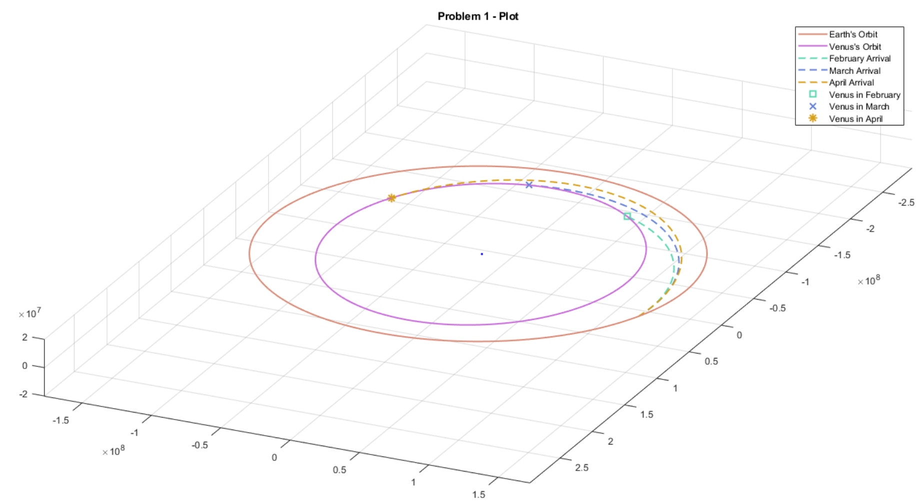
Table of Contents

Setting all the universal stuff

Problem 1

```
*********************************************************************************
Problem 1
     A February arrival would cost 9.3313 [km/s]

     A March arrival would cost 6.5224 [km/s]

     A April arrival would cost 6.6416 [km/s]

My recommendation would be to arrive on March 1, 2006 in order to
 minimize delta-V requirments

------------------

This is reasonable because Lambert's solution is fairly fine tuned
 when it comes to delta-V because
 because the synodic period of the orbit's will dictate a phase angle
 at arrival that has optimal
 delta-V and unless the we are arriving in that window the delta-V
 will shoot up.
```



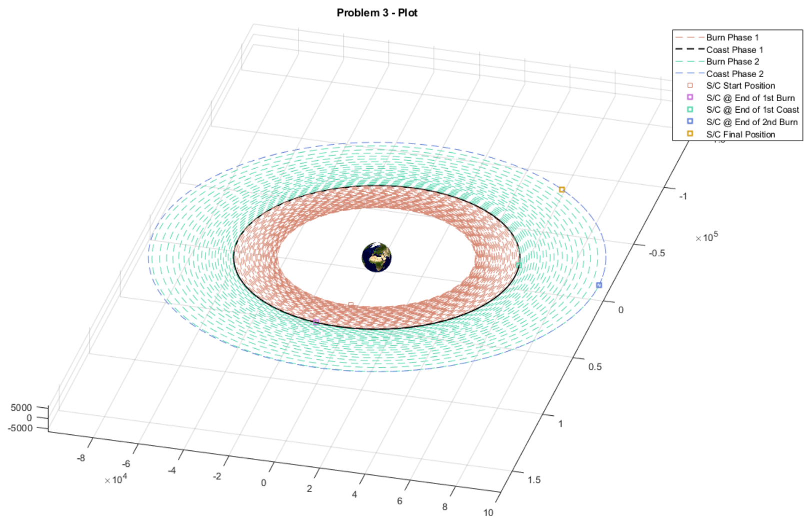
Problem 2

```
*********************************************************************************
Problem 2
      The final heliocentric speed of the spacecraft is:

28.5934 km/s

     The total imparted delta-V is 2.1538 km/s

------------------

 The delta-V imparted by the sun increases the S/C's heliocentric
 speed
 because it is doing a trailing edge, sunlit side maneuver, where the
 planet is coming up
 from behind the S/C and basically slingshotting it forward, giving
 the S/C
 some of the angular momentum of Earth
```

Problem 3

```
*********************************************************************************
Problem 3

     The final mass of the spacecraft is 721.4303 kg

     The final radius of the spacecraft is 97917.8146 km

------------------

 This graph has 2 distinct types of trajectory that one would expect,
```

first, is the spiraling behavior which you see in the burn phases,
and next,
the coast phases which only appear as circles since the orbit is
constant at those points



Problem 4

****
Problem 4

The S/C will need to burn 3 times to escape Earth's orbit.

It will take 9.011 hours to achieve escape velocity from the
first burn.

This only taking 3 burns makes sense because, from a given circular
orbit,
the velocity only needs to increase by 41.4 percent and the dV added
each time is about
14 percent of the initial velocity, so after 3 burns it will be just
enough to
get it into interplanetary space

Problem 5

****

Problem 5

Part B

The ecc, inc, RAAN, AoP, and TA from the state vector are:

[0.0004356000000002 51.6444 235.4504 265.446000000031 30.7086574835437]
Which are exactly what is given in the TLE

Part C

The final position from ODE45 is:

6797.0252 km

The final speed from ODE45 is:

