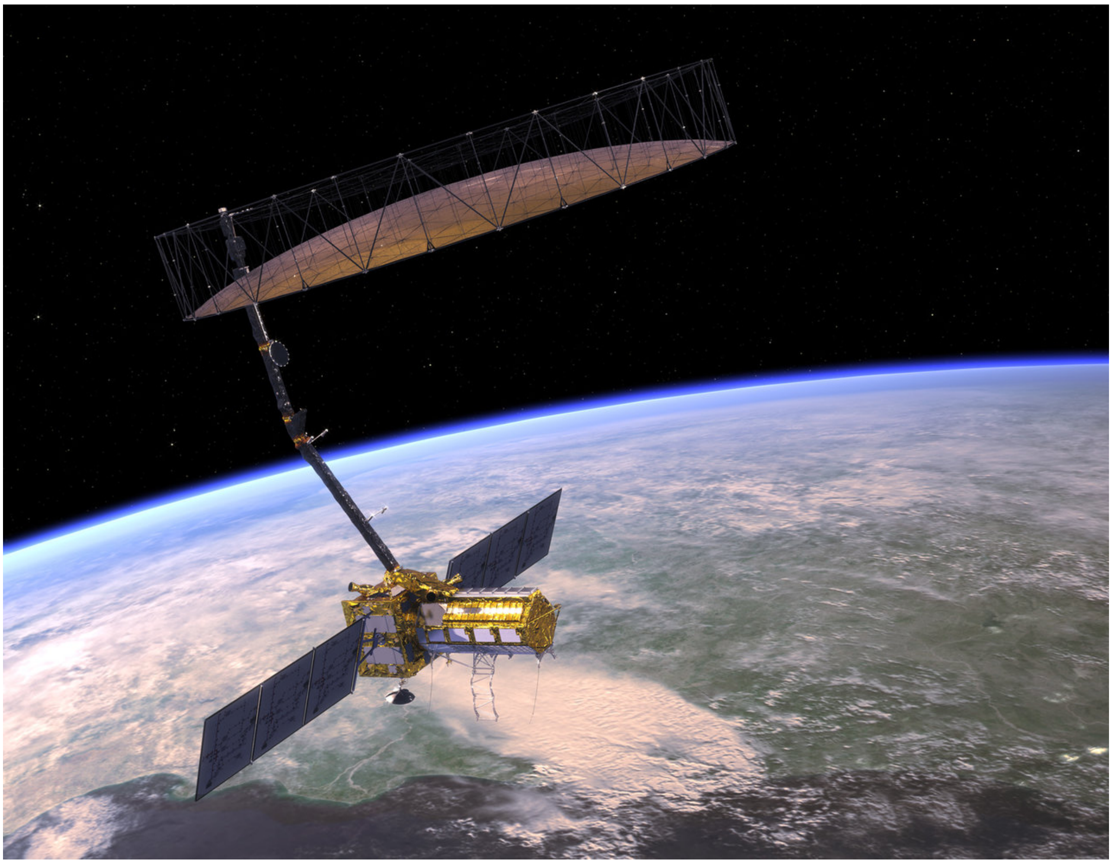
SATELLITE DYNAMICS AND ATTITUDE CONTROL

Kusal Uprety, Harry Zhao

8 May 2024



REVISION HISTORY

VERSION	REVISION NOTES
PS1	- Created document - Added PS1 material
PS2	- Added PS2 material - Updated title page - Updated moment of inertia for center of mass
PS3	- Added PS3 material - Updated title page
PS4	- Added PS4 material - Updated title page - Switched to 312 Euler angles
PS5	- Added PS5 material - Updated title page - Fix PS2 figure caption typos - Fix PS4 figure numbering

Table 1: Summary of project revisions.

TABLE OF CONTENTS

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, incorportating 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 that 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 terrestial 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.

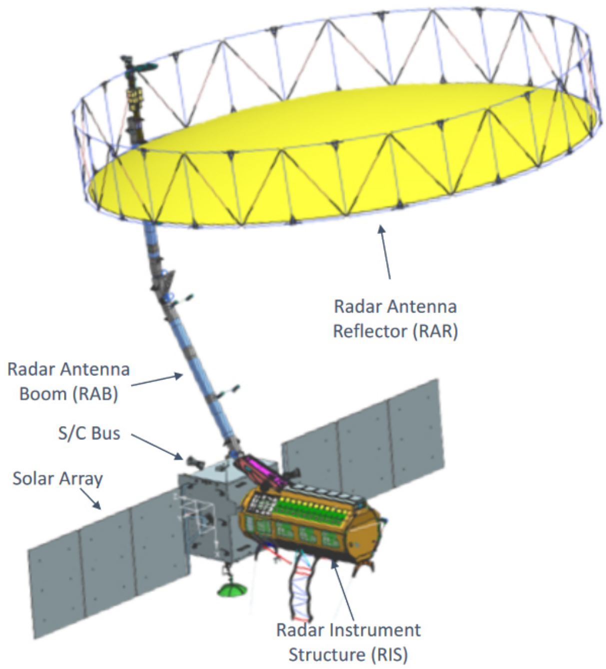


Figure 1: The basic components of the NISAR satellite

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^2 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^2 and a typical solar panel mass per area is 2.06 kg/m^2 [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^2 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° about the y-axis (relative to the x-axis), while the RAB is modeled as a single beam with an angle approximately -18° 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.

```
function cm = computeCM(filename)
    data = readmatrix(filename);
    x = data(:,1);
    y = data(:,2);
    z = data(:,3);
    m = data(:,4);
    cm = [dot(x,m); ...
        dot(y,m); ...
        dot(z,m)] / sum(m);
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:

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

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

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

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.

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

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° from vertical (z-axis).

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

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.

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

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.

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

Applying a rotation:

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

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:

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

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

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

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.

```matlab
function I = computeMOI(filename,reference)
    data = readmatrix(filename);
    x = data(:,1) - reference(1);
    y = data(:,2) - reference(2);
    z = data(:,3) - reference(3);
    m = data(:,4);
    m_bus = m(1);
    m_RIS = m(2);
    m_panel = m(3);
    m_RAB = m(5);
    m_RAR = m(6);

    L_bus = 1.2;
    W_bus = 1.8;
    H_bus = 1.9;
    I_bus = m_bus * [(W_bus^2 + H_bus^2) / 12, 0, 0; ...
        0, (L_bus^2 + H_bus^2) / 12, 0; ...
        0, 0, (L_bus^2 + W_bus^2) / 12];

    L_RIS = 2.5;
    S_RIS = 0.459;
    a_RIS = S_RIS / (2 * tan(deg2rad(22.5)));
    R_avg = mean([a_RIS sqrt(a_RIS^2 + (S_RIS / 2)^2)]);
    I_RIS = m_RIS * [S_RIS^2 / 24 + a_RIS^2 / 2, 0, 0; ...
        0, L_RIS^2 / 12 + R_avg^2 / 4, 0; ...
        0, 0, L_RIS^2 / 12 + R_avg^2 / 4];

    W_panel = 6;
    H_panel = 1.9;
    I_panel = m_panel * [(W_panel^2 + H_panel^2) / 12, 0, 0; ...
        0, H_panel^2 / 12, 0; ...
        0, 0, W_panel^2 / 12];

    L_RAB = 0.1778;
    W_RAB = 0.1778;
    H_RAB = 9;
    deg_RAB = -18;
    rot_RAB = [cosd(deg_RAB), 0, sind(deg_RAB); ...
        0, 1, 0; ...
        -sind(deg_RAB), 0, cosd(deg_RAB)];
    I_RAB = m_RAB * [(W_RAB^2 + H_RAB^2) / 12, 0, 0; ...
        0, (L_RAB^2 + H_RAB^2) / 12, 0; ...
        0, 0, (L_RAB^2 + W_RAB^2) / 12];
    I_RAB_rot = rot_RAB * I_RAB * rot_RAB';

    R_RAR = 6;
    deg_RAR = -3.87;
    rot_RAR = [cosd(deg_RAR), 0, sind(deg_RAR); ...
        0, 1, 0; ...
        -sind(deg_RAR), 0, cosd(deg_RAR)];
    I_RAR = m_RAR * [R_RAR^2 / 4, 0, 0; ...
        0, R_RAR^2 / 4, 0; ...
        0, 0, R_RAR^2 / 2];
    I_RAR_rot = rot_RAR * I_RAR * rot_RAR';
```

```

    I_c = {I_bus, I_RIS, I_panel, I_panel, I_RAB_rot, I_RAR_rot};
    I = zeros([3 3]);
    for i = 1:length(I_c)
        D = [y(i)^2 + z(i)^2, -x(i) * y(i), -x(i) * z(i); ...
            -y(i) * x(i), x(i)^2 + z(i)^2, -y(i) * z(i); ...
            -z(i) * x(i), -y(i) * z(i), x(i)^2 + y(i)^2];
        I = I + I_c{i} + m(i) * D;
    end
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:

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

	Barycenter [m]			**Normal**			
Surface	**x**	**y**	**z**	**x**	**y**	**z**	**Area [m^2]**
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.

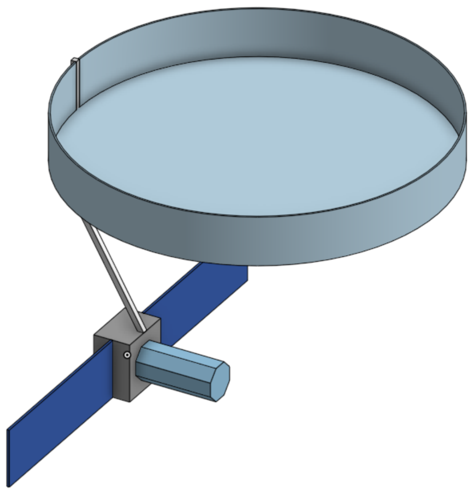


Figure 2: A 3D model of the satellite

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.

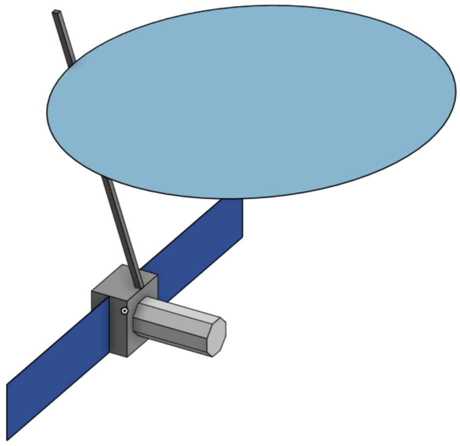


Figure 3: A 3D model of the simplified satellite geometry

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.

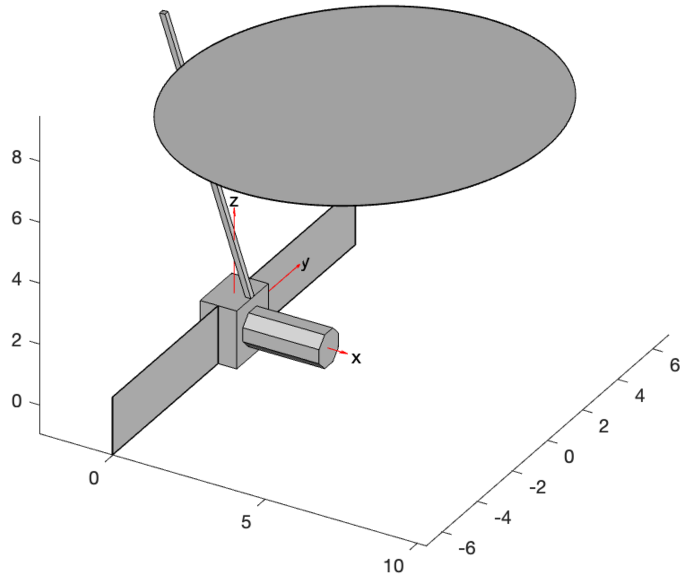


Figure 4: Satellite model in MATLAB with body axes shown

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	Ω	ω	ν
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 ν into perifocal (PQW) coordinates, using a and e to find the semi-latus rectum and a, e, and ν to find the distance to the central body (Earth). Then, we perform a series of rotations on these coordinates parameterized by ω, i, and Ω to obtain new coordinates in the ECI frame.

```
function yECI = oe2eci(a,e,i,O,w,nu)
    i = deg2rad(i);
    O = deg2rad(O);
    w = deg2rad(w);
    nu = deg2rad(nu);
    p = a * (1 - e^2);
    r = p / (1 + e * cos(nu));
    rPQW = [r * cos(nu); r * sin(nu); 0];
    vPQW = sqrt(3.986 * 10^5 / p) * [-sin(nu); e + cos(nu); 0];
    Rzw = [cos(-w), sin(-w), 0;...
        -sin(-w), cos(-w), 0;...
        0, 0, 1];
    Rxi = [1, 0, 0;...
        0, cos(-i), sin(-i);...
        0, -sin(-i), cos(-i)];
    RzO = [cos(-O), sin(-O), 0;...
        -sin(-O), cos(-O), 0;...
        0, 0, 1];
    rECI = RzO * Rxi * Rzw * rPQW;
    vECI = RzO * Rxi * Rzw * vPQW;
    yECI = [rECI; vECI];
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`.

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

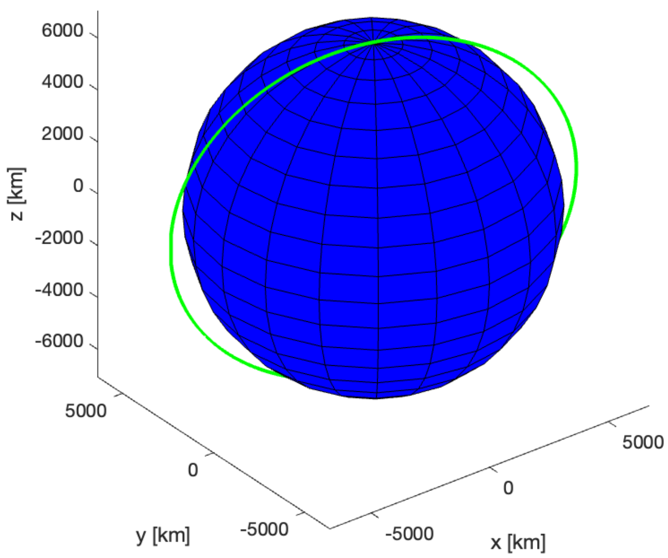


Figure 5: A single orbit for NISAR in ECI coordinates (no perturbations)

2.2 PROBLEM 2

In general the body axes are not the principal axes. Identify principal axes through the eigenvector/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.

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

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

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.

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

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

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.

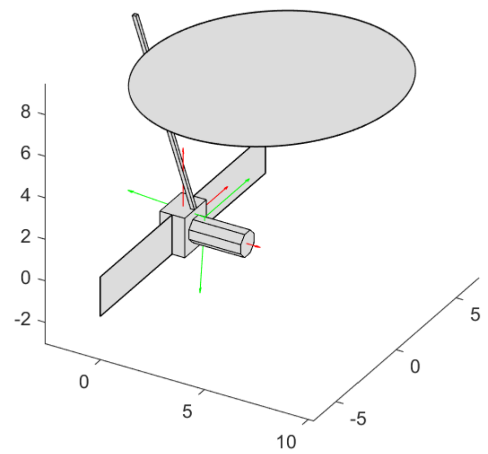


Figure 6: Principal axes at center of mass (green) and body axes at origin (red)

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 ($M_x, M_y, M_z = 0$).

$$
\begin{aligned}
I_x\dot{\omega}_x + (I_z - I_y)\omega_y\omega_z &= M_x \\
I_y\dot{\omega}_y + (I_x - I_z)\omega_z\omega_x &= M_y \\
I_z\dot{\omega}_z + (I_y - I_x)\omega_x\omega_y &= M_z
\end{aligned}
$$

