Problem 1

```
%%%%%%
clear all; clc; close all

mu = 3.9860044189e5;
Re = 6378; %radius of earth
mo = 1800; %kg
rp_1 = 520 + Re;
ra_1 = 900 + Re;
e1 = (ra_1-rp_1)/(ra_1+rp_1);
h1 = sqrt(rp_1*mu*(1+e1));
vp_1 = h1/rp_1;
va_1 = h1/ra_1;

rp_2 = 520+Re;
ra_2 = 14000+Re;
e2 = (ra_2-rp_2)/(ra_2+rp_2);
h2 = sqrt(rp_2*mu*(1+e2));
vp_2 = h2/rp_1;
T = 12; % thrust in kN
Isp = 291;
g=9.81;
options = odeset('RelTol',1e-6);
dv = vp_2 - vp_1;
dt = 150;
i = 0;
while 1
    [T,Z] = ode45('OrbEq_thrust',[0 dt],[ra_1 0 0 0 va_1 0 mo],
 options);
    deltaV = -Isp*g*log(Z(end,7)/mo)/1000;
    dm = mo*(1-exp(-deltaV*1000/(Isp*g)));
    if dv-deltaV>0
        dt = dt + 0.1;
    else
        dt = dt - 0.1;
    end
    if abs(dv-deltaV)<.001
        break
    end
    i = i+1;
end

fprintf('Burn time: %0.1f s \n',dt)
fprintf('Total mass expended: %0.0f kg \n',dm)
fprintf('New eccentricity %0.2f \n',e2)
```

```
Burn time: 182.9 s
Total mass expended: 768 kg
New eccentricity 0.49
```

Published with MATLAB® R2017a

```
function [vecDeriv] = OrbEq(t,z)

mu = 3.9860044189e5;
r = sqrt(z(1)^2+z(2)^2+z(3)^2);
v = sqrt(z(4)^2+z(5)^2+z(6)^2);
T = 12;
Isp = 291;
go = 9.81;
vecDeriv(1) = z(4);
vecDeriv(2) = z(5);
vecDeriv(3) = z(6);
vecDeriv(4) = -mu/r^3*z(1) + T*z(4)/(v*z(7));
vecDeriv(5) = -mu/r^3*z(2) + T*z(5)/(v*z(7));
vecDeriv(6) = -mu/r^3*z(3) + T*z(6)/(v*z(7));
vecDeriv(7) = -T/(Isp*go)*1000;
vecDeriv = vecDeriv.';

end
```

```
function [RX,VX,r,v,QXx] = OrbElem2StateVec(h,e,i,w,W,true_ano)
%This function will take orbital elements and compute two state vectors r
and v
% r and v must be 3-D vectors
mu = 3.9860044189e5;
r = (h^2/mu)/(1+e*cos(true_ano))*[cos(true_ano) sin(true_ano) 0]';
v = mu/h*[-sin(true_ano) (e+cos(true_ano)) 0]';
R3_W = [cos(W)  sin(W) 0;
      -sin(W) cos(W) 0;
      0       0      1;];
R1_i = [1 0 0;
        0 cos(i) sin(i);
        0 -sin(i) cos(i);];
R3_w = [ cos(w) sin(w) 0;
        -sin(w) cos(w) 0
        0 0 1;];
QXx = (R3_w) *(R1_i) *(R3_W);

RX = QXx.'*r;
VX = QXx.'*v;

end
```

