Question 1:

function [posT,velT,timeT,dV1,dV2]=twoImpulseHohmann(r1,r2,i1,i2,Omega1,Omega2,mu)

%This function calculates the two impulses needed to complete a Hohmann

%transfer

%Input: r1, radius of orbit 1

%Input: r2, radius of orbit 2

%Input: i1, inclination of orbit 1

%Input: i2, inclination of orbit 2

%Input: Omega1, longitude of ascending node of orbit 1

%Input: Omega1, longitude of ascending node of orbit 1

%Input: mu, gravitaional parameter

%Output: posT, 500x3 matrix containing postions along the transfer orbit

%Output: timeT, column matrix of times associated with positions in posT

%Output: dV1, impulse at periapsis of transfer

%Output: dV2, impulse at apoapsis of transfer

nu=0:0.001:2\*pi;

e=0;

omega=0;

oe1=[r1,e,Omega1,i1,omega,0];

oe2=[r2,e,Omega2,i2,omega,0];

position1=zeros(length(nu),3);

velocity1=zeros(length(nu),3);

position2=zeros(length(nu),3);

velocity2=zeros(length(nu),3);

for i=1:length(nu)

oe1(6)=nu(i);

oe2(6)=nu(i);

[r,v]=oe2rv\_Hackbardt\_Chris(oe1,mu);

position1(i,:)=r';

velocity1(i,:)=v';

[r,v]=oe2rv\_Hackbardt\_Chris(oe2,mu);

position2(i,:)=r';

velocity2(i,:)=v';

end

v1=sqrt(mu/r1);

v2=sqrt(mu/r2);

hvec1=cross(position1(1,:),velocity1(1,:));

hvec2=cross(position2(1,:),velocity2(1,:));

lvec=cross(hvec1,hvec2);

lvec=lvec/norm(lvec);

position1T=r1\*lvec;

aT=(r1+r2)/2;

deltaV1=sqrt(((2\*mu)/r1)-(mu/aT))-v1;

u1vec=cross(hvec1,lvec)/norm(hvec1);

V1b4=v1\*u1vec;

dV1=deltaV1\*u1vec;

V1aft=V1b4+dV1;

M=500;

tauT=2\*pi\*sqrt(aT^3/mu);

timeT=linspace(0,tauT/2,M);

posT=zeros(length(timeT),3);

velT=zeros(length(timeT),3);

for i=1:length(timeT)

[r,v] = propagateKepler\_Hackbardt\_Chris(position1T,V1aft,0,timeT(i),mu);

posT(i,:)=r';

velT(i,:)=v';

end

v2bf=sqrt(((2\*mu)/r2)-(mu/aT));

V2b4=-v2bf\*u1vec;

u2vec=-cross(hvec2,lvec)/norm(hvec2);

V2aft=v2\*u2vec;

dV2=V2aft-V2b4;

end

Question 2:

function [posT,timeT,dVP,dVA,totDV]=twoNImpulseOrbitTransfer(r1,r2,i1,i2,Omega1,Omega2,N,mu)

%This function calculates the impulses needed to complete N Hohmann

%transfers

%Input: r1, radius of orbit 1

%Input: r2, radius of orbit 2

%Input: i1, inclination of orbit 1

%Input: i2, inclination of orbit 2

%Input: Omega1, longitude of ascending node of orbit 1

%Input: Omega1, longitude of ascending node of orbit 1

%Input: N, number of Ho

%Input: mu, gravitaional parameter

%Output: posT, 500x3 matrix containing postions along the transfer orbit

%Output: timeT, column matrix of times associated with positions in posT

%Output: dV1, impulse at periapsis of transfer

%Output: dV2, impulse at apoapsis of transfer

%Output: totDV, the total delta V required to complete the transfer

rn=r1+((1:N)/N)\*(r2-r1);

Omegan=Omega1+((1:N)/N)\*(Omega2-Omega1);

in=i1+((1:N)/N)\*(i2-i1);

rn=[r1,rn];

Omegan=[Omega1,Omegan];

in=[i1,in];

dOmega=Omegan(2)-Omegan(1);

dVP=zeros(N,1);

dVA=zeros(N,1);

posT=[];

velT=[];

timeT=[0];

