

# SATELLITE DYNAMICS AND ATTITUDE CONTROL

Kusal Upadhyay, Harry Zhao

15 May 2024



# REVISION HISTORY

VERSION	REVISION NOTES
PS1	<ul style="list-style-type: none"><li>- Created document</li><li>- Added PS1 material</li></ul>
PS2	<ul style="list-style-type: none"><li>- Added PS2 material</li><li>- Updated title page</li><li>- Updated moment of inertia for center of mass</li></ul>
PS3	<ul style="list-style-type: none"><li>- Added PS3 material</li><li>- Updated title page</li></ul>
PS4	<ul style="list-style-type: none"><li>- Added PS4 material</li><li>- Updated title page</li><li>- Switched to 312 Euler angles</li></ul>
PS5	<ul style="list-style-type: none"><li>- Added PS5 material</li><li>- Updated title page</li><li>- Fix PS2 figure caption typos</li><li>- Fix PS4 figure numbering</li></ul>
PS6	<ul style="list-style-type: none"><li>- Introduce Simulink model</li><li>- Added PS6 material</li><li>- Updated title page</li><li>- Fixed PS5 plots</li></ul>

Table 1: Summary of project revisions.

# TABLE OF CONTENTS

<b>1 PROBLEM SET 1</b>	<b>5</b>
1.1 PROBLEM 1 . . . . .	5
1.2 PROBLEM 2 . . . . .	5
1.3 PROBLEM 3 . . . . .	6
1.4 PROBLEM 4 . . . . .	6
1.5 PROBLEM 5 . . . . .	7
1.6 PROBLEM 6 . . . . .	13
1.7 PROBLEM 7 . . . . .	14
<b>2 PROBLEM SET 2</b>	<b>16</b>
2.1 PROBLEM 1 . . . . .	16
2.2 PROBLEM 2 . . . . .	17
2.3 PROBLEM 3 . . . . .	18
2.4 PROBLEM 4 . . . . .	18
2.5 PROBLEM 5 . . . . .	19
2.6 PROBLEM 6 . . . . .	19
2.7 PROBLEM 7 . . . . .	21
2.8 PROBLEM 8 . . . . .	21
2.9 PROBLEM 9 . . . . .	22
<b>3 PROBLEM SET 3</b>	<b>28</b>
3.1 PROBLEM 1 . . . . .	28
3.2 PROBLEM 2 . . . . .	28
3.3 PROBLEM 3 . . . . .	29
3.4 PROBLEM 4 . . . . .	30
3.5 PROBLEM 5 . . . . .	32
3.6 PROBLEM 6 . . . . .	33
3.7 PROBLEM 7 . . . . .	34
<b>4 PROBLEM SET 4</b>	<b>39</b>
4.1 PROBLEM 1 . . . . .	39
4.2 PROBLEM 2 . . . . .	41
4.3 PROBLEM 3 . . . . .	43
4.4 PROBLEM 4 . . . . .	49
<b>5 PROBLEM SET 5</b>	<b>54</b>
5.1 PROBLEM 1 . . . . .	54
5.2 PROBLEM 2 . . . . .	58
5.3 PROBLEM 3 . . . . .	62
<b>6 PROBLEM SET 6</b>	<b>65</b>
6.1 PROBLEM 1 . . . . .	65
6.2 PROBLEM 2 . . . . .	65
6.3 PROBLEM 3 . . . . .	67
6.4 PROBLEM 4 . . . . .	68
6.5 PROBLEM 5 . . . . .	69

6.6 PROBLEM 6 . . . . .	72
<b>7 REFERENCES</b>	<b>76</b>
<b>A Appendix A</b>	<b>77</b>
A.1 Problem Set 1 . . . . .	77
A.2 Problem Set 2 . . . . .	78
A.3 Problem Set 3 . . . . .	84
A.4 Problem Set 4 . . . . .	90
A.5 Problem Set 5 . . . . .	97
A.6 Problem Set 6 . . . . .	107

# 1 PROBLEM SET 1

## 1.1 PROBLEM 1

*Select some key ADCS characteristics of your mission, including orbit (e.g., LEO, MEO, GEO, HEO, Interplanetary), target attitude (e.g., Sun pointing, Inertial pointing, Earth pointing, Resident Space Object pointing), attitude state representation (e.g., Euler angles, Gibbs vector, Quaternions Direction Cosine Matrix, Euler Axis/Angle), sensors suite (Gyros, Magnetometers, Star Trackers, Earth Sensor, Sun Sensor), actuator suite (Thrusters, Magnetorquers, Reaction Wheels, Momentum Wheel, Gravity Gradient).*

Our mission will utilize a satellite with synthetic aperture radar (SAR), designed to gather key remote sensing and environmental data for the Earth. The satellite will be in low Earth orbit (LEO) and use quaternions to describe its orientation, avoiding gimbal lock effects of other conventions. For state estimation, the spacecraft will require gyroscopes, star trackers, and a potentially a sun sensor. For actuation, the spacecraft will likely utilize thrusters, reaction wheels, and magnetorquers.

## 1.2 PROBLEM 2

*Conduct survey of satellites which have characteristics similar to selected project. Use internet, publications, and books as resources.*

Space agencies such as NASA have been constructing SAR satellites to gather satellite images and data of Earth for over a decade. Additionally, there exist commercial entities also utilizing SAR in their spacecraft.

For example, Soil Moisture Active Passive (SMAP) is a NASA satellite launched in 2015 that utilizes L-band synthetic aperture radar (SAR) technology to measure soil moisture from LEO. This data has applications in climate change research climate change research applications (such as updating climate models) and some day-to-day activities (such as improving weather forecasts). SMAP is unique in that it had a large deployable reflector, held above the spacecraft body by a deployable boom [1].

Companies EOS and Capella Space are also developing satellites that use SAR technology in the X-band and S-band frequencies for commercial applications ranging from agriculture to mining [2], [3]. The commercial applicability of SAR is substantial, especially as SAR can penetrate cloud cover while generating high-resolution data, making it superior to many other forms of remote sensing technology. EOS claims to obtain resolution of up to 0.25 m, while Capella Space claims a capability of up to 0.5 m. These satellites all operate in LEO, which enables high-frequency monitoring of the Earth's surface.

NASA and ISRO have partnered to create a SAR satellite as well. The joint project between NASA JPL and ISRO has resulted in the NASA-ISRO Synthetic Aperture Radar (NISAR), a satellite that captures data in the L-band and S-band SAR frequencies [4]. NISAR's high resolution will permit the detailed measurement of the Earth's surface, enabling better observation of changes in Earth's crust for disaster prevention and mitigation. NISAR will also support science goals such as monitoring ice sheets and the oceans, and its orbit is designed to cover the entire Earth every 12 days.

### 1.3 PROBLEM 3

Select preferred existing satellite and payload for project. Similarity is helpful, but not strictly required.

We select the NASA JPL and ISRO NISAR mission as the mission on which to base our satellite. In the following section, we describe mission details and basic specifications. We simplify the satellite geometry and compute the center of mass and inertia tensor of the satellite. We will also analyze the satellite's external surfaces, which are relevant to disturbances such as atmospheric drag and solar radiation pressure.

### 1.4 PROBLEM 4

Collect basic information on mission, requirements, ADCS sensors and actuators, mechanical layout, mass, mass distribution, and inertia properties.

NISAR is a joint Earth-observation satellite mission between NASA and ISRO. It is the first satellite to operate in two different Synthetic Aperture Radar (SAR) bands, incorporating both L- and S-band SAR instruments. Both frequencies can penetrate clouds for reliable data collection, but the L-band can also penetrate thicker vegetation than the S-band cannot. Uniquely, NISAR is intended to be used for a wide range of science objectives, including disaster response and agriculture [5].

NISAR ADCS requirements are  $<273$  arcseconds for pointing and  $<500$  m for orbit control [6]. The satellite duty cycle is specified as  $>30\%$ . NISAR will operate in LEO with nominal altitude of 747 km and 6 AM/6 PM orbit. NISAR's L- and S-band instruments operate at 24 cm and 12 cm wavelengths, respectively. NISAR collects terrestrial SAR imagery with an image swatch of 240 km using a sweep approach. The science payload can also perform polarimetry, with the SAR incorporating multiple polarization modes.

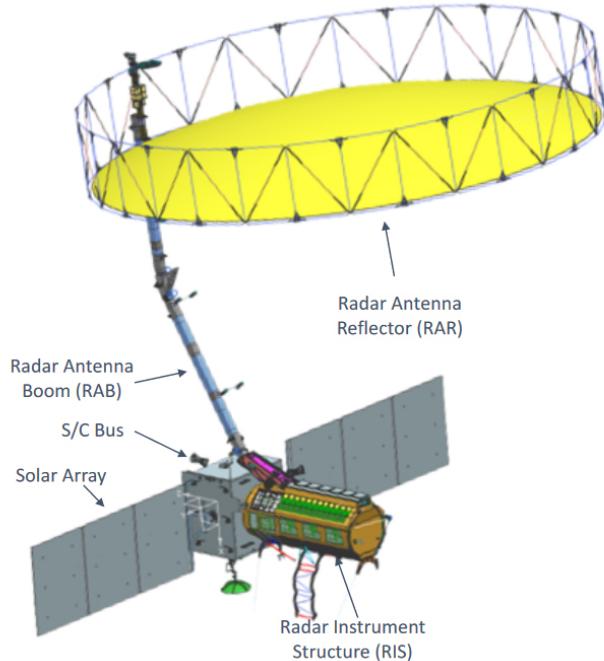


Figure 1: The basic components of the NISAR satellite

As shown in Figure 1, NISAR's satellite consists of a 1.2 m x 1.8 m x 1.9 m spacecraft bus cuboid with a 1.2 m wide octagonal Radar Instrument Structure (RIS). The spacecraft bus includes ADCS hardware, power subsystem, and engineering payload, while the RIS houses hardware for the L- and S-band SAR. The satellite is powered by 23 m<sup>2</sup> of solar panels, consisting of an array of two panels, one on each side of the satellite. Additionally, a 12 m diameter radar antenna is positioned above the body of the spacecraft, attached by a 9 m long boom. This boom consists of beams with 7 in x 7 in cross-section area [4].

Table 2 contains mass properties of the satellite. Unfortunately, detailed mass distribution and inertia properties of NISAR are not openly available, so we provide estimates of mass distribution based on known overall component-level masses. We are given total masses for the bus structure and RIS, and we also know the masses of the payloads located within, allowing us to compute an accurate mass for these components [4]. We estimate that solar panels have a mass of 23 kg each based on knowledge that NISAR's solar panels are 23 m<sup>2</sup> and a typical solar panel mass per area is 2.06 kg/m<sup>2</sup> [7]. We know that the entire radar antenna assembly has a mass of 292 kg, and we estimate that the reflector has a mass of approximately 100 kg based on a similar deployable SAR S- and L-band mesh antenna reflector [8]. We will use these masses to compute center of mass and moments of inertia in the following section. For our model, we neglect the effects of the truss structure supporting the antenna reflector, instead modeling the entire RAR as just the disk-shaped reflector mesh.

Table 2: Mass of NISAR components

Components	Mass [kg]
Bus	964.1
Radar Instrument Structure (RIS)	1375.9
Solar Panel +y	23
Solar Panel -y	23
Radar Antenna Boom (RAB)	192
Radar Antenna Reflector (RAR)	100

Since NISAR is a remote sensing satellite requiring high attitude control performance, it has an ADCS system with an array of sensors and actuators. Sensors include star sensors, sun sensors, GPS, and a 3-axis gyroscope for roll, pitch, and yaw. For actuators, NISAR has four 50 N·m reaction wheels mounted in tetrahedral configuration, three 565 and 350 A·m<sup>2</sup> magnetorquers, and fourteen thrusters (ten canted 11 N thrusters, one central 11 N thruster, and four 1 N thrusters for roll) [4].

## 1.5 PROBLEM 5

*Simplify spacecraft geometry, make assumptions on mass distribution, e.g. splitting it in its core parts, define body axes (typically related to geometry and payload), compute moments of inertia and full inertia tensor w.r.t. body axes. Show your calculations.*

We simplify the spacecraft geometry into six components, each individually assumed to have uniform mass distribution. These components of the simplified geometry are: bus structure (including ADCS hardware and engineering payload), RIS (Radar Instrument Structure), RAB (radar antenna boom), RAR (radar antenna reflector), and two solar panels (identified as the +y solar panel and -y solar panel). The bus structure is modeled as a rectangular prism, while the RIS is modeled as an octagonal prism. The RAB is also modeled as a rectangular prism,

while the RAR is modeled as a thin disk and the solar panels are modeled as thin rectangular plates. Within each geometry, our model assumes mass is distributed uniformly. From analyzing diagrams found in technical reports, we estimate that the RAR is tilted  $-3.87^\circ$  about the y-axis (relative to the x-axis), while the RAB is modeled as a single beam with an angle approximately  $-18^\circ$  from vertical (from the z-axis, about the y-axis in the x-z plane). Note that we have simplified the shape of the RAB from a beam of two angled segments to a single, straight beam.

We choose the body axes to have an origin at the center of the rectangular bus. This configuration is chosen because the bus houses the ADCS hardware, including actuators and sensors. The x-axis points in the direction of the RIS, and the z-axis points up vertically, normal to the upper surface of the bus. See Figure 4 for a visual depiction of the body axes relative to the spacecraft.

We compute the center of mass after extracting the centroid of each component. The mass of each component is previously found in Table 2. The centroid of each component is listed in Table 3.

Table 3: Component centroids [m]

Part	x	y	z
Bus	0	0	0
RIS	1.85	0	0
Panel +y	0	3.9	0
Panel -y	0	-3.9	0
RAB	-0.899	0	5.194
RAR	4.283	0	8.308

The center of mass can be found by taking the weighted average of each component centroid, weighted by the mass of each component. The center of mass formula is:

$$\vec{r}_{cm} = \frac{\sum m_i \vec{r}_i}{\sum m_i}$$

This yields a result for center of mass at  $[1.046, 0, 0.683]$  m relative to the origin we defined. We created the following MATLAB script to compute the center of mass from a CSV file containing centroid and mass data.

```

1 function cm = computeCM(filename)
2     data = readmatrix(filename);
3     x = data(:,1);
4     y = data(:,2);
5     z = data(:,3);
6     m = data(:,4);
7     cm = [dot(x,m); ...
8         dot(y,m); ...
9         dot(z,m)] / sum(m);
10 end

```

We compute the moment of inertia of the satellite, finding an inertia tensor in our body axes. To do this, we break the satellite into individual components, first finding the moment of inertia about the center of mass of each component. We then compute the moment of inertia of the entire satellite about the body axes by using parallel axis theorem and combining all the components.

To compute the moment of inertia, we need the following geometric properties of each component:

Table 4: Dimensions of modeled components [m]

Part	L (x-dim)	W (y-dim)	H (z-dim)	S (oct. side length)	R (radius)
Bus		1.8	1.9	-	-
RIS	2.5	-	-	0.459	-
Panel +y	-	6	1.9	-	-
Panel -y	-	6	1.9	-	-
RAB	0.1778	0.1778	9	-	-
RAR	-	-	-	-	6

For the bus, we choose to model the geometry as a rectangular prism. Since the bus is aligned with the body axes, we obtain a diagonal inertia tensor:

$$\begin{aligned}
 I_{bus} &= \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \\
 &= \begin{bmatrix} m \frac{W^2+H^2}{12} & 0 & 0 \\ 0 & m \frac{L^2+H^2}{12} & 0 \\ 0 & 0 & m \frac{L^2+W^2}{12} \end{bmatrix} \\
 &= \begin{bmatrix} 550.340 & 0 & 0 \\ 0 & 405.725 & 0 \\ 0 & 0 & 375.9993 \end{bmatrix} \text{ kg m}^2
 \end{aligned}$$

For the solar panels, we approximate their geometry as a flat plate. These axes are also aligned, so the tensor can be diagonal.

$$\begin{aligned}
 I_{panel} &= \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \\
 &= \begin{bmatrix} m \frac{W^2+H^2}{12} & 0 & 0 \\ 0 & m \frac{H^2}{12} & 0 \\ 0 & 0 & m \frac{W^2}{12} \end{bmatrix} \\
 &= \begin{bmatrix} 75.919 & 0 & 0 \\ 0 & 6.919 & 0 \\ 0 & 0 & 69 \end{bmatrix} \text{ kg m}^2
 \end{aligned}$$

For the RIS, we model the geometry as an octagonal prism. For the moment of inertia about the axisymmetric axis of the octagon, we use the formula  $m \left( \frac{S^2}{24} + \frac{a^2}{2} \right)$ , where  $S$  is the side length and  $a$  is the apothem length, where the apothem is the perpendicular length from a side of the octagon to the center [9]. When calculating the moment of inertia about the non-axisymmetric axes, we approximate the geometry as a cylinder with radius equal to the average of the octagonal radius (distance from center to vertex) and the apothem. As will be shown later, this approximation yields a very close result to the inertia tensor generated from the CAD model.

$$\begin{aligned}
I_{RIS} &= \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \\
&= \begin{bmatrix} m \left( \frac{S^2}{24} + \frac{a^2}{2} \right) & 0 & 0 \\ 0 & m \left( \frac{L^2}{12} + \frac{R_{avg}^2}{4} \right) & 0 \\ 0 & 0 & m \left( \frac{L^2}{12} + \frac{R_{avg}^2}{4} \right) \end{bmatrix} \\
&= \begin{bmatrix} 223.268 & 0 & 0 \\ 0 & 831.089 & 0 \\ 0 & 0 & 831.089 \end{bmatrix} \text{ kg m}^2
\end{aligned}$$

For the RAB, we first model the geometry as a rectangular prism. We also must rotate the inertia tensor to match the orientation of the body axes, as the RAB itself is rotated relative to the bus about the y-axis by  $-18^\circ$  from vertical (z-axis).

$$\begin{aligned}
I_{RAB} &= \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \\
&= \begin{bmatrix} m \frac{W^2+H^2}{12} & 0 & 0 \\ 0 & m \frac{L^2+H^2}{12} & 0 \\ 0 & 0 & m \frac{L^2+W^2}{12} \end{bmatrix} \\
&= \begin{bmatrix} 1296.506 & 0 & 0 \\ 0 & 1296.506 & 0 \\ 0 & 0 & 1.012 \end{bmatrix} \text{ kg m}^2
\end{aligned}$$

We apply the rotation to the inertia tensor using a rotation matrix about the y-axis. Note that doing so results in non-zero products of inertia, meaning our principal axes will not be aligned with our body axes.

$$\begin{aligned}
I_{RAB,rotated} &= R_y(-18^\circ) I_{RAB} R_y^T(-18^\circ) \\
&= \begin{bmatrix} 1172.797 & 0 & 380.736 \\ 0 & 1296.506 & 0 \\ 380.736 & 0 & 124.720 \end{bmatrix} \text{ kg m}^2
\end{aligned}$$

Finally, we model the RAR as a flat disk. Similar to the RAB, we must rotate the inertia tensor to match the orientation of the body axes.

$$\begin{aligned}
I_{RAR} &= \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \\
&= \begin{bmatrix} m\frac{R^2}{4} & 0 & 0 \\ 0 & m\frac{R^2}{4} & 0 \\ 0 & 0 & m\frac{R^2}{2} \end{bmatrix} \\
&= \begin{bmatrix} 900 & 0 & 0 \\ 0 & 900 & 0 \\ 0 & 0 & 1800 \end{bmatrix} \text{ kg m}^2
\end{aligned}$$

Applying a rotation:

$$\begin{aligned}
I_{RAR,rotated} &= R_y(-3.87^\circ) I_{RAR} R_y^\top(-3.87^\circ) \\
&= \begin{bmatrix} 904.100 & 0 & -60.605 \\ 0 & 900 & 0 \\ -60.605 & 0 & 1795.900 \end{bmatrix} \text{ kg m}^2
\end{aligned}$$

Now, we use parallel axis theorem to compute the moment of inertia of each component about the body axes at the specified origin. We can use the following displacement tensor,

$$D = \begin{bmatrix} y^2 + z^2 & -xy & -xz \\ -yx & x^2 + z^2 & -yz \\ -zx & -yz & x^2 + y^2 \end{bmatrix},$$

where  $x, y, z$  are the coordinates of the center of mass of the component, giving the moment of inertia about a new point:

$$I' = I_c + mD$$

Performing the parallel axis theorem on each component and summing the inertia tensors, we obtain the following inertia tensor for the entire spacecraft:

$$I_{NISAR,body} = \begin{bmatrix} 15783.996 & 0 & -2341.659 \\ 0 & 22227.752 & 0 \\ -2341.659 & 0 & 10663.970 \end{bmatrix} \text{ kg m}^2$$

Compare this with the inertia tensor computed by SolidWorks CAD software:

$$I_{NISAR,body} = \begin{bmatrix} 15780.361 & 0 & -2336.285 \\ 0 & 22225.721 & 0 \\ -2336.285 & 0 & 10665.796 \end{bmatrix} \text{ kg m}^2$$

The errors are 0.0230%, 0.00914%, 0.0171%, and 0.230% for  $I_{xx}$ ,  $I_{yy}$ ,  $I_{zz}$ , and  $I_{xz}$ , respectively. We created the following MATLAB script to compute the inertia tensor.

```

1 function I = computeMOI(filename, reference)
2     data = readmatrix(filename);
3     x = data(:,1) - reference(1);
4     y = data(:,2) - reference(2);
5     z = data(:,3) - reference(3);
6     m = data(:,4);
7     m_bus = m(1);
8     m_RIS = m(2);
9     m_panel = m(3);
10    m_RAB = m(5);
11    m_RAR = m(6);
12
13    L_bus = 1.2;
14    W_bus = 1.8;
15    H_bus = 1.9;
16    I_bus = m_bus * [(W_bus^2 + H_bus^2) / 12, 0, 0; ...
17                      0, (L_bus^2 + H_bus^2) / 12, 0; ...
18                      0, 0, (L_bus^2 + W_bus^2) / 12];
19
20    L_RIS = 2.5;
21    S_RIS = 0.459;
22    a_RIS = S_RIS / (2 * tan(deg2rad(22.5)));
23    R_avg = mean([a_RIS sqrt(a_RIS^2 + (S_RIS / 2)^2)]);
24    I_RIS = m_RIS * [S_RIS^2 / 24 + a_RIS^2 / 2, 0, 0; ...
25                      0, L_RIS^2 / 12 + R_avg^2 / 4, 0; ...
26                      0, 0, L_RIS^2 / 12 + R_avg^2 / 4];
27
28    W_panel = 6;
29    H_panel = 1.9;
30    I_panel = m_panel * [(W_panel^2 + H_panel^2) / 12, 0, 0; ...
31                      0, H_panel^2 / 12, 0; ...
32                      0, 0, W_panel^2 / 12];
33
34    L_RAB = 0.1778;
35    W_RAB = 0.1778;
36    H_RAB = 9;
37    deg_RAB = -18;
38    rot_RAB = [cosd(deg_RAB), 0, sind(deg_RAB); ...
39                 0, 1, 0; ...
40                 -sind(deg_RAB), 0, cosd(deg_RAB)];
41    I_RAB = m_RAB * [(W_RAB^2 + H_RAB^2) / 12, 0, 0; ...
42                      0, (L_RAB^2 + H_RAB^2) / 12, 0; ...
43                      0, 0, (L_RAB^2 + W_RAB^2) / 12];
44    I_RAB_rot = rot_RAB * I_RAB * rot_RAB';
45
46    R_RAR = 6;
47    deg_RAR = -3.87;
48    rot_RAR = [cosd(deg_RAR), 0, sind(deg_RAR); ...
49                 0, 1, 0; ...
50                 -sind(deg_RAR), 0, cosd(deg_RAR)];
51    I_RAR = m_RAR * [R_RAR^2 / 4, 0, 0; ...
52                      0, R_RAR^2 / 4, 0; ...
53                      0, 0, R_RAR^2 / 2];
54    I_RAR_rot = rot_RAR * I_RAR * rot_RAR';

```

```

55
56     I_c = {I_bus, I_RIS, I_panel, I_panel, I_RAB_rot, I_RAR_rot};
57     I = zeros([3 3]);
58     for i = 1:length(I_c)
59         D = [y(i)^2 + z(i)^2, -x(i) * y(i), -x(i) * z(i); ...
60             -y(i) * x(i), x(i)^2 + z(i)^2, -y(i) * z(i); ...
61             -z(i) * x(i), -y(i) * z(i), x(i)^2 + y(i)^2];
62         I = I + I_c{i} + m(i) * D;
63     end
64 end

```

## 1.6 PROBLEM 6

*Discretize your spacecraft through its outer surfaces (geometry). Develop a Matlab/Simulink function to handle barycenter (geometry, no mass distribution) coordinates, size, and unit vectors normal to each outer surface of the spacecraft in body frame. You can list all this information in a Matrix. This will be essential later on to compute environmental torques acting on the spacecraft from forces, surface orientation, and the vectors connecting the satellite's center of mass to each surface's center of mass.*

For the purpose of discretizing the spacecraft into surfaces, we consider the outer faces of the bus, RIS, and RAB. We also consider the faces of the solar panels and RAR, which are modeled as thin plates. The centroid (barycenter) coordinates and area for each surface are obtained using the surface properties tool in SolidWorks, and a unit normal vector is manually computed based on the orientation of the surface. We then enter this data into a CSV file, which can be read into MATLAB using the following function:

```

1 function [barycenter, normal, area] = surfaces(filename, rotm)
2     data = readmatrix(filename);
3     barycenter = rotm * data(:,1:3)';
4     normal = rotm * data(:,4:6)';
5     area = data(:,7)';
6 end

```

This function stores the data into arrays of barycenter coordinates, unit normal vector components, and area. Each row of an array corresponds to a particular surface. The data is shown in Table 5, annotated with the identity of each surface.

Table 5: Surface parameters

Surface	Barycenter [m]			Normal			Area [m <sup>2</sup> ]
	x	y	z	x	y	z	
Bus front, minus RIS (+x)	0.6	0	0	1	0	0	2.4
Bus rear (-x)	-0.6	0	0	-1	0	0	3.42
Bus side (+y)	0	0.9	0	0	1	0	2.28
Bus side (-y)	0	-0.9	0	0	-1	0	2.28
Bus top (+z)	0	0	0.95	0	0	1	2.16
Bus bottom (-z)	0	0	-0.95	0	0	-1	2.16
RIS front (+x)	3.1	0	0	1	0	0	1.02
RIS top (+z)	1.85	0	0.55	0	0	1	1.15
RIS bottom (-z)	1.85	0	-0.55	0	0	-1	1.15

RIS side (+y)	1.85	0.55	0	0	1	0	1.15
RIS side (-y)	1.85	-0.55	0	0	-1	0	1.15
RIS angle face (y-z I)	1.85	0.39	0.39	0	0.707	0.707	1.15
RIS angle face (y-z II)	1.85	-0.39	0.39	0	-0.707	0.707	1.15
RIS angle face (y-z III)	1.85	-0.39	-0.39	0	-0.707	-0.707	1.15
RIS angle face (y-z IV)	1.85	0.39	-0.39	0	0.707	-0.707	1.15
Panel +y front (+x)	0	3.9	0	1	0	0	11.4
Panel +y rear (-x)	0	3.9	0	-1	0	0	11.4
Panel -y front (+x)	0	-3.9	0	1	0	0	11.4
Panel -y rear (-x)	0	-3.9	0	-1	0	0	11.4
RAB front (x-z, +x)	-0.81	0	5.21	0.951	0	0.309	1.6
RAB rear (x-z, -x)	-0.99	0	5.18	-0.951	0	-0.309	1.6
RAB side (+y)	-0.9	-0.09	5.19	0	1	0	1.6
RAB side (-y)	-0.9	0.09	5.19	0	-1	0	1.6
RAB top (+z)	-2.31	0	9.47	0	0	1	0.03
RAR top (+z)	4.28	0	8.31	-0.067	0	0.998	113.1
RAR bottom (-z)	4.28	0	8.31	0.067	0	-0.998	113.1

## 1.7 PROBLEM 7

*At this stage you should have a simple 3D model of your spacecraft including geometry and mass properties of each element. Plot body axes (triad) in 3D superimposed to spacecraft 3D model.*

A 3D model of the spacecraft is shown in Figure 2. The model shown is a screen capture from the SolidWorks CAD software.

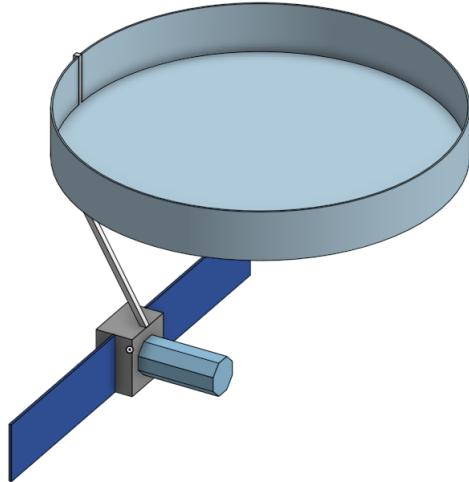


Figure 2: A 3D model of the satellite

We also show a simplified model of the spacecraft in Figure 3. This is the model we use to compute our mass and surface properties.

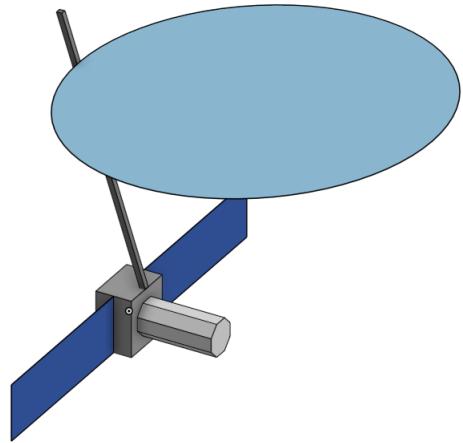


Figure 3: A 3D model of the simplified satellite geometry

We also plot the model in MATLAB by importing an STL version of the CAD model. We show the body axes in Figure 4, with the origin chosen as the center of the spacecraft bus.

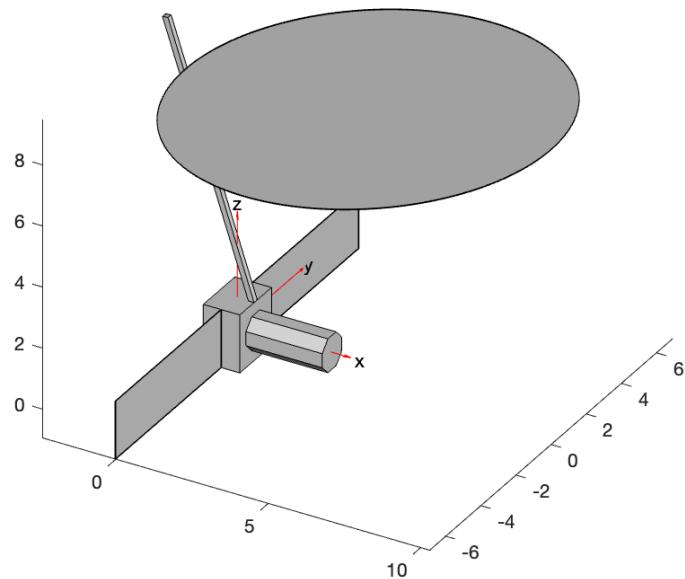


Figure 4: Satellite model in MATLAB with body axes shown

## 2 PROBLEM SET 2

### 2.1 PROBLEM 1

Define orbit initial conditions and make sure you can propagate the orbit of the satellite over multiple orbits using either a Keplerian propagator or a numerical integration scheme (see AA279A material). Best would be to use a numerical integrator, so that you can later try to feed the same environmental forces for orbit propagation which are applied for attitude propagation (very cool!).

From the science users' handbook, we obtain the following orbital elements [10].

OE	$a$	$e$	$i$	$\Omega$	$\omega$	$\nu$
Value	7125.48662 km	0.0011650	98.40508°	-19.61601°	89.99764°	-89.99818°

We convert these using a MATLAB function into ECI coordinates that can be fed into a numerical orbital propagator. Notice that we first convert the orbital elements  $a$ ,  $e$ , and  $\nu$  into perifocal (PQW) coordinates, using  $a$  and  $e$  to find the semi-latus rectum and  $a$ ,  $e$ , and  $\nu$  to find the distance to the central body (Earth). Then, we perform a series of rotations on these coordinates parameterized by  $\omega$ ,  $i$ , and  $\Omega$  to obtain new coordinates in the ECI frame.

```

1 function yECI = oe2eci(a,e,i,O,w,nu)
2     i = deg2rad(i);
3     O = deg2rad(O);
4     w = deg2rad(w);
5     nu = deg2rad(nu);
6     p = a * (1 - e^2);
7     r = p / (1 + e * cos(nu));
8     rPQW = [r * cos(nu); r * sin(nu); 0];
9     vPQW = sqrt(3.986 * 10^5 / p) * [-sin(nu); e + cos(nu); 0];
10    Rzw = [cos(-w), sin(-w), 0; ...
11            -sin(-w), cos(-w), 0; ...
12            0, 0, 1];
13    Rxi = [1, 0, 0; ...
14            0, cos(-i), sin(-i); ...
15            0, -sin(-i), cos(-i)];
16    Rzo = [cos(-O), sin(-O), 0; ...
17            -sin(-O), cos(-O), 0; ...
18            0, 0, 1];
19    rECI = Rzo * Rxi * Rzw * rPQW;
20    vECI = Rzo * Rxi * Rzw * vPQW;
21    yECI = [rECI; vECI];
22 end

```

Then, we can numerically propagate in MATLAB using `ode113` using a function that computes the time derivative of the ECI state. This is accomplished simply by setting the time derivative of position equal to the velocity portion of the state and setting the time derivative of velocity equal to an acceleration computed using the law of universal gravitation. Note that while our propagator does not include disturbance forces, it will be easy to incorporate these later. See the appendix corresponding to Problem Set 2 for application of `ode113`.

```

1 function [stateDot] = orbitSimple(t, state)
2     r = state(1:3);
3     v = state(4:6);
4     stateDot = zeros(6,1);
5     stateDot(1:3) = v; % km/s
6     stateDot(4:6) = (-3.986 * 10^5 / norm(r)^2) * r / norm(r); % km/s^2
7 end

```

Now, we plot the trajectory for one orbit in Figure 5. Plotting multiple orbits (for example, over 12 days) yields the same plot, as `ode113` is very stable for this application.