```matlab
function [wDot] = eulerEquation(t,w,Ix,Iy,Iz)
    wx = w(1);
    wy = w(2);
    wz = w(3);
    wDot = zeros(3,1);
    wDot(1) = (Iy - Iz) / Ix * wy * wz;
    wDot(2) = (Iz - Ix) / Iy * wz * wx;
    wDot(3) = (Ix - Iy) / Iz * wx * wy;
end
```

2.5 PROBLEM 5

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

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

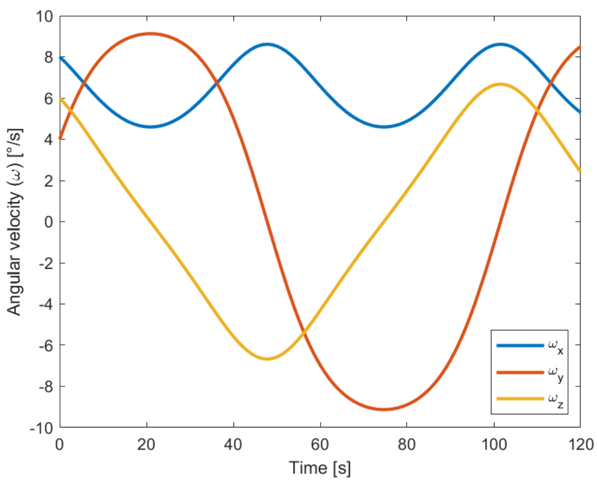


Figure 7: Results from numerical integration of Euler equations

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: $\omega_x = 0.2332\,\mathrm{rad\,s^{-1}}$, $\omega_y = 0.1697\,\mathrm{rad\,s^{-1}}$, and $\omega_z = 0.1524\,\mathrm{rad\,s^{-1}}$. 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: $\omega_x = 0.3115\,\mathrm{rad\,s^{-1}}$, $\omega_y = 0.1649\,\mathrm{rad\,s^{-1}}$, and $\omega_z = 0.1330\,\mathrm{rad\,s^{-1}}$. 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.

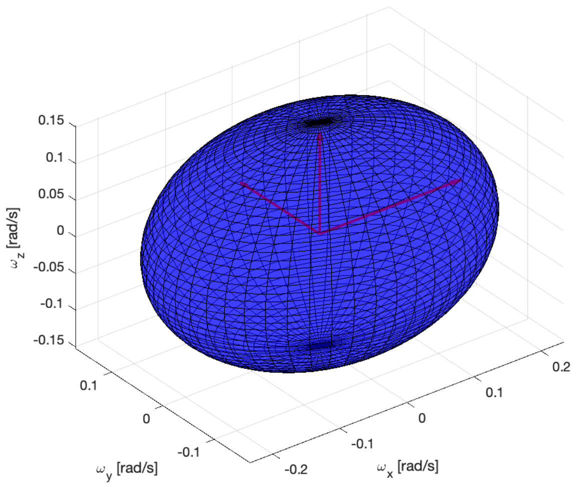


Figure 8: Energy ellipsoid with axes in red

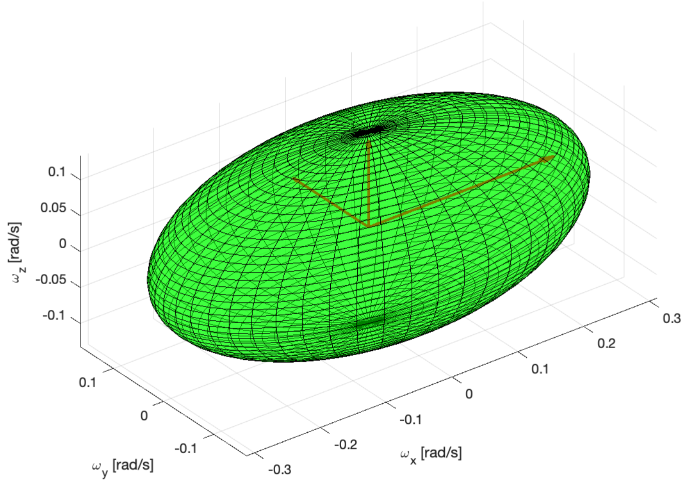


Figure 9: Momentum ellipsoid with axes in red

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.

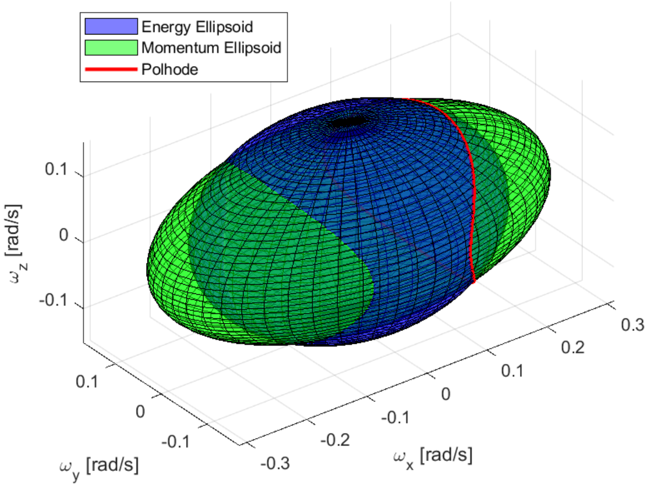


Figure 10: Energy and momentum ellipsoids with polhode

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.

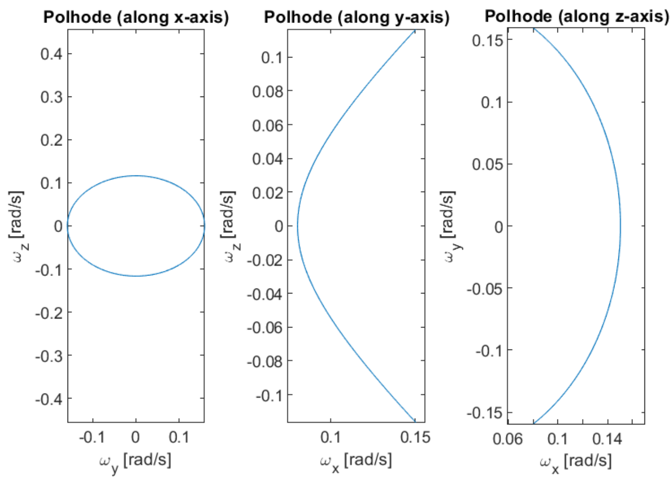


Figure 11: 2D views of polhode

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	ω_x **(deg/s)**	ω_y **(deg/s)**	ω_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.

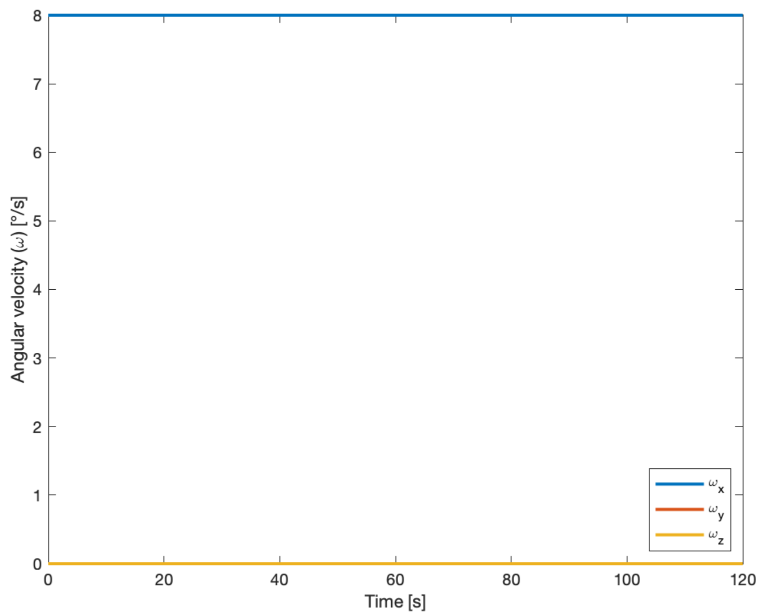


Figure 12: Angular velocity evolution for angular velocity vector for Case 1

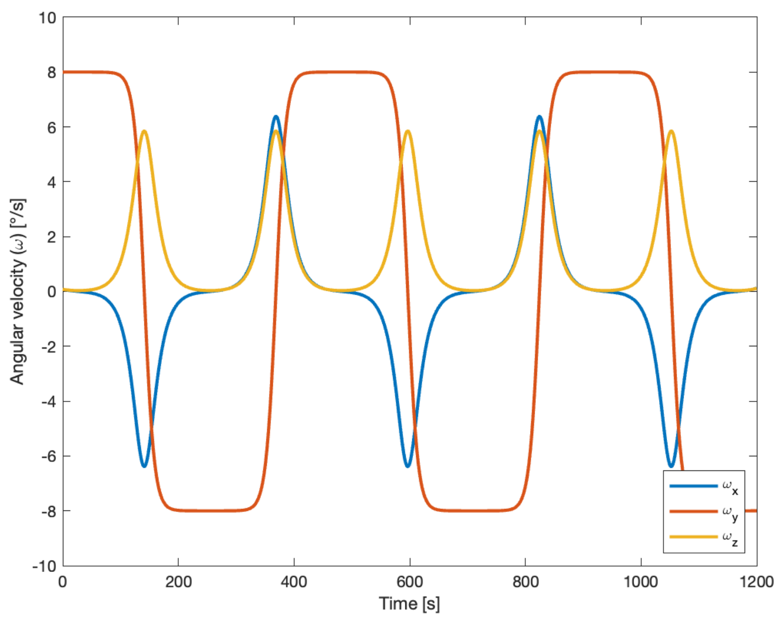


Figure 13: Angular velocity evolution for angular velocity vector for Case 2

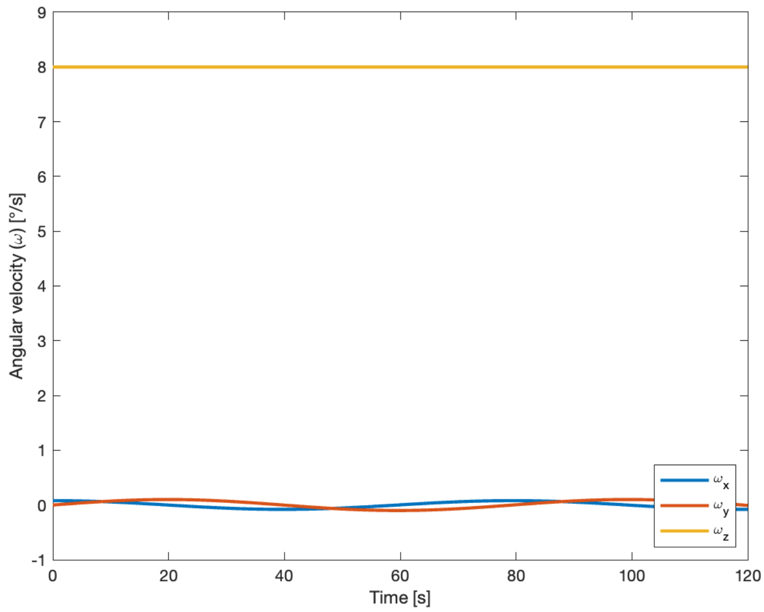


Figure 14: Angular velocity evolution for angular velocity vector for Case 3

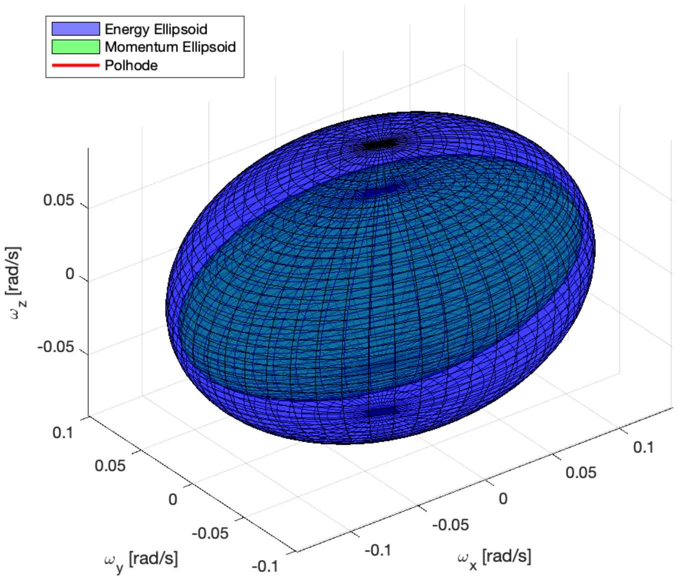


Figure 15: Polhode and ellipsoids for angular velocity vector for Case 1

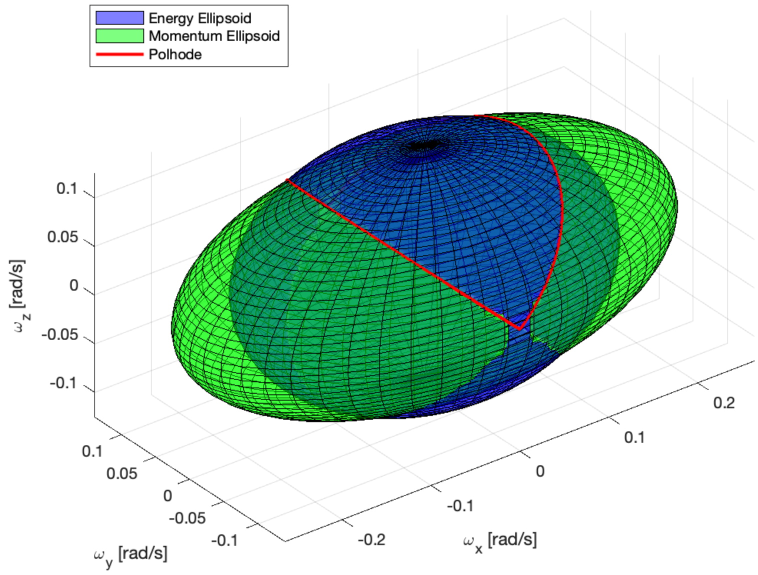


Figure 16: Polhode and ellipsoids for angular velocity vector for Case 2

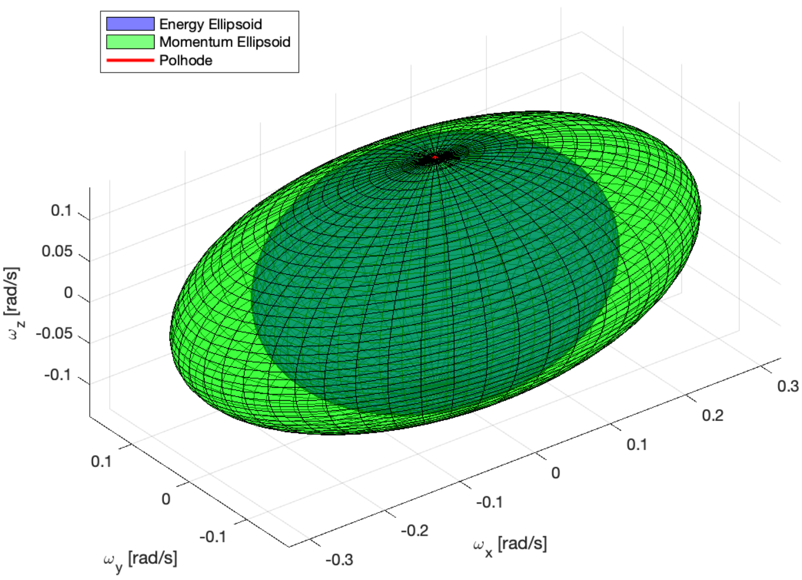


Figure 17: Polhode and ellipsoids for angular velocity vector for Case 3

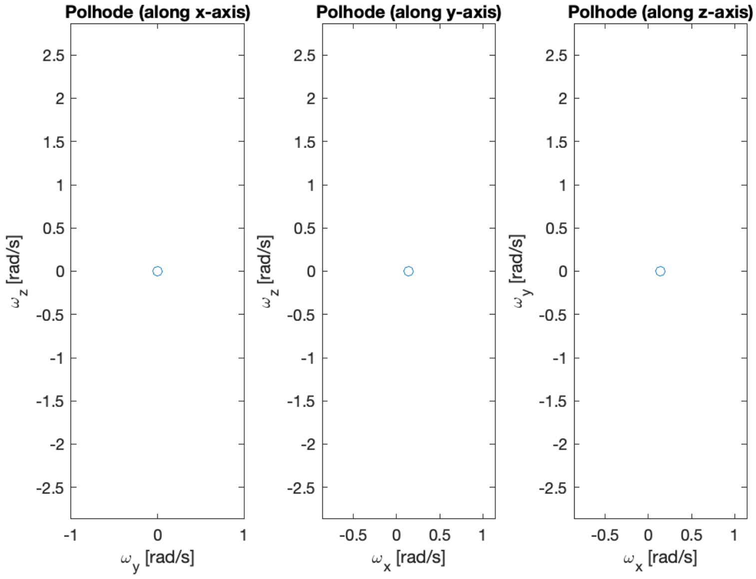


Figure 18: 2D views of polhode for angular velocity vector for Case 1

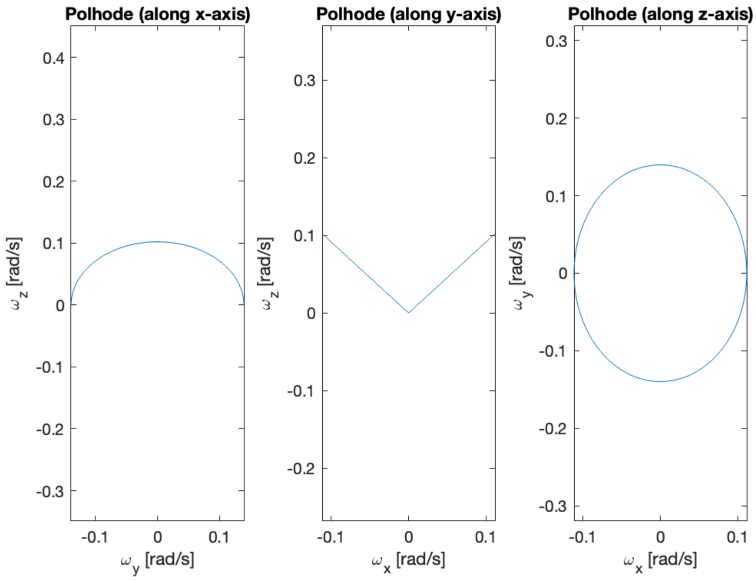


Figure 19: 2D views of polhode for angular velocity vector for Case 2

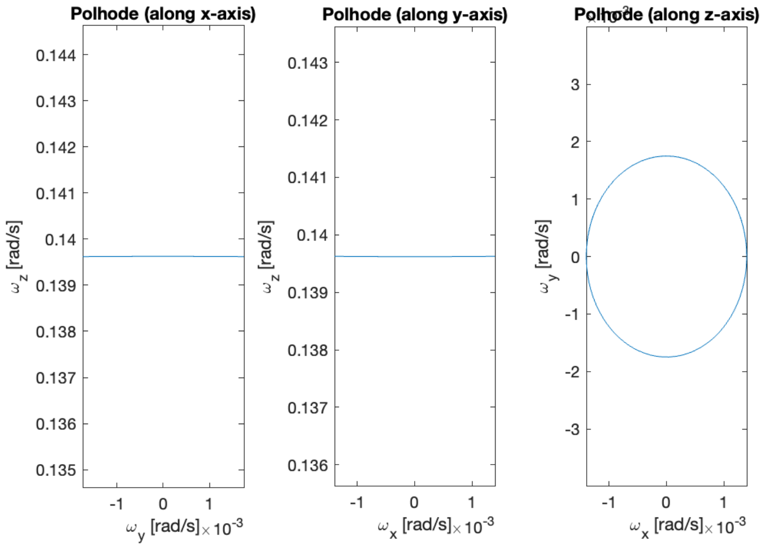


Figure 20: 2D views of polhode for angular velocity vector for Case 3

3 PROBLEM SET 3

3.1 PROBLEM 1

Impose that satellite is axial-symmetric (Ix=Iy≠Iz). Repeat numerical simulation from previous pset using initial condition 4) from previous pset.

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

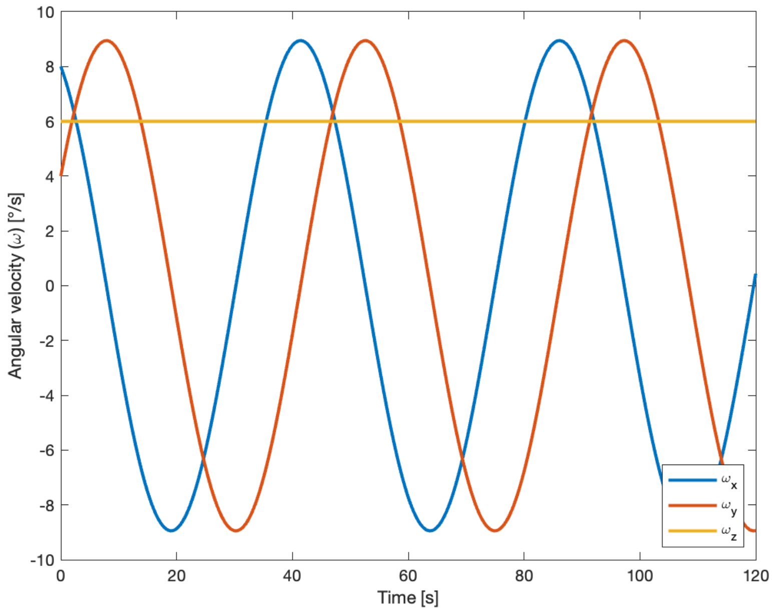


Figure 21: Numerical solution results

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 λ and ω_{xy}, as defined below.

$$\lambda = \frac{I_z - I_x}{I_x}\omega_{z0}$$
$$\omega_{xy} = (\omega_{x0} + i\omega_{y0})e^{i\lambda t}$$

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

$$\omega_x = \mathrm{Re}(\omega_{xy})$$
$$\omega_y = \mathrm{Im}(\omega_{xy})$$
$$\omega_z = \omega_{z0}$$

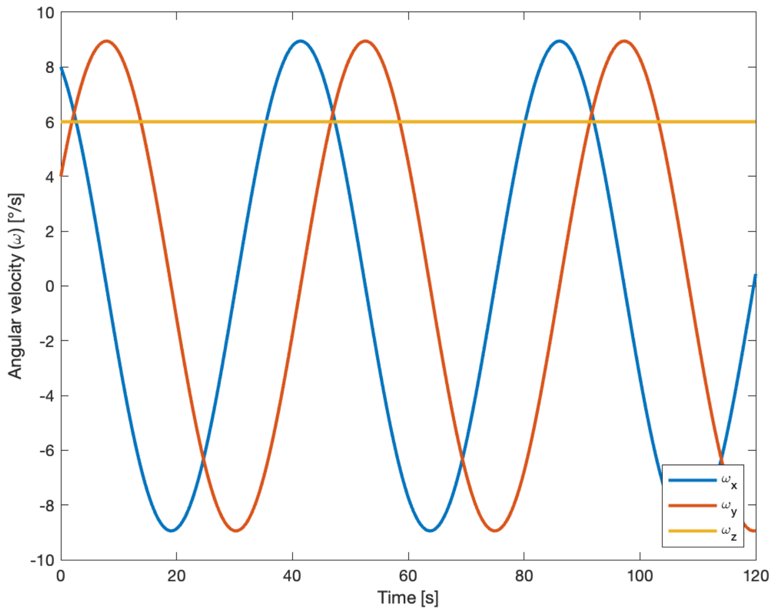


Figure 22: Analytical solution results

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.

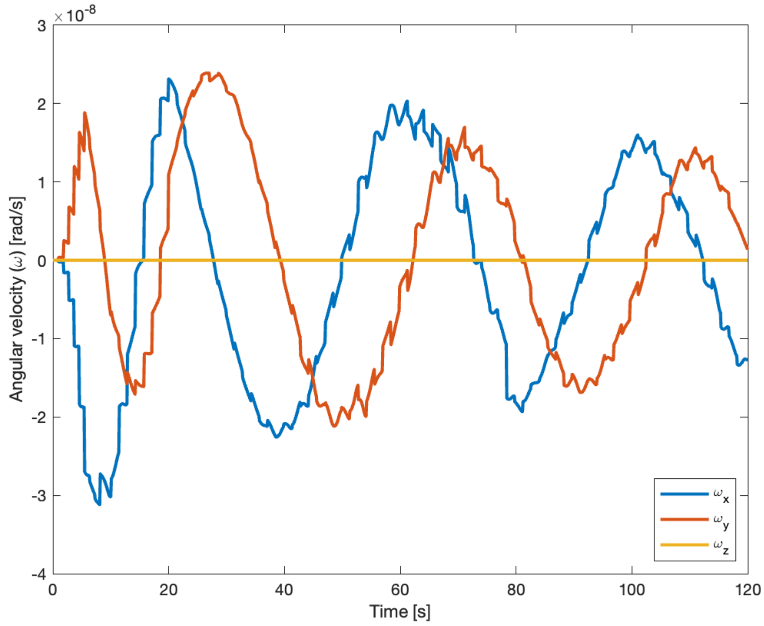


Figure 23: Error between numerical and analytical solutions

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.

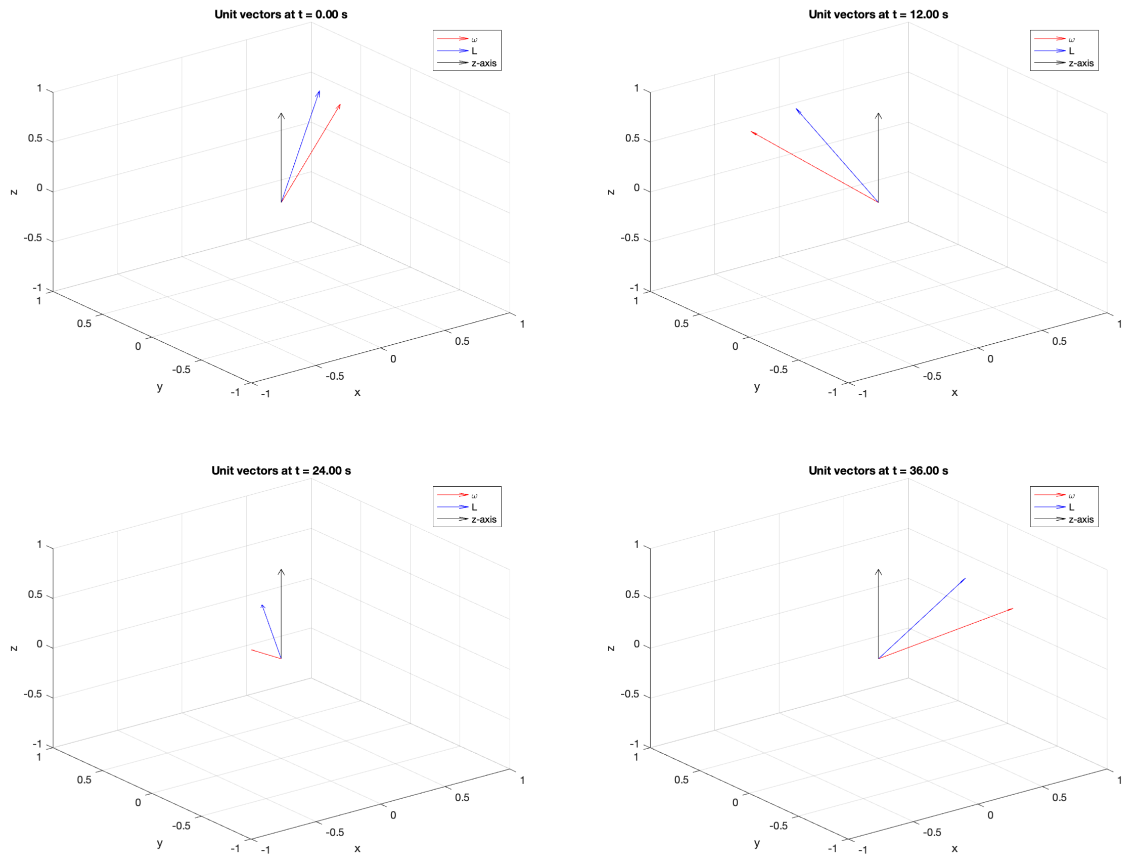


Figure 24: Angular velocity (red) and angular momentum (blue) unit vectors over time.

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