7.6588 km

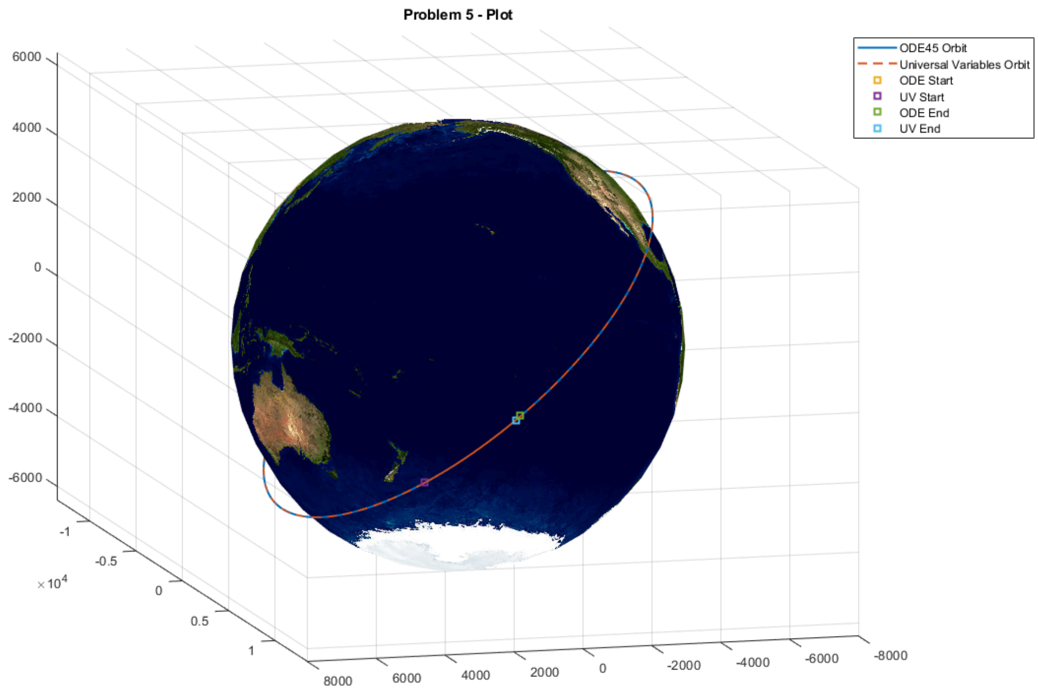
The final radius from Universal Variables is:

6797.0252 km

The final speed from Universal Variables is:

7.6588 km

These orbits are accurate down the meter, the slight discrepancy in the trailing terms
leads to the difference in plotting, so, though the final positions look slightly off,
on the plot, I can confidently say that they are the same point. The same is true for velocity.
The discrepancy comes from the fact that each method has different rounding errors and such
that add up with each step.



Published with MATLAB® R2021a

Table of Contents


```
% AERO - 351: Final Exam

clear
close all
clc
```

Setting all the universal stuff

```
mu_sun = 1.3271e11;

mu_earth = 3.986e5;

mu_venus = 324859;

r_Earth = 6378; % km

r_Venus = 6052; % km

d2sec = 24*3600;

AU = 149598000; % 1 AU in km

terminator
 = '******************************************************************************
\n';

cutoff = '------------------\n';

colors = ["#DB8E7B", "#C870DB", "#5ADBB1", "#6586DB", "#DBA41A"];

options = odeset('RelTol',1e-8,'AbsTol',1e-8);

[X,Y,Z] = sphere;

sunX = 696340.*X;
sunY = 696340.*Y;
sunZ = 696340.*Z;

[X,Y,Z] = sphere;

eX = 6378.*X;
```

```
eY = 6378.*Y;
eZ = 6378.*Z;

earth = imread('landOcean.jpg');

clear X Y Z
```

