoshBisection: the function handle f returns complex numbers for
 tested
values. JoshBisection was written for use on real functions only, but will
 still
return a result that may be useful.
------------P1------------
My workings for this problem have the following results:
The minimum dv for the trip is 11.2931 km/s with the mission plan: Prograde -
 June 1, 2023
The all possible dvs are as follows:
    "Prograde - April 1, 2023 - 14.8392 km/s"
    "Prograde - May 1, 2023 - 12.7519 km/s"
    "Prograde - June 1, 2023 - 11.2931 km/s"
    "Retrograde - April 1, 2023 - 16.5178 km/s"
    "Retrograde - May 1, 2023 - 16.6872 km/s"
    "Retrograde - June 1, 2023 - 16.6759 km/s"

the radius of perihelion of the mission plan Retrograde - April 1, 2023 is:
 1074308.3818 km, which is 378308.3818 km under the surface of the sun, so I
 wouldn't fly that one.
H/C: Best dv happens on date closest to Hohmaan, i.e. closest to 180 degrees
 from eachother
H/C: long/retrograde method is more expensive in dv than prograde.
```

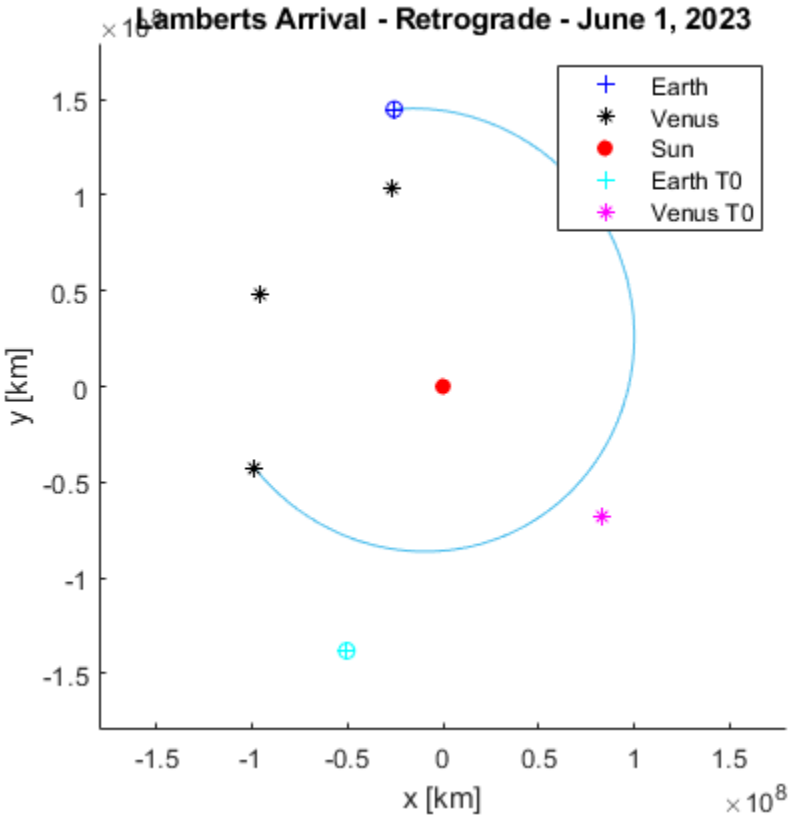

Lamberts Arrival - Prograde - April 1, 2023
Earth
Venus
Sun
Earth T0
Venus T0
y [km]
x [km]
×10^8


Lamberts Arrival - Prograde - May 1, 2023
Earth
Venus
Sun
Earth T0
Venus T0
y [km]
x [km]
×10^8


Lamberts Arrival - Prograde - June 1, 2023
Earth
Venus
Sun
Earth T0
Venus T0
y [km]
x [km]
×10^8


Lamberts Arrival - Retrograde - April 1, 2023
Earth
Venus
Sun
Earth T0
Venus T0
y [km]
x [km]
×10^8

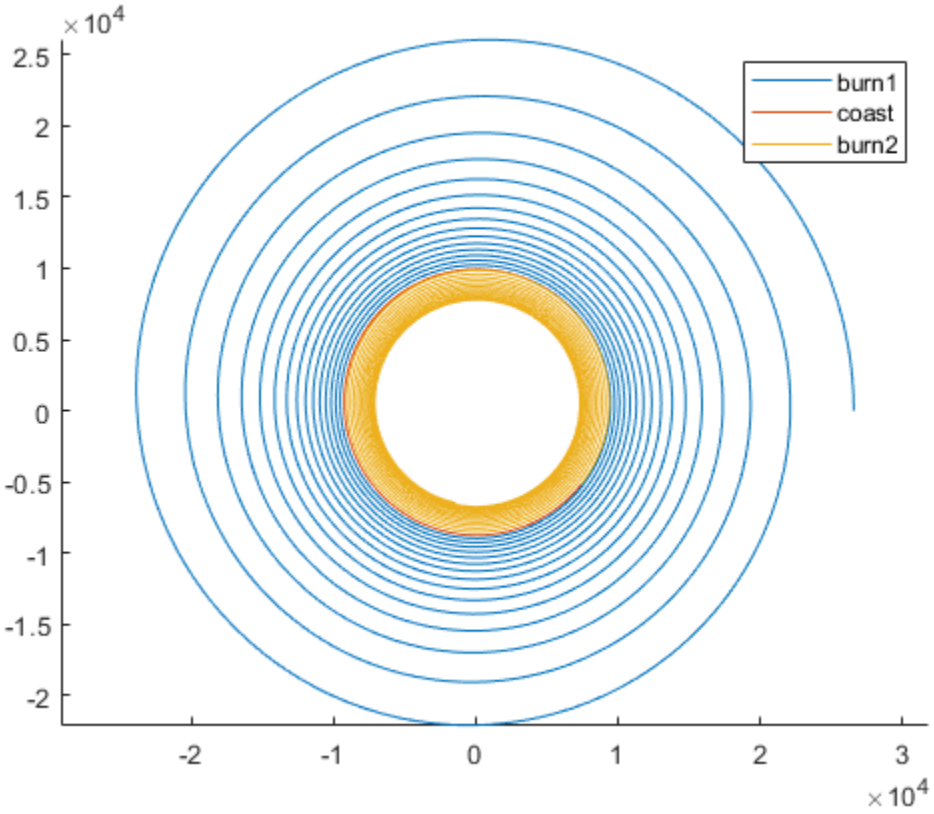

Lamberts Arrival - Retrograde - May 1, 2023
Earth
Venus
Sun
Earth T0
Venus T0
y [km]
x [km]
×10^8


Lamberts Arrival - Retrograde - June 1, 2023
Earth
Venus
Sun
Earth T0
Venus T0
y [km]
x [km]
×10^8

2 - flyby

