Table of Contents

Roshan Jaiswal-Ferri

```
%Aero 452 Homework 3: 11/4/25
```

Workspace Prep

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

Question 1 (Cowell, Encke, VoP):

```
mu = 398600;
Re = 6378; %km

Cd = 2.2; %assumption from in class
mass = 100; %kg
area = pi*(1e-3/2)^2; %km^2

zp = 215; % km;
za = 939; % km;
rp = Re + zp;
ra = Re + za;
a0 = (ra+rp)/2;
e0 = (ra-rp)/(ra+rp);

raan0 = 340; % degs
inc0 = 65.2; % degs
omega0 = 58; % degs
theta0 = 332; % degs

[R0, V0] = coes2rvd(a0,e0,inc0,raan0,omega0,theta0,mu);
Rsc = R0;
```

```
Vsc = V0;

we = 72.9211e-6; %rad/s for the angular velocity of the earth
wE = [0; 0; 7.2921159e-5];

totalTime = 120*86400; %120 days in seconds
tspan = [0, totalTime];
state = [Rsc; Vsc];
```

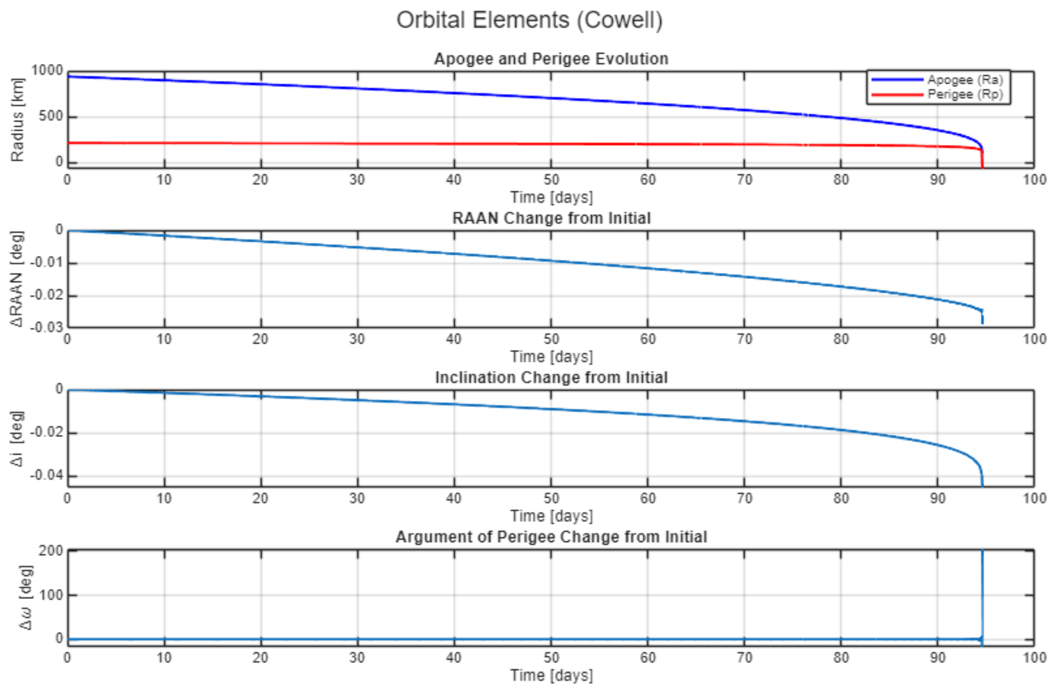
Simple 2-Body

```
% options = odeset('RelTol',1e-12,'AbsTol',1e-12);
% [time2End,twobody] = ode45(@twobodymotion,tspan,state,options,mu);
%
% Rbody = [twobody(:,1),twobody(:,2),twobody(:,3)];
% Vbody = [twobody(:,4),twobody(:,5),twobody(:,6)];
%
% for i = 1:length(Rbody) %takes 19 seconds
%     R = Rbody(i,:);
%     V = Vbody(i,:);
%     [~,~,~,~,inc2(i),RAAN2(i),w2(i),~,~,~,Ra2(i),Rp2(i)] =
rv2coes(R,V,mu,Re);
%     inc2 = rad2deg(inc2);
%     RAAN2 = rad2deg(RAAN2);
%     w2 = rad2deg(w2);
% end
```