Problem 1

```
%

% We are leaving on December 1, 2005 (assuming 00:00:00 UTC)

depDate = myJulianDate(2005,12,1,0,0,0);

% and Arriving on either Feb., March, or April 1st, 2006

arrDate1 = myJulianDate(2006,2,1,0,0,0);

arrDate2 = myJulianDate(2006,3,1,0,0,0);

arrDate3 = myJulianDate(2006,4,1,0,0,0);

% Now we need to know where the planets will be at all of those dates

[earthPos, earthVel] = planetStates(depDate,3);

[venusPosFeb,venusVelFeb] = planetStates(arrDate1,2);

[venusPosMar,venusVelMar] = planetStates(arrDate2,2);

[venusPosApr,venusVelApr] = planetStates(arrDate3,2);

% Knowing these we can calculate the lambert's solution for all

deltaTfeb = (arrDate1 - depDate) * d2sec;
deltaTmar = (arrDate2 - depDate) * d2sec;
deltaTapr = (arrDate3 - depDate)  * d2sec;

tm = 1;

tol = 1e-8;

ztol = 8;

[vDepFeb,vArrFeb,v_depFeb,v_arrFeb] =
 lambertsGeneral(earthPos,venusPosFeb,deltaTfeb,tm,tol,ztol,mu_sun);

[vDepMar,vArrMar,v_depMar,v_arrMar] =
 lambertsGeneral(earthPos,venusPosMar,deltaTmar,tm,tol,ztol,mu_sun);

[vDepApr,vArrApr,v_depApr,v_arrApr] =
 lambertsGeneral(earthPos,venusPosApr,deltaTapr,tm,tol,ztol,mu_sun);
```

```
vInf_dep1 = norm(earthVel - v_depFeb);
vInf_dep2 = norm(earthVel - v_depMar);
vInf_dep3 = norm(earthVel - v_depApr);

vInf_arr1 = norm(v_arrFeb - venusVelFeb);
vInf_arr2 = norm(v_arrMar - venusVelMar);
vInf_arr3 = norm(v_arrApr - venusVelApr);

% Now that we have the transfer we have to look at the initial and
 final
% orbits around Venus and Earth

    % Terrestrial Parking Orbit - 300km circular orbit

    z_park = 300; % km

    r_park = z_park + r_Earth;

    v_park = sqrt(mu_earth/r_park);

    % Venusian Capture Orbit - 200 km x 10000 km elliptical orbit

    z_peri = 200; % km

    z_apo = 10000;

    r_peri = z_peri + r_Venus;

    r_apo = z_apo + r_Venus;

    eccCapt = (r_apo - r_peri)/(r_apo + r_peri);

    h_capture = sqrt(mu_venus * r_peri  * (1 + eccCapt));

    v_capture = (mu_venus/h_capture)*(1 + eccCapt);

% And to find the delta-V for each

depV1 = sqrt(vInf_dep1^2 + ((2*mu_earth)/r_park));
depV2 = sqrt(vInf_dep2^2 + ((2*mu_earth)/r_park));
depV3 = sqrt(vInf_dep3^2 + ((2*mu_earth)/r_park));

arrV1 = sqrt(vInf_arr1^2 + ((2*mu_venus)/r_peri));
arrV2 = sqrt(vInf_arr2^2 + ((2*mu_venus)/r_peri));
arrV3 = sqrt(vInf_arr3^2 + ((2*mu_venus)/r_peri));


deltaVDep1 = depV1 - v_park;
deltaVDep2 = depV2 - v_park;
deltaVDep3 = depV3 - v_park;

deltaVArr1 = v_capture - arrV1;
deltaVArr2 = v_capture - arrV2;
```

```
deltaVArr3 = v_capture - arrV3;

totalDVfeb = abs(deltaVDep1) + abs(deltaVArr1);
totalDVmar = abs(deltaVDep2) + abs(deltaVArr2);
totalDVapr = abs(deltaVDep3) + abs(deltaVArr3);

fprintf(terminator)
fprintf("Problem 1")
fprintf(" \n")
fprintf("     A February arrival would cost " + num2str(totalDVfeb)
 + " [km/s]\n")
fprintf(" \n")
fprintf("     A March arrival would cost " + num2str(totalDVmar)
 + " [km/s]\n")
fprintf(" \n")
fprintf("     A April arrival would cost " + num2str(totalDVapr)
 + " [km/s]\n")
fprintf(" \n")
fprintf("My recommendation would be to arrive on March 1, 2006 in
 order to minimize delta-V requirments\n")
fprintf(" \n")
fprintf(cutoff)
fprintf(" \n")
fprintf("This is reasonable because Lambert's solution is fairly fine
 tuned when it comes to delta-V because\n because the synodic period
 of the orbit's will dictate a phase angle at arrival that has optimal
 \n delta-V and unless the we are arriving in that window the delta-V
 \n will shoot up.\n")


% Let's plot everything

tspanE = [0 365*34*3600];
tspanV = [0 243*34*3600];

stateEarth = [earthPos, earthVel];
stateVenus = [venusPosMar,venusVelMar];


[~, earthFinal] = ode45(@fun, tspanE, stateEarth, options, mu_sun);
[~, venusFinal] = ode45(@fun, tspanV, stateVenus, options, mu_sun);

tTrans1 = [0 deltaTfeb];
tTrans2 = [0 deltaTmar];
tTrans3 = [0 deltaTapr];

state1trans = [earthPos v_depFeb];
state2trans = [earthPos v_depMar];
state3trans = [earthPos v_depApr];

[~, transfer1] = ode45(@fun, tTrans1, state1trans, options, mu_sun);
[~, transfer2] = ode45(@fun, tTrans2, state2trans, options, mu_sun);
[~, transfer3] = ode45(@fun, tTrans3, state3trans, options, mu_sun);
```

```
f1 = figure(1);
f1.Position = [0 0 1920 1080];
hold on
plot3(earthFinal(:,1),earthFinal(:,2),earthFinal(:,3),'Color',colors(1),'LineWidth
plot3(venusFinal(:,1),venusFinal(:,2),venusFinal(:,3),'Color',colors(2),'LineWidth
plot3(transfer1(:,1),transfer1(:,2),transfer1(:,3),'--','Color',colors(3),'LineWid
plot3(transfer2(:,1),transfer2(:,2),transfer2(:,3),'--','Color',colors(4),'LineWid
plot3(transfer3(:,1),transfer3(:,2),transfer3(:,3),'--','Color',colors(5),'LineWid
plot3(venusPosFeb(1),venusPosFeb(2),venusPosFeb(3),'sq','Color',colors(3),'MarkerS
plot3(venusPosMar(1),venusPosMar(2),venusPosMar(3),'x','Color',colors(4),'MarkerSi
plot3(venusPosApr(1),venusPosApr(2),venusPosApr(3),'*','Color',colors(5),'MarkerSi
surf(sunX,sunY,sunZ,'FaceColor','b','EdgeColor','none')
zlim([-20e6 20e6])
legend("Earth's Orbit", "Venus's Orbit", "February Arrival", "March
 Arrival", "April Arrival", "Venus in February", "Venus in
 March","Venus in April")
grid on
axis equal
view([115,22.5])
title("Problem 1 - Plot")


%}
```

Problem 2

```
%

% We know the radius and period of the S/C's initial orbit

T_helio = .75 * 365 * 24 * 3600;

r_apoH = 1*AU;

% From which we can determine its orbital characteristics, assuming
 it's
% coplanar with the orbit of earth

aHelio = ((T_helio*sqrt(mu_sun))/(2*pi))^(2/3);

r_periH = 2*aHelio - r_apoH;

% Need to use Quadratic formula to find eccentricity knowing a, r, and
 the
% fact the object is at apogee

ecc_Helio = (r_apoH - r_periH)/(r_apoH + r_periH);

h_Helio = sqrt(mu_sun * r_apoH * (1 - ecc_Helio));
```

```
% We can then figure out the heliocentric velocity at apogee when it
% encounters the Earth's SOI

vInit_helio = (h_Helio/r_apoH);


vHelio_vec = [vInit_helio 0];


% And we'll assume the Earth is in a circular orbit for simplicity's
 sake

vEarth = sqrt(mu_sun/AU);

vEarth_vec = [vEarth 0];

% Now the hyperbolic excess speed will be the difference in
 Heliocentric
% velocities

% The hyperbolic excess velocity

hypExcess = vInit_helio - vEarth;

phi1 = 180; % at Apogee going against the direction of earth

VvecInf_in = [hypExcess*cosd(phi1) hypExcess*sind(phi1)];

% Now we need to find the eccentricity of the hyperbolic flyby orbit

r_pHyp = 500 + r_Earth;

ecc_Hyp = 1 + (hypExcess^2 * r_pHyp)/mu_earth;

TurnAngle = 2*asind(1/ecc_Hyp);

phi2 = phi1 - TurnAngle;

VvecInf_out = [hypExcess*cosd(phi2) hypExcess*sind(phi2)];


% Now that we have vectors of each we can calculate the imparted
 deltaV

prob2_dVvect = VvecInf_out - VvecInf_in;

prob2_deltaV = norm(prob2_dVvect);

Vfinal_Helio = vEarth_vec + VvecInf_out;

final_HelioV = norm(Vfinal_Helio);

dvTest = final_HelioV - vInit_helio;
```

```
fprintf(terminator)
fprintf("Problem 2\n")
fprintf(" ")
fprintf("     The final heliocentric speed of the spacecraft is:\n")
fprintf(" \n")
fprintf(final_HelioV + " km/s\n")
fprintf(" \n")
fprintf("     The total imparted delta-V is " + num2str(dvTest) + "
 km/s\n")
fprintf(" \n")
fprintf(cutoff)
fprintf("\n The delta-V imparted by the sun increases the S/C's
 heliocentric speed \n because it is doing a trailing edge, sunlit
 side maneuver, where the planet is coming up \n from behind the S/C
 and basically slingshotting it forward, giving the S/C \n some of the
 angular momentum of Earth \n")


%}
```