```
------------P2------------
My workings for this problem have the following results:
The resulting heliocentric speed is: 29.7776 km/s
The dv of this this manuver is: 4.5783 km/s
The change in heliocentric speed is: 3.3286 km/s, so the s/c is going faster
 after the flyby.
H/C: increase in heliocentric speed makes sense for a trailing edge flyby
H/C: dv is within ranges we'd expect for an Earth flyby
```

3 - deorbit, nonimpulsive



4 - multiburn escape

```
Warning: joshCOE will assume that R and V are normal if the inputs are scalar
ie: the craft is in a circular orbit or is at periapse or apoapse
Warning: joshCOE will assume that R and V are normal if the inputs are scalar
ie: the craft is in a circular orbit or is at periapse or apoapse
------------P4------------
My workings for this problem have the following results:
The spacecraft will require 3 burns to escape.
The total time from the first burn to the last is: 9.011 hrs.
H/C this time seems reasonable for middle altitude orbits.
```

5 - ecc in terms of c

```
------------P5------------
My workings for this problem have the following results:
I used symbolic mathtoolbox to keep track of my variables.
From problem statement:

vp =

c*va

ecc formula:

ecc =

(ra - rp)/(ra + rp)

Solve for ra in terms of other vars from h formula:

h =

ra*va == c*rp*va


ra =

c*rp

plug ra and into rp into ecc formula:

ecc =

-(rp - c*rp)/(rp + c*rp)

simplify to find ecc in terms of c:

ecc =

(c - 1)/(c + 1)

H/C: ecc is only in terms of c
```

6 - trip to neptune Tsyn

```
------------P6------------
My workings for this problem have the following results:
The synodic period of Neptune relative to Eath: 367.4969 days
The transfer time for a Homaan transfer to Neptune is: 30.5819 Earth years.
The lead angle of Neptune at departure is: 112.9787 degrees
The lead angle of Neptune at arrival is: -29.3704 degrees, ie it is lagging by
 29.3704 degrees
```

```
H/C: The transfer time seems really long but it is actually resoable
 considering the orbital period of Neptune is: 5200777912.32 Earth years.
H/C: Neptunr is leading Earth at departure which makes sense because it will
 travel some but less than half of its orbit in 30 years.
H/C: the synodic period is greater than the wait time. This makes sense
 beacuse the synodic period is an upper bound for possible wait times.
```

dependancies

```
------------Dependancies------------
My code uses the following functions:
```

funcitons

function [r,v] = prop(r0,v0,dt) X0 = [r0;v0]; options = odeset('RelTol', 1e-8,'AbsTol',1e-13); [~,X] = ode45(@orbitODEFun,[0,dt],X0,options); X = X(end,:); r = X(1:3)'; v = X(4:6)'; end

```
------------P3------------
My workings for this problem have the following results:
The final altitude of the spacecraft is: 251.4879 km/s, so it is still
 orbiting.
The final mass of the spacecraft is 557.7103 kg, which is 42.2897 kg less than
 the initial mass.
H/C: the final altitude is lower which is expected for a deorbit.
H/C: the final mass is lower which is expected for consuming fuel.
H/C: the trajectory is a fairly circular spiral which is expected for low
 thrust/long duration burns.
```

Published with MATLAB® R2022a

Table of Contents

cleanup

```
clear all
close all
clc

% addpath("C:\joshFunctionsMatlab\")
```

1 - three dates to venus

```
clear all

au = 149598000;

mu_sun = 132712440018;

mu_e = 398600;
mu_v = 324859;

r_e = 6378; % km
r_v = 6052; % km

tString = "January 1, 2023";
t = datetime(tString);
[jd(1),~ ,~, ~, ~, ~, T0] =joshJulian(t);

tString = "April 1, 2023";
t = datetime(tString);
% [~,~,T1]= joshJulian(t);
[jd(2),~ ,~, ~, ~, ~, T1] =joshJulian(t);
tString = "May 1, 2023";
t = datetime(tString);
% [~,~,T2]= joshJulian(t);
[jd(3),~ ,~, ~, ~, ~, T2] =joshJulian(t);
tString = "June 1, 2023";
t = datetime(tString);
% [~,~,T3]= joshJulian(t);
[jd(4),~ ,~, ~, ~, ~, T3] =joshJulian(t);
```

```
clear t tString



[coes(:,1)] = abercrombyAERO351planetary_elements2(3,T0);
[coes(:,2)] = abercrombyAERO351planetary_elements2(2,T1);
[coes(:,3)] = abercrombyAERO351planetary_elements2(2,T2);
[coes(:,4)] = abercrombyAERO351planetary_elements2(2,T3);