Problem 2

```
clear;clc;close all;
mu = 3.9860044189e5;
% parameters for Spacecraft A
A.e = 0.025;
A.W = 40*pi/180;
A.i = 60*pi/180;
A.w = 30*pi/180;
A.theta = 40*pi/180;
A.h = 52060;
% parameters for Spacecraft B
B.e = 0.007;
B.W = 40*pi/180;
B.i = 50*pi/180;
B.w = 120*pi/180;
B.theta = 40*pi/180;
B.h = 52360;
%find relative position, velocity, and acceleration of B relative to A

[rA, vA] = OrbElem2StateVec(A.h,A.e,A.i,A.w,A.W,A.theta);
[rB, vB] = OrbElem2StateVec(B.h,B.e,B.i,B.w,B.W,B.theta);


hA = cross(rA,vA);
ihat = rA/norm(rA);
khat = hA/norm(hA);
jhat = cross(khat,ihat);

W = hA/norm(rA)^2;
Wdot = -2*dot(vA,rA)/norm(rA)^2*W;

aA = -mu/norm(rA)^3*rA;
aB = -mu/norm(rB)^3*rB;
Rrel = rB-rA;
Vrel = vB - vA - cross(W,Rrel);
Arel = aB-aA - cross(Wdot,Vrel) - cross(W,cross(W,Rrel)) -
 2*cross(W,Vrel);
QXx = [ihat(1) jhat(1) khat(1);
        ihat(2) jhat(2) khat(2);
        ihat(3) jhat(3) khat(3);
        ];
r_lvlh = QXx.'*Rrel;
v_lvlh = QXx.'*Vrel;
a_lvlh = QXx.'*Arel;

fprintf('Relative position, velocity, and acceleration of B, relative
 to A, in local vertical/ local horizontal of A \n')
fprintf('Relative position [%0.1f %0.1f %0.1f] km \n',r_lvlh)
fprintf('Relative velocity [%0.3f %0.3f %0.3f] km/s \n',v_lvlh)
fprintf('Relative acceleration [%0.3s %0.3s %0.3s] km/s^2 \n',a_lvlh)
```

```
Relative position, velocity, and acceleration of B, relative to A, in
 local vertical/ local horizontal of A
Relative position [-6705.0 6829.1 -406.3] km
Relative velocity [0.313 0.111 1.247] km/s
Relative acceleration [8.317e-05 1.078e-04 5.058e-04] km/s^2
```

Published with MATLAB® R2017a

Problem 3;

```
clear;clc;close all;

Vxo = -0.18;
Vyo = -0.07;
Vzo = -0.08;
dt_1 = 5*60;
dt_2 = 60*60;

options = odeset('RelTol',1e-5);
[~,Z] = ode45('RelMotion',[0 dt_1],[0 0 0 Vxo Vyo Vzo],options);
fprintf('Cygnus distance from the STS after 5 min %.1f m
 \n',norm(Z(end,(1:3))))
[~,Z] = ode45('RelMotion',[0 dt_2],[0 0 0 Vxo Vyo Vzo],options);
fprintf('Cygnus distance from the STS after 60 min %.1f m
 \n',norm(Z(end,(1:3))))
```

```
Cygnus distance from the STS after 5 min 64.3 m
Cygnus distance from the STS after 60 min 1522.4 m
```