Problem 3

```
%

% Givens:

m_0 = 747; % [Mg]

Isp = 3100; % [s]

Th = 90e-6; % [kN]

R_0 = [42000 0 0]; % [km]

V_0 =  [0 3.0807 0]; % [km/s]

y_0 = [R_0'; V_0'; m_0];

burnTime = [50 54 104 108] .* d2sec;

thrust = [Th 0 Th 0];

prob3Function = @(t,Y) Y_bar(t,Y,Isp,mu_earth,thrust(1));

tspan = [0 burnTime(1)];

[~, trajectory1_prob3] = ode45(prob3Function,tspan,y_0,options);

tspan = [burnTime(1) burnTime(2)];
y_1 = trajectory1_prob3(end,:);
prob3Function = @(t,Y) Y_bar(t,Y,Isp,mu_earth,thrust(2));
```

```
[~, trajectory2_prob3] = ode45(prob3Function,tspan,y_1,options);

tspan = [burnTime(2) burnTime(3)];
y_2 = trajectory2_prob3(end,:);
prob3Function = @(t,Y) Y_bar(t,Y,Isp,mu_earth,thrust(3));


[~, trajectory3_prob3] = ode45(prob3Function,tspan,y_2,options);

tspan = [burnTime(3) burnTime(4)];
y_3 = trajectory3_prob3(end,:);
prob3Function = @(t,Y) Y_bar(t,Y,Isp,mu_earth,thrust(4));


[~, trajectory4_prob3] = ode45(prob3Function,tspan,y_3,options);

fprintf(terminator)
fprintf("Problem 3\n")
fprintf(" \n")
fprintf("     The final mass of the spacecraft is " +
 num2str(trajectory4_prob3(end,7)) + " kg\n")
fprintf(" \n")
fprintf("     The final radius of the spacecraft is " +
 num2str(norm(trajectory4_prob3(end,1:3))) + " km\n")
fprintf(" \n")
fprintf(cutoff)
fprintf("\n This graph has 2 distinct types of trajectory that one
 would expect, \n first, is the spiraling behavior which you see in
 the burn phases, and next,\n the coast phases which only appear as
 circles since the orbit is constant at those points\n")

f3 = figure(3);
f3.Position = [0 0 1920 1080];
hold on
plot3(trajectory1_prob3(:,1),trajectory1_prob3(:,2),trajectory1_prob3(:,3),'--','C
plot3(trajectory2_prob3(:,1),trajectory2_prob3(:,2),trajectory2_prob3(:,3),'--','C
plot3(trajectory3_prob3(:,1),trajectory3_prob3(:,2),trajectory3_prob3(:,3),'--','C
plot3(trajectory4_prob3(:,1),trajectory4_prob3(:,2),trajectory4_prob3(:,3),'--','C
plot3(trajectory1_prob3(1,1),trajectory1_prob3(1,2),trajectory1_prob3(1,3),'sq','C
plot3(trajectory1_prob3(end,1),trajectory1_prob3(end,2),trajectory1_prob3(end,3),'
plot3(trajectory2_prob3(end,1),trajectory2_prob3(end,2),trajectory2_prob3(end,3),'
plot3(trajectory3_prob3(end,1),trajectory3_prob3(end,2),trajectory3_prob3(end,3),'
plot3(trajectory4_prob3(end,1),trajectory4_prob3(end,2),trajectory4_prob3(end,3),'
sEarth = surf(eX, eY, flip(eZ));
sEarth.FaceColor = 'texturemap';
sEarth.EdgeColor = 'none';
sEarth.CData = earth;
legend('Burn Phase 1', 'Coast Phase 1', 'Burn Phase 2','Coast Phase
 2', 'S/C Start Position', 'S/C @ End of 1st Burn', 'S/C @ End of 1st
 Coast', 'S/C @ End of 2nd Burn','S/C Final Position')
axis equal
grid on
view([105,30])
title("Problem 3 - Plot")
```

```
%}
```

Problem 4

```
%
% We start in a circular orbit around earth with velocity

z_circ = 200; % km

r_circ = z_circ + r_Earth;

velCirc = sqrt(mu_earth/r_circ);

% To escape from this orbit, we need at a minimum...

v_escape = sqrt(2)*velCirc;

% Burning at Perigee makes

dvBurn = 1.075; % km/s

vWhile = velCirc;

eccWh = 0;

aWh = r_circ;

Twh = ((2*pi)/sqrt(mu_earth))*(aWh)^(3/2);

i = 1;

while vWhile < v_escape

    vWhile(i + 1) = vWhile(i) + dvBurn;

    aWh(i + 1) = -(mu_earth/2)*(1/((vWhile(i + 1)^2)/2 - mu_earth/
r_circ));

    if aWh(i + 1) > 0
    Twh(i + 1) = ((2*pi)/sqrt(mu_earth))*(aWh(i + 1))^(3/2);
    else
    Twh(i + 1) = 0;
    end

    i = i + 1;

end

totalT = sum(Twh(2:4))/3600;

fprintf(terminator)
```

```
fprintf("Problem 4\n")
fprintf(" \n")
fprintf("     The S/C will need to burn " + num2str(i - 1) + " times
 to escape Earth's orbit.\n")
fprintf(" \n")
fprintf("     It will take " + num2str(totalT) + " hours to achieve
 escape velocity from the first burn.\n")
fprintf(" \n")
fprintf(cutoff)
fprintf("\nThis only taking 3 burns makes sense because, from a given
 circular orbit, \n the velocity only needs to increase by 41.4
 percent and the dV added each time is about \n 14 percent of the
 initial velocity, so after 3 burns it will be just enough to \n get
 it into interplanetary space \n")

%}
```