[coes(:,5)] = abercrombyAERO351planetary_elements2(3,T3);
[coes(:,6)] = abercrombyAERO351planetary_elements2(2,T0);


for i =1:6
    a(i) = coes(1,i);
    ecc(i) = coes(2,i);
    inc(i) = deg2rad(coes(3,i));
    raan(i) = deg2rad(coes(4,i));
    w_hat(i) = deg2rad(coes(5,i));
    L(i) = deg2rad(coes(6,i));
    aop(i) = (w_hat(i)-raan(i));
    Me(i) = (L(i) - w_hat(i));
    [theta(i),~,E(i)] = joshAnomalyCalculator(ecc(i),Me(i),"Me");

 [r(:,i),v(:,i)]=joshCOE2rv(a(i),ecc(i),theta(i),inc(i),raan(i),aop(i),mu_sun);
end
clear a ecc inc raan w_hat aop Me


for i = 2:4
    dt = (jd(i)-jd(1))*24*3600;
    [v1(:,i-1),v2(:,i-1)] =
 joshIntegratedLamberts(r(:,1),r(:,i),dt,"pro",mu_sun);
    [v1(:,i+2),v2(:,i+2)] =
 joshIntegratedLamberts(r(:,1),r(:,i),dt,"retro",mu_sun);
end

for i = 1:3
    vinf(:,i) = [norm(v(:,1)-v1(:,i));norm(v(:,i+1)-v2(:,i))]; % from pro
    vinf(:,i+3) = [norm(v(:,1)-v1(:,i+3));norm(v(:,i+1)-v2(:,i+3))]; % from
 retro
end

rp_d = 500+r_e;
rp_a = 200+r_v;
ra_a = 10000+r_v;
vbo_a = sqrt(vinf(1,:).^2+2*mu_v/rp_a);
vbo_d = sqrt(vinf(2,:).^2+2*mu_e/rp_d);

ecc_a = (ra_a-rp_a)/(ra_a+rp_a);
vp_d = sqrt(mu_e/rp_d);
h_a = sqrt(rp_a*mu_v*(1-ecc_a^2));

dv_d = vbo_d-vinf(1,:);
```

```
dv_a = vbo_a-vinf(1,:);

dv = [dv_d;dv_a];
for i = 1:6
    dvtot(i) = sum(dv(:,i));
end


X={};
% [~,~,~,X{1}]=joshOrbitCoastOde(r(:,1),v(:,1),(jd(4)-jd(1))*24*3600,mu_sun);
 % planets coasting
% [~,~,~,X{2}]=joshOrbitCoastOde(r(:,6),v(:,6),(jd(4)-jd(1))*24*3600,mu_sun);


for i = 1:3
    [~,~,~,X{i+2}]=joshOrbitCoastOde(r(:,1),v1(:,i),(jd(i+1)-
jd(1))*24*3600,mu_sun); % spacecraft coasting
    [~,~,~,X{i+5}]=joshOrbitCoastOde(r(:,1),v1(:,i+3),(jd(i+1)-
jd(1))*24*3600,mu_sun); % spacecraft coasting
end

myString = ["April 1, 2023","May 1, 2023","June 1, 2023"];
myString = ["Prograde - " + myString,"Retrograde - " + myString];

for i = 3:8
    figure
    title("Lamberts Arrival - "+myString(i-2))
    hold on

    scatter3(r(1,1),r(2,1),r(3,1),'blue','+')
    scatter3(r(1,2:4),r(2,2:4),r(3,2:4),'black','*')
    scatter3(0,0,0,'red','filled')

    scatter3(r(1,5),r(2,5),r(3,5),'cyan','+')
    scatter3(r(1,6),r(2,6),r(3,6),'magenta','*')
    plot3(X{i}(:,1),X{i}(:,2),X{i}(:,3))

    axis('equal')
    xlabel("x [km]")
    ylabel("y [km]")
    zlabel("z [km]")
    xlim(1.2*au*[-1, 1])
    ylim(1.2*au*[-1, 1])
    zlim(1.2*au*[-1, 1])
    scatter3(r(1,1),r(2,1),r(3,1),'blue','o')
    scatter3(r(1,5),r(2,5),r(3,5),'cyan','o')
    legend("Earth","Venus","Sun","Earth T0","Venus T0")
end


[a,ecc,theta,inc,raan,aop,h,T,E] =joshCOE(r(:,1),v1(:,4),mu_sun,"magnitude");

ra = norm(r(:,1));
rp = a*2-ra;
```

```
r_sun =  696000;

[m,i]=min(dvtot);

disp("------------P1------------")
disp("My workings for this problem have the following results:")
disp("The minimum dv for the trip is "+string(m)+" km/s with the mission plan:
 "+myString(i))
disp("The all possible dvs are as follows: ")
disp((myString+" - "+dvtot+" km/s")')
disp("the radius of perihelion of the mission plan "+myString(4)+" is:
 "+string(rp)+" km, which is "+string(abs(r_sun-rp))+" km under the surface of
 the sun, so I wouldn't fly that one.")
disp("H/C: Best dv happens on date closest to Hohmaan, i.e. closest to 180
 degrees from eachother")
disp("H/C: long/retrograde method is more expensive in dv than prograde.")
```

2 - flyby