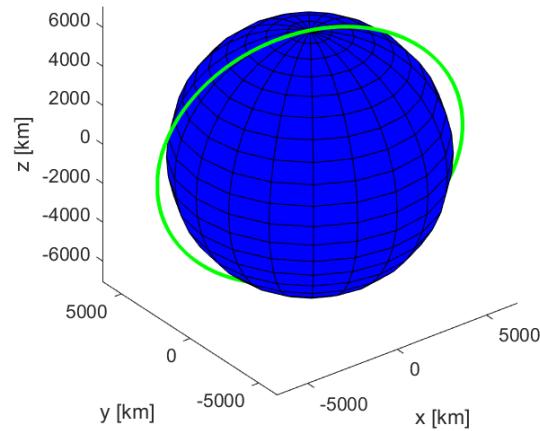


Figure 5: A single orbit for NISAR in ECI coordinates (no perturbations)

## 2.2 PROBLEM 2

*In general the body axes are not the principal axes. Identify principal axes through the eigen-vector/eigenvalue problem discussed in class and compute the rotation matrix from body to principal axes.*

The unit vectors of the principal axes with respect to the body axes ( $\vec{e}_i$ ) and the inertia tensor in the principal axes ( $I_i$ ) can be found by taking the eigenvalue decomposition of the inertia tensor in the body axis. This can be seen in the two equations below.

$$I_i \cdot \vec{e}_i = I_i \cdot \vec{I}_{body} \quad i = x, y, z$$

$$\vec{I}_{principal} = \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix} = \begin{bmatrix} 7707.07 & 0 & 0 \\ 0 & 14563.2 & 0 \\ 0 & 0 & 18050.4 \end{bmatrix} \text{ kg m}^2$$

We follow convention  $I_z > I_y > I_x$  for defining principal axes.

The unit vectors of the principal axes ( $\vec{e}_i$ ) can then be used to find the rotation matrix ( $\vec{R}$ ), as shown below.

$$\vec{R} = [\vec{e}_x \quad \vec{e}_y \quad \vec{e}_z] = \begin{bmatrix} -0.06278 & -0.99803 & 0 \\ 0 & 0 & 1 \\ -0.99803 & 0.06278 & 0 \end{bmatrix}$$

$$\vec{I}_{body} = \vec{R} \vec{I}_{principal} \vec{R}^\top$$

## 2.3 PROBLEM 3

*At this stage you should have a simple 3D model of your spacecraft including geometry and mass properties of each element. This includes at least two coordinate systems, body and principal axes respectively, and the direction cosine matrix between them. Plot axes of triads in 3D superimposed to spacecraft 3D model.*

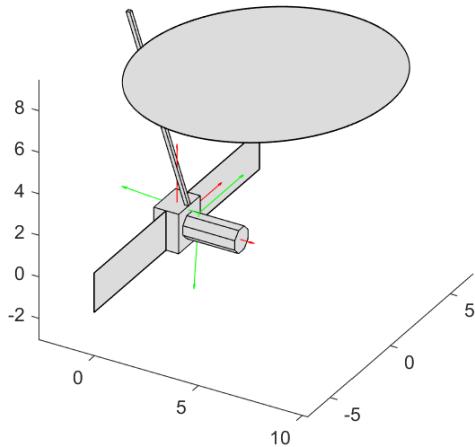


Figure 6: Principal axes at center of mass (green) and body axes at origin (red)

## 2.4 PROBLEM 4

*Program Euler equations in principal axes (e.g. in Matlab/Simulink). No external torques.*

We use the following equations with zero external moments ( $M_x, M_y, M_z = 0$ ).

$$I_x \dot{\omega}_x + (I_z - I_y) \omega_y \omega_z = M_x$$

$$I_y \dot{\omega}_y + (I_x - I_z) \omega_z \omega_x = M_y$$

$$I_z \dot{\omega}_z + (I_y - I_x) \omega_x \omega_y = M_z$$

```
1 function [wDot] = eulerEquation(t,w,Ix,Iy,Iz)
2     wx = w(1);
```

```

3      wy = w(2);
4      wz = w(3);
5      wDot = zeros(3,1);
6      wDot(1) = (Iy - Iz) / Ix * wy * wz;
7      wDot(2) = (Iz - Ix) / Iy * wz * wx;
8      wDot(3) = (Ix - Iy) / Iz * wx * wy;
9  end

```

## 2.5 PROBLEM 5

*Numerically integrate Euler equations from arbitrary initial conditions ( $\omega < 10^\circ/\text{s}$ ,  $\omega_i \neq 0$ ). Multiple attitude revolutions.*

We choose arbitrary initial conditions  $\omega_x = 8^\circ \text{s}^{-1}$ ,  $\omega_y = 4^\circ \text{s}^{-1}$ , and  $\omega_z = 6^\circ \text{s}^{-1}$ . The results of numerical integration using `ode113` are shown in Figure 7.

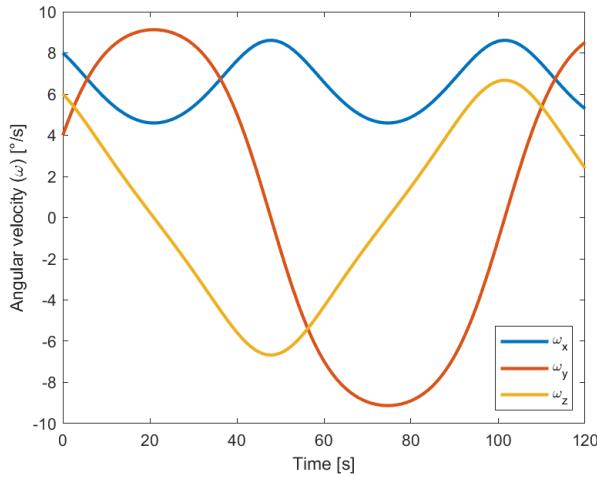


Figure 7: Results from numerical integration of Euler equations

## 2.6 PROBLEM 6

*Plot rotational kinetic energy and momentum ellipsoids in 3D (axis equal) corresponding to chosen initial conditions. Verify that semi-axis of ellipsoids corresponds to theoretical values.*

For the energy ellipsoid, we compute our surface using rotational kinetic energy based on initial conditions and principal axes inertia tensor.

$$2T = \omega_x^2 I_x + \omega_y^2 I_y + \omega_z^2 I_z$$

$$\frac{\omega_x^2}{2T/I_x} + \frac{\omega_y^2}{2T/I_y} + \frac{\omega_z^2}{2T/I_z} = 1$$

For the given initial conditions, we get semi-major axes of the following lengths:  $\omega_x = 0.2332 \text{ rad s}^{-1}$ ,  $\omega_y = 0.1697 \text{ rad s}^{-1}$ , and  $\omega_z = 0.1524 \text{ rad s}^{-1}$ . These values make sense given the equation for the energy ellipsoid.

Similarly, we compute our surface for the momentum ellipsoid with angular momentum based on our initial conditions and the principal axes inertia tensor.

$$L = \omega_x^2 I_x^2 + \omega_y^2 I_y^2 + \omega_z^2 I_z^2$$

$$\frac{\omega_x^2}{(L/I_x)^2} + \frac{\omega_y^2}{(L/I_y)^2} + \frac{\omega_z^2}{(L/I_z)^2} = 1$$

For the given initial conditions, we get semi-major axes of the following lengths:  $\omega_x = 0.3115 \text{ rad s}^{-1}$ ,  $\omega_y = 0.1649 \text{ rad s}^{-1}$ , and  $\omega_z = 0.1330 \text{ rad s}^{-1}$ . These values make sense given the equation for the momentum ellipsoid and are shown in the plots below.

We plot the energy ellipsoid in Figure 8 and the momentum ellipsoid in Figure 9.

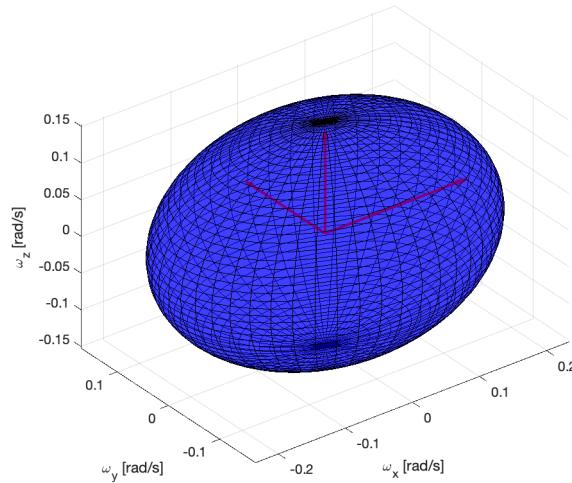


Figure 8: Energy ellipsoid with axes in red

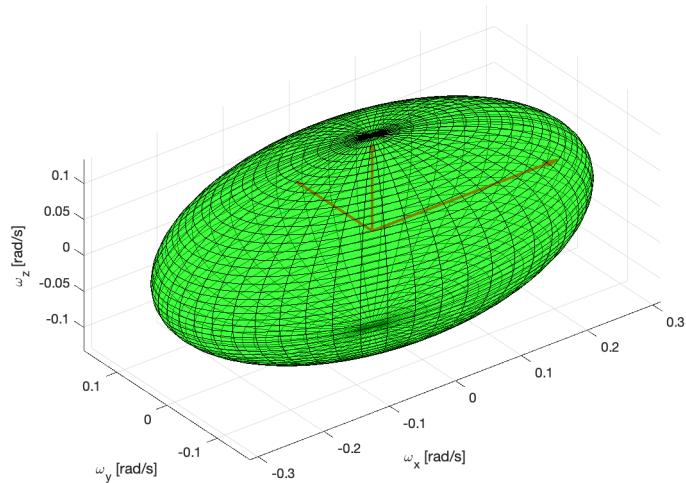


Figure 9: Momentum ellipsoid with axes in red

## 2.7 PROBLEM 7

*Plot polhode in same 3D plot. Verify that it is the intersection between the ellipsoids.*

For a polhode plot to be real, the condition below must be verified.

$$I_x < \frac{L^2}{2T} < I_z$$

Based on previously calculated values ( $I_x = 7707.1$ ,  $\frac{L^2}{2T} = 13752.1$ ,  $I_z = 18050.4$ ) we can verify that the polhode here will be real.

Figure 11 shows that the polhode is indeed the intersection between the ellipsoids.

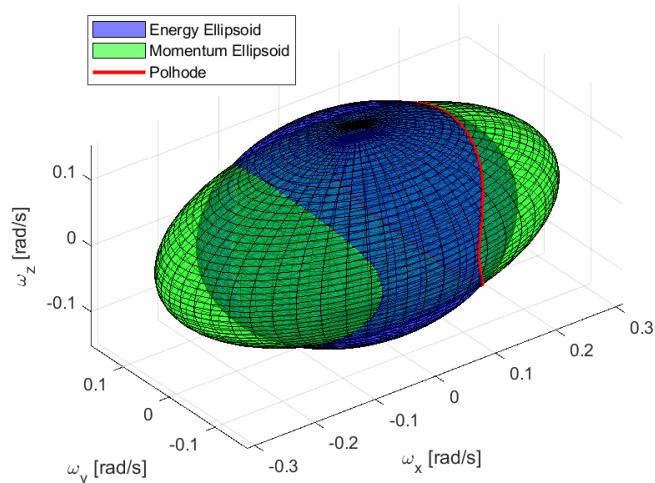


Figure 10: Energy and momentum ellipsoids with polhode

## 2.8 PROBLEM 8

*Plot polhode in three 2D planes identified by principal axes (axis equal). Verify that shapes of resulting conic sections correspond to theory.*

The polhode conic sections in Figure 11 match expected theory. The polhode as seen along the x-axis is an ellipse, while the polhode along the y-axis is a hyperbola. We also see that when seen along the z-axis, the polhode also forms an ellipse, shown as a half-ellipse in our plot.

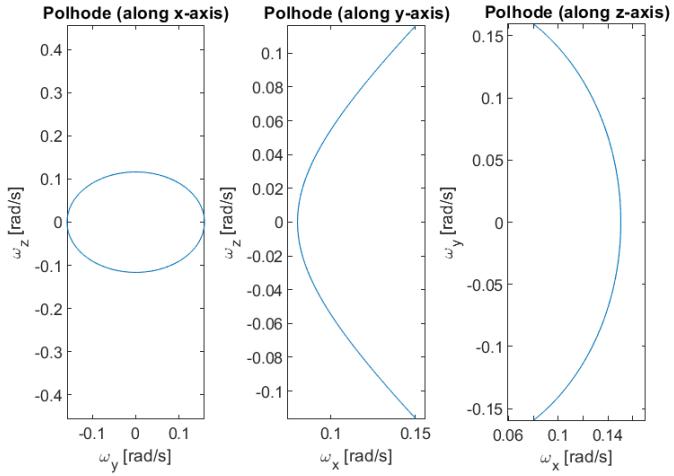


Figure 11: 2D views of polhode

## 2.9 PROBLEM 9

*Repeat above steps changing initial conditions, e.g. setting angular velocity vector parallel to principal axis. Is the behavior according to expectations?*

We show the angular velocity evolution with the initial conditions shown in Table 2.9. Case 1 involves rotation about the principal x-axis, Case 2 involves rotation about the principal y-axis with a slight disturbance, and Case 3 involved rotation about the z-axis with a slight disturbance.

Case	$\omega_x$ (deg/s)	$\omega_y$ (deg/s)	$\omega_z$ (deg/s)
1	8	0	0
2	0.08	8	0.08
3	0.08	0	8

The specifics of Case 1 are shown in the angular velocity plot in Figure 12, the polhode and ellipsoids in Figure 15, and the 2D views of the polhode in Figure 18. The behavior shown is as expected—when the angular velocity is parallel to the principal axis, we do not have coupling with the other components of angular velocity, and the polhode views in 2D become points rather than conic sections.

For Case 2, Figure 13 shows that the satellite's rotational behavior will oscillate as expected, owing to the properties of the intermediate axis. Additionally, Figure 16, and the 2D views in Figure 19 show a larger polhode, with the slight disturbances leading to ellipsoids with a substantial intersection. Interestingly, there seems to be a very sharp hyperbola in the xz-plane of the polhode.

Figure 14 illustrates a slight oscillation of angular velocities about the x- and y-axes in Case 3. Meanwhile, the actual region of intersection in the polhode as shown in Figures 17 and 20 is much smaller than in other cases, but not a single point like in the Case 1.

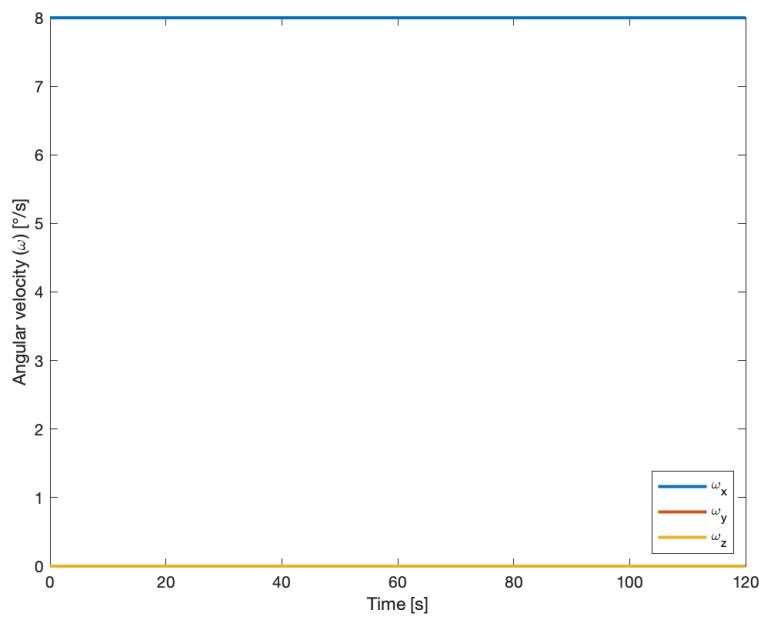


Figure 12: Angular velocity evolution for angular velocity vector for Case 1

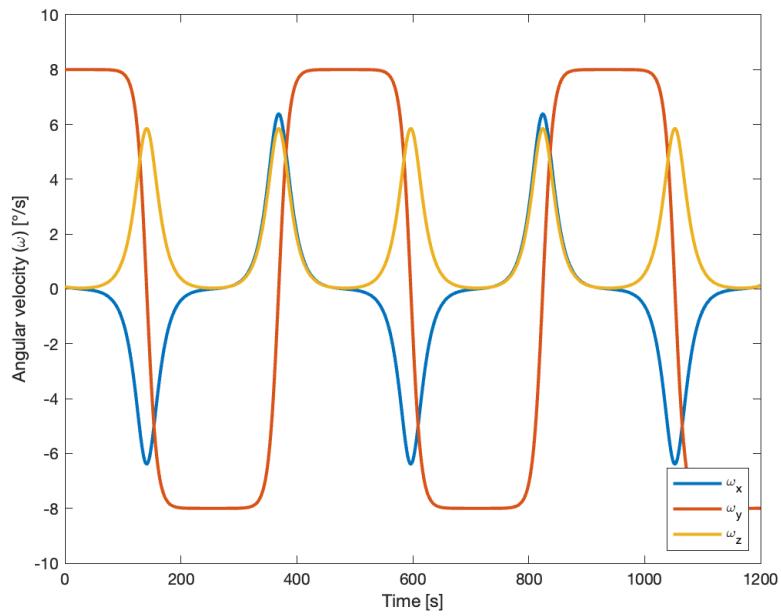


Figure 13: Angular velocity evolution for angular velocity vector for Case 2

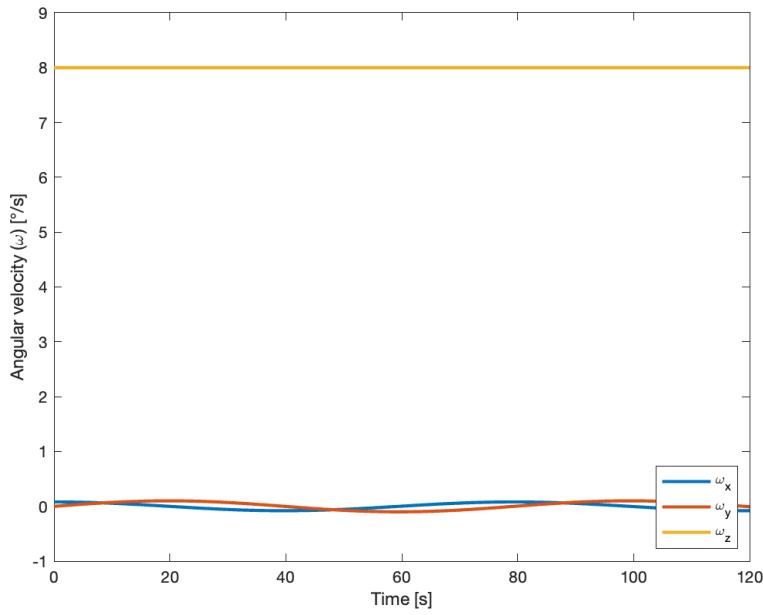


Figure 14: Angular velocity evolution for angular velocity vector for Case 3

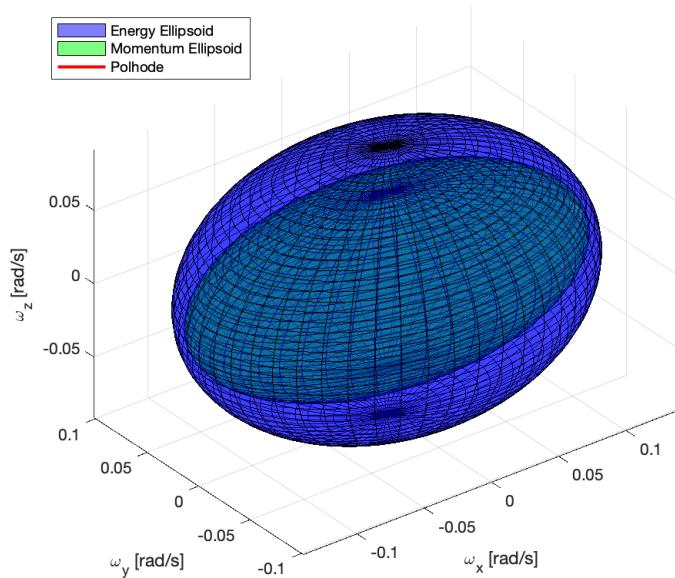


Figure 15: Polhode and ellipsoids for angular velocity vector for Case 1

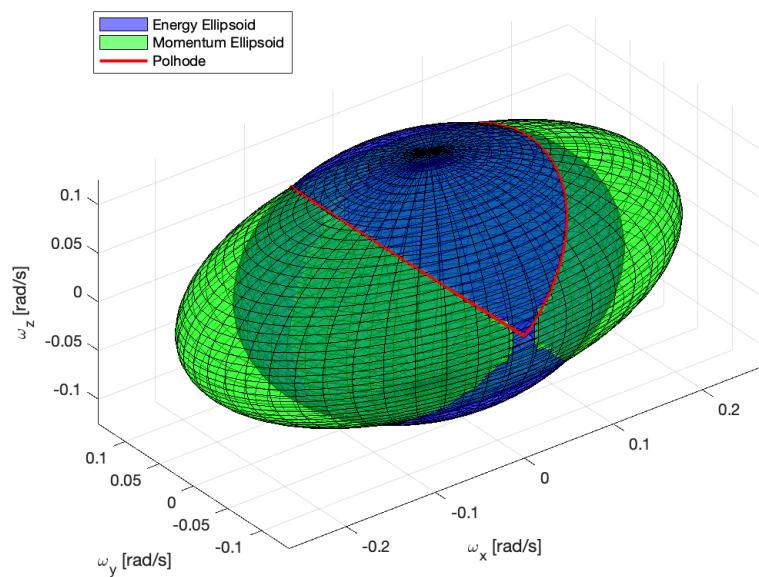


Figure 16: Polhode and ellipsoids for angular velocity vector for Case 2

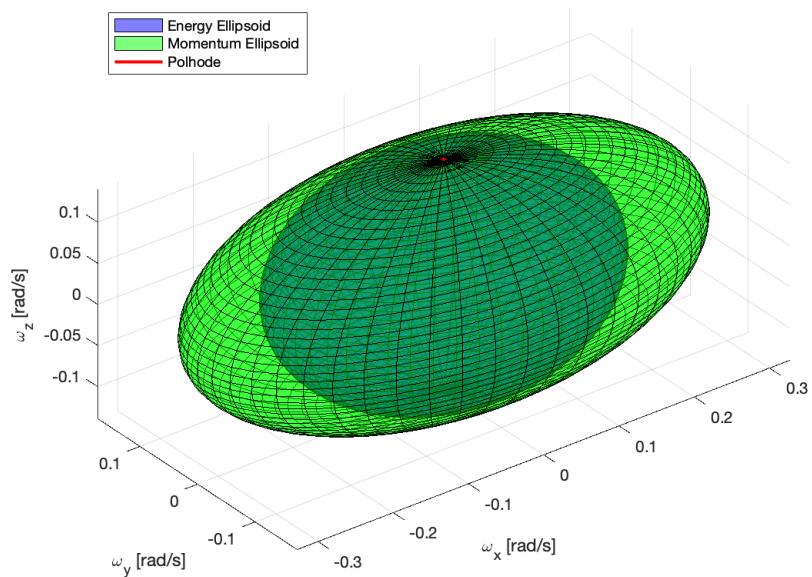


Figure 17: Polhode and ellipsoids for angular velocity vector for Case 3

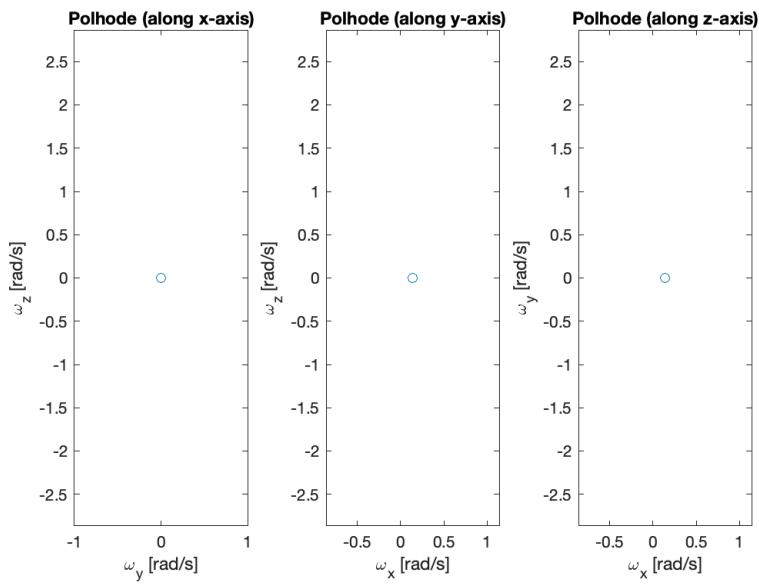


Figure 18: 2D views of polhode for angular velocity vector for Case 1

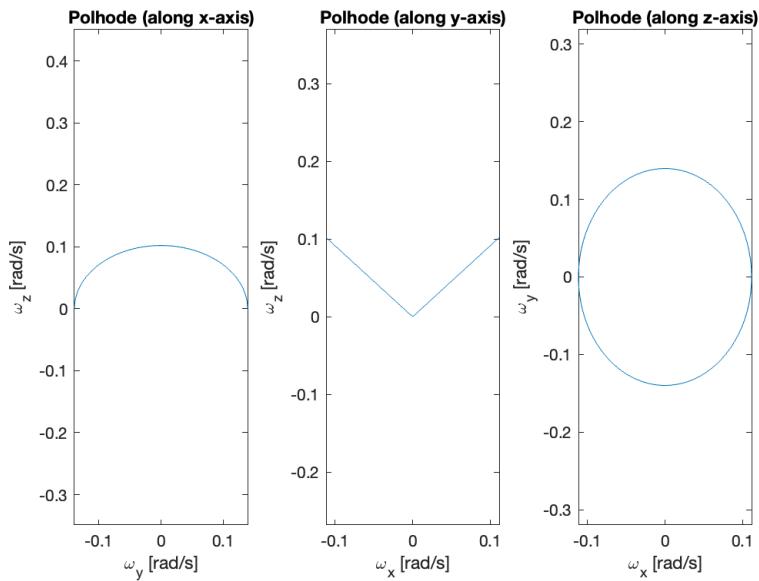


Figure 19: 2D views of polhode for angular velocity vector for Case 2

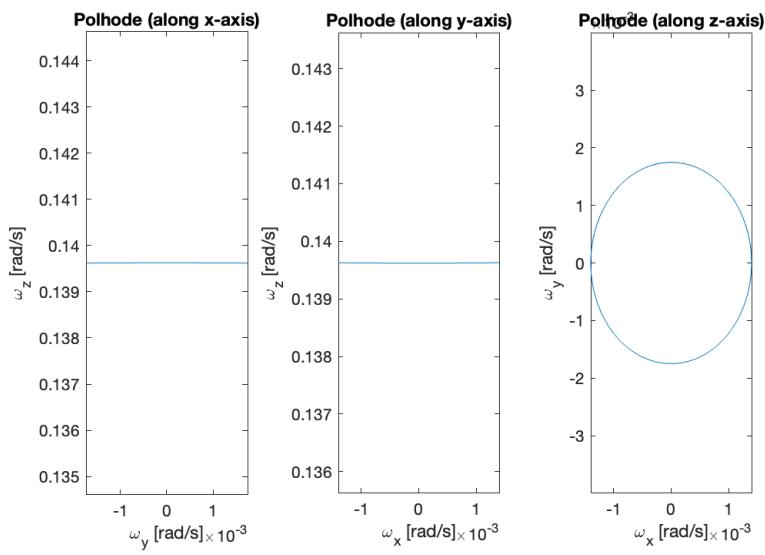


Figure 20: 2D views of polhode for angular velocity vector for Case 3

## 3 PROBLEM SET 3

### 3.1 PROBLEM 1

*Impose that satellite is axial-symmetric ( $I_x = I_y \neq I_z$ ). Repeat numerical simulation from previous pset using initial condition 4) from previous pset.*

Problem 1 was solved by setting  $I_x = I_y = 7707.07 \text{ kg} \cdot \text{m}^2$  and using the same Euler equation solver from Problem Set 2, Problem 5 with the same initial conditions ( $\omega_x = 8^\circ \text{ s}^{-1}$ ,  $\omega_y = 4^\circ \text{ s}^{-1}$ ).

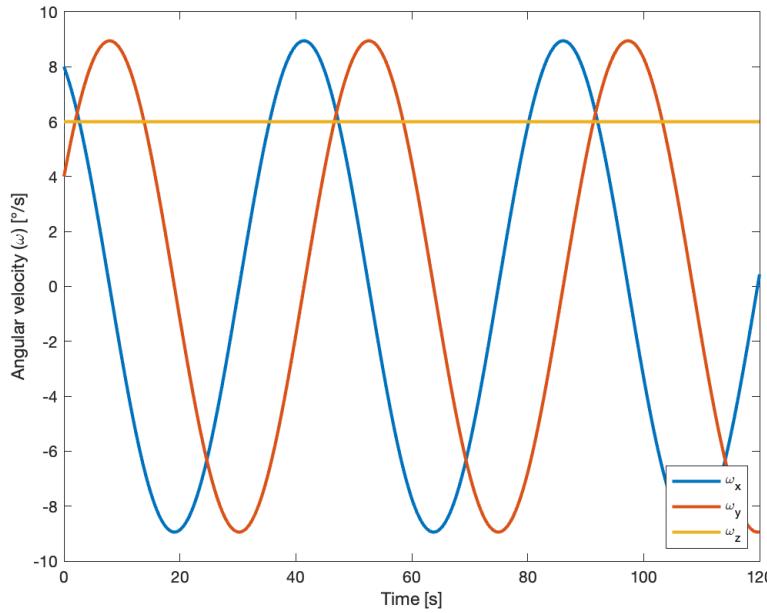


Figure 21: Numerical solution results

### 3.2 PROBLEM 2

*Program analytical solution for axial-symmetric satellite. Compute it at same time steps and from same initial conditions.*

The analytical solution to the Euler equations for an axial-symmetric satellite is based on variables  $\lambda$  and  $\omega_{xy}$ , as defined below.

$$\lambda = \frac{I_z - I_x}{I_x} \omega_{z_0}$$

$$\omega_{xy} = (\omega_{x_0} + i\omega_{y_0}) e^{i\lambda t}$$

We take the real and imaginary parts of this result to obtain an analytical solution.

$$\omega_x = \operatorname{Re}(\omega_{xy})$$

$$\omega_y = \operatorname{Im}(\omega_{xy})$$

$$\omega_z = \omega_{z_0}$$

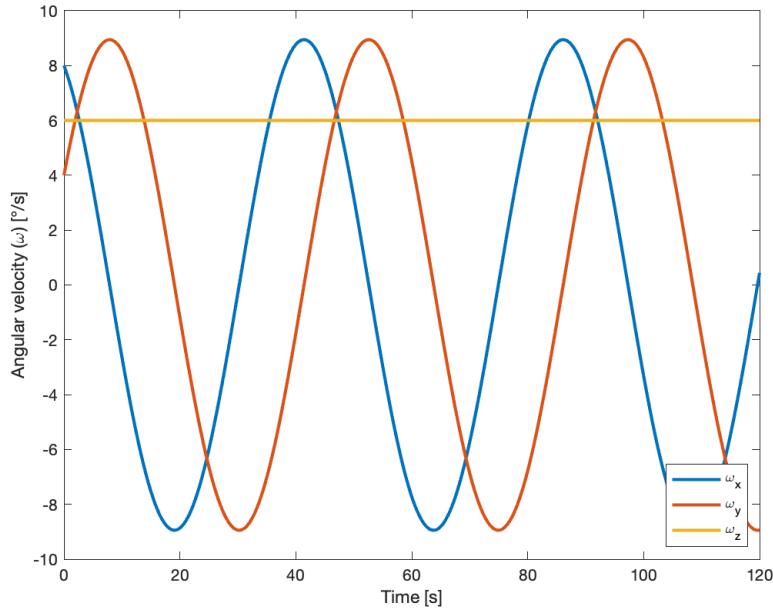


Figure 22: Analytical solution results

### 3.3 PROBLEM 3

*Compare numerical and analytical solutions. Plot differences (errors), do not only superimpose absolute values. Tune numerical integrator for large discrepancies. Are angular velocity vector and angular momentum vector changing according to theory in principal axes?*

Figure 23 is the error between the numerical and analytical solutions. We observe that the error is very small, thus our numerical solution is a good candidate.