Published with MATLAB® R2017a

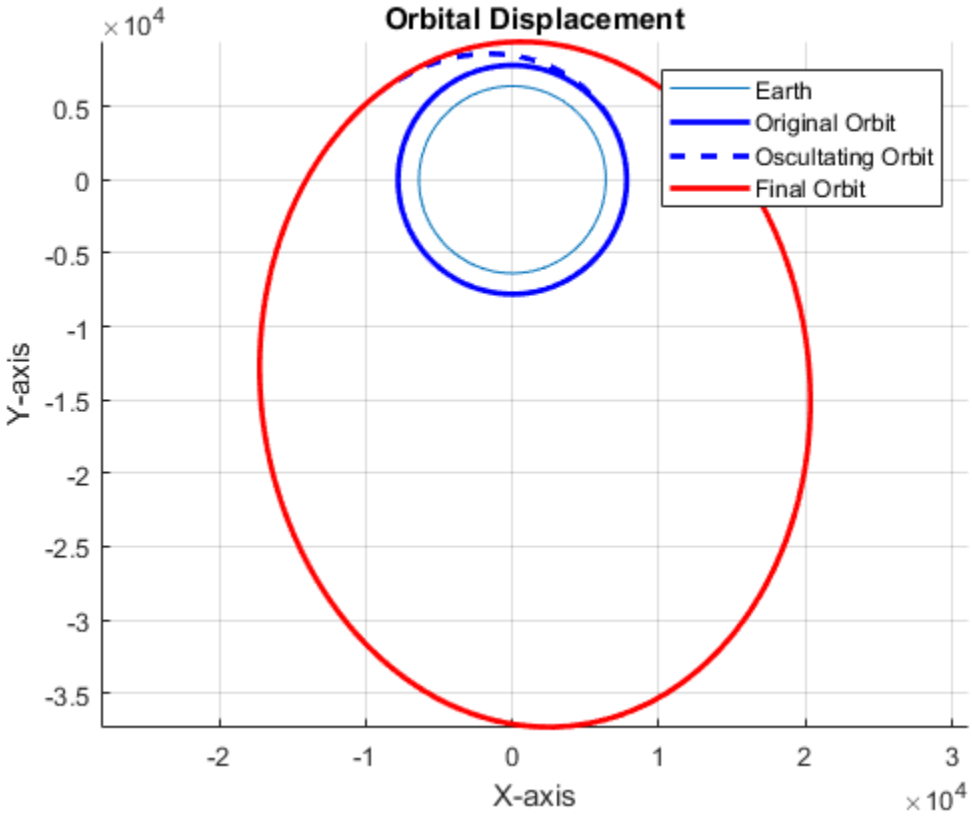
```
function [vecDeriv] = RelMotion(t,z)

mu = 3.9860044189e5;
r = 650+6378;
n = sqrt(mu/r^3);

vecDeriv(1) = z(4);
vecDeriv(2) = z(5);
vecDeriv(3) = z(6);
vecDeriv(4) = 3*n^2*z(1)+2*n*z(5);
vecDeriv(5) = -2*n*z(4);
vecDeriv(6) = -n^2*z(3);
vecDeriv = vecDeriv.';
end
```

Problem 4;

```
clear;clc;close all;
mu = 3.9860044189e5;

ri = 7800;
options = odeset('RelTol',1e-5);
h = sqrt(ri*mu);
vp = h/ri;
[To,Zo] = ode45('OrbEq',[0 3600*5], [ri 0 0 0 vp 0],options);
hold on
t = linspace(0,2*pi);plot(6378*cos(t),6378*sin(t));

    k = 1;
    dt = 0.85*3600;
while 1

    mo = 1740;
    T = 0.8;
    Isp = 220;
    delta_IC = [0 0 0 0 0 0];
    rho_IC = [ri 0 0 0 vp 0];
    IC = [delta_IC rho_IC mo];
    [T,Z] = ode45('cowell',[0 dt],IC,options);

    r = [Z(end,1)+Z(end,7) Z(end,2)+Z(end,8) Z(end,3)+Z(end,9)];
    v = [Z(end,4)+Z(end,10) Z(end,5)+Z(end,11) Z(end,6)+Z(end,12)];
    [~,e,~,~,~,~] = stateVec2OrbElem(r,v);
    e = norm(e);

    dt = dt + .1 ;
    k = k + 1;
    if (abs(e - 0.6) < 0.0001 ) || (k == 25000)
        break
    end
end

[Tf,Zf] = ode45('OrbEq',[0 50*3600],[r v],options);
plot(Zo(:,1),Zo(:,2),'LineWidth',2,'Color','b'); grid on; axis equal
plot(Z(:,1)+Z(:,7),Z(:,2)+Z(:,8),'--','LineWidth',2,'Color','b')
plot(Zf(:,1),Zf(:,2),'r','LineWidth',2)
title('Orbital Displacement');xlabel('X-axis')
ylabel('Y-axis');
legend('Earth','Original Orbit','Oscultating Orbit','Final Orbit')
burnT = dt/3600;
dm = Z(1,13)-Z(end,13);
for i = 1:length(T)
    r(i) = norm(Zf(i,1),Zf(i,2));
end
ra = max(r)-6378;

fprintf('Burn Time %.4f hrs \n',burnT);
```

```
fprintf('Mass expended %.1f kg \n',dm);
fprintf('New apogee altitude %.1f m \n',ra);
```

```
Burn Time 0.8801 hrs
Mass expended 1174.5 kg
New apogee altitude 13954.1 m
```