```
clear all
mu_e = 398600;
mu_h = 132712440018;
r_e = 6378; % km

yr = 365.242*24*3600; % sec in a year
T = (3/4)*yr;
rah = 149598000;

ah = (T*sqrt(mu_h)/(2*pi))^(2/3);
rph = 2*ah-rah;
ecch = (rah-rph)/(rah+rph);

h0 = sqrt(ah*(1-ecch^2)*mu_h);

vah = h0/rah;
vah = [0;vah];
v0 = norm(vah);

ve = sqrt(mu_h/rah);
ve = [0;ve];

vinf1 = vah-ve;

zpe = 10000;
rpe = zpe+r_e;

ecce = 1+(rpe*norm(vinf1)^2)/mu_e;
beta = acos(1/ecce);

vinf2 = norm(vinf1)*[-sin(2*beta);cos(2*beta)];

vh2 = vinf2+ve;
finalSpeed = norm(vh2);
```

```
dv = norm(vh2-vah);

disp("------------P2------------")
disp("My workings for this problem have the following results:")
disp("The resulting heliocentric speed is: "+string(finalSpeed)+" km/s")
disp("The dv of this this manuver is: "+string(dv)+" km/s")
disp("The change in heliocentric speed is: "+string(finalSpeed-norm(vah))+"
 km/s, so the s/c is going faster after the flyby.")
disp("H/C: increase in heliocentric speed makes sense for a trailing edge
 flyby")
disp("H/C: dv is within ranges we'd expect for an Earth flyby")
```

3 - deorbit, nonimpulsive

```
clear all
r_e = 6378; % km

ISP = 5000;
r0 = [26578;0;0];
v0 = [0;3.8726;0];
m0 = 600; %kg
X0 = [r0;v0;m0];

figure
hold on
axis('equal')


options = odeset('RelTol', 1e-8,'AbsTol',1e-13);
[t,X] = ode45(@orbitODEFun,[0,3*24*60*60],X0,options,1);
Xs = [X];
ts = [t];
X0 = Xs(end,:);
plot3(X(:,1),X(:,2),X(:,3))


[t,X] = ode45(@orbitODEFun,[0,4*24*60*60],X0,options,0);
Xs = [Xs;X];
ts = [ts;t+ts(end)];
X0 = Xs(end,:);
plot3(X(:,1),X(:,2),X(:,3))



[t,X] = ode45(@orbitODEFun,[0,1*24*60*60],X0,options,1);
Xs = [Xs;X];
ts = [ts;t+ts(end)];
plot3(X(:,1),X(:,2),X(:,3))
legend("burn1","coast","burn2")

X = Xs(end,:);
rfinal = norm(Xs(end,1:3));
zfinal = rfinal-r_e;
```

```
mfinal = X(7);
disp("------------P3------------")
disp("My workings for this problem have the following results:")
disp("The final altitude of the spacecraft is: "+string(zfinal)+" km/s, so it
 is still orbiting.")
disp("The final mass of the spacecraft is "+string(mfinal)+" kg, which is
 "+string(m0-mfinal)+" kg less than the initial mass.")
disp("H/C: the final altitude is lower which is expected for a deorbit.")
disp("H/C: the final mass is lower which is expected for consuming fuel.")
disp("H/C: the trajectory is a fairly circular spiral which is expected for
 low thrust/long duration burns.")
```

4 - multiburn escape

```
clear all
mu_e = 398600;
r_e = 6378; % km
z0 = 200;

r0 = r_e+z0; % initial r
v0 = sqrt(mu_e/r0);
vinf = sqrt(2)*v0; % vescape = root(2)*vcirc
vinc = 1.075; % incremental increase in v at each perigee
vinf = vinf-v0; % total dv required
n = ceil(vinf/vinc); % the number of vinc burns needed at perigee to escape.
 rounded up since partial burns don't make sense

vp1 = v0+vinc; % velocity after first burn
vp2 = v0+2*vinc; % velocity after second burn
% vp3 = v0+2*vinc; % velocity after thrid burn - not needed since problem ends
 after 3rd burn

[~,~,~,~,~,~,~,T1] = joshCOE(r0,vp1,mu_e,'magnitude');
[~,~,~,~,~,~,~,T2] = joshCOE(r0,vp2,mu_e,'magnitude');
totTime = T1+T2;
totTime= totTime/(60*60); % hrs
disp("------------P4------------")
disp("My workings for this problem have the following results:")
disp("The spacecraft will require "+string(n)+" burns to escape.")
disp("The total time from the first burn to the last is: "+string(totTime)+"
 hrs.")
disp("H/C this time seems reasonable for middle altitude orbits.")
```

5 - ecc in terms of c

```
disp("------------P5------------")
disp("My workings for this problem have the following results:")
disp("I used symbolic mathtoolbox to keep track of my variables.")
syms c va rp ra
disp("From problem statement:")
vp = c*va
disp("ecc formula:")
ecc = (ra-rp)/(ra+rp)
```

```
disp("Solve for ra in terms of other vars from h formula:")
h = va*ra == vp*rp
ra = solve(h,ra)
disp("plug ra and into rp into ecc formula:")
ecc = subs(ecc)
disp("simplify to find ecc in terms of c:")
ecc = simplify(ecc)
disp("H/C: ecc is only in terms of c")
```

6 - trip to neptune Tsyn

```
T_e = 365.256; %days
T_n= 164.8; %earth years
T_n = T_n*T_e; %days
T_e = T_e*24*3600; % seconds
T_n = T_n*24*3600; % seconds

r_n = 4.495e9; %km
r_e = 149.6e6; %km
mu_sun = 132.71*10^9; %km^3/s^2

n_e = 0.9856; %deg/day
n_e = n_e/(24*3600); % deg/s
n_e = deg2rad(n_e); % rad/s

n_n = 0.0060; %deg/day
n_n = n_n/(24*3600); % deg/s
n_n = deg2rad(n_n); % rad/s


Tsyn = 2*pi/(n_e-n_n);
% Tsyn = Tsyn/(24*3600) %solar days

v_e = sqrt(mu_sun/r_e);
v_n = sqrt(mu_sun/r_n);

[dv1,dv2,dv,T,ht,ecct,vt1,vt2] = joshHomann(r_e,v_e,r_n,v_n,mu_sun);

dtheta_e = n_e*T;
dtheta_n = n_n*T;

theta1_e = 0;
[~,theta2_e] = joshQuadrant(dtheta_e);

theta1_n = pi-dtheta_n;
theta2_n = pi;

nLeadDepart = theta1_n - theta1_e;
nLeadArrive = theta2_n-theta2_e;

n_rel = n_e-n_n;
dtheta_en1 = 2*pi+nLeadArrive;
thetaCatchup = dtheta_en1-nLeadDepart;
```