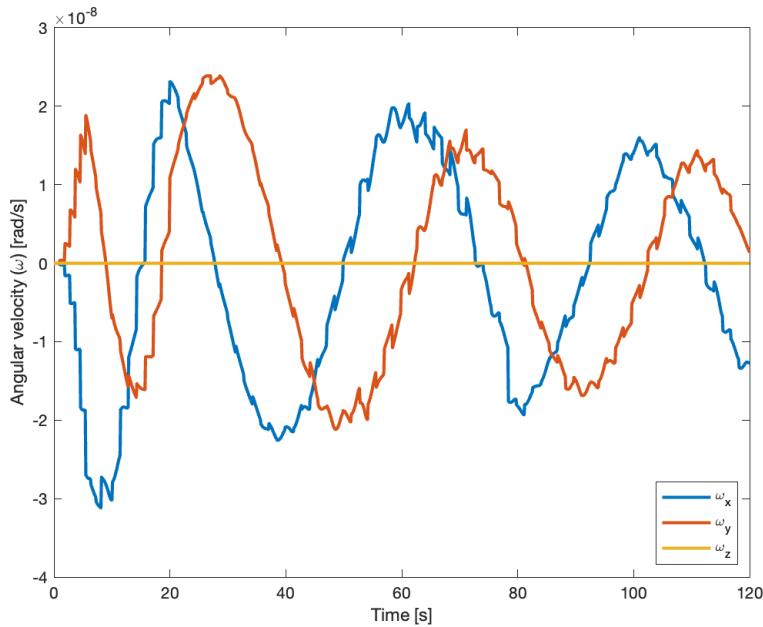


Figure 23: Error between numerical and analytical solutions

The angular velocity vector and angular momentum vectors rotate in a plane, offset at a constant angle from the z-axis, as observed in Figure 24. This matches the expected theoretical behavior.

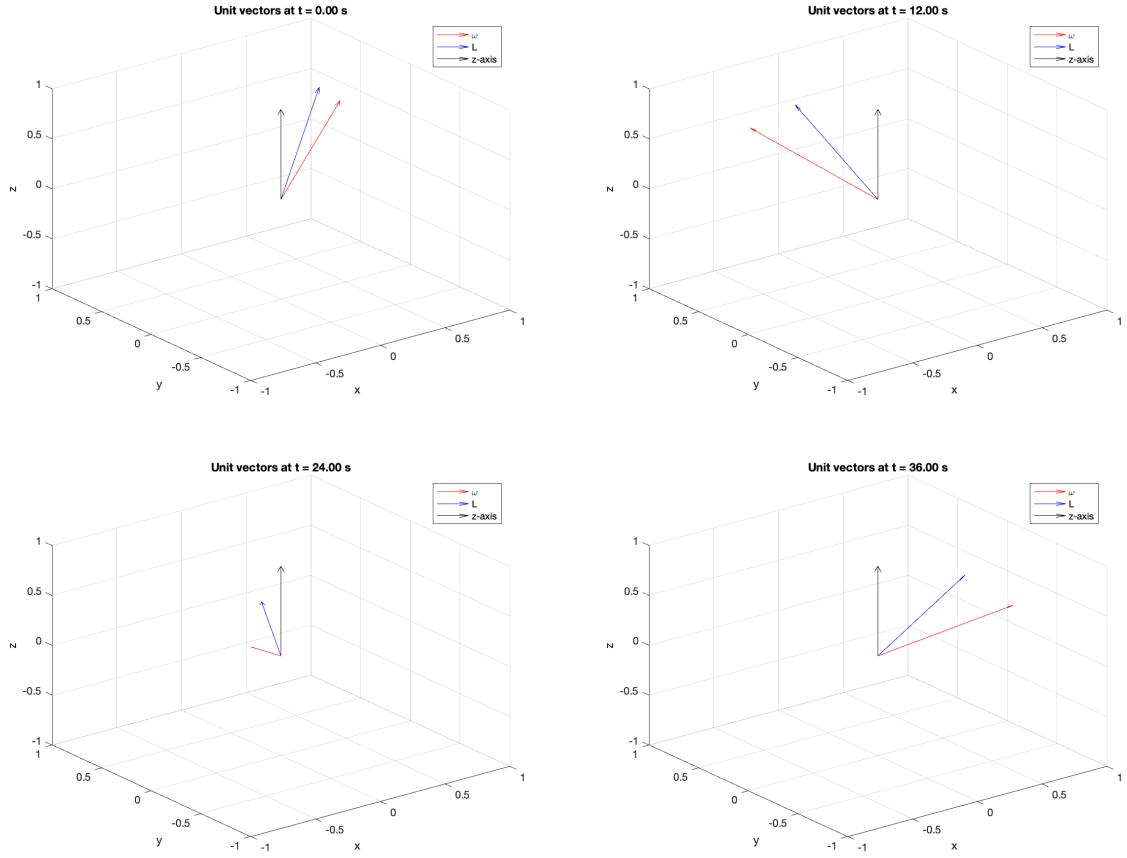


Figure 24: Angular velocity (red) and angular momentum (blue) unit vectors over time.

### 3.4 PROBLEM 4

*Program Kinematic equations of motion correspondent to a nominal attitude parameterization of your choice.*

We choose a nominal attitude parameterization of quaternions, our choice being based on the absence of singularities. The following function computes the time derivative for a state consisting of quaternions (4 parameters) and angular velocity (3 parameters).

The equations below describe the propagation of kinematics using quaternions.

$$\vec{\Omega} = \begin{bmatrix} 0 & \omega_z & -\omega_y & \omega_x \\ -\omega_z & 0 & \omega_x & \omega_y \\ \omega_y & -\omega_x & 0 & \omega_z \\ -\omega_x & -\omega_y & -\omega_z & 0 \end{bmatrix}$$

$$\frac{d\vec{q}}{dt} = \frac{1}{2}\vec{\Omega}\vec{q}(t)$$

The following script shows the computation of the time derivative for quaternions

```

1 function stateDot = kinQuaternion(t,state,Ix,Iy,Iz)
2 % Computes state derivatives for quaternions, angular velocity
3 % Assign variables
4 q = state(1:4);
5 wx = state(5);
6 wy = state(6);
7 wz = state(7);
8
9 stateDot = zeros(7,1);
10 % Angular velocity time derivatives
11 stateDot(5) = (Iy - Iz) / Ix * wy * wz;
12 stateDot(6) = (Iz - Ix) / Iy * wz * wx;
13 stateDot(7) = (Ix - Iy) / Iz * wx * wy;
14 sigma = [0, wz, -wy, wx; ...
15 -wz, 0, wx, wy; ...
16 wy, -wx, 0, wz; ...
17 -wx, -wy, -wz, 0];
18 % Quaternion time derivative
19 qDot = 0.5 * sigma * q;
20 stateDot(1:4) = qDot;
21 end

```

We can use the previous function to perform a forward Euler numerical integration. We call the previous function over a fixed time step to compute the evolution of the state.

```

1 function [q,w] = kinQuaternionForwardEuler(q0,w0,Ix,Iy,Iz,tFinal,tStep)
2 % Forward Euler integration for quaternions, angular velocity
3 nStep = ceil(tFinal/tStep);
4 q = nan(nStep+1,4);
5 w = nan(nStep+1,3);
6 q(1,:) = q0';
7 w(1,:) = w0';
8 for i = 1:nStep
9 t = i * tStep;
10 qi = q(i,:)';
11 wi = w(i,:)';
12 state = [qi;wi];
13 stateDot = kinQuaternion(t,state,Ix,Iy,Iz);
14 nextState = state + tStep * stateDot;
15 q(i+1,:) = nextState(1:4) / norm(nextState(1:4));
16 w(i+1,:) = nextState(5:7);
17 end
18 end

```

For improved precision, we implement a 4th order Runge-Kutta method, which uses a weighted sum of slopes to obtain a better result. This also calls the time derivative function, but does so with different values of the state, which are weighted to obtain the next state for each step.

```

1 function [q,w] = kinQuaternionRK4(q0,w0,Ix,Iy,Iz,tFinal,tStep)
2 % 4th order Runge-Kutta integration for quaternions, angular ...
3 % velocity
4 nStep = ceil(tFinal/tStep);
5 q = nan(nStep+1,4);
6 w = nan(nStep+1,3);

```

```

6      q(1,:) = q0';
7      w(1,:) = w0';
8      for i = 1:nStep
9          t = i * tStep;
10         qi = q(i,:)';
11         wi = w(i,:)';
12         state = [qi;wi];
13         k1 = kinQuaternion(t,state,Ix,Iy,Iz);
14         k2 = kinQuaternion(t+tStep/2,state+(k1*tStep/2),Ix,Iy,Iz);
15         k3 = kinQuaternion(t+tStep/2,state+(k2*tStep/2),Ix,Iy,Iz);
16         k4 = kinQuaternion(t+tStep,state+(k3*tStep),Ix,Iy,Iz);
17         nextState = state + tStep * (k1/6 + k2/3 + k3/3 + k4/6);
18         q(i+1,:) = nextState(1:4) / norm(nextState(1:4));
19         w(i+1,:) = nextState(5:7);
20     end
21 end

```

### 3.5 PROBLEM 5

*Program Kinematic equations of motion correspondent to a different attitude parameterization from the previous step. This is used for comparison, to get familiar with different approaches, and as fall back solution in the case of singularities.*

Similarly, we create a function that computes the time derivative of a state consisting of Euler angles and angular velocity using the 3-1-3 symmetric Euler angle sequence.

The equations for the propagation of kinematics for Euler angles are below.

$$\begin{aligned}\frac{d\phi}{dt} &= \frac{\omega_x \sin(\psi) + \omega_y \cos(\psi)}{\sin(\theta)} \\ \frac{d\theta}{dt} &= \omega_x \cos(\psi) - \omega_y \sin(\psi) \\ \frac{d\psi}{dt} &= \omega_z - (\omega_x \sin(\psi) + \omega_y \cos(\psi)) \cot(\theta)\end{aligned}$$

```

1 function stateDot = kinEulerAngle(t,state,Ix,Iy,Iz)
2 % Computes state derivative for Euler angles, angular velocity
3 % Assign variables
4 phi = state(1);
5 theta = state(2);
6 w = state(4:6);
7
8 stateDot = zeros(6,1);
9 % Angular velocity time derivatives
10 stateDot(4) = (Iy - Iz) / Ix * w(2) * w(3);
11 stateDot(5) = (Iz - Ix) / Iy * w(3) * w(1);
12 stateDot(6) = (Ix - Iy) / Iz * w(1) * w(2);
13 % Euler angle time derivatives
14 % 312
15 EPrimeInv = [sin(phi)*sin(theta) cos(phi)*sin(theta) cos(theta); ...
16               cos(theta)*cos(phi) -sin(phi)*cos(theta) 0; ...
17               sin(phi) cos(phi) 0] * (1 / cos(theta));
18 % 313
19 % EPrimeInv = [-sin(phi)*cos(theta) -cos(phi)*cos(theta) ...
20               sin(theta); ...

```

```

20      %      cos(phi)*sin(theta) -sin(phi)*sin(theta) 0; ...
21      %      sin(phi) cos(phi) 0] * (1 / sin(theta));
22      stateDot(1:3) = EPrimeInv * w;
23 end

```

We can propagate this with forward Euler, as in the previous section.

```

1 function [state] = ...
kinEulerAngleForwardEuler(state0,Ix,Iy,Iz,tFinal,tStep)
2 % Forward Euler integration for state Euler angles, angular velocity
3 nStep = ceil(tFinal/tStep);
4 state = nan(nStep+1,6);
5 state(1,:) = state0;
6 for i = 1:nStep
7     t = i * tStep;
8     statei = state(i,:)';
9     stateDot = kinEulerAngle(t,statei,Ix,Iy,Iz);
10    state(i+1,:) = statei + tStep * stateDot;
11 end
12 end

```

For our actual implementation, we choose to use the time derivative function with `ode113` for improved accuracy. Note that this cannot be done as simply for quaternions, as they require normalization at each step, hence our decision to implement RK4.

### 3.6 PROBLEM 6

*Numerically integrate Euler AND Kinematic equations from arbitrary initial conditions (warning: stay far from singularity of adopted parameterization). Multiple revolutions. The output is the evolution of the attitude parameters over time. These attitude parameters describe orientation of principal axes relative to inertial axes.*

We integrate our attitude parameterizations (including angular velocity using Euler equations).

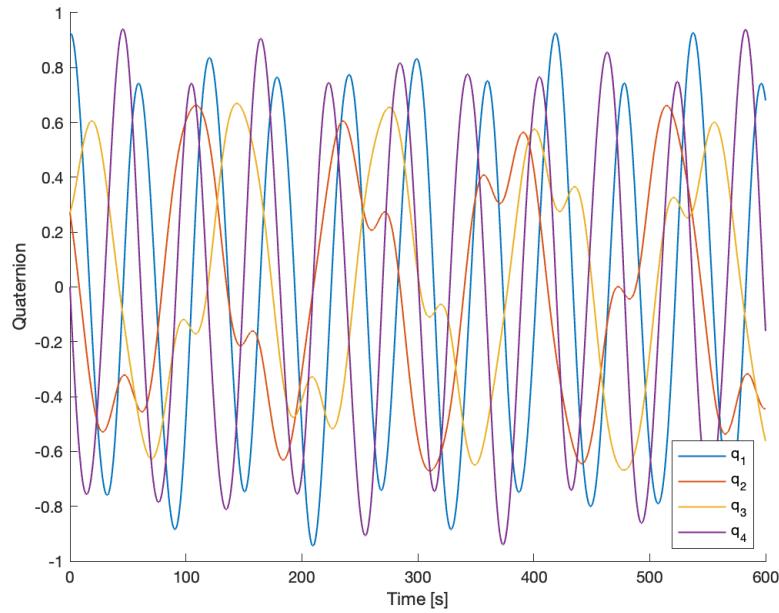


Figure 25: Evolution of quaternions

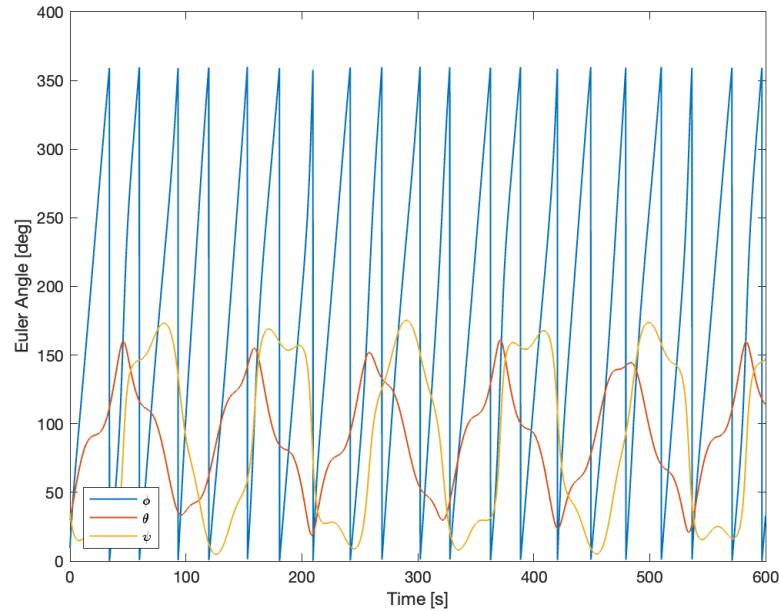


Figure 26: Evolution of Euler angles

### 3.7 PROBLEM 7

a. Compute angular momentum vector in inertial coordinates and verify that it is constant (not only its magnitude as in PS2) by plotting its components.

Figure 27 shows the components of the angular momentum vector over time, as computed from our primary attitude representation of quaternions. The angular momentum vector (and its individual components) remains constant in the absence of external torques.

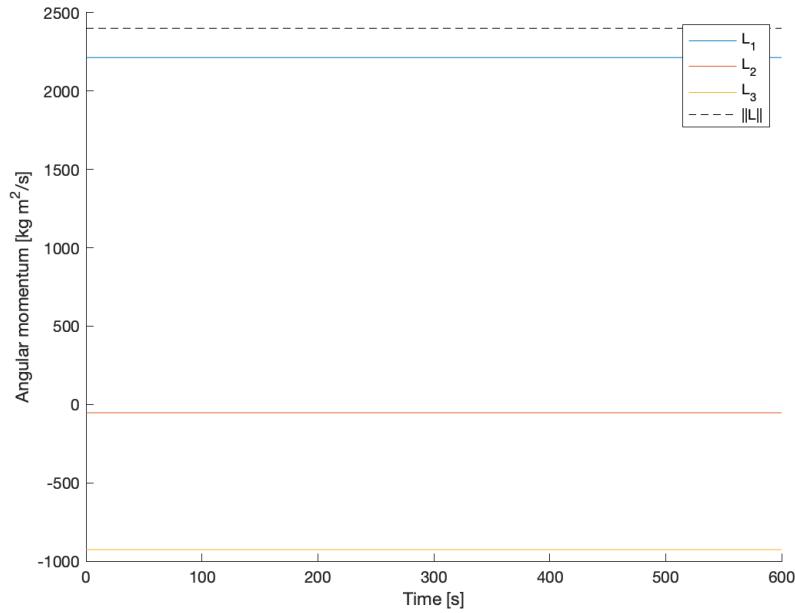


Figure 27: Angular momentum in inertial coordinates is constant

b. Compute angular velocity vector in inertial coordinates and plot the herpolhode in 3D (line drawn in inertial space by angular velocity). Is the herpolhode contained in a plane perpendicular to the angular momentum vector? Show it.

Figure 28 shows the angular momentum and angular velocity vectors overlaid with the herpolhode. The animation (see caption) shows the evolution of the herpolhode and provides a better visualization of the herpolhode's orientation in a plane perpendicular to the angular momentum vector.

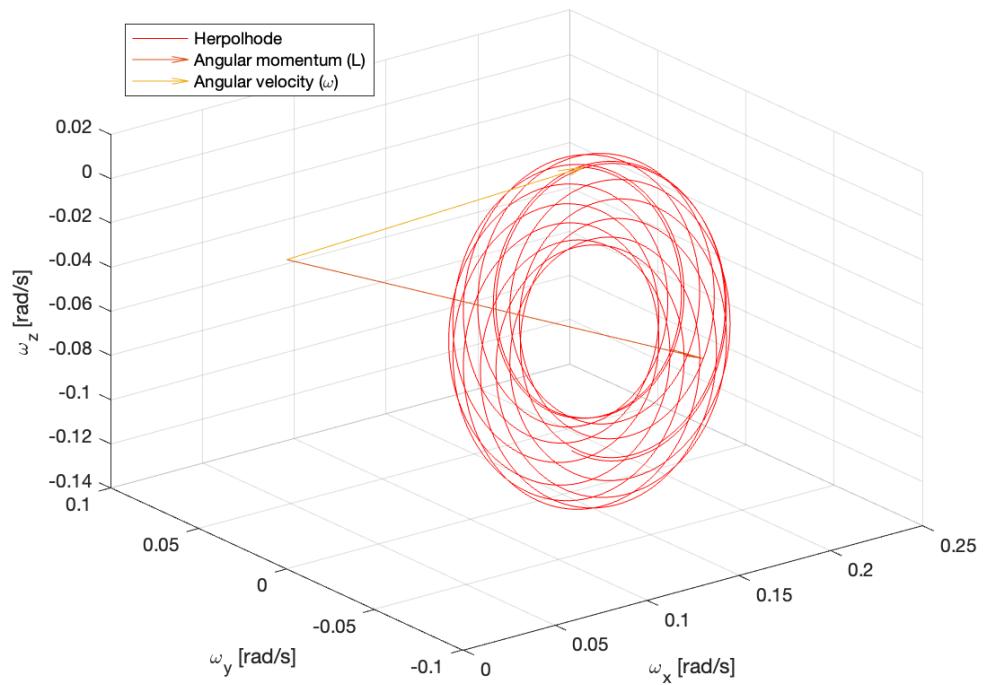


Figure 28: Herpolhode (Animated: <https://tinyurl.com/herpolhode>)

c. Compute and plots unit vectors of orbital frame, body axes, and principal axes in 3D as a function of time in inertial coordinates. (Be creative on how to show moving vectors in 3D).

Figures 29, 30, and 31 include the plots of the orbital (RTN), body, and principal axes over the course of a single orbit. The RTN frame varies with rotation about the orbit, while rotation can be seen in the body and principal axis plots based on the rotation we chose in our initial condition.

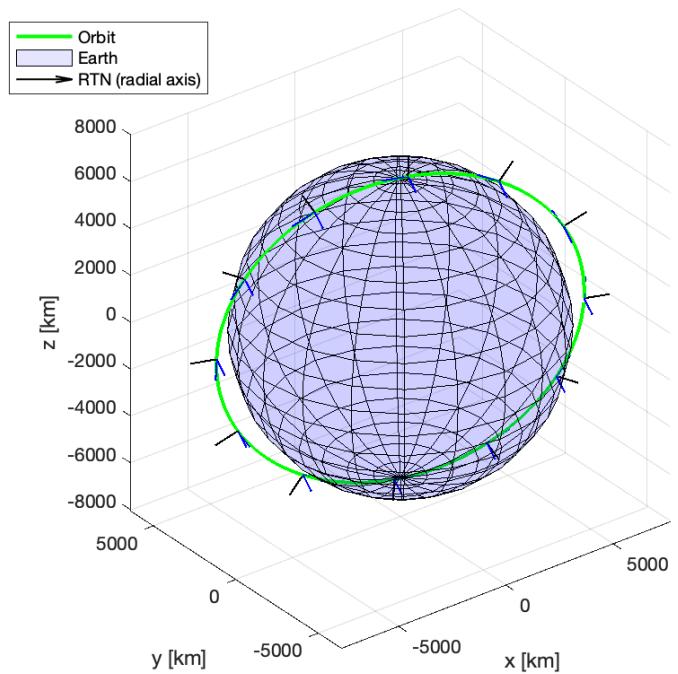


Figure 29: Propagation of RTN frame

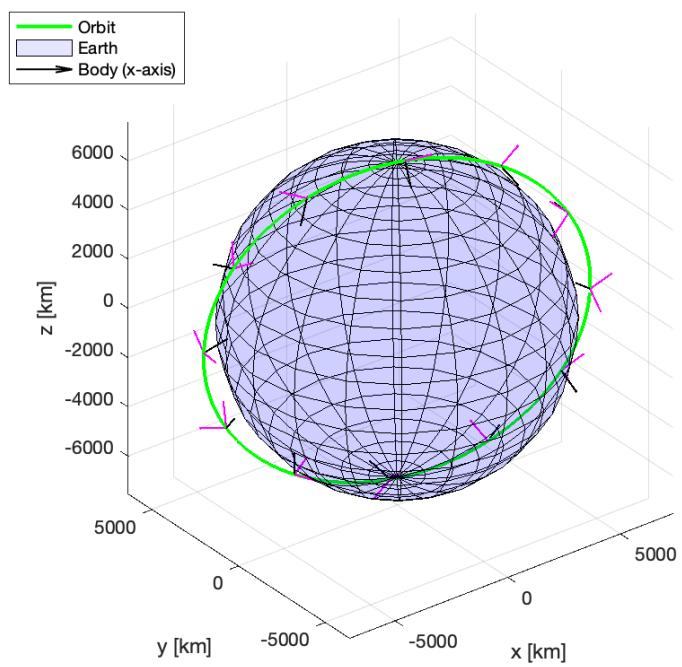


Figure 30: Propagation of body axes

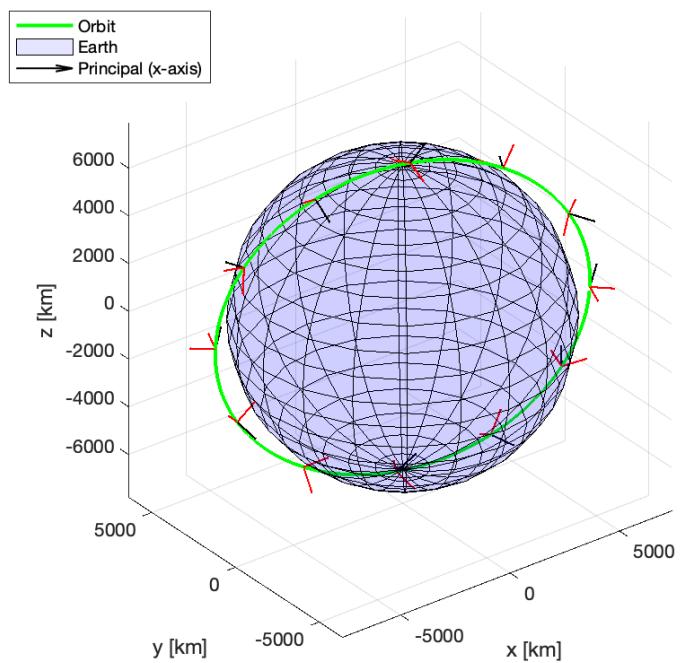


Figure 31: Propagation of principal axes

## 4 PROBLEM SET 4

### 4.1 PROBLEM 1

*Equilibrium tests*

- a. Assume that 2 components of the initial angular velocities are zero and that the principal axes are aligned with the inertial frame (e.g., zero Euler angles). Verify that during the simulation the 2 components of angular velocity remain zero and that the attitude represents a pure rotation about the rotation axis (e.g., linearly increasing Euler angle). Plot velocities and angles.

To test the equilibrium state, we choose the following initial angular velocity,

$$\vec{\omega} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \text{ rad s}^{-1},$$

and we set all Euler angles to zero. We use a 312 convention for Euler angles, which avoids singularities for this configuration.

Figure 32 shows results of the simulation, where  $\omega_x$  and  $\omega_y$  remain zero and  $\omega_z$  maintains a constant value.

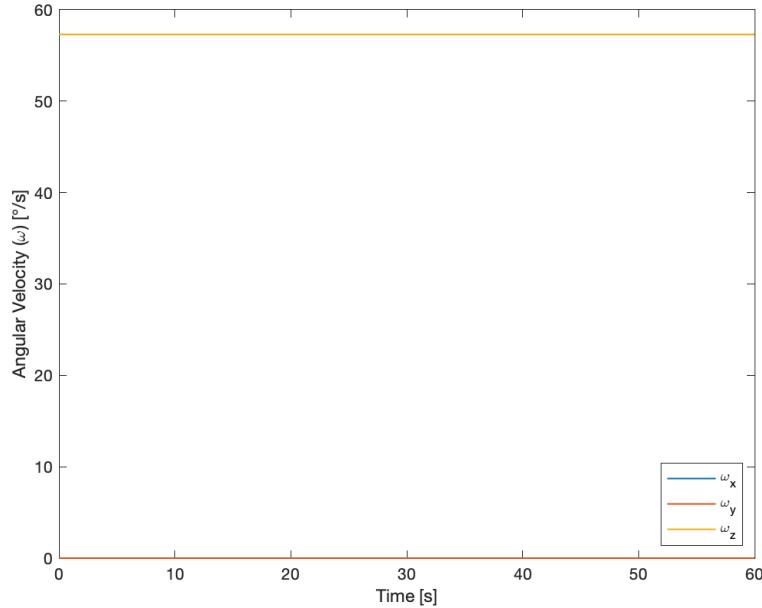


Figure 32: Evolution of angular velocity

Similarly, in Figure 33,  $\phi$  and  $\theta$  Euler angles remain at zero while the  $\psi$  Euler angle increases linearly.

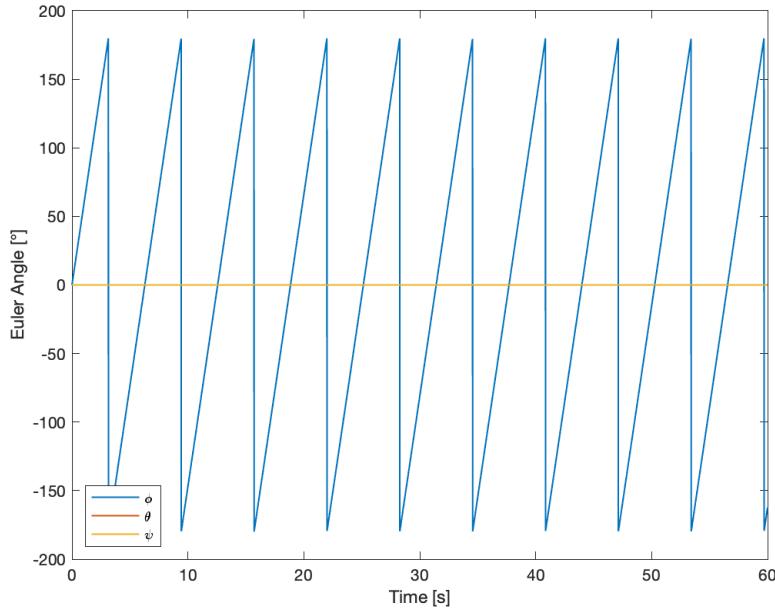


Figure 33: Evolution of Euler angles

*b. Repeat a. by setting the initial attitude to match the RTN frame. Set the initial angular velocity to be non-zero only about N. Show the evolution of attitude motion in the RTN frame and give an interpretation of the results (recall that you might have J2 effects in orbit propagation, consider removing them for verification).*

We choose to align our principal axes with the RTN frame. For selected initial orbital conditions taken from the NISAR science users' handbook, we obtain the initial position and compute the RTN frame, which we then use to find initial aligned Euler angles.

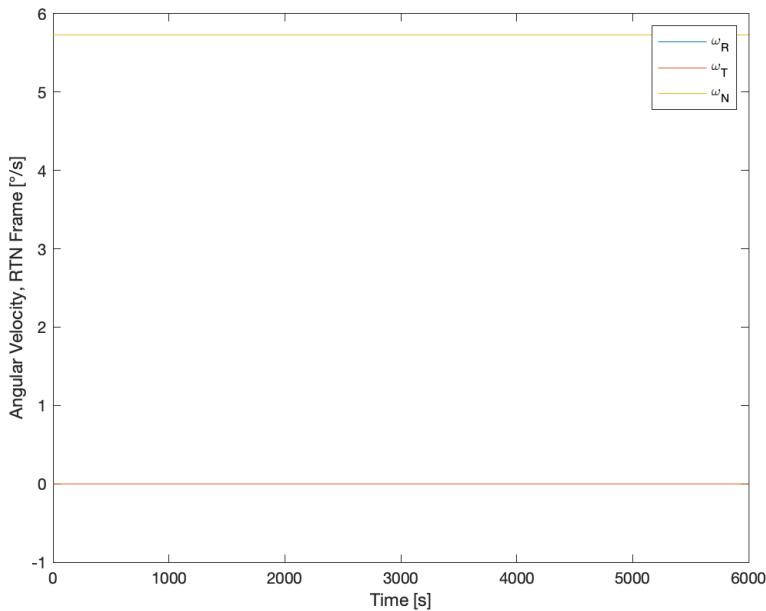


Figure 34: Evolution of angular velocity

We set a nonzero  $\omega_z$  angular velocity, which is aligned with the normal direction of the RTN frame, and we choose all other angular velocities to be zero. Propagating the orbit and attitude, we find that the angular velocity remains constant throughout the orbit, even relative to the RTN frame, as seen in Figure 34. From Figure 35, we see that the  $\theta$  and  $\psi$  Euler angles related to the RTN frame are constant while  $\phi$  varies.

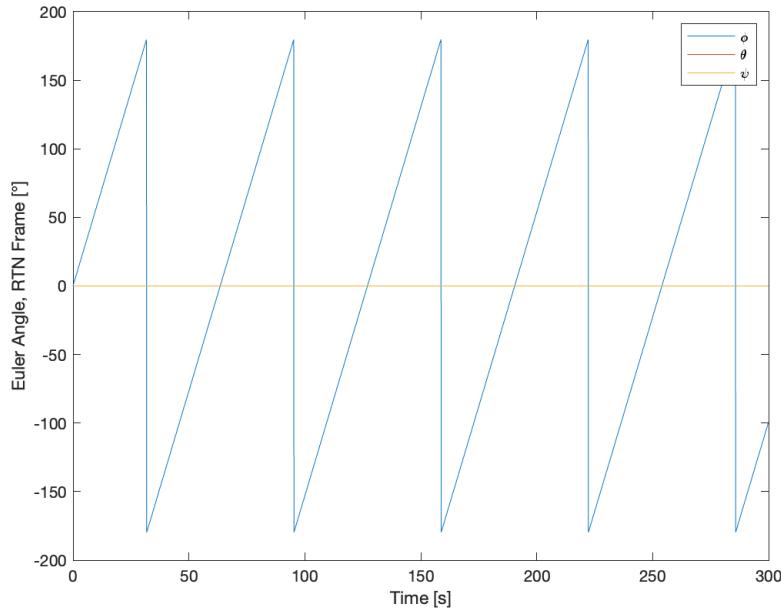


Figure 35: Evolution of Euler angles

This result shows our satellite's maximum inertia principal axis remains aligned with the normal direction of the orbit such that the maximum inertia principal axis remains normal to the plane of the orbit, given the initial condition that axes are aligned with the RTN frame and angular velocity along other axes is nonzero.

