AERO 452: Spaceflight Dynamics II

HW 3

Justin Self

November 12, 2023

1: Work by hand

AERO 452 : HW #3

JUSTIN SELF
FALL 2023

1. Compare Encke's, Cowell's, VoP, and 2body (no perts) results for a geocentric object for 120 days or until the object reenters due to the presence of drag. Use Exponential Model for density. The satellite is 1 m in diameter and 100 kg in mass. Use 72.9211e-6 rad/s for the angular velocity of the earth. (Note this problem is an example from Curtis 12.1 so check there to make sure your code is working correctly at least for Cowell's.). Plot the orbital path (apogee and perigee) as well as the change from the initial element for RAAN, inc, and argument of perigee for all models. In addition, report the time difference between the types of propagation methods for all models. The initial orbital parameters are (at t0):

← STANDARD ATMOSPHERE

zp = 215 km;
za = 939 km;
raan0 = 340 degs
inc0 = 65.2 degs
omega0 = 58 degs
theta0 = 332 degs

PLOT

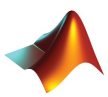
- ORBITAL PATH (a, p)
- CHANGE FROM INITIAL ELEMENT FOR:
  - i) RAAN
  - ii) inc
  - iii) ω

METHOD	RUN TIME
COWELL'S	4.06 s
ENCKE'S	1.82 s
VoP	2.09 s

REPORT

- HEART CHECKS
- COMPUTATION TIME FOR EACH METHOD (tic, toc)

* SEE MATLAB RESULTS FOR HANDWRITTEN NOTES AND ♡ CHECKS.

SEE  FOR ALL WORK.

2. Use any propagation method to determine the effect of J2 on an orbit for 48 hours. Then compare the results when you add in J3 as well. The initial orbital parameters are (at t0):
   zp = 300 km;
   za = 3092 km;
   raan0 = 45 degs
   inc0 = 28 degs
   omega0 = 30 degs
   theta0 = 40 degs

} SEE FIGURES

THE MATLAB OUTPUT PLOTS FOR THIS ARE A LITTLE HARD TO SEE. I'VE INCLUDED THEM BELOW.

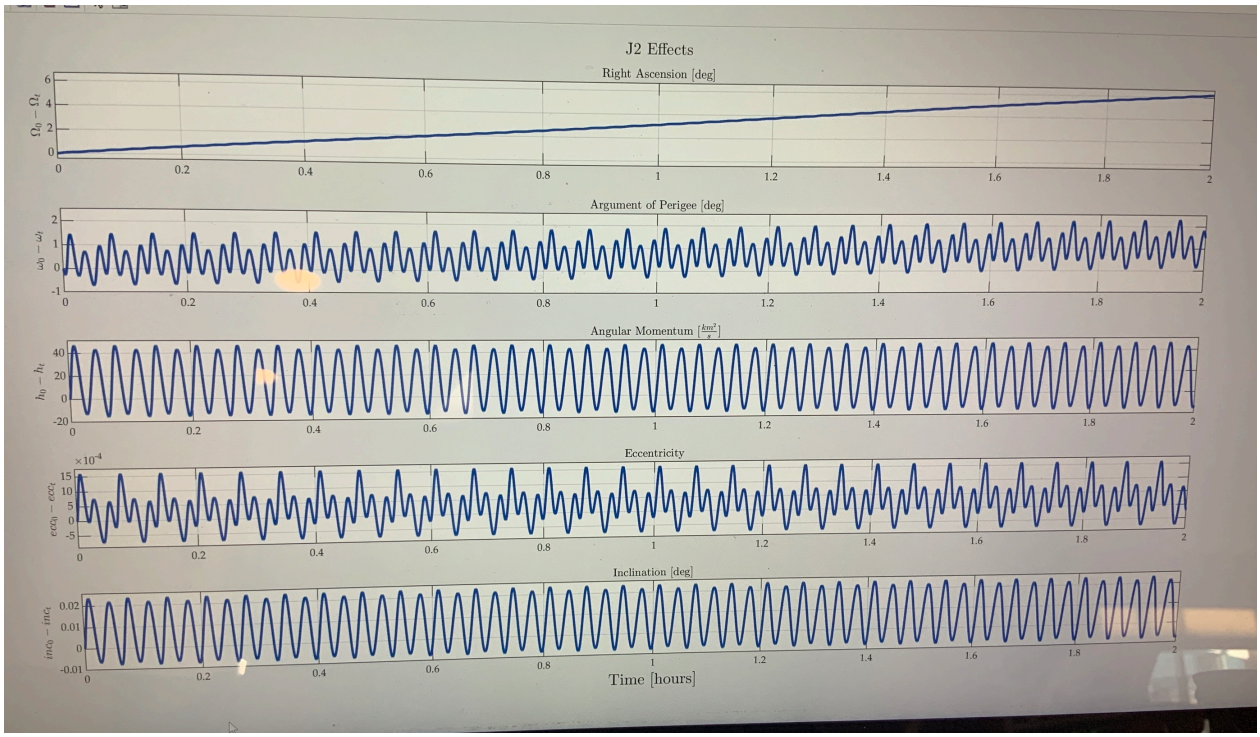


FIGURE 9 (MORE CLEAR)

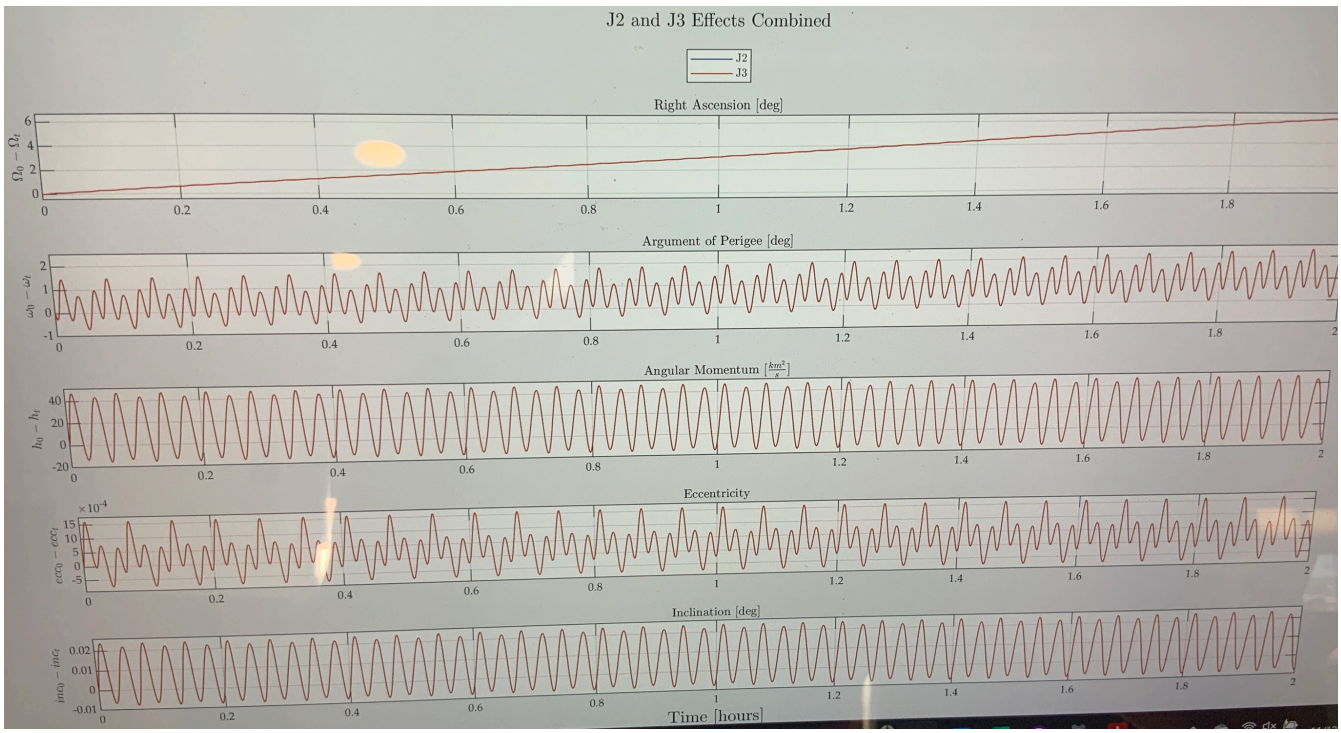


FIGURE 10 (MORE CLEAR)

SINCE THE ADDITION OF J_3 TO THE J_2 MODEL IS NEARLY IMPOSSIBLE TO SEE IN FIGURE 10, HERE IS A ZOOMED-IN VERSION.

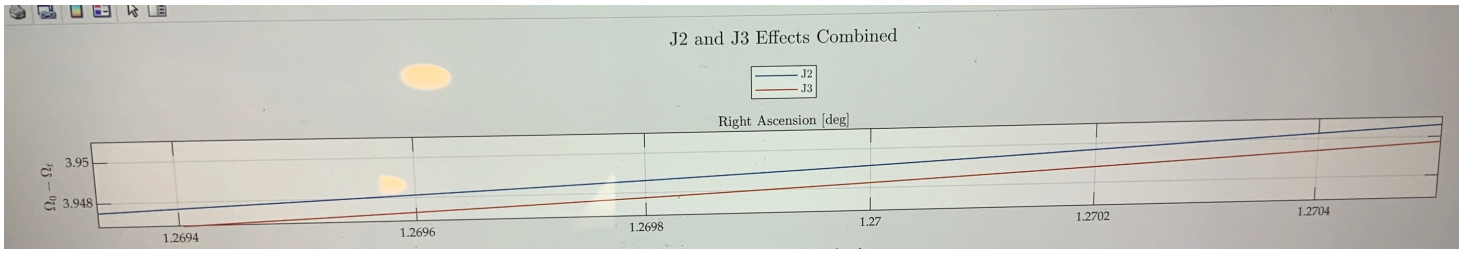


FIGURE 10 (ZOOMED IN FOR CLARITY)

♡ CHECK. $J_3 \simeq \frac{1}{1000} J_2$, SO THIS IS EXPECTED.

3. Repeat Problem 1 using the NRLMSISE drag model using only 1 of the propagation methods. Show the variations between the two models.

SEE FIGURE 13 BELOW.

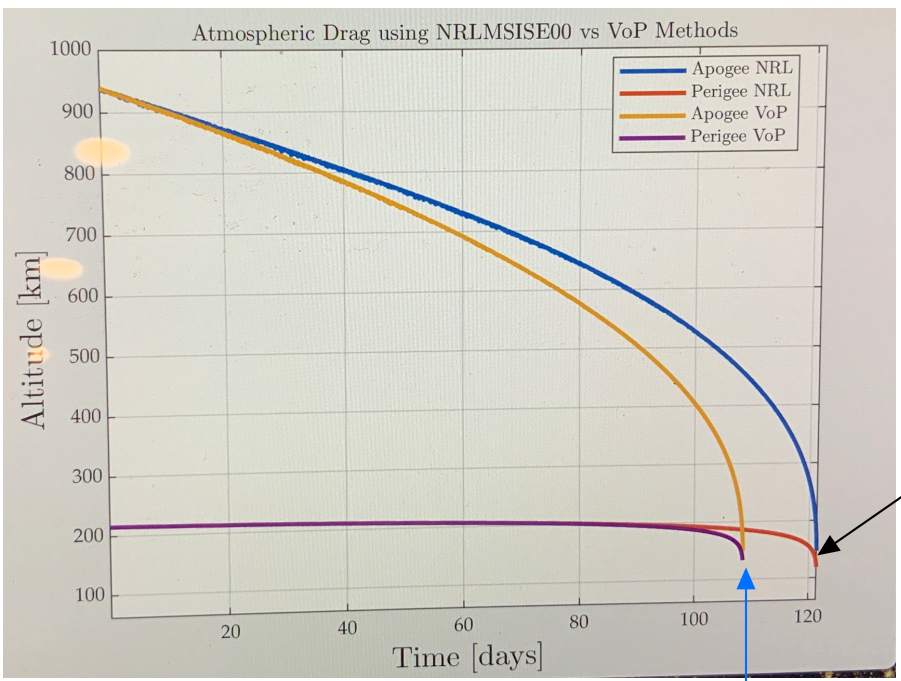


FIGURE 13.

NRLMSISE00 (USING VOP) MODEL DEORBITS IN ~120 DAYS.