```
Twait = thetaCatchup/n_rel;

disp("------------P6------------")
disp("My workings for this problem have the following results:")
disp("The synodic period of Neptune relative to Eath: "+string(Tsyn/
(24*3600))+" days")
disp("The transfer time for a Homaan transfer to Neptune is: "+string(T/
(T_e))+" Earth years.")
disp("The lead angle of Neptune at departure is:
 "+string(rad2deg(nLeadDepart))+" degrees")
disp("The lead angle of Neptune at arrival is:
 "+string(rad2deg(nLeadArrive))+" degrees, ie it is lagging by
 "+string(rad2deg(-nLeadArrive))+" degrees")
disp("H/C: The transfer time seems really long but it is actually resoable
 considering the orbital period of Neptune is: "+string(T_n)+" Earth years.")
disp("H/C: Neptunr is leading Earth at departure which makes sense because it
 will travel some but less than half of its orbit in 30 years.")
disp("H/C: the synodic period is greater than the wait time. This makes sense
 beacuse the synodic period is an upper bound for possible wait times.")
```

dependancies

```
disp("------------Dependancies------------")
disp("My code uses the following functions: ")
depends = matlab.codetools.requiredFilesAndProducts('C:\AERO351\FinalFinal
\Final.m');
% disp(depends')
```

funcitons

function [r,v] = prop(r0,v0,dt) X0 = [r0;v0]; options = odeset('RelTol', 1e-8,'AbsTol',1e-13); [~,X] = ode45(@orbitODEFun,[0,dt],X0,options); X = X(end,:); r = X(1:3)'; v = X(4:6)'; end

```
function Xdot = orbitODEFun(t,X,burn)
mu = 398600;
g0 =  9.80665; % m/s^2
g0 = g0/1000; % km/s^2
ISP = 5000;

r = X(1:3);
v = X(4:6);
m = X(7);
uv = v/norm(v);
if burn
    T = 6; % N = kg.m/s^s
    T = T/1000; % kN = kg.km/s^2
else
    T = 0;
end
mdot = -T/(ISP*g0);
vdot = (-mu/norm(r)^3)*r - (T/m)*uv; % minus because retrograde burn
(T/m)*uv;
rdot = v;
```

```
Xdot = [rdot;vdot;mdot];
end

function [r, count, output] = JoshBisection(f,bracket,TOL)
% this function (algorithm sourced from canvas) will use the bisection method
 to solve roots given a
% braket using Bisection Method
% f must have a single root on the interval bracket or this function will
% not return the expected root bracket
% JoshBisection is kinda janky but it works some of the time *shrug*

% Check inputs
arguments
    f
    bracket (1,2) {mustBeNumeric,mustBeReal}
    TOL (1,1) {mustBeNumeric,mustBeReal} = .1
end
if ~isa(f,'function_handle')
    throw(MException("JoshBisection:invalidInput","f must be a
 function_handle"))
end
% set vars
a = bracket(1);
b = bracket(2);

if ((sign(f(a)) == sign(f(b))) || (a > b))
    error('JoshBisection:invalidInput','f must not have the same sign on both
 sides of the bracket')
end

% create output array
output = [a, b, f((a+b)/2)];
% start count = 0
count = 0;

% while err estimate > tol

isComplex = 0;
while ((b - a)/2 > TOL)
    % increment count
    count = count + 1;
    % create guess in middle of range
    c = (a + b)/2;
    % check if guess is root
    if f(c) == 0
        break;
    end
    % check if root is on left or right of guess
    if ~isreal(f(a)) || ~isreal(f(b)) || ~isreal(f(c))
        isComplex = 1;
    end

    if (f(a)*f(c) < 0)
```

```
        b = c;
    else
        a = c;
    end
    % update output array by appending new output
    output = [output; [a b f((a+b)/2)]];
end
% after err<=tol , return r is in middle of last range
if isComplex
    warning("JoshBisection: the function handle f returns complex numbers for
 tested values. JoshBisection was written for use on real functions only, but
 will still return a result that may be useful.")
end

r = (a+b)/2;
end

function [planet_coes] = abercrombyAERO351planetary_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:
% T0 must be in julian centuries, i.e.

% planet_id - planet identifier:
% 1 = Mercury
% 2 = Venus
% 3 = Earth
% 4 = Mars
% 5 = Jupiter
% 6 = Saturn
% 7 = Uranus
% 8 = Neptune
if abs(T)>1
warning("abercrombyAERO351planetary_elements2: T should be in
 centuries, and most likely should be less than 1 century from J2000.
 abercrombyAERO351planetary_elements2 will return but answers are likely
 incorrect.")
end

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

function [Cx,Cy,Cz] = joshAxisRotation(opt)
arguments
        opt {mustBeMember(opt,{'degree','radian'})} = 'radian'
end


if strcmp(opt,'degree')
Cx = @(theta)...
    [[1 0 0];...
    [0 cosd(theta) sind(theta)];...
    [0 -sind(theta) cosd(theta)]];

Cy = @(theta)...
    [[cosd(theta) 0 -sind(theta)];...
    [ 0 1 0];...
    [sind(theta) 0 cosd(theta)]];

Cz = @(theta)...
    [[cosd(theta) sind(theta) 0];...
    [-sind(theta) cosd(theta) 0];...
    [0 0 1]];
else
Cx = @(theta)...
    [[1 0 0];...
    [0 cos(theta) sin(theta)];...
    [0 -sin(theta) cos(theta)]];

Cy = @(theta)...
    [[cos(theta) 0 -sin(theta)];...
    [ 0 1 0];...
    [sin(theta) 0 cos(theta)]];