N=1:N;

for i=1:length(N)

%elliptic

[post,velt,timet,dV1,dV2]=twoImpulseHohmann(rn(i),rn(i+1),in(i),in(i+1),Omegan(i),Omegan(i+1),mu);

posT=[posT;post];

velT=[velT;velt];

timet=timet+timeT(length(timeT));

timeT=[timeT;timet'];

dVP(i)=norm(dV1);

dVA(i)=norm(dV2);

%circular

if i<length(N)

taut=2\*pi\*sqrt(rn(i+1)^3/mu);

dOmegaTime=dOmega/sqrt(mu/rn(i+1)^3);

[timet, post] = propagateOnCircle(post(end,:),velt(end,:)+dV2,timeT(end),timeT(end)+(taut/2)+dOmegaTime,mu,500);

posT=[posT;post];

timeT=[timeT;timet];

end

end

timeT(1)=[];

totDV=sum(dVP)+sum(dVA);

fprintf('The total time to complete the transfer is %g seconds or %g hours\n',timeT(end),timeT(end)/3600)

%Plots 3D view of transfer

figure

hold on

earthSphere

plot3(posT(:,1),posT(:,2),posT(:,3))

hold off

%Plots the ground track;

OmegaE = (2\*pi)/((23\*3600)+(56\*60)+4);

rvValuesECEF = eci2ecef(timeT,posT,OmegaE);

[lonE,lat] = ecef2LonLat(rvValuesECEF);

figure

earth = imread('earth.jpg');

hold on

image('CData',earth,'XData',[-180 180],'YData',[90 -90])

plot(lonE\*180/pi,lat\*180/pi,'r\*');

end

Question 3:

clc;clear;

mu=398600;

N=[1,2,3,4,5,6];

r1=500+6378.145;

i1=28.5;

Omega1=60;

r2=20200+6378.145;

i2=57;

Omega2=120;

i1=deg2rad(i1);

i2=deg2rad(i2);

Omega1=deg2rad(Omega1);

Omega2=deg2rad(Omega2);

dVAs=[];

dVPs=[];

totdVs=[];

totTimes=[];

for i=1:length(N)

[posT,timeT,dVP,dVA,totDV]=twoNImpulseOrbitTransfer(r1,r2,i1,i2,Omega1,Omega2,N(i),mu);

dVAs(i)=sum(dVA);

dVPs(i)=sum(dVP);

totdVs(i)=totDV;

totTimes(i)=timeT(end);

end

figure

plot(N,dVAs)

title('Magnitude of Apoapsis Impulses vs N')

xlabel('N')

ylabel('Apoapsis Impulses (km/s)')

set(gca, 'XTick', 0:N(end))

figure

plot(N,dVPs)

title('Magnitude of Periapsis Impulses vs N')

xlabel('N')

ylabel('Periapsis Impulses (km/s)')

set(gca, 'XTick', 0:N(end))

figure

plot(N,totdVs)

title('Magnitude of Total Impulse vs N')

xlabel('N')

ylabel('Total Impulse (km/s)')

set(gca, 'XTick', 0:N(end))

figure

plot(N,totTimes)

title('Total Time Required for Transfer vs Time')

xlabel('N')

ylabel('Time (sec)')

set(gca, 'XTick', 0:N(end))