## 4.2 PROBLEM 2

### Stability tests

a. Pretend you have a single-spin satellite. Set initial conditions to correspond alternatively to the 3 possible equilibrium configurations (rotation about principal axes of inertia). Slightly perturb initial condition. Is the attitude stable or unstable? In angles and velocities? If stable, periodically or asymptotically? Show it.

For a single spin satellite, the three possible equilibrium configurations are rotation about the minimum inertia principal axis, rotation about the intermediate axis, and rotation about the maximum inertia principal axis. Figures 36, 37, 38 show that the Euler angles are stable about the minimum and maximum axes, but it is unstable about the intermediate axes. This is as expected for our system, with the minimum and maximum axes exhibiting periodic stability with small oscillations in angular velocity.

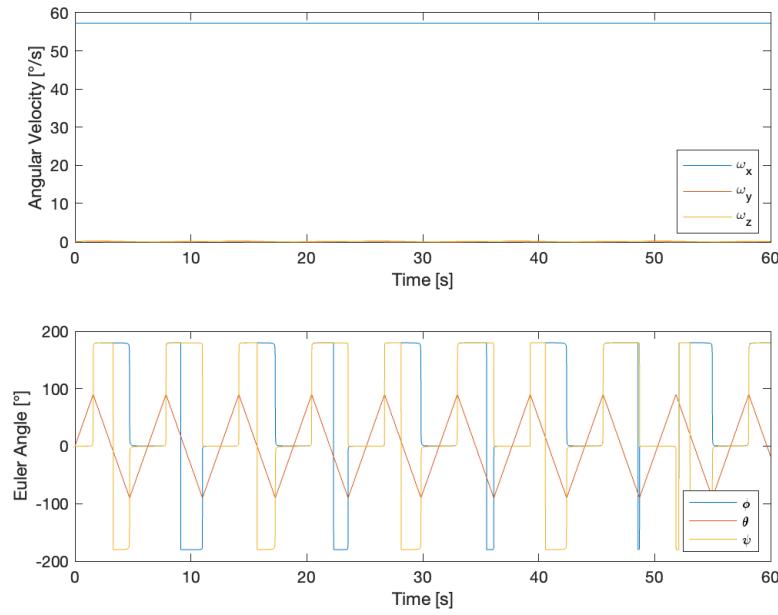


Figure 36: Simulation of satellite spinning on its minimum principal axis

Note that while in Figure 36 the angular velocities are periodically stable, the Euler angles oscillate. This is likely a consequence of the sequence of rotations used for our choice of Euler angle convention.

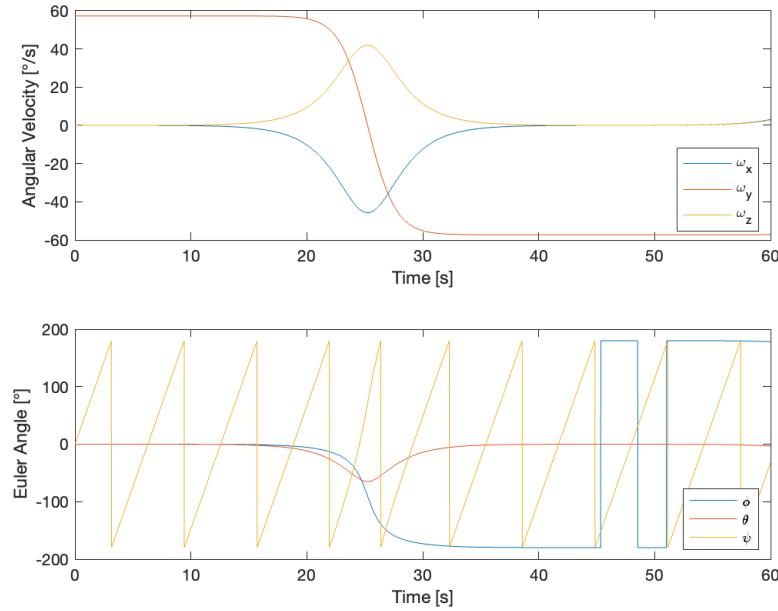


Figure 37: Simulation of satellite spinning on its intermediate principal axis

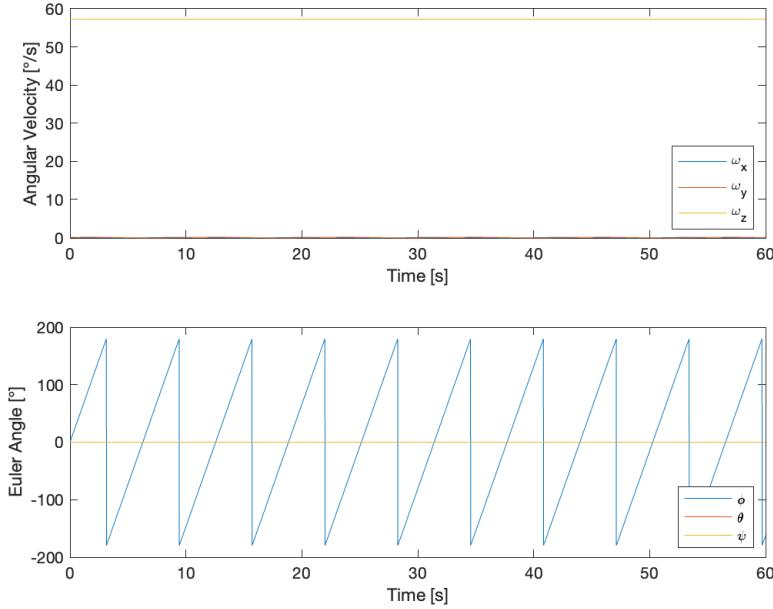


Figure 38: Simulation of satellite spinning on its maximum principal axis

### 4.3 PROBLEM 3

*Adding a momentum wheel or rotor (dual-spin satellite)*

*a. Re-program Euler equations to include a generic momentum wheel or rotor with rotation axis aligned with one of the principal axes of inertia. Ideally the wheel or rotor has specs representative of commercial products (inertia, rotational speed).*

We choose to use specifications from the RSI 68 momentum wheel, for which datasheets are readily available online [11]. This particular momentum wheel is intended for spacecraft in the 1,500 to 5,000 kg range, which matches our mission. We use a diameter of 347 mm and mass of 8.9 kg and model our reaction wheel as a hoop with mass concentrated about the outer diameter. We also use 2,500 RPM, which yields approximately the nominal angular momentum from the datasheet—the maximum angular velocity is 6,000 RPM. The following Euler equations are used to model a momentum wheel as a rotor. For this problem, we set the torques on the right side of each equation to zero, as we are not considering external torques.

$$\begin{aligned}
 I_x \dot{\omega}_x + I_r \dot{\omega}_r r_x + (I_z - I_y) \omega_y \omega_z + I_r \omega_r (\omega_y r_z - \omega_z r_y) &= M_x \\
 I_y \dot{\omega}_y + I_r \dot{\omega}_r r_y + (I_x - I_z) \omega_z \omega_x + I_r \omega_r (\omega_z r_x - \omega_x r_z) &= M_y \\
 I_z \dot{\omega}_z + I_r \dot{\omega}_r r_z + (I_y - I_x) \omega_x \omega_y + I_r \omega_r (\omega_x r_y - \omega_y r_x) &= M_z \\
 I_r \dot{\omega}_r &= M_r
 \end{aligned}$$

The function `kinEulerAngleWheel`, shown below, is used in addition to `ode113` to simulate the angular velocities over time.

```

1 function stateDot = kinEulerAngleWheel(t,state,M,r,Ix,Iy,Iz,Ir)
2 % Computes state derivative for Euler angles, angular velocity

```

```

3      % Adds momentum wheel
4      % Assign variables
5      phi = state(1);
6      theta = state(2);
7      w = state(4:7);
8
9      stateDot = zeros(7,1);
10     % Angular velocity time derivatives
11     wDot = eulerEquationWheel(t,w,M,r,Ix,Iy,Iz,Ir);
12     stateDot = zeros(7,1);
13     stateDot(4) = wDot(1);
14     stateDot(5) = wDot(2);
15     stateDot(6) = wDot(3);
16     stateDot(7) = wDot(4);
17     % Euler angle time derivatives
18     % 312
19     EPrimeInv = [sin(phi)*sin(theta) cos(phi)*sin(theta) cos(theta); ...
20                  cos(theta)*cos(phi) -sin(phi)*cos(theta) 0; ...
21                  sin(phi) cos(phi) 0] * (1 / cos(theta));
22     % 313
23     % EPrimeInv = [-sin(phi)*cos(theta) -cos(phi)*cos(theta) ...
24     %                 sin(theta); ...
25     %                 cos(phi)*sin(theta) -sin(phi)*sin(theta) 0; ...
26     %                 sin(phi) cos(phi) 0] * (1 / sin(theta));
27     stateDot(1:3) = EPrimeInv * w(1:3);
28 end

```

b. Numerically integrate Euler AND Kinematic equations from equilibrium initial condition. Verify that integration is correct as from previous tests (conservation laws, rotations, etc.).

Simulating the Euler and kinematic equations, Figure 39 shows the angular momentum vector remain constant in the inertial frame, as expected.

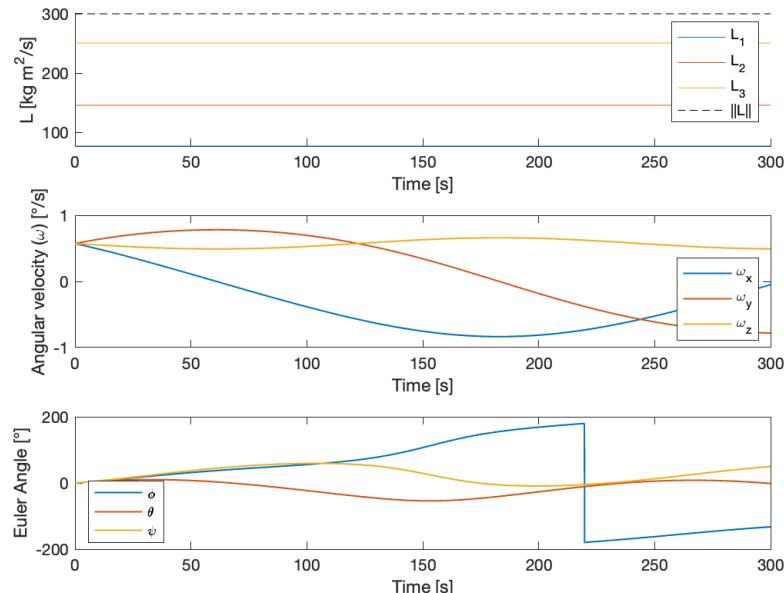


Figure 39: Angular momentum conserved with angular momentum components constant

c. Verify equilibrium and its stability similar to previous pset.

By linearizing the Euler equations about equilibrium, the following result shows that periodic stability (in this case, about the z-axis) can be met with a reaction wheel if one of the following conditions are met:

- 1)  $I_r\omega_r > (I_y - I_z)\omega_z$  AND  $I_r\omega_r > (I_x - I_z)\omega_z$
- 2)  $I_r\omega_r < (I_y - I_z)\omega_z$  AND  $I_r\omega_r < (I_x - I_z)\omega_z$

We demonstrate equilibrium and stability for this new system with the reaction wheel. Figures 40, 41, 42 show the analysis for each of the principal axes. As before, the minimum and maximum inertia principal axes are periodically stable, but the intermediate axis is unstable.

This behavior resembles that found in Problem 2 above. In this case, we perturb each angular velocity as well as the rotor angular velocity. Our rotor velocity in this case is not enough to stabilize spin about the intermediate axis.

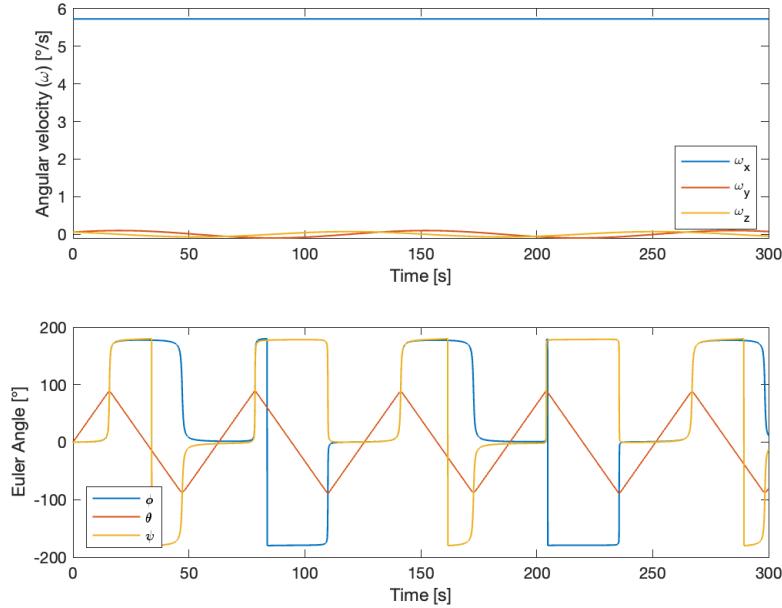


Figure 40: Stability analysis about minimum inertia principal axis

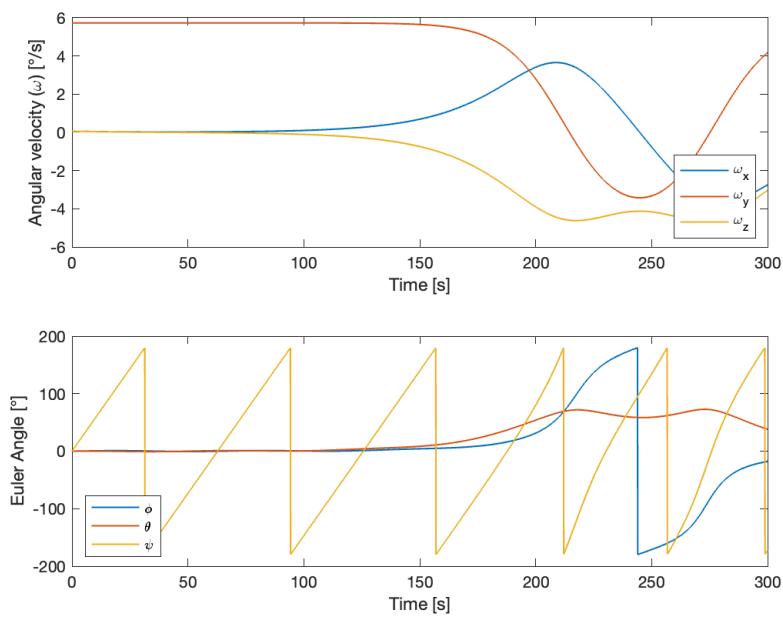


Figure 41: Stability analysis about intermediate axis

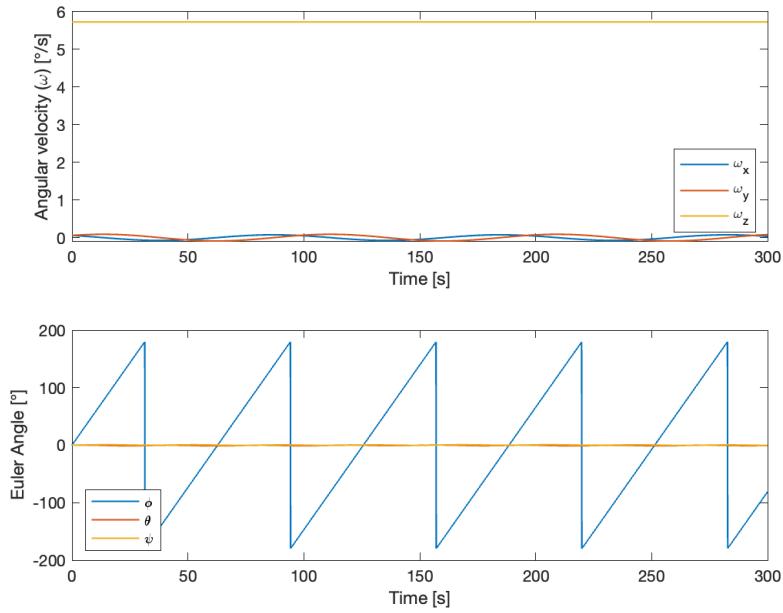


Figure 42: Stability analysis about maximum inertia principal axis

d. Use the stability condition to make attitude motion stable for rotation about intermediate moment of inertia by changing moment of inertia and/or angular velocity of the momentum wheel or rotor.

By increasing the angular velocity of the rotor by a factor of 10, we obtain a stable system for spin about the intermediate axes.

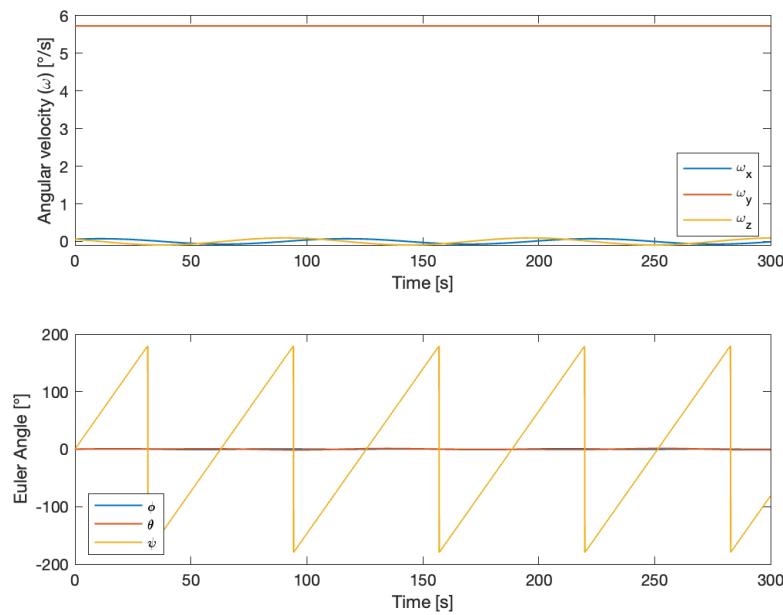


Figure 43:

e. Try to make rotation about another arbitrary axis (potentially relevant to your project) stable through a generic momentum wheel or rotor.

We choose to stabilize spin about the body x-axis, which is important for pointing our satellite and appropriately sweeping the target with our SAR. We use the rotation previously found between the principal and body axes to achieve this, applying the rotation to the angular velocity and the angular momentum vector direction of the momentum wheel.

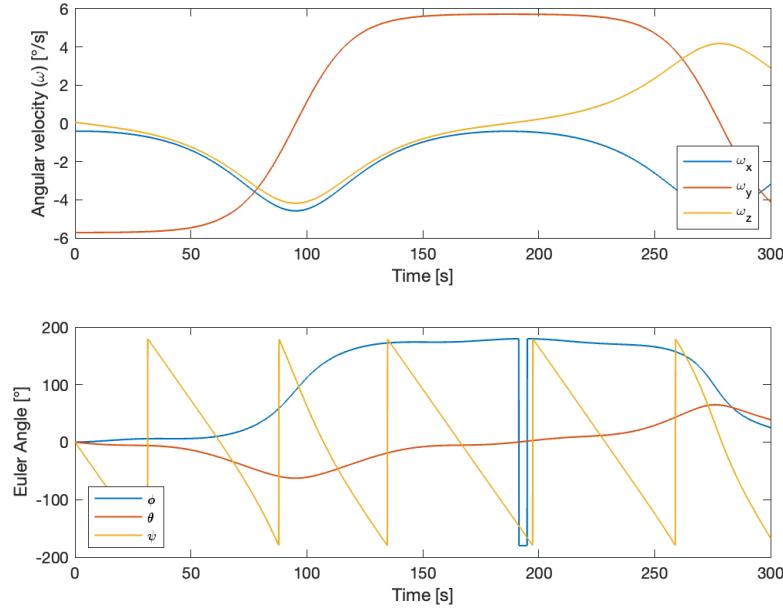


Figure 44: Initially unstable attitude with low momentum wheel angular velocity

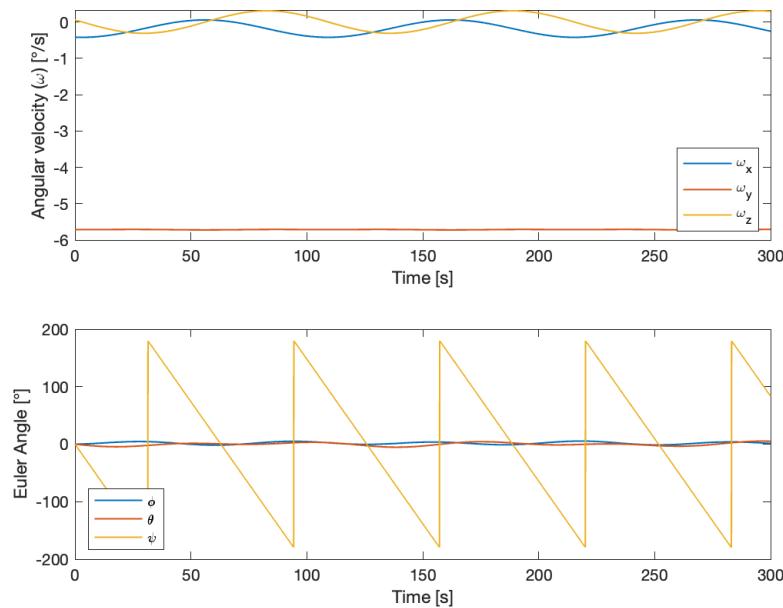


Figure 45: Periodically stable attitude after increasing momentum wheel angular velocity 10x

## 4.4 PROBLEM 4

*Gravity gradient torque (modeling)*

a. Remove rotor.

A new function was created without effect of the rotor.

b. Program gravity gradient torque. Feed torque to Euler equations. This is the first perturbation you model resulting from the interaction of the spacecraft with the environment. Hint: change your orbit to make gravity gradient significant if that's not the case.

The equations for the gravity gradient torque are below.

$$\begin{aligned} I_x \dot{\omega}_x + (I_z - I_y) \omega_y \omega_z &= 3n^2 (I_z - I_y) c_y c_z \\ I_y \dot{\omega}_y + (I_x - I_z) \omega_z \omega_x &= 3n^2 (I_x - I_z) c_z c_x \\ I_z \dot{\omega}_z + (I_y - I_x) \omega_x \omega_y &= 3n^2 (I_y - I_x) c_x c_y \end{aligned}$$

In the above equation,  $\vec{c} = [c_x, c_y, c_z]^\top$  is the normalized direction of  $\vec{R}$ .

The function `gravGrad` was developed to be used with `ode113` to propagate the Euler equations and kinematics with gravity gradient torques.

```

1 function [stateDot] = gravGrad(t, state, Ix, Iy, Iz, n)
2     % Orbit position and velocity
3     r = state(1:3);
4     v = state(4:6);
5
6     % Angular velocity
7     w = state(7:9);
8
9     % Euler angles
10    phi = state(10);
11    theta = state(11);
12
13    stateDot = zeros(12,1);
14    stateDot(1:3) = v;
15    stateDot(4:6) = (-3.986 * 10^5 / norm(r)^2) * r / norm(r); % km/s^2
16
17    radial = r / norm(r);
18    A_ECI2P = e2A(state(10:12));
19    c = A_ECI2P * radial;
20    M = gravGradTorque(Ix, Iy, Iz, n, c);
21    stateDot(7) = (M(1) - (Iz - Iy) * w(2) * w(3)) / Ix;
22    stateDot(8) = (M(2) - (Ix - Iz) * w(3) * w(1)) / Iy;
23    stateDot(9) = (M(3) - (Iy - Ix) * w(1) * w(2)) / Iz;
24
25     % Euler angle time derivatives
26     % 312
27     EPrimeInv = [sin(phi)*sin(theta) cos(phi)*sin(theta) cos(theta); ...
28                   cos(theta)*cos(phi) -sin(phi)*cos(theta) 0; ...
29                   sin(phi) cos(phi) 0] * (1 / cos(theta));

```

```

30      % 313
31      % EPrimeInv = [-sin(phi)*cos(theta) -cos(phi)*cos(theta) ...
32          sin(theta); ...
33          cos(phi)*sin(theta) -sin(phi)*sin(theta) 0; ...
34          sin(phi) cos(phi) 0] * (1 / sin(theta));
35 stateDot(10:12) = EPrimeInv * w;
36 end

```

```

1 function M = gravGradTorque(Ix,Iy,Iz,n,c)
2     cx = c(1);
3     cy = c(2);
4     cz = c(3);
5     M(1) = 3 * n^2 * (Iz - Iy) * cy * cz;
6     M(2) = 3 * n^2 * (Ix - Iz) * cz * cx;
7     M(3) = 3 * n^2 * (Iy - Ix) * cx * cy;
8 end

```

c. Verify that the magnitude of the modeled torque is consistent with the orbit and inertia tensor of your satellite. Hint: use simplified formulas from class on modeling of gravity gradient torque.

We can estimate the order of magnitude for gravity gradient torques using the following equation.

$$\vec{M} = \frac{3\mu}{a^3} \begin{bmatrix} (I_z - I_y)c_y c_z \\ (I_x - I_z)c_z c_x \\ (I_y - I_x)c_x c_y \end{bmatrix}$$

The parameters used were the known moments of inertia ( $I_x = 7707$ ,  $I_y = 14563$ ,  $I_z = 18050 \text{ kg} \cdot \text{m}^2$ ), known values for Earth ( $a = 7125.49 \text{ km}$ ,  $\mu = 398600 \text{ km}^3/\text{s}^2$ ), and an arbitrary  $\vec{c} = [\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}]$ . With these values, we arrive at:

$$\vec{M} = \begin{bmatrix} 0.384174 \\ 1.13956 \\ -0.755387 \end{bmatrix} \cdot 10^{-2} \text{ N m}$$

These values are in line with what we see in Figure 50.

*d. Numerically integrate Euler and Kinematic equations including gravity gradient from initial conditions corresponding to body axes aligned with the orbital frame (RTN). Verify that gravity gradient torque is zero, besides numerical errors. Hint: you may need to simplify the orbit to unperturbed circular to achieve this. Check that initial angular velocity matches mean motion.*

Figures 46, 47 demonstrate zero torque when aligned with RTN frame and constant angular velocity. Simulating for longer periods reveals the torque error is periodically stable.

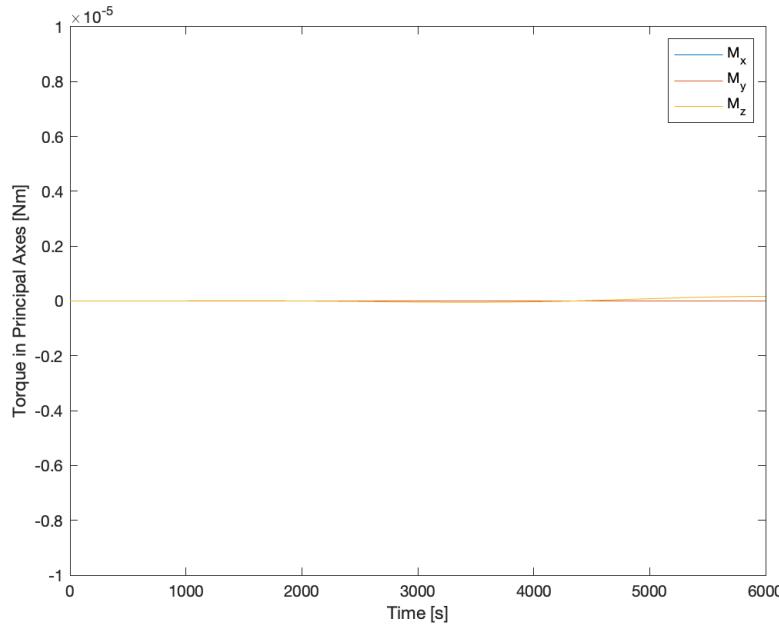


Figure 46: Zero gravity gradient torque for satellite aligned with RTN frame

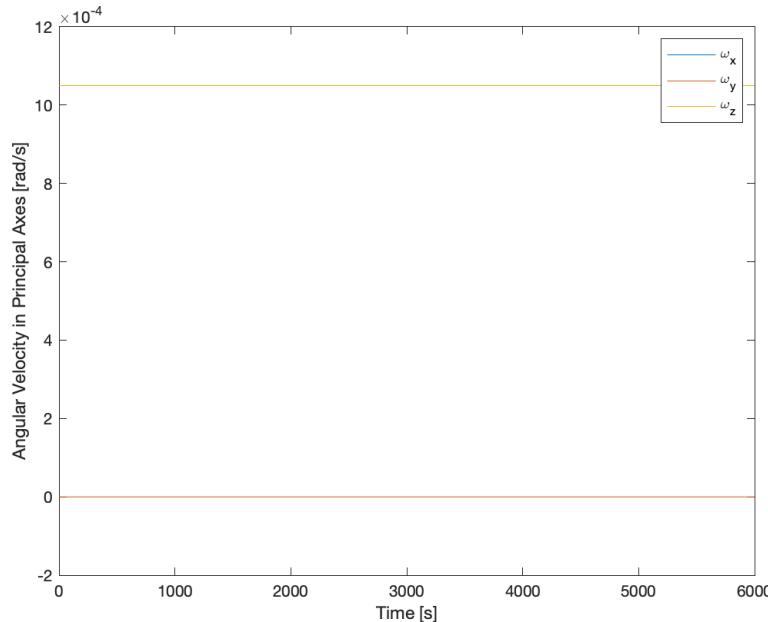


Figure 47: Angular velocity parallel to RTN normal equal to mean motion, all others zero

e. Numerically integrate Euler and Kinematic equations including gravity gradient from arbitrary initial conditions (e.g., relevant to your project). Plot external torque (3 components w.r.t. time) and resulting attitude motion (depends on attitude parameterization, add Euler angles for better geometrical interpretation) over multiple orbits. Comment on results.

We choose initial conditions by aligning our body axes to RTN, which will more closely resemble the orientation of the satellite when collecting science data.

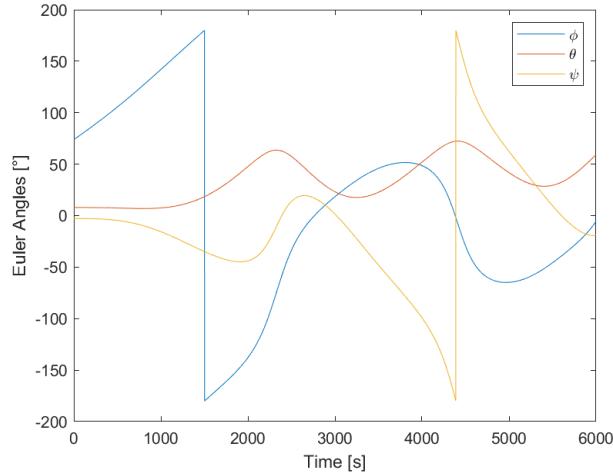


Figure 48: Euler angles with gravity gradient for satellite with arbitrary initial conditions

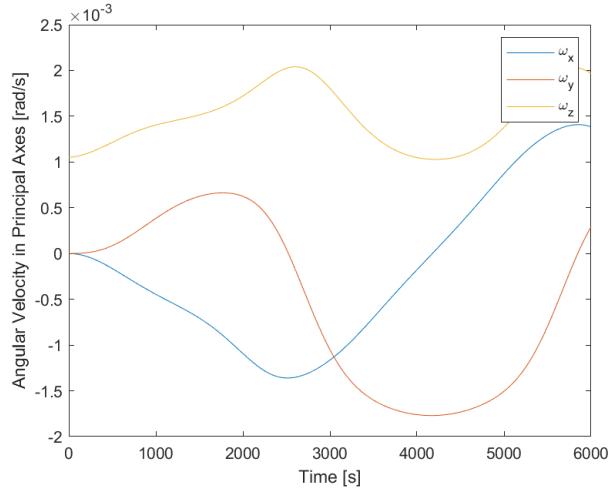


Figure 49: Angular velocity with gravity gradient for satellite with arbitrary initial conditions

The figures show that there is a noticeable effect of gravity gradient torque on the satellite, although the magnitude of the torque is low. This causes changes to our Euler angles throughout the orbit, meaning we will need a control system to stabilize and point our satellite in order to meet mission requirements.

