Table of Contents

Roshan Jaiswal-Ferri

```
%Aero 452 Homework 1: 9/24/25
```

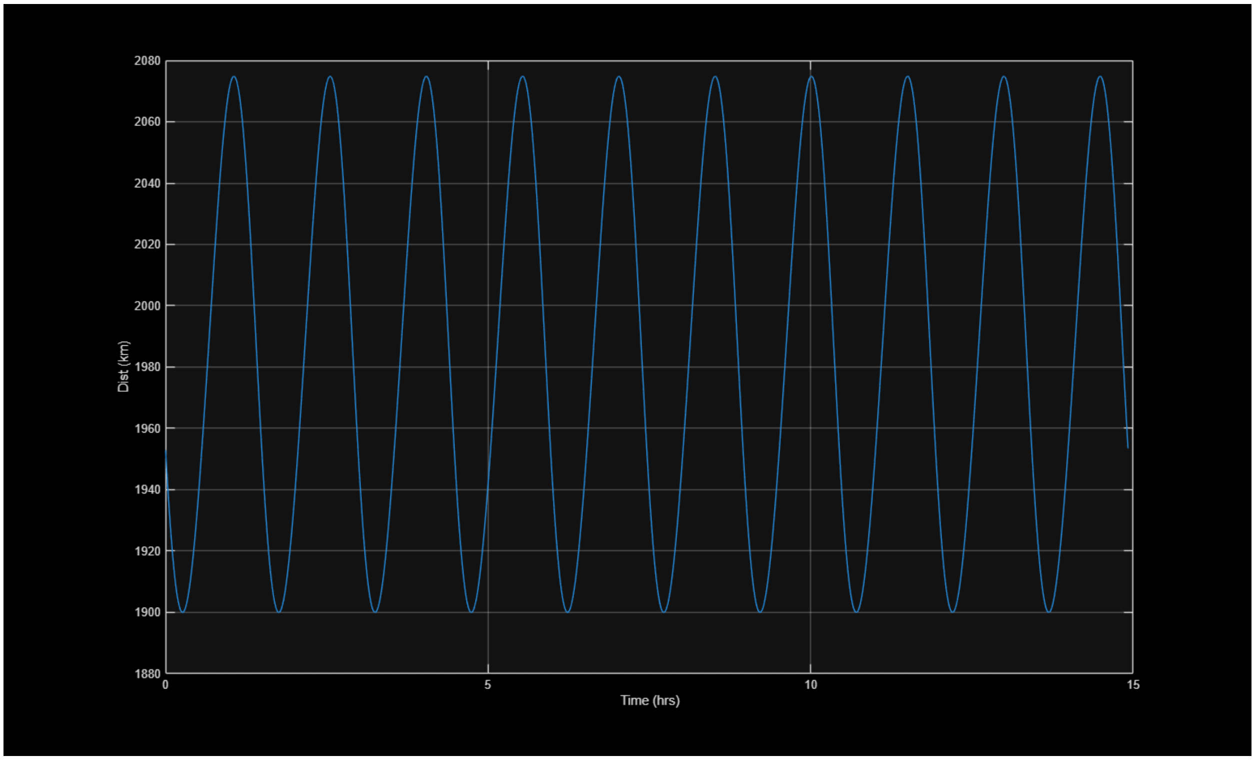
Workspace Prep

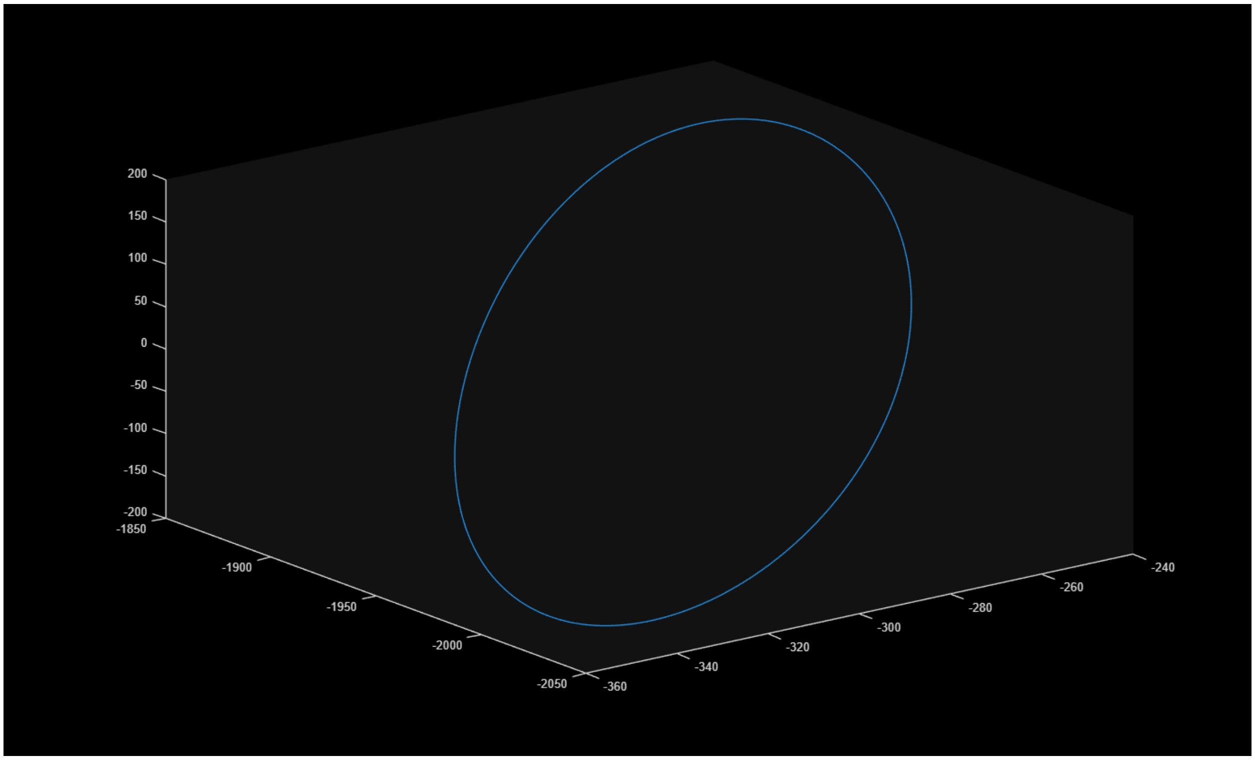
```
format long       %Allows for more accurate decimals
close all;        %Clears all
clear all;        %Clears Workspace
clc;              %Clears Command Window
```

PART 1: Closest Approach

```
mu = 398600;

SCa.h = 51400; % km2/s
SCa.ecc = 0.0006387;
SCa.inc = 51.65; %deg -> rad
SCa.raan = 15;
SCa.omega = 157;
SCa.theta = 15;

SCb.h = 51398; % km2/s
SCb.ecc = 0.0072696;
SCb.inc = 50;
SCb.raan = 15;
SCb.omega = 140;
SCb.theta = 15;

p = SCa.h^2/mu; % semi-latus rectum (km)
a = p/(1 - SCa.ecc^2);
Ta = 2*pi*sqrt(a^3/mu); % period of sc a (s)

[Ra_ECI, Va_ECI] = coes2rvd(a,SCa.ecc, SCa.inc, SCa.raan, SCa.omega,
SCa.theta, mu);
[Rb_ECI, Vb_ECI] = coes2rvd(a,SCb.ecc, SCb.inc, SCb.raan, SCb.omega,
SCb.theta, mu);
```

```
dt = Ta/1000; %1 second
x = 1;

for i = 1:dt:Ta*10
    rho_ECI = Rb_ECI - Ra_ECI;

    Q = ECI2LVLH(Ra_ECI,Va_ECI);

    rho(:,x) = Q*rho_ECI;
    rhomag(x) = norm(Q*rho_ECI);
    x = x + 1;

    [ra, va] = UniVarRV(Ra_ECI,Va_ECI,dt,mu);
    [rb, vb] = UniVarRV(Rb_ECI,Vb_ECI,dt,mu);

    Va_ECI = va;
    Vb_ECI = vb;
    Ra_ECI = ra;
    Rb_ECI = rb;

end

time = (Ta*10); %time in hours
time2 = (linspace(1,time,length(rhomag)))/3600;
[val, idx] = min(rhomag);

disp('Problem 1')
disp(['The Closest approach is at ', num2str(val), ' km'])
disp(['The Closest approach occurs at ', num2str(time2(idx)), ' hours'])
disp(' ')

figure('Name','Distance')
plot(time2,rhomag)
hold on
grid on
xlabel('Time (hrs)')
ylabel('Dist (km)')

figure('Name','3D Chaser')
plot3(rho(1,:),rho(2,:),rho(3,:))

% Discussion:
% The cyclical distance pattern between the two spacecraft makes sense
% because both of the spacecraft are in slightly different orbits that are
% modeled without disturbances. This means that the relative distance
% between the two spacecraft will decrease as they approach eachother in
% their orbits and increase as they fly apart. One reason why the relative
% distance does not stay exactly the same/perfectky cyclical over time is
% because the orbits do not have the same period, so each cycle the
% relative distance changes a little bit until after many orbits the
% relative distance will eventually also cycle back to the initial
% positions (without disturbances). This is why you see the graph is ever
% so slightly tilted.
```