```
function stateDot = kinQuaternion(t,state,Ix,Iy,Iz)
    % Computes state derivatives for quaternions, angular velocity
    % Assign variables
    q = state(1:4);
    wx = state(5);
    wy = state(6);
    wz = state(7);

    stateDot = zeros(7,1);
    % Angular velocity time derivatives
    stateDot(5) = (Iy - Iz) / Ix * wy * wz;
    stateDot(6) = (Iz - Ix) / Iy * wz * wx;
    stateDot(7) = (Ix - Iy) / Iz * wx * wy;
    sigma = [0, wz, -wy, wx; ...
        -wz, 0, wx, wy; ...
        wy, -wx, 0, wz; ...
        -wx, -wy, -wz, 0];
    % Quaternion time derivative
    qDot = 0.5 * sigma * q;
    stateDot(1:4) = qDot;
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.

```
function [q,w] = kinQuaternionForwardEuler(q0,w0,Ix,Iy,Iz,tFinal,tStep)
    % Forward Euler integration for quaternions, angular velocity
    nStep = ceil(tFinal/tStep);
    q = nan(nStep+1,4);
    w = nan(nStep+1,3);
    q(1,:) = q0';
    w(1,:) = w0';
    for i = 1:nStep
        t = i * tStep;
        qi = q(i,:)';
        wi = w(i,:)';
        state = [qi;wi];
        stateDot = kinQuaternion(t,state,Ix,Iy,Iz);
        nextState = state + tStep * stateDot;
        q(i+1,:) = nextState(1:4) / norm(nextState(1:4));
        w(i+1,:) = nextState(5:7);
    end
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.

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

```
    q(1,:) = q0';
    w(1,:) = w0';
    for i = 1:nStep
        t = i * tStep;
        qi = q(i,:)';
        wi = w(i,:)';
        state = [qi;wi];
        k1 = kinQuaternion(t,state,Ix,Iy,Iz);
        k2 = kinQuaternion(t+tStep/2,state+(k1*tStep/2),Ix,Iy,Iz);
        k3 = kinQuaternion(t+tStep/2,state+(k2*tStep/2),Ix,Iy,Iz);
        k4 = kinQuaternion(t+tStep,state+(k3*tStep),Ix,Iy,Iz);
        nextState = state + tStep * (k1/6 + k2/3 + k3/3 + k4/6);
        q(i+1,:) = nextState(1:4) / norm(nextState(1:4));
        w(i+1,:) = nextState(5:7);
    end
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.

$$\frac{d\phi}{dt} = \frac{\omega_x \sin(\psi) + \omega_y \cos(\psi)}{\sin(\theta)}$$

$$\frac{d\theta}{dt} = \omega_x \cos(\psi) - \omega_y \sin(\psi)$$

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

```
function stateDot = kinEulerAngle(t,state,Ix,Iy,Iz)
    % Computes state derivative for Euler angles, angular velocity
    % Assign variables
    phi = state(1);
    theta = state(2);
    w = state(4:6);

    stateDot = zeros(6,1);
    % Angular velocity time derivatives
    stateDot(4) = (Iy - Iz) / Ix * w(2) * w(3);
    stateDot(5) = (Iz - Ix) / Iy * w(3) * w(1);
    stateDot(6) = (Ix - Iy) / Iz * w(1) * w(2);
    % Euler angle time derivatives
    % 312
    EPrimeInv = [sin(phi)*sin(theta) cos(phi)*sin(theta) cos(theta); ...
        cos(theta)*cos(phi) -sin(phi)*cos(theta) 0; ...
        sin(phi) cos(phi) 0] * (1 / cos(theta));
    % 313
    % EPrimeInv = [-sin(phi)*cos(theta) -cos(phi)*cos(theta) ...
       sin(theta); ...
```