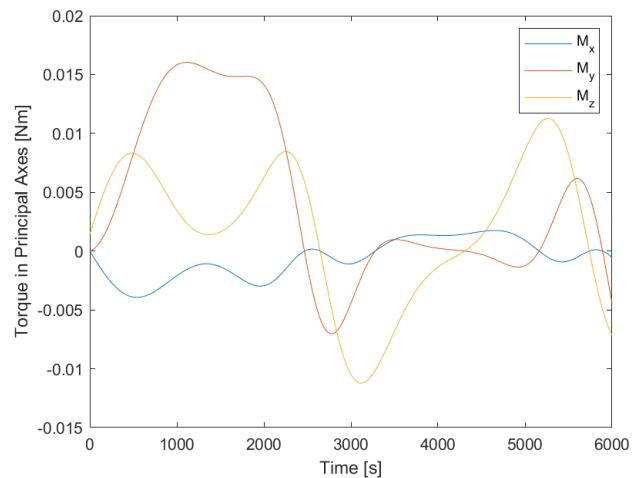


Figure 50: Gravity gradient torques for satellite with arbitrary initial conditions

## 5 PROBLEM SET 5

### 5.1 PROBLEM 1

*Gravity gradient torque (stability)*

- a. Calculate the coefficients  $K_i$  of the moments of inertia which drive stability under gravity gradient. Compute and plot regions of stable and unstable motion similar to the picture below:

The following equations show the relationships for gravity gradient stability in terms of moments of inertia. The first inequality hold for when the pitch is stable, and the last two inequalities hold for when the roll and yaw are stable.

$$k_N = \frac{I_T - I_R}{I_N}, \quad k_T = \frac{I_N - I_R}{I_T}, \quad k_R = \frac{I_N - I_T}{I_R}$$

$$k_T > k_R, \quad k_R k_T > 0, \quad 1 + 3k_T + k_R k_T > 4\sqrt{k_R k_T}$$

We show the plot of stable and unstable motion under gravity gradient. When computing the coefficients using moments of inertia about the principal axes, we obtain the following plot. In this case, we investigate the stability for the satellite when it is rotated such that the spacecraft is oriented appropriately for SAR operations, that is, principal x-axis is anti-radial, principal y-axis is opposite of cross-track, and principal z-axis is opposite of along-track. However, this orientation is unstable, as shown below.

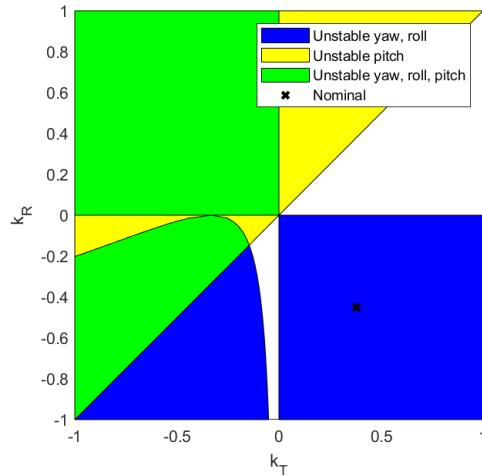


Figure 51: Stability for nominal axes

- b. Considering the results from 1a, comments on the expected stability of the attitude motion of your satellite about equilibrium. Try to reproduce stable and unstable motion by setting proper initial conditions and perturbing those conditions slightly (e.g., by 1%). Plot attitude parameters (e.g., Euler angles) to show stability or instability.

First, without perturbations, the satellite can maintain steady attitude, demonstrating that this chosen orientation is indeed an equilibrium.

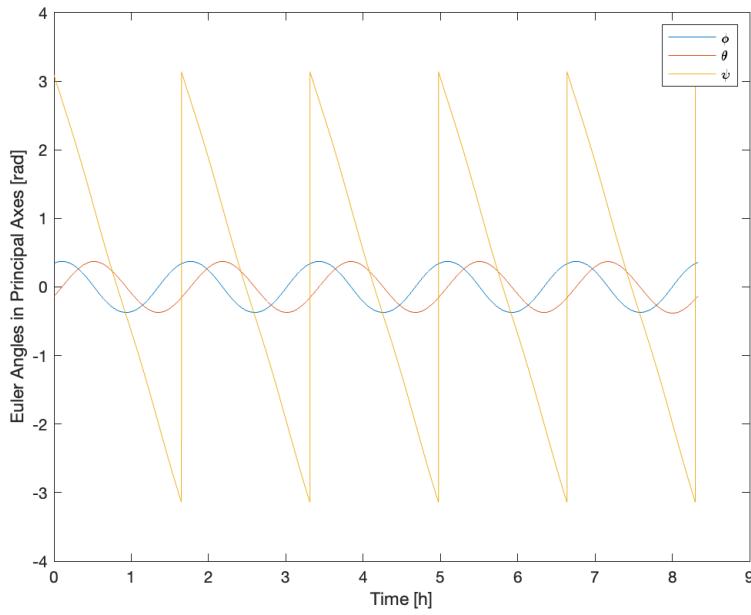


Figure 52: Attitude evolution for unstable orientation without perturbations

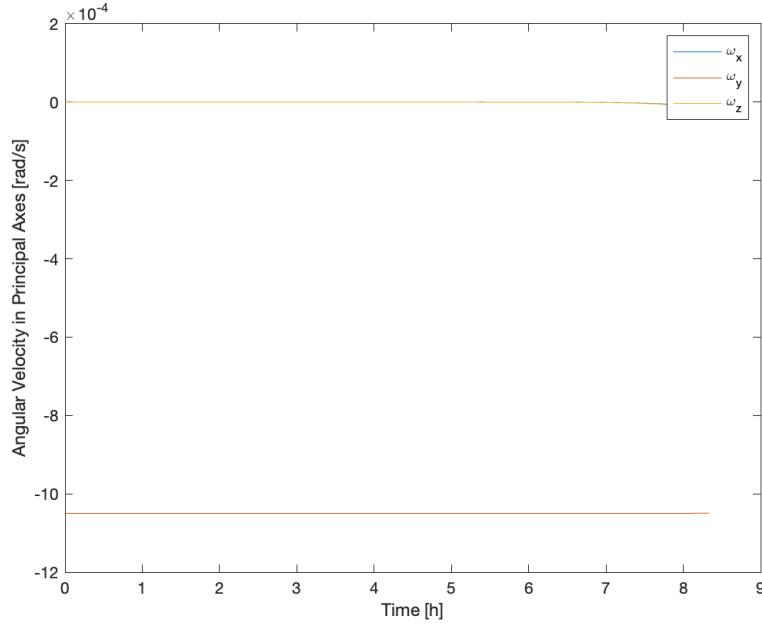


Figure 53: Angular velocity evolution for unstable orientation without perturbations

We expect unstable behavior for small perturbations. Previously, we have already shown that aligning principal axes with RTN produces stable behavior when there are no perturbations. Now, we will introduce small perturbations, causing the system to leave equilibrium and demonstrating that it is unstable.

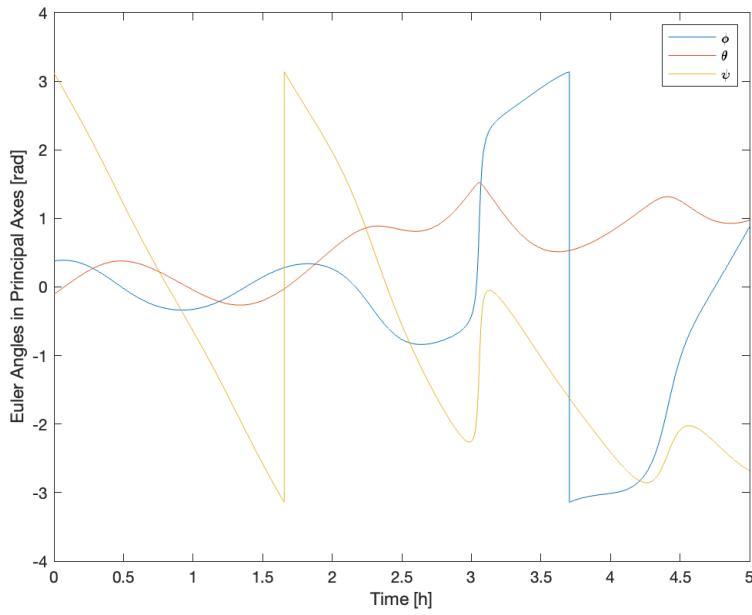


Figure 54: Attitude evolution for unstable orientation with 1% perturbations

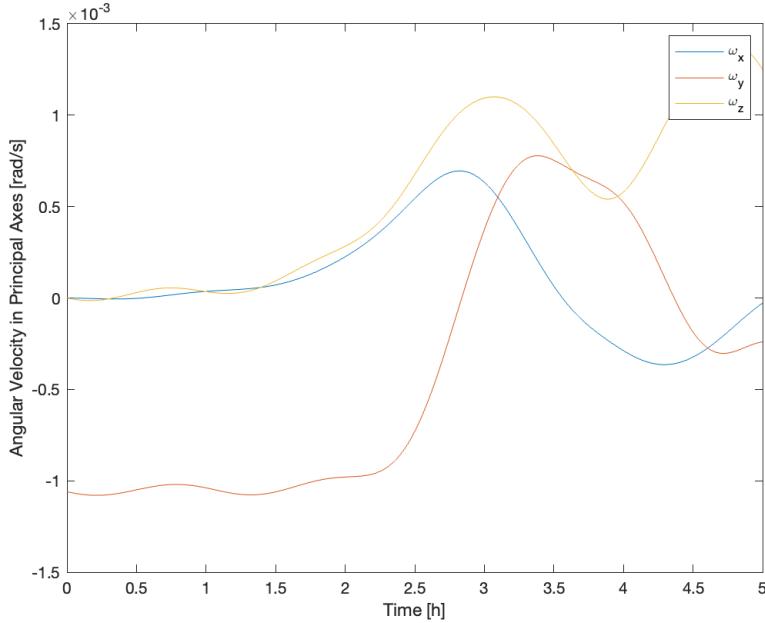


Figure 55: Angular velocity evolution for unstable orientation with 1% perturbations

c. How would you need to change the mass distribution and/or nominal attitude of your satellite to obtain stable motion from the gravity gradient torque? Would it make sense for your project? Show a couple of potential configurations in the  $Ki$  plane and resulting stability of attitude motion at the equilibrium. This is done by changing your inertia tensor and simulating numerically.

We will have to align the principal axes XYZ with RTN (respectively) to obtain a stable attitude motion.

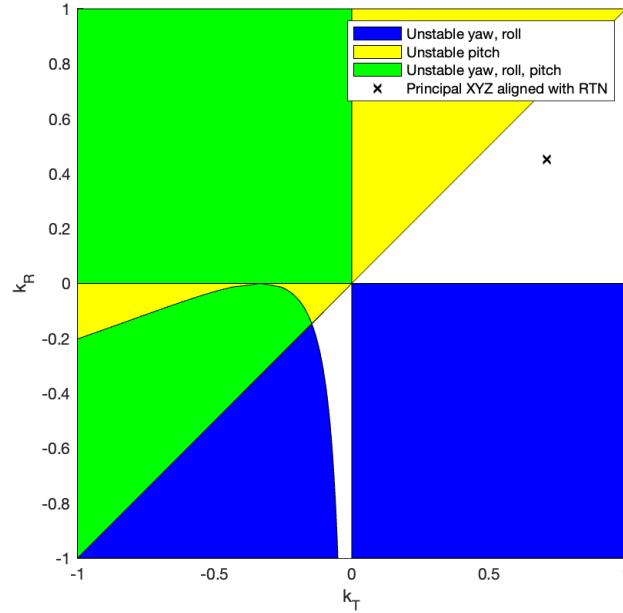


Figure 56: Stability for aligned principal axes

We expect stable behavior for small perturbations. Previously, we have already shown that aligning principal axes with RTN produces stable behavior when there are no perturbations. Now, we will introduce small perturbations.

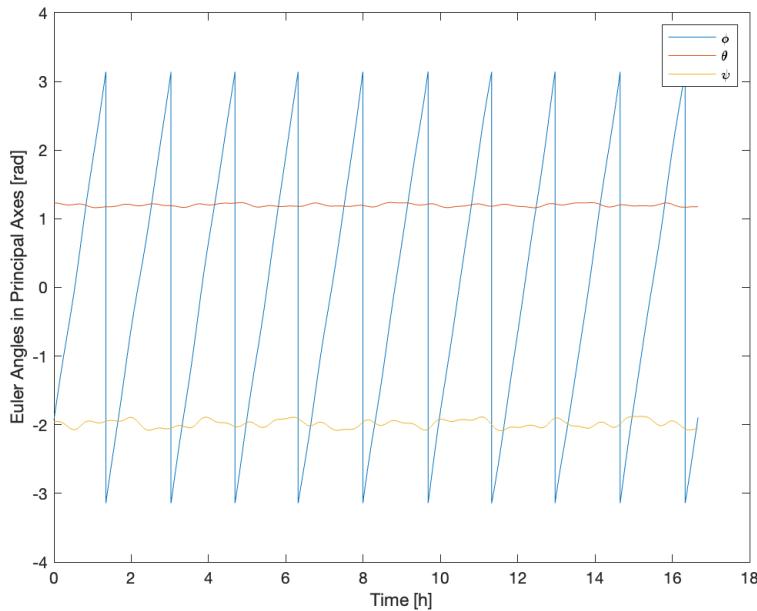


Figure 57: Attitude evolution for stable orientation with 1% perturbations

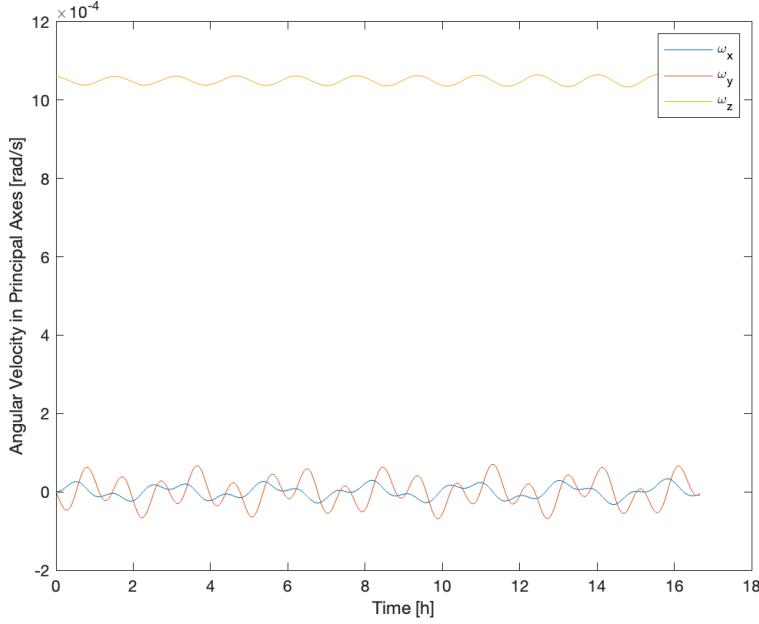


Figure 58: Angular velocity evolution for stable orientation with 1% perturbations

As expected, our attitude motion (angular velocities and Euler angles) are periodically stable with a small perturbation in initial condition.

We cannot maintain this orientation if we want to properly point the radar antenna located at the top of the spacecraft, so it does not make sense to orient the satellite along principal axes for gravity gradient stability. Instead, we will likely require magnetorquers to offset angular momentum changes from environmental torques, including gravity gradient torques due to our unstable equilibrium.

## 5.2 PROBLEM 2

*In addition to gravity gradient, start programming perturbation torques due to magnetic field, solar radiation pressure, and atmospheric drag. Note 1: You should apply a very minimal/basic model for perturbations that are not relevant (negligible) to your project. It is expected that you do not ignore them. Note 2: All perturbations can be grouped into a single large subsystem called environment or similar whose output feed the Euler equations. Note 3: Re-use as many functions as possible for solar radiation pressure and atmospheric drag.*

In addition to gravity gradient torques, we modeled torque from magnetic field interactions, solar radiation pressure, and atmospheric drag.

The equation for the torque from the magnetic field interactions is shown below.

$$\vec{M}_m = \vec{m}_{sat} \times \vec{B}_{Earth}$$

Since the satellite is operates in LEO, the spherical harmonic model up to  $n = 4$  was used for maximum accuracy. This model is based on the geocentric distance ( $R$ ), colatitude ( $\theta$ ), and

longitude ( $\phi$ ). The model requires Gaussian coefficients ( $g^{n,m}$ ,  $h^{n,m}$ ) and Legendre functions ( $P^{n,m}$ ) and their derivatives ( $\frac{\delta P^{n,m}(\theta)}{\delta \theta}$ ), which are explained in more detail in Wertz [12].

$$\begin{aligned} B_R &= \sum_{n=1}^4 \left( \frac{R_{Earth}}{R} \right)^{n+2} (n+1) \sum_{m=0}^n (g^{n,m} \cos m\phi + h^{n,m} \sin m\phi) P^{n,m}(\theta) \\ B_\theta &= - \sum_{n=1}^4 \left( \frac{R_{Earth}}{R} \right)^{n+2} \sum_{m=0}^n (g^{n,m} \cos m\phi + h^{n,m} \sin m\phi) \frac{\delta P^{n,m}(\theta)}{\delta \theta} \\ B_\phi &= - \frac{1}{\sin \theta} \sum_{n=1}^4 \left( \frac{R_{Earth}}{R} \right)^{n+2} \sum_{m=0}^n m (-g^{n,m} \sin m\phi + h^{n,m} \cos m\phi) P^{n,m}(\theta) \end{aligned}$$

The equations below define the solar radiation torque, where  $C_S$  is the specular reflection coefficient,  $C_d$  is the diffuse reflection coefficient,  $\vec{S}$  is the vector facing the Sun, and  $e_i$  is 1 if the surface is illuminated and 0 otherwise. In implementation, we represent  $e_i$  as a boolean tensor in tensor operations for fast computation.

$$\begin{aligned} \vec{M}_S &= \sum_{i=1}^n \vec{r}_i \times e_i \int_{S_i} d\vec{f}_{total_i} \\ d\vec{f}_{total} &= -P((1 - C_S)\hat{\vec{S}} + 2(C_S \cos \theta + \frac{1}{3}C_d) \cos \theta dA \end{aligned}$$

Finally, the equation below define the aerodynamic torque. Velocity is relative velocity of the spacecraft to the atmosphere, and we can compute the atmosphere's relative motion using a cross product of the position vector and Earth's rotational rate in ECI.

$$d\vec{f}_{aero} = -\frac{1}{2} C_D \rho V^2 (\hat{\vec{V}} \cdot \hat{\vec{N}}) \hat{\vec{V}} dA$$

The following function enables the propagation of the orbit with the specified perturbation torques.

```

1 function [stateDot] = orbitTorque(t,state,Ix,Iy,Iz, ...
2     CD,Cd,Cs,P,m,UT1, ...
3     barycenter,normal,area,cm,n)
4 warning('off','aero:atmosnrlmsise00:setf107af107aph')
5
6 % Orbit position and velocity
7 r = state(1:3);
8 v = state(4:6);
9 rEarth = state(13:15);
10 vEarth = state(16:18);
11
12 % Angular velocity
13 w = state(7:9);
14
```

```

15    % Euler angles
16    phi = state(10);
17    theta = state(11);
18
19    % Gravity gradient torque
20    radial = r / norm(r);
21    A_ECI2P = e2A(state(10:12));
22    c = A_ECI2P * radial;
23    Mgg = gravGradTorque(Ix,Iy,Iz,n,c);
24
25    % Drag torque
26    % Hard-coded with Earth radius for now
27    [~,density] = atmosnrlmsise00(1000 * (norm(r) - ...
28        6378.1),0,0,2000,1,0);
29    rho = density(6);
30    vPrincipal = A_ECI2P * (v + cross([0; 0; 7.2921159E-5],r));
31    [~,Md] = drag(vPrincipal,rho,CD,barycenter,normal,area,cm);
32
33    % Solar radiation pressure torque
34    % Hard-coded with Earth axial tilt for now
35    Rx = [1 0 0; 0 cosd(23.5) -sind(23.5); 0 sind(23.5) cosd(23.5)];
36    s = A_ECI2P * (-Rx * rEarth - r); % SCI -> ECI -> XYZ
37    [~,Msrp] = srp(s,P,Cd,Cs,barycenter,normal,area,cm);
38
39    % Magnetic field torque
40    % Hard-coded with Earth radius for now
41    Mm = magFieldTorque(m,r,state(10:12),t,6378.1,UT1);
42
43    % Compute net moments
44    Mx = Mgg(1) + Md(1) + Msrp(1) + Mm(1);
45    My = Mgg(2) + Md(2) + Msrp(2) + Mm(2);
46    Mz = Mgg(3) + Md(3) + Msrp(3) + Mm(3);
47    % Mx = 0; My = 0; Mz = 0; % Use this to test without disturbances
48
49    % Time derivatives
50    stateDot = zeros(12,1);
51    stateDot(1:3) = v;
52    stateDot(4:6) = (-3.986e5 / norm(r)^2) * r / norm(r); % km/s^2
53    stateDot(7) = (Mx - (Ix - Iy) * w(2) * w(3)) / Ix;
54    stateDot(8) = (My - (Ix - Iz) * w(3) * w(1)) / Iy;
55    stateDot(9) = (Mz - (Iy - Ix) * w(1) * w(2)) / Iz;
56
57    % 312 Euler angle time derivatives
58    EPrimeInv = [sin(phi)*sin(theta) cos(phi)*sin(theta) cos(theta); ...
59                  cos(theta)*cos(phi) -sin(phi)*cos(theta) 0; ...
60                  sin(phi) cos(phi) 0] * (1 / cos(theta));
61    stateDot(10:12) = EPrimeInv * w;
62
63    % Sun position
64    stateDot(13:15) = vEarth;
65    stateDot(16:18) = (-1.327E11 / norm(rEarth)^2) * ...
66        rEarth / norm(rEarth); % km/s^2
67 end

```

```

1 function [F,M] = drag(v,rho,CD,barycenter,normal,area,cm)
2     % Compute drag in principal axes

```

```

3      % RE = 6378.1 % km
4      u = v / norm(v);
5      N = normal;
6      Aeff = ((u' * N) > 0) .* area;
7      rC = barycenter - cm;
8      D = -0.5 * CD * rho * norm(v)^2 * (u' * (N .* Aeff)) .* u;
9      F = sum(D,2);
10     M = sum(cross(rC,D),2);
11
12     % Slow loop function (obsolete)
13     % u = v / norm(v);
14     % F = zeros([3 1]);
15     % M = zeros([3 1]);
16     % for i = length(area)
17     %     n = normal(:,i);
18     %     if dot(u,n) < 0
19     %         rC = (barycenter(:,i) - cm);
20     %         A = area(1,i);
21     %         D = -0.5 * CD * rho * norm(v)^2 * dot(u,n) * u * A;
22     %         M = M + cross(rC,D);
23     %         F = F + D;
24     %     end
25     % end
26 end

```

```

1 function [F,M] = srp(s,P,Cd,Cs,barycenter,normal,area,cm)
2     % Compute solar radiation pressure in principal axes
3     u = s / norm(s);
4     N = normal;
5     Aeff = ((u' * N) > 0) .* area;
6     rC = barycenter - cm;
7     theta = acos((u' * N) ./ (norm(u) * vecnorm(N)));
8     SRP = -P * cos(theta) .* Aeff .* ...
9         ((1 - Cs) * u + 2 * (Cs * cos(theta) + Cd / 3) .* N);
10    F = sum(SRP,2);
11    M = sum(cross(rC,SRP),2);
12
13    % Slow loop function (obsolete)
14    % F = zeros([3 1]);
15    % M = zeros([3 1]);
16    % for i = length(area)
17    %     n = normal(:,i);
18    %     if dot(u,n) > 0
19    %         rC = barycenter(:,i) - cm;
20    %         theta = acos(dot(u,n) / (norm(u) * norm(n)));
21    %         A = area(1,i);
22    %         SRP = -P * cos(theta) * A * ...
23    %             ((1 - Cs) * u + 2 * (Cs * cos(theta) + Cd / 3) * n);
24    %         M = M + cross(rC,SRP);
25    %         F = F + SRP;
26    %     end
27    % end
28 end

```

```
1 function [M, B_ECEF, F, XYZ] = magFieldTorque(m,R,eulerAngle,t,RE,UT1)
```

```

2      % Calculated the expected torque due to Earth's magnetic field
3      % Inputs:
4      % - m: magnetic moment of satellite [N * m / T]
5      % - R: position vector of satellite [km]
6      % - t: time of simulation [s]
7      % - RE: radius of Earth [km] (6378 km)
8      % - UT1: start time of simulation
9
10     % Find lambda (longitude) and theta (colatitude)
11     GMST = time2GMST(t,UT12MJD(UT1));
12     [lat,lon,~] = ECEF2Geoc(ECI2ECEF(R,GMST),t);
13     lambda = lat;
14     phi = lon;
15     theta = pi/2 - phi;
16
17     [B_R,B_theta,B_phi] = magFieldEarth(R,lambda,theta,RE);
18     % B_latlon = [B_R, B_theta, B_phi];
19
20     timeVec = UT1;
21     timeVec(2) = timeVec(2) + t/86400;
22     [XYZ,H,D,I,F] = wrldmagn(norm(R)-RE,lat,lon,decyear(timeVec));
23
24     delta = phi;
25     alpha = lambda + GMST;
26
27     B_x = (B_R * cos(delta) + B_theta * sin(delta)) * ...
28         cos(alpha) - B_phi * sin(alpha);
29     B_y = (B_R * cos(delta) + B_theta * sin(delta)) * ...
30         sin(alpha) + B_phi * cos(alpha);
31     B_z = (B_R * sin(delta) - B_theta * cos(delta));
32
33     B_ECI = [B_x;B_y;B_z];
34     B_ECEF = ECI2ECEF(B_ECI, GMST);
35     B = e2A(eulerAngle) * B_ECI;
36
37     M = cross(m,B);
38 end

```

### 5.3 PROBLEM 3

*Include all torques you have been able to model in numerical integration. Please show comparison of numerically computed disturbance torques with expected values and trend from theory (model) and tables (Wertz) referenced in class. Plot all torque components in principal axes over time. Plot the resultant (sum) of all torques in principal axes. Make sure that your model is not too ideal, i.e. make sure that center of pressure and center of mass do not coincide.*

For most torques, the worst-case estimated magnitudes were found by using existing properties of the spacecraft model. We use simplified equations (compared to those in the previous section) to estimate these, making assumptions for worst-case torques. However, the magnetic torque calculation assumes a model of the spacecraft's magnetic field as a single coil wrapped around the surface of the RIS and satellite bus. The estimated maximum values of each torque are listed in the table below.

Magnetic Field	Estimated Maximum Value (N-m)
$M_{gg}$	1.7093e-02
$M_{srp}$	1.8294e-02
$M_{drag}$	1.3682e-03
$M_{mag}$	1.3751e-10

In Figures 59, 60, 61, 62, we show the numerical simulation results for each type of disturbance based on the equations and code in the previous section. The numerical simulations indicate that the simulated results and estimated values are roughly within an order of magnitude for each type of disturbance torque shown.

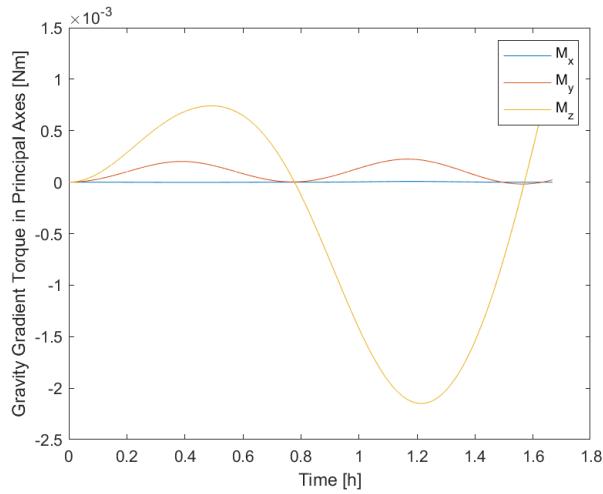


Figure 59: Numerical simulation of gravity gradient torques

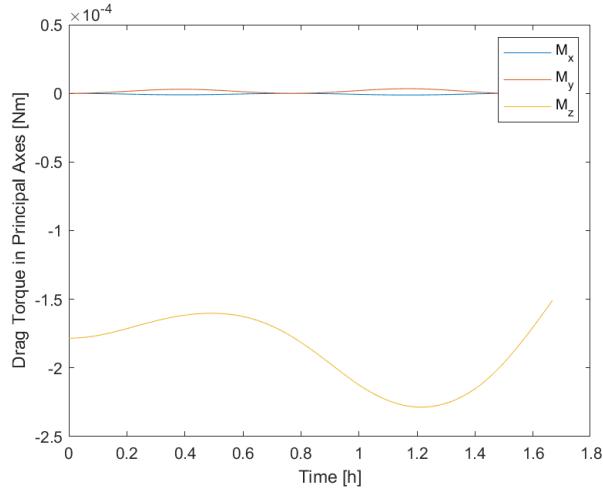


Figure 60: Numerical simulation of drag torques

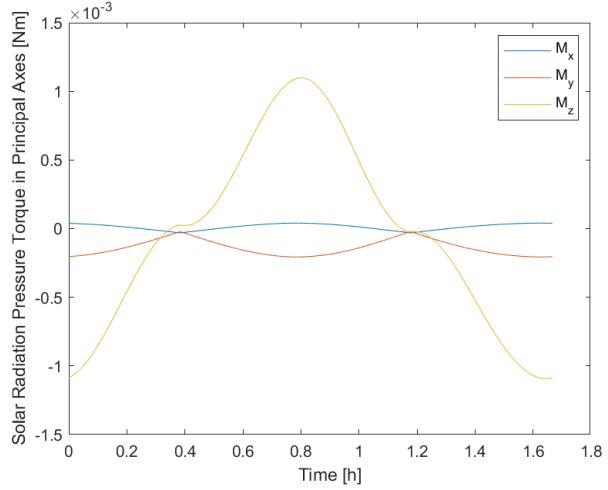


Figure 61: Numerical simulation of solar radiation pressure torques

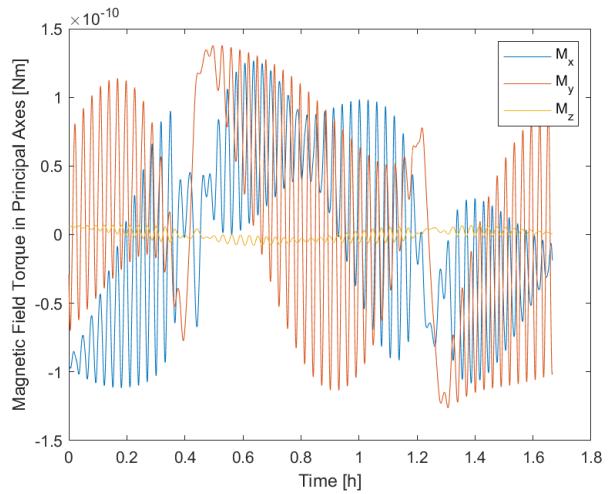


Figure 62: Numerical simulation of magnetic field torques

## 6 PROBLEM SET 6

### 6.1 PROBLEM 1

*Complete the modeling and verification of perturbation torques as described in the previous pset*

We have implemented models of the drag, gravity gradient, magnetic field, and solar radiation pressure effects on our satellite. Our magnetic field model uses a fourth-order representation suitable for LEO, and we also propagate the motion of Earth around the sun for accurate determination of solar radiation pressure. Please see the relevant portions of the previous section for more detail.

### 6.2 PROBLEM 2

*Compute the attitude control error even if a controller is not implemented yet. The attitude control error represents the rotation between the desired and actual attitude. Plot the attitude control error and give its interpretation. Note that this step requires the definition and computation of the desired or nominal or target attitude of the spacecraft. In general, this can be expressed in body or principal axes.*

In operation, NISAR will point its antenna towards Earth such that the principal x-axis facing the negative radial direction and the principal y-axis is pointing in the negative normal direction. The angular velocity about the principal y-axis is the negative of mean motion.

$$\vec{P}_{desired} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & -1 & 0 \end{bmatrix} \vec{P}_{RTN}$$

We first simulate attitude without disturbance torques, calculating error as the rotation between target and actual attitude. Figures 63, 64, 65 demonstrate that the error is approximately zero.