Cz = @(theta)...
    [[cos(theta) sin(theta) 0];...
    [-sin(theta) cos(theta) 0];...
    [0 0 1]];
end
end

function [a,ecc,theta,inc,raan,aop,h,T,E] = joshCOE(R,V,u,magOrVec)

%%%%%%%%%%%%%%%%%%
% Revamped to do rads and fit new naming convention %
%%%%%%%%%%%%%%%%%%


%COESOATESJOSHUA Takes postion and velocity vector and returns COES, all in
%ECI frame of reffrenece, km and seconds as units and degrees
% a = semi major axis
% ecc = eccentricity
% i = inclination
```

```
% raan = right accention acending node
% aop = argument of periapsis
% theta = true anomaly

% will return T in s as a period
% and E (sometimes epsilon) in km^2/s^2 as specific mechanical energy
% h is agular momentum

% magOrVec is a parameter that can be set for vector inputs to return
% vector h and ecc

% scalar entry should only be used if the spacecraft is at apoapse or
% periapse

arguments
    R {mustBeNumeric, mustBeReal}
    V  {mustBeNumeric, mustBeReal}
    u (1,1) {mustBeNumeric, mustBeReal, mustBePositive} = 3.986004418 *
 (10^5) %km^3/s^2
    magOrVec {mustBeMember(magOrVec,{'magnitude','vector'})} = 'magnitude'
end

[m1,n1] = size(R);
[m2,n2] = size(V);

if joshIsOnes([m1 n1 m2 n2])
    magOrVec = 'magnitude';
    R = [0 0 R];
    V = [V 0 0];
    warning("joshCOE will assume that R and V are normal if the inputs are
 scalar ie: the craft is in a circular orbit or is at periapse or apoapse")
elseif (~joshIsOnes([m1 n1] == [m2 n2]))|~((n1==1&m1==3)|(n1==3&m1==1))
    throw(MException("COEsOatesJoshua:invalidInput","R and V must be either
 1x3 vectors or scalars"))
end

% uearth = 3.986004418(8) x 10^14 m^3/s^2
ihat=[1,0,0];
khat=[0,0,1];

Rm = norm(R);
Vm = norm(V);

%calculate orbital constants
h = cross(R,V); %angular momentum vector
hm = norm(h);
E = ( ( Vm^2 ) / 2) - ( u / Rm ) ; %specfic mechanical energy
%calculate COEs

a = -u / (2 * E ); %semi major axis in km
T = 2*pi*sqrt((a^3)/u); %period in s

e = (1/u) * (((Vm^2)-(u/Rm) ) * R - (dot(R,V) * V)); %eccentricity vector
em = norm (e); %magnitude of e
```

```
inc = acos((dot(khat,h))/hm); %inclination
n = cross(khat,h); %node vector
nm = norm(n); %magnitude n

%raan
raan = acos(dot(ihat,n)/nm);
if n(2) < 0  %checks the vector relative to j to see if angle is positive or
 negative
    raan = 2*pi - raan;
end

%aop
aop = acos(dot(n,e)/(nm*em));
if e(3) < 0
    aop = 2*pi - aop;
end

%theta
theta = acos(dot(e,R)/(em*Rm));
if(dot(R,V) < 0) % cehck flight path angle to see if it is postive or negative
    theta = 2*pi -theta;
end

if strcmp(magOrVec, 'magnitude') %for magnitude mode, vectors will not be
 returned
    h = hm;
    e = em;
    if joshIsOnes([m1 n1 m2 n2]) % for scalar inputs it is not possible to
 calculate these values
        theta = NaN;
        inc = NaN;
        raan = NaN;
        aop = NaN;
    end
end
ecc = e;

end

function [r,v] = joshCOE2rv(a,ecc,theta,inc,raan,aop,mu)
% all angles in rads
[Cx,Cy,Cz]=joshAxisRotation(); % just rotation matrix about x and z of theta

rp = a*(1-ecc^2)/(1+ecc); % cos(0) = 1
h = sqrt(mu*rp*(1+ecc*cos(theta)));

r = (h^2/mu)/(1+ecc*cos(theta));
rperi = [cos(theta);sin(theta);0]*r; % r vector in perifocal
[vaz,vr] = joshVazVr(theta,ecc,h,mu); % v vector in local horizontal vertical
vloc = [vr;vaz;0];

vperi = Cz(-theta)*vloc; % v vector rotated to perifocal
```

```
Q = Cz(raan)*Cx(inc)*Cz(aop);
Q = Q'; % Perifocal -> ECI

r = Q*rperi; % rotation to get to Inertial frame
v = Q*vperi;

r = r';
v = v';
end

function [dv1,dv2,dv,T,ht,ecct,vt1,vt2] = joshHomann(r1,v1,r2,v2,mu)
% takes the magnitudes of r1 v1 at either apoapse or periapse of orbit 1
% and r2 v2 at the apoapse or periapse of orbit 2 as scalars.
% it is assumed that the apses of orbits 1 and 2 are on oposite sides of
% of the foci and that they lie on the same apse line.
% it is assumed that both orbits have the same grade, ie both pro- or retro-
% grade. parameters should be positive values but the returned dv's
% may be negative to corresponding to the retrograde burn.
% The first 3 returned values correspond to delta V's
% the 4th returned value corresponds to transfer time ie 1/2 of period
% The 5th-8th returned values correspond to properties of the transfer orbit

arguments
    r1(1,1) double {mustBeNonnegative}
    v1(1,1) double {mustBeNonnegative}
    r2(1,1) double {mustBeNonnegative}
    v2(1,1) double {mustBeNonnegative}
    mu (1,1) {mustBeNumeric, mustBeReal, mustBePositive} = 3.986004418 *
 (10^5) %km^3/s^2
end
    ecct = ((r2-r1)/(r1+r2)); % absolute value so that if r2 > r1, ecct is
 postive
    ht = sqrt(mu*r1*(1+ecct)); % this assumes we're at periapse
    vt1 = ht/r1;
    vt2 = ht/r2;
    dv1 = vt1-v1;
    dv2 = v2-vt2;
    dv = abs(dv1)+abs(dv2);
    at = (r1+r2)/2;
    T = at^1.5*pi/sqrt(mu);
end

function [v1,v2] = joshIntegratedLamberts(r1,r2,dt,grade,mu)
% this function will call joshfLamberts but requires less thinking to use.
% the downside is that for situations where r and v vectors aren't known,
% This function cannot be used and because this function will generate
% stumpf functions for you (which is convient and i shouldve done a long time
 ago)
% it is computationally less effcient.