The total time to complete the transfer is 10766 seconds or 2.99055 hours

The total time to complete the transfer is 34796 seconds or 9.66556 hours

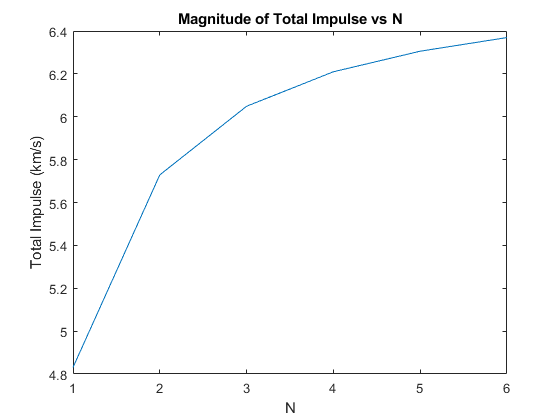
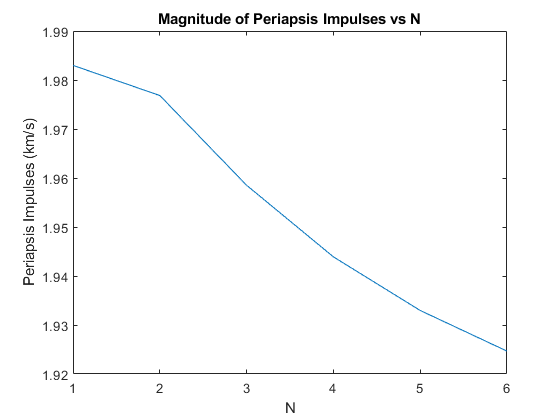
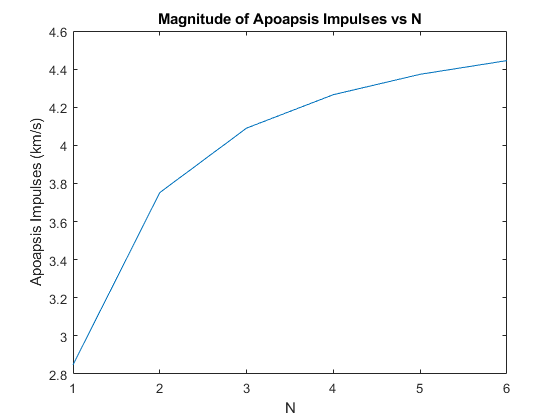
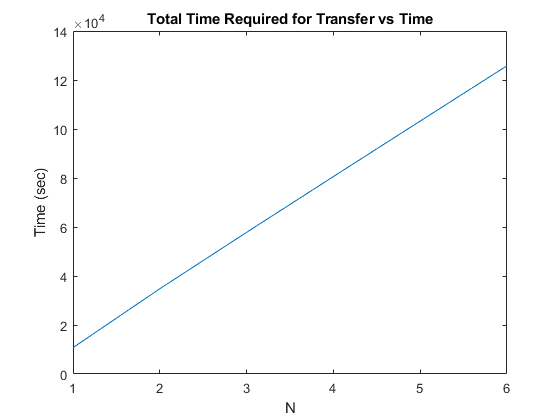
The total time to complete the transfer is 57825.5 seconds or 16.0626 hours

The total time to complete the transfer is 80586 seconds or 22.385 hours

The total time to complete the transfer is 103236 seconds or 28.6766 hours

The total time to complete the transfer is 125829 seconds or 34.9526 hours

The total impulse required increases as a function of N because it is inefficient to change the plane of an orbit at a lower orbit than the final, highest orbit. So as N increases, more of the plane changes are completed at lower orbits.



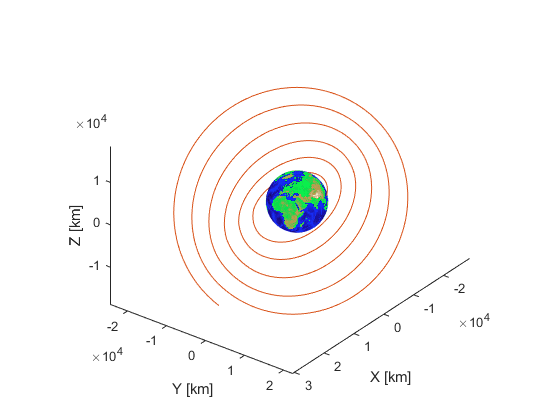
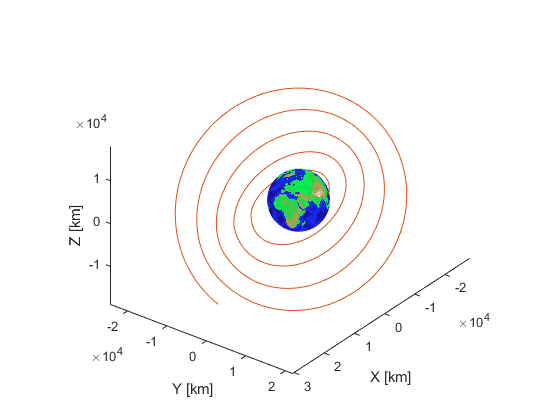
3D Views:

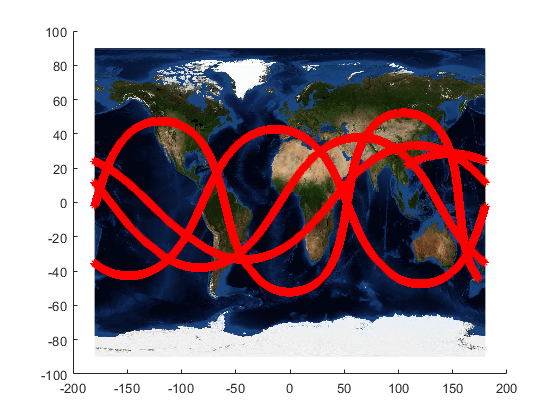
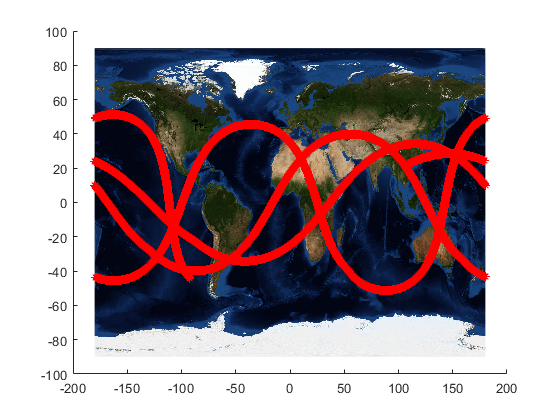
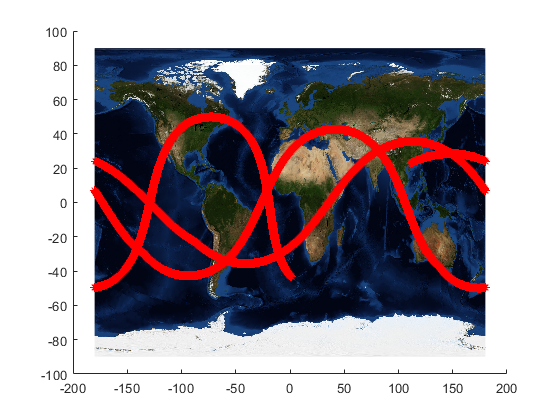
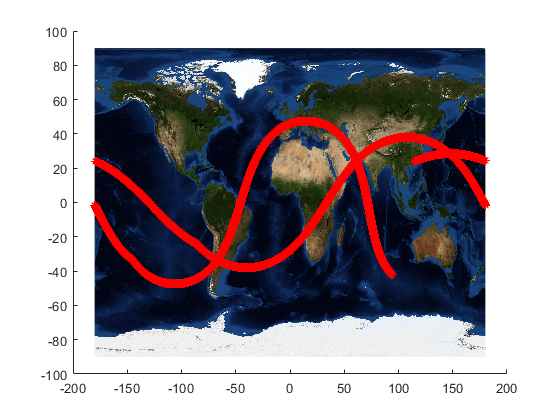
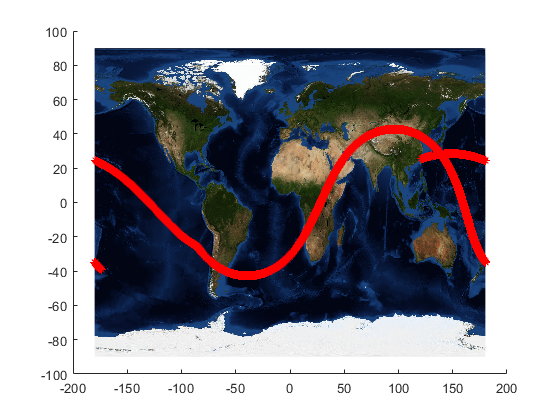
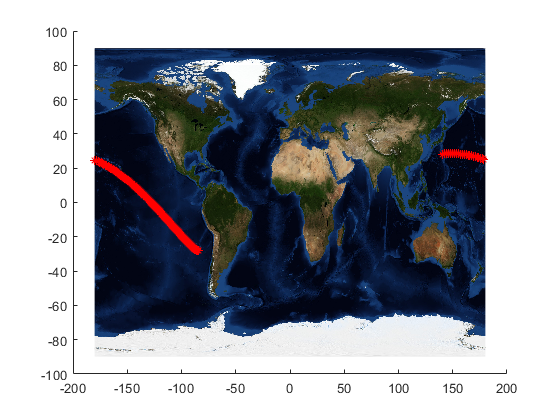
Chart