VOP (ALONE) DEORBITS IN ~108 DAYS

END HW#3

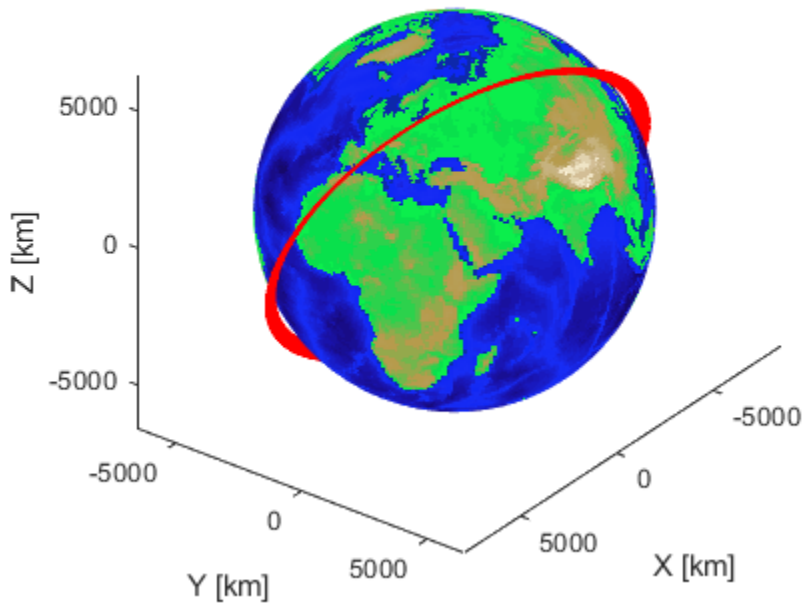
2: MATLAB RESULTS

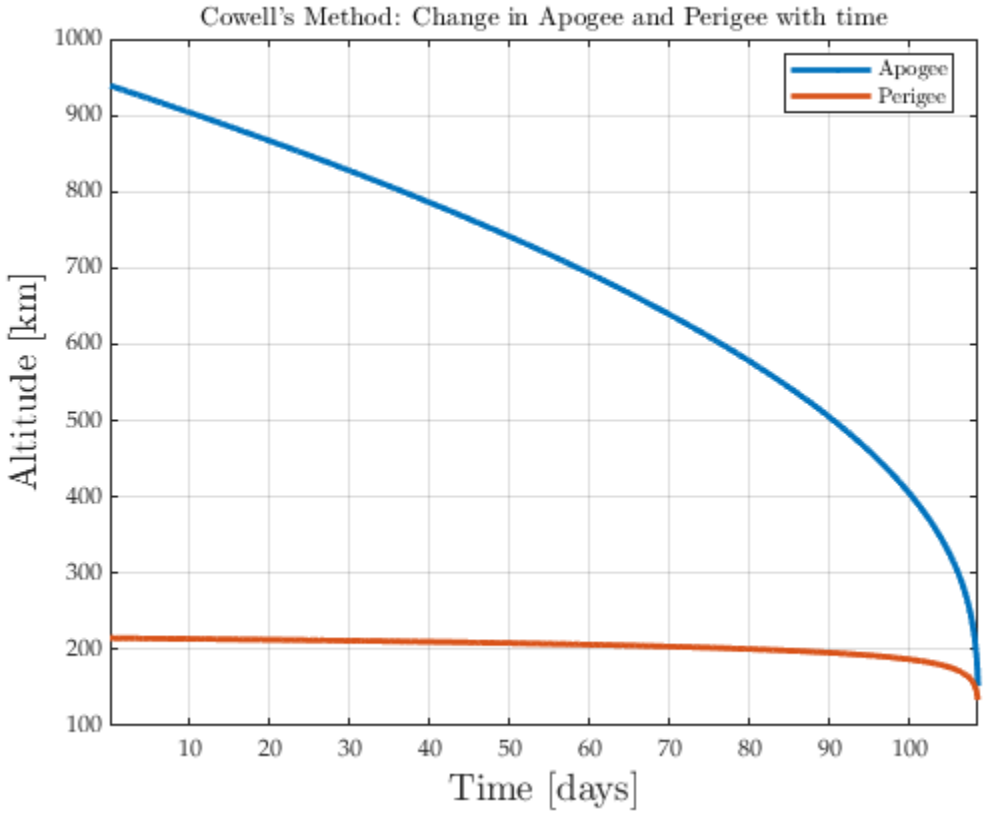
Table of Contents

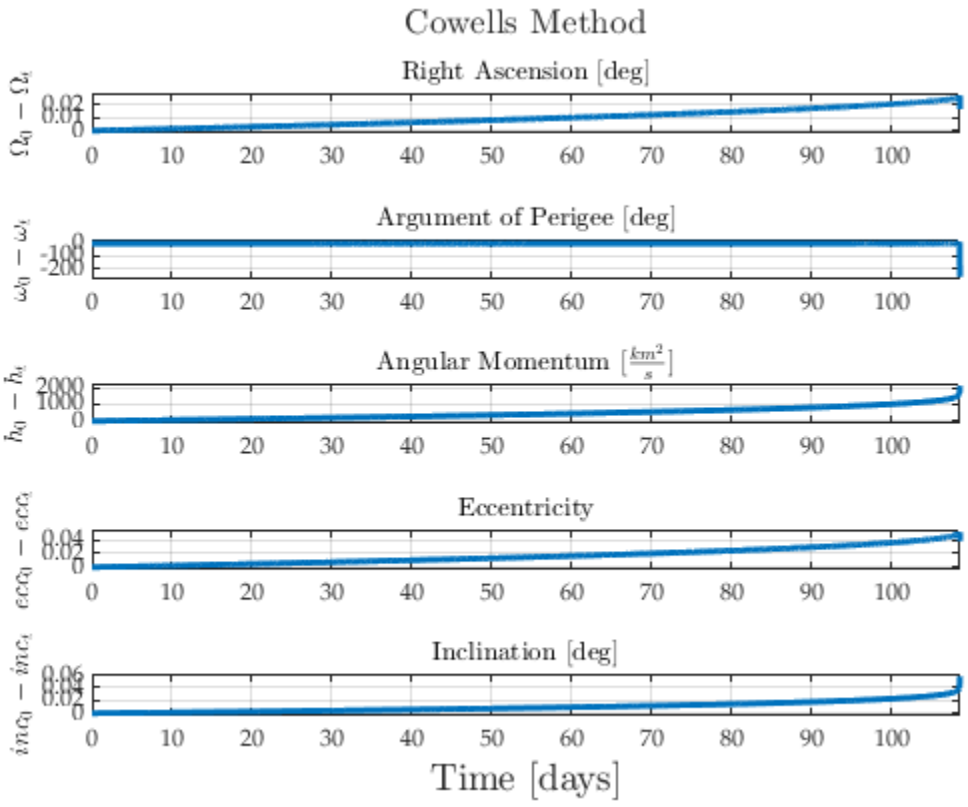
Problem #1 inputs

Cowell's Method

Cowell's Method run time is: 3.8492 seconds

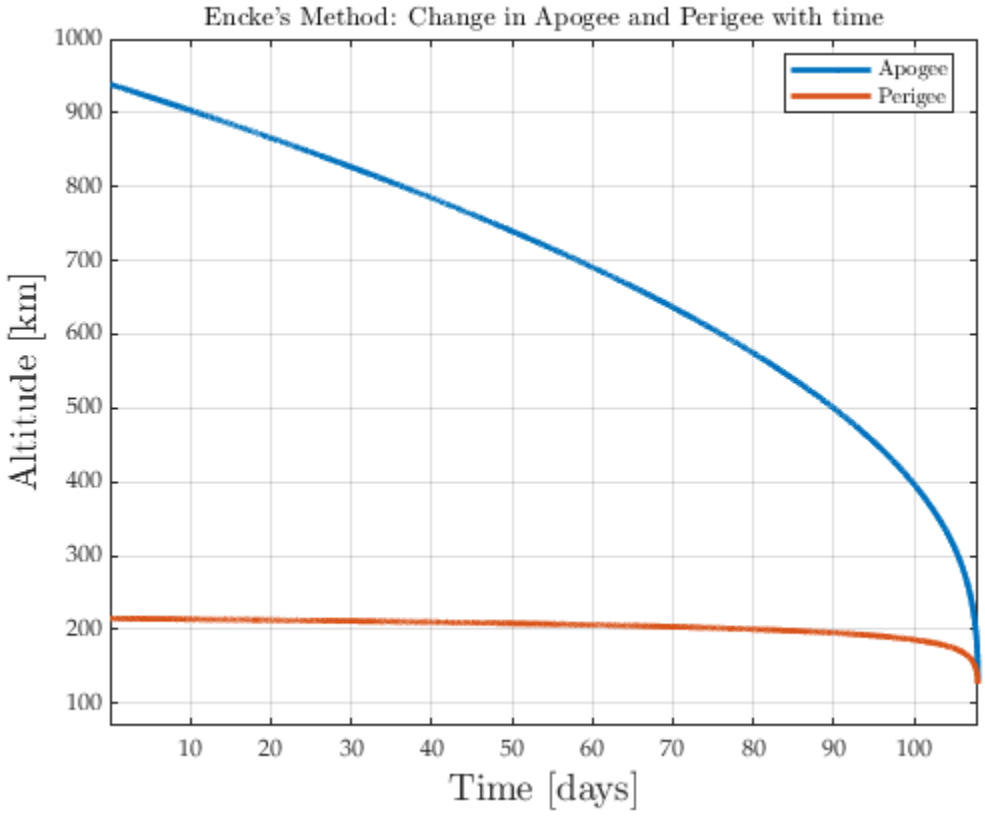


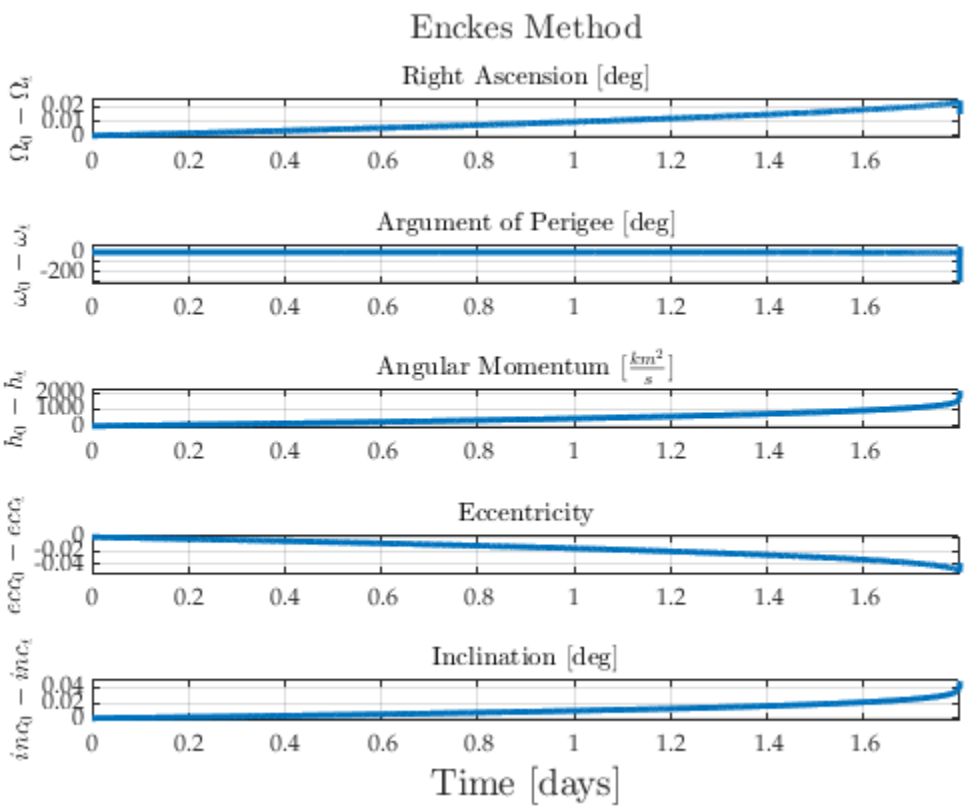

Cowell's Method: Change in Apogee and Perigee with time
Apogee
Perigee
Altitude [km]
Time [days]


Cowells Method
Right Ascension [deg]
Argument of Perigee [deg]
Angular Momentum [$\frac{km^2}{s}$]
Eccentricity
Inclination [deg]
Time [days]

Encke's method

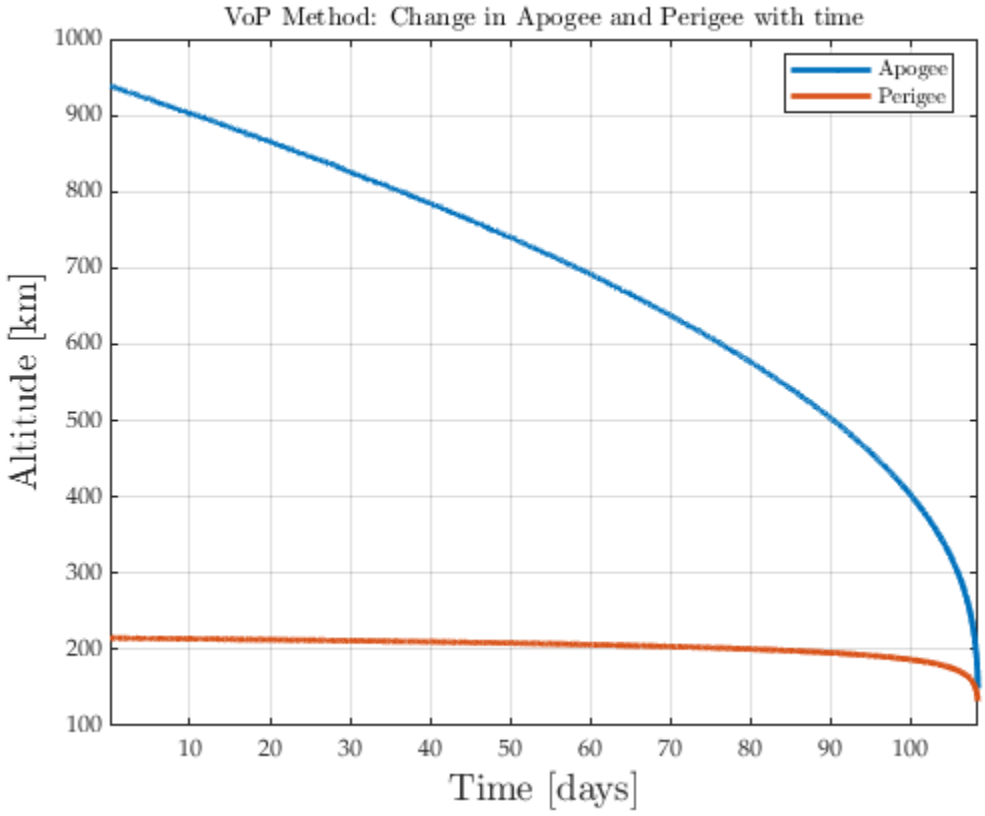
Encke's Method run time is: 3.5813 seconds