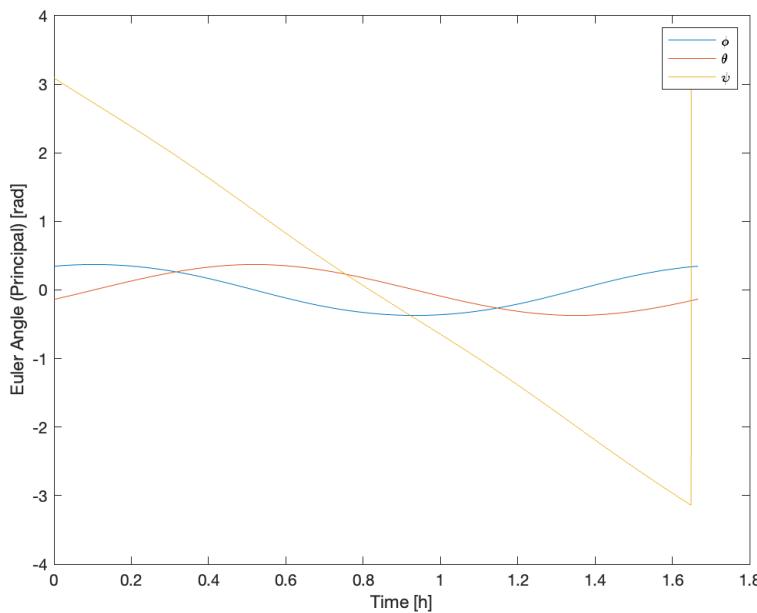


Figure 63: Actual attitude relative to inertial frame, without perturbations

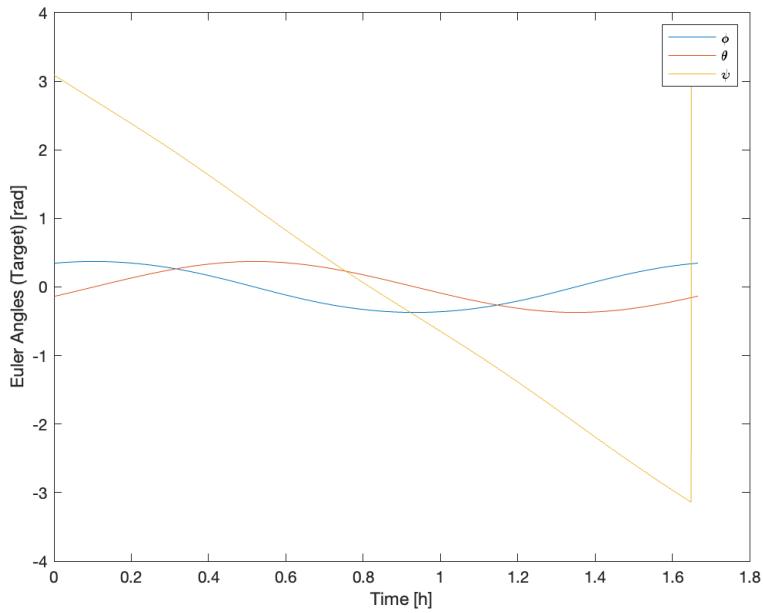


Figure 64: Target attitude relative to inertial frame, without perturbations

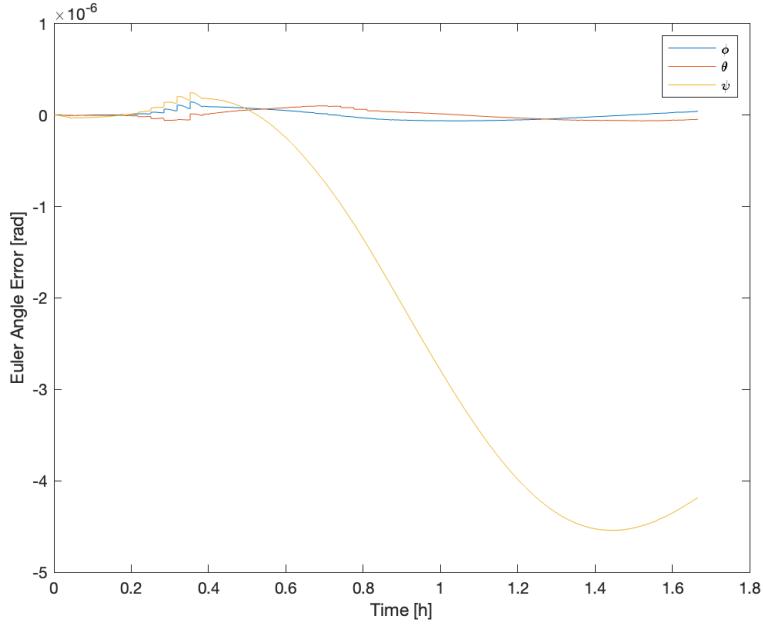


Figure 65: Error between target and actual attitude, without perturbations

For computing errors, we multiply the DCM representing the rotation from the inertial frame to our actual attitude with the inverse of the DCM representing the rotation from the inertial frame to the target attitude. We observe that there is a very small error (approximately  $10^{-7}$  to  $10^{-6}$ ) in our result in Figure 65. This is most likely arising from our use of numerical integration.

### 6.3 PROBLEM 3

Note that the attitude control error represents a rotation matrix (DCM) which quantifies how far the actual attitude is from the true attitude. You can use any parameterization to plot the attitude control errors corresponding to this DCM. Give interpretation of the attitude control errors given the applied disturbances.

With perturbations, we observe a small but noticeable change to actual attitude in Figure 66.

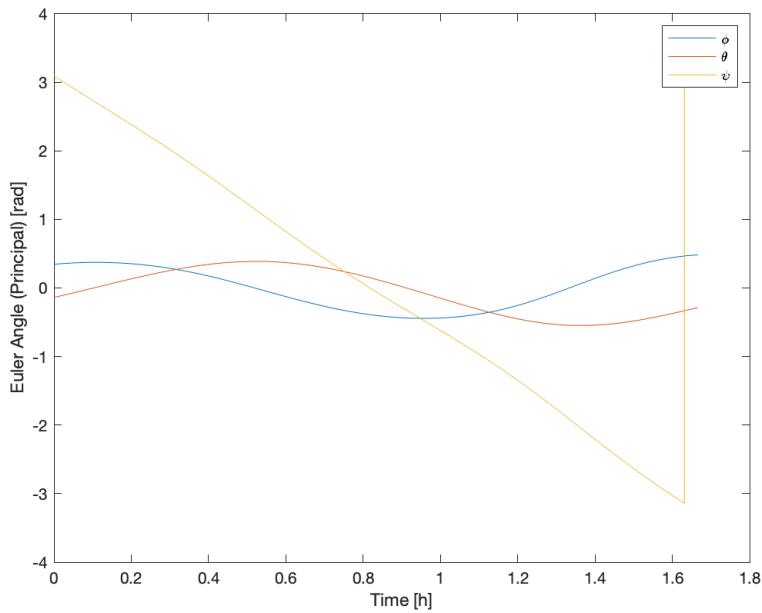


Figure 66: Actual attitude relative to inertial frame, with perturbations

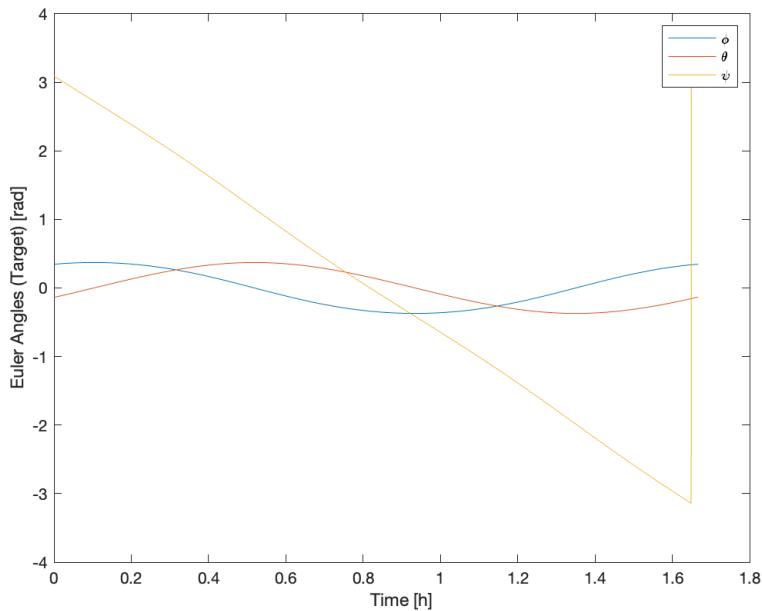


Figure 67: Target attitude relative to inertial frame, with perturbations

A much larger error appears in Figure 68. When compared with the disturbance-free error in Figure 65 (which is approximately zero), we can interpret the attitude control error as the amount which the disturbance torques rotate our satellite during the simulation period, which in this case is one orbital period.

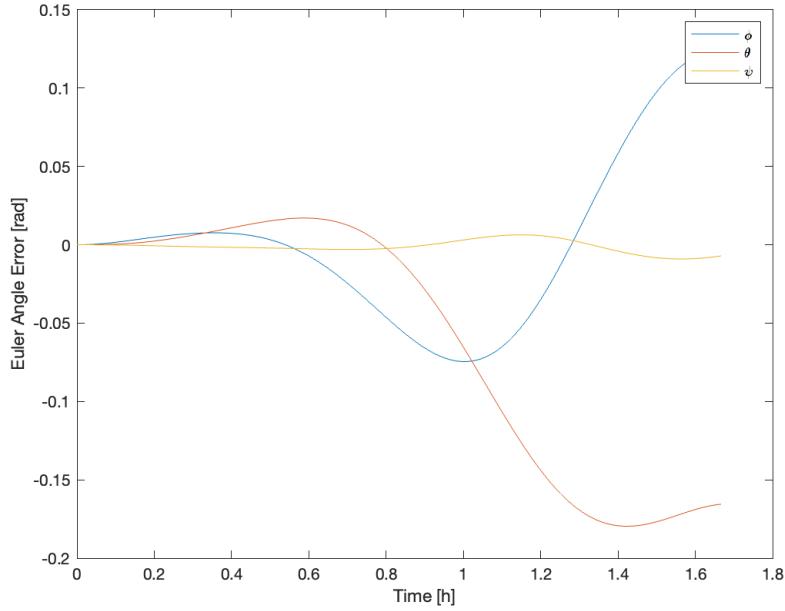


Figure 68: Error between target and actual attitude, with perturbations

#### 6.4 PROBLEM 4

You can now start modeling the Simulink spacecraft subsystem which is what the satellite believes is happening (on-board). Initially, the sensors provide ideal measurements (no bias or noise). Just use an empty box for those. The outputs of the sensors are measurements which are used for attitude determination. These measurements are computed from the reference truth or oracle.

We create a Simulink model using the same functions and variables previously modeled in MATLAB and shown in this paper, shown in Figure 69. Notably, we have a simulation subsystem which is responsible for simulating the dynamics, kinematics, and disturbances of the system, which we have modeled in previous sections. We now add an attitude determination subsystem for this step. Since we choose to model noise-free measurements for now, we directly generate measurements from the true state without injecting noise.

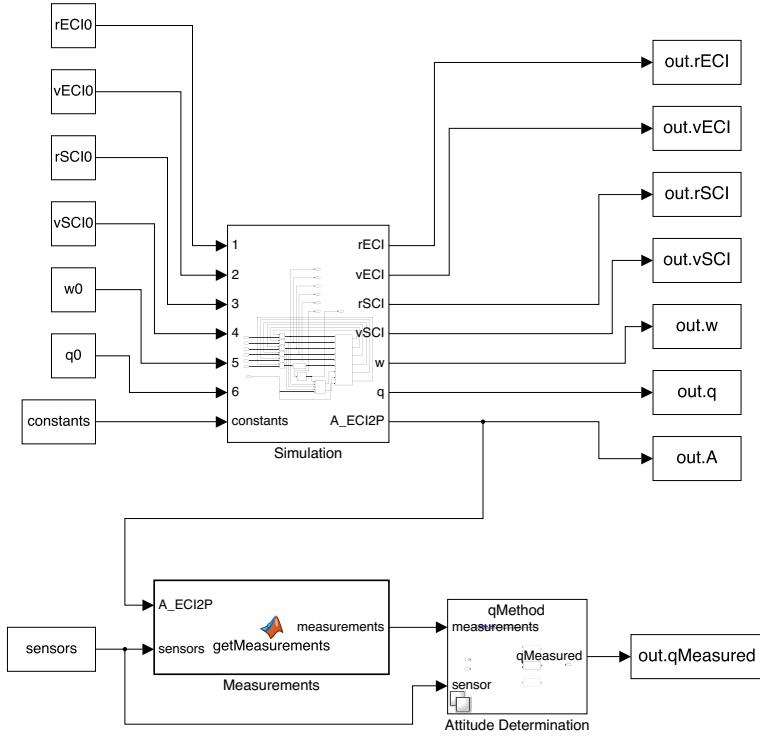


Figure 69: Simulink model for NISAR spacecraft

## 6.5 PROBLEM 5

*Assume a certain set of sensors. In general, a number of unit vectors and angular velocities can be considered as measurements*

We initially model our attitude determination subsystem using “dummy” sensors and measurements. Each of the algorithms is fed randomly generated unit vectors as a noise-free measurement. These unit vectors represent fixed directions in the celestial sphere such that an orientation can be computed when a sufficient number of vectors is available. This represents the functionality of a star tracker sensor, which is able to determine the orientation of the satellite’s axes relative to an inertial reference using unit vector directions of known stars. We do not yet model the inner workings of a star tracker, such as the identification and computation of star positions.

In our test implementation, we generate five such random unit vectors. These unit vectors are both our ground truth (analogous to known directions from a star catalog) and our measurement signal, since we do not inject any noise at this step. Thus, we expect to obtain an exact solution from each attitude determination algorithm in the absence of such noise.

Additionally, we can compute the attitude by integration of our angular velocity, equivalent to using a rate gyro sensor that measures angular rate. We can simply use the same kinematics equations used to propagate our attitude to model the evolution of our attitude for the rate gyro. In simulation and without noise, we also expect an exact solution from this technique.

*Implement the deterministic attitude determination algorithm discussed in class and its variant which uses fictitious measurements to spread the errors across the measurements*

For measurements  $\vec{M}$  and ground truth  $\vec{V}$ , we can use deterministic attitude determination to find the attitude by taking the pseudoinverse.

$$\vec{M} = \begin{bmatrix} \vdots & \vdots & \vdots \\ \vec{m}_1 & \vec{m}_2 & \vec{m}_3 \\ \vdots & \vdots & \vdots \end{bmatrix} = \vec{A} \begin{bmatrix} \vdots & \vdots & \vdots \\ \vec{v}_1 & \vec{v}_2 & \vec{v}_3 \\ \vdots & \vdots & \vdots \end{bmatrix} = \vec{A}\vec{V}$$

$$\vec{A} = \vec{M}\vec{V}^{-1}$$

The implementation of the deterministic attitude determination method above assumes that we have three or more measurements. However, if there are only two measurements, a third measurement can be “created” using the cross product. This technique creates measurements and corresponding ground truth vectors  $p$ ,  $q$ , and  $r$  as shown below.

$$\begin{aligned} \tilde{\vec{m}}_1 &= \frac{\vec{m}_1 + \vec{m}_2}{2} & \tilde{\vec{m}}_2 &= \frac{\vec{m}_1 - \vec{m}_2}{2} & \tilde{\vec{v}}_1 &= \frac{\vec{v}_1 + \vec{v}_2}{2} & \tilde{\vec{v}}_2 &= \frac{\vec{v}_1 - \vec{v}_2}{2} \\ \vec{p}_m &= \tilde{\vec{m}}_1; & \vec{q}_m &= \frac{\tilde{\vec{m}}_1 \times \tilde{\vec{m}}_2}{\|\tilde{\vec{m}}_1 \times \tilde{\vec{m}}_2\|} & \vec{r}_m &= \vec{p}_m \times \vec{v}_m \\ \vec{p}_v &= \tilde{\vec{v}}_1; & \vec{q}_v &= \frac{\tilde{\vec{v}}_1 \times \tilde{\vec{v}}_2}{\|\tilde{\vec{v}}_1 \times \tilde{\vec{v}}_2\|} & \vec{r}_v &= \vec{p}_v \times \vec{v}_v \end{aligned}$$

*Implement the statistical attitude determination algorithm discussed in class (q-method)*

Given measurements  $\vec{M}$ , ground truth  $\vec{V}$ , and sensor weights  $\vec{W}$ , we can use statistical attitude determination (aka the q-method). This method ultimately finds the quaternions using the eigenvalue/eigenvector below, where the eigenvector associated with the largest eigenvalue is the quaternion vector.

$$\vec{K}\vec{q} = \lambda\vec{q}$$

In the eigenvalue decomposition above,  $\vec{K}$  can be found with the following relation.

$$\begin{aligned} \vec{w}_i &= \sqrt{w_1}\vec{m}_1 & \vec{u}_i &= \sqrt{w_1}\vec{v}_1 \\ \vec{W} &= [\vec{w}_1 \quad \vec{w}_2 \quad \dots \quad \vec{w}_n] & \vec{U} &= [\vec{u}_1 \quad \vec{u}_2 \quad \dots \quad \vec{u}_n] \\ \vec{B} &= \vec{W}\vec{U}^T & \vec{S} &= \vec{B} + \vec{B}^T & \sigma &= \text{trace}(\vec{B}) \\ Z &= [B_{23} - B_{32} \quad B_{31} - B_{13} \quad B_{12} - B_{21}]^T \\ K &= \begin{bmatrix} \vec{S} - \sigma\vec{I} & \vec{Z} \\ \vec{Z}^T & \sigma \end{bmatrix} \end{aligned}$$

*Implement angular velocity measurements and the reconstruction of the attitude from those (through kinematic equations coded identically to ground truth but replicated in the spacecraft on-board computer)*

We simply duplicate our previously-implemented kinematic equations and pass in the angular velocity to reconstruct attitude evolution for this technique.

The methods for attitude determination are implemented in the following functions. We include implementations for deterministic attitude determination both for two measurements and for three or more measurements.

```

1 function A = DAD_twoVecs(m1, m2, v1, v2)
2     % Implement the Deterministic Attitude Determination algorithm
3     % Inputs:
4     % - m1, m2: measured vectors
5     % - v1, v2: ground truth vector
6     % Outputs:
7     % - A: DCM between principal axes and ECI
8
9     m1_tilde = (m1 + m2)/2;
10    m2_tilde = (m1 - m2)/2;
11    v1_tilde = (v1 + v2)/2;
12    v2_tilde = (v1 - v2)/2;
13
14    m1_tilde = m1_tilde / norm(m1_tilde);
15    m2_tilde = m2_tilde / norm(m2_tilde);
16    v1_tilde = v1_tilde / norm(v1_tilde);
17    v2_tilde = v2_tilde / norm(v2_tilde);
18
19    pm = m1_tilde;
20    cross_qm = cross(m1_tilde, m2_tilde);
21    qm = cross_qm/norm(cross_qm);
22    rm = cross(pm, qm);
23
24    pv = v1_tilde;
25    cross_qv = cross(v1_tilde, v2_tilde);
26    qv = cross_qv/norm(cross_qv);
27    rv = cross(pv, qv);
28
29    M = [pm qm rm];
30    V = [pv qv rv];
31    A = M / V;
32 end

```

```

1 function A = DAD(m, v)
2     % Implement the Deterministic Attitude Determination algorithm
3     % Inputs:
4     % - m: measured vectors ([m1 m2 . . . ])
5     % - v: ground truth vector ([v1 v2 . . . ])
6     % Outputs:
7     % - A: DCM between principal axes and ECI
8
9     A = m * pinv(v);
10
11 end

```

```

1 function q = qMethod(m, v, w)
2     % Implement the Deterministic q method
3     % Inputs:
4     % - m: measured vectors
5     % - v: ground truth vector
6     % - w: weights of sensor measurements
7     % Outputs:
8     % - A: DCM between principal axes and ECI
9
10    U = sqrt(w) .* v;
11    W = sqrt(w) .* m;
12
13    B = W * U';
14    S = B + B';
15    Z = [B(2,3)-B(3,2), B(3,1)-B(1,3), B(1,2)-B(2,1)]';
16    sigma = trace(B);
17
18    K = [S - eye(size(S))*sigma, Z;
19          Z', sigma];
20
21    [V, lamda] = eig(K);
22
23    [~, nMax] = max(diag(lamda));
24
25    q = V(:, nMax);
26    q = A2q(q2A(q));
27 end

```

```

1 function [eulerAngs, omega] = kinEulerStepRK4(eulerAngs, omega, Ix, ...
2     Iy, Iz, tStep)
3     state = zeros(6,1);
4     state(1:3) = eulerAngs;
5     state(4:6) = omega;
6
7     t = 0;
8
9     k1 = kinEulerAngle(t, state, Ix, Iy, Iz);
10    k2 = kinEulerAngle(t+tStep/2, state+(k1*tStep/2), Ix, Iy, Iz);
11    k3 = kinEulerAngle(t+tStep/2, state+(k2*tStep/2), Ix, Iy, Iz);
12    k4 = kinEulerAngle(t+tStep, state+(k3*tStep), Ix, Iy, Iz);
13    nextState = state + tStep * (k1/6 + k2/3 + k3/3 + k4/6);
14
15    eulerAngs = nextState(1:3);
16    omega = nextState(4:6);
17 end

```

## 6.6 PROBLEM 6

*Plot the resulting estimated attitude in the absence of sensor errors. Show that it is identical to the true attitude (except for numerical errors).*

Figures 70, 71, 72, and 73 depict the attitude determination using the different methods dis-

cussed earlier. All are identical, which is expected, since each algorithm is given perfect measurements free of noise. Error for each is on the order of  $10^{-16}$ , which is purely numerical.

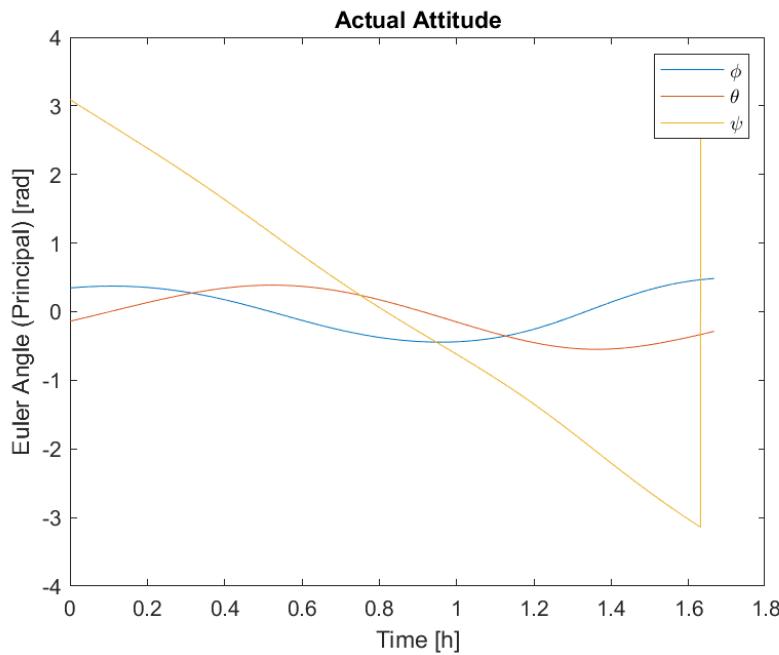


Figure 70: True satellite attitude relative to inertial frame

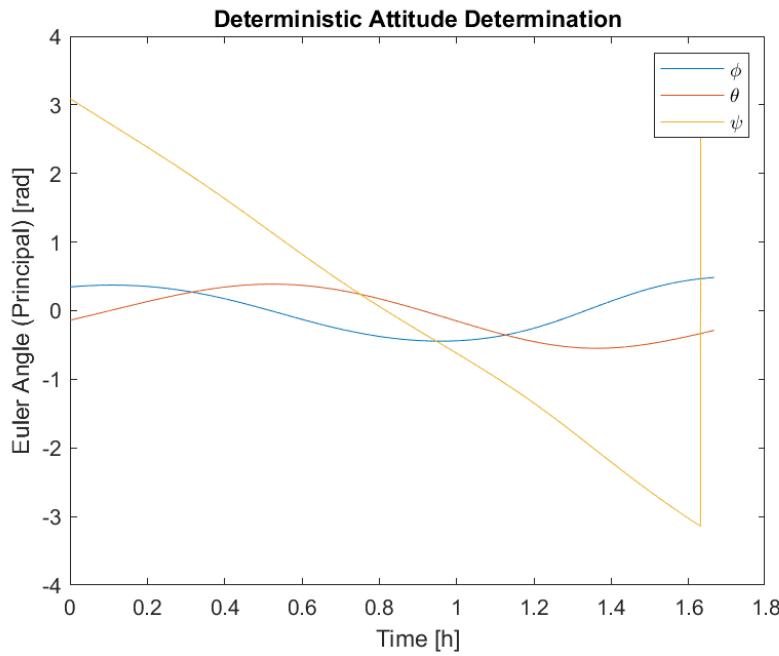


Figure 71: Deterministic attitude determination

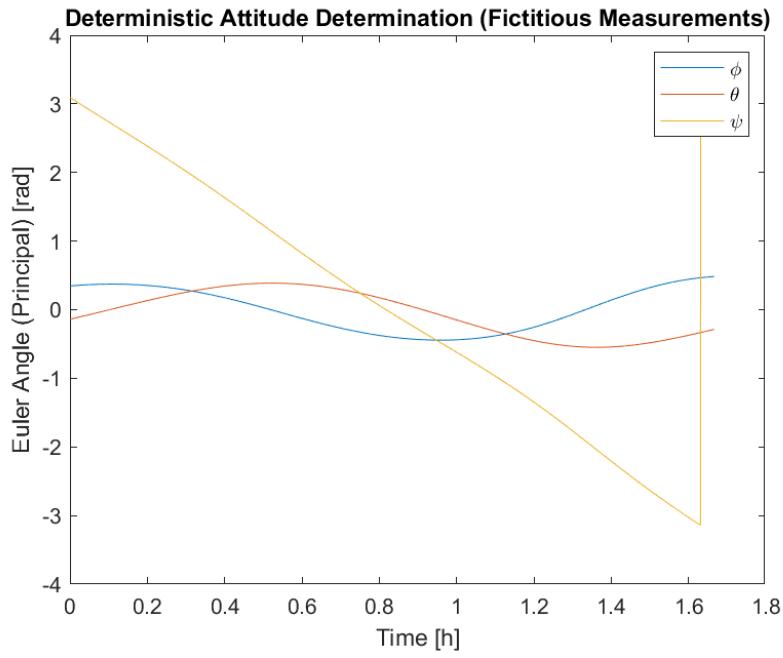


Figure 72: Deterministic attitude determination with fictitious measurements

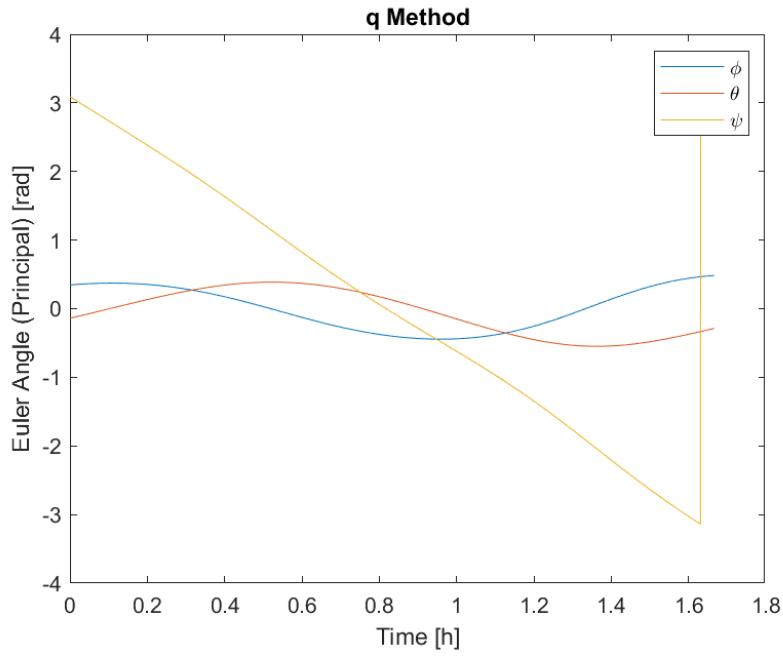


Figure 73: q-method

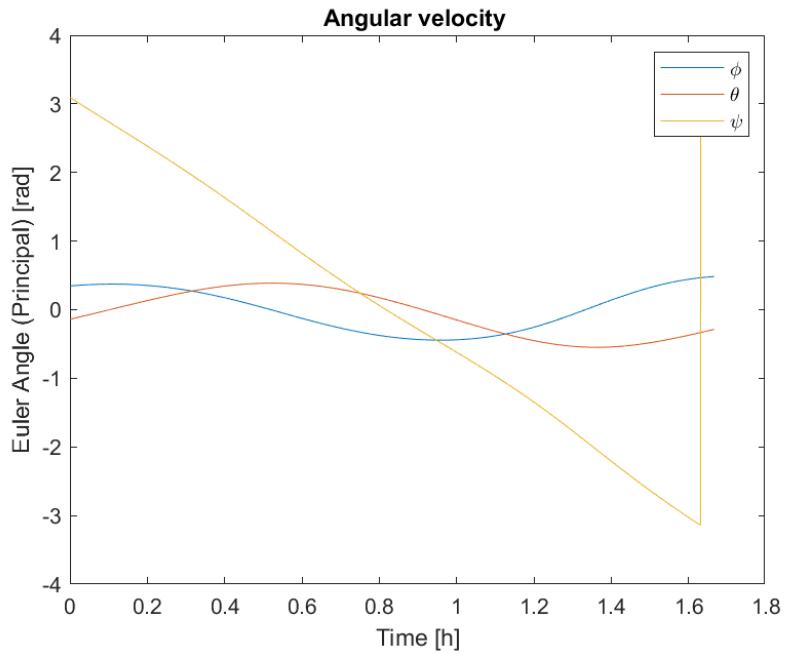


Figure 74: Attitude reconstruction using kinematic equations

## 7 REFERENCES

- [1] K. H. Kellogg, S. Thurman, W. Edelstein, *et al.*, “NASA’s SMAP Observatory,” in *2013 IEEE Aerospace Conference*, Jet Propulsion Laboratory, National Aeronautics and Space Administration, 2013. DOI: 2014/44370. [Online]. Available: <https://hdl.handle.net/2014/44370>.
- [2] *EOS SAR Satellites*, Sep. 2021. [Online]. Available: <https://eossar.com/technology/>.
- [3] *Capella Space*, Apr. 2024. [Online]. Available: <https://www.capellaspace.com/>.
- [4] K. H. Kellogg, P. Barela, R. Sagi, *et al.*, “NASA-ISRO Synthetic Aperture Radar (NISAR) Mission,” in *2020 IEEE Aerospace Conference*, Jet Propulsion Laboratory, National Aeronautics and Space Administration, 2020. DOI: 2014/51150. [Online]. Available: <https://hdl.handle.net/2014/51150>.
- [5] N. N. Stavros, S. Owen, C. Jones, and B. Osmanoglu, *NISAR Applications*, version V2, 2018. DOI: 2014/49181. [Online]. Available: <https://hdl.handle.net/2014/49181>.
- [6] P. Siqueira, *The NISAR Mission*, 2018. [Online]. Available: [https://climate.esa.int/sites/default/files/D1\\_S1\\_T7\\_Siqueira.pdf](https://climate.esa.int/sites/default/files/D1_S1_T7_Siqueira.pdf).
- [7] Spectrolab, *Space solar panels datasheet*. [Online]. Available: <https://www.spectrolab.com/DataSheets/Panel/panels.pdf>.
- [8] L3Harris, *Prebuilt 12-meter s- or l-band reflector*, Mar. 2021. [Online]. Available: <https://www.l3harris.com/sites/default/files/2021-03/l3harris-prebuilt-12m-unfurlable-mesh-reflector-spec-sheet-sas.pdf>.
- [9] S. G. McCarron, “The effect of a change in orientation of a rectangular four-paddle solar array on the spin rate of a satellite,” Goddard Space Flight Center, National Aeronautics and Space Administration, Tech. Rep., Jan. 1966.
- [10] “NASA-ISRO SAR (NISAR) Mission Science Users’ Handbook,” Jet Propulsion Laboratory, National Aeronautics and Space Administration, Tech. Rep., 2019. [Online]. Available: [https://nisar.jpl.nasa.gov/system/documents/files/26\\_NISAR\\_FINAL\\_9-6-19.pdf](https://nisar.jpl.nasa.gov/system/documents/files/26_NISAR_FINAL_9-6-19.pdf).
- [11] Rockwell Collins, *Ht-rsi high motor torque momentum and reaction wheels 14 – 68 nms with integrated wheel drive electronics*, 2007.
- [12] J. R. Wertz, *Spacecraft Attitude Determination and Control*. Springer Dordrecht, 1978. DOI: <https://doi.org/10.1007/978-94-009-9907-7>.

