ASEN 3200 Orbital Project Part 1

Table of Contents

Problem Statement:

Author: Caleb Bristol Collaborators: N/A Date: 12/01/21

Workspace Cleaning

```
clc
clear
close all;
```

Preliminary Work

Stuff not specified in any part but required for calculations.

Constants

```
mu = 3.986e14 * 1e-9; %[km^3/s^2] 1e-9 to convert from m^3 to km^3
Re = 6378.137; %[km]
J2 = 1082.63*10^(-6);
```

Read in Cities & Coastline

Used for plotting later need conversion to cartesian for 3D

```
cities_data = readtable('worldcities.csv');
cities_ll = [cities_data.lng cities_data.lat];
coastlines_ll = load('world_coastline_low.txt');
```

```
   [cities(:,1),cities(:,2),cities(:,3)] =
sph2cart(deg2rad(cities_ll(:,1)),deg2rad(cities_ll(:,2)),Re);

   [coastlines(:,1),coastlines(:,2),coastlines(:,3)] =
sph2cart(deg2rad(coastlines_ll(:,1)),deg2rad(coastlines_ll(:,2)),Re);
```

Part i)

Read in a JSON constellation design file

Call loadConstellation Function

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

Part ii)

Propagate the constellation through time for a full mean solar day (in 30 second time intervals)

Define Mean Solar Day (30s time steps)

```
   MSD = 0:30:24*3600;
```

Propagate Constellation with propagateState function

```
   for i = 1:length(satellite_list)
       x_ = zeros(6,length(MSD));
       for j = 1:length(MSD)
           x_(:,j) =
propagateState(satellite_list(i).oe0,MSD(j),MSD(1),mu,J2,Re);
       end
       satellite_list(i).x = x_;
   end
```

Part iii)

Compute the number of spacecraft in line of site for each city i at each time step

Rotate Cities/Coastline in ECI

The spacecraft should remain the same, I'm going to do everything in ECI because it's easier testLoS function called at each time, city, and spacecraft

```
   % Earth Rotation: Sidereal Time
   omega_earth = 2*pi/(23*3600 + 56*60 + 4.1); %[rad/s]
   theta_earth = omega_earth * MSD; %[rad]
```

```
% Create num_LoS matrix, rows are times, columns are cities
num_LoS = zeros(length(MSD),length(cities));

% Iterate every time, city, and spacecraft for LoS
for i = 1:length(MSD)
    % Account for rotation of Earth
    cities_ = (angle2dcm(theta_earth(i),0,0,'ZXZ') * cities')';
    for j = 1:length(cities_)
        for k = 1:length(satellite_list)
            num_LoS(i,j) = num_LoS(i,j) +
testLoS(cities_(j,:),satellite_list(k).x(1:3,i),pi/8);
        end
    end
end
```

Part iv)

Plot a 3D render of constellation orbits and the earth (with coastlines and cities) at the final time

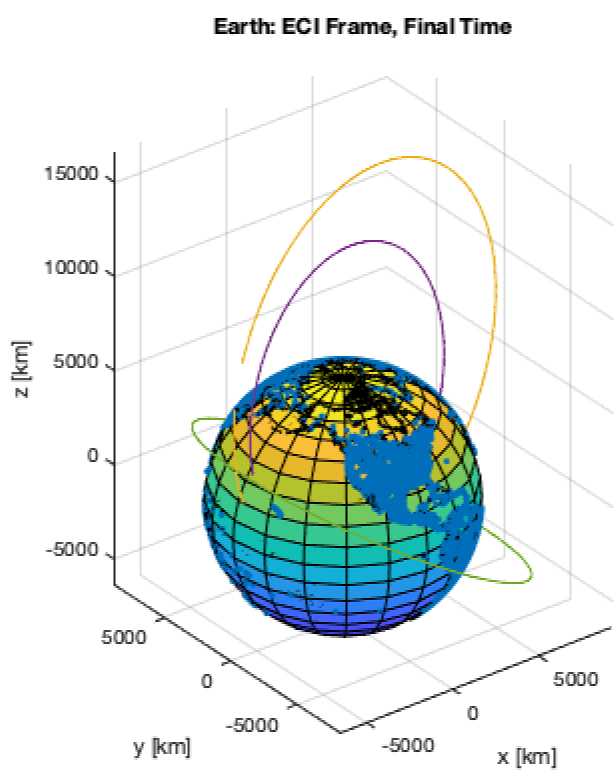
Propogate Final Orbits with ODE45

```
for i = 1:length(satellite_list)

    % Some orbital elements
    a = satellite_list(i).oe0(1);
    P = 2*pi*sqrt(a^3/mu);
    r_0 = satellite_list(i).x(:,end);

    % Propagate exactly one period
    t = [0 P];

    [t,X_i] = ode45(@(t,X_i)
positionfunc(t,X_i,mu),t,r_0,odeset('RelTol',1e-9,'AbsTol',1e-9));

    r = X_i(:,1:3)';
    r_dot = X_i(:,4:6)';

    satellite_list(i).finalorbit = r;
end
```