PART 2: Relative Positions

```
Re = 6378; %km
ra0 = [0; Re+300; 0]; %all in ECI
```

```
va0 = [0; 0; sqrt(mu/norm(ra0))];
ha = cross(ra0,va0);

rb0 = [0; 0; Re+250];
vb0 = [0; -sqrt(mu/norm(rb0)); 0];

%pos
rho_ECI = rb0 - ra0;
Q = ECI2LVLH(ra0,va0);
rho = Q*rho_ECI; %relative radius

%vel
omega = ha / (norm(ra0)^2);
rhodot_ECI = vb0-va0 - cross(omega,rho_ECI);
rhodot = Q*rhodot_ECI;

%acc
omegadot = ( (-2*(dot(va0,ra0)))/(ra(2)^2) )*omega;
aa = -mu*(ra0/(norm(ra0)^3));
ab = -mu*(rb0/(norm(rb0)^3));
rhodoubledot_ECI = ab-aa - 2*(cross(omega,rhodot_ECI))...
    - cross(omegadot,rho_ECI) - cross(omega,cross(omega,rho_ECI));
rhodoubledot = Q*rhodoubledot_ECI;

disp('Problem 2, Curtis 7.1 (All Results in LVLH):')
disp(['Relative Position: ', num2str(rho'), ' km']);
disp(['Relative Velocity: ', num2str(rhodot'), ' km/s']); %All in LVLH
disp(['Relative Acceleration: ', num2str(rhodoubledot'), ' km/s^2']);
disp(' ')

% Discussion:
% The calculated results make sense as in the current LVLH frame and ECI
% both spacecraft a & b sit on specific axis and are on the same plane, so
% the two component pos vector (x y) and one component velocity vector (x)
% make sense. The acceleration was calculated with the newton's two body
% equations of motion (same that we used in ode45 in 351) and then
% subtituted to find the relative acceleration or rho double dot.
```

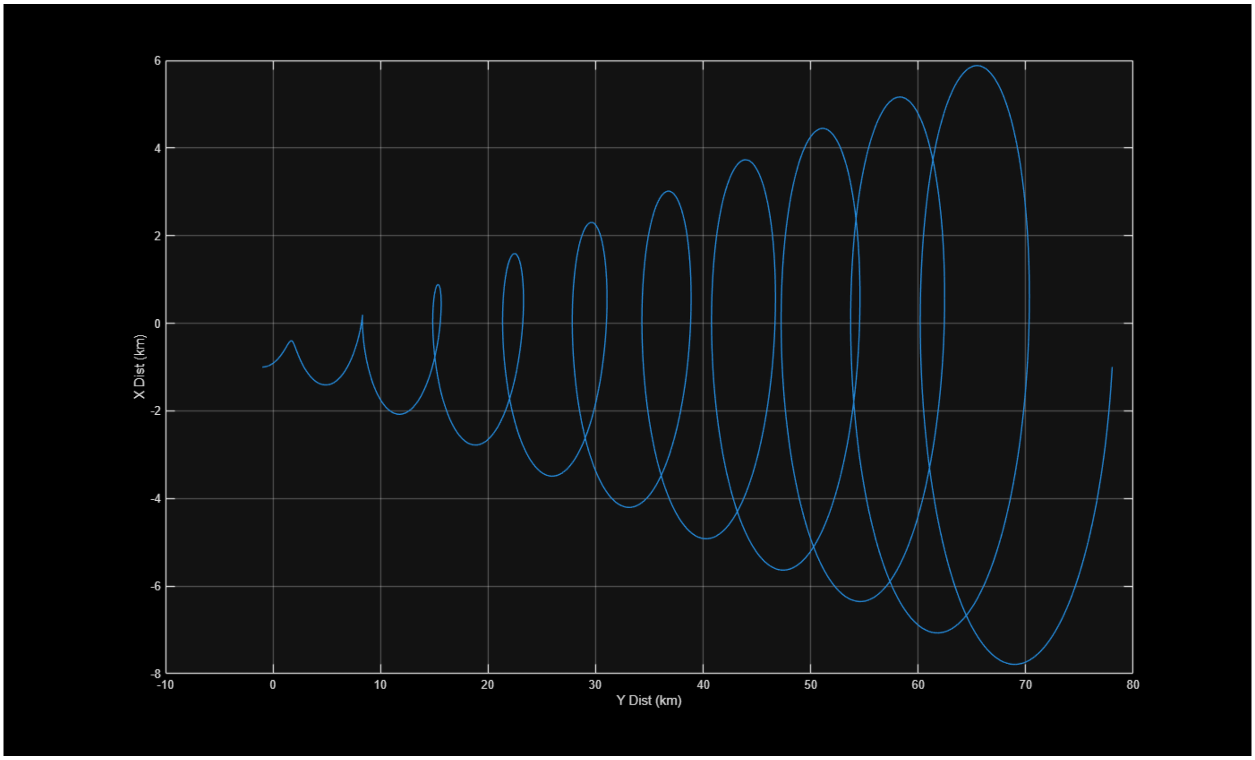
```
Problem 2, Curtis 7.1 (All Results in LVLH):
Relative Position: -6678   6628      0 km
Relative Velocity: -0.086932           0           0 km/s
Relative Acceleration: -1.7347e-18 -1.1402e-06           0 km/s^2
```