Cowell

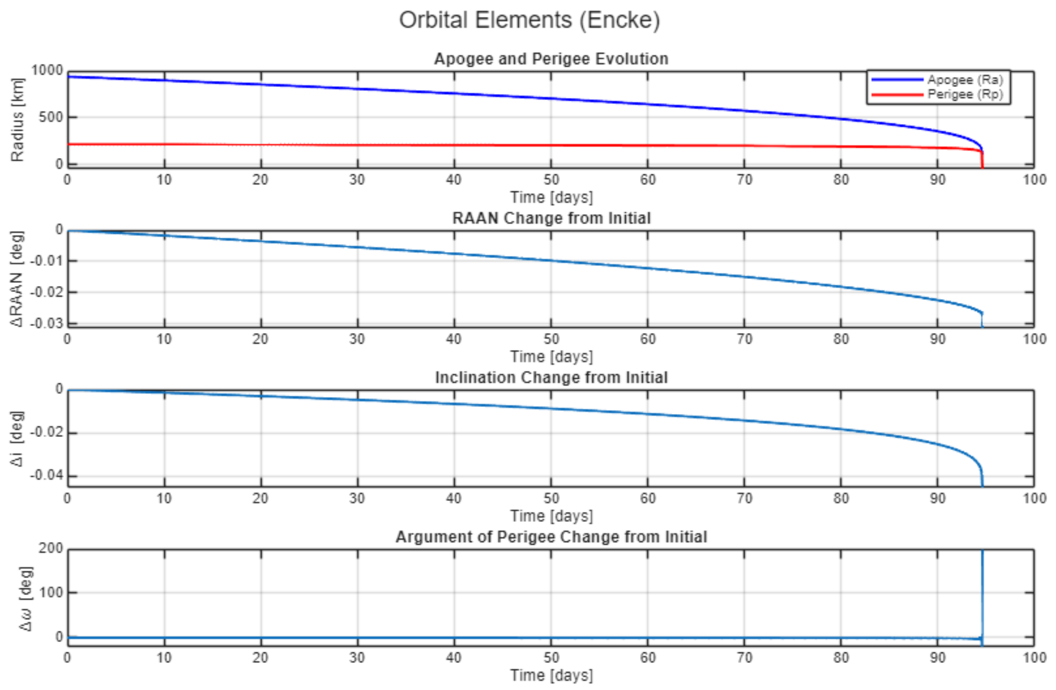
```
options = odeset('RelTol',1e-12,'AbsTol',1e-12,'Events',@reentryEvent);

%[timeCend,cowellMotion] =
ode45(@cowell,tspan,state,options,mu,mass,area,Cd); %takes 29 seconds to run
load("cowellData.mat")

% R & V vectors
Rcowell = [cowellMotion(:,1),cowellMotion(:,2),cowellMotion(:,3)];
Vcowell = [cowellMotion(:,4),cowellMotion(:,5),cowellMotion(:,6)];


% for i = 1:length(Rcowell) %takes 19 seconds
%     R = Rcowell(i,:);
%     V = Vcowell(i,:);
%     [~,~,~,~,incC(i),RAANC(i),wC(i),~,~,~,RaC(i),RpC(i)] =
rv2coes(R,V,mu,Re);
% end

% incC = rad2deg(incC);
% RAANC = rad2deg(RAANC);
% wC = rad2deg(wC);

time_daysc = timeCend / 86400;
```

```
% figure
% plot3(Rcowell(:,1),Rcowell(:,2),Rcowell(:,3))
% hold on
% grid on
% plot3(Rcowell(1,1),Rcowell(1,2),Rcowell(1,3),'*')
% %plot3(Rbody(:,1),Rbody(:,2),Rbody(:,3))

figure('Name','Orbital Elements (Cowell)','NumberTitle','off');
subplot(4,1,1);
plot(time_daysc, RaC, 'b', 'LineWidth', 1.5); hold on;
plot(time_daysc, RpC, 'r', 'LineWidth', 1.5);
grid on;
xlabel('Time [days]');
ylabel('Radius [km]');
title('Apogee and Perigee Evolution');
legend('Apogee (Ra)','Perigee (Rp)','Location','best');

subplot(4,1,2);
plot(time_daysc, RAANC - raan0, 'LineWidth', 1.5);
grid on;
xlabel('Time [days]');
ylabel('\DeltaRAAN [deg]');
title('RAAN Change from Initial');

subplot(4,1,3);
plot(time_daysc, incC - inc0, 'LineWidth', 1.5);
grid on;
xlabel('Time [days]');
ylabel('\Deltai [deg]');
title('Inclination Change from Initial');

subplot(4,1,4);
plot(time_daysc, wC - omega0, 'LineWidth', 1.5);
grid on;
xlabel('Time [days]');
ylabel('\Delta\omega [deg]');
title('Argument of Perigee Change from Initial');

sgtitle('Orbital Elements (Cowell)');
```



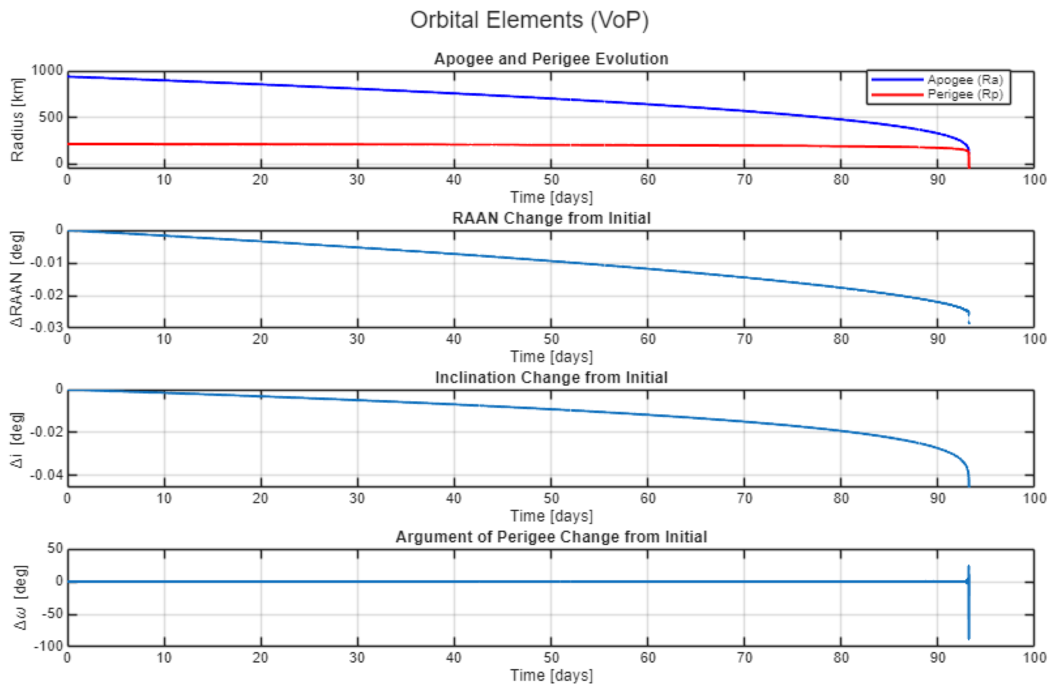
Encke

```
aream = pi*(0.5)^2;
dt = 60;

% tic
% [timeEend, ~, Rencke, Vencke] = encke(R0, V0, dt, totalTime, Cd, mass,
aream); %takes 27.9 seconds
% toc
load("enckeData.mat")

% figure
% plot3(Rencke(:,1),Rencke(:,2),Rencke(:,3))
% hold on
% grid on
% plot3(Rencke(1,1),Rencke(1,2),Rencke(1,3),'*')

% tic
% for i = 1:length(Rencke) %takes 76 seconds
%     % R = Rencke(i,:);
%     % V = Vencke(i,:);
%     [~,~,~,~,ince(i),RAANe(i),we(i),~,~,~,Rae(i),Rpe(i)] =
rv2coes(Rencke(i,:),Vencke(i,:),mu,Re);
% end
% toc
%
% ince = rad2deg(ince);
% RAANe = rad2deg(RAANe);
% we = rad2deg(we);

time_dayse = timeEend / 86400;

figure('Name','Orbital Elements (Encke)','NumberTitle','off');
subplot(4,1,1);
plot(time_dayse, Rae, 'b', 'LineWidth', 1.5); hold on;
plot(time_dayse, Rpe, 'r', 'LineWidth', 1.5);
grid on;
xlabel('Time [days]');
ylabel('Radius [km]');
title('Apogee and Perigee Evolution');
legend('Apogee (Ra)','Perigee (Rp)','Location','best');

subplot(4,1,2);
plot(time_dayse, RAANe - raan0, 'LineWidth', 1.5);
grid on;
xlabel('Time [days]');
ylabel('\DeltaRAAN [deg]');
title('RAAN Change from Initial');

subplot(4,1,3);
plot(time_dayse, ince - inc0, 'LineWidth', 1.5);
grid on;
xlabel('Time [days]');
ylabel('\Deltai [deg]');
title('Inclination Change from Initial');

subplot(4,1,4);
plot(time_dayse, we - omega0, 'LineWidth', 1.5);
grid on;
xlabel('Time [days]');
ylabel('\Delta\omega [deg]');
title('Argument of Perigee Change from Initial');

sgtitle('Orbital Elements (Encke)');
```