Published with MATLAB® R2017a

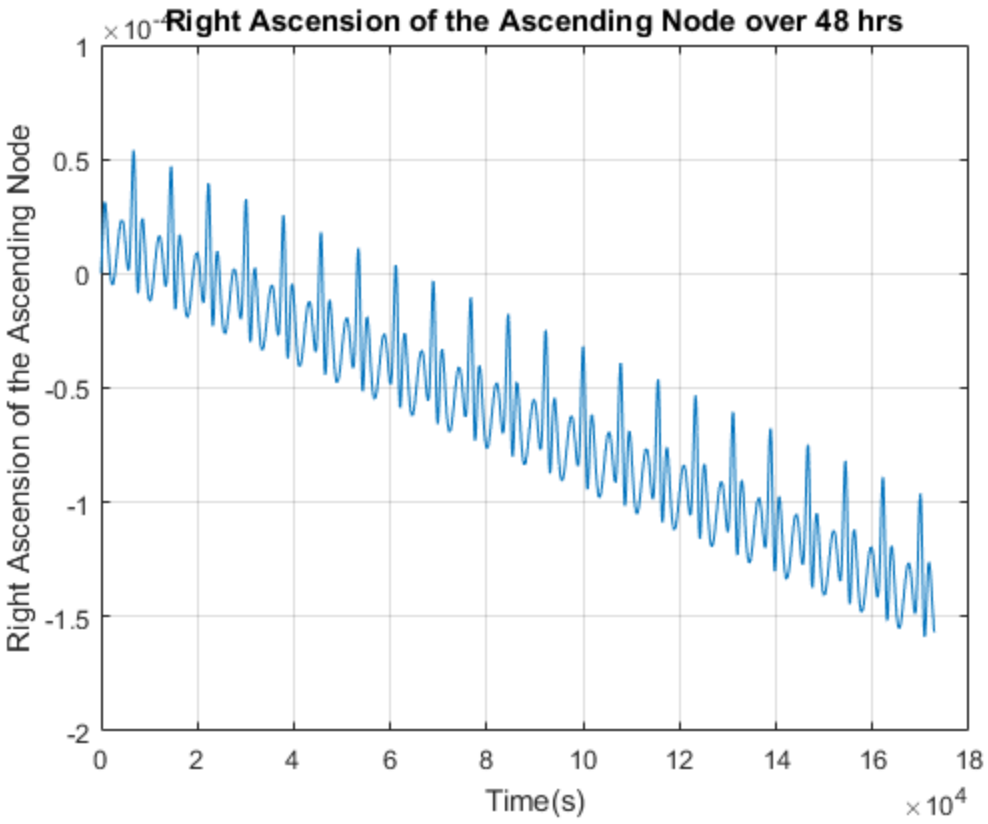
```
function [vecDeriv] = cowell(t,z)
mu = 3.9860044189e5;
mo = 1740;
T = 0.8;
Isp = 220;
g = 9.81;
rho = sqrt(z(7)^2+z(8)^2+z(9)^2);

phi = -(z(7)*z(1)+z(8)*z(2)+z(9)*z(3))/rho^2;
F = 1-(1-2*phi)^(-3/2);
rx = z(1)+z(7);
ry = z(2)+z(8);
rz = z(3)+z(9);
r = sqrt(rx^2+ry^2+rz^2);
vx = z(4)+z(10);
vy = z(5)+z(11);
vz = z(6)+z(12);
v = sqrt(vx^2+vy^2+vz^2);

vecDeriv(1) = z(4);
vecDeriv(2) = z(5);
vecDeriv(3) = z(6);
vecDeriv(4) = mu/rho^3*(F*rx-z(1))+ T*(vx)/(v*z(13));
vecDeriv(5) = mu/rho^3*(F*ry-z(2))+ T*(vy)/(v*z(13));
vecDeriv(6) = mu/rho^3*(F*rz-z(3))+ T*(vz)/(v*z(13));
vecDeriv(7) = z(10);
vecDeriv(8) = z(11);
vecDeriv(9) = z(12);
vecDeriv(10) = -mu/rho^3*z(7);
vecDeriv(11) = -mu/rho^3*z(8);
vecDeriv(12) = -mu/rho^3*z(9);
vecDeriv(13) = -T/(Isp*g)*1000;
vecDeriv = vecDeriv.';

end
```

