Table of Contents

8.2

```
------------P8.2------------
My calculations have the following results:
dv: 10.1536 km/s
H/C: This dv seems reasonable for an interplanetry launch to mars.
H/C: ecc of transfer: 0.5471seems to be reasonable for this trajectory (is
 ellpitical but not too elliptical).
```

8.4

```
------------P8.4------------
My calculations have the following results:
T-syn Mars/Jupiter: 816.0487 solar days
Note: solar days was the prefered unit in the book.
H/C: since this is larger than the orbital period of Mars, and Jupiter is
 moving much slower than Mars, this T-syn seems reasonable.
```

8.6

```
------------P8.6------------
My calculations have the following results:
SOI radii (km):
Saturn: 54787326.7306
Uranus: 51785640.1727
Neptune: 86596294.0734
H/C: these large SOI values correspond to gas giants.
H/C: these values are close to published valeus.
```

8.7

```
------------P8.7------------
My calculations have the following results:
Delta-V required: 3.337 km/s
Excess-V: 1.5789 km/s
H/C: ecc: 0.10247 is within a sensible range for this elliptical orbit.
H/C: Delta-V seems to be reasonable for this manouver.
```

8.12

```
------------P8.12------------
My calculations have the following results:
Delta-V: 10.5654 km/s
a: 4787748776.8487 km
ecc: 0.84522
H/C: the imparted Delta-V is well within the range for a Jupiter flyby, at a
 high altitude this relatively small (for jupiter) Delta-V makes sense.
H/C: the magnitude of Excess-V for arrival and departure is the same.
```

8.16

```
Warning: joshfLambert: This function may be useful but it is not well tested
and complete argument validation has not been implimented.
------------P8.16------------
My calculations have the following results:
Delta-V: 4.9724 km/s
H/C: 'long way' and 'short way' are almost the same, this makes sense since
 arrival is almost on the opposite side of the sun from deprature.
H/C: Delta-V is in a resonable range for a transfer to Mars.
Note: My answer doesn't match exactly with the book. In debugging I hard-coded
 in the book's 'r' vectors and noticed that my lambert's solver is working
 and also my coes->'r'&'v' vectors function is working. I also double checked
 my departure and arrival times. I think there may be something wrong with
 Curtis's 'rates' table or else there is just a decent amount of error in
 propogating planet's paths this way.
```

dependancies

```
------------Dependancies------------
My code uses the following functions:
    {'C:\AERO351\A351HW4\HW4.m'                              }
    {'C:\joshFunctionsMatlab\curtisPlanet_elements_and_sv.m'}
    {'C:\joshFunctionsMatlab\joshAnomalyCalculator.m'       }
    {'C:\joshFunctionsMatlab\joshAxisRotation.m'            }
    {'C:\joshFunctionsMatlab\joshCOE.m'                     }
    {'C:\joshFunctionsMatlab\joshCOE2rv.m'                  }
    {'C:\joshFunctionsMatlab\joshHomann.m'                  }
    {'C:\joshFunctionsMatlab\joshIsOnes.m'                  }
    {'C:\joshFunctionsMatlab\joshJulian.m'                  }
    {'C:\joshFunctionsMatlab\joshStumpffCoeffs.m'           }
    {'C:\joshFunctionsMatlab\joshStumpffZ.m'                }
    {'C:\joshFunctionsMatlab\joshVazVr.m'                   }
    {'C:\joshFunctionsMatlab\joshfLambert.m'                }
```

Published with MATLAB® R2022a

Table of Contents


```
clear all
close all
clc

addpath("C:\joshFunctionsMatlab\")

% Problems
% 8.2
% 8.4
% 8.6
% 8.7
% 8.12
% % 8.16
```

8.2

```
mu_s = 132712440018;
r1 = 227.9e6;
r2 = 778.5e6;
v1 = sqrt(mu_s/r1);
v2 = sqrt(mu_s/r2);

[dv1,dv2,dv,T,ht,ecct,vt1,vt2] = joshHomann(r1,v1,r2,v2,mu_s);

disp("------------P8.2------------")
disp("My calculations have the following results:")
disp("dv: "+string(dv)+" km/s")
disp("H/C: This dv seems reasonable for an interplanetry launch to mars.")
disp("H/C: ecc of transfer: "+string(ecct)+"seems to be reasonable for this
 trajectory (is ellpitical but not too elliptical).")
```