VoP

```
% options = odeset('RelTol',1e-12,'AbsTol',1e-12,'Events',@reentryEventCoes);
% [h,~,ecc,theta,i,RAAN,w] = rv2coes(R0,V0,mu,Re);
% state = [h,ecc,RAAN,i,w,theta];
% tic
% [timeVend,VoPMotion] = ode45(@VoP,tspan,state,options,mu,mass,aream,Cd);
%takes 22.7 seconds to run
% toc
load("VoPData.mat")
%
% % R & V
% coes = [VoPMotion(:,1),VoPMotion(:,2),VoPMotion(:,3),VoPMotion(:,4),...
%     VoPMotion(:,5),VoPMotion(:,6)];
%
hV = coes(:,1);
eccV = coes(:,2);
raanV = coes(:,3);
incV = coes(:,4);
wV = coes(:,5);
thetaV = coes(:,6);
%
% tic
% for k = 1:length(coes)
%     [R,V] = coes2rv(hV(k), eccV(k), incV(k), raanV(k), wV(k), thetaV(k),
mu);
%     a  = hV(k)^2 / (mu*(1 - eccV(k)^2));
%     RaV(k) = a*(1 + eccV(k)) - Re;
%     RpV(k) = a*(1 - eccV(k)) - Re;
% end
% toc

time_daysv = timeVend / 86400;

figure('Name','Orbital Elements (VoP)','NumberTitle','off');
subplot(4,1,1);
plot(time_daysv, RaV, 'b', 'LineWidth', 1.5); hold on;
plot(time_daysv, RpV, 'r', 'LineWidth', 1.5);
grid on;
xlabel('Time [days]');
ylabel('Radius [km]');
title('Apogee and Perigee Evolution');
legend('Apogee (Ra)','Perigee (Rp)','Location','best');

subplot(4,1,2);
plot(time_daysv, rad2deg(raanV) - raan0, 'LineWidth', 1.5);
grid on;
xlabel('Time [days]');
ylabel('\DeltaRAAN [deg]');
title('RAAN Change from Initial');

subplot(4,1,3);
plot(time_daysv, rad2deg(incV) - inc0, 'LineWidth', 1.5);
grid on;
xlabel('Time [days]');
ylabel('\Deltai [deg]');
title('Inclination Change from Initial');

subplot(4,1,4);
plot(time_daysv, rad2deg(wV) - omega0, 'LineWidth', 1.5);
grid on;
xlabel('Time [days]');
ylabel('\Delta\omega [deg]');
title('Argument of Perigee Change from Initial');

sgtitle('Orbital Elements (VoP)');
```

