Table of Contents

 1
Load Data 1
Propagate constelation in time for a mean solar day in 30 sec time steps 1
Compute the number of satellites in line of sight of city i at each timestep 3
Plot orbits at the final time step 6
Functions 7


```
%Orbital Final Project Main Script Part 1
%Zak Reichenbach
%11/23/21

%House Keeping
clc
clear all
close all
```

Load Data

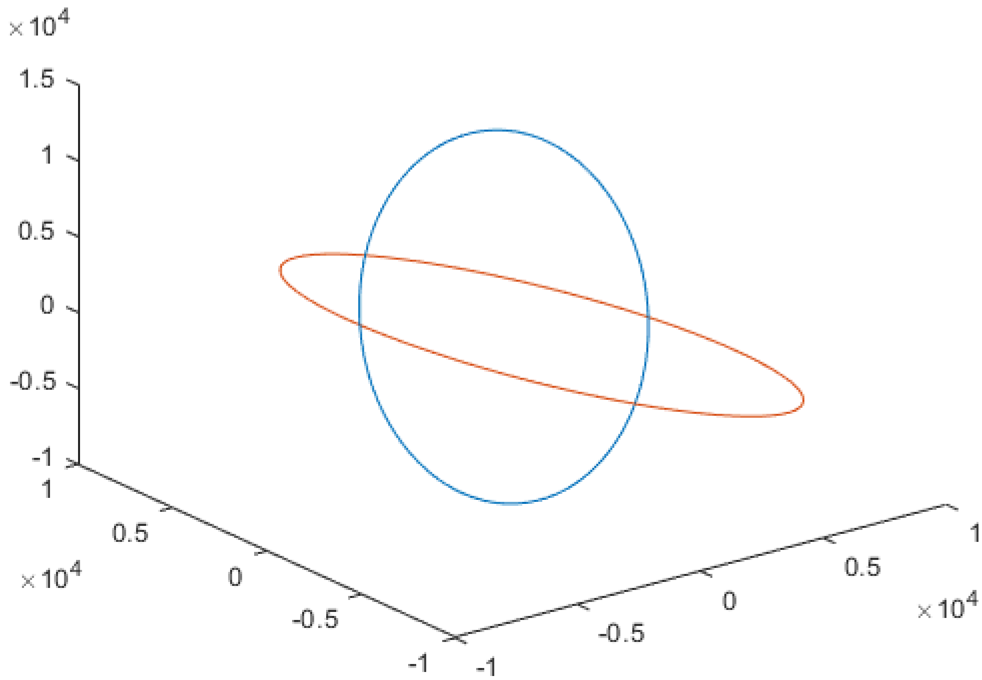
```
filename = 'example_constellation.json';
[num_launches, num_spacecraft, satellite_list] =
 loadConstellation(filename);

cities = readtable('worldcities.xlsx');
coasts = readtable('world_coastline_low.txt');
```

Propagate constelation in time for a mean solar day in 30 sec time steps

```
%in ECI frame
Re = 6371;
MU = 0.39860*10^6;
t0 = 0;
t = 0:30:24*(60)^3;
J2 = 1.087e-3;



for j = 1:length(satellite_list)
% oe0       Orbit elements [a,e,i,Om,om,f] at time t0 (km,s,rad)
Oe0(j,:) = cell2mat(struct2cell(satellite_list(j).oe0));
    k = 6*j;

    if j == 1
        W = 1;
    else
```

```
        W = 1 + (6*(j-1));
    end
    fprintf('Propagating Orbit %f\n',j)

    for i = 1:length(t)
    x(W:k,i) = propagateState(Oe0(j,:),t(i),t0,MU,J2,Re); %x =
 [r,rdot]
    end

end

%Plot Orbits
    plot3(x(1,:),x(2,:),x(3,:))
hold on
    plot3(x(7,:),x(8,:),x(9,:))


%Cartesian Coordinates of Cities and Coastlines

for k = 1:height(cities)
   [xtmp(k),ytmp(k),ztmp(k)] =
 sph2cart(deg2rad(table2array(cities(k,4))),deg2rad(table2array(cities(k,3))),6371
end

for k = 1:height(coasts)
   [xCoast(k),yCoast(k),zCoast(k)] =
 sph2cart(deg2rad(table2array(coasts(k,1))),deg2rad(table2array(coasts(k,2))),6371
end
```