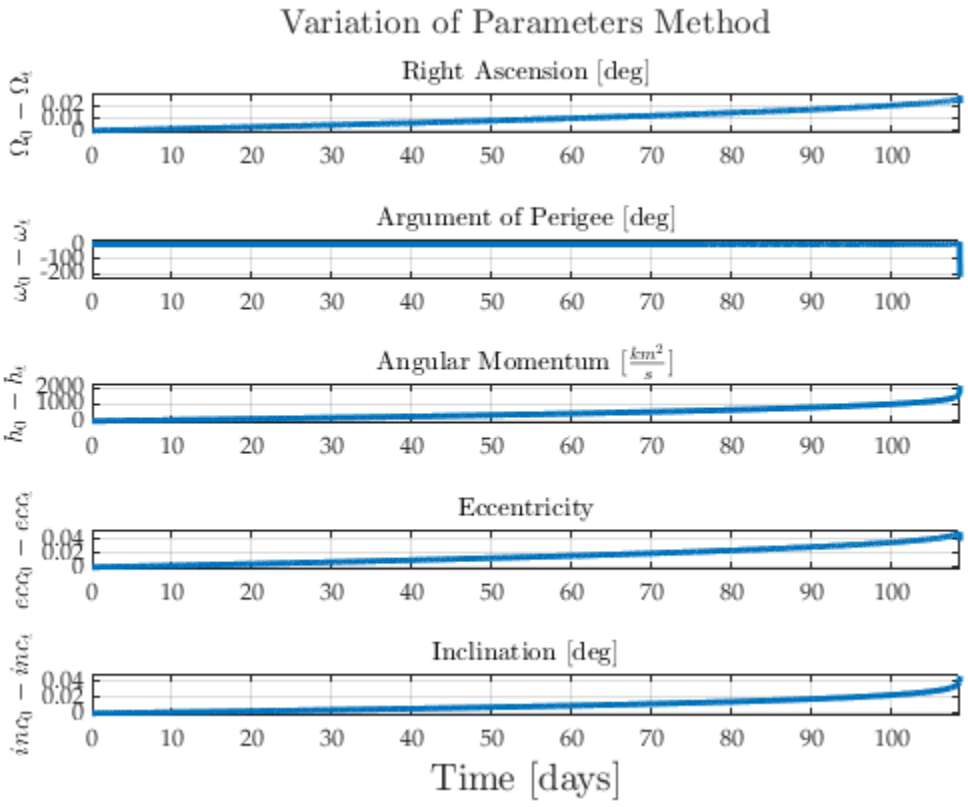




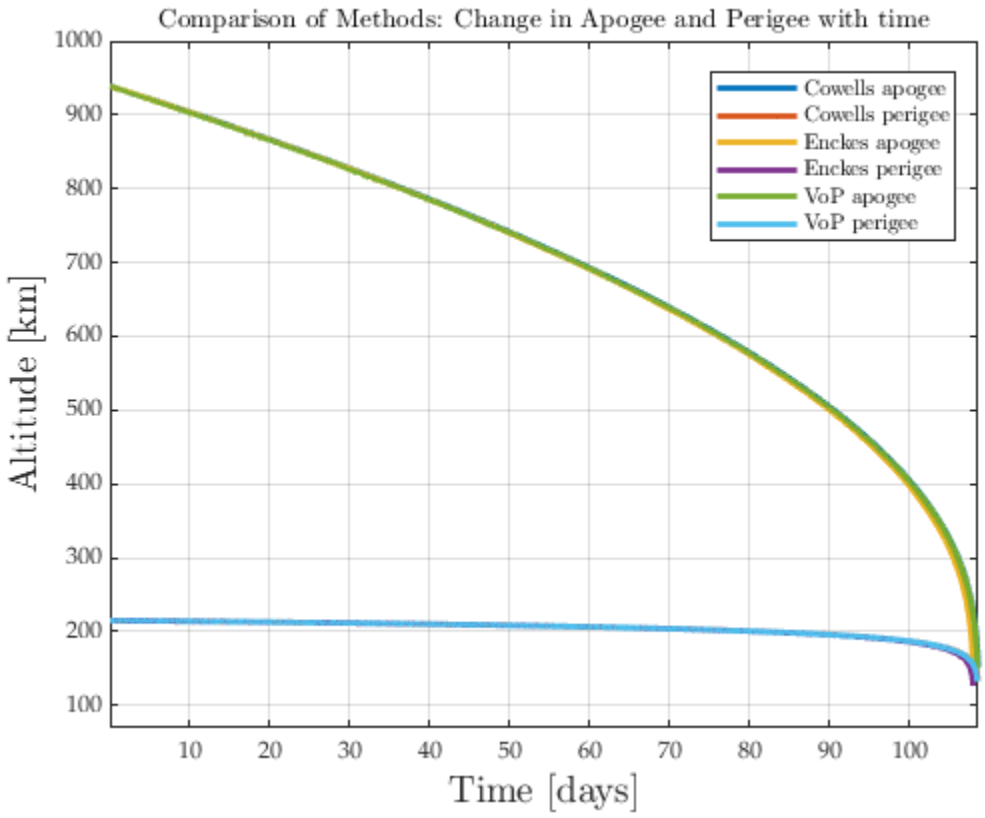
Variation of Parameters

VoP Method run time is: 2.282 seconds


VoP Method: Change in Apogee and Perigee with time
Apogee
Perigee
Altitude [km]
Time [days]

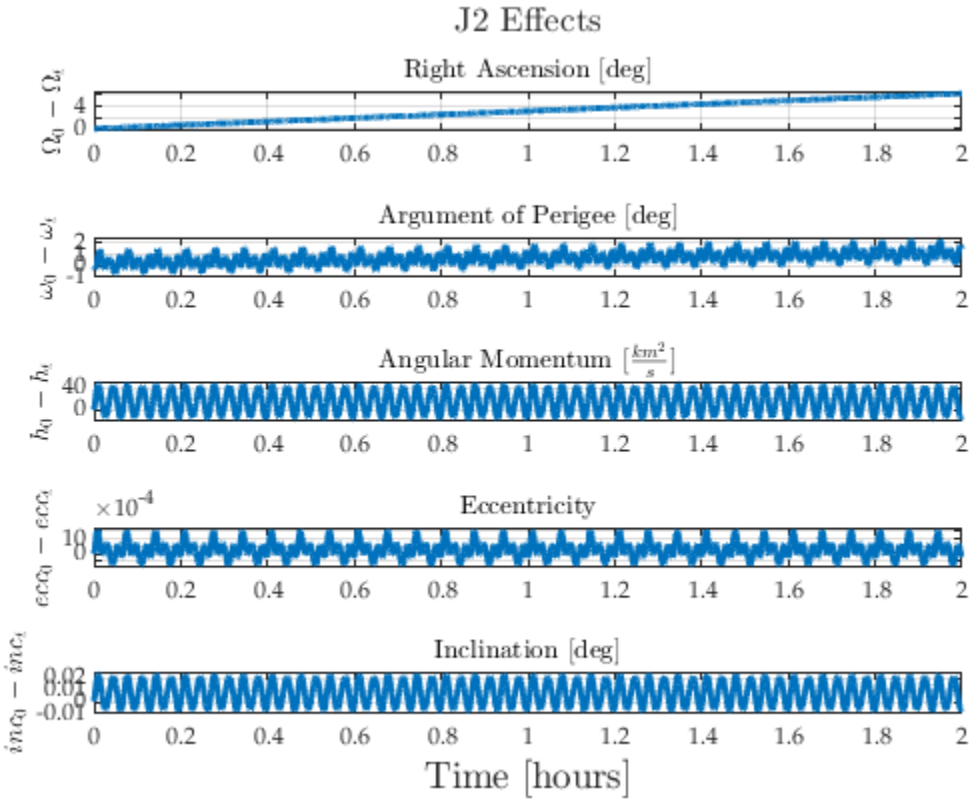

Variation of Parameters Method
Right Ascension [deg]
Argument of Perigee [deg]
Angular Momentum [$\frac{km^2}{s}$]
Eccentricity
Inclination [deg]
Time [days]

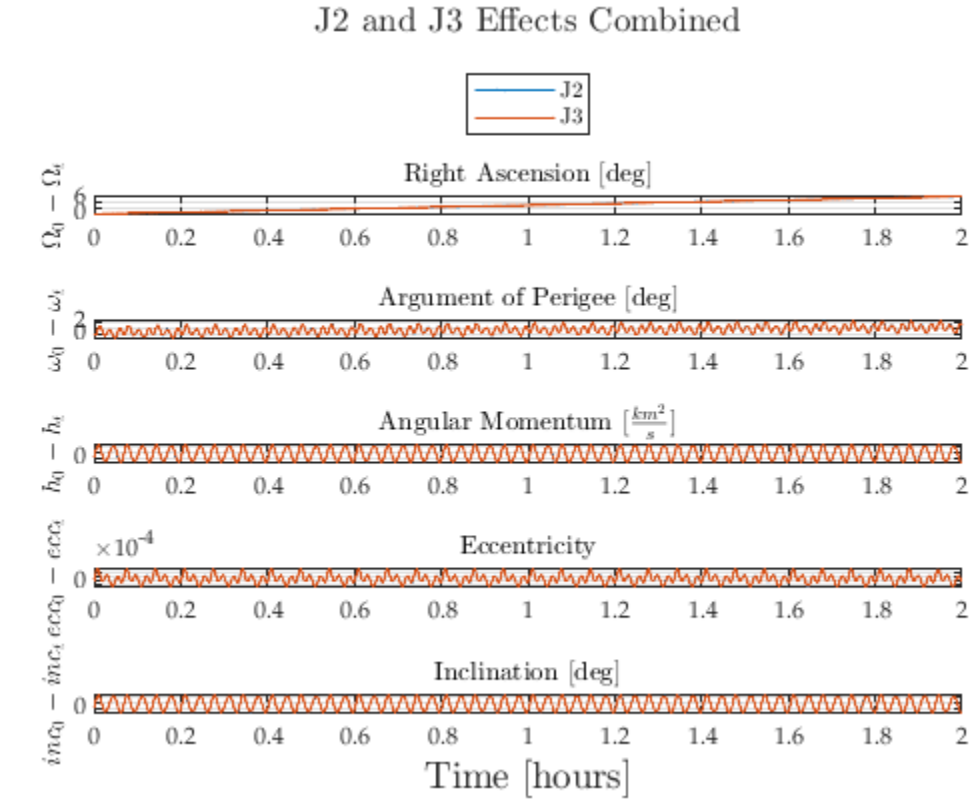
Plot all three methods for drag disturbance deorbits



J2 Using VoP

Time to run J2 using Cowell's is: 0.11767 sec
Time to run J2 with J3 using Cowell's is: 0.11767 sec
Adding J3 to J2 is barely noticable on the plots. Clearly, J2 effects are much larger than J3.

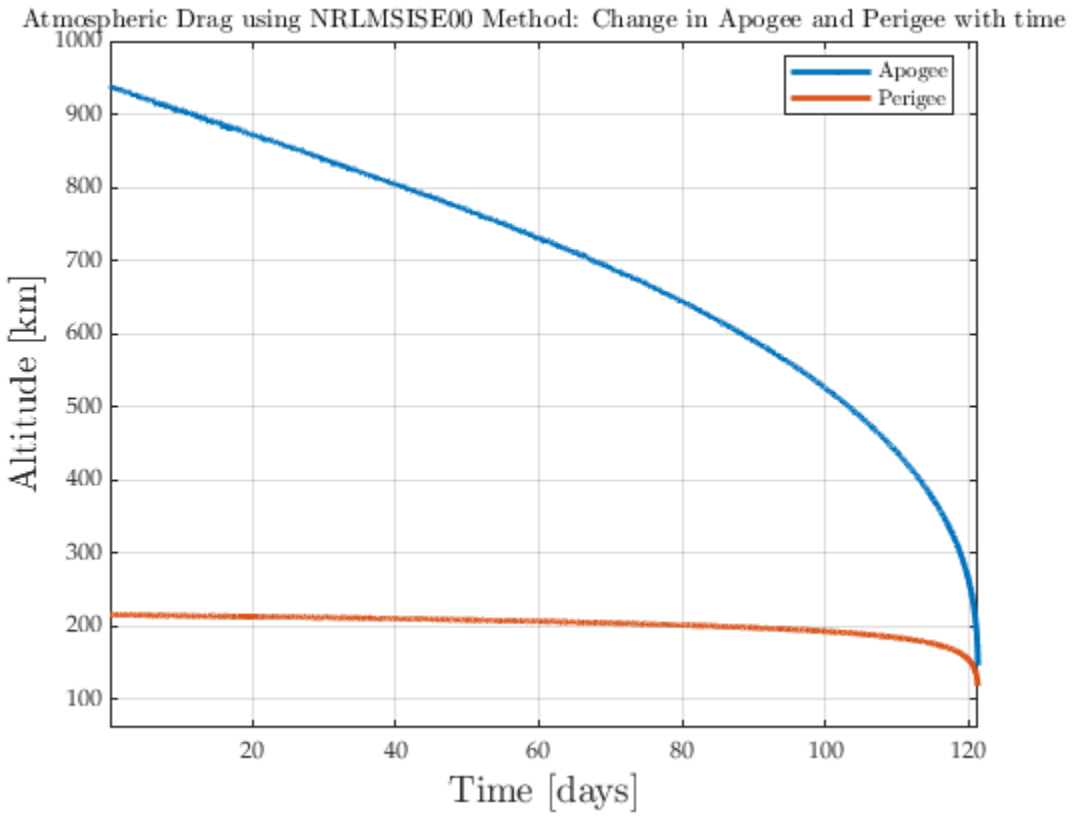

J2 Effects
Right Ascension [deg]
$\Omega_0 - \Omega_i$
Argument of Perigee [deg]
$\omega_0 - \omega_i$
Angular Momentum $[\frac{km^2}{s}]$
$h_0 - h_i$
Eccentricity
$\times 10^{-4}$
$ecc_0 - ecc_i$
Inclination [deg]
$inc_0 - inc_i$
Time [hours]