## A Appendix A

The following MATLAB code are used in problem sets, in addition to functions already listed in the body of the document. The full code (including helper functions not shown here) and resource files can be found in the GitHub repository: <https://github.com/zhao-harry/aa-279c-project>

### A.1 Problem Set 1

```
1 %% Center of mass
2 cm = computeCM('res/mass.csv');
3
4 %% Moment of inertia
5 origin = [0;0;0];
6 I = computeMOI('res/mass.csv',origin);
7
8 %% Surface properties
9 [barycenter,normal,area] = surfaces('res/area.csv');
10
11 %% Plot spacecraft with body axes
12 figure
13 gm = importGeometry('res/NISAR.stl');
14 pdegplot(gm);
15 quiver = findobj(gca,'type','Quiver');
16 textx = findobj(gca,'type','Text','String','x');
17 texty = findobj(gca,'type','Text','String','y');
18 textz = findobj(gca,'type','Text','String','z');
19 set(quiver,'XData',[0;0;0])
20 set(quiver,'YData',[0;0;0])
21 set(quiver,'ZData',[0;0;0])
22 set(textx,'Position',[4 0 0])
23 set(texty,'Position',[0 4 0])
24 set(textz,'Position',[0 0 4])
25 saveas(gcf,'Images/ps1_model.png');
```

## A.2 Problem Set 2

```
1 %% Problem Set 2
2 clear; close all; clc;
3
4 %% Problem 1
5 a = 7125.48662; % km
6 e = 0.0011650;
7 i = 98.40508; % degree
8 O = -19.61601; % degree
9 w = 89.99764; % degree
10 nu = -89.99818; % degree
11
12 yECI = oe2eci(a,e,i,O,w,nu);
13
14 days = 0.5;
15 tspan = 0:days*86400;
16 options = odeset('RelTol',1e-6,'AbsTol',1e-9);
17 [t,y] = ode113(@orbitSimple,tspan,yECI,options);
18
19 plot3(y(:,1),y(:,2),y(:,3),'LineWidth',2,'Color','green')
20 xlabel('x [km]')
21 ylabel('y [km]')
22 zlabel('z [km]')
23 axis equal
24 hold on
25 [xE,yE,zE] = ellipsoid(0,0,0,6378.1,6378.1,6378.1,20);
26 surface(xE,yE,zE,'FaceColor','blue','EdgeColor','black');
27 hold off
28 saveas(gcf,'Images/ps2_problem1.png');
29
30 %% Problem 2
31 cm = computeCM('res/mass.csv');
32 I = computeMOI('res/mass.csv',cm);
33
34 [rot,IPrincipal] = eig(I);
35 Ix = IPrincipal(1,1);
36 Iy = IPrincipal(2,2);
37 Iz = IPrincipal(3,3);
38 xPrincipal = rot(:,1);
39 yPrincipal = rot(:,2);
40 zPrincipal = rot(:,3);
41
42 %% Problem 3
43 figure
44 gm = importGeometry('res/NISAR.stl');
45 pdegplot(gm);
46
47 quiver = findobj(gca,'type','Quiver');
48 textx = findobj(gca,'type','Text','String','x');
49 texty = findobj(gca,'type','Text','String','y');
50 textz = findobj(gca,'type','Text','String','z');
51 set(quiver,"XData",[0;0;0])
52 set(quiver,"YData",[0;0;0])
53 set(quiver,"ZData",[0;0;0])
54 set(textx,"Position",[4 0 0])
55 set(texty,"Position",[0 4 0])
```

```

56 set(textz,"Position",[0 0 4])
57
58 quiverPrincipal = copyobj(quiver,gca);
59 textxPrincipal = copyobj(textx,gca);
60 textyPrincipal = copyobj(texty,gca);
61 textzPrincipal = copyobj(textz,gca);
62 set(quiver,"Color",[0 1 0])
63 set(quiver,"UData",4.14 * rot(1,:)')
64 set(quiver,"VData",4.14 * rot(2,:)')
65 set(quiver,"WData",4.14 * rot(3,:)')
66 set(quiver,"XData",repmat(cm(1),3,1))
67 set(quiver,"YData",repmat(cm(2),3,1))
68 set(quiver,"ZData",repmat(cm(3),3,1))
69 set(textx,"String",'x''')
70 set(texty,"String",'y''')
71 set(textz,"String",'z''')
72 set(textx,"Position",4 * xPrincipal + cm)
73 set(texty,"Position",4 * yPrincipal + cm)
74 set(textz,"Position",4 * zPrincipal + cm)
75 saveas(gcf,'Images/ps2_model.png');
76
77 %% Problem 5
78 w0Deg = [8;4;6];
79 w0 = deg2rad(w0Deg);
80 tspan = 0:120;
81 w = eulerPropagator(w0,Ix,Iy,Iz,tspan,'Images/ps2_euler_equations.png');
82
83 %% Problem 6
84 [XE,YE,ZE] = ellipsoidEnergy(IPrincipal, ...
85     w0, ...
86     'Images/ps2_problem6_energy.png');
87 [XM,YM,ZM] = ellipsoidMomentum(IPrincipal, ...
88     w0, ...
89     'Images/ps2_problem6_momentum.png');
90
91 %% Problem 7
92 w = polhode(XE,YE,ZE,XM,YM,ZM,w,'Images/ps2_problem7.png');
93
94 %% Problem 8
95 w = polhode2D(w,'none','Images/ps2_problem8.png');
96
97 %% Problem 9, x-axis
98 axis = 'x';
99 w0Deg = 8*[1;0;0];
100 w0 = deg2rad(w0Deg);
101 tspan = 0:120;
102 marker = 'o';
103
104 %% Problem 9, y-axis
105 axis = 'y';
106 w0Deg = 8*[0.01;1;0.01];
107 w0 = deg2rad(w0Deg);
108 tspan = 0:1200;
109 marker = 'none';
110
111 %% Problem 9, z-axis
112 axis = 'z';
113 w0Deg = 8*[0.01;0;1];

```

```

114 w0 = deg2rad(w0Deg);
115 tspan = 0:120;
116 marker = 'none';
117
118 %% Problem 9
119 w = eulerPropagator(w0,Ix,Iy,Iz,tspan, ...
120     ['Images/ps2_problem9_euler_equations_', axis, '.png']);
121
122 [XE,YE,ZE] = ellipsoidEnergy(IPrincipal,w0, ...
123     ['Images/ps2_problem9_energy_', axis, '.png']);
124 [XM,YM,ZM] = ellipsoidMomentum(IPrincipal,w0, ...
125     ['Images/ps2_problem9_momentum_', axis, '.png']);
126
127 w = polhode(XE,YE,ZE,XM,YM,ZM,w, ...
128     ['Images/ps2_problem9_p7_', axis, '.png']);
129
130 w = polhode2D(w,marker, ...
131     ['Images/ps2_problem9_p8_', axis, '.png']);

```

```

1 function [t,y] = plotECI(a,e,i,O,w,nu,tspan)
2 yECI = oe2eci(a,e,i,O,w,nu);
3 options = odeset('RelTol',1e-6,'AbsTol',1e-9);
4 [t,y] = ode113(@orbitSimple,tspan,yECI,options);
5 plot3(y(:,1),y(:,2),y(:,3),'LineWidth',2,'Color','green')
6 xlabel('x [km]')
7 ylabel('y [km]')
8 zlabel('z [km]')
9 axis equal
10 grid on
11 hold on
12 [xE,yE,zE] = ellipsoid(0,0,0,6378.1,6378.1,6378.1,20);
13 surface(xE,yE,zE, ...
14     'FaceColor','blue', ...
15     'EdgeColor','black', ...
16     'FaceAlpha',0.1);
17 hold off
18 end

```

```

1 function w = eulerPropagator(w0,Ix,Iy,Iz,tspan,filename)
2 options = odeset('RelTol',1e-6,'AbsTol',1e-9);
3 [t,w] = ode113(@(t,w) eulerEquation(t,w,Ix,Iy,Iz),tspan,w0,options);
4 wDeg = rad2deg(w);
5
6 figure(1)
7 plot(t,wDeg,'LineWidth',2)
8 legend(['\omega_x','\omega_y','\omega_z', ...
9     'Location','southeast'])
10 xlabel('Time [s]')
11 ylabel(['Angular velocity (\omega) [ \pi char(176) /s]'])
12 saveas(1,filename)
13 end

```

```
1 function [XE,YE,ZE] = ellipsoidEnergy(IPrincipal,w0,filename)
```

```

2     Ix = IPrincipal(1,1);
3     Iy = IPrincipal(2,2);
4     Iz = IPrincipal(3,3);
5     T = sum(IPrincipal * w0.^2,"all") / 2;
6     L = sqrt(sum((w0.*IPrincipal).^2,"all"));
7     [XE,YE,ZE] = ...
8         ellipsoid(0,0,0,sqrt(2*T/Ix),sqrt(2*T/Iy),sqrt(2*T/Iz),50);
9     ellipsoidAxes = [sqrt(2*T/Ix), sqrt(2*T/Iy), sqrt(2*T/Iz)];
10
11    % Plot energy ellipsoid
12    figure(1)
13    surf(XE,YE,ZE, ...
14        'FaceAlpha',0.5, ...
15        'FaceColor','blue', ...
16        'DisplayName','Energy Ellipsoid');
17    axis equal
18    hold on
19    quiver3(0, 0, 0, ellipsoidAxes(1), 0, 0, 'Color', 'r', ...
20            'LineWidth', 2)
21    quiver3(0, 0, 0, 0, ellipsoidAxes(2), 0, 'Color', 'r', ...
22            'LineWidth', 2)
23    quiver3(0, 0, 0, 0, 0, ellipsoidAxes(3), 'Color', 'r', ...
24            'LineWidth', 2)
25    xlabel('omega_x [rad/s]')
26    ylabel('omega_y [rad/s]')
27    zlabel('omega_z [rad/s]')
28    hold off
29    saveas(1,filename)
30
31    I = L^2/(2*T);
32    if (Ix <= I || ismembertol(Ix, I, 1e-7)) && I <= Iz
33        fprintf("The polhode is real!\n")
34    else
35        error("The polhode is NOT real!\n")
36    end
37 end

```

```

1 function [XM,YM,ZM] = ellipsoidMomentum(IPrincipal,w0,filename)
2     Ix = IPrincipal(1,1);
3     Iy = IPrincipal(2,2);
4     Iz = IPrincipal(3,3);
5     T = sum(IPrincipal * w0.^2,"all") / 2;
6     L = sqrt(sum((w0.*IPrincipal).^2,"all"));
7     [XM,YM,ZM] = ellipsoid(0,0,0,L/Ix,L/Iy,L/Iz,50);
8     momentumAxes = [L/Ix, L/Iy, L/Iz];
9
10    % Plot momentum ellipsoid
11    figure(1)
12    surf(XM,YM,ZM, ...
13        'FaceAlpha',0.5, ...
14        'FaceColor','green', ...
15        'DisplayName','Momentum Ellipsoid');
16    axis equal
17    hold on
18    quiver3(0, 0, 0, momentumAxes(1), 0, 0, 'Color', 'r', ...
19            'LineWidth', 2)

```

```

19 quiver3(0, 0, 0, 0, momentumAxes(2), 0, 'Color', 'r', ...
20     'LineWidth', 2)
21 quiver3(0, 0, 0, 0, 0, momentumAxes(3), 'Color', 'r', ...
22     'LineWidth', 2)
23 xlabel('\omega_x [rad/s]')
24 ylabel('\omega_y [rad/s]')
25 zlabel('\omega_z [rad/s]')
26 hold off
27 saveas(1,filename)
28
29 I = L^2/(2*T);
30 if (Ix <= I || ismembertol(Ix, I, 1e-7)) && I <= Iz
31     fprintf("The polhode is real!\n")
32 else
33     error("The polhode is NOT real!\n")
34 end
35 end

```

```

1 function w = polhode(XE,YE,ZE,XM,YM,ZM,w,filename)
2 figure(1)
3 surf(XE,YE,ZE, ...
4     'FaceAlpha',0.5, ...
5     'FaceColor','blue', ...
6     'DisplayName','Energy Ellipsoid');
7 xlabel('\omega_x [rad/s]')
8 ylabel('\omega_y [rad/s]')
9 zlabel('\omega_z [rad/s]')
10 axis equal
11 hold on
12 surf(XM,YM,ZM, ...
13     'FaceAlpha',0.5, ...
14     'FaceColor','green', ...
15     'DisplayName','Momentum Ellipsoid');
16 plot3(w(:,1),w(:,2),w(:,3), ...
17     'LineWidth',2, ...
18     'Color','red', ...
19     'DisplayName','Polhode')
20 legend('Location','northwest')
21 hold off
22 saveas(1,filename)
23 end

```

```

1 function w = polhode2D(w,marker,filename)
2 subplot(1,3,1)
3 plot(w(:,2),w(:,3),'Marker',marker)
4 title('Polhode (along x-axis)')
5 xlabel('\omega_y [rad/s]')
6 ylabel('\omega_z [rad/s]')
7 axis equal
8
9 subplot(1,3,2)
10 plot(w(:,1),w(:,3),'Marker',marker)
11 title('Polhode (along y-axis)')
12 xlabel('\omega_x [rad/s]')
13 ylabel('\omega_z [rad/s]')

```

```
14     axis equal
15
16     subplot(1,3,3)
17     plot(w(:,1),w(:,2), 'Marker',marker)
18     title('Polhode (along z-axis)')
19     xlabel('\omega_x [rad/s]')
20     ylabel('\omega_y [rad/s]')
21     axis equal
22
23     saveas(l,filename)
24 end
```

### A.3 Problem Set 3

```

1 clear; close all; clc
2
3 %% Problem 1
4 IPrincipal = [7707.07451493673 0 0; ...
5     0 7707.0745149367 0; ...
6     0 0 18050.0227594212];
7 Ix = IPrincipal(1,1);
8 Iy = IPrincipal(2,2);
9 Iz = IPrincipal(3,3);
10
11 w0Deg = [8;4;6];
12 w0 = deg2rad(w0Deg);
13 tspan = 0:0.1:120;
14 w = eulerPropagator(w0,Ix,Iy,Iz,tspan,'Images/ps3_problem1.png');
15
16 %% Problem 2
17 lambda = w0(3) * (Iz - Iy) / Ix;
18 wxy = (w0(1) + w0(2) * 1j) * exp(1j * lambda * tspan);
19 wx = real(wxy);
20 wy = imag(wxy);
21 wz = w0(3) * ones(size(wxy));
22 wAnalytical = [wx',wy',wz'];
23 wDegAnalytical = rad2deg(wAnalytical);
24
25 figure(1)
26 plot(tspan,wDegAnalytical,'LineWidth',2)
27 legend('\omega_x','\omega_y','\omega_z', ...
28     'Location','southeast')
29 xlabel('Time [s]')
30 ylabel(['Angular velocity (\omega) [' char(176) '/s]'])
31 saveas(1,'Images/ps3_problem2.png')
32
33 %% Problem 3
34 % Error plots
35 error = w - wAnalytical;
36 plot(tspan,error,'LineWidth',2)
37 legend('\omega_x','\omega_y','\omega_z', ...
38     'Location','southeast')
39 xlabel('Time [s]')
40 ylabel('Angular velocity (\omega) [rad/s]')
41 saveas(gcf,'Images/ps3_problem3.png')
42
43 % Verify L and omega
44 L_principal = [Ix Iy Iz] .* w;
45 keyTimes = [1, 61, 121, 181, 241, 361];
46 for n = keyTimes
47     figure(1)
48     L_unit = L_principal(n,:)/norm(L_principal(n,:));
49     w_unit = w(n,:)/norm(w(n,:));
50     quiver3(0,0,0,w_unit(1),w_unit(2),w_unit(3),1,'r')
51     hold on
52     quiver3(0,0,0,L_unit(1),L_unit(2),L_unit(3),1,'b')
53     quiver3(0,0,0,0,0,1,'k')
54     xlim([-1 1]); ylim([-1 1]); zlim([-1 1]);
55     xlabel('x'); ylabel('y'); zlabel('z');

```

```

56     legend('omega', 'L', 'z-axis', 'Location', 'northeast')
57     title(sprintf('Unit vectors at t = %.2f s', tspan(n)))
58     hold off
59     saveas(1, sprintf('Images/ps3_problem3_vectors_%i.png', n))
60 end
61
62 %% Non-Axisymmetric Satellite
63 cm = computeCM('res/mass.csv');
64 I = computeMOI('res/mass.csv', cm);
65
66 [rot, IPrincipal] = eig(I);
67 Ix = IPrincipal(1,1);
68 Iy = IPrincipal(2,2);
69 Iz = IPrincipal(3,3);
70 xPrincipal = rot(:,1);
71 yPrincipal = rot(:,2);
72 zPrincipal = rot(:,3);
73
74 %% Problem 6 (Quaternions)
75 axang0 = [sqrt(1/2) sqrt(1/2) 0 pi/4];
76 q0 = axang2quat(axang0).';
77 tFinal = 600;
78 tStep = 0.1;
79 t = 0:tStep:tFinal;
80
81 % Forward Euler
82 % [q,w] = kinQuaternionForwardEuler(q0,w0,Ix,Iy,Iz,tFinal,tStep);
83
84 % RK4
85 [q,w] = kinQuaternionRK4(q0,w0,Ix,Iy,Iz,tFinal,tStep);
86
87 figure(2)
88 hold on
89 plot(t,q,'LineWidth',1)
90 legend('q-{1}', 'q-{2}', 'q-{3}', 'q-{4}', ...
91     'Location', 'Southeast')
92 xlabel('Time [s]')
93 ylabel('Quaternion')
94 hold off
95 saveas(2, 'Images/ps3_problem6_quaternions.png')
96
97 %% Problem 6 (Euler Angles)
98 eulerAngle0 = rotm2eul(axang2rotm(axang0))';
99 state0 = [eulerAngle0;w0];
100
101 tFinal = 600;
102 tStep = 0.1;
103 t = 0:tStep:tFinal;
104
105 % Forward Euler
106 % state = kinEulerAngleForwardEuler(state0,Ix,Iy,Iz,tFinal,tStep);
107
108 % ode113
109 tspan = 0:tStep:tFinal;
110 options = odeset('RelTol',1e-6, 'AbsTol', 1e-9);
111 [t,state] = ode113(@(t,state) kinEulerAngle(t,state,Ix,Iy,Iz), ...
112     tspan, state0, options);
113

```

```

114 eulerAngle = wrapTo360(rad2deg(state(:,1:3)));
115
116 figure(3)
117 plot(t,eulerAngle,'LineWidth',1)
118 legend('\phi','\theta','\psi', ...
119     'Location','southwest')
120 xlabel('Time [s]')
121 ylabel('Euler Angle [deg]')
122 saveas(3,'Images/ps3_problem6_euler.png')
123
124 %% Problem 7(a)
125 % Part a: Angular momentum
126 tLen = length(t);
127 L_principal = [Ix Iy Iz] .* w;
128 L_inertial = nan(size(L_principal));
129 L_norm = nan(1, tLen);
130
131 % Part b: Herpolhode
132 w_inertial = nan(size(w));
133
134 for i = 1:tLen
135     % Get rotation matrix
136     qi = q(i,:);
137     A = q2A(qi);
138
139     % Angular momentum
140     L_inertial(i,:) = A' * L_principal(i,:)';
141     L_norm(i) = norm(L_inertial(i,:));
142
143     % Angular velocity
144     w_inertial(i,:) = A' * w(i,:)';
145 end
146
147 figure(4)
148 hold on
149 plot(t, L_inertial)
150 plot(t, L_norm, 'k--')
151 xlabel('Time [s]')
152 ylabel('Angular momentum [kg m^2/s]')
153 legend("L_{1}", "L_{2}", "L_{3}", "||L||")
154 hold off
155 saveas(4, 'Images/ps3_problem7a.png')
156
157 %% Problem 7(b)
158 figure(5)
159 plot3(w_inertial(:,1), w_inertial(:,2), w_inertial(:,3), 'r')
160 grid on
161 hold on
162 quiver3(0, 0, 0, ...
163     L_inertial(1,1), L_inertial(1,2), L_inertial(1,3), ...
164     1e-4)
165 quiver3(0, 0, 0, ...
166     w_inertial(1,1), w_inertial(1,2), w_inertial(1,3), ...
167     1)
168 xlabel('\omega_x [rad/s]')
169 ylabel('\omega_y [rad/s]')
170 zlabel('\omega_z [rad/s]')
171 legend('Herpolhode', ...

```

```

172      'Angular momentum (L)', ...
173      'Angular velocity (\omega)', ...
174      'Location','northwest')
175 hold off
176 saveas(5, 'Images/ps3_problem7b.png')
177
178 %% For fun kinda thing
179 saveGif = true;
180 tGif = 240 / tStep;
181
182 L_unit = nan(size(L_inertial));
183 w_unit = nan(size(w_inertial));
184 if saveGif == true
185     gif = figure;
186     for i = 1:tLen
187         w_unit(i,:) = w_inertial(i,:)./norm(w_inertial(i,:));
188         L_unit(i,:) = L_inertial(i,:)./norm(L_inertial(i,:));
189     end
190
191     for i = 1:20:tGif
192         plot3(w_unit(1:i,1), w_unit(1:i,2), w_unit(1:i,3), 'r')
193         grid on
194         hold on
195         quiver3(0, 0, 0, w_unit(i,1), w_unit(i,2), w_unit(i,3),1)
196         quiver3(0, 0, 0, L_unit(i,1), L_unit(i,2), L_unit(i,3),1)
197         hold off
198         xlim([-1 1])
199         ylim([-1 1])
200         zlim([-1 1])
201         xlabel('x')
202         ylabel('y')
203         zlabel('z')
204         title('Note: all vectors are normalized')
205         legend('Hepolhode','\omega','L','Location','northeast')
206         exportgraphics(gif,'Images/ps3_problem7b.gif','Append',true);
207     end
208 end
209
210 %% Problem 7(c)
211 % Generate orbit
212 a = 7125.48662; % km
213 e = 0.0011650;
214 i = 98.40508; % degree
215 O = -19.61601; % degree
216 w = 89.99764; % degree
217 nu = -89.99818; % degree
218
219 days = 0.069;
220 tFinal = days * 86400;
221 tStep = 1;
222 tspan = 0:tStep:tFinal;
223
224 figure(6)
225 [t,y] = plotECI(a,e,i,O,w,nu,tspan);
226 hold on
227 figure(7)
228 plotECI(a,e,i,O,w,nu,tspan);
229 hold on

```

```

230 figure(8)
231 plotECI(a,e,i,O,w,nu,tspan);
232 hold on
233
234 [q,w] = kinQuaternionRK4(q0,w0,Ix,Iy,Iz,tFinal,tStep);
235
236 tLen = length(t);
237 for i = 1:500:tLen
238     % Get rotation matrix
239     qi = q(i,:);
240     A = q2A(qi);
241     % Body axes
242     B = rot * A * rot';
243     % Position
244     pos = y(i,1:3);
245     radial = pos / norm(pos);
246     tangential = y(i,4:6) / norm(y(i,4:6));
247     normal = cross(radial,tangential);
248     RTN = [radial' tangential' normal'];
249     figure(6);
250     plotTriad(gca,pos,A,1e3,'r');
251     figure(7);
252     plotTriad(gca,pos,B,1e3,'m');
253     figure(8);
254     plotTriad(gca,pos,RTN,1e3,'b');
255 end
256 figure(6);
257 legend('Orbit','Earth','Principal (x-axis)','Location','northwest')
258 hold off
259 saveas(gcf,'Images/ps3_problem7c_principal.png');
260 figure(7);
261 legend('Orbit','Earth','Body (x-axis)','Location','northwest')
262 hold off
263 saveas(gcf,'Images/ps3_problem7c_body.png');
264 figure(8);
265 legend('Orbit','Earth','RTN (radial axis)','Location','northwest')
266 hold off
267 saveas(gcf,'Images/ps3_problem7c_rtn.png');

```

```

1 function M = plotTriad(ax,o,M,scale,colorString)
2     quiver3(ax, ...
3             o(1),o(2),o(3), ...
4             M(1,1),M(2,1),M(3,1), ...
5             scale, ...
6             'LineWidth',1, ...
7             'Color','k')
8     quiver3(ax, ...
9             o(1),o(2),o(3), ...
10            M(1,2),M(2,2),M(3,2), ...
11            scale, ...
12            'LineWidth',1, ...
13            'Color',colorString)
14     quiver3(ax, ...
15             o(1),o(2),o(3), ...
16             M(1,3),M(2,3),M(3,3), ...
17             scale, ...

```

```
18      'LineWidth',1, ...
19      'Color',colorString)
20 end
```

## A.4 Problem Set 4

```
1 close all; clear; clc
2 savePlot = false;
3
4 %% Import mass properties
5 cm = computeCM('res/mass.csv');
6 I = computeMOI('res/mass.csv',cm);
7
8 [rot,IPrincipal] = eig(I);
9 Ix = IPrincipal(1,1);
10 Iy = IPrincipal(2,2);
11 Iz = IPrincipal(3,3);
12
13 %% Problem 1(a)
14 tFinal = 60;
15 tStep = 0.01;
16 tspan = 0:tStep:tFinal;
17
18 eulerAngle0 = [0; 0; 0];
19 w0 = [0; 0; 1];
20 state0 = [eulerAngle0;w0];
21
22 options = odeset('RelTol',1e-6,'AbsTol',1e-9);
23 [t,state] = ode113(@(t,state) kinEulerAngle(t,state,Ix,Iy,Iz), ...
24     tspan,state0,options);
25
26 state = rad2deg(state);
27
28 % Plot
29 figure()
30 plot(t,state(:,4:6),'LineWidth',1)
31 legend('\omega_x','\omega_y','\omega_z', ...
32     'Location','southeast')
33 xlabel('Time [s]')
34 ylabel(['Angular Velocity (\omega) [' char(176) '/s]'])
35 if savePlot
36     saveas(gcf,'Images/ps4_problemla_angvel.png')
37 end
38
39 figure()
40 plot(t,wrapTo180(state(:,1:3)),'LineWidth',1)
41 legend('\phi','\theta','\psi', ...
42     'Location','southwest')
43 xlabel('Time [s]')
44 ylabel(['Euler Angle [' char(176) ']'])
45 if savePlot
46     saveas(gcf,'Images/ps4_problemla_angle.png')
47 end
48
49 %% Problem 1(b)
50 a = 7125.48662; % km
51 e = 0;
52 i = 98.40508; % degree
53 O = -19.61601; % degree
54 w_deg = 89.99764; % degree
55 nu = -89.99818; % degree
```

```

56
57 muE = 3.986 * 10^5;
58
59 n = sqrt(muE / a^3);
60
61 tFinal = 6000;
62 tStep = 0.1;
63 tspan = 0:tStep:tFinal;
64 tTrunc = 300;
65 nTrunc = find((tspan == tTrunc) == 1);
66
67 [~,y] = plotECI(a,e,i,O,w_deg,nu,tspan);
68 close all
69 format long
70
71 % Initialize angular velocity aligned with normal
72 r0 = y(1,1:3);
73 v0 = y(1,4:6);
74 h = cross(r0,v0);
75 radial = r0 / norm(r0);
76 normal = h / norm(h);
77 tangential = cross(normal,radial);
78 A	RTN = [radial' tangential' normal']';
79 w0	RTN = [0; 0; 0.1];
80 euler0	RTN = A2e(A_RTN);
81 state0 = [euler0_RTN; w0_RTN];
82 options = odeset('RelTol',1e-6,'AbsTol',1e-9);
83 [t,state] = ode113(@(t,state) kinEulerAngle(t,state,Ix,Iy,Iz), ...
84 tspan,state0,options);
85
86 w_RTN = nan(size(state(:,1:3)));
87 euler_RTN = nan(size(state(:,1:3)));
88
89 for n = 1:length(t)
90 pos = y(n,1:3);
91 vel = y(n,4:6);
92 h = cross(pos,vel);
93 radial = pos / norm(pos);
94 normal = h / norm(h);
95 tangential = cross(normal,radial);
96 tangential = tangential / norm(tangential);
97 A_RTN = [radial' tangential' normal']';
98
99 % Get rotation matrixes (to ECI)
100 euler = state(n,1:3);
101 w_principal = state(n,4:6)';
102 A_principal = e2A(euler);
103 A_P2R = A_RTN * A_principal';
104 w_RTN(n,:) = A_P2R*w_principal;
105 euler_RTN(n,:) = A2e(A_P2R');
106 end
107
108 figure()
109 plot(t, rad2deg(w_RTN))
110 xlabel('Time [s]')
111 ylabel(['Angular Velocity, RTN Frame [' char(176) '/s]'])
112 legend('\omega_{R}', '\omega_{T}', '\omega_{N}')
113 if savePlot == true

```

```

114     saveas(gcf, 'Images/ps4_problem1b_angvel.png')
115 end
116
117 figure()
118 plot(t(1:nTrunc), wrapTo180(rad2deg(euler_RTN(1:nTrunc,:))))
119 xlabel('Time [s]')
120 ylabel(['Euler Angle, RTN Frame [' char(176) ']'])
121 legend('\phi', '\theta', '\psi')
122 if savePlot == true
123     saveas(gcf, 'Images/ps4_problem1b_euler.png')
124 end
125
126 %% Problem 2
127 % Initial conditions
128 perturbation = 0.001;
129 w0x = [1; perturbation; perturbation];
130 w0y = [perturbation; 1; perturbation];
131 w0z = [perturbation; perturbation; 1];
132
133 w0Mat = {w0x, w0y, w0z};
134 eulerAngle0 = [0; 0; 0];
135 tStep = 0.01;
136 tFinal = 60;
137
138 for n = 1:3
139     w0 = w0Mat{n};
140     state0 = [eulerAngle0;w0];
141
142     tspan = 0:tStep:tFinal;
143     options = odeset('RelTol',1e-6,'AbsTol',1e-9);
144     [t,state] = ode13(@(t,state) kinEulerAngle(t,state,Ix,Iy,Iz), ...
145         tspan,state0,options);
146
147     eulerAngle = wrapTo180(rad2deg(state(:,1:3)));
148     w = rad2deg(state(:,4:6));
149
150     figure()
151     subplot(2,1,1)
152     plot(t,w)
153     xlabel('Time [s]')
154     ylabel(['Angular Velocity [' char(176) '/s']'])
155     legend('\omega_x', '\omega_y', '\omega_z', ...
156         'Location', 'Southeast')
157
158     subplot(2,1,2)
159     plot(t,eulerAngle)
160     xlabel('Time [s]')
161     ylabel(['Euler Angle [' char(176) ']'])
162     legend('\phi', '\theta', '\psi', 'Location', 'Southeast')
163     if savePlot
164         saveas(gcf,['Images/ps4_problem2a_' sprintf('%i',n) '.png'])
165     end
166 end
167
168 %% Momentum wheel setup
169 % Based on RSI 68
170 mr = 8.9; % kg
171 r = 0.347 / 2; % m

```

```

172 Ir = mr * r^2;
173 wrRPM = 2500; % RPM
174 wr = wrRPM * 0.1047198;
175
176 % Initial Euler angle
177 eulerAngle0 = [0; 0; 0];
178
179 % No external torques
180 M = [0; 0; 0; 0];
181
182 % Time
183 tFinal = 300;
184 tStep = 0.1;
185
186 %% Problem 3(b)
187 w0 = [0.01; 0.01; 0.01; wr];
188 r = [0; 0; 1];
189
190 namePlot = 'Images/ps4_problem3b.png';
191 plotPS4Problem3(eulerAngle0,w0,tStep,tFinal, ...
192 M,r,Ix,Iy,Iz,Ir,namePlot,savePlot);
193
194 %% Problem 3(c)
195 w0 = [0.1; 0.001; 0.001; wr + 0.001 * rand()];
196 r = [1; 0; 0];
197 namePlot = 'Images/ps4_problem3c_x.png';
198 plotPS4Problem3(eulerAngle0,w0,tStep,tFinal, ...
199 M,r,Ix,Iy,Iz,Ir,namePlot,savePlot);
200
201 w0 = [0.001; 0.1; 0.001; wr + 0.001 * rand()];
202 r = [0; 1; 0];
203 namePlot = 'Images/ps4_problem3c_y.png';
204 plotPS4Problem3(eulerAngle0,w0,tStep,tFinal, ...
205 M,r,Ix,Iy,Iz,Ir,namePlot,savePlot);
206
207 w0 = [0.001; 0.001; 0.1; wr + 0.001 * rand()];
208 r = [0; 0; 1];
209 namePlot = 'Images/ps4_problem3c_z.png';
210 plotPS4Problem3(eulerAngle0,w0,tStep,tFinal, ...
211 M,r,Ix,Iy,Iz,Ir,namePlot,savePlot);
212
213 %% Problem 3(d)
214 w0 = [0.001; 0.1; 0.001; wr * 10];
215 r = [0; 1; 0];
216
217 namePlot = 'Images/ps4_problem3d.png';
218 plotPS4Problem3(eulerAngle0,w0,tStep,tFinal, ...
219 M,r,Ix,Iy,Iz,Ir,namePlot,savePlot);
220
221 %% Problem 3(e)
222 w0 = [rot' * [0.1; 0.001; 0.001]; 0];
223 r = rot' * [1; 0; 0];
224
225 namePlot = 'Images/ps4_problem3e_unstable.png';
226 plotPS4Problem3(eulerAngle0,w0,tStep,tFinal, ...
227 M,r,Ix,Iy,Iz,Ir,namePlot,savePlot);
228
229 w0 = [rot' * [0.1; 0.001; 0.001]; wr * 10];