8.4

```
clear all
mu_s = 132712440018;
r1 = 227.9e6;
r2 = 778.5e6;
n1 = sqrt(mu_s/r1^3);
```

```
n2 = sqrt(mu_s/r2^3);
Tsyn = 2*pi/(n1-n2);
Tsyn = Tsyn/(60*60*24); %solar days


disp("------------P8.4------------")
disp("My calculations have the following results:")
disp("T-syn Mars/Jupiter: "+string(Tsyn)+" solar days")
disp("Note: solar days was the prefered unit in the book.")
disp("H/C: since this is larger than the orbital period of Mars, and Jupiter
 is moving much slower than Mars, this T-syn seems reasonable.")
```

8.6

```
clear all
mu_sun = 132712440018;
% rj = 778.5e6;
rs = 1.433e9;
ru = 2.872e9;
rn = 4.495e9;
% mu_j = 126686534;
mu_s = 37931187;
mu_u = 5793939;
mu_n = 6836529;

% rsoij = rj*(mu_j/mu_sun)^(2/5);
rsois = rs*(mu_s/mu_sun)^(2/5);
rsoiu = ru*(mu_u/mu_sun)^(2/5);
rsoin = rn*(mu_n/mu_sun)^(2/5);

disp("------------P8.6------------")
disp("My calculations have the following results:")
disp("SOI radii (km):")
disp("Saturn: "+ rsois)
disp("Uranus: "+ rsoiu)
disp("Neptune: "+ rsoin)
disp("H/C: these large SOI values correspond to gas giants.")
disp("H/C: these values are close to published valeus.")
```

8.7

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

mu_sun = 132712440018;
ra = 147.4e6;
rp = 120e6;
v1 = sqrt(mu_sun/ra);
a2 = (ra+rp)/2;
ecc2 = (ra-rp)/(ra+rp);
h2 = sqrt(mu_sun*ra*(1+ecc2*cos(pi)));
```

```
va = h2/ra;
vp = h2/rp;
vinf = v1-va;

vpark = sqrt(mu_e/rpark);
vbo = sqrt(vinf^2+2*mu_e/rpark);

dv = vbo-vpark;


disp("------------P8.7------------")
disp("My calculations have the following results:")
disp("Delta-V required: "+string(dv)+" km/s")
disp("Excess-V: "+string(vinf)+" km/s")
disp("H/C: ecc: "+string(ecc2)+" is within a sensible range for this
 elliptical orbit.")
disp("H/C: Delta-V seems to be reasonable for this manouver.")
```

8.12

```
clear all
mu_s = 132712440018;
mu_j = 126686534;
r_j = 778.5e6;
r_e = 149.6e6;
v_j = sqrt(mu_s/r_j);
v_e = sqrt(mu_s/r_e);

rsoij = 4.8215e7;

[dv1,dv2,dv,T,ht,ecct,vt1,vt2] = joshHomann(r_e,v_e,r_j,v_j,mu_s);

vsc1 = [vt2,0];
z = 200000;
rpj = 71490+z;
v_j = [1,0]*v_j;
vinf1 = vsc1-v_j;
sinf = norm(vinf1);
ecc = 1+(rpj*sinf^2)/mu_j;
h = mu_j*sqrt(ecc^2-1)/sinf;
B = acos(1/ecc);
d = 2*B;
vinf2 = sinf*[cos(d),-sin(d)];

vsc2 = vinf2+v_j;
dv = vsc2-vsc1;
dv = norm(dv);

vsc2 = [vsc2,0];
rsc2 = [0,r_j,0];
[a2,ecc2,theta2,inc2,raan2,aop2,h2,T2,E2] =
 joshCOE(rsc2,vsc2,mu_s,"magnitude");
```

```
disp("------------P8.12------------")
disp("My calculations have the following results:")
disp("Delta-V: "+string(dv)+" km/s")
disp("a: "+string(a2)+" km")
disp("ecc: "+string(ecc2))
disp("H/C: the imparted Delta-V is well within the range for a Jupiter flyby,
 at a high altitude this relatively small (for jupiter) Delta-V makes sense.")
disp("H/C: the magnitude of Excess-V for arrival and departure is the same.")
```

8.16

```
clear all

mu_sun = 132712440018;

tString = "August 15, 2005";
t = datetime(tString);
[~,~,j2000_1]= joshJulian(t);

tString = "March 15, 2006";
t = datetime(tString);
[~,~,j2000_2]= joshJulian(t);

T0_1 = j2000_1/36525; % confirmed T0 is curtis
T0_2 = j2000_2/36525;

% T0_1 = (jd1-2451545)/36525;
% T0_2 = (jd2-2451545)/36525;