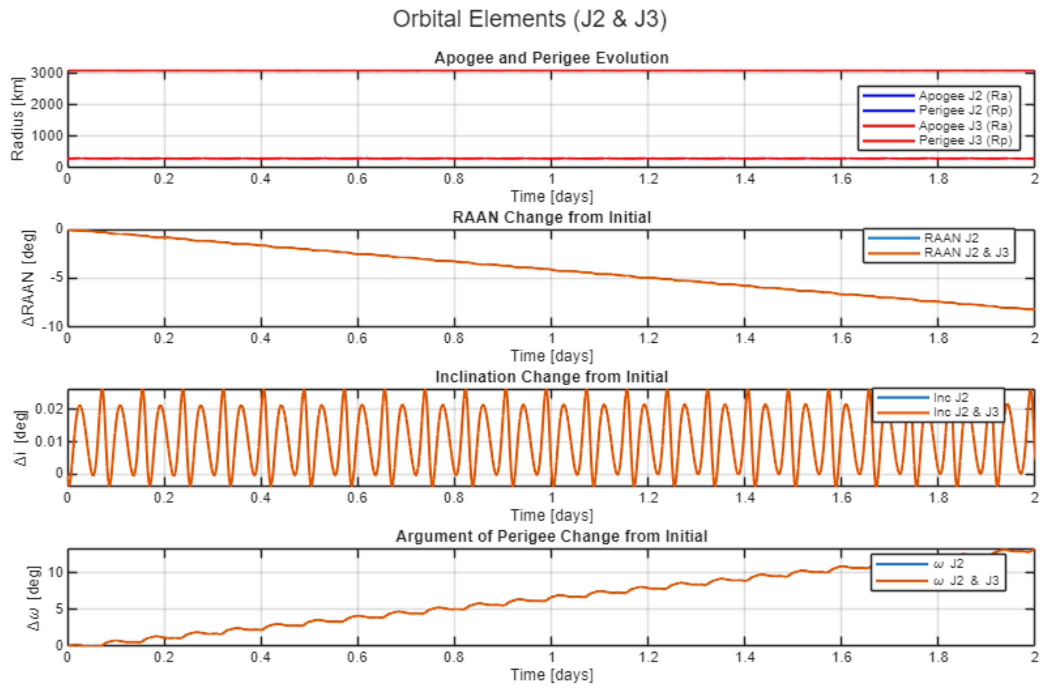


Question 2 (J2 & J3):

```
zp2 = 300; % km;
za2 = 3092; %km;
raan2 = 45; % degs
inc2 = 28; % degs
omega2 = 30; % degs
theta2 = 40; % degs

rp2 = Re + zp2;
ra2 = Re + za2;
a2 = (ra2+rp2)/2;
e2 = (ra2-rp2)/(ra2+rp2);

[R2, V2] = coes2rvd(a2,e2,inc2,raan2,omega2,theta2,mu);

tspan2 = [0, 48*3600];
state2 = [R2; V2];

options = odeset('RelTol',1e-12,'AbsTol',1e-12,'Events',@reentryEvent);

[timeJ2end,J2Motion] = ode45(@J2,tspan2,state2,options,mu); %takes x seconds
to run
[timeJ3end,J3Motion] = ode45(@J3,tspan2,state2,options,mu); %takes x seconds
to run

%load("cowellData.mat")
```

```
% R & V vectors
RJ2 = [J2Motion(:,1),J2Motion(:,2),J2Motion(:,3)];
VJ2 = [J2Motion(:,4),J2Motion(:,5),J2Motion(:,6)];

RJ3 = [J3Motion(:,1),J3Motion(:,2),J3Motion(:,3)];
VJ3 = [J3Motion(:,4),J3Motion(:,5),J3Motion(:,6)];

% figure
% plot3(RJ2(:,1),RJ2(:,2),RJ2(:,3))
% hold on
% grid on
% plot3(RJ2(1,1),RJ2(1,2),RJ2(1,3),'*')

for i = 1:length(RJ2)
    [~,~,~,~,incJ2(i),RAANJ2(i),wJ2(i),~,~,~,RaJ2(i),RpJ2(i)] =
rv2coes(RJ2(i,:),VJ2(i,:),mu,Re);
end

for i = 1:length(RJ3)
    [~,~,~,~,incJ3(i),RAANJ3(i),wJ3(i),~,~,~,RaJ3(i),RpJ3(i)] =
rv2coes(RJ3(i,:),VJ3(i,:),mu,Re);
end

incJ2 = rad2deg(incJ2);
RAANJ2 = rad2deg(RAANJ2);
wJ2 = rad2deg(wJ2);

incJ3 = rad2deg(incJ3);
RAANJ3 = rad2deg(RAANJ3);
wJ3 = rad2deg(wJ3);

time_daysJ2 = timeJ2end / 86400;
time_daysJ3 = timeJ2end / 86400;

figure('Name','Orbital Elements (J2 & J3)','NumberTitle','off');
subplot(4,1,1);
plot(time_daysJ2, RaJ2, 'b', 'LineWidth', 1.5); hold on;
plot(time_daysJ2, RpJ2, 'b', 'LineWidth', 1.5);
plot(time_daysJ3, RaJ3, 'r', 'LineWidth', 1.5);
plot(time_daysJ3, RpJ3, 'r', 'LineWidth', 1.5);
grid on;
xlabel('Time [days]');
ylabel('Radius [km]');
title('Apogee and Perigee Evolution');
legend('Apogee J2 (Ra)','Perigee J2 (Rp)','Apogee J3 (Ra)','Perigee J3
(Rp)','Location','best');

subplot(4,1,2);
plot(time_daysJ2, RAANJ2 - raan2, 'LineWidth', 1.5); hold on;
plot(time_daysJ3, RAANJ3 - raan2, 'LineWidth', 1.5);
grid on;
xlabel('Time [days]');
ylabel('\DeltaRAAN [deg]');
title('RAAN Change from Initial');
```

```
legend('RAAN J2','RAAN J2 & J3','Location','best');


subplot(4,1,3);
plot(time_daysJ2, incJ2 - inc2, 'LineWidth', 1.5); hold on;
plot(time_daysJ3, incJ3 - inc2, 'LineWidth', 1.5);
grid on;
xlabel('Time [days]');
ylabel('\Deltai [deg]');
title('Inclination Change from Initial');
legend('Inc J2','Inc J2 & J3','Location','best');

subplot(4,1,4);
plot(time_daysJ2, wJ2 - omega2, 'LineWidth', 1.5); hold on;
plot(time_daysJ3, wJ3 - omega2, 'LineWidth', 1.5);
grid on;
xlabel('Time [days]');
ylabel('\Delta\omega [deg]');
title('Argument of Perigee Change from Initial');
legend('\omega J2','\omega J2 & J3','Location','best');

sgtitle('Orbital Elements (J2 & J3)');
```



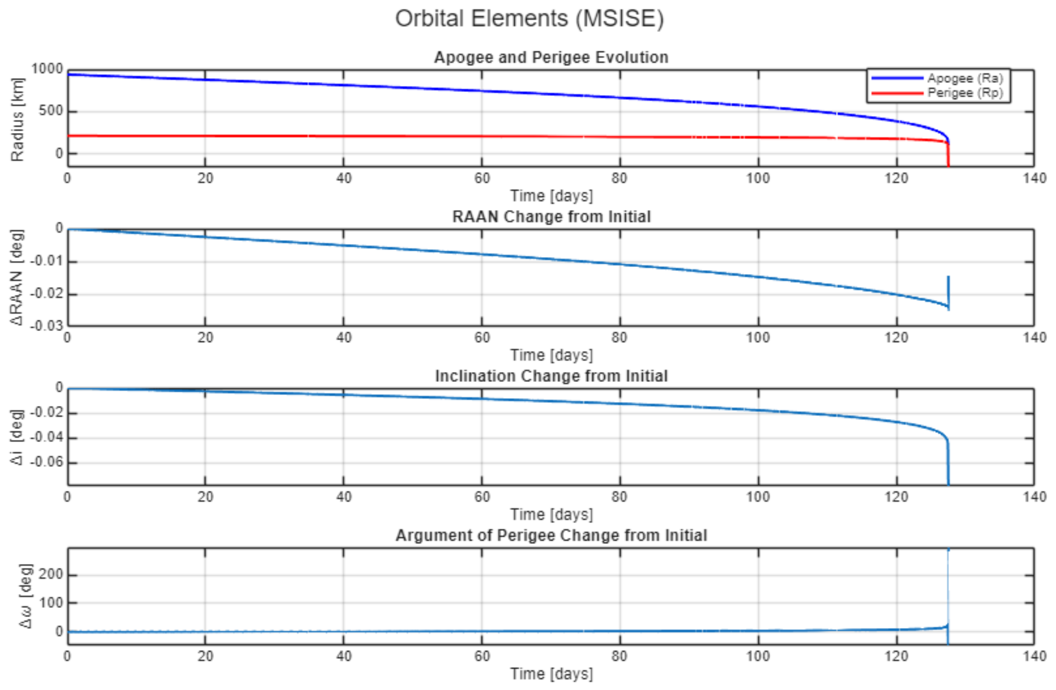
Question 3 (NRLMSISE)