PART 3: Chief & Target Propogation

```
palt = 250; %km
e = 0.1;
inc = 51; %deg
raan = 0;
argp = 0;
theta = 0;
```

```
rp = Re + palt; %pergiee
a = rp/(1-e);

[R, V] = coes2rvd(a,e,inc,raan,argp,theta,mu);
RC = [-1; -1; 0]; % R and V vectors of chaser in LVLH/relative
VC = [0; 0.002; 0];

[~,~,~,~,~,~,~,p] = rv2coes(R,V,mu,Re); %period in seconds

tspan = [0, p*10];

state = [R; V; RC; VC];
options = odeset('RelTol',1e-8,'AbsTol',1e-8);

[~,relativeOrbits] = ode45(@relativeMotion,tspan,state,options,mu);

% R & V vectors for target and chaser
RT = [relativeOrbits(end,1),relativeOrbits(end,2),relativeOrbits(end,3)];
VT = [relativeOrbits(end,4),relativeOrbits(end,5),relativeOrbits(end,6)];

RC = [relativeOrbits(:,7),relativeOrbits(:,8),relativeOrbits(:,9)];
%personal note for when I copy paste: remember the line above is colons
%and contains the entire matrix of position over time the other four are
%only end positions
VC = [relativeOrbits(end,10),relativeOrbits(end,11),relativeOrbits(end,12)];

disp('Problem 3')
disp('Results are plotted')
disp(' ')

figure('Name','Relative Distance')
plot(RC(:,2),RC(:,1))
hold on
grid on
xlabel('Y Dist (km)')
ylabel('X Dist (km)')

% Discussion:
% This is a really cool graph that shows the relative distance of a target
% and chaser spacecraft. This graph shows the relative distance in the xy
% plane, it is interesting because it shows how there is more change in one
% direction than the other, it also shows the same pattern in problem one
% where the relative distance is cyclical, however using these equations of
% motion the results will lose accuracy quickly as the distance between
% the two spacecraft increases beyond a (relatively) small amount.
```

Problem 3
Results are plotted



PART 4: CW Solutions

```
T   = 90*60; % s
n   = 2*pi/T; % rad/s
t   = 15*60; % s
dr0 = [1; 0; 0]; % km
dv0 = [0; 0.01; 0]; % km/s

[Phi_rr, Phi_rv, Phi_vr, Phi_vv] = SolveCW(n, t);

% Propagate
dr = Phi_rr*dr0 + Phi_rv*dv0;
dv = Phi_vr*dr0 + Phi_vv*dv0;

disp('Problem 4, Curtis 7.7')
disp(['Relative position at 15 min: ', num2str(dr'), ' km']);
disp(['Relative speed at 15 min: ', num2str(dv'), ' km/s']);
disp(['Distance from station: ', num2str(norm(dr)), ' km']);

% Discussion:
% These results (while matching the book) also feel correct: there is a
% low relative velocity and both the relative position and velocity have no
% z component. The small relative distance is important because the CW
% equations only work with the assumption of close proximity. There is also
% no continuous burns and the orbits are circular.
```