Description automatically generatedChart

Description automatically generatedDiagram

Description automatically generatedChart, diagram

Description automatically generated

Groundtracks:

Additional Code Used:

function oe = rv2oe\_Hackbardt\_Chris(rPCI,vPCI,mu)

% This function calculates the six orbital elements from the position vector and velocity vector

% Function Call: oe = rv2oe\_Hackbardt\_Chris(rPCI,vPCI,mu)

%

% Inputs:

% rPCI: Cartesian planet-centered inertial (PCI) position (3 by 1)

% vPCI: Cartesian planet-centered inertial (PCI) velocity (3 by 1)

% mu: gravitational parameter of centrally attacting body

% Outputs: orbital elements

% oe(1): semi-major axis

% oe(2): eccentricity

% oe(3): longitude of the ascending node (rad)

% oe(4): inclination (rad)

% oe(5): argument of the periapsis (rad)

% oe(6): true anomaly (rad)

rvec = rPCI;

vvec = vPCI;

Ix = [1;0;0];

Iy = [0;1;0];

Iz = [0;0;1];

hvec = cross(rvec,vvec);

h = norm(hvec);

p = h^2/mu;

r = norm(rvec);

evec = ((cross(vvec,hvec))/mu)-(rvec/r);

e = norm(evec);

a = p/(1-e^2);

nvec = cross(Iz,hvec);

n = norm(nvec);

Omega = atan2(dot(nvec,Iy),dot(nvec,Ix));

if Omega < 0

Omega = Omega + (2\*pi);

end

inc = atan2(dot(hvec,cross(nvec,Iz)),n\*dot(hvec,Iz));

omega = atan2(dot(evec,cross(hvec,nvec)),h\*dot(evec,nvec));

if omega < 0

omega = omega + (2\*pi);

end

nu = atan2(dot(rvec,cross(hvec,evec)),h\*dot(rvec,evec));

if nu < 0

nu = nu + (2\*pi);

end

oe = [a; e; Omega; inc; omega; nu];

end

function [rPCI,vPCI] = oe2rv\_Hackbardt\_Chris(oe,mu)

% This function calculates the postion and velocity vector with respect to the planet using orbital elements.

% Function Call: [rPCI,vPCI] = oe2rv\_Hackbardt\_Chris(oe,mu)

% Input: Orbital Elements: (6 by 1 column vector)

% oe(1): Semi-major axis

% oe(2): Eccentricity

% oe(3): Longitude of the ascending node (rad)

% oe(4): Inclination (rad)

% oe(5): Argument of the periapsis (rad)

% oe(6): True anomaly (rad)

% mu: Planet gravitational parameter (scalar)

% Outputs:

% rPCI: Planet-Centered Inertial (PCI) Cartesian position

% (3 by 1 column vector)

% vPCI: Planet-Centered Inertial (PCI) Cartesian inertial velocity

% (3 by 1 column vector)

a = oe(1);

e = oe(2);

Omega = oe(3);

i = oe(4);

omega = oe(5);

nu = oe(6);

p = a\*(1-e^2);

r = p/(1+(e\*cos(nu)));

rvecP = [r\*cos(nu);r\*sin(nu);0];

vvecP = (sqrt(mu/p))\*[-sin(nu);e+cos(nu);0];