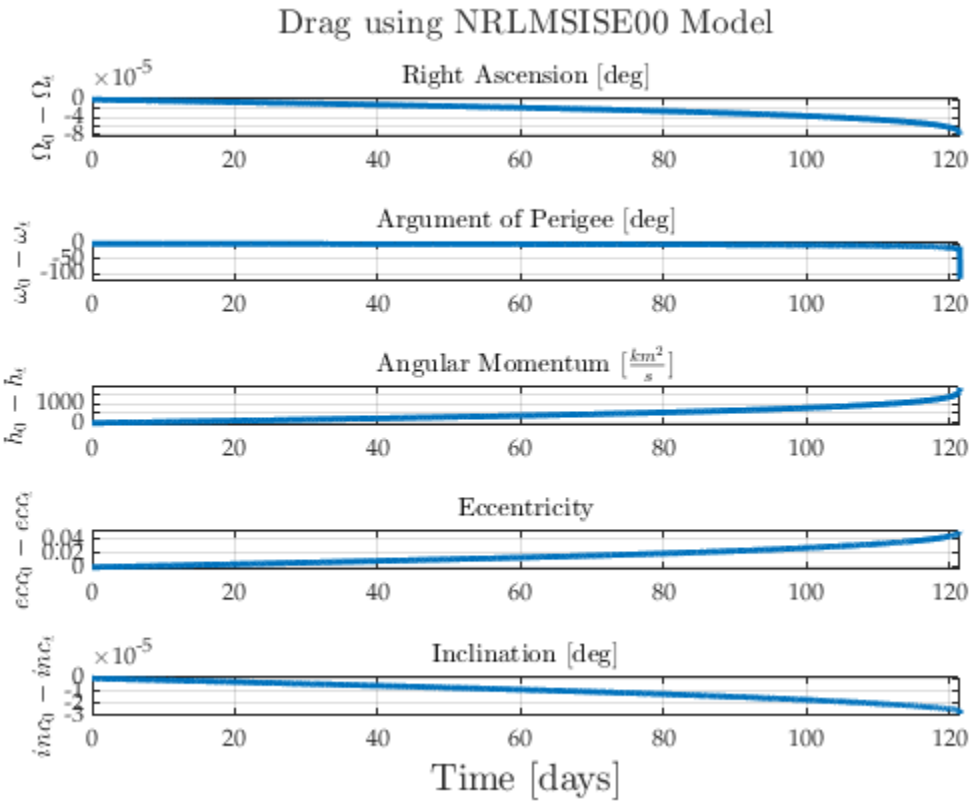

J2 and J3 Effects Combined
J2
J3
Right Ascension [deg]
$\Omega_0 - \Omega_i$
Argument of Perigee [deg]
$\omega_0 - \omega_i$
Angular Momentum $[\frac{km^2}{s}]$
$h_0 - h_i$
Eccentricity
$\times 10^{-4}$
$ecc_0 - ecc_i$
Inclination [deg]
$inc_0 - inc_i$
Time [hours]

Problem 4: NRLMSISE00

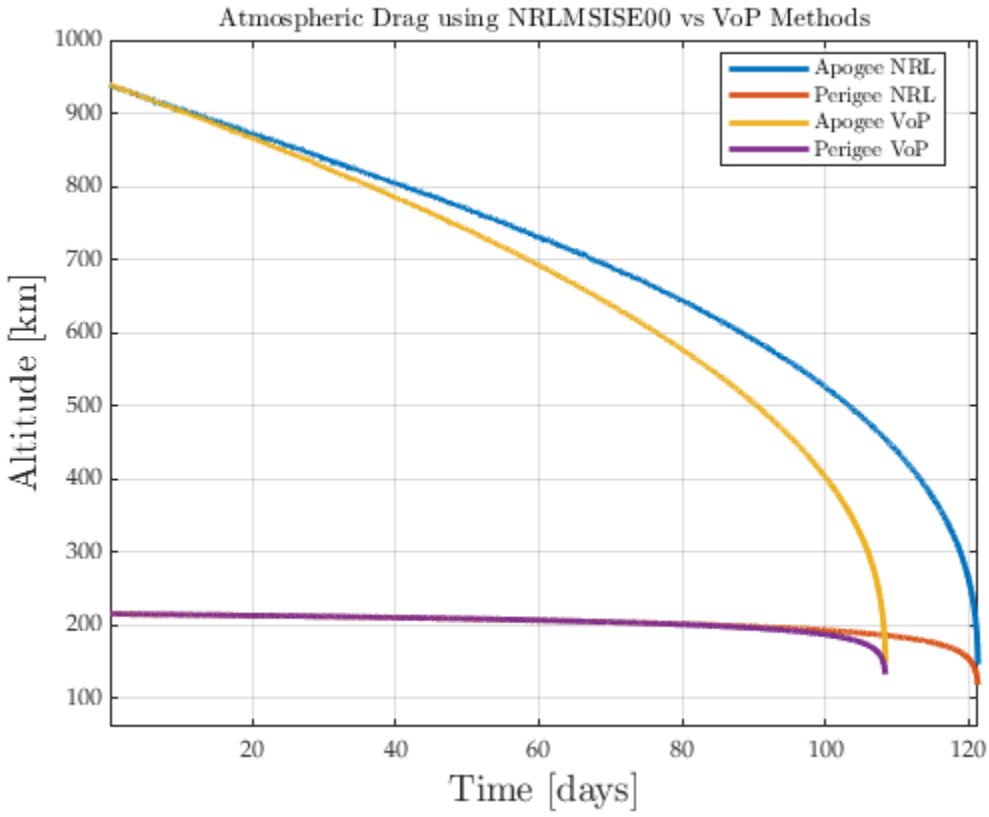
Run 150 days. Expect deorbit around 120-130 using this model.

NRLMSISE Method (using VoP) run time is: 425.2827 seconds




Drag using NRLMSISE00 Model
Right Ascension [deg]
Argument of Perigee [deg]
Angular Momentum [$\frac{km^2}{s}$]
Eccentricity
Inclination [deg]
Time [days]

Comparison Plots



Published with MATLAB® R2023b

3: MATLAB SCRIPT

Table of Contents


```
%{
AERO 452: Spaceflight Dynamics II
HW #3
Author: Justin Self
%}

% Housekeeping
clear all; close all; clc;

mu = 398600; % km3/s2
r_earth = 6378; % km
sec2days = 1 / (86400);
days2sec = 86400;
days = 120; % how long to run sim

addpath('C:/MATLAB_CODE/Orbits/')

% % Problem 1 (Curtis Ex. 10.1)
```

Problem #1 inputs

```
% Initial epoch parameters
zp = 215; % km
za = 939; % km
raan0 = 340; % deg
inc0 = 65.2; % deg
omega0 = 58; % deg
theta0 = 332; % deg

% Additional initial orbital parameters
rp = zp + r_earth; % km
ra = za + r_earth; % km
ecc0 = (ra - rp) / (ra + rp);
a0 = (ra + rp) /2; % semimajor axis
h0 = sqrt(mu*a0*(1-ecc0^2)); % angular momentum
T0 = 2*pi*sqrt(a0^3/mu); % seconds; orbital period

% Spacecraft parameters
```

```
wE = [0 0 72.9211e-6]; % earth angular velocity; rad/sec
m = 100; % s/c mass; kg
d = 1/(1000); % s/c diameter; km now
A = pi*(d/2)^2; % km2
Cd = 2.2;

% Find r,v state vectors from COEs.
[rECI,vECI] =
r_and_v_from_COEs(deg2rad(raan0),deg2rad(inc0),deg2rad(omega0),h0,ecc0,deg2ra
d(theta0));
```

Cowell's Method

```
t = days*days2sec;
tspan = [0 t];

% Run Cowell's here; comment out for other sections
options = odeset('RelTol', 1e-8, 'AbsTol',1e-8); % Make sure TOL is on order
of perts!!
options = odeset(options,'Events',@eventfun);
state = [rECI,vECI];

tstart = tic;
% Propogate 1 period TARGET
[timenew, statenew] = ode45(@perts_cowells,tspan,state,options,mu,wE,Cd,A,m);
tCowells = toc(tstart);
disp("Cowell's Method run time is: " + tCowells + " seconds")

% EARTH PLOT
figure()
   h1 = gca;
   earth_sphere(h1)
   hold on

plot3(statenew(:,1),statenew(:,2), statenew(:,3),'r')

% Find apogee/perigee
altitude = zeros(length(timenew),1);
for i = 1:length(timenew)
   altitude(i) = norm(statenew(i,1:3)) - r_earth;
end

[max_altitude,imax] = findpeaks(altitude);
[~,imin] = findpeaks(-altitude);

min_altitude = zeros(length(imin),1);
for i = 1:length(imin)
    min_altitude(i) = norm(statenew(imin(i),1:3)) - r_earth;
end

maxima = [timenew(imax) max_altitude]; % Maximum altitudes and times
minima = [timenew(imin) min_altitude]; % Min altitudes and times
```

```
% [time, state]
apogee.cowells = sortrows(maxima,1); % Maxima sorted with time
perigee.cowells = sortrows(minima,1); % Minima sorted with time

figure()
plot(apogee.cowells(:,1)*sec2days,apogee.cowells(:,2),'LineWidth',2)
hold on
plot(perigee.cowells(:,1)*sec2days,perigee.cowells(:,2),'LineWidth',2)

% Graph pretty
ylim padded
xlim tight
xLab = xlabel('Time [days]','Interpreter','latex');
yLab = ylabel('Altitude [km]','Interpreter','latex');
plotTitle = title("Cowell's Method: Change in Apogee and Perigee with
time",'interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set([xLab, yLab],'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
set([xLab, yLab],'FontSize', 14)
grid on
legend('Apogee','Perigee', 'interpreter','latex','Location', 'best')

% % Cowell's: Change in COEs
n = length(timenew); % length of state vector out of Cowell's

% Preallocate for speed
hnew = zeros(length(n),1);
incnew = hnew;
RAANnew = hnew;
eccnew = hnew;
wnew = hnew;
thetanew = hnew;

% Preallocate
r0Vect = zeros(length(n),1);
v0Vect = r0Vect;

for i = 1:n
r0Vect = statenew(i,1:3);
v0Vect = statenew(i,4:6);
[hnew(i), incnew(i), RAANnew(i), eccnew(i), wnew(i), thetanew(i), ~, ~, ~] =
rv2COEs(r0Vect,v0Vect,mu);
end

dRAAN = raan0 - RAANnew;
dinc = inc0 - incnew;
dw = omega0 - wnew;
dh = h0 - hnew;
decc = ecc0 - eccnew;
time = timenew .* sec2days;

% Plot changes in COEs with time [COWELL'S METHOD]
```

```
figure()
tiledlayout("vertical")

% Plot delta RAAN
nexttile
plot(time,dRAAN,'Linewidth',2)
% Graph pretty
ylim padded
xlim tight
yLab = ylabel('$\Omega_0 - \Omega_t$','Interpreter','latex');
plotTitle = title('Right Ascension [deg]','interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set(yLab,'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
grid on

% Plot delta omega
nexttile
plot(time,dw,'Linewidth',2)
% Graph pretty
ylim padded
xlim tight
yLab = ylabel('$\omega_0 - \omega_t$','Interpreter','latex');
plotTitle = title('Argument of Perigee [deg]','interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set(yLab,'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
grid on

% Plot angular momentum
nexttile
plot(time,dh,'Linewidth',2)
% Graph pretty
ylim padded
xlim tight
yLab = ylabel('$h_0 - h_t$','Interpreter','latex');
plotTitle = title('Angular Momentum [$\frac{km^2}{s}
$]','interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set(yLab,'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
grid on

% Plot eccentricity
nexttile
plot(time,decc,'Linewidth',2)
% Graph pretty
ylim padded
xlim tight
yLab = ylabel('$ecc_0 - ecc_t$','Interpreter','latex');
plotTitle = title('Eccentricity','interpreter','latex');
```

```
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set(yLab,'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
grid on

% Plot delta inclination
nexttile
plot(time,dinc,'Linewidth',2)
% Graph pretty
ylim padded
xlim tight
xLab = xlabel('Time [days]','Interpreter','latex');
yLab = ylabel('$inc_0 - inc_t$','Interpreter','latex');
plotTitle = title('Inclination [deg]','interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set(yLab,'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
set(xLab,'FontSize', 14)
grid on
% Overall title
sgtitle('Cowells Method','interpreter','latex')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Encke's method

```
% Algorithm (KJ Abercromby)

% Start timer
tstart = tic;

% Step 1
dr = [0;0;0];
dv = dr;
dt = 60; % rect every minute
t = 120*days2sec;

% Define R, V osculating (grav only)
Rosc = rECI;
Vosc = vECI;

% Step 2: Step forward in delta t
% Preallocate
R = zeros(3,1);
V = R;

for i = 1:t/dt % 1/60 of full time vect; 60-sec rectifications.
r = norm(Rosc);
Vrel = Vosc - cross(wE',Rosc); % relative velocity wrt rotating atmosphere
```

```
vrel = norm(Vrel);
z = r - r_earth;

if z <100
    break
end

% altitude
rho = atmosphere(z);                                    % in kg/m3
rho = rho*1000^3;

Ap = -(1/2) * rho*vrel * (Cd*A/m) .* Vrel;              % drag

Fq = 0;

da = Ap - (mu./Rosc.^3) .* (dr - Fq.*Rosc);
dv = da * dt + dv;
dr = (1/2) .* da .* dt^2 + dv .* dt + dr;

% Step 3: Universal variable
TOL = 1e-8;
[Rosc,Vosc] = universalVar(mu,dt,Rosc,Vosc,TOL);

% Step 4: If choice = rectify; go back to step one; else go to step 5
R(:,i) = Rosc + dr;
V(:,i) = Vosc + dv;

% Go back to Step 1: (reset Dr,Dv)
dr = [0;0;0];
dv = [0;0;0];
da = [0;0;0];

Rosc = R(:,i);
Vosc = V(:,i);

end
R = R';
V = V';

% optional: add if statment to check for dr/r > 0.01 (maybe at the
% beginning)

% End timer
tEnckes = toc(tstart);
disp("Encke's Method run time is: " + tEnckes + " seconds")

% Plot Encke's Method results
% Find apogee/perigee
sec2days = 1 / (86400);
altitude = zeros(length(R),1);

for i = 1:length(R)
    altitude(i) = norm(R(i,1:3)) - r_earth;
end
```

```
[max_altitude,imax] = findpeaks(altitude);
[x,imin] = findpeaks(-altitude);

min_altitude = zeros(length(imin),1);
for i = 1:length(imin)
    min_altitude(i) = norm(R(imin(i),1:3)) - r_earth;
end
time = linspace(0,length(R),length(R))';
% Find maxima; minima
maxima = [time(imax) max_altitude]; % Maximum altitudes and times
minima = [time(imin) min_altitude]; % Min altitudes and times