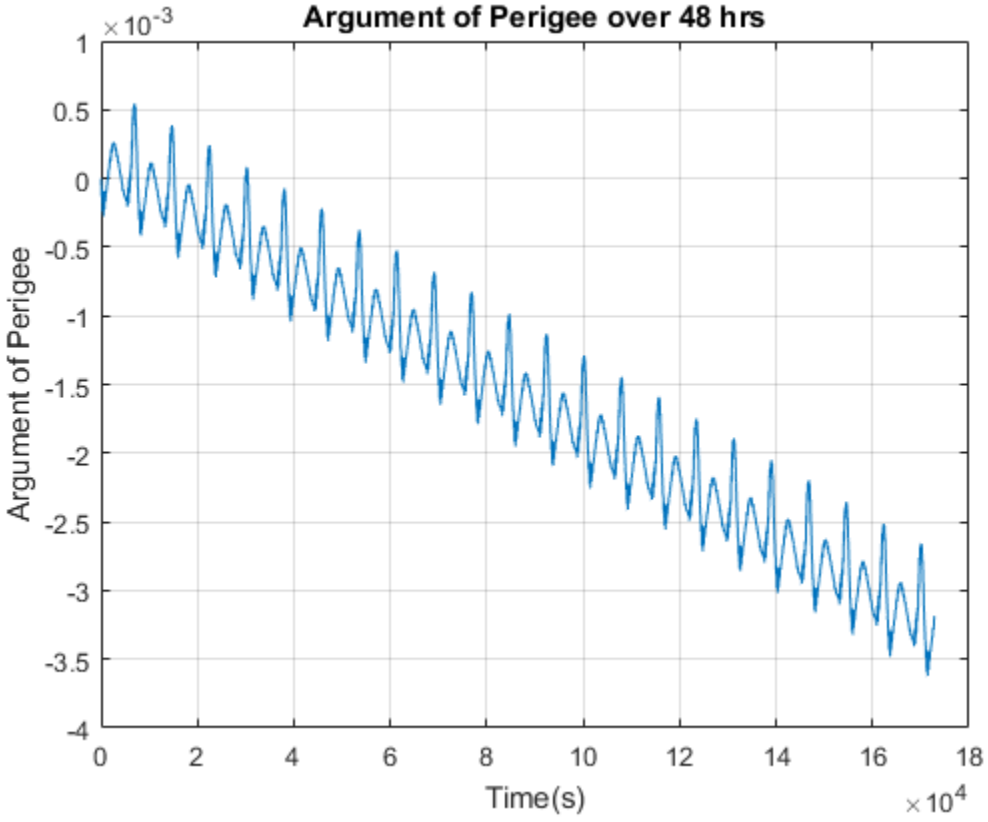
```
function [vecDeriv] = OrbEq(t,z)
mu = 3.9860044189e5;
r = sqrt(z(1)^2+z(2)^2+z(3)^2);
vecDeriv(1) = z(4);
vecDeriv(2) = z(5);
vecDeriv(3) = z(6);
vecDeriv(4) = -mu/r^3*z(1);
vecDeriv(5) = -mu/r^3*z(2);
vecDeriv(6) = -mu/r^3*z(3);
vecDeriv = vecDeriv.';
end
```