% clear t tString j0_2 j0_1

% coe1 = abercrombyAERO351planetary_elements2(3,T0_1);
%
% a1 = coe1(1);
% ecc1 = coe1(2);
% inc1 = deg2rad(coe1(3));
% raan1 = deg2rad(coe1(4));
% w_hat1 = deg2rad(coe1(5));
% L1 = deg2rad(coe1(6));
% aop1 = deg2rad(w_hat1-raan1);
% Me1 = deg2rad(L1 - w_hat1);
% [theta1,~,E1] = joshAnomalyCalculator(ecc1,Me1,"Me")
%
% coe2 = abercrombyAERO351planetary_elements2(4,T0_2);


[coe1, r1, vearth, jd1] = curtisPlanet_elements_and_sv(3, 2005, 8, 15, 0, 0,
 0);
[coe2, r2, vmars, jd2] = curtisPlanet_elements_and_sv(4, 2006, 3, 15, 0, 0,
 0);
```

```
coefs = 15;
[Cc,Sc]=joshStumpffCoeffs(coefs);
C = @(z) sum(Cc.*joshStumpffZ(z,coefs));
S = @(z) sum(Sc.*joshStumpffZ(z,coefs));



dt = (j2000_2-j2000_1)*24*3600;

z = cross(r1,r2);
z = z(3);

theta1 = acos(dot(r1,r2)/(norm(r1)*norm(r2)));
theta2 = 2*pi - theta1;

[fz,y,A,z,flag,glag,gdotlag] =
 joshfLambert(norm(r1),norm(r2),dt,theta1,mu_sun,Cc,Sc);

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

vinf1 = abs(norm(v1) - norm(vearth));
vinf2 = abs(norm(v2) - norm(vmars));
clear A C Cc coe1 coe2 coefs dt flag fz gdotlag glag j2000_1 j2000_2 jd1 jd2 mu_sun r1 r2

r_e = 6378; % km
mu_e = 398600;
r_m = 3396;
mu_m = 42828;

z = 190;
rbo = z+r_e;
mu_e = 398600;
v0 = sqrt(mu_e/rbo);
vbo = sqrt(vinf1^2+(2*mu_e/rbo));
dv1 = abs(vbo-v0);

zp2 = 300;
rp2 = zp2 + r_m;

P = 35;% hr
P = P*60*60; % sec
a = (P*sqrt(mu_m)/(2*pi))^(2/3);
ra2 = 2*a-rp2;
ecc2 = (ra2-rp2)/(ra2+rp2);

h2 = sqrt(mu_m*rp2*(1+ecc2));
vp2 = h2/rp2;
vba = sqrt(vinf2^2+(2*mu_m/rp2));
dv2 = norm(vba-vp2);

dv = dv1+dv2;
```

```
disp("------------P8.16------------")
disp("My calculations have the following results:")
disp("Delta-V: "+string(dv)+" km/s")
disp("H/C: 'long way' and 'short way' are almost the same, this makes sense
 since arrival is almost on the opposite side of the sun from deprature.")
disp("H/C: Delta-V is in a resonable range for a transfer to Mars.")
disp("Note: My answer doesn't match exactly with the book. In debugging I
 hard-coded in the book's 'r' vectors and noticed that my lambert's solver is
 working and also my coes->'r'&'v' vectors function is working. I also double
 checked my departure and arrival times. I think there may be something wrong
 with Curtis's 'rates' table or else there is just a decent amount of error in
 propogating planet's paths this way.")
```

dependancies

```
disp("------------Dependancies------------")
disp("My code uses the following functions: ")
depends = matlab.codetools.requiredFilesAndProducts('C:\AERO351\A351HW4\HW4');
disp(depends')
```

Published with MATLAB® R2022a

```
function [M,E] = joshAnomalyCalculator(ecc,theta)
% 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}
    theta (1,1) double {mustBeReal}
end

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
end
```

Published with MATLAB® R2022a

```
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
```

Published with MATLAB® R2022a

```
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 = 360 - raan;
end

%aop
aop = acos(dot(n,e)/(nm*em));
if e(3) < 0
    aop = 360 - 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 = 360 -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
```

Published with MATLAB® R2022a

```matlab
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
```

Published with MATLAB® R2022a

```
function [fz,y,A,z,flag,glag,gdotlag] =
 joshfLambert(r1,r2,dt,theta,mu,C,S)%,z0)%,pr)
%{
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)));

% disp("josh")
% disp(theta)
% disp(r1)
% disp(r2)
% disp(A)

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
z = fzero(fz,0);

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)));