% Sort
apogee.enckes = sortrows(maxima,1); % Maxima sorted with time
perigee.enckes = sortrows(minima,1); % Minima sorted with time

figure()
plot(apogee.enckes(:,1)*60*sec2days,apogee.enckes(:,2),'LineWidth',2)
hold on
plot(perigee.enckes(:,1)*60*sec2days,perigee.enckes(:,2),'LineWidth',2)

% Graph pretty
ylim padded
xlim tight
xLab = xlabel('Time [days]','Interpreter','latex');
yLab = ylabel('Altitude [km]','Interpreter','latex');
plotTitle = title("Encke's Method: Change in Apogee and Perigee with
time",'interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set([xLab, yLab],'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
set([xLab, yLab],'FontSize', 14)
grid on
legend('Apogee','Perigee', 'interpreter','latex','Location', 'best')

% % Change in COEs for Encke's Method

n = length(R); % length of time vector over deorbit time for Encke's

% Preallocate for speed
hnew = zeros(length(n),1);
incnew = hnew;
RAANnew = hnew;
eccnew = hnew;
wnew = hnew;
thetanew = hnew;

for i = 1:n
r0Vect = R(i,1:3);
v0Vect = V(i,1:3);
[hnew(i), incnew(i), RAANnew(i), eccnew(i), wnew(i), thetanew(i), ~, ~, ~] =
rv2COEs(r0Vect,v0Vect,mu);
```

```
end

dRAAN = raan0 - RAANnew;
dinc = inc0 - incnew;
dw = omega0 - wnew;
dh = h0 - hnew;
decc = eccnew - ecc0;
time = time .* sec2days;

% Plot changes in COEs with time [ENCKE'S METHOD]
figure()
tiledlayout("vertical")

% Plot delta RAAN
nexttile
plot(time,dRAAN,'Linewidth',2)
% Graph pretty
ylim padded
xlim tight
yLab = ylabel('$\Omega_0 - \Omega_t$','Interpreter','latex');
plotTitle = title('Right Ascension [deg]','interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set(yLab,'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
grid on

% Plot delta omega
nexttile
plot(time,dw,'Linewidth',2)
% Graph pretty
ylim padded
xlim tight
yLab = ylabel('$\omega_0 - \omega_t$','Interpreter','latex');
plotTitle = title('Argument of Perigee [deg]','interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set(yLab,'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
grid on

% Plot angular momentum
nexttile
plot(time,dh,'Linewidth',2)
% Graph pretty
ylim padded
xlim tight
yLab = ylabel('$h_0 - h_t$','Interpreter','latex');
plotTitle = title('Angular Momentum [$\frac{km^2}{s}
$]','interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set(yLab,'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
```

```
grid on

% Plot eccentricity
nexttile
plot(time,decc,'Linewidth',2)
% Graph pretty
ylim padded
xlim tight
yLab = ylabel('$ecc_0 - ecc_t$','Interpreter','latex');
plotTitle = title('Eccentricity','interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set(yLab,'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
grid on

% Plot delta inclination
nexttile
plot(time,dinc,'Linewidth',2)
% Graph pretty
ylim padded
xlim tight
xLab = xlabel('Time [days]','Interpreter','latex');
yLab = ylabel('$inc_0 - inc_t$','Interpreter','latex');
plotTitle = title('Inclination [deg]','interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set(yLab,'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
set(xLab,'FontSize', 14)
grid on

% Overall title
sgtitle('Enckes Method','interpreter','latex')


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Variation of Parameters

```
% Define COEs vector (in RADIANS)
state =
[deg2rad(raan0);deg2rad(inc0);deg2rad(omega0);h0;ecc0;deg2rad(theta0)];

% Prepare to call ode45
options = odeset('RelTol', 1e-9, 'AbsTol',1e-9); % Make sure TOL is on order
of perts!!
options = odeset(options,'Events',@eventfun_VOP);
tspan = [0 t];

% Time and call function
```

```
tstart = tic;
[timenew, COEsnew] =
ode45(@VariationOfParameters,tspan,state,options,mu,wE,Cd,A,m);
tVoP = toc(tstart);

% Extract outputs (RADIANS RADIANS RADIANS)
raanNew = COEsnew(:,1);
incNew = COEsnew(:,2);
wNew = COEsnew(:,3);
hNew = COEsnew(:,4);
eccNew = COEsnew(:,5);
thetaNew = COEsnew(:,6);

% Print timing
disp("VoP Method run time is: " + tVoP + " seconds")

%..........Plot apogee/perigee change with time

% Find r, v vectors. Best way I know how is to run COES2RV
r = zeros(3,length(timenew));
altitude = zeros(length(timenew),1);

for i = 1:length(timenew)
[r(:,i),~] =
r_and_v_from_COEs(raanNew(i),incNew(i),wNew(i),hNew(i),eccNew(i),thetaNew(i))
;
altitude(i) = norm(r(1:3,i)) - r_earth;
end
r = r';

[max_altitude,imax] = findpeaks(altitude);
[~,imin] = findpeaks(-altitude);

min_altitude = zeros(length(imin),1);
for i = 1:length(imin)
    min_altitude(i) = norm(r(imin(i),1:3)) - r_earth;
end

apogee.vop = [timenew(imax) max_altitude]; % Maximum altitudes and times
perigee.vop = [timenew(imin) min_altitude]; % Min altitudes and times

% [time, state]
%apogee = sortrows(maxima,1); % Maxima sorted with time
%perigee = sortrows(minima,1); % Minima sorted with time

figure()
plot(apogee.vop(:,1)*sec2days,apogee.vop(:,2),'LineWidth',2)
hold on
plot(perigee.vop(:,1)*sec2days,perigee.vop(:,2),'LineWidth',2)

% Graph pretty
ylim padded
xlim tight
xLab = xlabel('Time [days]','Interpreter','latex');
```

```
yLab = ylabel('Altitude [km]','Interpreter','latex');
plotTitle = title("VoP Method: Change in Apogee and Perigee with
time",'interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set([xLab, yLab],'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
set([xLab, yLab],'FontSize', 14)
grid on
legend('Apogee','Perigee', 'interpreter','latex','Location', 'best')

%..........Plot changes in COEs with time [Variation of Parameter Method]
% Convert back to degrees for plots
raanNew = rad2deg(raanNew);
incNew = rad2deg(incNew);
wNew = rad2deg(wNew);

dRAAN = raan0 - raanNew;
dinc = inc0 - incNew;
dw = omega0 - wNew;
dh = h0 - hNew;
decc = ecc0 - eccNew;

time = timenew .* sec2days;

figure()
tiledlayout("vertical")

% Plot delta RAAN
nexttile
plot(time,dRAAN,'LineWidth',2)
% Graph pretty
ylim padded
xlim tight
yLab = ylabel('$\Omega_0 - \Omega_t$','Interpreter','latex');
plotTitle = title('Right Ascension [deg]','interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set(yLab,'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
grid on

% Plot delta omega
nexttile
plot(time,dw,'LineWidth',2)
% Graph pretty
ylim padded
xlim tight
yLab = ylabel('$\omega_0 - \omega_t$','Interpreter','latex');
plotTitle = title('Argument of Perigee [deg]','interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set(yLab,'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
```

```
grid on

% Plot angular momentum
nexttile
plot(time,dh,'LineWidth',2)
% Graph pretty
ylim padded
xlim tight
yLab = ylabel('$h_0 - h_t$','Interpreter','latex');
plotTitle = title('Angular Momentum [$\frac{km^2}{s}
$]','interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set(yLab,'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
grid on

% Plot eccentricity
nexttile
plot(time,decc,'LineWidth',2)
% Graph pretty
ylim padded
xlim tight
yLab = ylabel('$ecc_0 - ecc_t$','Interpreter','latex');
plotTitle = title('Eccentricity','interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set(yLab,'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
grid on

% Plot delta inclination
nexttile
plot(time,dinc,'LineWidth',2)
% Graph pretty
ylim padded
xlim tight
xLab = xlabel('Time [days]','Interpreter','latex');
yLab = ylabel('$inc_0 - inc_t$','Interpreter','latex');
plotTitle = title('Inclination [deg]','interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set(yLab,'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
set(xLab,'FontSize', 14)
grid on

% Overall title
sgtitle('Variation of Parameters Method','interpreter','latex')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Plot all three methods for drag disturbance deorbits

```
figure()
% Plot Cowell's
plot(apogee.cowells(:,1)*sec2days,apogee.cowells(:,2),'LineWidth',2)
hold on
plot(perigee.cowells(:,1)*sec2days,perigee.cowells(:,2),'LineWidth',2)
% Plot Encke's
plot(apogee.enckes(:,1)*sec2days*60,apogee.enckes(:,2),'LineWidth',2)
plot(perigee.enckes(:,1)*sec2days*60,perigee.enckes(:,2),'LineWidth',2)
% Plot VoP
plot(apogee.vop(:,1)*sec2days,apogee.vop(:,2),'LineWidth',2)
plot(perigee.vop(:,1)*sec2days,perigee.vop(:,2),'LineWidth',2)

% All plots
% Graph pretty
ylim padded
xlim tight
xLab = xlabel('Time [days]','Interpreter','latex');
yLab = ylabel('Altitude [km]','Interpreter','latex');
plotTitle = title("Comparison of Methods: Change in Apogee and Perigee with
time",'interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set([xLab, yLab],'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
set([xLab, yLab],'FontSize', 14)
grid on
legend('Cowells apogee','Cowells perigee','Enckes apogee','Enckes
perigee',...
    'VoP apogee','VoP perigee','interpreter','latex','Location', 'best')



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

J2 Using VoP

```
t = 48*3600; % sec

% J2
% Define COEs vector [r;v] initial
state = [rECI,vECI];

% Prepare to call ode45
options = odeset('RelTol', 1e-8, 'AbsTol',1e-8); % Make sure TOL is on order
of perts!!
options = odeset(options,'Events',@eventfun_J2_cowells);
```

```
tspan = [0 t];

% Time and call function
tstart = tic;
[timenew, statenew] = ode45(@J2_cowells,tspan,state,options,mu);
tJ2 = toc(tstart);
disp("Time to run J2 using Cowell's is: " + tJ2 + " sec")

% Change in COEs J2 with J3
n = length(timenew); % length of state vector out of ode

% Preallocate for speed
hnew = zeros(length(n),1);
incnew = hnew;
RAANnew = hnew;
eccnew = hnew;
wnew = hnew;
thetanew = hnew;

% Preallocate
r0Vect = zeros(length(n),1);
v0Vect = r0Vect;

for i = 1:n
r0Vect = statenew(i,1:3);
v0Vect = statenew(i,4:6);
[hnew(i), incnew(i), RAANnew(i), eccnew(i), wnew(i), thetanew(i), ~, ~, ~] =
rv2COEs(r0Vect,v0Vect,mu);
end

dRAAN = raan0 - RAANnew;
dinc = inc0 - incnew;
dw = omega0 - wnew;
dh = h0 - hnew;
decc = ecc0 - eccnew;
time = timenew .* sec2days;

figure()
tiledlayout("vertical")

% Plot delta RAAN
nexttile
plot(time,dRAAN,'LineWidth',2)
% Graph pretty
ylim padded
xlim tight
yLab = ylabel('$\Omega_0 - \Omega_t$','Interpreter','latex');
plotTitle = title('Right Ascension [deg]','interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set(yLab,'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
grid on
```

```
% Plot delta omega
nexttile
plot(time,dw,'LineWidth',2)
% Graph pretty
ylim padded
xlim tight
yLab = ylabel('$\omega_0 - \omega_t$','Interpreter','latex');
plotTitle = title('Argument of Perigee [deg]','interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set(yLab,'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
grid on

% Plot angular momentum
nexttile
plot(time,dh,'LineWidth',2)
% Graph pretty
ylim padded
xlim tight
yLab = ylabel('$h_0 - h_t$','Interpreter','latex');
plotTitle = title('Angular Momentum [$\frac{km^2}{s}
$]','interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set(yLab,'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
grid on

% Plot eccentricity
nexttile
plot(time,decc,'LineWidth',2)
% Graph pretty
ylim padded
xlim tight
yLab = ylabel('$ecc_0 - ecc_t$','Interpreter','latex');
plotTitle = title('Eccentricity','interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set(yLab,'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
grid on

% Plot delta inclination
nexttile
plot(time,dinc,'LineWidth',2)
% Graph pretty
ylim padded
xlim tight
xLab = xlabel('Time [hours]','Interpreter','latex');
yLab = ylabel('$inc_0 - inc_t$','Interpreter','latex');
plotTitle = title('Inclination [deg]','interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
```

```
set(yLab,'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
set(xLab,'FontSize', 14)
grid on

% Overall title
sgtitle('J2 Effects','interpreter','latex')

% Add J3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Add J3 %%%%%%%%%%%%%%%

% Prepare to call ode45
options = odeset('RelTol', 1e-8, 'AbsTol',1e-8); % Make sure TOL is on order
of perts!!
options = odeset(options,'Events',@eventfun_J2_cowells);
tspan = [0 t];

% Time and call function
tstart = tic;
[timenew, statenew] = ode45(@J2_J3_cowells,tspan,state,options,mu);
tJ2J3 = toc(tstart);
disp("Time to run J2 with J3 using Cowell's is: " + tJ2 + " sec")

% Change in COEs J2 with J3
n = length(timenew); % length of state vector out of ode

% Preallocate for speed
hnewJ3 = zeros(length(n),1);
incnewJ3 = hnewJ3;
RAANnewJ3 = hnewJ3;
eccnewJ3 = hnewJ3;
wnewJ3 = hnewJ3;
thetanewJ3 = hnewJ3;

% Preallocate
r0Vect = zeros(length(n),1);
v0Vect = r0Vect;

for i = 1:n
r0Vect = statenew(i,1:3);
v0Vect = statenew(i,4:6);
[hnewJ3(i), incnewJ3(i), RAANnewJ3(i), eccnewJ3(i), wnewJ3(i),
thetanewJ3(i), ~, ~, ~] = rv2COEs(r0Vect,v0Vect,mu);
end

dRAANJ3 = raan0 - RAANnewJ3;
dincJ3 = inc0 - incnewJ3;
dwJ3 = omega0 - wnewJ3;
dhJ3 = h0 - hnewJ3;
deccJ3 = ecc0 - eccnewJ3;
time = timenew .* sec2days;

figure()
tiledlayout("vertical")
```

```
% Plot delta RAAN
nexttile
plot(time,dRAAN,'LineWidth',1)
hold on
plot(time,dRAANJ3,'LineWidth',1)
% Graph pretty
ylim padded
xlim tight
yLab = ylabel('$\Omega_0 - \Omega_t$','Interpreter','latex');
plotTitle = title('Right Ascension [deg]','interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set(yLab,'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
grid on
legend('J2','J3','interpreter','latex','Location','northoutside')

% Plot delta omega
nexttile
plot(time,dw,'LineWidth',1)
hold on
plot(time,dwJ3,'LineWidth',1)
% Graph pretty
ylim padded
xlim tight
yLab = ylabel('$\omega_0 - \omega_t$','Interpreter','latex');
plotTitle = title('Argument of Perigee [deg]','interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set(yLab,'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
grid on

% Plot angular momentum
nexttile
plot(time,dh,'LineWidth',1)
hold on
plot(time,dhJ3,'LineWidth',1)
% Graph pretty
ylim padded
xlim tight
yLab = ylabel('$h_0 - h_t$','Interpreter','latex');
plotTitle = title('Angular Momentum [$\frac{km^2}{s}
$]','interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set(yLab,'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
grid on

% Plot eccentricity
nexttile
plot(time,decc,'LineWidth',1)
hold on
```

```
plot(time,deccJ3,'LineWidth',1)
% Graph pretty
ylim padded
xlim tight
yLab = ylabel('$ecc_0 - ecc_t$','Interpreter','latex');
plotTitle = title('Eccentricity','interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set(yLab,'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
grid on

% Plot delta inclination
nexttile
plot(time,dinc,'LineWidth',1)
hold on
plot(time,dincJ3,'LineWidth',1)
% Graph pretty
ylim padded
xlim tight
xLab = xlabel('Time [hours]','Interpreter','latex');
yLab = ylabel('$inc_0 - inc_t$','Interpreter','latex');
plotTitle = title('Inclination [deg]','interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set(yLab,'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
set(xLab,'FontSize', 14)
grid on

% Overall title
sgtitle('J2 and J3 Effects Combined','interpreter','latex')

% Comment on this.
disp("Adding J3 to J2 is barely noticable on the plots. Clearly, J2 effects
are much larger than J3.")

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Problem 4: NRLMSISE00

Run 150 days. Expect deorbit around 120-130 using this model.