Problem 5

```
%

prob5_TLE = [51.6444 235.4504 .0004356 265.4460 266.3380
 15.487215013];

[R5,V5] = TLEtoStateVector(prob5_TLE);

[a5,ecc5,incl5,RAAN5,AoP5,nu5,hn5] =
 COEs_bornhorstMatthew(R5,V5,mu_earth);

coes5 = [a5,ecc5,incl5,RAAN5,AoP5,nu5,hn5];

stateVector = [R5,V5];

deltaT = 100*60; % 100 mins in secs

tspan5 = [0 deltaT]; %

[t5, odeProb5] = ode45(@fun,tspan5, stateVector, options, mu_earth);

tol = 1e-12;

ztol = 8;

dtVect = linspace(0,deltaT,length(t5) + 1);

Rs = zeros(length(dtVect),3);

Vs = zeros(length(dtVect),3);

Rs(1,:) = R5;

Vs(1,:) = V5;
```

```
for i = 1:length(dtVect)-1

   dt =  dtVect(i + 1) - dtVect(i);

   [Rs(i + 1,:),Vs(i + 1,:)] =
 RVwithUV(Rs(i,:),Vs(i,:),dt,tol,ztol,mu_earth);

end

finalPosODE = norm(odeProb5(end,1:3));
finalVelODE = norm(odeProb5(end,4:6));

finalPosUV = norm(Rs(end,:));
finalVelUV = norm(Vs(end,:));

fprintf(terminator)
fprintf(" \n")
fprintf("Problem 5\n")
fprintf(" \n")
fprintf("   Part B\n")
fprintf(" \n")
fprintf("The ecc, inc, RAAN, AoP, and TA from the state vector are:
 \n")
fprintf(" \n")
fprintf(mat2str(coes5(2:6)))
fprintf(" \n")
fprintf("Which are exactly what is given in the TLE \n")
fprintf(" \n")
fprintf("   Part C\n")
fprintf(" \n")
fprintf("     The final position from ODE45 is: \n")
fprintf(" \n")
fprintf(num2str(finalPosODE) + " km\n")
fprintf(" \n")
fprintf("     The final speed from ODE45 is: \n")
fprintf(" \n")
fprintf(num2str(finalVelODE) + " km\n")
fprintf(" \n")
fprintf("     The final radius from Universal Variables is: \n")
fprintf(" \n")
fprintf(num2str(finalPosUV) + " km\n")
fprintf(" \n")
fprintf("     The final speed from Universal Variables is: \n")
fprintf(" \n")
fprintf(num2str(finalVelUV) + " km\n")
fprintf(" \n")
fprintf("   These orbits are accurate down the meter, the slight
 discrepancy in the trailing terms\n")
fprintf("leads to the difference in plotting, so, though the final
 positions look slightly off,\n")
fprintf("on the plot, I can confidently say that they are the same
 point. The same is true for velocity.\n")
fprintf("The discrepancy comes from the fact that each method has
 different rounding errors and such\n")
```

```
fprintf("that add up with each step.\n")

f5 = figure(5);
f5.Position = [0 0 1920 1080];
hold on
plot3(odeProb5(:,1),odeProb5(:,2),odeProb5(:,3),'-','LineWidth',1.5)
plot3(Rs(:,1),Rs(:,2),Rs(:,3),'--','LineWidth',1.5)
plot3(odeProb5(1,1),odeProb5(1,2),odeProb5(1,3),'sq','LineWidth',1.5)
plot3(Rs(1,1),Rs(1,2),Rs(1,3),'sq','LineWidth',1.5)
plot3(odeProb5(end,1),odeProb5(end,3),odeProb5(end,2),'sq','LineWidth',1.5)
plot3(Rs(end,1),Rs(end,2),Rs(end,3),'sq','LineWidth',1.5)
sEarth = surf(eX, eY, flip(eZ));
sEarth.FaceColor = 'texturemap';
sEarth.EdgeColor = 'none';
sEarth.CData = earth;
legend('ODE45 Orbit','Universal Variables Orbit','ODE Start','UV
 Start','ODE End','UV End')
grid on
axis equal
view([285,-10])
title("Problem 5 - Plot")



%}

% Functions

% The ODE equation for general stuff
function dstate = fun(t, state, mu_earth)

% Define state functions

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

r = norm([x y z]);

% equations of motion
ddx = -mu_earth*x/r^3;
ddy = -mu_earth*y/r^3;
ddz = -mu_earth*z/r^3;
dstate = [dx; dy; dz; ddx; ddy; ddz];
end

% ODE Equation with thrust

function ydot = Y_bar(t,Y,Isp,mu,thrust)

T = thrust;
```

```
g0 = 9.81/1000; % Gravity in km/s^2

rvec = [Y(1) Y(2) Y(3)];

vvec = [Y(4) Y(5) Y(6)];

r = norm(rvec);

v = norm(vvec);

dx = Y(4);

dy = Y(5);

dz =Y(6);

ddx = - mu*(Y(1)/r^3) + (T*Y(4))/(Y(7)*v);

ddy = - mu*(Y(2)/r^3) + (T*Y(5))/(Y(7)*v);

ddz = - mu*(Y(3)/r^3) + (T*Y(6))/(Y(7)*v);

mdot = -1*(T/(Isp*g0));


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


end

{
% Universal Variables

function [Chi_f] = universalVariables(r_0,vr_0,a,dt,tol,ztol,mu)
%Calculates the universal anomaly at t for a given orbit

alp = 1/a;

Chi(1) = sqrt(mu)*abs(alp)*dt;

ratio = 1;

i = 1;

tolVect = [1:ztol];
fotC = factorial(2*tolVect+2);
fotS = factorial(2*tolVect+3);


a1 = ((r_0*vr_0)/sqrt(mu));

a2 = (1-alp*r_0);
```

```
a3 = sqrt(mu)*dt;

while abs(ratio) > tol || i<100

    z = alp*Chi(i)^2;

    [C,S] = stumpffFunction(z,ztol,fotC,fotS);

    f = a1*(Chi(i)^2)*C + a2*(Chi(i)^3)*S + r_0*Chi(i) - a3;

    fprime = a1*Chi(i)*(1-z*S) + a2*(Chi(i)^2)*C + r_0;

    ratio = f/fprime;

    if abs(ratio) < tol
       Chi_f = Chi(i);

       return
    else

        Chi(i+1) = Chi(i) - ratio;

        i = i + 1;
    end



end

end

% RV from UV