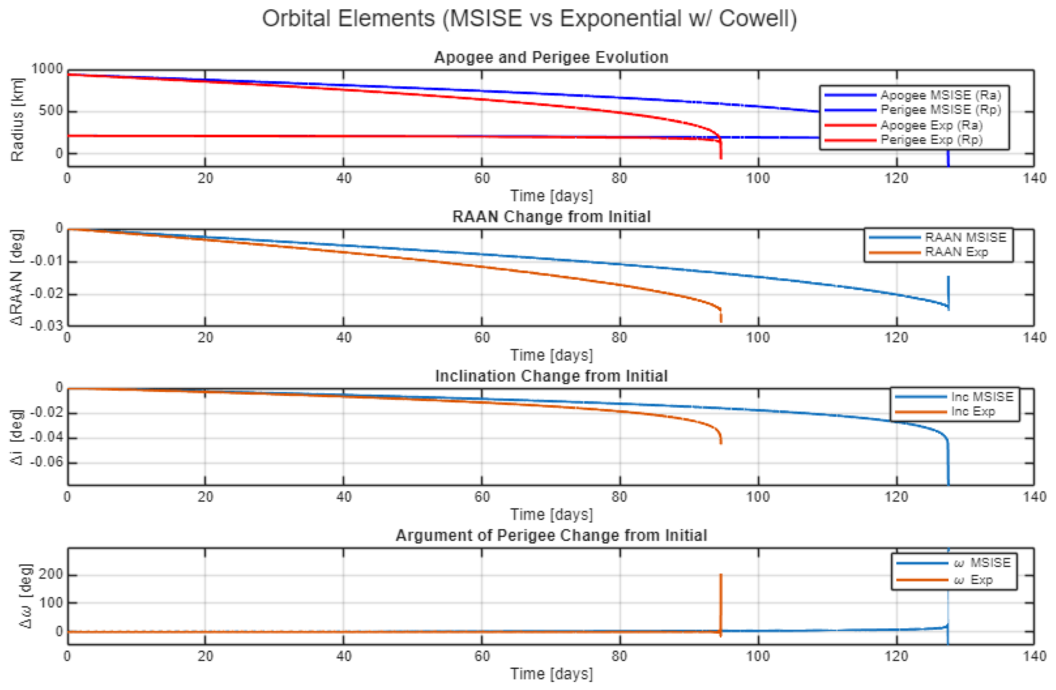
```
%You need a date for problem 3 so use 11/5/25 at 00:00:00 UT.

epochUTC = datetime(2025,11,5,0,0,0,'TimeZone','UTC');  % 11/5/25 00:00:00 UT
```

```
% options = odeset('RelTol',1e-12,'AbsTol',1e-12,'Events',@reentryEvent);
% totalTime = 140*86400; %140 days in seconds
% tspan = [0, totalTime];
%
% disp('started')
% tic
% [timeMend, MSISEMotion] = ode45( ...
%     @(t,x) cowellMSISE(t, x, mu, mass, area, Cd, epochUTC), ... %
2121.395287 seconds
%     tspan, state, options);
% toc
%
% % R & V vectors
% Rm = [MSISEMotion(:,1),MSISEMotion(:,2),MSISEMotion(:,3)];
% Vm = [MSISEMotion(:,4),MSISEMotion(:,5),MSISEMotion(:,6)];
load("MSISEdata.mat")
% figure
% plot3(Rm(:,1),Rm(:,2),Rm(:,3))
% hold on
% grid on
% plot3(Rm(1,1),Rm(1,2),Rm(1,3),'*')

for i = 1:length(Rm)
    [~,~,~,~,incM(i),RAANM(i),wM(i),~,~,~,RaM(i),RpM(i)] =
rv2coes(Rm(i,:),Vm(i,:),mu,Re);
end

incM = rad2deg(incM);
RAANM = rad2deg(RAANM);
wM = rad2deg(wM);

time_daysM = timeMend / 86400;

figure('Name','Orbital Elements (MSISE)','NumberTitle','off');
subplot(4,1,1);
plot(time_daysM, RaM, 'b', 'LineWidth', 1.5); hold on;
plot(time_daysM, RpM, 'r', 'LineWidth', 1.5);
grid on;
xlabel('Time [days]');
ylabel('Radius [km]');
title('Apogee and Perigee Evolution');
legend('Apogee (Ra)','Perigee (Rp)','Location','best');

subplot(4,1,2);
plot(time_daysM, RAANM - raan0, 'LineWidth', 1.5);
grid on;
xlabel('Time [days]');
ylabel('\DeltaRAAN [deg]');
title('RAAN Change from Initial');

subplot(4,1,3);
plot(time_daysM, incM - inc0, 'LineWidth', 1.5);
grid on;
xlabel('Time [days]');
```

```
ylabel('\Deltai [deg]');
title('Inclination Change from Initial');

subplot(4,1,4);
plot(time_daysM, wM - omega0, 'LineWidth', 1.5);
grid on;
xlabel('Time [days]');
ylabel('\Delta\omega [deg]');
title('Argument of Perigee Change from Initial');

sgtitle('Orbital Elements (MSISE)');

figure('Name','Orbital Elements (MSISE vs Exponential w/
Cowell)','NumberTitle','off');
subplot(4,1,1);
plot(time_daysM, RaM, 'b', 'LineWidth', 1.5); hold on;
plot(time_daysM, RpM, 'b', 'LineWidth', 1.5);
plot(time_daysc, RaC, 'r', 'LineWidth', 1.5);
plot(time_daysc, RpC, 'r', 'LineWidth', 1.5);
grid on;
xlabel('Time [days]');
ylabel('Radius [km]');
title('Apogee and Perigee Evolution');
legend('Apogee MSISE (Ra)','Perigee MSISE (Rp)','Apogee Exp (Ra)','Perigee
Exp (Rp)','Location','best');

subplot(4,1,2);
plot(time_daysM, RAANM - raan0, 'LineWidth', 1.5); hold on;
plot(time_daysc, RAANC - raan0, 'LineWidth', 1.5);
grid on;
xlabel('Time [days]');
ylabel('\DeltaRAAN [deg]');
title('RAAN Change from Initial');
legend('RAAN MSISE','RAAN Exp','Location','best');


subplot(4,1,3);
plot(time_daysM, incM - inc0, 'LineWidth', 1.5); hold on;
plot(time_daysc, incC - inc0, 'LineWidth', 1.5);
grid on;
xlabel('Time [days]');
ylabel('\Deltai [deg]');
title('Inclination Change from Initial');
legend('Inc MSISE','Inc Exp','Location','best');

subplot(4,1,4);
plot(time_daysM, wM - omega0, 'LineWidth', 1.5); hold on;
plot(time_daysc, wC - omega0, 'LineWidth', 1.5);
grid on;
xlabel('Time [days]');
ylabel('\Delta\omega [deg]');
title('Argument of Perigee Change from Initial');
legend('\omega MSISE','\omega Exp','Location','best');
```

```
sgtitle('Orbital Elements (MSISE vs Exponential w/ Cowell)');
```