```
% Plot VoP using NRLMSISE00
% Define COEs vector (in RADIANS)
state =
[deg2rad(raan0);deg2rad(inc0);deg2rad(omega0);h0;ecc0;deg2rad(theta0)];

% Prepare to call ode45
options = odeset('RelTol', 1e-8, 'AbsTol',1e-8);
options = odeset(options,'Events',@eventfun_VOP_NRLMSISE00);
```

```
t = 150*days2sec;
tspan = [0 t];

% Pick my birthday (11/26/2022 at 20:50 IN UTC)
y = 1987;
month = 11;
d = 4; % UTC remember
UT = [04 50 00]; % UTC conversion from 20:50 on 11/26
[jd_initial, ~, ~] = juliandateFunction(y,month,d,UT);

% Time and call function - VOP USING NRLMSISE00
tstart = tic;
[timenew, COEsnew] =
ode45(@VoP_NRLMSISE00,tspan,state,options,mu,wE,Cd,A,m,jd_initial);
tnrl = toc(tstart);

% Extract outputs (RADIANS RADIANS RADIANS)
raanNew = COEsnew(:,1);
incNew = COEsnew(:,2);
wNew = COEsnew(:,3);
hNew = COEsnew(:,4);
eccNew = COEsnew(:,5);
thetaNew = COEsnew(:,6);

% Print timing
disp("NRLMSISE Method (using VoP) run time is: " + tnrl + " seconds")

%..........Plot apogee/perigee change with time

% Find r, v vectors. Best way I know how is to run COES2RV
r = zeros(3,length(timenew));
altitude = zeros(length(timenew),1);

for i = 1:length(timenew)
[r(:,i),~] =
r_and_v_from_COEs(raanNew(i),incNew(i),wNew(i),hNew(i),eccNew(i),thetaNew(i))
;
altitude(i) = norm(r(1:3,i)) - r_earth;
end
r = r';

[max_altitude,imax] = findpeaks(altitude);
[~,imin] = findpeaks(-altitude);

min_altitude = zeros(length(imin),1);
for i = 1:length(imin)
    min_altitude(i) = norm(r(imin(i),1:3)) - r_earth;
end

apogee.nrl = [timenew(imax) max_altitude]; % Maximum altitudes and times
perigee.nrl = [timenew(imin) min_altitude]; % Min altitudes and times

% [time, state]
%apogee = sortrows(maxima,1); % Maxima sorted with time
```

```
%perigee = sortrows(minima,1); % Minima sorted with time

figure()
plot(apogee.nrl(:,1)*sec2days,apogee.nrl(:,2),'LineWidth',2)
hold on
plot(perigee.nrl(:,1)*sec2days,perigee.nrl(:,2),'LineWidth',2)

% Graph pretty
ylim padded
xlim tight
xLab = xlabel('Time [days]','Interpreter','latex');
yLab = ylabel('Altitude [km]','Interpreter','latex');
plotTitle = title("Atmospheric Drag using NRLMSISE00 Method: Change in
Apogee and Perigee with time",'interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set([xLab, yLab],'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
set([xLab, yLab],'FontSize', 14)
grid on
legend('Apogee','Perigee', 'interpreter','latex','Location', 'best')


%..........Plot changes in COEs with time [NRLMSISE00 Method]
% Convert back to degrees for plots
raanNew = rad2deg(raanNew);
incNew = rad2deg(incNew);
wNew = rad2deg(wNew);

dRAAN = raan0 - raanNew;
dinc = inc0 - incNew;
dw = omega0 - wNew;
dh = h0 - hNew;
decc = ecc0 - eccNew;

time = timenew .* sec2days;

figure()
tiledlayout("vertical")

% Plot delta RAAN
nexttile
plot(time,dRAAN,'LineWidth',2)
% Graph pretty
ylim padded
xlim tight
yLab = ylabel('$\Omega_0 - \Omega_t$','Interpreter','latex');
plotTitle = title('Right Ascension [deg]','interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set(yLab,'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
grid on
```

```
% Plot delta omega
nexttile
plot(time,dw,'LineWidth',2)
% Graph pretty
ylim padded
xlim tight
yLab = ylabel('$\omega_0 - \omega_t$','Interpreter','latex');
plotTitle = title('Argument of Perigee [deg]','interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set(yLab,'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
grid on

% Plot angular momentum
nexttile
plot(time,dh,'LineWidth',2)
% Graph pretty
ylim padded
xlim tight
yLab = ylabel('$h_0 - h_t$','Interpreter','latex');
plotTitle = title('Angular Momentum [$\frac{km^2}{s}
$]','interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set(yLab,'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
grid on

% Plot eccentricity
nexttile
plot(time,decc,'LineWidth',2)
% Graph pretty
ylim padded
xlim tight
yLab = ylabel('$ecc_0 - ecc_t$','Interpreter','latex');
plotTitle = title('Eccentricity','interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set(yLab,'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
grid on

% Plot delta inclination
nexttile
plot(time,dinc,'LineWidth',2)
% Graph pretty
ylim padded
xlim tight
xLab = xlabel('Time [days]','Interpreter','latex');
yLab = ylabel('$inc_0 - inc_t$','Interpreter','latex');
plotTitle = title('Inclination [deg]','interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
```

```
set(yLab,'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
set(xLab,'FontSize', 14)
grid on

% Overall title
sgtitle('Drag using NRLMSISE00 Model','interpreter','latex')


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Comparison Plots

```
figure()
% NRL plots
plot(apogee.nrl(:,1)*sec2days,apogee.nrl(:,2),'LineWidth',2)
hold on
plot(perigee.nrl(:,1)*sec2days,perigee.nrl(:,2),'LineWidth',2)

% VoP plots
plot(apogee.vop(:,1)*sec2days,apogee.vop(:,2),'LineWidth',2)
plot(perigee.vop(:,1)*sec2days,perigee.vop(:,2),'LineWidth',2)

% Graph pretty
ylim padded
xlim tight
xLab = xlabel('Time [days]','Interpreter','latex');
yLab = ylabel('Altitude [km]','Interpreter','latex');
plotTitle = title("Atmospheric Drag using NRLMSISE00 vs VoP
Methods",'interpreter','latex');
set(plotTitle,'FontSize',14,'FontWeight','bold')
set(gca,'FontName','Palatino Linotype')
set([xLab, yLab],'FontName','Palatino Linotype')
set(gca,'FontSize', 9)
set([xLab, yLab],'FontSize', 14)
grid on
legend('Apogee NRL','Perigee NRL','Apogee VoP','Perigee VoP',
'interpreter','latex','Location', 'best')
```

Published with MATLAB® R2023b

4: MATLAB FUNCTIONS

Table of Contents


```
% _____________________________________________________________
function density = atmosphere(z)
%
% ATMOSPHERE calculates density for altitudes from sea level
% through 1000 km using exponential interpolation.
% Citation:
% Curtis, Howard D. Orbital Mechanics for Engineering Students: Revised
Reprint. Butterworth-Heinemann, 2020.
% _____________________________________________________________
%...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 4.008e-2 1.841e-2 3.996e-3 1.027e-3 3.097e-4 8.283e-5 ...
1.846e-5 3.416e-6 5.606e-7 9.708e-8 2.222e-8 8.152e-9 3.831e-9 ...
2.076e-9 5.194e-10 2.541e-10 6.073e-11 1.916e-11 7.014e-12 2.803e-12 ...
1.184e-12 5.215e-13 1.137e-13 3.070e-14 1.136e-14 5.759e-15 3.561e-15];

%...Scale heights (km):
H = ...
[ 7.310 6.427 6.546 7.360 8.342 7.583 6.661 ...
5.927 5.533 5.703 6.782 9.973 13.243 16.322 ...
21.652 27.974 34.934 43.342 49.755 54.513 58.019 ...
60.980 65.654 76.377 100.587 147.203 208.020];

%...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
% _____________________________________________________________________
```

Published with MATLAB® R2023b

Table of Contents


```
function [r_ECI,v_ECI] = r_and_v_from_COEs(RAAN,inc,w,h,Ecc,theta)
% Finds the r and v vectors in an Earth Centered Inertial reference frame
% Inputs
    % inc = inclination [deg]
    % w = argiment of perigee [deg]
    % RAAN = right ascension of the ascending node [deg]
    % h = angular momentum [km2/s]
    % Ecc = eccentricity
    % theta = true anomaly [deg]
% Outputs
    % r_vect = position vector in ECI frame [km]
    % v_vect = velocity vector in ECI frame [km/s]

% constant
muEarth = 398600;

% Preallocate for speed
cr = cos(RAAN);
sr = sin(RAAN);
ci = cos(inc);
si = sin(inc);
cw = cos(w);
sw = sin(w);

% determine perifocal to GEO rotation matrix
Cz_RAAN =        [  cr     sr     0;
                   -sr     cr     0;
                    0      0      1]; % 3

Cx_inc =         [  1      0      0;
                    0      ci     si;
                    0     -si     ci]; % 1

Cz_w =           [  cw     sw     0;
                   -sw     cw     0;
                    0      0      1]; % 3


Qgeo2peri = Cz_w * Cx_inc * Cz_RAAN;
Qperi2geo = Qgeo2peri';

% find r and v (Perifocal)
r_PF = (((h^2)/muEarth)*(1/(1+Ecc*cos(theta))))*[cos(theta);sin(theta);0];
v_PF = (muEarth/h)*[-sin(theta); (Ecc+cos(theta)); 0];

% calc r and v relative to the geocentric reference frame
% GEO = ECI
```

```
r_ECI = Qperi2geo*r_PF;
v_ECI = Qperi2geo*v_PF;
```

Published with MATLAB® R2023b

Table of Contents


```
function [value, isterminal, direction] = eventfun(time,state,mu,wE,Cd,A,m)

% This creates an event function that shuts off the ode when propogating
% orbits under 100 km (aka; deorbit)

r = norm(state(1:3));
z = r - 6378; % altitude

value = z - 100;
isterminal = 1; % stop
direction = -1; %

end
```

Published with MATLAB® R2023b

Table of Contents


```
function [dstate] = perts_cowells(time,state,mu,wE,Cd,A,m)
%{

This function is for numerical integration of orbital trajectory (i.e.,
plug into ode45) including orbital perturbations using Cowell's Method


INPUTS:
    INITIAL CONDITIONS
    state vector(1:3) = position (rx,ry,rz),                          km
    state vector(4:6) = velocity (vx,vy,vz),                          km/s
    mu = [c] gravitational parameter of main body,                    km3/s2
    wE = [3x1] angular velocity of planet (in this case, Earth),      rad/s
    Cd = [c] drag coefficient
    A = [c] ram direction area,                                 *******    KM 2
    m = [c] s/c mass,                                                 kg

OUTPUT: New state vector after integration.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Justin Self, Cal Poly, Fall 2023: AERO 452 | Spaceflight Dynamics II
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%}

% Extract state vector into r, v components
rx = state(1);
ry = state(2);
rz = state(3);
vx = state(4);
vy = state(5);
vz = state(6);

r_vect = [rx ry rz];
r = norm(r_vect);

% Compute relative velocity vector @ each timestep
vVect = [vx vy vz];
v_relVect = vVect - cross(wE,r_vect); % relative velocity wrt rotating
atmosphere
v_rel = norm(v_relVect);

% Find rho at every altitude; call 1976 standard atmosphere
% Curtis code (see citation inside function)
z = r - 6378; % altitude now
rho = atmosphere(z); % in kg/m3
rho = rho*1000^3; % into kg/km3

% Change in accelareation due to orbital perturbations
```

```
p = -(1/2) * rho*v_rel * (Cd*A/m) * v_relVect; % drag (ONLY UNDER 1000 KM)

px = p(1);
py = p(2);
pz = p(3);

% Cowell's Method
xdot = vx;
ydot = vy;
zdot = vz;
xddot = (-mu*rx)/(r^3) + px;
yddot = (-mu*ry)/(r^3) + py;
zddot = (-mu*rz)/(r^3) + pz;

dstate = [xdot; ydot; zdot; xddot; yddot; zddot];


end
```