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

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

```
function [state] = ...
    kinEulerAngleForwardEuler(state0,Ix,Iy,Iz,tFinal,tStep)
    % Forward Euler integration for state Euler angles, angular velocity
    nStep = ceil(tFinal/tStep);
    state = nan(nStep+1,6);
    state(1,:) = state0;
    for i = 1:nStep
        t = i * tStep;
        statei = state(i,:)';
        stateDot = kinEulerAngle(t,statei,Ix,Iy,Iz);
        state(i+1,:) = statei + tStep * stateDot;
    end
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).

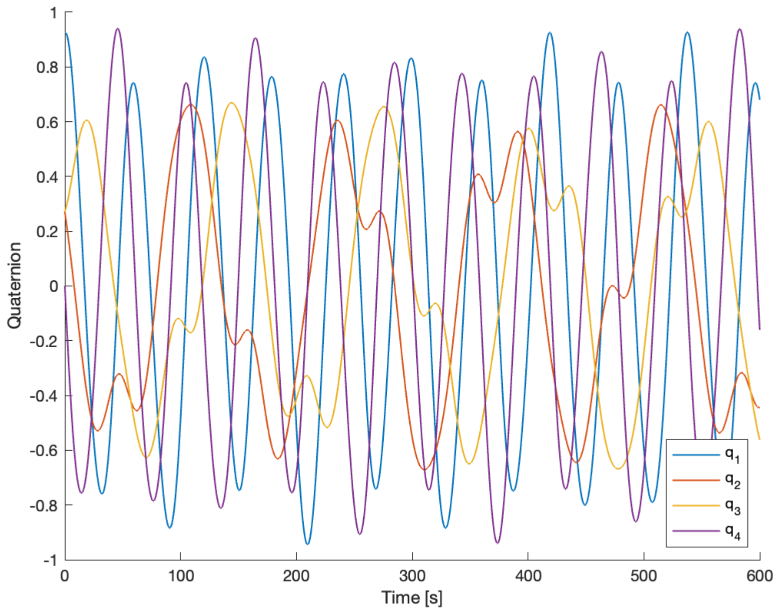


Figure 25: Evolution of quaternions

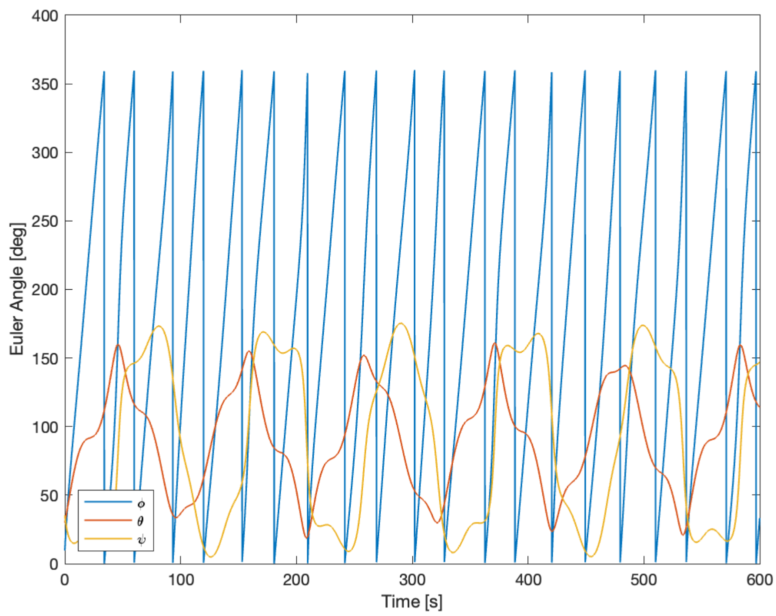


Figure 26: Evolution of Euler angles

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.

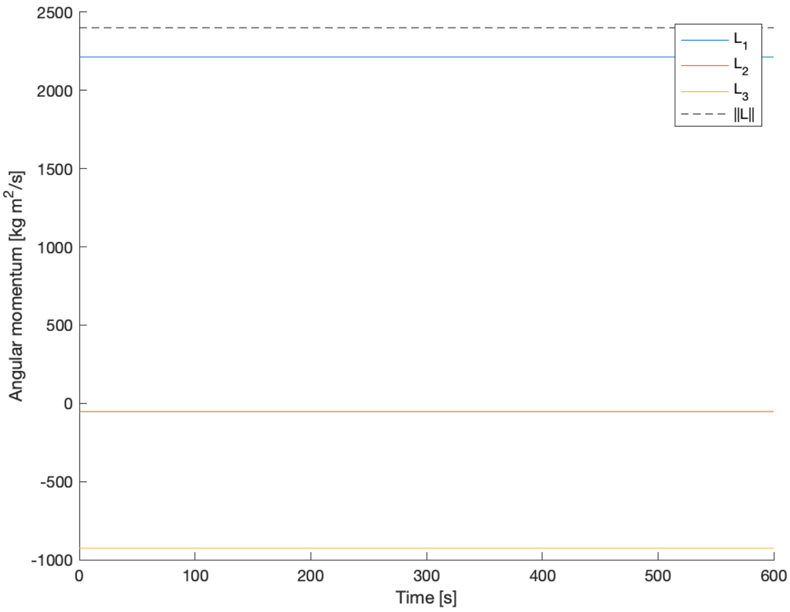


Figure 27: Angular momentum in inertial coordinates is constant

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.

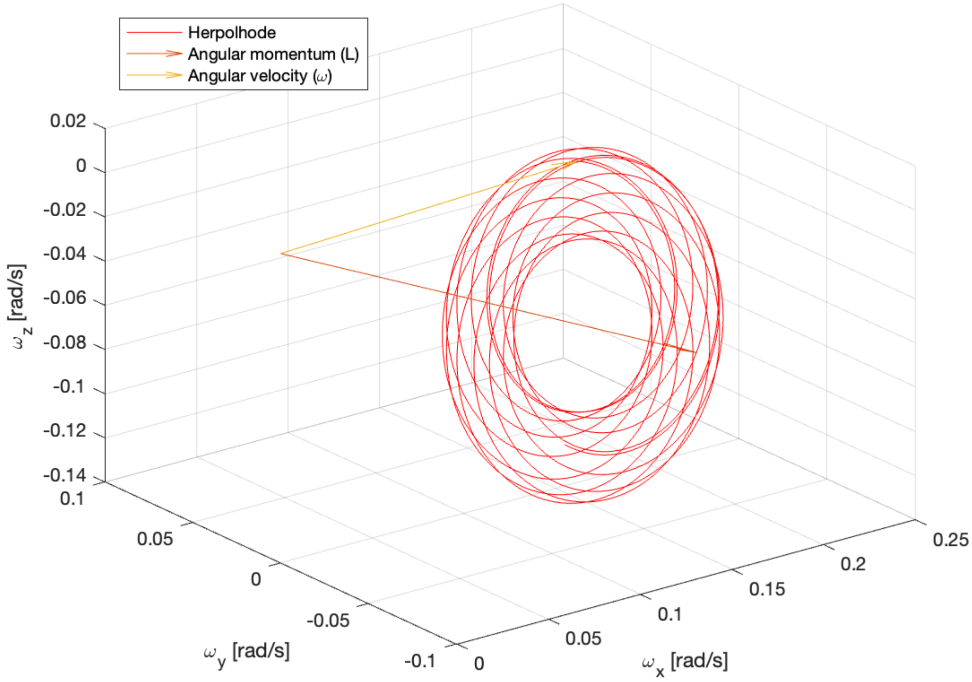


Figure 28: Herpolhode (Animated: `https://tinyurl.com/herpolhode`)

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.

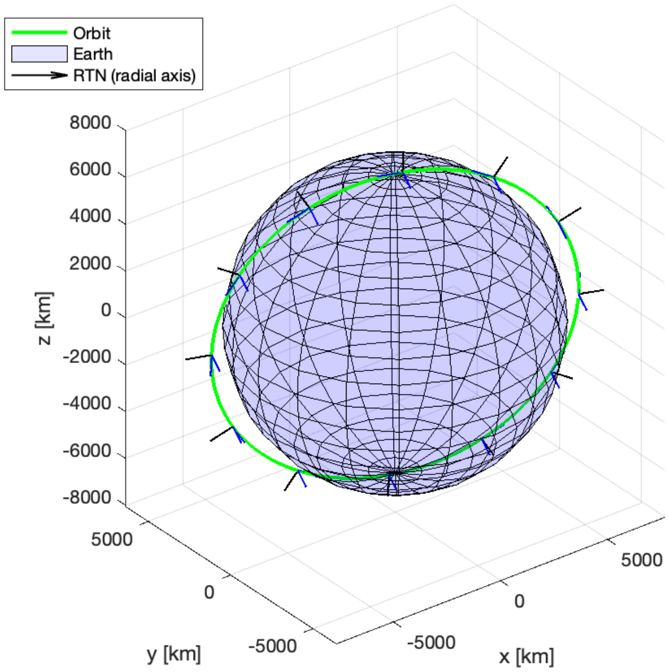


Figure 29: Propagation of RTN frame

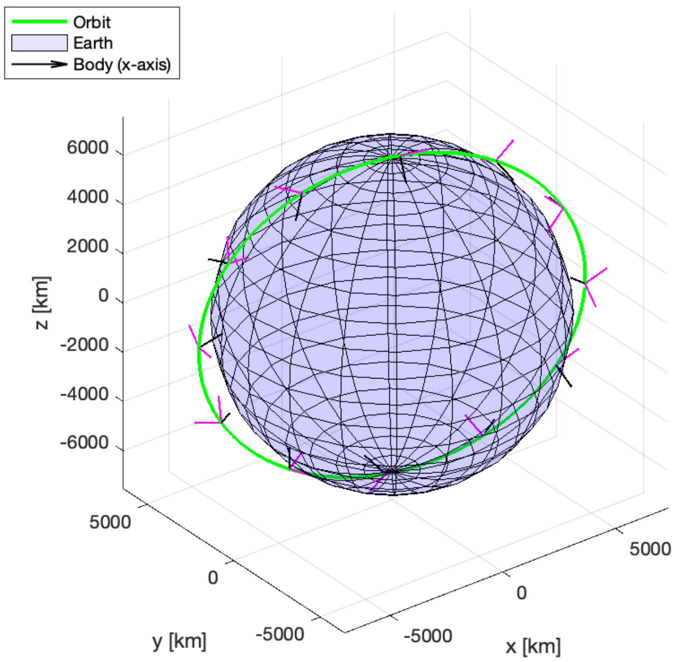


Figure 30: Propagation of body axes

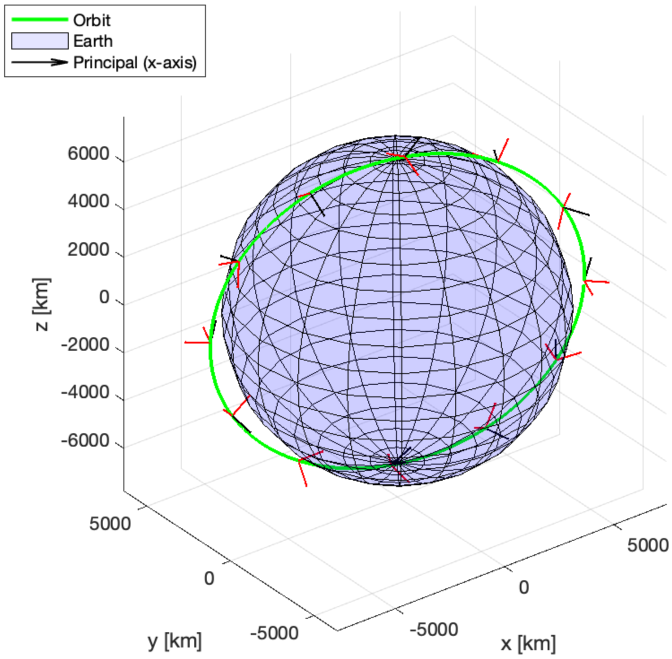


Figure 31: Propagation of principal axes

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,

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

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 ω_x and ω_y remain zero and ω_z maintains a constant value.

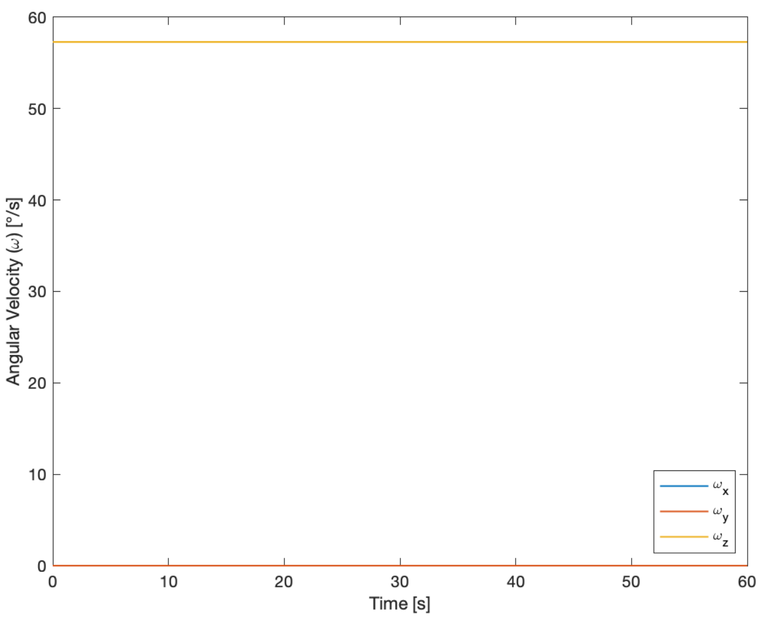


Figure 32: Evolution of angular velocity

Similarly, in Figure 33, ϕ and θ Euler angles remain at zero while the ψ Euler angle increases linearly.

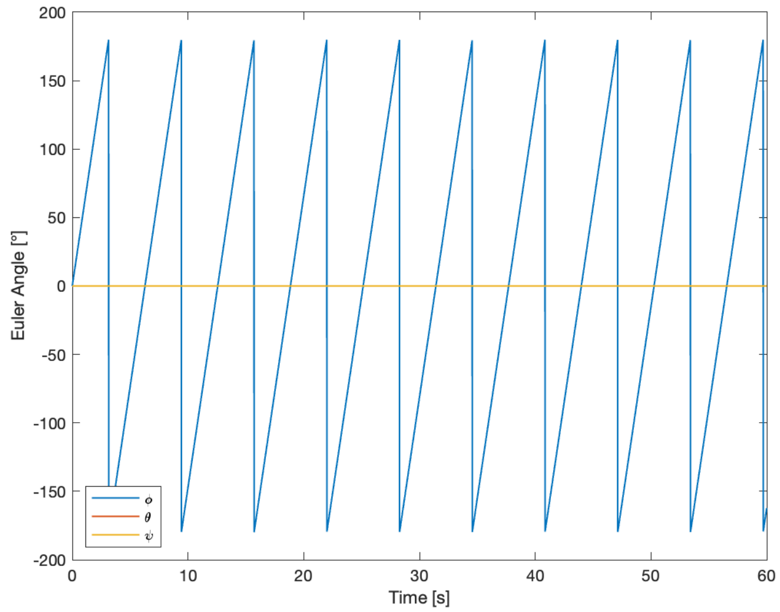


Figure 33: Evolution of Euler angles

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.

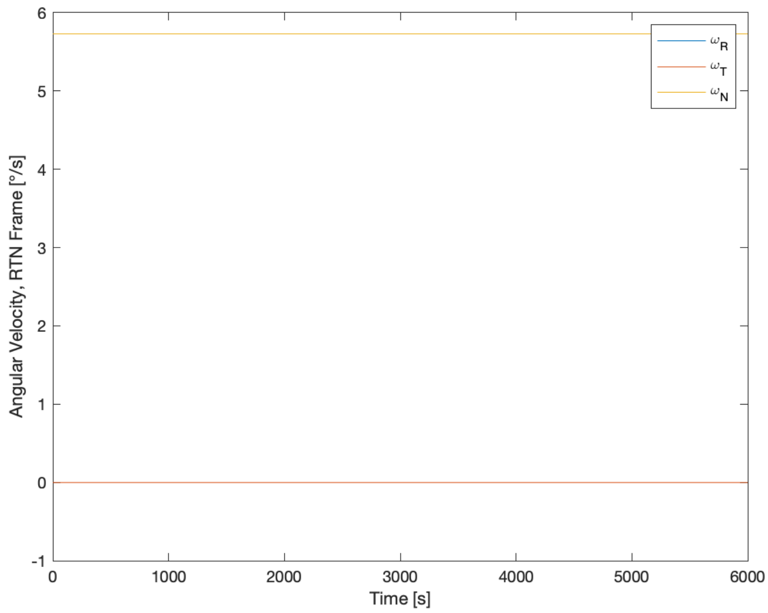


Figure 34: Evolution of angular velocity

We set a nonzero ω_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 θ and ψ Euler angles related to the RTN frame are constant while ϕ varies.

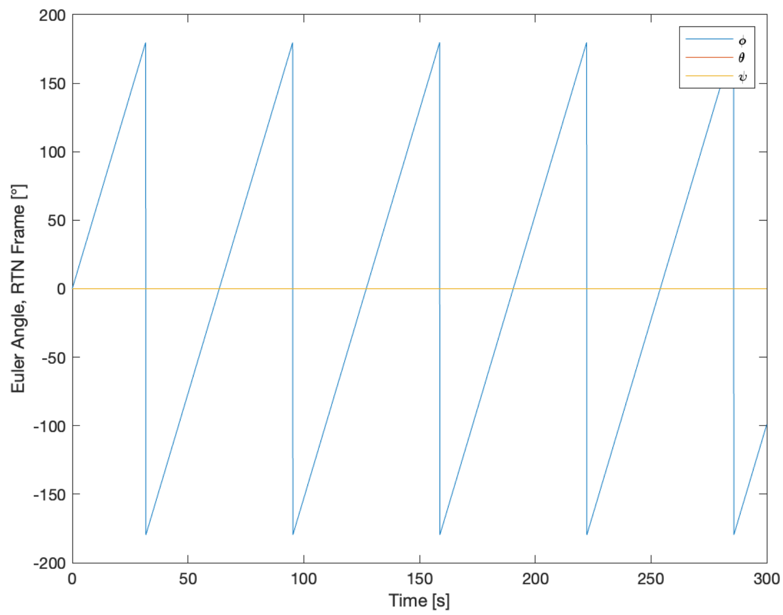


Figure 35: Evolution of Euler angles

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.

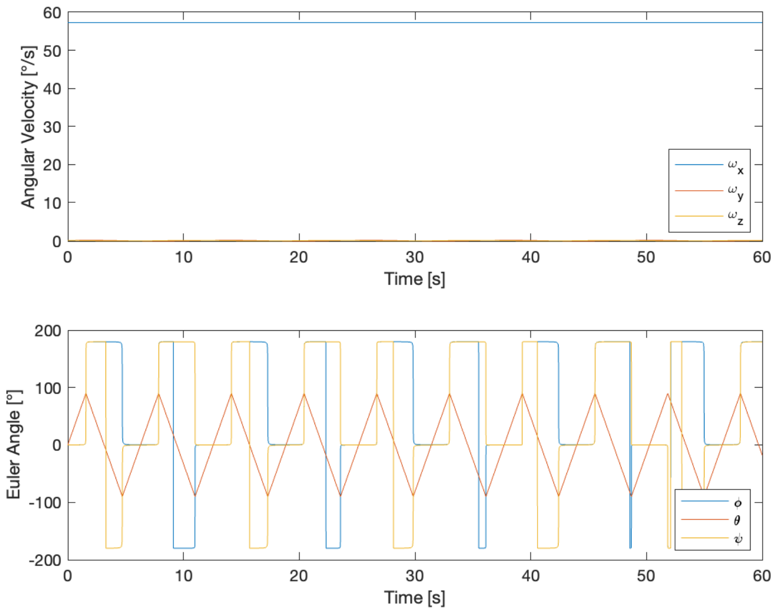


Figure 36: Simulation of satellite spinning on its minimum principal axis

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.

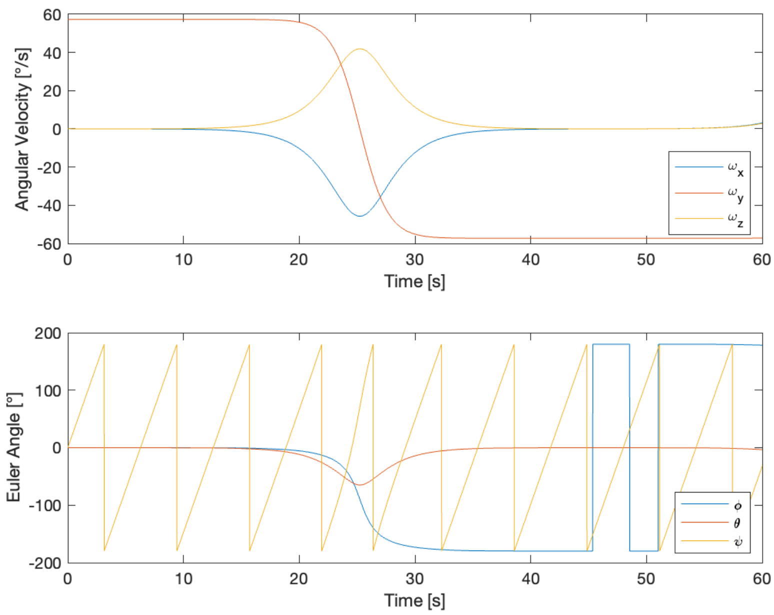


Figure 37: Simulation of satellite spinning on its intermediate principal axis

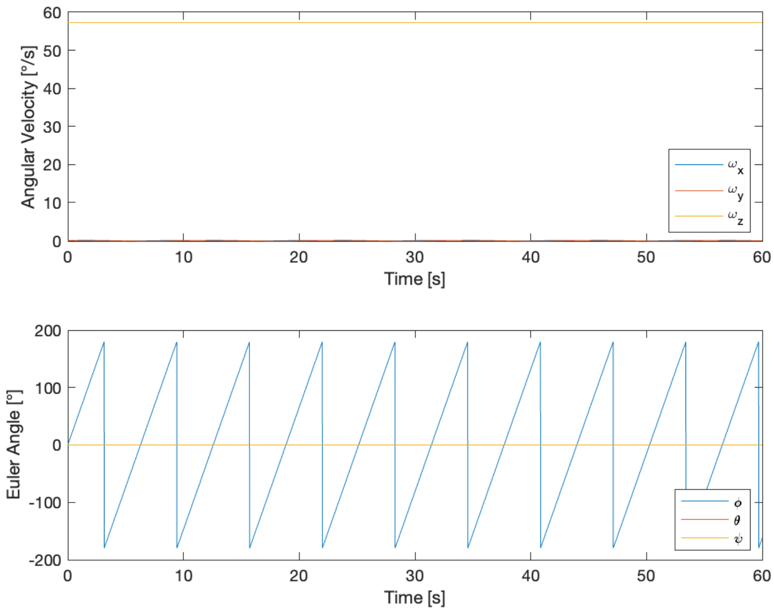


Figure 38: Simulation of satellite spinning on its maximum principal axis

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.

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