function [R,V] = RVwithUV(R_0,V_0,dt,tol,ztol,mu)
% This function computes the new R and V vectors using universal
 variables

% 1a - Compute magnitudes of R and V
r_0 = norm(R_0);

v_0 = norm(V_0);

% 1b - Compute the radial component of velocity

vr_0 = dot(R_0,V_0)/r_0;

% 1c - compute the semi-major axis

a = ((2/r_0) - (v_0^2/mu))^-1;

% 2 - plug that info into the universal variables code to find Chi

[Chi] = universalVariables(r_0,vr_0,a,dt,tol,ztol,mu);
```

```
z_final = Chi^2/a;

% 3 - with chi, a, dt, and r_0 compute f and g

tolVect = [1:ztol];
fotC = factorial(2*tolVect+2);
fotS = factorial(2*tolVect+3);

[C,S] = stumpffFunction(z_final,ztol,fotC,fotS);

f = 1 - (Chi^2/r_0)*C;

g = dt - (1/sqrt(mu))* Chi^3 * S;

% 4 - From 3 we can compute R, and with the new r magnitude we can
 compute f_dot and g_dto

R = f.*R_0 + g.*V_0;

r = norm(R);

f_dot = (sqrt(mu)/(r*r_0))*((Chi^3 * S)/a - Chi);

g_dot = 1 - (Chi^2/r)*C;

% 5 - which we use to find the new V

V = f_dot.*R_0 + g_dot.*V_0;

end

% Stumpff Functions

function [C,S] = stumpffFunction(z,ztol,fotC,fotS)
%Gives solutions to both Stumpff Function for a given z

% Define the denominators for C and S
k = 1;

S = (1/6);

C = (1/2);


for k = 1:ztol

    C = C + ((-1)^k)*((z^k)/(fotC(k)));


    S = S + ((-1)^k)*((z^k)/(fotS(k)));

end
```

```
end

% TLE to State Vector

function [R,V] = TLEtoStateVector(TLE)
% Determines the State Vector (Radius and Velocity) given fields 3 - 8
 of
% line two of a two line element set
% [Inc RAAN ecc AoP Me n]

deg = pi/180;

mu = 3.986e5;

inc = TLE(1);

RAAN = TLE(2);

ecc = TLE(3);

AoP = TLE(4);

Me = TLE(5)*deg;

n = (TLE(6))/(24*3600);

Per = 1/n;

a = (((Per*sqrt(mu))/(2*pi)))^(2/3);

TA = (TAfromMe(Me,ecc))/deg;

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

[R,V] = rvFromCOEs(h,ecc,RAAN,inc,AoP,TA,mu);


R = R';

V = V';
function [TA] = TAfromMe(Me,ecc)
% Newton's Method to find E


TOL = 1e-8;

if Me < pi
   E_0 = Me + (ecc/2); % rad
else
    E_0 = Me - (ecc/2); % rad
end
```

```
f = @(E) Me - E + ecc*sin(E);

fprime = @(E) -1 + ecc*cos(E);

[E ,~,~,~,~] = Bornhorst_newtons(f, fprime, E_0, TOL);

TA = 2*atan((tan(E/2))/sqrt((1-ecc)/(1+ecc)));

end

function [R,V] = rvFromCOEs(h,ecc,RAAN,inc,AoP,TA,mu)
% Finds the R and V vecotrs in Major Body Centric Coordinates from
 COEs

% Compute the radius and velocity in the perifocal plane

r = ((h^2)/mu) * (1/(1 + ecc*cos(TA)));
v = (mu/h);

r_pf = r.*[cos(TA);sin(TA);0];
v_pf = v.*[-sin(TA);ecc + cos(TA);0];

% Rotate from Perifocal to Heliocentric Equatorial

[R,V] = DCM_PerifocalToECI(r_pf,v_pf,inc,RAAN,AoP);

end

function [r_ECI,v_ECI] = DCM_PerifocalToECI(r_pf,v_pf,inc,RAAN,AoP)
%Direction Cosine Matrix to rotate from Perifocal Frame to Centered
 Inertial Reference Frame Reference
%Frame. Taking in Euler Angles in Degrees

% Compute DCMs

R3_AoP = [cosd(AoP) sind(AoP) 0; -sind(AoP) cosd(AoP) 0; 0 0 1];

R1_inc = [1 0 0; 0 cosd(inc) sind(inc); 0 -sind(inc) cosd(inc)];

R3_RAAN = [cosd(RAAN) sind(RAAN) 0; -sind(RAAN) cosd(RAAN) 0; 0 0 1];

DCM_ECItoPF = R3_AoP*R1_inc*R3_RAAN;

r_ECI = DCM_ECItoPF'*r_pf;

v_ECI = DCM_ECItoPF'*v_pf;

end

function [r, count, x, error, errorRatio] = Bornhorst_newtons(f,
 fprime, x_0, TOL)
% Newton's Method of root approximation only requires one inital guess
 x_0
% to find the root of a funtion
```

```
% It utilizes the derivative of the function to approximate the root

% The iterations take the form of x_(n+1) = (x_n) - f(x_n)/f'(x_n).
 which
% is a form of the slope equation of a line, where x_(n+1) is the
 intercept of
% the tangent of (x_n)
x_1 = x_0 - f(x_0)/fprime(x_0);

% With the iteration being defined above the count begins at 1
count = 1;

% Define the list of all x values used in the iteration
x = [x_0; x_1];

% The error in the function is simple the difference between x_n and
% x_(n+1), as a root is found when the difference between iterations
 is
% lower than the given tolerance
err = abs(x_1 - x_0);
error = err;
errorRatio = 1;

while err > TOL

    % It begins by setting x_0 to the previous x_1
    x_0 = x_1;
    % Then sets x_1 to be the intercept of the tangent of f(x_0)
    x_1 = x_0 - f(x_0)/fprime(x_0);

    % Add to the counter
    count = count + 1;

    % Calculate the error
    errorSquared = err^2;
    err = abs(x_1 - x_0);
    ratio = err/errorSquared;

    x = [x; x_1];
    error = [error; err];
    errorRatio = [errorRatio; ratio];
    % Repeat until the root is found
end
% When a root is found output the last x value found which should be a