T\_NI = [cos(Omega),-sin(Omega),0;sin(Omega),cos(Omega),0;0,0,1];

T\_QN = [1,0,0;0,cos(i),-sin(i);0,sin(i),cos(i)];

T\_PQ = [cos(omega),-sin(omega),0;sin(omega),cos(omega),0;0,0,1];

T\_PI = T\_NI \* T\_QN \* T\_PQ;

rPCI = T\_PI \* rvecP;

vPCI = T\_PI \* vvecP;

end

function E = KeplerSolver\_Hackbardt\_Chris(a,e,mu,t0,t,E0,k)

%This function solves Kepler's equation using fixed point iteration

%

%Input: a, semi-major axis

%Input: e, eccentricity

%Input: mu, gravitational parameter

%Input: t0, the initial time in seconds

%Input: t, the final time in seconds

%Input: E0, initial eccentric anamoly

%Input: k, number of periapsis crossings

%

%Output: E, eccentric anamoly

C=sqrt(mu/a^3)\*(t-t0)-(2\*pi\*k)+(E0-(e\*sin(E0)));

f = @(x) e\*sin(x)+C;

n=10\*ceil(1/(1-e));

guessE=E0-(e\*sin(E0));

for i=1:n

guessE=f(guessE);

end

E=guessE;

end

function [r,v,E,nu] = propagateKepler\_Hackbardt\_Chris(r0,v0,t0,t,mu)

%This function finds the position and velocity in an orbit at final time, t

%

%Input: r0, a column vector with the initial position

%Input: v0, a column vector with the initial velocity

%Input: t0, the initial time in seconds

%Input: t, the final time in seconds

%Input: mu, gravitational parameter

%

%Output: r, a column vector with the position

%Output: v, a column vector with the velocity

%Output: E, eccentric anamoly

%Output: nu, true anamoly

oe = rv2oe\_Hackbardt\_Chris(r0,v0,mu);

a = oe(1);

e = oe(2);

nu0 = oe(6);

E0 = 2\*atan2(sqrt(1-e)\*sin(nu0/2),sqrt(1+e)\*cos(nu0/2));

tau=2\*pi\*sqrt(a^3/mu);

k=floor((t)/tau);

E = KeplerSolver\_Hackbardt\_Chris(a,e,mu,t0,t,E0,k);

nu= 2\*atan2(sqrt(1+e)\*sin(E/2),sqrt(1-e)\*cos(E/2));

oe(6)=nu;

[r,v] = oe2rv\_Hackbardt\_Chris(oe,mu);

end

function [times, positions, velocities] = propagateOnCircle(pos,vel,t0,tf,mu,N)

%This function calculates position and veolocity N times evenly spaced betwen t0 and tf

%

%Function call: [times, postions, velocities] = propagateOnCircle(pos,vel,t0,tf,mu,N);

%

%Input: pos, a column vector with the initial ECI postion

%Input: vel, a column vector with the initial ECI inertial velocity

%Input: t0, the initial time. Time from apoapsis to initial postion

%Input: tf, the final time

%Input: mu, gravitational parameter

%Input: N, number of time intervals

%

%Output: times, column vector of times, starting at t0 and ending at tf. Length of N with even spacing

%Output: postions, matrix of size Nx3 containing ECI positions stored row-wise at every time in the times vector

%Output: velocities, matrix of size Nx3 containing ECI velocities stored row-wise at every time in the times vector

%

times=linspace(t0,tf,N);

times=times';

oe = rv2oe\_Hackbardt\_Chris(pos,vel,mu);

nui=oe(6);

a=oe(1);

thetaDot=sqrt(mu/a^3);

positions=zeros(N,3);

velocities=zeros(N,3);

positions(1,:)=pos';

velocities(1,:)=vel';

for i=2:N

theta=thetaDot\*(times(i)-times(i-1));

nuOfT=nui+theta;

oe(6)=nuOfT;

[rPCI,vPCI] = oe2rv\_Hackbardt\_Chris(oe,mu);

positions(i,:)=rPCI';

velocities(i,:)=vPCI';

nui=nuOfT;

end

end