```
    % Adds momentum wheel
    % Assign variables
    phi = state(1);
    theta = state(2);
    w = state(4:7);

    stateDot = zeros(7,1);
    % Angular velocity time derivatives
    wDot = eulerEquationWheel(t,w,M,r,Ix,Iy,Iz,Ir);
    stateDot = zeros(7,1);
    stateDot(4) = wDot(1);
    stateDot(5) = wDot(2);
    stateDot(6) = wDot(3);
    stateDot(7) = wDot(4);
    % Euler angle time derivatives
    % 312
    EPrimeInv = [sin(phi)*sin(theta) cos(phi)*sin(theta) cos(theta); ...
        cos(theta)*cos(phi) -sin(phi)*cos(theta) 0; ...
        sin(phi) cos(phi) 0] * (1 / cos(theta));
    % 313
    % EPrimeInv = [-sin(phi)*cos(theta) -cos(phi)*cos(theta) ...
        sin(theta); ...
    %     cos(phi)*sin(theta) -sin(phi)*sin(theta) 0; ...
    %     sin(phi) cos(phi) 0] * (1 / sin(theta));
    stateDot(1:3) = EPrimeInv * w(1:3);
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.

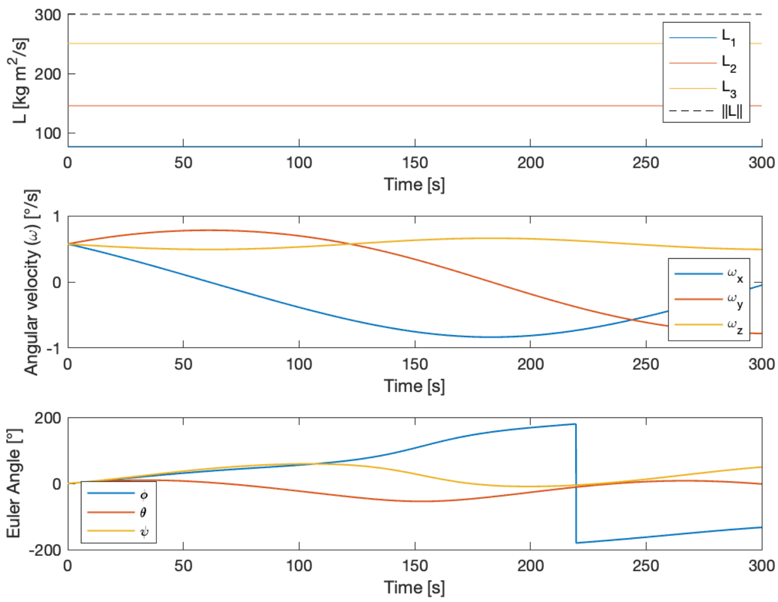


Figure 39: Angular momentum conserved with angular momentum components constant

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.

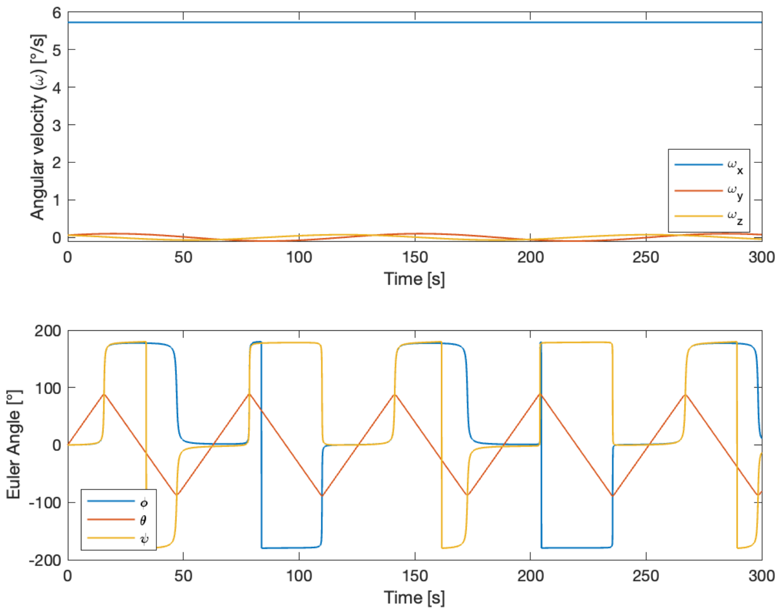


Figure 40: Stability analysis about minimum inertia principal axis

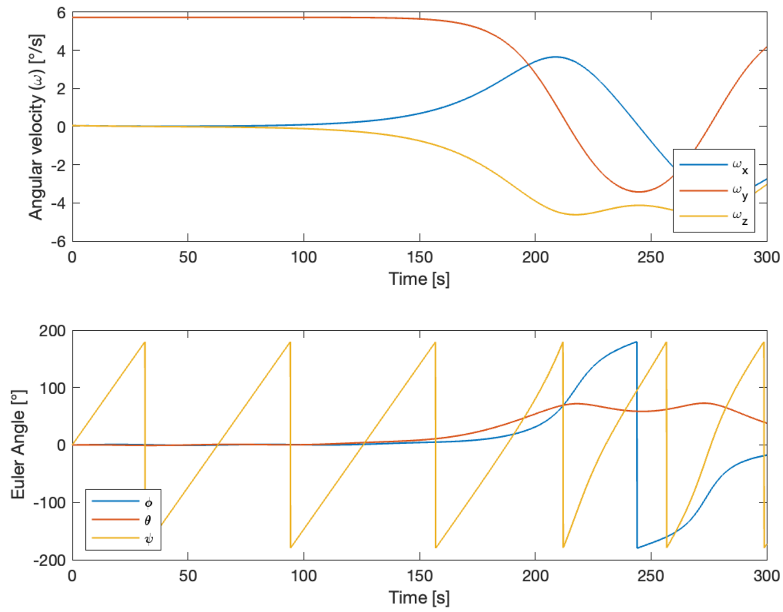


Figure 41: Stability analysis about intermediate axis

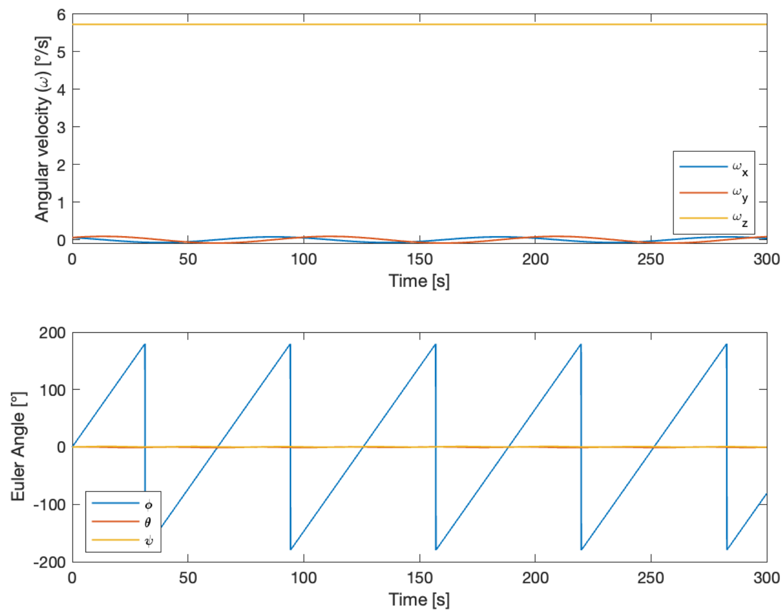


Figure 42: Stability analysis about maximum inertia principal axis

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.

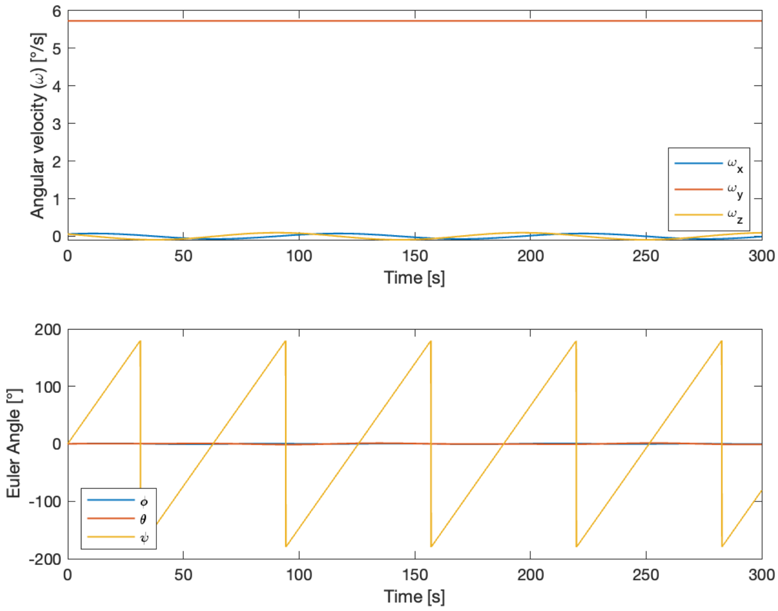


Figure 43:

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.

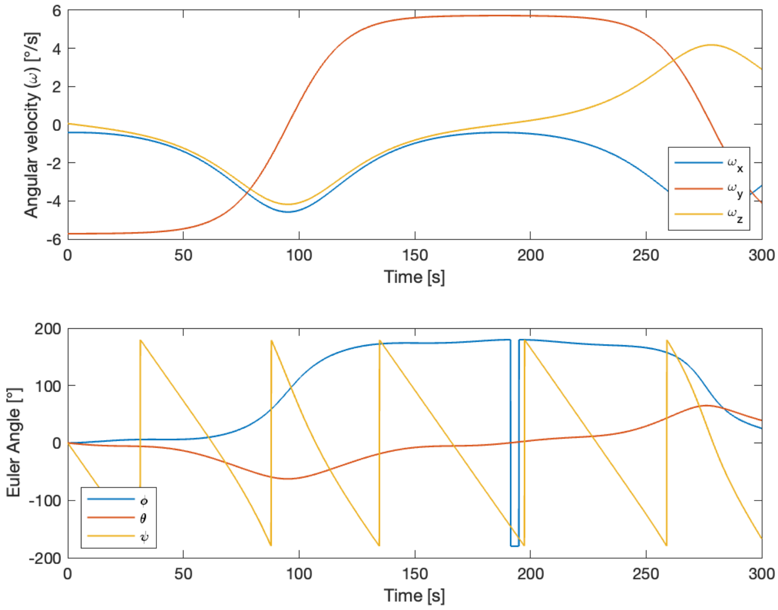


Figure 44: Initially unstable attitude with low momentum wheel angular velocity

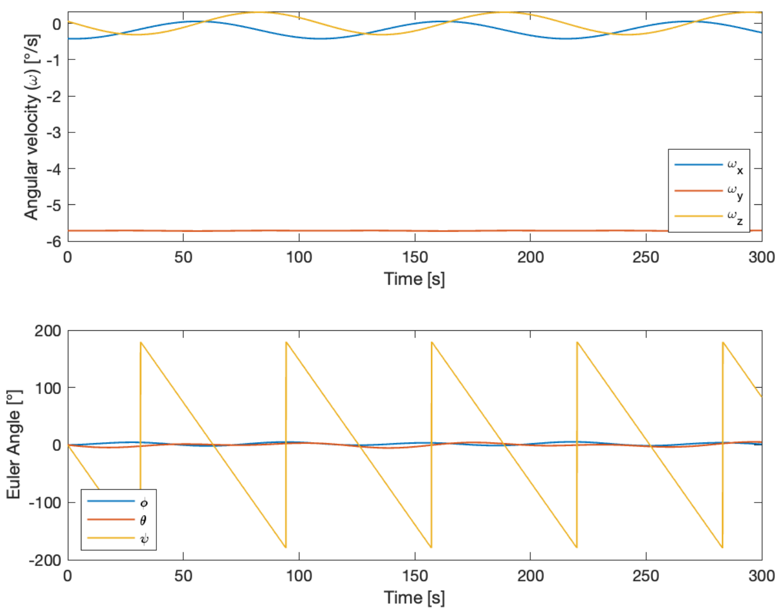


Figure 45: Periodically stable attitude after increasing momentum wheel angular velocity 10x