arguments
    r1 (3,1) double {mustBeReal, mustBeNonNan}
    r2 (3,1) double {mustBeReal, mustBeNonNan}
%     magOrVec {mustBeMember(magOrVec,{'magnitude','vector'})} = 'magnitude'
```

```
    dt (1,1) double {mustBePositive}
    grade string {mustBeMember(grade,{'pro','retro'})} = 'pro'
    mu (1,1) double {mustBePositive} = 398600
end



% ready to run lamberts from obj2 to obj3
theta = acos(dot(r1,r2)/(norm(r1)*norm(r2)));
if strcmp(grade,'retro')
    theta = 2*pi - theta;
end
% set up stumpffys
% coefs = 15;
% [Cc,Sc]=joshStumpffCoeffs(coefs);
% C = @(z) sum(Cc.*joshStumpffZ(z,coefs));
% S = @(z) sum(Sc.*joshStumpffZ(z,coefs));


[fz,y,A,z,flag,glag,gdotlag] =
 joshfLambert(norm(r1),norm(r2),dt,theta,mu);%,Cc,Sc);
% [v1rb, v2rb] =lambertsRB(r1,r2,dt,1);

v1 = (1/glag)*(r2-flag*r1);
v2 = (1/glag)*(gdotlag*r2-r1);


end

function [fz,y,A,z,flag,glag,gdotlag] = joshfLambert(r1,r2,dt,theta,mu,C,S)
%{
this function will return the lagrange coeffs along with z from the
universal variable, A and yfrom lamberts problem and fz from lamberts
problem such that the zero of fz will give you the solution z to lamberts
problem. r1 r2 should be scalars, dt is transit tim
%}


arguments
    r1 (1,1) double {mustBeReal, mustBePositive}
    r2 (1,1) double {mustBeReal, mustBePositive}
    dt (1,1) double {mustBePositive}
    theta (1,1) double {mustBePositive}
    mu (1,1) double {mustBePositive} = 398600
    C (1,:) double {mustBeReal} = nan
    S (1,:) double {mustBeReal} = nan
    %     z0 (1,1) double {mustBeReal} = nan
end


% warning("joshfLambert: This function may be useful but it is not well tested
 and complete argument validation has not been implimented.")
```

```
if isnan(C)|isnan(S)
    [C,S] = joshStumpffCoeffs();
end

coefs = length(C);
if length(S)~=coefs
    throw(MException("joshfLambert:invalidInput","S and C should be the same
 length"))
end

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

C = @(z) sum(C.*joshStumpffZ(z,coefs));
S = @(z) sum(S.*joshStumpffZ(z,coefs));

y = @(z)(r1+r2+  A*((z*S(z)-1))/sqrt(C(z))); % y is correct
fz = @(z) S(z)*(y(z)/C(z))^(1.5)+  A*sqrt(y(z))-sqrt(mu)*dt; % f is correct

z0 = JoshBisection(fz,[-10,100]);
z = fzero(fz,z0);

flag = 1-(y(z)/norm(r1));
glag = A*sqrt(y(z)/mu);
% fdot = (sqrt(mu_e)/(norm(r1)*norm(r2)))*sqrt(y(z)/C(z))*(z*S(z)-1);
gdotlag = (1-(y(z)/norm(r2)));


end

function [isOnes] = joshIsOnes(M)
% takes a value (persumably a logical type matrix) and returns true iff all
% entries are true
[m,n] = size(M);
isOnes = true;
for i = 1:m
    for j = 1:n
        if M(i,j) ~= 1
            isOnes = false;
        end
    end
end
end

function [jd,thetaTime ,j2000, j0, ut, thetaG, jcent2000] =
 joshJulian(t,thetaLongitude)
% takes t as a datetime object in UT and a longitude
% jd - juliandate day
% thetaTime - local sidereal time in degrees
% j2000 - julian date from 2000 (jd-j2000_0)
% j0 - julian days
% ut - UT in hours
% thetaG - Grennich sidereal time
% number of julian centuries from j2000
arguments
```

```
    t (1,1) datetime
    thetaLongitude (1,1) {mustBeReal} = 0
end


j2000_0 = 2451545;

[yr,mo,da] = ymd(t);
[hr,mn,sc] = hms(t);

j0 = 367*yr-floor((7*(yr+floor((mo+9)/12)))/4)+floor((275*mo)/9)+da+1721013.5;
ut = hr + mn/60 + sc/3600;
jd = j0 + (ut/24);
j2000 = jd - j2000_0;

t0 = (j0 - j2000_0)/36525;

thetaG0 = 100.4606184 + 36000.77004*t0 + 0.000387933*t0^2 -
 2.583*(10e-8)*t0^3;
thetaG = thetaG0 + 360.98564724* (ut/24);
thetaTime = thetaG + thetaLongitude;

thetaG = mod(thetaG,360);
thetaTime = mod(thetaTime,360);

jcent2000 = j2000/36525;