Discussion

```
fprintf(['The time differences reported here may not be super accurate as
\n' ...
    'they are much longer than expected, but the general trends should still
appear \n' ...
    'Cowells method took 29 seconds with the exponential model, and 35
minutes \n' ...
    'with the MSISE model. Enckes method took 27.9 seconds, and VoP took
22.7 seconds. \n' ...
    'VoP looks to be fastest because it has relatively cheaper math and  is
\n' ...
    'calculating the slowly changing coes directly instead of R & V. \n' ...
    'Enckes is a little slower because of the repitition of the two body
problem \n' ...
    'with the UV, but is faster than cowell because of the time stepping
being constant. \n' ...
    'Also a heart check for J2/J3 is that it does change at about 5 degrees
per day. \n' ...
    'The MSISE model shows slightly less atmospheric drag than the
exponential model \n' ...
    'which is seen in the slightly longer deorbit time. \n'])
```

```
The time differences reported here may not be super accurate as
they are much longer than expected, but the general trends should still
appear
Cowells method took 29 seconds with the exponential model, and 35 minutes
with the MSISE model. Enckes method took 27.9 seconds, and VoP took 22.7
seconds.
VoP looks to be fastest because it has relatively cheaper math and  is
calculating the slowly changing coes directly instead of R & V.
Enckes is a little slower because of the repitition of the two body problem
with the UV, but is faster than cowell because of the time stepping being
constant.
Also a heart check for J2/J3 is that it does change at about 5 degrees per
day.
The MSISE model shows slightly less atmospheric drag than the exponential
model
which is seen in the slightly longer deorbit time.
```