```

```

230
231 namePlot = 'Images/ps4_problem3e_stable.png';
232 plotPS4Problem3(eulerAngle0,w0,tStep,tFinal, ...
233 M,r,Ix,Iy,Iz,Ir,namePlot,savePlot);
234
235 %% Problem 4(d)
236 % Should put this into a function and call it for (d-e)
237 tFinal = 6000;
238 tStep = 1;
239 tspan = 0:tStep:tFinal;
240
241 a = 7125.48662; % km
242 e = 0;
243 i = 98.40508; % degree
244 O = -19.61601; % degree
245 w = 89.99764; % degree
246 nu = -89.99818; % degree
247 muE = 3.986 * 10^5;
248 n = sqrt(muE / a^3);
249
250 y = oe2eci(a,e,i,O,w,nu);
251 r0 = y(1:3);
252 v0 = y(4:6);
253 h = cross(r0,v0);
254 radial = r0 / norm(r0);
255 normal = h / norm(h);
256 tangential = cross(normal,radial);
257 A RTN = [radial tangential normal]';
258
259 state0 = zeros(12,1);
260 state0(1:6) = y;
261 state0(7:9) = [0; 0; n];
262 state0(10:12) = A2e(A RTN);
263
264 options = odeset('RelTol',1e-6,'AbsTol',1e-9);
265 [t,state] = ode113(@(t,state) gravGrad(t,state,Ix,Iy,Iz,n), ...
266 tspan,state0,options);
267
268 c = zeros(size(state(:,1:3)));
269 M = zeros(size(state(:,1:3)));
270 for i = 1:length(t)
271 r = state(i,1:3);
272 radial = r / norm(r);
273 A_ECI2P = e2A(state(i,10:12));
274 c(i,1:3) = A_ECI2P * radial';
275 M(i,1:3) = gravGradTorque(Ix,Iy,Iz,n,c(i,1:3));
276 end
277
278 figure()
279 plot(t,M)
280 xlabel('Time [s]')
281 ylabel('Torque in Principal Axes [Nm]')
282 legend('M_{x}', 'M_{y}', 'M_{z}')
283 ylim([-1e-5 1e-5])
284 if savePlot == true
285 saveas(gcf, 'Images/ps4_problem4d_torque.png')
286 end
287

```

```

288 figure()
289 plot(t,state(:,7:9))
290 xlabel('Time [s]')
291 ylabel('Angular Velocity in Principal Axes [rad/s]')
292 legend('\omega_{x}', '\omega_{y}', '\omega_{z}')
293 if savePlot == true
294     saveas(gcf, 'Images/ps4_problem4d_angvel.png')
295 end
296
297 %% Problem 4(e)
298 tFinal = 6000;
299 tStep = 1;
300 tspan = 0:tStep:tFinal;
301
302 a = 7125.48662; % km
303 e = 0;
304 i = 98.40508; % degree
305 O = -19.61601; % degree
306 w = 89.99764; % degree
307 nu = -89.99818; % degree
308 muE = 3.986 * 10^5;
309 n = sqrt(muE / a^3);
310
311 y = oe2eci(a,e,i,O,w,nu);
312 r0 = y(1:3);
313 v0 = y(4:6);
314 h = cross(r0,v0);
315 radial = r0 / norm(r0);
316 normal = h / norm(h);
317 tangential = cross(normal,radial);
318 A_RTN = [radial tangential normal]';
319 A_Body = rot' * A_RTN;
320
321 state0 = zeros(12,1);
322 state0(1:6) = y;
323 state0(7:9) = [0; 0; n];
324 state0(10:12) = A2e(A_Body);
325
326 options = odeset('RelTol',1e-6,'AbsTol',1e-9);
327 [t,state] = ode113(@(t,state) gravGrad(t,state,Ix,Iy,Iz,n), ...
328     tspan,state0,options);
329
330 c = zeros(size(state(:,1:3)));
331 M = zeros(size(state(:,1:3)));
332 for i = 1:length(t)
333     r = state(i,1:3);
334     radial = r / norm(r);
335     A_ECI2P = e2A(state(i,10:12));
336     c(i,1:3) = A_ECI2P * radial';
337     M(i,1:3) = gravGradTorque(Ix,Iy,Iz,n,c(i,1:3));
338 end
339
340 figure()
341 plot(t,M)
342 xlabel('Time [s]')
343 ylabel('Torque in Principal Axes [Nm]')
344 legend('M_{x}', 'M_{y}', 'M_{z}')
345 if savePlot == true

```

```

346     saveas(gcf, 'Images/ps4_problem4e_torque.png')
347 end
348
349 figure()
350 plot(t,state(:,7:9))
351 xlabel('Time [s]')
352 ylabel('Angular Velocity in Principal Axes [rad/s]')
353 legend('\omega_x', '\omega_y', '\omega_z')
354 if savePlot == true
355     saveas(gcf, 'Images/ps4_problem4e_angvel.png')
356 end
357
358 figure()
359 plot(t,wrapTo180(rad2deg(state(:,10:12))))
360 xlabel('Time [s]')
361 ylabel(['Euler Angles [ ' char(176) ' ]'])
362 legend('\phi', '\theta', '\psi')
363 if savePlot == true
364     saveas(gcf, 'Images/ps4_problem4e_angle.png')
365 end

```

## A.5 Problem Set 5

```

1 close all; clear; clc
2 savePlot = false;
3
4 %% Import mass properties
5 cm = computeCM('res/mass.csv');
6 I = computeMOI('res/mass.csv',cm);
7
8 [rot,IPrincipal] = eig(I);
9 Ix = IPrincipal(1,1);
10 Iy = IPrincipal(2,2);
11 Iz = IPrincipal(3,3);
12
13 %% Problem 1(a)
14 IR = Ix;
15 IT = Iz;
16 IN = Iy;
17
18 kT = (IN - IR) / IT;
19 kR = (IN - IT) / IR;
20
21 plotGravGradStability(kR,kT,'Nominal','Images/ps5_problema.png');
22
23 %% Problem 1(b) (Unstable, Unperturbed)
24 tFinal = 6000 * 5; % 5 orbits
25 tStep = 1;
26 tspan = 0:tStep:tFinal;
27
28 a = 7125.48662; % km
29 e = 0;
30 i = 98.40508; % degree
31 O = -19.61601; % degree
32 w = 89.99764; % degree
33 nu = -89.99818; % degree
34 muE = 3.986 * 10^5;
35 n = sqrt(muE / a^3);
36
37 y = oe2eci(a,e,i,O,w,nu);
38 r0 = y(1:3);
39 v0 = y(4:6);
40 h = cross(r0,v0);
41 radial = r0 / norm(r0);
42 normal = h / norm(h);
43 tangential = cross(normal,radial);
44 A_Nominal = [-radial -normal -tangential]';
45
46 state0 = zeros(12,1);
47 state0(1:6) = y;
48 state0(7:9) = [0; -n; 0];
49 state0(10:12) = A2e(A_Nominal);
50
51 options = odeset('RelTol',1e-6,'AbsTol',1e-9);
52 [t,state] = ode113(@(t,state) gravGrad(t,state,Ix,Iy,Iz,n), ...
53 tspan,state0,options);
54 c = zeros(size(state(:,1:3)));

```

```

56 M = zeros(size(state(:,1:3)));
57 for i = 1:length(t)
58     r = state(i,1:3);
59     radial = r / norm(r);
60     A_ECI2P = e2A(state(i,10:12));
61     c(i,1:3) = A_ECI2P * radial';
62     M(i,1:3) = gravGradTorque(Ix,Iy,Iz,n,c(i,1:3));
63 end
64
65 figure()
66 plot(t / 3600,state(:,7:9))
67 xlabel('Time [h]')
68 ylabel('Angular Velocity in Principal Axes [rad/s]')
69 legend('\omega_x','\omega_y','\omega_z')
70 if savePlot == true
71     saveas(gcf,'Images/ps5_problemlb_angvel_unperturbed.png')
72 end
73
74 figure()
75 plot(t / 3600,wrapToPi(state(:,10:12)))
76 xlabel('Time [h]')
77 ylabel('Euler Angles in Principal Axes [rad]')
78 legend('\phi','\theta','\psi')
79 if savePlot == true
80     saveas(gcf,'Images/ps5_problemlb_angle_unperturbed.png')
81 end
82
83 %% Problem 1(b) (Unstable, Perturbed)
84 tFinal = 6000 * 3; % 2 orbits
85 tStep = 1;
86 tspan = 0:tStep:tFinal;
87
88 a = 7125.48662; % km
89 e = 0;
90 i = 98.40508; % degree
91 O = -19.61601; % degree
92 w = 89.99764; % degree
93 nu = -89.99818; % degree
94 muE = 3.986 * 10^5;
95 n = sqrt(muE / a^3);
96
97 y = oe2eci(a,e,i,O,w,nu);
98 r0 = y(1:3);
99 v0 = y(4:6);
100 h = cross(r0,v0);
101 radial = r0 / norm(r0);
102 normal = h / norm(h);
103 tangential = cross(normal,radial);
104 A_Nominal = [-radial -normal -tangential]';
105
106 state0 = zeros(12,1);
107 state0(1:6) = y;
108 state0(7:9) = [0; -n; 0] * 1.01;
109 state0(10:12) = A2e(A_Nominal) + pi * [0.01; 0.01; 0.01];
110
111 options = odeset('RelTol',1e-6,'AbsTol',1e-9);
112 [t,state] = ode113(@(t,state) gravGrad(t,state,Ix,Iy,Iz,n), ...
113     tspan,state0,options);

```

```

114
115 c = zeros(size(state(:,1:3)));
116 M = zeros(size(state(:,1:3)));
117 for i = 1:length(t)
118     r = state(i,1:3);
119     radial = r / norm(r);
120     A_ECI2P = e2A(state(i,10:12));
121     c(i,1:3) = A_ECI2P * radial';
122     M(i,1:3) = gravGradTorque(Ix,Iy,Iz,n,c(i,1:3));
123 end
124
125 figure()
126 plot(t / 3600,state(:,7:9))
127 xlabel('Time [h]')
128 ylabel('Angular Velocity in Principal Axes [rad/s]')
129 legend('\omega_x','\omega_y','\omega_z')
130 if savePlot == true
131     saveas(gcf,'Images/ps5_problemlb_angvel.png')
132 end
133
134 figure()
135 plot(t / 3600,wrapToPi(state(:,10:12)))
136 xlabel('Time [h]')
137 ylabel('Euler Angles in Principal Axes [rad]')
138 legend('\phi','\theta','\psi')
139 if savePlot == true
140     saveas(gcf,'Images/ps5_problemlb_angle.png')
141 end
142
143 %% Problem 1(c)
144 IR = Ix;
145 IT = Iy;
146 IN = Iz;
147 kT = (IN - IR) / IT;
148 kR = (IN - IT) / IR;
149
150 plotGravGradStability(kR,kT, ...
151     'Principal XYZ aligned with RTN', ...
152     'Images/ps5_problemlc.png');
153
154 %% Problem 1(c) (Stable, Perturbed)
155 tFinal = 6000 * 10; % 10 orbits
156 tStep = 1;
157 tspan = 0:tStep:tFinal;
158
159 a = 7125.48662; % km
160 e = 0;
161 i = 98.40508; % degree
162 O = -19.61601; % degree
163 w = 89.99764; % degree
164 nu = -89.99818; % degree
165 muE = 3.986 * 10^5;
166 n = sqrt(muE / a^3);
167
168 y = oe2eci(a,e,i,O,w,nu);
169 r0 = y(1:3);
170 v0 = y(4:6);
171 h = cross(r0,v0);

```

```

172 radial = r0 / norm(r0);
173 normal = h / norm(h);
174 tangential = cross(normal,radial);
175 A_RTN = [radial tangential normal]';
176
177 state0 = zeros(12,1);
178 state0(1:6) = y;
179 state0(7:9) = [0; 0; n] * 1.01;
180 state0(10:12) = A2e(A_RTN) + pi * [0.01; 0.01; 0.01];
181
182 options = odeset('RelTol',1e-6,'AbsTol',1e-9);
183 [t,state] = ode13(@(t,state) gravGrad(t,state,Ix,Iy,Iz,n), ...
184 tspan,state0,options);
185
186 c = zeros(size(state(:,1:3)));
187 M = zeros(size(state(:,1:3)));
188 for i = 1:length(t)
189     r = state(i,1:3);
190     radial = r / norm(r);
191     A_ECI2P = e2A(state(i,10:12));
192     c(i,1:3) = A_ECI2P * radial';
193     M(i,1:3) = gravGradTorque(Ix,Iy,Iz,n,c(i,1:3));
194 end
195
196 figure()
197 plot(t / 3600,state(:,7:9))
198 xlabel('Time [h]')
199 ylabel('Angular Velocity in Principal Axes [rad/s]')
200 legend('\omega_x','\omega_y','\omega_z')
201 if savePlot == true
202     saveas(gcf,'Images/ps5_problem1c_angvel.png')
203 end
204
205 figure()
206 plot(t / 3600,wrapToPi(state(:,10:12)))
207 xlabel('Time [h]')
208 ylabel('Euler Angles in Principal Axes [rad]')
209 legend('\phi','\theta','\psi')
210 if savePlot == true
211     saveas(gcf,'Images/ps5_problem1c_angle.png')
212 end
213
214 %% Problem 3
215 tFinal = 6000;
216 tStep = 1;
217 tspan = 0:tStep:tFinal;
218
219 % Satellite orbit initial conditions
220 a = 7125.48662; % km
221 e = 0;
222 i = 98.40508; % degree
223 O = -19.61601; % degree
224 w = 89.99764; % degree
225 nu = -89.99818; % degree
226 muE = 3.986 * 10^5; % km^3 / s^2
227 n = sqrt(muE / a^3);
228
229 % Compute initial position and attitude

```

```

230 y = oe2eci(a,e,i,O,w,nu);
231 r0 = y(1:3);
232 v0 = y(4:6);
233 h = cross(r0,v0);
234 radial = r0 / norm(r0);
235 normal = h / norm(h);
236 tangential = cross(normal,radial);
237 A_RTN = [radial tangential normal]';
238
239 % Earth orbit initial conditions
240 aE = 149.60E6; % km
241 eE = 0.0167086;
242 iE = 7.155; % degree
243 OE = 174.9; % degree
244 wE = 288.1; % degree
245 nuE = 0;
246 muSun = 1.327E11; % km^3 / s^2
247 nE = sqrt(muSun / aE^3);
248 ySun = oe2eci(aE,eE,iE,OE,wE,nuE);
249
250 % Initial conditions
251 state0 = zeros(12,1);
252 state0(1:6) = y;
253 state0(7:9) = [0; 0; n];
254 state0(10:12) = A2e(A_RTN);
255 state0(13:18) = ySun;
256
257 % Properties
258 [barycenter,normal,area] = surfaces('res/area.csv',rot');
259 cm = computeCM('res/mass.csv');
260 I = computeMOI('res/mass.csv',cm);
261 [rot,~] = eig(I);
262 cmP = rot' * cm;
263
264 % Parameters
265 CD = 2;
266 Cd = 0; Cs = 0.9;
267 P = 1358 / 3e8;
268 S_sat = 24.92;
269 m_max = 4e-7 * pi * S_sat * 0.1;
270 m_direction.body = [1; 0; 0];
271 m_direction = rot * m_direction.body;
272 m = m_max * m_direction / norm(m_direction); % Arbitrary sat dipole
273 UT1 = [2024 1 1];
274
275 % Run numerical method
276 options = odeset('RelTol',1e-6,'AbsTol',1e-9);
277 [t,state] = ode113(@(t,state) orbitTorque(t,state,Ix,Iy,Iz, ...
278     CD,Cd,Cs,P,m,UT1, ...
279     barycenter,normal,area,cmP,n), ...
280     tspan,state0,options);
281
282 % Compute torques (since ode113 does not allow returning these)
283 Rx = [1 0 0; 0 cosd(23.5) -sind(23.5); 0 sind(23.5) cosd(23.5)];
284 c = zeros(size(state(:,1:3)));
285 Mgg = zeros(size(state(:,1:3)));
286 Md = zeros(size(state(:,1:3)));
287 Msrp = zeros(size(state(:,1:3)));

```

```

288 Mm = zeros(size(state(:,1:3)));
289
290 for i = 1:length(t)
291     r = state(i,1:3)';
292     v = state(i,1:3)';
293     radial = r / norm(r);
294     rEarth = state(i,13:15)';
295     A_ECI2P = e2A(state(i,10:12));
296
297     c(i,1:3) = A_ECI2P * radial;
298     Mgg(i,1:3) = gravGradTorque(Ix,Iy,Iz,n,c(i,1:3));
299
300     [~,density] = atmosnrlmsise00(1000 * (norm(r) - ...
301         6378.1),0,0,2000,1,0);
302     rho = density(6);
303     vPrincipal = A_ECI2P * (v + cross([0; 0; 7.2921159E-5],r));
304     [~,M] = drag(vPrincipal,rho,CD,barycenter,normal,area,cmP);
305     Md(i,1:3) = M;
306
307     s = A_ECI2P * (-Rx * rEarth - r);
308     [~,M] = srp(s,P,Cd,Cs,barycenter,normal,area,cmP);
309     Msrp(i,1:3) = M;
310
311     M = magFieldTorque(m,r,state(i,10:12),t(i),6378.1,UT1);
312     Mm(i,1:3) = M;
313 end
314
315 figure()
316 plot(t / 3600,state(:,7:9))
317 xlabel('Time [h]')
318 ylabel('Angular Velocity in Principal Axes [rad/s]')
319 legend('\omega_x','\omega_y','\omega_z')
320 if savePlot == true
321     saveas(gcf,'Images/ps5_problem3_angvel.png')
322 end
323
324 figure()
325 plot(t / 3600,Mgg)
326 xlabel('Time [h]')
327 ylabel('Gravity Gradient Torque in Principal Axes [Nm]')
328 legend('M_x','M_y','M_z')
329 if savePlot == true
330     saveas(gcf,'Images/ps5_problem3_grav.png')
331 end
332
333 figure()
334 plot(t / 3600,Md)
335 xlabel('Time [h]')
336 ylabel('Drag Torque in Principal Axes [Nm]')
337 legend('M_x','M_y','M_z')
338 if savePlot == true
339     saveas(gcf,'Images/ps5_problem3_drag.png')
340 end
341
342 figure()
343 plot(t / 3600,Msrp)
344 xlabel('Time [h]')
345 ylabel('Solar Radiation Pressure Torque in Principal Axes [Nm]')

```

```

345 legend('M_{x}', 'M_{y}', 'M_{z}')
346 if savePlot == true
347     saveas(gcf, 'Images/ps5_problem3_srp.png')
348 end
349
350 figure()
351 plot(t / 3600, Mm)
352 xlabel('Time [h]')
353 ylabel('Magnetic Field Torque in Principal Axes [Nm]')
354 legend('M_{x}', 'M_{y}', 'M_{z}')
355 if savePlot == true
356     saveas(gcf, 'Images/ps5_problem3_mag.png')
357 end
358
359 %%
360 figure()
361 plot(t, state(:,10:12))
362
363 %% Problem 3 Maximum Torques
364 % Parameters
365 CD = 2;
366 Cd = 0; Cs = 0.9;
367 q = Cd + Cs;
368 P = 1358 / 3E8;
369 S_sat = 24.92;
370 m_max = 4e-7 * pi * S_sat * 0.1;
371 UT1 = [2024 1 1];
372 rE3_B0 = 7.943e15; % Wb-km
373
374 r_norm = norm(state(1,1:3));
375 Mgg_max = 3 / 2 * muE / (r_norm^3) * abs(max([Ix Iy Iz]) - min([Ix ...
    Iy Iz]));
376 Mm_max = 2 * m_max * rE3_B0 / ((r_norm * 1e3)^3);
377 Msrp_max = 0;
378 Md_max = 0;
379
380 vMax = max(vecnorm(state(:,4:6))) * 1e3;
381
382 for n = 1:length(area)
383     Msrp_max = Msrp_max + P * area(n) * (1+q) * norm(barycenter(:,n) ...
        - cmP);
384     Md_max = Md_max + 0.5 * rho * CD * vMax^2 * area(n) * ...
        norm(barycenter(:,n) - cmP);
385 end
386
387 fprintf("Maximum expected values: \n" +
388         "M_gg: %d Nm \n" +
389         "M_m: %d Nm \n" +
390         "M_srp: %d Nm \n" +
391         "M_d: %d Nm\n", ...
392         Mgg_max, Mm_max, Msrp_max, Md_max);

```

```

1 function [kR,kT] = plotGravGradStability(kR,kT,nameText,namePlot)
2     % Gather points
3     fimplicit(@(x,y) 1 + 3 * x + y * x + 4 * sqrt(y * x),[-1,1,-1,1])
4     leftBranch = findobj(gcf,'Type','ImplicitFunctionLine');

```

```

5      xLeft = leftBranch.XData;
6      yLeft = leftBranch.YData;
7      close()
8      fimplicit(@(x,y) 1 + 3 * x + y * x - 4 * sqrt(y * x), [-1,1,-1,1])
9      rightBranch = findobj(gcf, 'Type', 'ImplicitFunctionLine');
10     xRight = rightBranch.XData;
11     yRight = rightBranch.YData;
12     close()
13     hold on
14
15 % Plot yaw, roll unstable
16 plot(polyshape([0 0 1 1], [-1 0 0 -1]), ...
17      'FaceAlpha', 1, ...
18      'FaceColor', 'b', ...
19      'DisplayName', 'Unstable yaw, roll')
20 plot(polyshape([xRight(27:end) -1], [yRight(27:end) -1]), ...
21      'FaceAlpha', 1, ...
22      'FaceColor', 'b', ...
23      'HandleVisibility', 'off')
24
25 % Plot pitch unstable
26 plot(polyshape([0 1 0], [0 1 1]), ...
27      'FaceAlpha', 1, ...
28      'FaceColor', 'y', ...
29      'DisplayName', 'Unstable pitch')
30 plot(polyshape([xLeft -1], [yLeft 0]), ...
31      'FaceAlpha', 1, ...
32      'FaceColor', 'y', ...
33      'HandleVisibility', 'off')
34 plot(polyshape([xRight(1:27) 0], [yRight(1:27) 0]), ...
35      'FaceAlpha', 1, ...
36      'FaceColor', 'y', ...
37      'HandleVisibility', 'off')
38
39 % Plot yaw, roll, pitch unstable
40 plot(polyshape([-1 -1 0 0], [0 1 1 0]), ...
41      'FaceAlpha', 1, ...
42      'FaceColor', 'g', ...
43      'DisplayName', ...
44      'Unstable yaw, roll, pitch')
45 plot(polyshape([flip(xRight(1:27)) xLeft -1], ...
46      [flip(yRight(1:27)) yLeft -1]), ...
47      'FaceAlpha', 1, ...
48      'FaceColor', 'g', ...
49      'HandleVisibility', 'off')
50
51 % Plot spacecraft location
52 plot(kT,kR, 'x', 'Color', 'k', 'LineWidth', 2, 'DisplayName', nameText)
53
54 axis equal
55 legend()
56 xlabel('k_{T}')
57 ylabel('k_{R}')
58 xlim([-1 1])
59 ylim([-1 1])
60 hold off
61 saveas(gcf, namePlot)
62 end

```

```

1 function [B_R,B_theta,B_phi] = magFieldEarth(R,phi,theta,RE)
2     % NOTE: slides calls phi lambda (not sure if it's the same thing ...
3         % or not)
4
5     % Make sure that R is normalized
6     R = norm(R);
7
8     % For g & h matrix (row = n, col = m + 1)
9     g = [-30186 -2036 0 0 0; ...
10         -1898 2997 1551 0 0; ...
11         1299 -2144 1296 805 0; ...
12         951 807 462 -393 235] * 1e-9; % T
13     h = [0 5735 0 0 0; ...
14         0 -2124 -37 0 0; ...
15         0 -361 249 -253 0; ...
16         0 148 -264 37 -307] * 1e-9; % T
17
18     B_R = 0;
19     B_theta = 0;
20     B_phi = 0;
21     for n = 1:4
22         BR_temp = 0;
23         BTheta_temp = 0;
24         BPhi_temp = 0;
25
26         for mInd = 1:n
27             m = mInd - 1;
28
29             P_nm = getPnm(theta,n,m);
30             dPnm_dtheta = getdPdTheta(theta,n,m);
31
32             BR_temp = BR_temp + ...
33                 (g(n,mInd) * cos(m * phi) + h(n,mInd) * sin(m * ...
34                     phi)) * ...
35                     P_nm;
36             BTheta_temp = BTheta_temp + ...
37                 (g(n,mInd) * cos(m * phi) + h(n,mInd) * sin(m * phi)) ...
38                     * ...
39                     dPnm_dtheta;
40             BPhi_temp = BPhi_temp + ...
41                 (-g(n,mInd) * sin(m * phi) + h(n,mInd) * cos(m * ...
42                     phi)) * ...
43                     m * P_nm;
44
45         end
46
47         B_theta = -B_theta;
48         B_phi = -1 / sin(theta) * B_phi;
49
50     end

```

```

1 function dPdTheta = getdPdTheta(theta,n,m)

```

```

2     if n < m
3         error("n >= m for Legendre functions")
4     end
5     if n == m && m == 0
6         dPdTheta = 0;
7     elseif n == m
8         dPdTheta = sin(theta) * getdPdTheta(theta,n-1,n-1) + ...
9             cos(theta) * getPnm(theta,n-1,n-1);
10    else
11        K = getKnm(n,m);
12        if K == 0
13            dPdTheta = cos(theta) * getdPdTheta(theta,n-1,m) - ...
14                sin(theta) * getPnm(theta,n-1,m);
15        else
16            dPdTheta = cos(theta) * getdPdTheta(theta,n-1,m) - ...
17                sin(theta) * getPnm(theta,n-1,m) - ...
18                K * getdPdTheta(theta,n-2,m);
19        end
20    end
21 end

```

```

1 function K = getKnm(n,m)
2     if n == 1
3         K = 0;
4     else
5         K = ((n - 1)^2 - m^2)/((2 * n - 1)*(2 * n - 3));
6     end
7 end

```

```

1 function P = getPnm(theta,n,m)
2     if n == 0 && m == 0
3         P = 1;
4     elseif n == m
5         P = sin(theta) * getPnm(theta,n-1,n-1);
6     else
7         K = getKnm(n,m);
8         if K == 0
9             P = cos(theta) * getPnm(theta,n-1,m);
10        else
11            P = cos(theta) * getPnm(theta,n-1,m) - K * ...
12                getPnm(theta,n-2,m);
13        end
14    end
15 end

```

## A.6 Problem Set 6

```
1 close all; clear; clc
2
3 % From PS6 onward, we use Simulink to model the spacecraft
4 savePlots = true;
5
6 %% Import mass properties
7 cm = computeCM('res/mass.csv');
8 I = computeMOI('res/mass.csv',cm);
9
10 [rot,IPrincipal] = eig(I);
11 Ix = IPrincipal(1,1);
12 Iy = IPrincipal(2,2);
13 Iz = IPrincipal(3,3);
14
15 %% Problem 2-3
16 % Problem 2 assumes no disturbances. We edit orbitTorque to do this.
17 tFinal = 6000;
18 tStep = 1;
19 tspan = 0:tStep:tFinal;
20
21 % Satellite orbit initial conditions
22 a = 7125.48662; % km
23 e = 0;
24 i = 98.40508; % degree
25 O = -19.61601; % degree
26 w = 89.99764; % degree
27 nu = -89.99818; % degree
28 muE = 3.986e5; % km^3 / s^2
29 n = sqrt(muE / a^3);
30
31 % Compute initial position and attitude
32 y = oe2eci(a,e,i,O,w,nu);
33 r0 = y(1:3);
34 v0 = y(4:6);
35 h = cross(r0,v0);
36 radial = r0 / norm(r0);
37 normal = h / norm(h);
38 tangential = cross(normal,radial);
39 A_Nominal = [-radial -normal -tangential]';
40
41 % Earth orbit initial conditions
42 aE = 149.60E6; % km
43 eE = 0.0167086;
44 iE = 7.155; % degree
45 OE = 174.9; % degree
46 wE = 288.1; % degree
47 nuE = 0;
48 muSun = 1.327E11; % km^3 / s^2
49 nE = sqrt(muSun / aE^3);
50 ySun = oe2eci(aE,eE,iE,OE,wE,nuE);
51
52 % Initial conditions
53 state0 = zeros(12,1);
54 state0(1:6) = y;
55 state0(7:9) = [0; -n; 0];
```

```

56 state0(10:12) = A2e(A_Nominal);
57 state0(13:18) = ySun;
58
59 % Properties
60 [barycenter,normal,area] = surfaces('res/area.csv',rot');
61 cm = computeCM('res/mass.csv');
62 I = computeMOI('res/mass.csv',cm);
63 [rot,~] = eig(I);
64 cmP = rot' * cm;
65
66 % Parameters
67 CD = 2;
68 Cd = 0; Cs = 0.9;
69 P = 1358 / 3e8;
70 S_sat = 24.92;
71 m_max = 4e-7 * pi * S_sat * 0.1;
72 m_direction_body = [1; 0; 0];
73 m_direction = rot * m_direction_body;
74 m = m_max * m_direction / norm(m_direction); % Arbitrary sat dipole
75 UT1 = [2024 1 1];
76
77 % Run numerical method
78 options = odeset('RelTol',1e-6,'AbsTol',1e-9);
79 [t,state] = ode113(@(t,state) orbitTorque(t,state,Ix,Iy,Iz, ...
80     CD,Cd,Cs,P,m,UT1, ...
81     barycenter,normal,area,cmP,n), ...
82     tspan,state0,options);
83
84 eulerError = zeros(size(state(:,10:12)));
85 eulerTarget = zeros(size(state(:,10:12)));
86 A_Sats = zeros([3,3,length(t)]);
87 for i = 1:length(t)
88     r = state(i,1:3)';
89     v = state(i,4:6)';
90     h = cross(r,v);
91     radial = r / norm(r);
92     normal = h / norm(h);
93     tangential = cross(normal,radial);
94     A_Target = [-radial -normal -tangential]'; % ECI -> RTN
95     A_Sat = e2A(state(i,10:12)); % ECI -> Principal
96     A_Sats(:,:,i) = A_Sat;
97     eulerTarget(i,1:3) = A2e(A_Target);
98     eulerError(i,1:3) = A2e(A_Sat * A_Target');
99 end
100
101 figure()
102 plot(t / 3600,wrapToPi(state(:,10:12)))
103 xlabel('Time [h]')
104 ylabel('Euler Angle (Principal) [rad]')
105 legend('\phi','\theta','\psi')
106 saveAsBool(gcf,'Images/ps6_problem2_principal.png',savePlots)
107 % saveAsBool(gcf,'Images/ps6_problem3_principal.png',savePlots)
108
109 figure()
110 plot(t / 3600,wrapToPi(eulerTarget))
111 xlabel('Time [h]')
112 ylabel('Euler Angles (Target) [rad]')
113 legend('\phi','\theta','\psi')

```

```

114 saveAsBool(gcf, 'Images/ps6_problem2_target.png', savePlots)
115 % saveAsBool(gcf, 'Images/ps6_problem3_target.png', savePlots)
116
117 figure()
118 plot(t / 3600, wrapToPi(eulerError))
119 xlabel('Time [h]')
120 ylabel('Euler Angle Error [rad]')
121 legend('\phi', '\theta', '\psi')
122 saveAsBool(gcf, 'Images/ps6_problem2_error.png', savePlots)
123 % saveAsBool(gcf, 'Images/ps6_problem3_error.png', savePlots)
124
125 %% Problem 5
126 numReadings = 5;
127
128 v = rand([3, numReadings]);
129 v = v ./ vecnorm(v);
130 w = [100 100 ones([1, numReadings-2])];
131
132 % m = A_Nominal * v;
133
134 eulerReals = state(:,10:12);
135 euler_DAD2 = nan([3, length(t)]);
136 euler_DAD = nan([3, length(t)]);
137 euler_qMethod = nan([3, length(t)]);
138 euler_kins = nan([3, length(t)]);
139
140 euler_kins(:,1) = state0(10:12);
141 omega = [0; -n; 0];
142
143 for n = 1:length(t)
144     m = A_Sats(:,:,n) * v;
145     m1 = m(:,1);
146     m2 = m(:,2);
147     v1 = v(:,1);
148     v2 = v(:,2);
149     A_DAD2 = DAD_twoVecs(m1, m2, v1, v2);
150     euler_DAD2(:,n) = A2e(A_DAD2);
151     A_DAD = DAD(m, v);
152     euler_DAD(:,n) = A2e(A_DAD);
153     qMeas_qMethod = qMethod(m, v, w);
154     euler_qMethod(:,n) = A2e(q2A(qMeas_qMethod));
155
156     % Propogate stateKin
157     [euler_kins(:,n+1), omega] = kinEulerStepRK4(euler_kins(:,1), ...
158         omega, Ix, Iy, Iz, tStep);
158 end
159
160 figure()
161 plot(t/3600, wrapToPi(eulerReals))
162 title("Actual Attitude")
163 xlabel('Time [h]')
164 ylabel('Euler Angle (Principal) [rad]')
165 legend('\phi', '\theta', '\psi')
166 saveAsBool(gcf, 'Images/ps6_problem6_actual.png', savePlots)
167
168
169 figure()
170 plot(t/3600, euler_DAD2)

```

```

171 title("Deterministic Attitude Determination (Fictitious Measurements)")
172 xlabel('Time [h]')
173 ylabel('Euler Angle (Principal) [rad]')
174 legend('\phi','\theta','\psi')
175 saveAsBool(gcf,'Images/ps6_problem6_DADFict.png',savePlots)
176
177 figure()
178 plot(t/3600, euler_DAD)
179 title("Deterministic Attitude Determination")
180 xlabel('Time [h]')
181 ylabel('Euler Angle (Principal) [rad]')
182 legend('\phi','\theta','\psi')
183 saveAsBool(gcf,'Images/ps6_problem6_DAD.png',savePlots)
184
185 figure()
186 plot(t/3600, euler_qMethod)
187 title("q Method")
188 xlabel('Time [h]')
189 ylabel('Euler Angle (Principal) [rad]')
190 legend('\phi','\theta','\psi')
191 saveAsBool(gcf,'Images/ps6_problem6_qMethod.png',savePlots)

```