% close approximation of x
r = x(end);

end

end

% Julian Date Function

function JD = myJulianDate(Year,Month,Day,Hour,Min,Sec)
```

```
% Calculate j_0

Y = Year;

M = Month;

D = Day;

hr = Hour;

min = Min;

sec = Sec;

day_hr = hr + (min/60) + (sec/3600);

% Eq. from Lecture 1

j_0 = 367*Y - floor((7*(Y + floor((M+9)/12)))/4) + floor((275*M)/9) +
 D + 1721013.5;

% Add time component to J_0

JD = j_0 + (day_hr/24);

end

% Lambert's Solver

function [v1,v2,v_1,v_2] = lambertsGeneral(r_1,r_2,dt,tm,tol,ztol,mu)
%Computes the solution to Lambert's Problem given a R vectors 1 & 2
 and the
%desired time delta. Also requires the input of a desired path (either
%prograde or retrograde, being either a 1 or -1).

mu = mu;
musq = sqrt(mu);


% Note: Also requires the desired tolerance for the stumpff functions
 to
% minimize computation time

tolVect = [1:ztol];
fotC = factorial(2*tolVect+2);
fotS = factorial(2*tolVect+3);

% Compute the norm of each vector
r1 = norm(r_1);
r2 = norm(r_2);
rcross = cross(r_1,r_2);

% Calculate d# using a dot product
```

```
if tm == 1

    if rcross(3) >= 0
        dTa = acos(dot(r_1,r_2)/(r1*r2));
    else
        dTa = 2*pi - acos(dot(r_1,r_2)/(r1*r2));
    end

else
    if rcross(3) < 0
        dTa = acos(dot(r_1,r_2)/(r1*r2));
    else
        dTa = 2*pi - acos(dot(r_1,r_2)/(r1*r2));
    end
end
dTaDegs = rad2deg(dTa);
% From there we can begin finding the solution to Lamberts Problem
 using
% Universal Variables

% Calculate constants from givens

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

% Define the function y(z)

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

% define the Newton's Method to converge on the true Z value

% Plot the graph and select a good Z_0
% j = linspace(-3,20);
% Fplot = zeros(1,100);
%     for h = 1:length(j)
%         [C,S] = stumpffFunction(j(h),ztol,fotC,fotS);
%
%         yz = y(j(h),S,C);
%
%         Fplot(h) = ((yz/C)^1.5)*S + A*sqrt(yz) - musq*dt;
%
%     end
%     % Skipping the plotting and selecting a guess
%     figure(1)
%     plot(j,Fplot)
%     grid on
%
%
%     zone = ginput(1);
% z0 = zone(1); % defines an initial z of 0 which gives a parabolic
 orbit
%
%    close figure 1

z0 = 2;
```

```
[C0,S0] = stumpffFunction(z0,ztol,fotC,fotS);

y0 = y(z0,S0,C0);

i = 1; % initialize the "I'm missing a parentheses somewhere" counter

ratio = 100;

z = z0;

    while abs(ratio) > tol && i < 1000

        [C,S] = stumpffFunction(z,ztol,fotC,fotS);

        yz = y(z,S,C);

        F = ((yz/C)^1.5)*S + A*sqrt(yz) - musq*dt;

        if z ~= 0
            AAA = ((yz/C)^1.5)*((1/(2*z))*(C - ((3*S)/(2*C))) +
 (3/4)*(S^2/C));

            BBB = (A/8)*((3*S/C)*sqrt(yz) + A*sqrt(C/yz));

            Fprime = AAA + BBB;
        else

            Fprime = (sqrt(2)/4)*y0^1.5 + (A/8)*(sqrt(y0)+ A*sqrt(1/
(2*y0)));

        end

        ratio = F/Fprime;

        if abs(ratio) < tol
            zf = z;

        else

            z = z - ratio;

            i = i + 1;
        end

    end

[Cf,Sf] = stumpffFunction(zf,ztol,fotC,fotS);

yf = y(zf,Sf,Cf);

f = 1 - (yf/r1);

g = A*sqrt(yf/mu);
```

```
gdot = 1 - (yf/r2);

v_1 = (1/g)*(r_2 - f*r_1);

v_2 = (1/g)*(gdot*r_2 - r_1);

v1 = norm(v_1);

v2 = norm(v_2);

end

% COEs

function [a,ecc,incl,RAAN,AoP,nu,hn] = COEs_bornhorstMatthew(R,V,mu)
% Solve for C.O.E's for a given radius and momentum vector
%   Write a MATLAB function to calculate the classical orbital
 elements
% from any state vector of a spacecraft in an orbit around Earth.
% The function should have ?? and ?? as inputs (expressed in the ECI
 frame)
% and output semi-major axis (?), eccentricity (?), true anomaly (nu),
 inclination (?),
% RAAN (?), and argument of perigee (?).


% Set parameter
% Earth Grav parameter
mu_earth = mu;
% SCI Frame
iHat = [1 0 0];
jHat = [0 1 0];
kHat = [0 0 1];

Rmag = norm(R);
Vmag = norm(V);

% Solve for constants
% Specific Mech Energy
specMechEng = ((Vmag^2)/2) - mu_earth / Rmag; % km^2/s^3
% Specific Angular Momentum
h = cross(R,V);

hn = norm(h);

% Solve for COEs
% solve for semi-major axis

a = -(mu_earth /(2*specMechEng)); % Km

% Solve for eccentricity

eccVect = (1/mu_earth)*((Vmag^2 - (mu_earth/Rmag)).*R - dot(R,V)*V);
```

```
ecc = norm(eccVect);

% solve for inclination

incl = acosd(dot(kHat,h)/(norm(kHat)*norm(h)));

    % Solve for node vector
    nodeVect = cross(kHat,h);

% solve for RAAN

RAAN = acosd(dot(iHat,nodeVect)/(norm(iHat)*norm(nodeVect)));

% Quadrant Check RAAN

if nodeVect(2) < 0