Propagating Orbit 1.000000
Propagating Orbit 2.000000



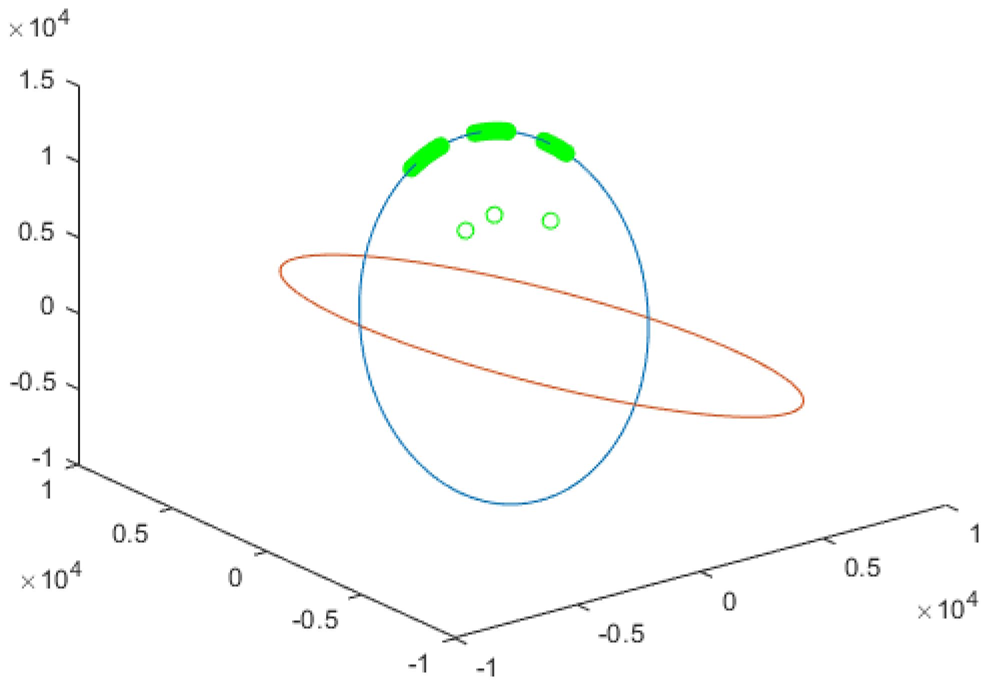
Compute the number of satellites in line of sight of city i at each timestep

```
%Limiting View angle

elevation_limit = 15;

%Test for plotting spot on surface and orbit
Q = 1;

%xtmp = cities
%x = orbits

%We are doing line of site for a limited number of cities

hold on

NumOfCities = 40;
Ratio = round(length(xtmp)/NumOfCities);

%Outter loop picks the city
counterI = 1;
counterJ = 1;
for i = 1:Ratio:length(xtmp)
```

```
    fprintf('On city %f\n',counterI)
    %Inner loop scans for line of sight
    counterJ = 1;
    for j = 1:length(t)
        r_site = [xtmp(i),ytmp(i),ztmp(i)];
        r_sc = [x(1,j),x(2,j),x(3,j)];
        inLoS = testLoS(r_site,r_sc,elevation_limit);

        if inLoS == 1
            count(counterI,counterJ) = 1;
%             fprintf('GOT ONE')
        plot3(x(1,j),x(2,j),x(3,j),'go')

        plot3(xtmp(i),ytmp(i),ztmp(i),'go')
        else
            count(counterI,counterJ) = 0;
        end

        r_sc = [x(7,j),x(8,j),x(9,j)];
        inLoS = testLoS(r_site,r_sc,elevation_limit);

        if inLoS == 1
            count(counterI,counterJ) = 2;
%             fprintf('GOT ONE')
        elseif inLoS == 1 && count(counterI,counterJ) == 0
            count(counterI,counterJ) = 1;
        plot3(x(1,j),x(2,j),x(3,j),'go')

        plot3(xtmp(i),ytmp(i),ztmp(i),'go')
        end
        counterJ = counterJ+1;

    end
    counterI = counterI+1;
end
```