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, $\vec{c} = [c_x, c_y, c_z]^\intercal$ 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.

```
function [stateDot] = gravGrad(t,state,Ix,Iy,Iz,n)
    % Orbit position and velocity
    r = state(1:3);
    v = state(4:6);

    % Angular velocity
    w = state(7:9);

    % Euler angles
    phi = state(10);
    theta = state(11);

    stateDot = zeros(12,1);
    stateDot(1:3) = v;
    stateDot(4:6) = (-3.986 * 10^5 / norm(r)^2) * r / norm(r); % km/s^2

    radial = r / norm(r);
    A_ECI2P = e2A(state(10:12));
    c = A_ECI2P * radial;
    M = gravGradTorque(Ix,Iy,Iz,n,c);
    stateDot(7) = (M(1) - (Iz - Iy) * w(2) * w(3)) / Ix;
    stateDot(8) = (M(2) - (Ix - Iz) * w(3) * w(1)) / Iy;
    stateDot(9) = (M(3) - (Iy - Ix) * w(1) * w(2)) / Iz;

    % Euler angle time derivatives
    % 312
    EPrimeInv = [sin(phi)*sin(theta) cos(phi)*sin(theta) cos(theta); ...
        cos(theta)*cos(phi) -sin(phi)*cos(theta) 0; ...
        sin(phi) cos(phi) 0] * (1 / cos(theta));
```

```
    % 313
    % EPrimeInv = [-sin(phi)*cos(theta) -cos(phi)*cos(theta) ...
      sin(theta); ...
    %      cos(phi)*sin(theta) -sin(phi)*sin(theta) 0; ...
    %      sin(phi) cos(phi) 0] * (1 / sin(theta));
    stateDot(10:12) = EPrimeInv * w;
end
```

```
function M = gravGradTorque(Ix,Iy,Iz,n,c)
    cx = c(1);
    cy = c(2);
    cz = c(3);
    M(1) = 3 * n^2 * (Iz - Iy) * cy * cz;
    M(2) = 3 * n^2 * (Ix - Iz) * cz * cx;
    M(3) = 3 * n^2 * (Iy - Ix) * cx * cy;
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 ($I_x = 7707$, $I_y = 14\,563$, $I_z = 18\,050\,\mathrm{kg \cdot m^2}$), known values for Earth ($a = 7125.49\,\mathrm{km}$, $\mu = 398\,600\,\mathrm{km^3/s^2}$), and an arbitrary $\vec{c} = [\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}]$. With these values, we arrive at:

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

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.

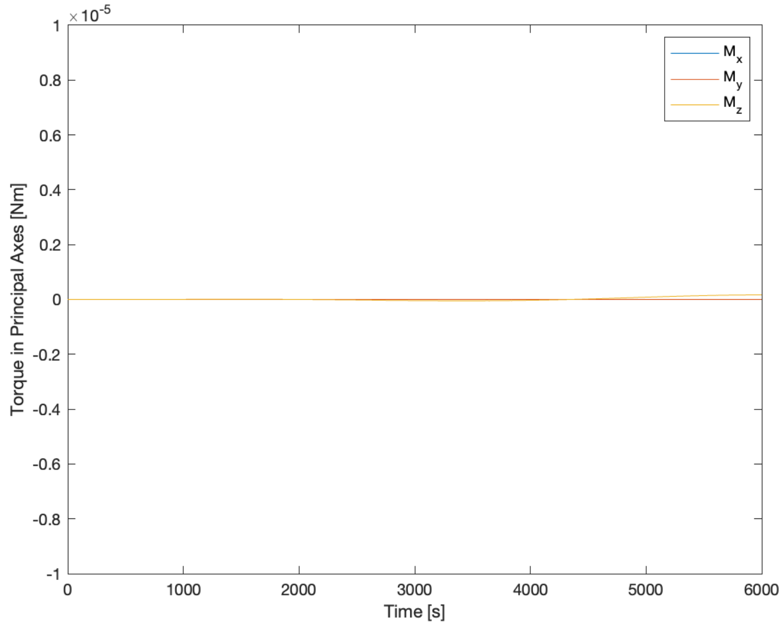


Figure 46: Zero gravity gradient torque for satellite aligned with RTN frame

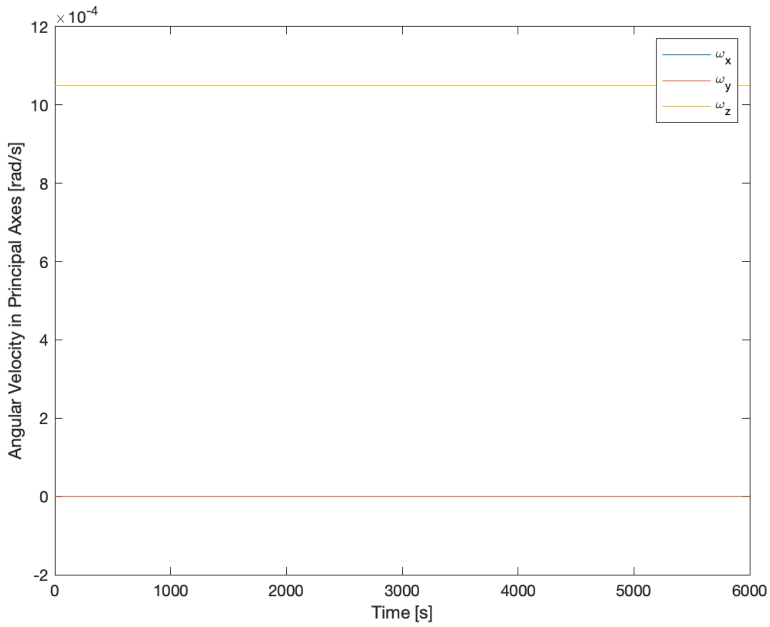


Figure 47: Angular velocity parallel to RTN normal equal to mean motion, all others zero

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.

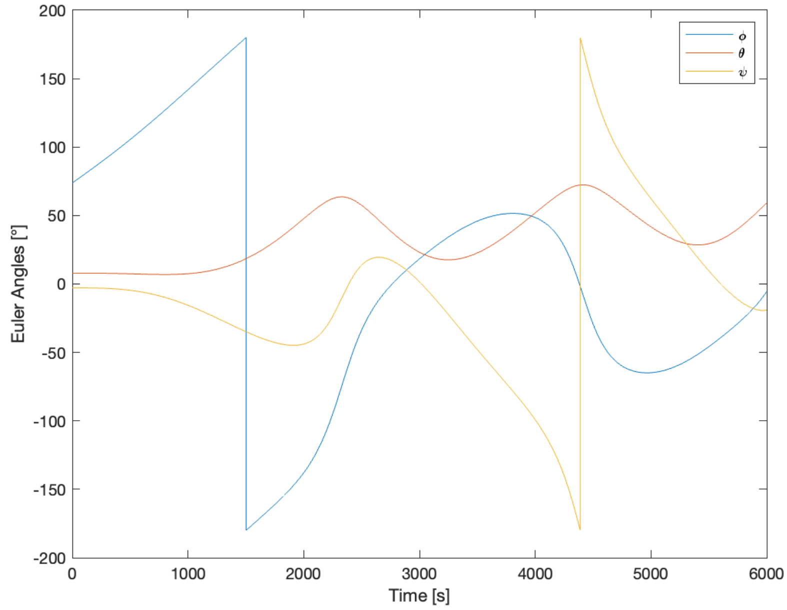


Figure 48: Euler angles with gravity gradient for satellite with arbitrary initial conditions

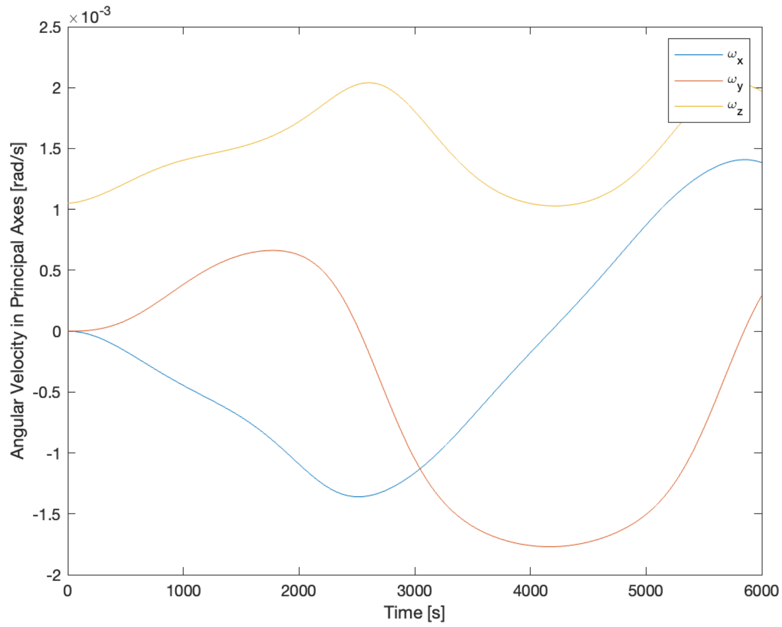


Figure 49: Angular velocity with gravity gradient for satellite with arbitrary initial conditions

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.

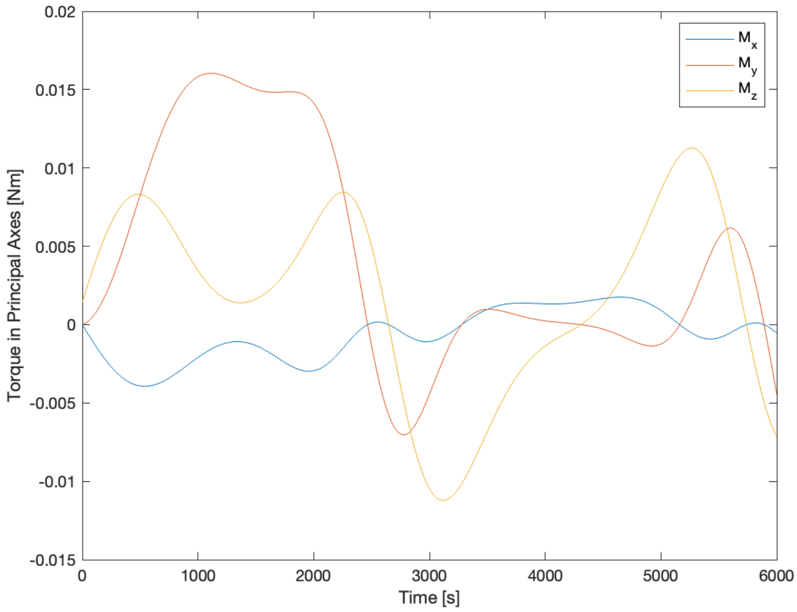


Figure 50: Gravity gradient torques for satellite with arbitrary initial conditions

5 PROBLEM SET 5

5.1 PROBLEM 1

Gravity gradient torque (stability)

a. Calculate the coefficients Ki 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 inequalites hold for when the roll and yaw are stable.

$$k_N = \frac{I_T - I_R}{I_N}, \qquad k_T = \frac{I_N - I_R}{I_T}, \qquad k_R = \frac{I_N - I_T}{I_R}$$
$$k_T > k_R, \qquad k_R k_T > 0, \qquad 1 + 3k_T + k_R k_T > 4\sqrt{k_R k_T}$$

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.

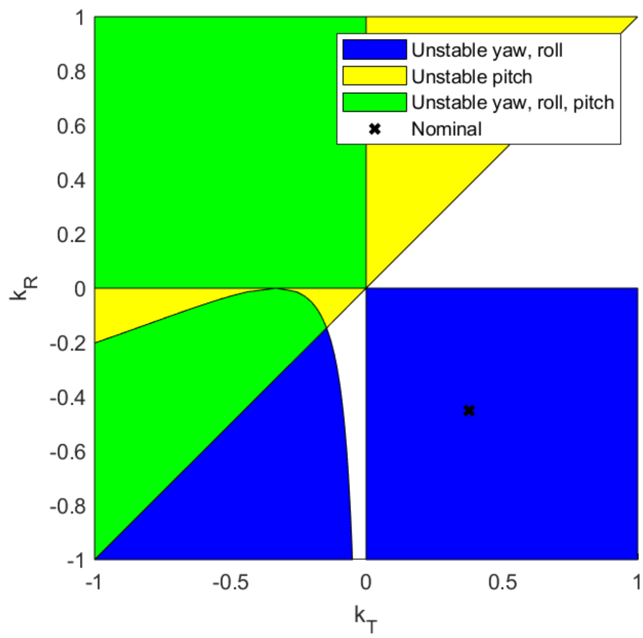


Figure 51: Stability for nominal axes

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.

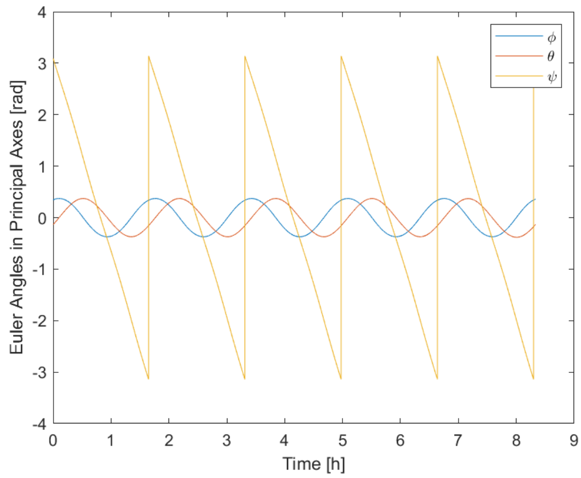


Figure 52: Attitude evolution for unstable orientation without perturbations

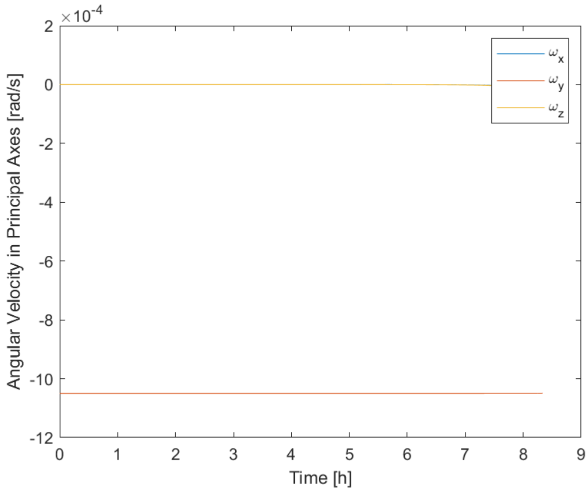


Figure 53: Angular velocity evolution for unstable orientation without perturbations

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.

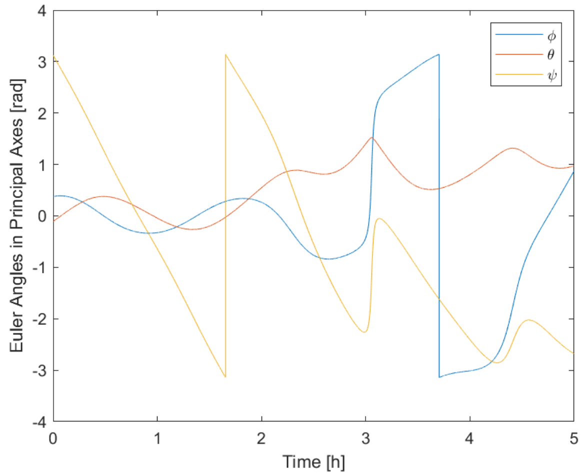


Figure 54: Attitude evolution for unstable orientation with 1% perturbations

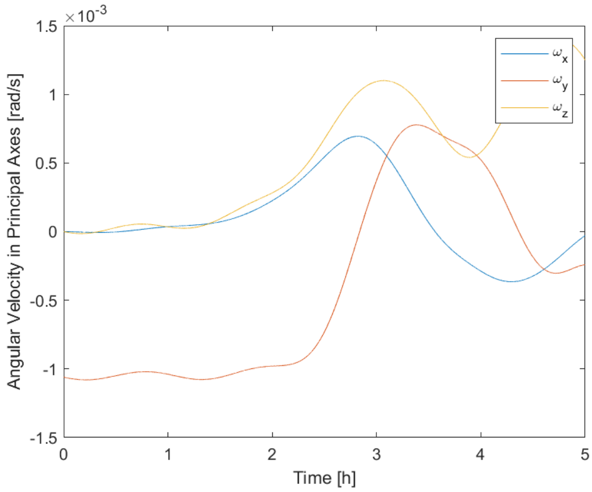


Figure 55: Angular velocity evolution for unstable orientation with 1% perturbations

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.

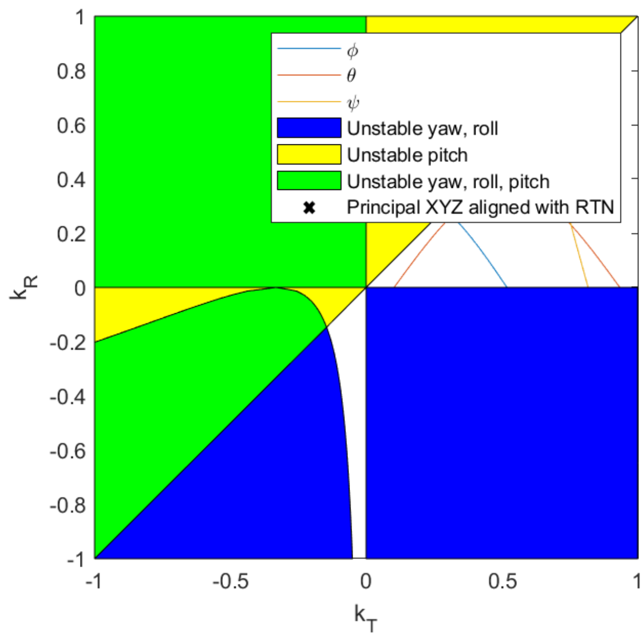


Figure 56: Stability for aligned principal axes

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.

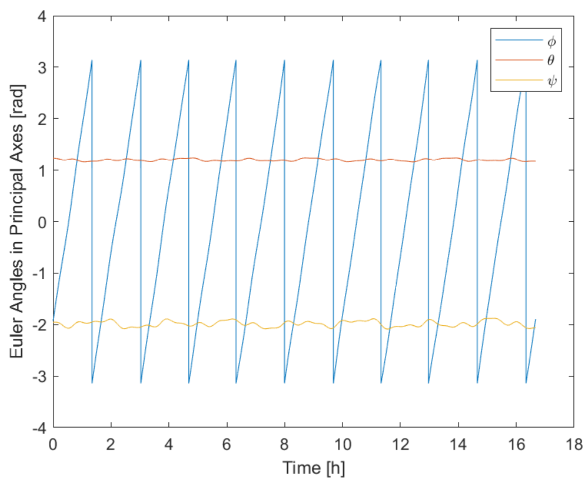


Figure 57: Attitude evolution for stable orientation with 1% perturbations

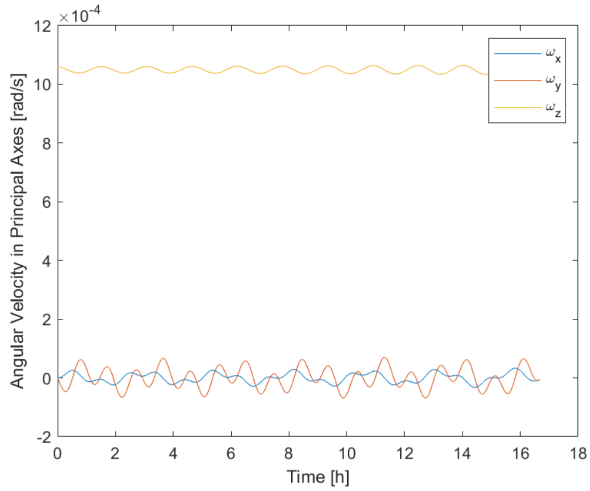


Figure 58: Angular velocity evolution for stable orientation with 1% perturbations

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 (θ), and longitude (ϕ). 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].

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

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.

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

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.