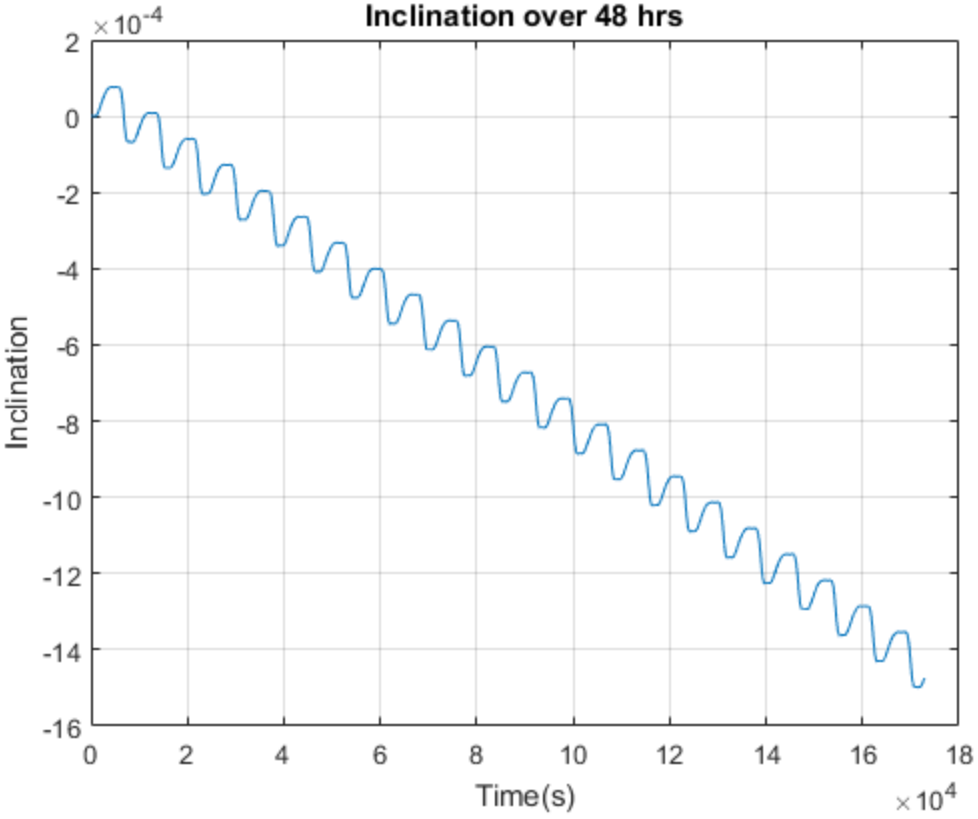
```
function [h,e,i,w,W,true_ano,N] = stateVec2OrbElem(r,v)
%This function will take two state vectors r and v and compute the
six orbital elements
% r and v must be 3-D vectors
% Currently this is only for Geocentric orbits
mu = 3.9860044189e5;

h = cross(r,v); % specific angular momentum
i = acos(h(3)/norm(h));  % inclination is hz divided by the
magnitude of h
N = cross([0 0 1],h); % line of nodes is the unit vector k cross h

    if N(2) >= 0
        % Right ascesnsion of the ascending Node
        W = acos(N(1)/norm(N));
    else
        W = 2*pi-acos(N(1)/norm(N));
    end
 % eccentricity vector
    vr = dot(r,v)/norm(r);
    e = ((norm(v)^2 - mu/norm(r))*r - norm(r)*vr*v)/mu;

    if e(3) >= 0
        % argument of perigee
        w = acos(dot(N,e)/(norm(N)*norm(e)));
    else
        w = 2*pi-acos(dot(N,e)/(norm(N)*norm(e)));
    end

    if vr >= 0
        %true anomaly
        true_ano = acos(dot(e,r)/(norm(e)*norm(r)));
    else
        true_ano = 2*pi-acos(dot(e,r)/(norm(e)*norm(r)));
    end


end
```