Functions

```
function [Phi_rr, Phi_rv, Phi_vr, Phi_vv] = SolveCW(n, t)
    nt = n*t;
    s = sin(nt);
    c = cos(nt);

    Phi_rr = [ 4-3*c,            0,    0;
               6*(s-nt),         1,    0;
               0,                0,    c ];

    Phi_rv = [ (1/n)*s,          (2/n)*(1-c), 0;
               (2/n)*(c-1),      (1/n)*(4*s-3*nt), 0;
               0,                0,        (1/n)*s ];

    Phi_vr = [ 3*n*s,            0,    0;
               6*n*(c-1),        0,    0;
               0,                0,   -n*s ];

    Phi_vv = [ c,                2*s, 0;
              -2*s,        4*c-3,   0;
               0,                0,    c ];
end

function dstate = relativeMotion(time,state,mu)
%Use Column vectors!
%INPUTS: first 6 rows: [x y z dx dy dz...] in ECI, target properties
%CONTD: second 6 rows: [...x y z dx dy dz] in LVLH, relative to target
%OUTPUT: follows same convention

    %unpack for clarity (t for target c for chaser):
    tx0 = state(1); %pos
    ty0 = state(2);
    tz0 = state(3);
    tdx0 = state(4); %vel
    tdy0 = state(5);
    tdz0 = state(6);

    cx0 = state(7); %pos
    cy0 = state(8);
    cz0 = state(9);
    cdx0 = state(10); %vel
    cdy0 = state(11);
    cdz0 = state(12);

    rvect = [tx0 ty0 tz0]; %r and v vectors for chaser from chief
    vvect = [tdx0 tdy0 tdz0];

    rt = norm([tx0 ty0 tz0]); %r vector magnitudes
    rc = rt; %norm([cx0 cy0 cz0]);
    hc = norm(cross(rvect,vvect));

    %target
```

```
    tddx = -mu*tx0/rt^3;
    tddy = -mu*ty0/rt^3;
    tddz = -mu*tz0/rt^3;

    dstate_t = [tdx0; tdy0; tdz0; tddx; tddy; tddz];

    %chaser
    cddx = ((2*mu/rc^3)+(hc^2/rc^4))*cx0 - 2*(dot(vvect,rvect))*(hc/
rc^4)*cy0+((2*hc)/(rc^2))*cdy0;
    cddy = ((-mu/rc^3)+(hc^2/rc^4))*cy0 + 2*(dot(vvect,rvect))*(hc/
rc^4)*cx0-2*(hc/rc^2)*cdx0;
    cddz = -(mu/rc^3)*cz0;

    dstate_c = [cdx0; cdy0; cdz0; cddx; cddy; cddz];

    dstate = [dstate_t; dstate_c];

end

function Q = ECI2LVLH(R_ECI,V_ECI) %this is my function I wrote the desc
%Rotation Matrix for satellite relative positioning
%Usage: Q = ECI2LVLH[R,V] where inputs are properties of chief/target
%Position should be a 3x1 col vector: Q*posVec_ECI = posVec_LVLH

    ha = cross(R_ECI,V_ECI);

    ihat = R_ECI/norm(R_ECI);
    khat = ha/norm(ha);
    jhat = cross(khat,ihat);

    Q = [ihat'; jhat'; khat'];

end

function [r, v] = UniVarRV(r0, v0, dt, mu)
% Algorithm 3.4 (Credit Howard Curtis): Given r0, v0, find r, v at time dt
later.
% Usage:
%   [r, v] = rv_from_r0v0(r0, v0, dt, mu)
% Inputs:
%   r0  - 3x1 initial position vector (km)
%   v0  - 3x1 initial velocity vector (km/s)
%   dt  - time of flight (s)
%   mu  - gravitational parameter (km^3/s^2).
% Outputs:
%   r   - 3x1 position vector at t0+dt (km)
%   v   - 3x1 velocity vector at t0+dt (km/s)

    r0 = r0(:); v0 = v0(:);
    r0n  = norm(r0);                % |r0|
    v0n  = norm(v0);                % |v0|
    vr0  = dot(r0, v0)/r0n;         % radial velocity component v_r0  (Alg.
3.4 Step 1b).
```

```
    % Reciprocal semimajor axis alpha = 2/|r0| - |v0|^2/mu (Alg. 3.4 Step
1c).
    alpha = 2/r0n - (v0n^2)/mu;

    % Solve universal Kepler's equation for chi (Algorithm 3.3)
    chi = kepler_U(dt, r0n, vr0, alpha, mu);

    % Lagrange coefficients f, g and derivatives fdot, gdot via universal
variables (Eqs. 3.69).
    [f, g, fdot, gdot, rmag] = f_and_g(chi, dt, r0n, alpha, mu);

    % Propagate state (Eqs. 3.67-3.68): r = f r0 + g v0; v = fdot r0 + gdot
v0.
    r = f.*r0 + g.*v0;
    v = fdot.*r0 + gdot.*v0;

    % Normalize any tiny numerical imaginary parts to real
    r = real(r); v = real(v);

    % --------- Nested dependencies ---------

    function chi = kepler_U(dt, r0n, vr0, alpha, mu)
        % Solve universal Kepler's equation for chi using Newton's method
(Alg. 3.3)
        sqrtmu = sqrt(mu);

        % Initial guess (Battin-style; robust across conic types)
        if abs(alpha) > 1e-12
            chi = sqrtmu*abs(alpha)*dt;
        else
            % Parabolic limit; use Barker-like guess
            h = norm(cross(r0, v0));
            s = 0.5*pi*sqrtmu*dt/(r0n);
            chi = sqrtmu*dt/(r0n); % scale with time
        end

        tol = 1e-8; maxit = 50;
        for k = 1:maxit
            z  = alpha*chi^2;
            [C, S] = stumpff(z);

            % Universal Kepler equation F(chi) = 0:
            F  = (r0n*vr0/sqrtmu)*chi^2*C + (1 - alpha*r0n)*chi^3*S +
r0n*chi - sqrtmu*dt;

            % Derivative dF/dchi (standard closed form):
            dF = (r0n*vr0/sqrtmu)*chi*(1 - z*S) + (1 - alpha*r0n)*chi^2*C +
r0n;

            delta = F/dF;
            chi   = chi - delta;

            if abs(delta) < tol, break; end
        end
```

```
    end

    function [f, g, fdot, gdot, rmag] = f_and_g(chi, dt, r0n, alpha, mu)
        % Lagrange coefficients and their derivatives using universal
variables (Eqs. 3.69)
        z = alpha*chi^2;
        [C, S] = stumpff(z);

        f = 1 - (chi^2/r0n)*C;
        g = dt - (1/sqrt(mu))*chi^3*S;

        % New radius magnitude via r = f r0 + g v0; but for coefficients we
need r = |r|
        r_vec = f.*r0 + g.*v0;
        rmag  = norm(r_vec);

        fdot =  sqrt(mu)/(rmag*r0n) * (alpha*chi^3*S - chi);
        gdot = 1 - (chi^2/rmag)*C;
    end
end
```

```
Problem 4, Curtis 7.7
Relative position at 15 min: 11.0944      1.68473             0 km
Relative speed at 15 min: 0.020344   -0.013491           0 km/s
Distance from station: 11.2216 km
```