Published with MATLAB® R2023b

Table of Contents


```
function [xx,yy,zz] = earth_sphere(varargin)

%
%
%
% EARTH_SPHERE Generate an earth-sized sphere.
%   [X,Y,Z] = EARTH_SPHERE(N) generates three (N+1)-by-(N+1)
%   matrices so that SURFACE(X,Y,Z) produces a sphere equal to
%   the radius of the earth in kilometers. The continents will be
%   displayed.
%
%   [X,Y,Z] = EARTH_SPHERE uses N = 50.
%
%   EARTH_SPHERE(N) and just EARTH_SPHERE graph the earth as a
%   SURFACE and do not return anything.
%
%   EARTH_SPHERE(N,'mile') graphs the earth with miles as the unit rather
%   than kilometers. Other valid inputs are 'ft' 'm' 'nm' 'miles' and 'AU'
%   for feet, meters, nautical miles, miles, and astronomical units
%   respectively.
%
%   EARTH_SPHERE(AX,...) plots into AX instead of GCA.
%
%  Examples:
%    earth_sphere('nm') produces an earth-sized sphere in nautical miles
%
%    earth_sphere(10,'AU') produces 10 point mesh of the Earth in
%    astronomical units
%
%    h1 = gca;
%    earth_sphere(h1,'mile')
%    hold on
%    plot3(x,y,z)
%      produces the Earth in miles on axis h1 and plots a trajectory from
%      variables x, y, and z
%   Clay M. Thompson 4-24-1991, CBM 8-21-92.
%   Will Campbell, 3-30-2010
```

Input Handling

```
[cax,args,nargs] = axescheck(varargin{:}); % Parse possible Axes input
error(nargchk(0,2,nargs)); % Ensure there are a valid number of inputs
% Handle remaining inputs.
```

```
% Should have 0 or 1 string input, 0 or 1 numeric input
j = 0;
k = 0;
n = 50; % default value
units = 'km'; % default value
for i = 1:nargs
    if ischar(args{i})
        units = args{i};
        j = j+1;
    elseif isnumeric(args{i})
        n = args{i};
        k = k+1;
    end
end
if j > 1 || k > 1
    error('Invalid input types')
end
```

Calculations

Scale factors

```
Scale = {'km' 'm'  'mile'            'miles'           'nm'
'au'                  'ft';
         1    1000 0.621371192237334 0.621371192237334 0.539956803455724
6.6845871226706e-009 3280.839895};
% Identify which scale to use
try
    myscale = 6378.1363*Scale{2,strcmpi(Scale(1,:),units)};
catch %#ok<*CTCH>
    error('Invalid units requested. Please use m, km, ft, mile, miles, nm,
or AU')
end

% -pi <= theta <= pi is a row vector.
% -pi/2 <= phi <= pi/2 is a column vector.
theta = (-n:2:n)/n*pi;
phi = (-n:2:n)'/n*pi/2;
cosphi = cos(phi); cosphi(1) = 0; cosphi(n+1) = 0;
sintheta = sin(theta); sintheta(1) = 0; sintheta(n+1) = 0;
x = myscale*cosphi*cos(theta);
y = myscale*cosphi*sintheta;
z = myscale*sin(phi)*ones(1,n+1);
```

Plotting

```
if nargout == 0
    cax = newplot(cax);
    % Load and define topographic data
    load('topo.mat','topo','topomap1');
    % Rotate data to be consistent with the Earth-Centered-Earth-Fixed
    % coordinate conventions. X axis goes through the prime meridian.
    % http://en.wikipedia.org/wiki/
```

```
Geodetic_system#Earth_Centred_Earth_Fixed_.28ECEF_or_ECF.29_coordinates
    %
    % Note that if you plot orbit trajectories in the Earth-Centered-
    % Inertial, the orientation of the contintents will be misleading.
    topo2 = [topo(:,181:360) topo(:,1:180)]; %#ok<NODEF>

    % Define surface settings
    props.FaceColor= 'texture';
    props.EdgeColor = 'none';
    props.FaceLighting = 'phong';
    props.Cdata = topo2;
    % Create the sphere with Earth topography and adjust colormap
    surface(x,y,z,props,'parent',cax)
    colormap(topomap1)
% Replace the calls to surface and colormap with these lines if you do
% not want the Earth's topography displayed.
%     surf(x,y,z,'parent',cax)
%     shading flat
%     colormap gray

    % Refine figure
    axis equal
    xlabel(['X [' units ']'])
    ylabel(['Y [' units ']'])
    zlabel(['Z [' units ']'])
    view(127.5,30)
else
    xx = x; yy = y; zz = z;
end
```


Published with MATLAB® R2023b

Table of Contents


```
function [r,v] = universalVar(mu,deltat,r0Vect,v0Vect,TOL)

% Self, Justin
% Aero351: Orbital Mechanics I

% This function uses Newton's method to solve the Stumpf functions for the
Universal Variable

% Inputs:
% mu: gravitational parameter of planet of interest
% deltat: time interval (fixed) desired between iterations
% v0Vect = initial velocity vector, 3x1
% r0Vect = initial position vector, 3x1
% TOL = tolerance desired (error); use 1e-8

% Output:
% [r, v] = new r, v vectors

% UNIVERSAL VARIABLE ALGORITHM

% ALGORITHM 3.4, Part 1.
% i. Find r and v
r0 = norm(r0Vect);
v0 = norm(v0Vect);

% ii. Find vr|0 by projecting v0 onto onto the direction of r0
vr0 = dot(v0Vect,r0Vect)* r0^(-1);

% iii. The reciprical of the semimajor axis, alpha, (using Eqn. 3.48)
alpha = (2/r0)  - (v0^2 / mu);

% OPTIONAL:
%{
% Learn something from the sign of alpha
if alpha < 0
    disp("This is a hyperbolic trajectory")
elseif alpha > 0
    disp("This is an elliptical orbit")
else
    disp("This is a parabolic orbit.")
end
%}

% ALGORITHM 3.4, Part 2.
```

```
% Use Algorithm 3.3 to find the universal anomaly X.
% i. use Eq. 3.66 for an initial estimate of X0.
% ii - v, use homebuilt Universal Variable function to solve for X.
```

Solve for X (UniversalVariable)

Figure out the best initial guess

```
x0 = sqrt(mu)*abs(alpha)*deltat;

%z = alpha*(X^2);

% Define f and fprime functions
f = @(X) ((r0 * vr0) / (mu^.5)) * (X^2) * Cstumpf(alpha,X) + (1 -
(alpha*r0))*(X^3) * Sstumpf(alpha,X) + r0*X - ((mu^.5)*deltat);
fprime = @(X) ((r0 * vr0) / (mu^.5)) * X * (1 -
alpha*(X^2)*Sstumpf(alpha,X))  + (1 - (alpha*r0)) *(X^2)* Cstumpf(alpha,X) +
r0;

% Newton's Method
x1 = x0 - f(x0) / fprime(x0);
x = [x0; x1];
err = abs(x1 - x0);

while err > TOL
    x0 = x1;
    x1 = x0 - f(x0) / fprime(x0);
    err = abs(x1 - x0);
    x = [x; x1];
end

X = x(end);
```

Use X to find new r, v vectors

```
% ALGORITHM 3.4, Part 3.
% Use Eqns (3.69a) and (3.69b) to obtain f and g
f = 1 - (X^2 / r0) * Cstumpf(alpha,X);
g = deltat - mu^(-.5)*(X^3)* Sstumpf(alpha,X);

% ALGORITHM 3.4, Part 4.
% Use Eqn (3.67) to compute r_vect_f and r_mag_f
rf_vect = f*r0Vect + g*v0Vect;
rfmag = norm(rf_vect);

% ALGORITHM 3.4, Part 5.
% Use Eqns (3.69c) and (3.69d) to obtain fdot and gdot.
fdot = ((mu^.5) / (rfmag*r0)) * (alpha*(X^3)*Sstumpf(alpha,X) - X);
gdot = 1 - ((X^2) / rfmag)*Cstumpf(alpha,X);

% ALGORITHM 3.4, Part 6.
vf_vect = fdot*r0Vect + gdot*v0Vect;
vfmag = norm(vf_vect);
```

```
% FINAL RESULTS
r = rf_vect;
v = vf_vect;

end % function
```

Published with MATLAB® R2023b

Table of Contents


```
function S = Sstumpf(alpha, X)

% This function takes alpha and X as part of the universal variable solving
% algorithm.

z = alpha*(X^2);

S = (1/6) - (z/120) + ((z^2)/5040) - ((z^3) / 362880); % enough for us!

end
```

Published with MATLAB® R2023b

Table of Contents


```
function C = Cstumpf(alpha, X)

% This function takes alpha and X as part of the universal variable solving
% algorithm.

z = alpha*(X^2);

C = (1/2) - (z/24) + ((z^2) / 720) - ((z^3) / 40320); % enough!

end
```

Published with MATLAB® R2023b

Table of Contents


```
function dCOEs = VariationOfParameters(time,COEs,mu,wE,Cd,A,m)

%{
Variation of Parameters method for propogating orbital perturbations.

INPUTS: 6x1 COEs initial state vector in RADIANS
        wE = angular velocity of the earth; ECI frame (3x1)

OUTPUT: dCOEs for integration using ode45

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Justin Self, Cal Poly, Fall 2023: AERO 452 | Spaceflight Dynamics II
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%}

% Noemclature:
% curtis,ABERCROMBY ==  r,R; s,T; w,N
% ECI == X,Y,Z

% Bring in COEs (RADIANS)
raan = COEs(1);
inc = COEs(2);
w = COEs(3);
h = COEs(4);
ecc = COEs(5);
theta = COEs(6);

% Calcuate r,v from COEs each time step.

% % Run rv from cose function here plainly for speed
%%%%%%%%%%%%%%%%%% %%%%%%%%%%%%%%%%%% %%%%%%%%%%%%%%%%%% %%%%%%%%%%%%%%%%%%
muEarth = 398600;

% Preallocate for speed
cr = cos(raan);
sr = sin(raan);
ci = cos(inc);
si = sin(inc);
cw = cos(w);
sw = sin(w);
ct = cos(theta);
st = sin(theta);
ti = tan(inc);
```

```
% determine perifocal to GEO rotation matrix
QXx =   [  cr*cw - ci*sr*sw, cw*sr + ci*cr*sw, si*sw;
         - cr*sw - ci*cw*sr, ci*cr*cw - sr*sw, cw*si;
           si*sr,            -cr*si,           ci];
QXx = QXx';

% find r and v (Perifocal)
r_PF = (((h^2)/muEarth)*(1/(1+ecc*ct)))*[ct;st;0];
v_PF = (muEarth/h)*[-st; (ecc+ct); 0];

% calc r and v relative to the geocentric reference frame
% GEO = ECI
rVect = QXx*r_PF;
v = QXx*v_PF;

%%%%%%%%%%%%%%%%%% %%%%%%%%%%%%%%%%%% %%%%%%%%%%%%%%%%%% %%%%%%%%%%%%%%%%%%
```

Direction Cosine Matrix from XYZ to RTN

```
% Q{ECI-->RTN} each time

% Convenient definition: "argument of latitude"
u = w + theta;
cu = cos(u);
su = sin(u);

% ECI to RTN
%QXr = [ -sr*ci*su + cr*cu,  cr*ci*su + sr*cu,   si*su;
 %       -sr*ci*su - cr*su,  cr*ci*cu - sr*su,   si*cu;
  %       sr*si,             -cr*si,                ci];
```

Drag

Compute relative velocity vector @ each timestep

```
v_relVect = v - cross(wE',rVect); % relative velocity wrt rotating atmosphere
v_rel = norm(v_relVect);
r = norm(rVect);

% Find rho at every altitude; call 1976 standard atmosphere
% Curtis code (see citation inside function)
z = r - 6378; % altitude now
rho = atmosphere(z); % in kg/m3
rho = rho*(1e9); % into kg/km3

% Drag acceleration IN ECI
p = -.5 * rho*v_rel * (Cd*A/m) * v_relVect; % drag (ONLY UNDER 1000 KM)

% Change from ECI to RTN frame

% ALTERNATE METHOD (THAT WORKS!)
R_hat = rVect/r;
```

```
N_hat = cross(rVect,v)/norm(cross(rVect,v));
T_hat = cross(N_hat,R_hat);
QXr = [R_hat T_hat N_hat];
QXr = QXr';

% /END ALTERNATE METHOD

p = QXr*p;

R = p(1);
T = p(2);
N = p(3);
```

COEs

```
% Recall: curtis,ABERCROMBY ==  r,R; s,T; w,N

% h
dh = r*T;

% ecc
decc = (h/mu) * st*R + (1/(mu*h)) * ((h^2 + mu*r)*cos(theta) + mu*ecc*r)*T;

% theta
twobodymotion = h/r^2;
dtheta_pert = (1/(ecc*h)) * ( (h^2/mu)*cos(theta)*R - ( (h^2/mu) +
r)*sin(theta)*T  );

% Final pert
dtheta = twobodymotion + dtheta_pert;

% inc
dinc =  (r/h) * cu * N;

% raan
draan = ((r*su) / (h*si)) * N;

% w
dw = -dtheta_pert - ((r*su) / (h*ti)) * N;

% Final output
dCOEs = [draan;dinc;dw;dh;decc;dtheta];

end
```