% my attempts
%WRONG:
% fp = @(z) (1/(2*sqrt(y(z)*C(z)^5)) )  *
 ( y(z)^2*((2*C(z)*dS(z))-3*dC(z)*S(z)) + dy(z)*(A*C(z)^(5/2) +
 3*C(z)*S(z)*y(z)) );
% dC = @(z) (1/(2*z)) * (C(z)-3*S(z));
% dS = @(z) (1/(2*z)) * (1-z*S(z)-2*C(z));
% dy = @(z)(A/(2*C(z)^1.5)) * ( (dC(z)*(1-z*S(z)) +  2*C(z)*(S(z)+z*dS(z))) );
% fpz0 = @(z)  sqrt(2)/40*y(0)^1.5 + A/8*(sqrt(y(0)) + A*sqrt(1/2/y(0)));
%
% fp = @(z) (y(z)/C(z))^1.5*(1/2/z*(C(z) - 3*S(z)/2/C(z))+ 3*S(z)^2/4/C(z)) +
 A/8*(3*S(z)/C(z)*sqrt(y(z))+ A*sqrt(C(z)/y(z)));
%%%%%%%%%%%


end
```

Published with MATLAB® R2022a

```
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 one of the returned dv's
% will 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
% 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
%     h1 = r1*v1;
%     h2 = r2*v2;
    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
```

Published with MATLAB® R2022a

```
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
```

Published with MATLAB® R2022a

```matlab
function [jd,thetaTime ,j2000, j0, ut, thetaG] = 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

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);
end
```

Published with MATLAB® R2022a

```
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
```

Published with MATLAB® R2022a

```
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
```

Published with MATLAB® R2022a

```
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}
    ecc (1,1) double {mustBeReal, mustBeNonnegative}
    h (1,1) double {mustBeReal,mustBePositive}
    mu (1,1) double {mustBeReal} = 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
```

Published with MATLAB® R2022a

```
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
function [coe, r, v, jd] = curtisPlanet_elements_and_sv ...
    (planet_id, year, month, day, hour, minute, second)
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
%{
This function calculates the orbital elements and the state
vector of a planet from the date (year, month, day)
and universal time (hour, minute, second).
mu - gravitational parameter of the sun (km^3/s^2)
deg - conversion factor between degrees and radians
pi - 3.1415926...
coe - vector of heliocentric orbital elements
[h e RA incl w TA a w_hat L M E],
where
h = angular momentum (km^2/s)
e = eccentricity
RA = right ascension (deg)
incl = inclination (deg)
w = argument of perihelion (deg)
TA = true anomaly (deg)
a = semimajor axis (km)
w_hat = longitude of perihelion ( = RA + w) (deg)
L = mean longitude ( = w_hat + M) (deg)
M = mean anomaly (deg)
E = eccentric anomaly (deg)
planet_id - planet identifier:
1 = Mercury
2 = Venus
3 = Earth
4 = Mars
5 = Jupiter
7 = Uranus
8 = Neptune
9 = Pluto
year - range: 1901 - 2099
month - range: 1 - 12
day - range: 1 - 31
hour - range: 0 - 23
minute - range: 0 - 60
second - range: 0 - 60
j0 - Julian day number of the date at 0 hr UT
ut - universal time in fractions of a day
jd - julian day number of the date and time
J2000_coe - row vector of J2000 orbital elements from Table 9.1
rates - row vector of Julian centennial rates from Table 9.1
t0 - Julian centuries between J2000 and jd
elements - orbital elements at jd
r - heliocentric position vector
v - heliocentric velocity vector
User M-functions required: J0, kepler_E, sv_from_coe
User subfunctions required: planetary_elements, zero_to_360
%}
```

```
% ---------------------------------------------------------------------
mu = 132712440018;
deg = pi/180;
%...Equation 5.48:

j0 = 367*year - fix(7*(year + fix((month + 9)/12))/4)+ fix(275*month/9) + day
 + 1721013.5;

ut = (hour + minute/60 + second/3600)/24;
%...Equation 5.47
jd = j0 + ut;
%...Obtain the data for the selected planet from Table 8.1:
[J2000_coe, rates] = planetary_elements(planet_id);
%...Equation 8.93a:
t0 = (jd - 2451545)/36525;
%...Equation 8.93b:
elements = J2000_coe + rates*t0;
a = elements(1);
e = elements(2);
%...Equation 2.71:
h = sqrt(mu*a*(1 - e^2));
%...Reduce the angular elements to within the range 0 - 360 degrees:
incl = elements(3);
RA = zero_to_360(elements(4));
w_hat = zero_to_360(elements(5));
L = zero_to_360(elements(6));
w = zero_to_360(w_hat - RA);
M = zero_to_360((L - w_hat));
%...Algorithm 3.1 (for which M must be in radians)
% E = kepler_E(e, M*deg);
[~,~,E] = joshAnomalyCalculator(e,M,'Me');