AERO 351 Functions

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

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

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

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

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

    RT = R1';
    VT = V1';
end

function [hM,a,e,nu,i,RAAN,w,p,t,en,Alta,Altp] = rv2coes(R,V,mu,r)
%Function for finding orbital state vectors RV
%   Input is in SI & %ALL ANGLES IN RADIANS!!
%   [hM,a,e,nu,i,RAAN,w,p,t,en,Ra,Rp] = rv2coes(R,V,mu,r)
```

```
%   hM = specific angular momentum
%   a = semi-major axis
%   e = eccentricity
%   nu = true anamoly
%   i = inc
%   RAAN = Right angle asending node
%   w = argument of periapsis
%   p = period (s)
%   t = time since perigee passage
%   en = orbit energy
%   Ra = Radius of Apogee
%   Rp = Radius of Perigee
%   r = radius of orbiting planet


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

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

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


% PART 1: Initial Calculations for later

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


% PART 2: Calculating semi-major axis

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

% PART 3: Genreal equation calculation for period

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

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

e = norm(eV);

% PART 5: inclination in rad

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

```matlab
% PART 6: RAAN in rad

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

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

% PART 7: Argument of Periapsis in rad

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

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

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

% PART 9: Time since perigee passage

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

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

% PART 10: Calculating Energy

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

% PART 11: Calculating Apogee and Perigee Altitude

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


end
```

```
Problem 1
The Closest approach is at 1899.9873 km
The Closest approach occurs at 13.6911 hours
```

Published with MATLAB® R2025b