    RAAN = 360 - RAAN;
end

% solve for argument of periapsis

AoP = acosd(dot(nodeVect,eccVect)/(norm(eccVect)*norm(nodeVect)));

% Quadrant Check AoP

if eccVect(3) < 0
    AoP = 360 - AoP;
end

% solve for true anomaly

nu = acosd(dot(eccVect,R)/(norm(eccVect)*norm(R)));

% Quadrant Check True Anomaly

if dot(R,V) < 0
    nu = 360 - nu;
end


end

% Planet States

function [R,V] = planetStates(JD,PlanetID)
%Given a planet, and a time in Julian Days, this function will
 calculate
%the state vector for a planet.

J2000 = 2451545.0; % Days

mu_sun = 1.3271e11;

deg = pi/180;
```

```
% 1. find Julian Date for the given date, but I'm having that as an
 input
% bc it makes more sense to just input the Julian Date bc thats an
 easy
% thing to do

% 2. find T_0 in julian Centuries

T_0 = (JD - J2000)/36525;


% 3. Use the matlab function provided by Dr. A to find the planetary
% elements at the given T0

[pCs] = AERO351planetary_elements2(PlanetID,T_0);


% Redefine them to make using them easier
a = pCs(1);

ecc = pCs(2);

inc = pCs(3)*deg ; % rad

RAAN = pCs(4)*deg ; % rad

w_bar = pCs(5)*deg ; % rad

L = pCs(6)*deg ;% rad

% 4/5. Find E from ecc and Me, and then just directly find TA bc its
 one
% more equation to add to the function
AoP = w_bar - RAAN;

Me = L - w_bar;% rad

TA = TAfromMe(Me,ecc);% rad

% 6. Find h

h = sqrt(a*mu_sun*(1-ecc^2));

% Get the state vector in HEI Frame from the COEs

[R,V] = rvFromCOEs(h,ecc,RAAN,inc,AoP,TA,mu_sun);

R = R';

V = V';
end

% Dr. A's Planetary Elements
```

```
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

if L > 360
    L = mod(L,360);
end

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

% Perifocal to ECI

function [r_ECI,v_ECI] = DCM_PerifocalToECI(r_pf,v_pf,inc,RAAN,AoP)
%Direction Cosine Matrix to rotate from Perifocal Frame to Centered
 Inertial Reference Frame Reference
%Frame. Taking in Euler Angles in Degrees

% Compute DCMs

R3_AoP = [cosd(AoP) sind(AoP) 0; -sind(AoP) cosd(AoP) 0; 0 0 1];

R1_inc = [1 0 0; 0 cosd(inc) sind(inc); 0 -sind(inc) cosd(inc)];

R3_RAAN = [cosd(RAAN) sind(RAAN) 0; -sind(RAAN) cosd(RAAN) 0; 0 0 1];

DCM_ECItoPF = R3_AoP*R1_inc*R3_RAAN;

r_ECI = DCM_ECItoPF'*r_pf;

v_ECI = DCM_ECItoPF'*v_pf;

end

% TA from Me

function [TA] = TAfromMe(Me,ecc)
% Newton's Method to find E


TOL = 1e-8;

if Me < pi
   E_0 = Me + (ecc/2); % rad
else
    E_0 = Me - (ecc/2); % rad
end
```

```
f = @(E) Me - E + ecc*sin(E);

fprime = @(E) -1 + ecc*cos(E);

[E ,~,~,~,~] = Bornhorst_newtons(f, fprime, E_0, TOL);

TA = 2*atan((tan(E/2))/sqrt((1-ecc)/(1+ecc)));

end

% Newtons Method

function [r, count, x, error, errorRatio] = Bornhorst_newtons(f,
 fprime, x_0, TOL)
% Newton's Method of root approximation only requires one inital guess
 x_0
% to find the root of a funtion
% It utilizes the derivative of the function to approximate the root

% The iterations take the form of x_(n+1) = (x_n) - f(x_n)/f'(x_n).
 which
% is a form of the slope equation of a line, where x_(n+1) is the
 intercept of
% the tangent of (x_n)
x_1 = x_0 - f(x_0)/fprime(x_0);

% With the iteration being defined above the count begins at 1
count = 1;

% Define the list of all x values used in the iteration
x = [x_0; x_1];

% The error in the function is simple the difference between x_n and
% x_(n+1), as a root is found when the difference between iterations
 is
% lower than the given tolerance
err = abs(x_1 - x_0);
error = err;
errorRatio = 1;

while err > TOL

    % It begins by setting x_0 to the previous x_1
    x_0 = x_1;
    % Then sets x_1 to be the intercept of the tangent of f(x_0)
    x_1 = x_0 - f(x_0)/fprime(x_0);

    % Add to the counter
    count = count + 1;

    % Calculate the error
    errorSquared = err^2;
    err = abs(x_1 - x_0);
```

```
    ratio = err/errorSquared;

    x = [x; x_1];
    error = [error; err];
    errorRatio = [errorRatio; ratio];
    % Repeat until the root is found
end
% When a root is found output the last x value found which should be a
% close approximation of x
r = x(end);

end

% RV from COES

function [R,V] = rvFromCOEs(h,ecc,RAAN,inc,AoP,TA,mu)
% Finds the R and V vecotrs in Major Body Centric Coordinates from
 COEs

% Compute the radius and velocity in the perifocal plane

r = ((h^2)/mu) * (1/(1 + ecc*cos(TA)));
v = (mu/h);

r_pf = r.*[cos(TA);sin(TA);0];
v_pf = v.*[-sin(TA);ecc + cos(TA);0];

% Rotate from Perifocal to Heliocentric Equatorial

[R,V] = DCM_PerifocalToECI(r_pf,v_pf,inc,RAAN,AoP);

end

%}
```

Published with MATLAB® R2021a