Functions

```
function dstate = cowellMSISE(t, state, mu, mass, area, Cd, epochUTC)
%FUNCTION put in descrip    %tspan state options rest
% State vec is: km, km/s. epochUTC: datetime('TimeZone','UTC')
    Re = 6378;
    wE = [0;0;7.2921159e-5];

    %define vars
    r_eci = state(1:3);
    v_eci = state(4:6);
    r = norm(r_eci);
    %timing
    thisUTC = epochUTC + seconds(t);
    [r_ecef_km, ~] = eci2ecef_iau(thisUTC, r_eci, v_eci);
    [lat_deg, lon_deg, alt_km] = ecef2lla_km(r_ecef_km);

    % Atmos density
    yr  = year(thisUTC);
    doy = day(thisUTC,'dayofyear');
    sec = hour(thisUTC)*3600 + minute(thisUTC)*60 + second(thisUTC);
    [~, rho_SI] = atmosnrlmsise00(alt_km*1000, lat_deg, lon_deg, yr, doy,
sec); % kg/m^3
    rho = rho_SI(6) * 1e9;   % -> kg/km^3

    % Relative speed (ECI)
    vrel = v_eci - cross(wE, r_eci);
    vrel_norm = norm(vrel);
    adrag = -0.5 * Cd * (area/mass) * rho * (vrel_norm * vrel); %km/s^2

    %accel: !!eqs of motion!!
    acc = -mu * r_eci / r^3 + adrag;

    dstate = [v_eci; acc];
end

function [r_ecef_km, v_ecef_kmps] = eci2ecef_iau(dtUTC, r_eci_km, v_eci_kmps)
% dtUTC is MATLAB datetime (timezone = 'UTC')
    omegaE = 7.2921159e-5; % rad/s
    theta = gmst_rad(dtUTC);  % Greenwich Mean Sidereal Time [rad]
    R3 = [ cos(theta)  sin(theta)  0;
          -sin(theta)  cos(theta)  0;
                0           0      1];
    r_ecef_km = R3 * r_eci_km;
    v_ecef_kmps = R3 * (v_eci_kmps - cross([0;0;omegaE], r_eci_km));
end

function theta = gmst_rad(dtUTC)
    jd = juliandate(dtUTC);
    T = (jd - 2451545.0)/36525; % centuries since J2000
    theta = deg2rad(280.46061837 + 360.98564736629*(jd-2451545) ...
            + 0.000387933*T.^2 - (T.^3)/38710000);
    theta = mod(theta, 2*pi);
end

function [lat_deg, lon_deg, alt_km] = ecef2lla_km(r_ecef_km)
    x = r_ecef_km(1); y = r_ecef_km(2); z = r_ecef_km(3);
    a = 6378.137; % km
    f = 1/298.257223563;
    e2 = f*(2-f);
    lon = atan2(y,x);

    p = hypot(x,y);
    lat = atan2(z, p*(1-e2));
    for k=1:3
        N = a / sqrt(1 - e2*sin(lat)^2);
        alt = p/cos(lat) - N;
        lat = atan2(z, p*(1 - e2*N/(N+alt)));
    end
    N = a / sqrt(1 - e2*sin(lat)^2);
    alt = p/cos(lat) - N;

    lat_deg = rad2deg(lat);
    lon_deg = rad2deg(wrapToPi(lon));
    alt_km = alt;
end

function dstate = J3(time,state,mu) %dstate is derivitve of state
%FUNCTION put in descrip
    Re = 6378; %km

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

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

    J_2= 1.08263e-3;
    J2x = ((-3*J_2*mu*(Re^2)*x)/(2*rmag^5))*(1-((5*z^2)/rmag^2)); %x
    J2y = ((-3*J_2*mu*(Re^2)*y)/(2*rmag^5))*(1-((5*z^2)/rmag^2)); %y
    J2z = ((-3*J_2*mu*(Re^2)*z)/(2*rmag^5))*(3-((5*z^2)/rmag^2)); %z

    J_3 = 2.535656e-6;
    J3x = ((-5*J_3*mu*(Re^2)*x)/(2*rmag^7))*(3*z-((7*z^3)/rmag^2));
    J3y = ((-5*J_3*mu*(Re^2)*y)/(2*rmag^7))*(3*z-((7*z^3)/rmag^2));
    J3z = ((-5*J_3*mu*(Re^2))/...
    (2*rmag^7))*(6*z^2-((7*z^4)/rmag^2)-(3*rmag^2)/5);

    %accel: !!eqs of motion!!
    ddx = (-mu*x/rmag^3) + J2x + J3x;
    ddy = (-mu*y/rmag^3) + J2y + J3y;
    ddz = (-mu*z/rmag^3) + J2z + J3z;

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

end

function dstate = J2(time,state,mu) %dstate is derivitve of state
%FUNCTION put in descrip
    Re = 6378; %km

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

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

    J_2= 1.08263e-3;
    J2x = ((-3*J_2*mu*(Re^2)*x)/(2*rmag^5))*(1-((5*z^2)/rmag^2)); %x
    J2y = ((-3*J_2*mu*(Re^2)*y)/(2*rmag^5))*(1-((5*z^2)/rmag^2)); %y
    J2z = ((-3*J_2*mu*(Re^2)*z)/(2*rmag^5))*(3-((5*z^2)/rmag^2)); %z

    %accel: !!eqs of motion!!
    ddx = (-mu*x/rmag^3) + J2x;
    ddy = (-mu*y/rmag^3) + J2y;
    ddz = (-mu*z/rmag^3) + J2z;

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

end

function dstate = VoP(time, state, mu, mass, area, Cd)
    Re = 6378;
    wE = [0, 0, 72.9211e-6]; %rad/sec

    h = state(1);
    ecc = state(2);
    raan = state(3);
    inc = state(4);
    w = state(5);
    theta = state(6);

    [r, v] = coes2rv(h,ecc,inc,raan,w,theta,mu);
    x = r(1);
    y = r(2);
    z = r(3);
    dx = v(1);
    dy = v(2);
    dz = v(3);


    % Magnitude of position vector
    r = norm([x, y, z]);
    r_vec = [x, y, z];
    v_vec = [dx, dy, dz];
    v = norm([dx, dy, dz]);

    vrel = (v_vec - cross(wE, r_vec))*1000;
    vrel_norm = norm(vrel);
    alt = r - Re;
    rho = ExponDensModData(alt); %kg/m^3
    ap = -0.5 * Cd * area/mass * rho * (vrel_norm^2) * (vrel / vrel_norm);
    ap = ap/1000; %m/s^2 to km/s^2
    % Unit vectors
    r_hat = r_vec / norm(r_vec);
    h_vec = cross(r_vec, v_vec);
    h_hat = h_vec / norm(h_vec);
    s_hat = cross(h_hat, r_hat);

    % Project perturbing acceleration
    pr = dot(ap, r_hat);
    ps = dot(ap, s_hat);
    pw = dot(ap, h_hat);

    dhdt     = r * ps;

    dedt     = -(h/mu) * sin(theta) * pr ...
               + (( (h^2 + mu*r) * cos(theta) + mu*ecc*r ) / (mu*h)) * ps;

    dthetadt = h/r^2 + (1/(ecc*h)) * ( (h^2/mu) * cos(theta) * pr ...
                                     - (r + h^2/mu) * sin(theta) * ps );

    draandt = (r / (h * sin(inc))) * sin(w + theta) * pw;

    dincdt   = (r / h) * cos(w + theta) * pw;

    dwdt     = -(1/(ecc*h)) * ( (h^2/mu) * cos(theta) * pr ...
                              - (r + h^2/mu) * sin(theta) * ps ) ...
               - (r * sin(w + theta) / (h * tan(inc))) * pw;


    % Return derivatives of state
    dstate = [dhdt;
    dedt;
    draandt;
    dincdt;
    dwdt;
    dthetadt];
end


function [time, dstate, Rencke, Vencke] = encke(R0,V0,dt,t,Cd,mass,area)
    Re = 6378;
    mu = 398600;
    wE = [0; 0; 7.2921159e-5]; %rotation of the earth
    rt = R0; %setting true condition to initial condition for first run
    vt = V0;
    Rosc = rt;
    Vosc = vt;

    dr = [0; 0; 0];
    dv = [0; 0; 0];
    dr0 = dr;
    dv0 = dv;

    i = 1;
    while (norm(rt(1:3)) - Re) >= 100

        vrel = (vt - cross(wE, rt))*1000;
        vrel_norm = norm(vrel);
        h = norm(rt(1:3)) - Re;
        rho = ExponDensModData(h); %*1e9; %1e9 to kg/km^3
        ap = -0.5 * Cd * area/mass * rho * (vrel_norm^2) * (vrel /
vrel_norm);
        ap = ap/1000;

        % q = (dot(dr, ((2*rt) - dr)))/norm(rt)^2;
        % Fq = ( ((q^2)-(3*q)+3)/(1+(1-q)^(3/2)) ) * q;

        Fq = (norm(Rosc)/norm(rt))^3 - 1;

        da = ap - (mu/norm(Rosc)^3) * (dr - Fq*rt);
        dv = da*dt; % + dv0;
        dr = (0.5*da*dt^2); %+dv*dt; %(0.5*da*dt^2) + dv0*dt + dr;
        % dv0 = dv;
        % dr0 = dr;

        [Rosc, Vosc] = UniVarRV(rt,vt,dt,mu);

        rt = Rosc + dr;
        vt = Vosc + dv;

        dr = [0; 0; 0]; %rectify