```
function [stateDot] = orbitTorque(t,state,Ix,Iy,Iz, ...
    CD,Cd,Cs,P,m,UT1, ...
    barycenter,normal,area,cm,n)
    warning('off','aero:atmosnrlmsise00:setf107af107aph')

    % Orbit position and velocity
    r = state(1:3);
    v = state(4:6);
    rEarth = state(13:15);
    vEarth = state(16:18);

    % Angular velocity
    w = state(7:9);

    % Euler angles
    phi = state(10);
    theta = state(11);
```

```

    % Gravity gradient torque
    radial = r / norm(r);
    A_ECI2P = e2A(state(10:12));
    c = A_ECI2P * radial;
    Mgg = gravGradTorque(Ix,Iy,Iz,n,c);

    % Drag torque
    % Hard-coded with Earth radius for now
    [~,density] = atmosnrlmsise00(1000 * (norm(r) - ...
        6378.1),0,0,2000,1,0);
    rho = density(6);
    vPrincipal = A_ECI2P * (v + cross([0; 0; 7.2921159E-5],r));
    [~,Md] = drag(vPrincipal,rho,CD,barycenter,normal,area,cm);

    % Solar radiation pressure torque
    % Hard-coded with Earth axial tilt for now
    Rx = [1 0 0; 0 cosd(23.5) -sind(23.5); 0 sind(23.5) cosd(23.5)];
    s = A_ECI2P * (-Rx * rEarth - r); % SCI -> ECI -> XYZ
    [~,Msrp] = srp(s,P,Cd,Cs,barycenter,normal,area,cm);

    % Magnetic field torque
    % Hard-coded with Earth radius for now
    Mm = magFieldTorque(m,r,state(10:12),t,6378.1,UT1);

    % Compute net moments
    Mx = Mgg(1) + Md(1) + Msrp(1) + Mm(1);
    My = Mgg(2) + Md(2) + Msrp(2) + Mm(2);
    Mz = Mgg(3) + Md(3) + Msrp(3) + Mm(3);

    % Time derivatives
    stateDot = zeros(12,1);
    stateDot(1:3) = v;
    stateDot(4:6) = (-3.986E5 / norm(r)^2) * r / norm(r); % km/s^2
    stateDot(7) = (Mx - (Iz - Iy) * w(2) * w(3)) / Ix;
    stateDot(8) = (My - (Ix - Iz) * w(3) * w(1)) / Iy;
    stateDot(9) = (Mz - (Iy - Ix) * w(1) * w(2)) / Iz;

    % 312 Euler angle time derivatives
    EPrimeInv = [sin(phi)*sin(theta) cos(phi)*sin(theta) cos(theta); ...
        cos(theta)*cos(phi) -sin(phi)*cos(theta) 0; ...
        sin(phi) cos(phi) 0] * (1 / cos(theta));
    stateDot(10:12) = EPrimeInv * w;

    % Sun position
    stateDot(13:15) = vEarth;
    stateDot(16:18) = (-1.327E11 / norm(rEarth)^2) * ...
        rEarth / norm(rEarth); % km/s^2
end
```

```
function [F,M] = drag(v,rho,CD,barycenter,normal,area,cm)
    % Compute drag in principal axes
    % RE = 6378.1 % km
    u = v / norm(v);
    N = normal;
    Aeff = ((u' * N) > 0) .* area;
```

```
    rC = barycenter - cm;
    D = -0.5 * CD * rho * norm(v)^2 * (u' * (N .* Aeff)) .* u;
    F = sum(D,2);
    M = sum(cross(rC,D),2);

    % Slow loop function (obsolete)
    % u = v / norm(v);
    % F = zeros([3 1]);
    % M = zeros([3 1]);
    % for i = length(area)
    %     n = normal(:,i);
    %     if dot(u,n) < 0
    %         rC = (barycenter(:,i) - cm);
    %         A = area(1,i);
    %         D = -0.5 * CD * rho * norm(v)^2 * dot(u,n) * u * A;
    %         M = M + cross(rC,D);
    %         F = F + D;
    %     end
    % end
end
```

```
function [F,M] = srp(s,P,Cd,Cs,barycenter,normal,area,cm)
    % Compute solar radiation pressure in principal axes
    u = s / norm(s);
    N = normal;
    Aeff = ((u' * N) > 0) .* area;
    rC = barycenter - cm;
    theta = acos((u' * N) ./ (norm(u) * vecnorm(N)));
    SRP = -P * cos(theta) .* Aeff .* ...
        ((1 - Cs) * u + 2 * (Cs * cos(theta) + Cd / 3) .* N);
    F = sum(SRP,2);
    M = sum(cross(rC,SRP),2);

    % Slow loop function (obsolete)
    % F = zeros([3 1]);
    % M = zeros([3 1]);
    % for i = length(area)
    %     n = normal(:,i);
    %     if dot(u,n) > 0
    %         rC = barycenter(:,i) - cm;
    %         theta = acos(dot(u,n) / (norm(u) * norm(n)));
    %         A = area(1,i);
    %         SRP = -P * cos(theta) * A * ...
    %             ((1 - Cs) * u + 2 * (Cs * cos(theta) + Cd / 3) * n);
    %         M = M + cross(rC,SRP);
    %         F = F + SRP;
    %     end
    % end
end
```

```
function [M, B_ECEF] = magFieldTorque(m,R,eulerAngle,t,RE,UT1)
    % Calculated the expected torque due to Earth's magnetic field
    % Inputs:
    % - m: magnetic moment of satellite [N * m / T]
    % - R: position vector of satellite [km]
```

```
    % - t: time of simulation [s]
    % - RE: radius of Earth [km] (6378 km)
    % - UT1: start time of simulation

    % Find lambda (longitude) and theta (colatitude)
    GMST = time2GMST(t,UT12MJD(UT1));
    [lat,lon,~] = ECEF2Geoc(ECI2ECEF(R,GMST),t);
    lambda = lat;
    phi = lon;
    theta = pi/2 - phi;

    [B_R,B_theta,B_phi] = magFieldEarth(R,lambda,theta,RE);
%     B_latlon = [B_R, B_theta, B_phi];

    timeVec = UT1;
    timeVec(2) = timeVec(2) + t/86400;
    [XYZ,H,D,I,F] = wrldmagm(norm(R)-RE,lat,lon,decyear(timeVec));

    delta = phi;
    alpha = lambda + GMST;

    B_x = (B_R * cos(delta) + B_theta * sin(delta)) * ...
        cos(alpha) - B_phi * sin(alpha);
    B_y = (B_R * cos(delta) + B_theta * sin(delta)) * ...
        sin(alpha) + B_phi * cos(alpha);
    B_z = (B_R * sin(delta) - B_theta * cos(delta));

    B_ECI = [B_x;B_y;B_z];
    B_ECEF = ECI2ECEF(B_ECI, GMST);
    B = e2A(eulerAngle) * B_ECI;

    M = cross(m,B);
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.

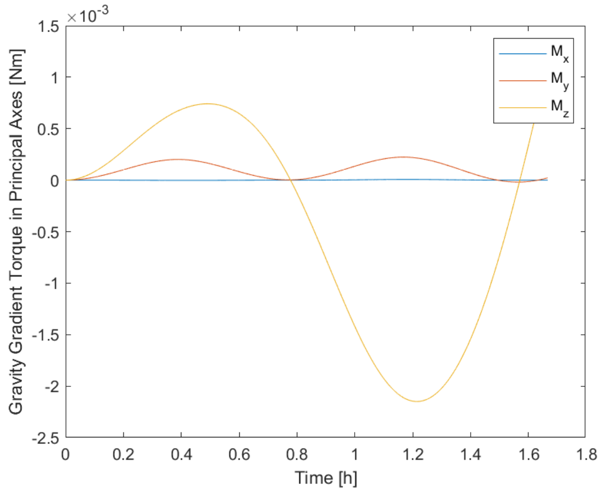


Figure 59: Numerical simulation of gravity gradient torques

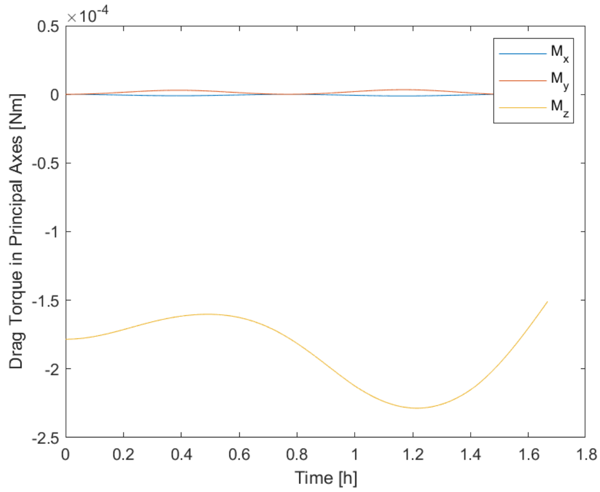


Figure 60: Numerical simulation of drag torques

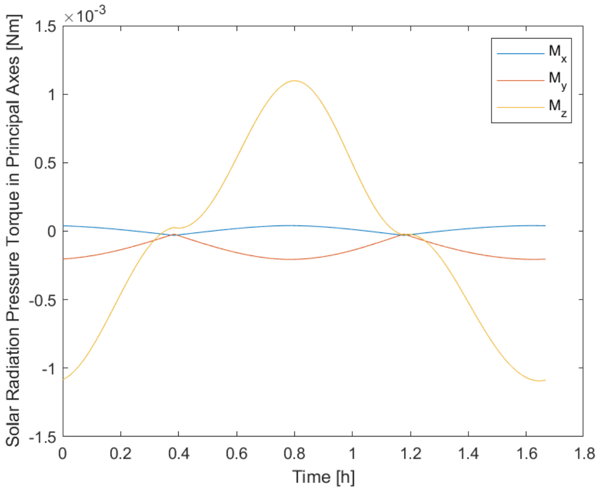


Figure 61: Numerical simulation of solar radiation pressure torques

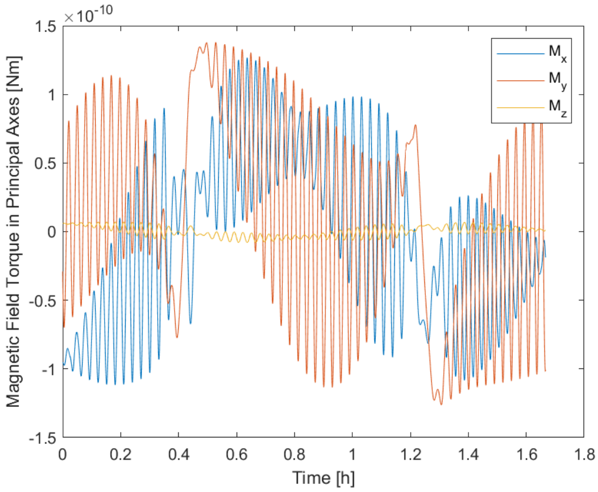


Figure 62: Numerical simulation of magnetic field torques

6 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.

[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.

[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

```
%% Center of mass
cm = computeCM('res/mass.csv');

%% Moment of inertia
origin = [0;0;0];
I = computeMOI('res/mass.csv',origin);

%% Surface properties
[barycenter,normal,area] = surfaces('res/area.csv');

%% Plot spacecraft with body axes
figure
gm = importGeometry('res/NISAR.stl');
pdegplot(gm);
quiver = findobj(gca,'type','Quiver');
textx = findobj(gca,'type','Text','String','x');
texty = findobj(gca,'type','Text','String','y');
textz = findobj(gca,'type','Text','String','z');
set(quiver,'XData',[0;0;0])
set(quiver,'YData',[0;0;0])
set(quiver,'ZData',[0;0;0])
set(textx,'Position',[4 0 0])
set(texty,'Position',[0 4 0])
set(textz,'Position',[0 0 4])
saveas(gcf,'Images/ps1_model.png');
```

A.2 Problem Set 2

```
%% Problem Set 2
clear; close all; clc;

%% Problem 1
a = 7125.48662; % km
e = 0.0011650;
i = 98.40508; % degree
O = -19.61601; % degree
w = 89.99764; % degree
nu = -89.99818; % degree

yECI = oe2eci(a,e,i,O,w,nu);

days = 0.5;
tspan = 0:days*86400;
options = odeset('RelTol',1e-6,'AbsTol',1e-9);
[t,y] = ode113(@orbitSimple,tspan,yECI,options);

plot3(y(:,1),y(:,2),y(:,3),'LineWidth',2,'Color','green')
xlabel('x [km]')
ylabel('y [km]')
zlabel('z [km]')
axis equal
hold on
[xE,yE,zE] = ellipsoid(0,0,0,6378.1,6378.1,6378.1,20);
surface(xE,yE,zE,'FaceColor','blue','EdgeColor','black');
hold off
saveas(gcf,'Images/ps2_problem1.png');

%% Problem 2
cm = computeCM('res/mass.csv');
I = computeMOI('res/mass.csv',cm);

[rot,IPrincipal] = eig(I);
Ix = IPrincipal(1,1);
Iy = IPrincipal(2,2);
Iz = IPrincipal(3,3);
xPrincipal = rot(:,1);
yPrincipal = rot(:,2);
zPrincipal = rot(:,3);

%% Problem 3
figure
gm = importGeometry('res/NISAR.stl');
pdegplot(gm);

quiver = findobj(gca,'type','Quiver');
textx = findobj(gca,'type','Text','String','x');
texty = findobj(gca,'type','Text','String','y');
textz = findobj(gca,'type','Text','String','z');
set(quiver,"XData",[0;0;0])
set(quiver,"YData",[0;0;0])
set(quiver,"ZData",[0;0;0])
set(textx,"Position",[4 0 0])
set(texty,"Position",[0 4 0])
```

```
set(textz,"Position",[0 0 4])

quiverPrincipal = copyobj(quiver,gca);
textxPrincipal = copyobj(textx,gca);
textyPrincipal = copyobj(texty,gca);
textzPrincipal = copyobj(textz,gca);
set(quiver,"Color",[0 1 0])
set(quiver,"UData",4.14 * rot(1,:)')
set(quiver,"VData",4.14 * rot(2,:)')
set(quiver,"WData",4.14 * rot(3,:)')
set(quiver,"XData",repmat(cm(1),3,1))
set(quiver,"YData",repmat(cm(2),3,1))
set(quiver,"ZData",repmat(cm(3),3,1))
set(textx,"String",'x''')
set(texty,"String",'y''')
set(textz,"String",'z''')
set(textx,"Position",4 * xPrincipal + cm)
set(texty,"Position",4 * yPrincipal + cm)
set(textz,"Position",4 * zPrincipal + cm)
saveas(gcf,'Images/ps2_model.png');

%% Problem 5
w0Deg = [8;4;6];
w0 = deg2rad(w0Deg);
tspan = 0:120;
w = eulerPropagator(w0,Ix,Iy,Iz,tspan,'Images/ps2_euler_equations.png');

%% Problem 6
[XE,YE,ZE] = ellipsoidEnergy(IPrincipal, ...
    w0, ...
    'Images/ps2_problem6_energy.png');
[XM,YM,ZM] = ellipsoidMomentum(IPrincipal, ...
    w0, ...
    'Images/ps2_problem6_momentum.png');

%% Problem 7
w = polhode(XE,YE,ZE,XM,YM,ZM,w,'Images/ps2_problem7.png');

%% Problem 8
w = polhode2D(w,'none','Images/ps2_problem8.png');

%% Problem 9, x-axis
axis = 'x';
w0Deg = 8*[1;0;0];
w0 = deg2rad(w0Deg);
tspan = 0:120;
marker = 'o';

%% Problem 9, y-axis
axis = 'y';
w0Deg = 8*[0.01;1;0.01];
w0 = deg2rad(w0Deg);
tspan = 0:1200;
marker = 'none';

%% Problem 9, z-axis
axis = 'z';
w0Deg = 8*[0.01;0;1];
```

```
w0 = deg2rad(w0Deg);
tspan = 0:120;
marker = 'none';

%% Problem 9
w = eulerPropagator(w0,Ix,Iy,Iz,tspan, ...
    ['Images/ps2_problem9_euler_equations_', axis, '.png']);

[XE,YE,ZE] = ellipsoidEnergy(IPrincipal,w0, ...
    ['Images/ps2_problem9_energy_', axis, '.png']);
[XM,YM,ZM] = ellipsoidMomentum(IPrincipal,w0, ...
    ['Images/ps2_problem9_momentum_', axis, '.png']);

w = polhode(XE,YE,ZE,XM,YM,ZM,w, ...
    ['Images/ps2_problem9_p7_', axis, '.png']);

w = polhode2D(w,marker, ...
    ['Images/ps2_problem9_p8_', axis, '.png']);
```

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

```
function w = eulerPropagator(w0,Ix,Iy,Iz,tspan,filename)
    options = odeset('RelTol',1e-6,'AbsTol',1e-9);
    [t,w] = ode113(@(t,w) eulerEquation(t,w,Ix,Iy,Iz),tspan,w0,options);
    wDeg = rad2deg(w);

    figure(1)
    plot(t,wDeg,'LineWidth',2)
    legend('\omega_{x}','\omega_{y}','\omega_{z}', ...
        'Location','southeast')
    xlabel('Time [s]')
    ylabel(['Angular velocity (\omega) [' char(176) '/s]'])
    saveas(1,filename)
end
```