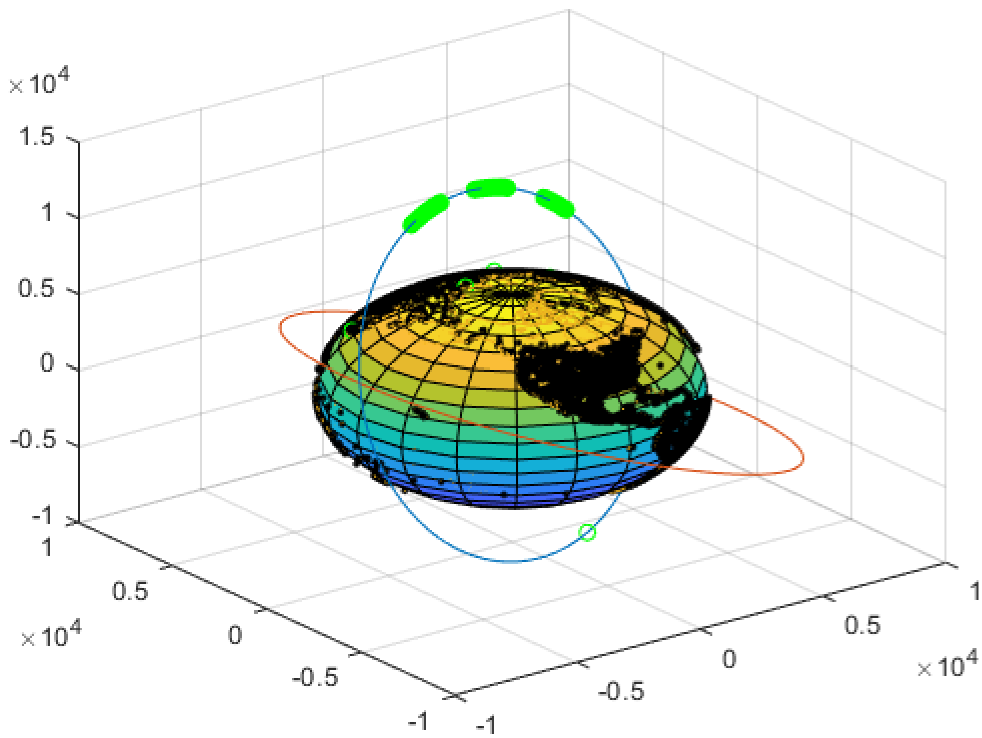
```
On city 1.000000
On city 2.000000
On city 3.000000
On city 4.000000
On city 5.000000
On city 6.000000
On city 7.000000
On city 8.000000
On city 9.000000
On city 10.000000
On city 11.000000
On city 12.000000
On city 13.000000
On city 14.000000
On city 15.000000
On city 16.000000
On city 17.000000
On city 18.000000
```

```
On city 19.000000
On city 20.000000
On city 21.000000
On city 22.000000
On city 23.000000
On city 24.000000
On city 25.000000
On city 26.000000
On city 27.000000
On city 28.000000
On city 29.000000
On city 30.000000
On city 31.000000
On city 32.000000
On city 33.000000
On city 34.000000
On city 35.000000
On city 36.000000
On city 37.000000
On city 38.000000
On city 39.000000
On city 40.000000
On city 41.000000
```



Plot orbits at the final time step

```
R = 6300;

[X,Y,Z] = sphere;
X = X*R;
Y = Y*R;
Z = Z*R;

figure(1)
surf(X,Y,Z);
grid on

hold on



plot3(xtmp,ytmp,ztmp,'k.')

plot3(xCoast,yCoast,zCoast)

plot3(x(1,1),x(2,1),x(3,1),'go')

plot3(xtmp(Q),ytmp(Q),ztmp(Q),'go')
```