        dv = [0; 0; 0];

        Rencke(i,1:3) = rt';
        Vencke(i,1:3) = vt';

        if norm(rt) < (Re + 100)
            break
        end

        time(i) = dt*i;
        i = i + 1;

        Rosc = rt;
        Vosc = vt;

        %disp(norm(h))

        if dt*i >= t
            error('Propogation time not long enough / no deorbit found')
        end

    end

    dstate = [Rencke,Vencke];
    time(i) = dt*i;

end
```

```
function dstate = cowell(time,state,mu,mass,area,Cd) %dstate is derivitve of
state
%FUNCTION put in descrip    %tspan state options rest
    wE = [0; 0; 7.2921159e-5]; %this is earth's angular velocity of the earth
    %define vars
    x = state(1);
    y = state(2);
    z = state(3);
    dx = state(4); %vel
    dy = state(5); %vel
    dz = state(6); %vel

    %mag of pos vector
    r = norm([x y z]);
    rvec = [x;y;z];
    vvec = [dx;dy;dz];

    % Radius and altitude (km)
    Re = 6378; % km
    h  = r - Re; % geometric altitude above mean radius

    % Relative velocity wrt atmosphere (km/s)
    vrel = vvec - cross(wE, rvec);
    vrel_norm = norm(vrel);



    %Vrel = Vsc - cross(wE,Rsc);
    rho = ExponDensModData(h)*1e9; %1e9 to kg/km^3
    %adrag = -0.5*((Cd*area)/mass)*rho*(Vrel.^2).*(Vrel/norm(Vrel));

    adrag = -0.5 * Cd * area/mass * rho * (vrel_norm^2) * (vrel / vrel_norm);

    %accel: !!eqs of motion!!
    ddx = -mu*x/r^3 + adrag(1);
    ddy = -mu*y/r^3 + adrag(2);
    ddz = -mu*z/r^3 + adrag(3);

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

end

function [value,isterminal,direction] = reentryEvent(t,state,varargin)
    Re = 6378; % km
    h = norm(state(1:3)) - Re; % km
    value = h - 100; % stop when h = 100 km
    isterminal = 1; % terminate integration
    direction = -1; % detect decreasing through zero
end

function [value,isterminal,direction] = reentryEventCoes(t,state,varargin)
    Re = 6378; % km
    mu = 398600;
    h = state(1);
    ecc = state(2);
    raan = state(3);
    inc = state(4);
    w = state(5);
    theta = state(6);

    [r] = coes2rv(h,ecc,inc,raan,w,theta,mu);
    h = norm(r) - Re; % km
    value = h - 100; % stop when h = 100 km
    isterminal = 1; % terminate integration
    direction = -1; % detect decreasing through zero
end

% Exp Density Calculation
function density = ExponDensModData(z)

% Courtesey of Dr. A
% ATMOSPHERE calculates density for altitudes from sea level
% through 1000 km using exponential interpolation.
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

%...Geometric altitudes (km):
h = ...
[  0  25  30  40  50  60   70 ...
  80  90 100 110 120 130  140 ...
 150 180 200 250 300 350  400 ...
 450 500 600 700 800 900 1000];

%...Corresponding densities (kg/m^3) from USSA76:
r = ...
[1.225     3.899e-2  1.774e-2  3.972e-3  1.057e-3  3.206e-4  8.770e-5 ...
 1.905e-5  3.396e-6  5.297e-7  9.661e-8  2.438e-8  8.484e-9  3.845e-9 ...
 2.070e-9  5.464e-10 2.789e-10 7.248e-11 1.916e-11 9.518e-12 3.725e-12 ...
 1.585e-12 6.967e-13 1.454e-13 3.614e-14 1.170e-14 5.245e-15 3.019e-15];

%...Scale heights (km):
H = ...
[ 7.249  6.349  6.682   7.554   8.382   7.714   6.549 ...
  5.799  5.382  5.877   7.263   9.473  12.636  16.149 ...
 22.523 29.740 37.105  45.546  53.628  58.515  60.828 ...
 63.822 71.835 88.667 124.64 181.05 268.00];

%...Handle altitudes outside of the range:
if z > 1000
    z = 1000;
elseif z < 0
    z = 0;
end

%...Determine the interpolation interval:
for j = 1:27
    if z >= h(j) && z < h(j+1)
        i = j;
```

```
    end
end
if z == 1000
    i = 27;
end

%...Exponential interpolation:
density = r(i)*exp(-(z - h(i))/H(i));

end  %atmopshere

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 [EtoP, PtoE] = ECI2PERI(omega,inc,RAAN,unit)
    %ECI2PERI Earth Centered Inertial Frame to Perifocal Frame
    %   [EtoP, PtoE] = ECI2PERI(omega,inc,RAAN,unit)
    %   for unit enter either 'r' or 'd'
    % EtoP = inv(rotz(omega))*inv(rotx(inc))*inv(rotz(RAAN));
    % PtoE = inv(EtoP);

    if nargin == 3
        unit = 'd';
    end

    if strcmp(unit,'d') == 1
        Z = [cosd(omega),sind(omega),0;...
            -sind(omega),cosd(omega),0;...
            0,0,1];
        X = [1,0,0; ...
            0,cosd(inc), sind(inc);...
            0, -sind(inc), cosd(inc)];
        Z2 = [cosd(RAAN),sind(RAAN),0;...
            -sind(RAAN),cosd(RAAN),0;...
            0,0,1];
    elseif strcmp(unit,'r') == 1
```

```
        Z = [cos(omega),sin(omega),0;...
            -sin(omega),cos(omega),0;...
            0,0,1];
        X = [1,0,0; ...
            0,cos(inc), sin(inc);...
            0, -sin(inc), cos(inc)];
        Z2 = [cos(RAAN),sin(RAAN),0;...
            -sin(RAAN),cos(RAAN),0;...
            0,0,1];
    end

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


% 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

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

function [C, S] = stumpff(z)
    % STUMPFF  Computes the Stumpff functions C(z) and S(z)
    %
    %   [C, S] = stumpff(z)
    %
    %   Inputs:
    %       z : scalar or array input
    %
    %   Outputs:
    %       C : Stumpff C(z)
    %       S : Stumpff S(z)

    if z > 1e-8
        s = sqrt(z);
        C = (1 - cos(s))/z;
        S = (s - sin(s))/(s^3);

    elseif z < -1e-8
        s = sqrt(-z);
        C = (cosh(s) - 1)/(-z);
        S = (sinh(s) - s)/(s^3);

    else
        % Series expansion around z = 0
        C = 1/2 - z/24 + z^2/720 - z^3/40320;
        S = 1/6 - z/120 + z^2/5040 - z^3/362880;
    end

end

function [R1,V1,RT,VT] = coes2rv(h,ecc,inc,RAAN,ArgP,theta,mu)
    %[R1,V1,RT,VT] = coes2rvd(a,ecc,inc,RAAN,ArgP,theta,mu)
    %COES2RVD - Converts classical orbital elements (in radians) to position
& velocity vectors.
    %
    % Inputs:
    %   specific angular momentum not a (- semi-major axis (km))
    %   ecc   - eccentricity
    %   inc   - inclination (radians)
    %   RAAN  - right ascension of ascending node (radians)
    %   ArgP  - argument of periapsis (radians)
    %   theta - true anomaly (radians)
    %   mu    - standard gravitational parameter (km^3/s^2)
    %
    % Outputs:
    %   R1, V1 - position and velocity vectors in ECI frame (3x1)
    %   RT, VT - transposed (1x3) versions of R1, V1

    % specific angular momentum
    % h = sqrt(mu * a * (1 - ecc^2));

    % position & velocity in perifocal (PQW) frame
    R = (h^2 / mu) / (1 + ecc * cos(theta)) * [cos(theta); sin(theta); 0];
    V = (mu / h) * [-sin(theta); ecc + cos(theta); 0];

    % rotation matrix from perifocal to ECI
    [~, Q] = ECI2PERI(ArgP, inc, RAAN, 'r');

    % transform to ECI frame
    R1 = Q * R;
    V1 = Q * V;

    % transposed for convenience
    RT = R1';
    VT = V1';
end
```

Published with MATLAB® R2025b