Plotting

```
% Account for rotation of Earth
cities_ = (angle2dcm(theta_earth(end),0,0,'ZXZ') * cities')';
coastlines_ = (angle2dcm(theta_earth(end),0,0,'ZXZ') *
coastlines')';

% Unit Sphere
[ex,ey,ez] = sphere;

% Plot:
figure()
```

```
    % Earth
    surf(Re*ex,Re*ey,Re*ez); hold on
    % Cities
    scatter3(cities_(:,1),cities_(:,2),cities_(:,3),'.')
    % Coastlines

 plot3(coastlines_(:,1),coastlines_(:,2),coastlines_(:,3),'k','Linewidth',2)
    % Spacecraft Orbits
    for i = 1:length(satellite_list)

 plot3(satellite_list(i).finalorbit(1,:),satellite_list(i).finalorbit(2,:),satelli
    end
    grid on
    xlabel('x [km]')
    ylabel('y [km]')
    zlabel('z [km]')
    title('Earth: ECI Frame, Final Time')
    axis equal
    hold off
```



Functions

```
% For ODE45
function drdt = positionfunc(t,r_0,mu)
    r_x = r_0(1);
    r_y = r_0(2);
```

```
    r_z = r_0(3);
    r_mag = norm(r_0);

    v_x = r_0(4);
    v_y = r_0(5);
    v_z = r_0(6);
    a_x = -(mu / (r_mag^3)) * r_x;
    a_y = -(mu / (r_mag^3)) * r_y;
    a_z = -(mu / (r_mag^3)) * r_z;

    drdt = [v_x;v_y;v_z;a_x;a_y;a_z];
end
```

Published with MATLAB® R2020a

```
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);

%1) extract the constellation structure from the json file

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

%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!
launches = val.launches;
num_launches = length(launches);
num_spacecraft = 0;
for i = 1:num_launches
    num_spacecraft = num_spacecraft + length(launches(i).payload);
end


%3) RECOMMENDED: Pre-allocate the satellite_list struct
satellite_list(num_spacecraft).name = '';
satellite_list(num_spacecraft).oe0 = NaN(6,1);

%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
```

```
icraft = 1;
for i = 1:num_launches
    for j = 1:length(launches(i).payload)
        satellite_list(icraft).name = launches(i).payload(j).name;
        satellite_list(icraft).oe0 = [launches(i).orbit.a ...
            launches(i).orbit.e launches(i).orbit.i ...
            launches(i).orbit.Om launches(i).orbit.om ...
            launches(i).payload(j).f];
        icraft = icraft + 1;
    end
end
```

```
Not enough input arguments.

Error in loadConstellation (line 24)
fid = fopen(filename);
```

Published with MATLAB® R2020a

```
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


%make sure that function has outputs
x = NaN(6,1);

%1) Compute the mean orbit elements oe(t) at time t due to J2
 perturbations
    a = oe0(1);
    e = oe0(2);
    i = oe0(3);
    Om = oe0(4);
    om = oe0(5);
    f = oe0(6);
    Om_dot = -((3/2)*((sqrt(mu)*J2*Re^2)/(2*(1-e^2)^2*a^(7/2)))) *
 cos(i);
    om_dot = Om_dot * ((5/2)*sin(i)^2 - 2) / cos(i);

    % Effects of peterbations
    Om = Om + Om_dot * (t - t_0);
    om = om + om_dot * (t - t_0);

    % Normalize to domain
    Om = Om - 2*pi*(floor(Om/(2*pi)));
    om = om - 2*pi*(floor(om/(2*pi)));

%2) Solve the time-of-flight problem to compute the true anomaly at
 tiem t
    h = sqrt(mu*a*(1-e^2));
    n = sqrt(mu/a^3);
    P = 2*pi/n;
    M = @(t) 2*pi/P*t;
    M_ = M(t-t_0);
    tol = 1e-9;

    [E_f,f_f] = keptof(2*pi*i/length(t),e,M_,tol);
```

```
%3) Compute r(t), rdot(t) in the perifocal frame
    r = a*(1-e^2) / (1 + e*cos(f));

    %Position
    r_f = [r*cos(f_f) r*sin(f_f) 0]';

    % Velocity
    r_dot_f = (mu/h)*[-sin(f_f) e+cos(f_f) 0]';

%4) Compute r(t), rdot(t) in the ECI frame, save into x
    PN = angle2dcm(Om,i,om,'ZXZ');

    r_ECI = PN' * r_f;
    r_dot_ECI = PN * r_dot_f;

    x = [r_ECI;r_dot_ECI];

function [E,f] = keptof(E_0,e,M,tol)
f_calc = @(E) 2*atan(sqrt((1+e)/(1-e)) * tan(E/2));
ratio = 1;
E_ = E_0;
%f_ = f_calc(E_0);

%i = 2;
    while ratio > tol
        ratio = (E_ - e*sin(E_) - M)/(1 - e*cos(E_));
        E_ = E_ - ratio;
        %f_(i) = f_calc(E_(i));
        %i = i+1;
    end
%iteration = 1:i-1;
E = E_;
f = f_calc(E_);
end
end
```

```
Not enough input arguments.

Error in propagateState (line 22)
    a = oe0(1);
```

Published with MATLAB® R2020a

```
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

%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)

angle = pi/2 - acos(dot(r_site,r_sc)/(norm(r_site)*norm(r_sc)));

inLoS = double(angle > elevation_limit);

end
```

```
Not enough input arguments.

Error in testLoS (line 18)
angle = pi/2 - acos(dot(r_site,r_sc)/(norm(r_site)*norm(r_sc)));
```

Published with MATLAB® R2020a