end

function [r,v,t,X] = joshOrbitCoastOde(r0,v0,dt,mu)
% a simple ode propegator for two body orbits
arguments
    r0 (3,1) double {mustBeReal,mustBeNonNan}
    v0 (3,1) double {mustBeReal,mustBeNonNan}
    dt (1,1) double {mustBeReal,mustBeNonNan}
    mu (1,1) double {mustBeReal,mustBeNonNan}= 398600;
end
X0 = [r0;v0];
options = odeset('RelTol', 1e-8,'AbsTol',1e-13);
[t,X] = ode45(@orbitODEFun1,[0,dt],X0,options,mu);
Xe = X(end,:);
r = Xe(1:3)';
v = Xe(4:6)';
end

function Xdot = orbitODEFun1(t,X,mu)
r = X(1:3);
v = X(4:6);
vdot = (-mu/norm(r)^3)*r;
rdot = v;
Xdot = [rdot;vdot];
end

function [quadrantInt,theta2,quadrantDec] = joshQuadrant(theta1)
```

```
% values in rads
% for any theta1 it will return a corresponding positive theta2 as well as
% the quadrant it is in and how many quadrants it is into the unit circle.
% ie, floor of quadrantDec is the quadrant its in.
arguments
    theta1 (1,1) double {mustBeReal,mustBeNonNan}
end
s = sign(theta1);
if s == 0
    theta2 = 0;
    quadrantInt = 0;
    quadrantDec = 0;
    return
end
    theta2 = mod(theta1,2*pi);
    quadrantDec = theta2/(pi/2)+1;
if mod(quadrantDec,1) == 0
    quadrantInt = 0;
else
    quadrantInt = floor(quadrantDec);
end

end

function [C,S] = joshStumpffCoeffs(n)
% AERO 351 code
% Generates the first n terms of the stumpff coeffcients for @S(z) and @C(z)
 in a vector

% these coeffs are used for the universal variable approach to orbital
 mechanics

% for use as companion function with joshStrumpffZ
% coeffs should be saved to workspace and reused to save compute time
% @S(z) == sum(S.*Z) == polyval(flip(S),z) : where Z = [z^0 z^1 ... z^n]
% @C(z) == sum(C.*Z) == polyval(flip(C),z) : where Z = [z^0 z^1 ... z^n]
arguments
    n (1,1) {mustBePositive,mustBeInteger} = 15;
end

C = zeros(1,n);
S = C;
for i = 1:n
    k = i-1;
    C(i) = (-1)^k*(1/factorial(2*k+2));
    S(i) = (-1)^k*(1/factorial(2*k+3));
end
end

function [Z] = joshStumpffZ(z,n)
% AERO 351 code
% Generates the first n terms of the stumpff coeffcients for @S(z) and @C(z)
 in a vector
% for use as companion function with joshStrumpffCoeffs
```

```
% @S(z) == sum(S.*Z) == polyval(flip(S),z) : where S given by (-1)^k*(1/
factorial(2*k+2))
% @C(z) == sum(C.*Z) == polyval(flip(C),z) : where C given by (-1)^k*(1/
factorial(2*k+3))
arguments
    z (1,1)
    n (1,1) {mustBePositive,mustBeInteger} = 15;
end
Z = ones(1,n);
for i = 2:n
    Z(i) = z*Z(i-1);
end
end

function [vaz,vr,gamma] = joshVazVr(theta,ecc,h,mu)
% gives magnitude of azmuthal velocity and radial velocity
% takes theta ecc and h
% optionally takes mu for the center body
% assumes shperical body and 2 body
% angles in rad
arguments
    theta (1,1) double {mustBeReal,mustBeNonNan}
    ecc (1,1) double {mustBeReal, mustBeNonnegative}
    h (1,1) double {mustBeReal,mustBePositive}
    mu (1,1) double {mustBeReal,mustBeNonNan} = 3.986004418 * (10^5) %km^3/s^2
 mu_earth
end
vr  = (mu/h)*ecc*sin(theta);
vaz = (mu/h)*(1+ecc*cos(theta));
gamma = atan2(vr,vaz);
end

function [theta,M,E] = joshAnomalyCalculator(ecc,anomaly,input)
% M will be Me,Mp or Mh depending on ecc
% E will be Eccentric Anomoly when applicable or F: hyperbolic Ecctric
% anomoly. E will be set to
% values in Rads
arguments
    ecc (1,1) double {mustBeReal,mustBeNonnegative,mustBeNonNan}
    anomaly (1,1) double {mustBeReal,mustBeNonNan}
    input {mustBeMember(input,{'theta','Me'})} = 'theta'
end

if strcmp(input,'theta')
    theta = anomaly;
    if ecc <1 % Me & E
        E = 2*atan(sqrt((1-ecc)/(1+ecc))*tan(theta/2)); % definintion of E,
 rewriten to solve E
        M = E-ecc*sin(E); % definition of M
    elseif ecc > 1 % Mh & F
        E = log((sqrt(ecc+1)+sqrt(ecc-1)*tan(theta/2))/(sqrt(ecc+1)-
sqrt(ecc-1)*tan(theta/2)));
        M = ecc(sinh(F)-F);
    else % ecc == 1 Mp
```

```
        E = nan; % This is a rare case and E doesnt have a definition for ecc
 == 1
        M = .5*tan(theta/2)+(1/6)*tan(theta/2)^3;
    end
elseif strcmp(input,'Me')
    if ecc>=1
        throw(MException("joshAnomalyCalculator:inputNotSupported","entering
 Me for non eliptical orbits is not supported"))
    end

    M = anomaly;
    if M<pi % initial E
        E0 = M+ecc/2;
    else
        E0 = M-ecc/2;
    end
    f = @(E) (E-ecc*sin(E)-M);
    E = fzero(f,E0);
    theta = 2*atan( tan(E/2)/sqrt((1-ecc)/(1+ecc)) );
end


end
```

Published with MATLAB® R2022a