Functions

```
function [num_launches, num_spacecraft, satellite_list] =
 loadConstellation(filename)
%DESCRIPTOIN: Ingests constellation description .json file and parses
 it
%into a list of structs with full initial orbit elements (km, s, rad)
 and
%satellite name.
%
%INPUTS:
% filename      A string indicating the name of the .json file to be
 parsed
%
%OUTPUTS:
% nl            Number of total launches
% ns            Total number of spacecraft between all launches
% satlist       Array of structs with 'name' and 'oe0' properties


%Temporary - just so the function runs the first time you use it.
%You'll need to change all of these!
num_launches = 0;
num_spacecraft = 0;
satellite_list.name = '';
satellite_list.oe0 = NaN(6,1);

fid = fopen(filename);
raw = fread(fid,Inf);
str = char(raw');
fclose(fid);
val = jsondecode(str);

% val.launches(1);
% val.launches(2);
%
%
% val.launches(1).payload;
% val.launches(2).payload;
%
% val.launches(1).launchName;
% val.launches.orbit;

%2) read all of the launches and payloads to understand how many
 launches
% and spacecraft are in the constellation; note, this will be useful
 in
% Part 2!]
    num_spacecraft = 0;

for i = 1:length(val.launches)
    num_launches = i;
    num_spacecraft = num_spacecraft + length(val.launches(i).payload);
```

```
end

%3) RECOMMENDED: Pre-allocate the satellite_list struct
satellite_list = struct();

%4) Populate each entry in the satellite struct list with its name and
%initial orbit elements [a,e,i,Om,om,f] at time t0

    for j = 1:length(val.launches)
    satellite_list(j).name = val.launches(j).launchName;
    satellite_list(j).oe0 = val.launches(j).orbit;

    end


satellite_list;

end


function x = propagateState(oe0,t,t_0,MU,J2,Re)
%DESCRIPTION: Computes the propagated position and velocity in km, km/
s
%accounting for approximate J2 perturbations
%
%INPUTS:
% oe0       Orbit elements [a,e,i,Om,om,f] at time t0 (km,s,rad)
% t         Current time (s)
% t0        Time at the initial epoch (s)
% MU        Central body's gravitational constant (km^3/s^2)
% J2        Central body's J2 parameter (dimensionless)
% Re        Radius of central body (km)
%
%OUTPUTS:
% x         Position and velocity vectors of the form [r; rdot] (6x1)
 at
%             time t



%1) Compute the mean orbit elements oe(t) at time t due to J2
 perturbations
a = oe0(1);
e = oe0(2);
i = oe0(3);
Om0 = oe0(4);
om0 = oe0(5);

n = sqrt(MU/a^3);

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

Omega_dot = -3/2*n*J2*(Re/p)^2*cos(i);
omega_dot = 3/2*n*J2*(Re/p)^2*(2-5/2*sin(i)^2);
```

```
Omega = Om0 + Omega_dot;
omega = om0 + omega_dot;

M = n*(t-t_0);


%Now the perifocal frame

M1 = [1 0 0; 0 cos(i) sin(i); 0 -sin(i) cos(i)];

M3 = [cos(Omega) sin(Omega) 0; -sin(Omega) cos(Omega) 0; 0 0 1];
M32 = [cos(omega) sin(omega) 0; -sin(omega) cos(omega) 0; 0 0 1];

NP = M32*M1*M3;
PN = NP';


%Newtons Method With Keplers Equation
%2) Solve the time-of-flight problem to compute the true anomaly at
 time t
error = 1e-8;

if M > pi
    E = M+e/2;
else
    E = M-e/2;
end

ratio = 1;
while abs(ratio) > error
    ratio = (E - e*sin(E) - M)/(1 - e*cos(E));
    E = E - ratio;
end

     f = 2*atan(sqrt((1+e)/(1-e))*tan(E/2));

%3) Compute r(t), rdot(t) in the perifocal frame

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

rE = r*cos(f);

rP = r*sin(f);

h = sqrt(MU*p);

f_dot = h/r^2;

vE = sqrt(MU/p)*-sin(f);
vP = sqrt(MU/p)*(e+cos(f));

%4) Compute r(t), rdot(t) in the ECI frame, save into x
R = PN * [rE;rP;0];
```

```
V = PN * [vE;vP;0];
%make sure that function has outputs
x = [R;V];


end



function inLoS = testLoS(r_site,r_sc,elevation_limit)
%DESCRIPTION: Determines whether the spacecraft is within line-of-
sight
%(LoS) of the site given an elevation limit
%
%INPUT:
% r_site            The position vector of the site (km, 3x1)
% r_sc              The position vector of the spacecraft (km, 3x1)
% elevation_limit   Lower elevation limit (above the horizon) (rad)
%
%OUTPUT:
% inLoS             A boolean flag (0 or 1); 1 indicates the
 spacecraft and
%                   the site have line-of-sight

r_tar = r_sc-r_site;

theta = acos(dot(r_tar,r_site)/(norm(r_tar)*norm(r_site)));


%placeholder - delete this

if theta <= deg2rad(elevation_limit)
    inLoS = 1;
elseif theta > deg2rad(elevation_limit)
    inLoS = 0;
end
end

%1) Compute whether the site and spacecraft have line of sight (hint,
 I
%suggest drawing a picture and writing this constraint as an
 inequality
%using a dot product)
```

Published with MATLAB® R2021a