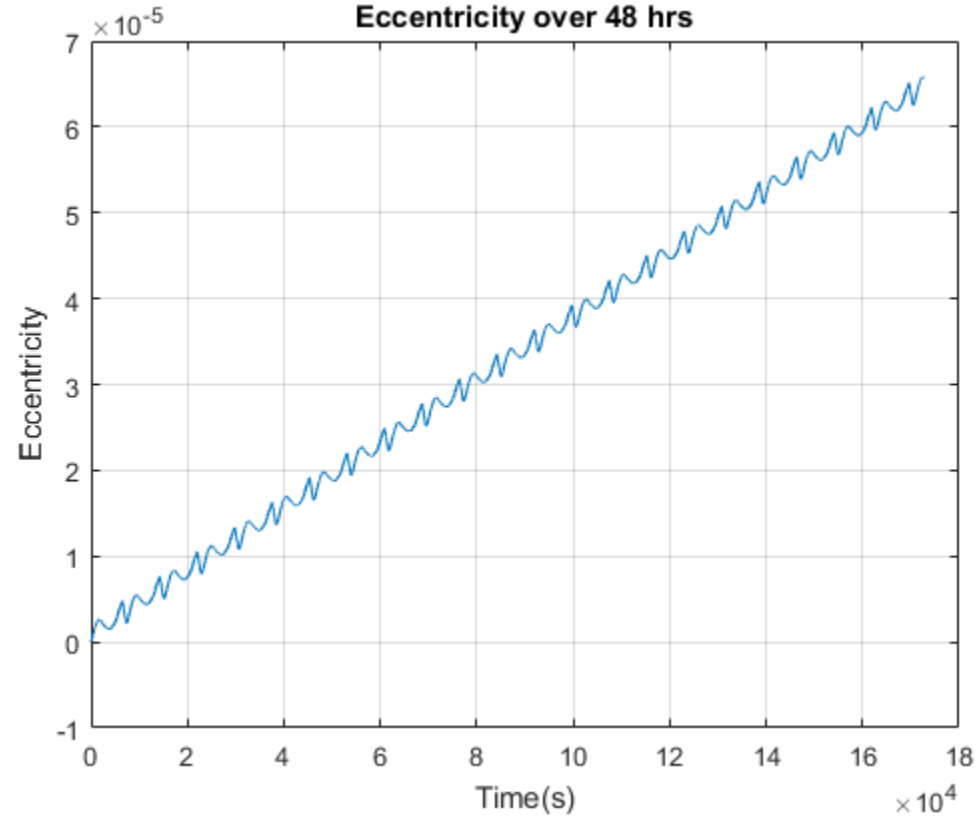
Problem 5;

```
clear;clc;close all;
mu = 3.9860044189e5;
Wo = 45*pi/180;
io = 30*pi/180;
wo = 30*pi/180;
thetao = 40*pi/180;
rp = 400+6378;
ra = 3800+6378;
dt = 48*3600;
eo = (ra-rp)/(ra+rp);
a = (rp+ra)/2;
ho = sqrt(a*mu*(1-eo^2));

[r,v] = OrbElem2StateVec(ho,eo,io,wo,Wo,thetao);

options = odeset('RelTol',1e-6);
[T,Z] = ode45('J3',[0 dt],[r v],options);
for k = 1:length(T)
    [h,e,i,w,W,true_ano] = stateVec2OrbElem(Z(k,(1:3)),Z(k,(4:6)));
    W_k(k) = W;
    w_k(k) = w;
    i_k(k) = i;
    e_k(k) = norm(e);
end

plot(T,W_k-Wo*180/pi); grid on
title('Right Ascension of the Ascending Node over 48 hrs')
xlabel('Time(s)');ylabel('Right Ascension of the Ascending Node')

figure
plot(T,w_k-wo*180/pi);grid on
title('Argument of Perigee over 48 hrs')
xlabel('Time(s)');ylabel('Argument of Perigee')

figure
plot(T,i_k-io*180/pi);grid on
title('Inclination over 48 hrs')
xlabel('Time(s)');ylabel('Inclination')

figure
plot(T,e_k-eo);grid on
title('Eccentricity over 48 hrs')
xlabel('Time(s)');ylabel('Eccentricity')
```