Published with MATLAB® R2023b

Table of Contents


```
function [value, isterminal, direction] =
eventfun_VOP(time,COEs,mu,wE,Cd,A,m)

% This creates an event function that shuts off the ode when propogating
% orbits under 100 km (aka; deorbit)

% Bring in COEs RADIANS
raan = COEs(1);
inc = COEs(2);
w = COEs(3);
h = COEs(4);
ecc = COEs(5);
theta = COEs(6);

% % Run rv from cose function here plainly for speed
%%%%%%%%%%%%%%%%%% %%%%%%%%%%%%%%%%%% %%%%%%%%%%%%%%%%%% %%%%%%%%%%%%%%%%%%
muEarth = 398600;

% Preallocate for speed
cr = cos(raan);
sr = sin(raan);
ci = cos(inc);
si = sin(inc);
cw = cos(w);
sw = sin(w);
ct = cos(theta);
st = sin(theta);

% determine perifocal to GEO rotation matrix
QXx =   [  cr*cw - ci*sr*sw, cw*sr + ci*cr*sw, si*sw;
         - cr*sw - ci*cw*sr, ci*cr*cw - sr*sw, cw*si;
           si*sr,            -cr*si,           ci];
QXx = QXx';

% find r and v (Perifocal)
r_PF = (((h^2)/muEarth)*(1/(1+ecc*ct)))*[ct;st;0];

% Convert to ECI
rVect = QXx*r_PF;

r = norm(rVect);
z = r - 6378; % altitude

value = z - 100;
isterminal = 1; % stop
direction = -1; %
```

```
end
```

Published with MATLAB® R2023b

Table of Contents


```
function [value, isterminal, direction] = eventfun_J2_cowells(time,state,mu)

% This creates an event function that shuts off the ode when propogating
% orbits under 100 km (aka; deorbit)

r = norm(state(1:3));
z = r - 6378; % altitude

value = z - 100;
isterminal = 1; % stop
direction = -1; %

end
```

Published with MATLAB® R2023b

Table of Contents


```
function [dstate] = J2_cowells(time,state,mu)
%{

This function is for numerical integration of orbital trajectory (i.e.,
plug into ode45) including orbital perturbations using Cowell's Method


INPUTS:
    INITIAL CONDITIONS
    state vector(1:3) = position (rx,ry,rz),                          km
    state vector(4:6) = velocity (vx,vy,vz),                          km/s
    mu = [c] gravitational parameter of main body,                    km3/s2
    wE = [3x1] angular velocity of planet (in this case, Earth),      rad/s
    Cd = [c] drag coefficient
    A = [c] ram direction area,                                       km^2
    m = [c] s/c mass,                                                 kg

OUTPUT: New state vector after integration.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Justin Self, Cal Poly, Fall 2023: AERO 452 | Spaceflight Dynamics II
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%}

% Extract state vector into r, v components
rx = state(1);
ry = state(2);
rz = state(3);
vx = state(4);
vy = state(5);
vz = state(6);
R = [rx ry rz];
r = norm(R);
re = 6378; % km

% J2
J2 = 0.0010826355; % From Vallado
aI = - ( (3*J2*mu*re^2*rx) / (2*r^5) ) * (1 - (5*rz^2)/(r^2) );
aJ = - ( (3*J2*mu*re^2*ry) / (2*r^5) ) * (1 - (5*rz^2)/(r^2) );
aK = - ( (3*J2*mu*re^2*rz) / (2*r^5) ) * (3 - (5*rz^2)/(r^2) );

% Cowell's Method
dx = vx;
dy = vy;
dz = vz;
ddx = (-mu*rx)/(r^3) + aI;
ddy = (-mu*ry)/(r^3) + aJ;
ddz = (-mu*rz)/(r^3) + aK;
```

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

end
```

Published with MATLAB® R2023b

Table of Contents


```
function [dstate] = J2_J3_cowells(time,state,mu)
%{

This function is for numerical integration of orbital trajectory (i.e.,
plug into ode45) including orbital perturbations using Cowell's Method


INPUTS:
    INITIAL CONDITIONS
    state vector(1:3) = position (rx,ry,rz),                          km
    state vector(4:6) = velocity (vx,vy,vz),                          km/s
    mu = [c] gravitational parameter of main body,                    km3/s2
    wE = [3x1] angular velocity of planet (in this case, Earth),      rad/s
    Cd = [c] drag coefficient
    A = [c] ram direction area,                                       km^2
    m = [c] s/c mass,                                                 kg

OUTPUT: New state vector after integration.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Justin Self, Cal Poly, Fall 2023: AERO 452 | Spaceflight Dynamics II
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%}

% Extract state vector into r, v components
rx = state(1);
ry = state(2);
rz = state(3);
vx = state(4);
vy = state(5);
vz = state(6);
R = [rx ry rz];
r = norm(R);
re = 6378; % km

% J2
J2 = 0.0010826355; % From Vallado
J2_I = - ( (3*J2*mu*re^2*rx) / (2*r^5) ) * (1 - (5*rz^2)/(r^2) );
J2_J = - ( (3*J2*mu*re^2*ry) / (2*r^5) ) * (1 - (5*rz^2)/(r^2) );
J2_K = - ( (3*J2*mu*re^2*rz) / (2*r^5) ) * (3 - (5*rz^2)/(r^2) );

% J3
J3 = J2*(-2.33936e-3); % From Vallado
J3_I = - ( (5*J3*mu*re^3*rx) / (2*r^7) ) * (3*rz - (7*rz^3)/(r^2) );
J3_J = - ( (5*J3*mu*re^3*ry) / (2*r^7) ) * (3*rz - (7*rz^3)/(r^2) );
J3_K = - ( (5*J3*mu*re^3) / (2*r^7) ) * (6*rz^2 - (7*rz^4)/(r^2) -
(3/5)*r^2 );
```

```
aI = J2_I + J3_I;
aJ = J2_J + J3_J;
aK = J2_K + J3_K;

% Cowell's Method
dx = vx;
dy = vy;
dz = vz;
ddx = (-mu*rx)/(r^3) + aI;
ddy = (-mu*ry)/(r^3) + aJ;
ddz = (-mu*rz)/(r^3) + aK;

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


end
```

Published with MATLAB® R2023b

Table of Contents


```
function [value, isterminal, direction] =
eventfun_VOP_NRLMSISE00(time,COEs,mu,wE,Cd,A,m,jd_initial)

% This creates an event function that shuts off the ode when propogating
% orbits under 100 km (aka; deorbit)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Justin Self, Cal Poly, Fall 2023: AERO 452 | Spaceflight Dynamics II
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Bring in COEs
raan = COEs(1);
inc = COEs(2);
w = COEs(3);
h = COEs(4);
ecc = COEs(5);
theta = COEs(6);

% % Run rv from cose function here plainly for speed
%%%%%%%%%%%%%%%%%% %%%%%%%%%%%%%%%%%% %%%%%%%%%%%%%%%%%% %%%%%%%%%%%%%%%%%%

% Preallocate for speed
cr = cos(raan);
sr = sin(raan);
ci = cos(inc);
si = sin(inc);
cw = cos(w);
sw = sin(w);
ct = cos(theta);
st = sin(theta);

% determine perifocal to GEO rotation matrix
QXx =    [  cr*cw - ci*sr*sw, cw*sr + ci*cr*sw, si*sw;
          - cr*sw - ci*cw*sr, ci*cr*cw - sr*sw, cw*si;
            si*sr,            -cr*si,           ci];
QXx = QXx';

% find r and v (Perifocal)
r_PF = (((h^2)/mu)*(1/(1+ecc*ct)))*[ct;st;0]; % p,q,w coordinate vects

% GEO = ECI
rVect = QXx*r_PF;
r = norm(rVect);

% altitude check
z = r - 6378;
```

```
value = z - 100;
isterminal = 1; % stop
direction = -1; %

end
```

Published with MATLAB® R2023b

Table of Contents


```
function [juliandateReturn, UT_decimal, j0] = juliandateFunction(y,m,d,UT)
%   This function outputs the julian date (JD) with inputs year, month,
%   day, and GMT (UT)
% Author: Justin Self AERO 351 Fall 2022

% y = four digit year
% m = two digit month
% d = two digit day
% UT = array that contains the UT in the form [hour min sec]

j0 = 367*y - floor((7*(y+floor((m+9)/12)))/4) + floor((275*m)/9) + d +
1721013.5;

% j0 = Julian date at 0 hours UT (noon)
% Now ww need to convert HH:MM:SS to HH.HH (decimal hours)

UT_decimal = UT(1) + UT(2)/60 + UT(3)/3600; % 60 min / hour, 3600 sec / hour
juliandateReturn = j0 + (UT_decimal/24);

end
```

Published with MATLAB® R2023b

Table of Contents


```
function dCOEs = VoP_NRLMSISE00(time,COEs,mu,wE,Cd,A,m,jd_initial)

%{
Variation of Parameters method for propogating orbital perturbations.

INPUTS: 6x1 COEs initial state vector in RADIANS
        wE = angular velocity of the earth; ECI frame (3x1)

OUTPUT: dCOEs for integration using ode45

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Justin Self, Cal Poly, Fall 2023: AERO 452 | Spaceflight Dynamics II
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%}

% Noemclature:
% curtis,ABERCROMBY ==  r,R; s,T; w,N
% ECI == X,Y,Z
```

Bring in COEs

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

Calcuate r,v from COEs each time step.

```
% Preallocate for speed
cr = cos(raan);
sr = sin(raan);
ci = cos(inc);
si = sin(inc);
cw = cos(w);
sw = sin(w);
ct = cos(theta);
```

```
st = sin(theta);
ti = tan(inc);

% determine perifocal to GEO rotation matrix
QXx =    [  cr*cw - ci*sr*sw, cw*sr + ci*cr*sw, si*sw;
          - cr*sw - ci*cw*sr, ci*cr*cw - sr*sw, cw*si;
            si*sr,            -cr*si,           ci];
QXx = QXx';

% find r and v (Perifocal)
r_PF = (((h^2)/mu)*(1/(1+ecc*ct)))*[ct;st;0];
v_PF = (mu/h)*[-st; (ecc+ct); 0];

% calc r and v relative to the geocentric reference frame
% GEO = ECI
rVect = QXx*r_PF;
vVect = QXx*v_PF;

r = norm(rVect);

%%%%%%%%%%%%%%%%%% %%%%%%%%%%%%%%%%%% %%%%%%%%%%%%%%%%%% %%%%%%%%%%%%%%%%%%
```

Direction Cosine Matrix from XYZ to RTN

Q{ECI-->RTN} each time

```
% Convenient definition: "argument of latitude"
u = w + theta;
cu = cos(u);
su = sin(u);

% ALTERNATE METHOD (THAT WORKS!)
R_hat = rVect/r;
N_hat = cross(rVect,vVect)/norm(cross(rVect,vVect));
T_hat = cross(N_hat,R_hat);
QXr = [R_hat T_hat N_hat];
QXr = QXr';

% /END ALTERNATE METHOD

% old method
%QXr = [ -sr*ci*su + cr*cu,  cr*ci*su + sr*cu,   si*su;
      %  -sr*ci*su - cr*su,  cr*ci*cu - sr*su,   si*cu;
     %    sr*si,             -cr*si,                ci];
```

Drag perturbation using NRLMSISE00

Use NRLMSISE00 to find density at altitude each time step

```
jd = jd_initial + time/86400; % seconds to days

t = datetime(jd,'ConvertFrom','juliandate'); % new time in datetime() vector
DateVector = datevec(t);
```

```
year = DateVector(1);

% Convert the position to LLA coordinates from ECI coordinates every time
% step.

% LLA input in meters.
rVect_meters = rVect.*1000;
rVect_meters = rVect_meters';

% ECI to LLA function (MATLAB)
lla = eci2lla(rVect_meters,DateVector);
altitude = lla(3);              % meters
lat = lla(1);                   % deg
long = lla(2);                  % deg

% Compute UT day of year (like 200)
%dayOfYear = day(t,'dayofyear');%UTseconds = dayOfYear*86400;

UTseconds = DateVector(4)*3600 + DateVector(5)*60 + DateVector(6);
dayOfYear = DateVector(2)*30 + DateVector(3) -27; % finds roughly current
day.

% Compute UT seconds of year (convert day to sec)
%UTseconds = dayOfYear*86400;

% Call function
[~,rho] =
atmosnrlmsise00(altitude,lat,long,year,dayOfYear,UTseconds,'NoOxygen','None')
;
rho = 1e9*rho(6); % into kg/km3
```

Drag acceleration IN ECI

Compute relative velocity vector @ each timestep

```
v_relVect = vVect - cross(wE',vVect); % relative velocity wrt rotating
atmosphere
v_rel = norm(v_relVect);
r = norm(rVect);
p = -.5 * rho*v_rel * (Cd*A/m) * v_relVect; % drag (ONLY UNDER 1000 KM)

% Change from ECI to RTN frame
p = QXr*p;

R = p(1);
T = p(2);
N = p(3);
```

COEs

```
% Recall: curtis,ABERCROMBY ==  r,R; s,T; w,N

% h
```

```
dh = r*T;

% ecc
decc = (h/mu) * st*R + (1/(mu*h)) * ((h^2 + mu*r)*cos(theta) + mu*ecc*r)*T;

% theta
twobodymotion = h/r^2;
dtheta_pert = (1/(ecc*h)) * ( (h^2/mu)*cos(theta)*R - ( (h^2/mu) +
r)*sin(theta)*T  );

% Final pert
dtheta = twobodymotion + dtheta_pert;

% inc
dinc =  (r/h) * cu * N;

% raan
draan = ((r*su) / (h*si)) * N;

% w
dw = -dtheta_pert - ((r*su) / (h*ti)) * N;

% Final output
dCOEs = [draan;dinc;dw;dh;decc;dtheta];

end
```

Published with MATLAB® R2023b