%...Equation 3.13 (converting the result to degrees):
TA = zero_to_360...
    (2*atan(sqrt((1 + e)/(1 - e))*tan(E/2))/deg);
coe = [h e RA incl w TA a w_hat L M E/deg];
%...Algorithm 4.5 (for which all angles must be in radians):

% testing my COE -> state vectors
% [r, v] = curtisSv_from_coe([h e RA*deg incl*deg w*deg TA*deg],mu)
[r,v] = joshCOE2rv(a,e,deg2rad(TA),deg2rad(incl),deg2rad(RA),deg2rad(w),mu);


return
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    function [J2000_coe, rates] = planetary_elements(planet_id)
        % ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        %{
This function extracts a planet's J2000 orbital elements and
centennial rates from Table 8.1.
planet_id - 1 through 9, for Mercury through Pluto
J2000_elements - 9 by 6 matrix of J2000 orbital elements for the nine
planets Mercury through Pluto. The columns of each
row are:
```

```
a = semimajor axis (AU)
e = eccentricity
i = inclination (degrees)
RA = right ascension of the ascending
node (degrees)
w_hat = longitude of perihelion (degrees)
L = mean longitude (degrees)
cent_rates - 9 by 6 matrix of the rates of change of the
J2000_elements per Julian century (Cy). Using "dot"
for time derivative, the columns of each row are:
a_dot (AU/Cy)
e_dot (1/Cy)
i_dot (arcseconds/Cy)
RA_dot (arcseconds/Cy)
w_hat_dot (arcseconds/Cy)
Ldot (arcseconds/Cy)
J2000_coe - row vector of J2000_elements corresponding
to "planet_id", with au converted to km
rates - row vector of cent_rates corresponding to
"planet_id", with au converted to km and
arcseconds converted to degrees
au - astronomical unit (km)
        %}
        % ---------------------------------------------------------------------
        J2000_elements = ...
            [ 0.38709893 0.20563069 7.00487 48.33167 77.45645 252.25084
            0.72333199 0.00677323 3.39471 76.68069 131.53298 181.97973
            1.00000011 0.01671022 0.00005 -11.26064 102.94719 100.46435
            1.52366231 0.09341233 1.85061 49.57854 336.04084 355.45332
            5.20336301 0.04839266 1.30530 100.55615 14.75385 34.40438
            9.53707032 0.05415060 2.48446 113.71504 92.43194 49.94432
            19.19126393 0.04716771 0.76986 74.22988 170.96424 313.23218
            30.06896348 0.00858587 1.76917 131.72169 44.97135 304.88003
            39.48168677 0.24880766 17.14175 110.30347 224.06676 238.92881];
        cent_rates = ...
            [ 0.00000066 0.00002527 -23.51 -446.30 573.57 538101628.29
            0.00000092 -0.00004938 -2.86 -996.89 -108.80 210664136.06
            -0.00000005 -0.00003804 -46.94 -18228.25 1198.28 129597740.63
            -0.00007221 0.00011902 -25.47 -1020.19 1560.78 68905103.78
            0.00060737 -0.00012880 -4.15 1217.17 839.93 10925078.35
            -0.00301530 -0.00036762 6.11 -1591.05 -1948.89 4401052.95
            0.00152025 -0.00019150 -2.09 -1681.4 1312.56 1542547.79
            -0.00125196 0.00002514 -3.64 -151.25 -844.43 786449.21
            -0.00076912 0.00006465 11.07 -37.33 -132.25 522747.90];
        J2000_coe = J2000_elements(planet_id,:);
        rates = cent_rates(planet_id,:);
        %...Convert from AU to km:
        au = 149597871;
        J2000_coe(1) = J2000_coe(1)*au;
        rates(1) = rates(1)*au;
        %...Convert from arcseconds to fractions of a degree:
        rates(3:6) = rates(3:6)/3600;
    end %planetary_elements
% ~~~~~~~~~~~~~~~~~~~~~~
```

```
    function y = zero_to_360(x)
        % ~~~~~~~~~~~~~~~~~~~~~~
        %{
This function reduces an angle to lie in the range 0 - 360 degrees.
x - the original angle in degrees
y - the angle reduced to the range 0 - 360 degrees
        %}
        % ---------------------------
        if x >= 360
            x = x - fix(x/360)*360;
        elseif x < 0
            x = x - (fix(x/360) - 1)*360;
        end
        y = x;
    end %zero_to_360
end %planet_elements_and_sv
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Published with MATLAB® R2022a