Right Ascension of the Ascending Node over 48 hrs
×10^-4
Right Ascension of the Ascending Node
Time(s)
×10^4


Argument of Perigee over 48 hrs
×10^-3
Argument of Perigee
Time(s)
×10^4


Inclination over 48 hrs
×10^-4
2
0
-2
-4
-6
-8
-10
-12
-14
-16
Inclination
0
2
4
6
8
10
12
14
16
18
Time(s)
×10^4


Eccentricity over 48 hrs
×10^-5
7
6
5
4
3
2
1
0
-1
Eccentricity
0
2
4
6
8
10
12
14
16
18
Time(s)
×10^4

```
function [vecDeriv] = J3(t,z)

mu = 3.9860044189e5;
r=sqrt(z(1)^2+z(2)^2+z(3)^2);
R = 6378;
J2 = 0.00108263;
J3 = -2.33936*10^(-3)*J2;

C = (1/2)*((J3*mu*R^3)/r^5);

apx = C*((5*z(1)/r)*((7*z(3)^3/r^3)-(3*z(3)/r)));
apy = C*((5*z(2)/r)*((7*z(3)^3/r^3)-(3*z(3)/r)));
apz = C*((35*z(3)^4/r^4)-(30*z(3)^2/r^2)+3);

vecDeriv(1) = z(4);
vecDeriv(2) = z(5);
vecDeriv(3) = z(6);
vecDeriv(4) = -mu/r^3*z(1) + apx;
vecDeriv(5) = -mu/r^3*z(2) + apy;
vecDeriv(6) = -mu/r^3*z(3) + apz;

vecDeriv = vecDeriv';

end
```

```
function [RX,VX,r,v,QXx] = OrbElem2StateVec(h,e,i,w,W,true_ano)
%This function will take orbital elements and compute two state vectors r
and v
% r and v must be 3-D vectors
mu = 3.9860044189e5;
r = (h^2/mu)/(1+e*cos(true_ano))*[cos(true_ano) sin(true_ano) 0]';
v = mu/h*[-sin(true_ano) (e+cos(true_ano)) 0]';
R3_W = [cos(W)  sin(W) 0;
      -sin(W) cos(W) 0;
      0       0      1;];
R1_i = [1 0 0;
        0 cos(i) sin(i);
        0 -sin(i) cos(i);];
R3_w = [ cos(w) sin(w) 0;
        -sin(w) cos(w) 0
        0 0 1;];
QXx = (R3_w) *(R1_i) *(R3_W);

RX = QXx.'*r;
VX = QXx.'*v;

end
```

```
function [h,e,i,w,W,true_ano,N] = stateVec2OrbElem(r,v)
%This function will take two state vectors r and v and compute the six
orbital elements
% r and v must be 3-D vectors
% Currently this is only for Geocentric orbits
mu = 3.9860044189e5;

h = cross(r,v); % specific angular momentum
i = acosd(h(3)/norm(h));  % inclination is hz divided by the magnitude
of h
N = cross([0 0 1],h); % line of nodes is the unit vector k cross h

    if N(2) >= 0
        % Right ascesnsion of the ascending Node
        W = acosd(N(1)/norm(N));
    else
        W = 360-acosd(N(1)/norm(N));
    end
 % eccentricity vector
    vr = dot(r,v)/norm(r);
    e = ((norm(v)^2 - mu/norm(r))*r - norm(r)*vr*v)/mu;

    if e(3) >= 0
        % argument of perigee
        w = acosd(dot(N,e)/(norm(N)*norm(e)));
    else
        w = 360-acosd(dot(N,e)/(norm(N)*norm(e)));
    end

    if vr >= 0
        %true anomaly
        true_ano = acosd(dot(e,r)/(norm(e)*norm(r)));
    else
        true_ano = 360-acosd(dot(e,r)/(norm(e)*norm(r)));
    end


end
```