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

```
    Ix = IPrincipal(1,1);
    Iy = IPrincipal(2,2);
    Iz = IPrincipal(3,3);
    T = sum(IPrincipal * w0.^2,"all") / 2;
    L = sqrt(sum((w0.*IPrincipal).^2,"all"));
    [XE,YE,ZE] = ...
        ellipsoid(0,0,0,sqrt(2*T/Ix),sqrt(2*T/Iy),sqrt(2*T/Iz),50);
    ellipsoidAxes = [sqrt(2*T/Ix), sqrt(2*T/Iy), sqrt(2*T/Iz)];

    % Plot energy ellipsoid
    figure(1)
    surf(XE,YE,ZE, ...
        'FaceAlpha',0.5, ...
        'FaceColor','blue', ...
        'DisplayName','Energy Ellipsoid');
    axis equal
    hold on
    quiver3(0, 0, 0, ellipsoidAxes(1), 0, 0, 'Color', 'r', ...
        'LineWidth', 2)
    quiver3(0, 0, 0, 0, ellipsoidAxes(2), 0, 'Color', 'r', ...
        'LineWidth', 2)
    quiver3(0, 0, 0, 0, 0, ellipsoidAxes(3), 'Color', 'r', ...
        'LineWidth', 2)
    xlabel('\omega_{x} [rad/s]')
    ylabel('\omega_{y} [rad/s]')
    zlabel('\omega_{z} [rad/s]')
    hold off
    saveas(1,filename)

    I = L^2/(2*T);
    if (Ix <= I || ismembertol(Ix, I, 1e-7)) && I <= Iz
        fprintf("The polhode is real!\n")
    else
        error("The polhode is NOT real!\n")
    end
end
```

```
function [XM,YM,ZM] = ellipsoidMomentum(IPrincipal,w0,filename)
    Ix = IPrincipal(1,1);
    Iy = IPrincipal(2,2);
    Iz = IPrincipal(3,3);
    T = sum(IPrincipal * w0.^2,"all") / 2;
    L = sqrt(sum((w0.*IPrincipal).^2,"all"));
    [XM,YM,ZM] = ellipsoid(0,0,0,L/Ix,L/Iy,L/Iz,50);
    momentumAxes = [L/Ix, L/Iy, L/Iz];

    % Plot momentum ellipsoid
    figure(1)
    surf(XM,YM,ZM, ...
        'FaceAlpha',0.5, ...
        'FaceColor','green', ...
        'DisplayName','Momentum Ellipsoid');
    axis equal
    hold on
    quiver3(0, 0, 0, momentumAxes(1), 0, 0, 'Color', 'r', ...
        'LineWidth', 2)
```

```
    quiver3(0, 0, 0, 0, momentumAxes(2), 0, 'Color', 'r', ...
        'LineWidth', 2)
    quiver3(0, 0, 0, 0, 0, momentumAxes(3), 'Color', 'r', ...
        'LineWidth', 2)
    xlabel('\omega_{x} [rad/s]')
    ylabel('\omega_{y} [rad/s]')
    zlabel('\omega_{z} [rad/s]')
    hold off
    saveas(1,filename)

    I = L^2/(2*T);
    if (Ix <= I || ismembertol(Ix, I, 1e-7)) && I <= Iz
        fprintf("The polhode is real!\n")
    else
        error("The polhode is NOT real!\n")
    end
end
```

```
function w = polhode(XE,YE,ZE,XM,YM,ZM,w,filename)
    figure(1)
    surf(XE,YE,ZE, ...
        'FaceAlpha',0.5, ...
        'FaceColor','blue', ...
        'DisplayName','Energy Ellipsoid');
    xlabel('\omega_{x} [rad/s]')
    ylabel('\omega_{y} [rad/s]')
    zlabel('\omega_{z} [rad/s]')
    axis equal
    hold on
    surf(XM,YM,ZM, ...
        'FaceAlpha',0.5, ...
        'FaceColor','green', ...
        'DisplayName','Momentum Ellipsoid');
    plot3(w(:,1),w(:,2),w(:,3), ...
        'LineWidth',2, ...
        'Color','red', ...
        'DisplayName','Polhode')
    legend('Location','northwest')
    hold off
    saveas(1,filename)
end
```

```
function w = polhode2D(w,marker,filename)
    subplot(1,3,1)
    plot(w(:,2),w(:,3),'Marker',marker)
    title('Polhode (along x-axis)')
    xlabel('\omega_{y} [rad/s]')
    ylabel('\omega_{z} [rad/s]')
    axis equal

    subplot(1,3,2)
    plot(w(:,1),w(:,3),'Marker',marker)
    title('Polhode (along y-axis)')
    xlabel('\omega_{x} [rad/s]')
    ylabel('\omega_{z} [rad/s]')
```

```
    axis equal

    subplot(1,3,3)
    plot(w(:,1),w(:,2),'Marker',marker)
    title('Polhode (along z-axis)')
    xlabel('\omega_{x} [rad/s]')
    ylabel('\omega_{y} [rad/s]')
    axis equal

    saveas(1,filename)
end
```

A.3 Problem Set 3

```matlab
clear; close all; clc

%% Problem 1
IPrincipal = [7707.07451493673 0 0; ...
    0 7707.0745149367 0; ...
    0 0 18050.0227594212];
Ix = IPrincipal(1,1);
Iy = IPrincipal(2,2);
Iz = IPrincipal(3,3);

w0Deg = [8;4;6];
w0 = deg2rad(w0Deg);
tspan = 0:0.1:120;
w = eulerPropagator(w0,Ix,Iy,Iz,tspan,'Images/ps3_problem1.png');

%% Problem 2
lambda = w0(3) * (Iz - Iy) / Ix;
wxy = (w0(1) + w0(2) * 1j) * exp(1j * lambda * tspan);
wx = real(wxy);
wy = imag(wxy);
wz = w0(3) * ones(size(wxy));
wAnalytical = [wx',wy',wz'];
wDegAnalytical = rad2deg(wAnalytical);

figure(1)
plot(tspan,wDegAnalytical,'LineWidth',2)
legend('\omega_{x}','\omega_{y}','\omega_{z}', ...
    'Location','southeast')
xlabel('Time [s]')
ylabel(['Angular velocity (\omega) [' char(176) '/s]'])
saveas(1,'Images/ps3_problem2.png')

%% Problem 3
% Error plots
error = w - wAnalytical;
plot(tspan,error,'LineWidth',2)
legend('\omega_{x}','\omega_{y}','\omega_{z}', ...
    'Location','southeast')
xlabel('Time [s]')
ylabel('Angular velocity (\omega) [rad/s]')
saveas(gcf,'Images/ps3_problem3.png')

% Verify L and omega
L_principal = [Ix Iy Iz] .* w;
keyTimes = [1, 61, 121, 181, 241, 361];
for n = keyTimes
    figure(1)
    L_unit = L_principal(n,:)/norm(L_principal(n,:));
    w_unit = w(n,:)/norm(w(n,:));
    quiver3(0,0,0,w_unit(1),w_unit(2),w_unit(3),1,'r')
    hold on
    quiver3(0,0,0,L_unit(1),L_unit(2),L_unit(3),1,'b')
    quiver3(0,0,0,0,0,1,'k')
    xlim([-1 1]); ylim([-1 1]); zlim([-1 1]);
    xlabel('x'); ylabel('y'); zlabel('z');
```

```
    legend('\omega','L','z-axis','Location','northeast')
    title(sprintf('Unit vectors at t = %.2f s', tspan(n)))
    hold off
    saveas(1, sprintf('Images/ps3_problem3_vectors_%i.png', n))
end

%% Non-Axisymmetric Satellite
cm = computeCM('res/mass.csv');
I = computeMOI('res/mass.csv',cm);

[rot,IPrincipal] = eig(I);
Ix = IPrincipal(1,1);
Iy = IPrincipal(2,2);
Iz = IPrincipal(3,3);
xPrincipal = rot(:,1);
yPrincipal = rot(:,2);
zPrincipal = rot(:,3);

%% Problem 6 (Quaternions)
axang0 = [sqrt(1/2) sqrt(1/2) 0 pi/4];
q0 = axang2quat(axang0).';
tFinal = 600;
tStep = 0.1;
t = 0:tStep:tFinal;

% Forward Euler
% [q,w] = kinQuaternionForwardEuler(q0,w0,Ix,Iy,Iz,tFinal,tStep);

% RK4
[q,w] = kinQuaternionRK4(q0,w0,Ix,Iy,Iz,tFinal,tStep);

figure(2)
hold on
plot(t,q,'LineWidth',1)
legend('q_{1}','q_{2}','q_{3}','q_{4}', ...
    'Location','Southeast')
xlabel('Time [s]')
ylabel('Quaternion')
hold off
saveas(2,'Images/ps3_problem6_quaternions.png')

%% Problem 6 (Euler Angles)
eulerAngle0 = rotm2eul(axang2rotm(axang0))';
state0 = [eulerAngle0;w0];

tFinal = 600;
tStep = 0.1;
t = 0:tStep:tFinal;

% Forward Euler
% state = kinEulerAngleForwardEuler(state0,Ix,Iy,Iz,tFinal,tStep);

% ode113
tspan = 0:tStep:tFinal;
options = odeset('RelTol',1e-6,'AbsTol',1e-9);
[t,state] = ode113(@(t,state) kinEulerAngle(t,state,Ix,Iy,Iz), ...
    tspan,state0,options);

```

```
eulerAngle = wrapTo360(rad2deg(state(:,1:3)));

figure(3)
plot(t,eulerAngle,'LineWidth',1)
legend('\phi','\theta','\psi', ...
    'Location','southwest')
xlabel('Time [s]')
ylabel('Euler Angle [deg]')
saveas(3,'Images/ps3_problem6_euler.png')

%% Problem 7(a)
% Part a: Angular momentum
tLen = length(t);
L_principal = [Ix Iy Iz] .* w;
L_inertial = nan(size(L_principal));
L_norm = nan(1, tLen);

% Part b: Herpolhode
w_inertial = nan(size(w));

for i = 1:tLen
    % Get rotation matrix
    qi = q(i,:);
    A = q2A(qi);

    % Angular momentum
    L_inertial(i,:) = A' * L_principal(i,:)';
    L_norm(i) = norm(L_inertial(i,:));

    % Angular velocity
    w_inertial(i,:) = A' * w(i,:)';
end

figure(4)
hold on
plot(t, L_inertial)
plot(t, L_norm, 'k--')
xlabel('Time [s]')
ylabel('Angular momentum [kg m^{2}/s]')
legend("L_{1}", "L_{2}", "L_{3}", "||L||")
hold off
saveas(4, 'Images/ps3_problem7a.png')

%% Problem 7(b)
figure(5)
plot3(w_inertial(:,1), w_inertial(:,2), w_inertial(:,3), 'r')
grid on
hold on
quiver3(0, 0, 0, ...
    L_inertial(1,1), L_inertial(1,2), L_inertial(1,3), ...
    1e-4)
quiver3(0, 0, 0, ...
    w_inertial(1,1), w_inertial(1,2), w_inertial(1,3), ...
    1)
xlabel('\omega_{x} [rad/s]')
ylabel('\omega_{y} [rad/s]')
zlabel('\omega_{z} [rad/s]')
legend('Herpolhode', ...
```

```
    'Angular momentum (L)', ...
    'Angular velocity (\omega)', ...
    'Location','northwest')
hold off
saveas(5, 'Images/ps3_problem7b.png')

%% For fun kinda thing
saveGif = true;
tGif = 240 / tStep;

L_unit = nan(size(L_inertial));
w_unit = nan(size(w_inertial));
if saveGif == true
    gif = figure;
    for i = 1:tLen
        w_unit(i,:) = w_inertial(i,:)./norm(w_inertial(i,:));
        L_unit(i,:) = L_inertial(i,:)./norm(L_inertial(i,:));
    end

    for i = 1:20:tGif
        plot3(w_unit(1:i,1), w_unit(1:i,2), w_unit(1:i,3), 'r')
        grid on
        hold on
        quiver3(0, 0, 0, w_unit(i,1), w_unit(i,2), w_unit(i,3),1)
        quiver3(0, 0, 0, L_unit(i,1), L_unit(i,2), L_unit(i,3),1)
        hold off
        xlim([-1 1])
        ylim([-1 1])
        zlim([-1 1])
        xlabel('x')
        ylabel('y')
        zlabel('z')
        title('Note: all vectors are normalized')
        legend('Herpolhode','\omega','L','Location','northeast')
        exportgraphics(gif,'Images/ps3_problem7b.gif','Append',true);
    end
end

%% Problem 7(c)
% Generate orbit
a = 7125.48662; % km
e = 0.0011650;
i = 98.40508; % degree
O = -19.61601; % degree
w = 89.99764; % degree
nu = -89.99818; % degree

days = 0.069;
tFinal = days * 86400;
tStep = 1;
tspan = 0:tStep:tFinal;

figure(6)
[t,y] = plotECI(a,e,i,O,w,nu,tspan);
hold on
figure(7)
plotECI(a,e,i,O,w,nu,tspan);
hold on
```

```
figure(8)
plotECI(a,e,i,O,w,nu,tspan);
hold on

[q,w] = kinQuaternionRK4(q0,w0,Ix,Iy,Iz,tFinal,tStep);

tLen = length(t);
for i = 1:500:tLen
    % Get rotation matrix
    qi = q(i,:);
    A = q2A(qi);
    % Body axes
    B = rot * A * rot';
    % Position
    pos = y(i,1:3);
    radial = pos / norm(pos);
    tangential = y(i,4:6) / norm(y(i,4:6));
    normal = cross(radial,tangential);
    RTN = [radial' tangential' normal'];
    figure(6);
    plotTriad(gca,pos,A,1e3,'r');
    figure(7);
    plotTriad(gca,pos,B,1e3,'m');
    figure(8);
    plotTriad(gca,pos,RTN,1e3,'b');
end
figure(6);
legend('Orbit','Earth','Principal (x-axis)','Location','northwest')
hold off
saveas(gcf,'Images/ps3_problem7c_principal.png');
figure(7);
legend('Orbit','Earth','Body (x-axis)','Location','northwest')
hold off
saveas(gcf,'Images/ps3_problem7c_body.png');
figure(8);
legend('Orbit','Earth','RTN (radial axis)','Location','northwest')
hold off
saveas(gcf,'Images/ps3_problem7c_rtn.png');
```

```
function M = plotTriad(ax,o,M,scale,colorString)
    quiver3(ax, ...
        o(1),o(2),o(3), ...
        M(1,1),M(2,1),M(3,1), ...
        scale, ...
        'LineWidth',1, ...
        'Color','k')
    quiver3(ax, ...
        o(1),o(2),o(3), ...
        M(1,2),M(2,2),M(3,2), ...
        scale, ...
        'LineWidth',1, ...
        'Color',colorString)
    quiver3(ax, ...
        o(1),o(2),o(3), ...
        M(1,3),M(2,3),M(3,3), ...
        scale, ...
```

```
        'LineWidth',1, ...
        'Color',colorString)
end
```

A.4 Problem Set 4

```
close all; clear; clc
savePlot = true;

%% Import mass properties
cm = computeCM('res/mass.csv');
I = computeMOI('res/mass.csv',cm);

[rot,IPrincipal] = eig(I);
Ix = IPrincipal(1,1);
Iy = IPrincipal(2,2);
Iz = IPrincipal(3,3);

%% Problem 1(a)
tFinal = 60;
tStep = 0.01;
tspan = 0:tStep:tFinal;

eulerAngle0 = [0; 0; 0];
w0 = [0; 0; 1];
state0 = [eulerAngle0;w0];

options = odeset('RelTol',1e-6,'AbsTol',1e-9);
[t,state] = ode113(@(t,state) kinEulerAngle(t,state,Ix,Iy,Iz), ...
    tspan,state0,options);

state = rad2deg(state);

% Plot
figure()
plot(t,state(:,4:6),'LineWidth',1)
legend('\omega_{x}','\omega_{y}','\omega_{z}', ...
    'Location','southeast')
xlabel('Time [s]')
ylabel(['Angular Velocity (\omega) [' char(176) '/s]'])
if savePlot
    saveas(gcf,'Images/ps4_problem1a_angvel.png')
end

figure()
plot(t,wrapTo180(state(:,1:3)),'LineWidth',1)
legend('\phi','\theta','\psi', ...
    'Location','southwest')
xlabel('Time [s]')
ylabel(['Euler Angle [' char(176) ']'])
if savePlot
    saveas(gcf,'Images/ps4_problem1a_angle.png')
end

%% Problem 1(b)
a = 7125.48662; % km
e = 0;
i = 98.40508; % degree
O = -19.61601; % degree
w_deg = 89.99764; % degree
nu = -89.99818; % degree
```

```
muE = 3.986 * 10^5;

n = sqrt(muE / a^3);

tFinal = 6000;
tStep = 0.1;
tspan = 0:tStep:tFinal;
tTrunc = 300;
nTrunc = find((tspan == tTrunc) == 1);

[~,y] = plotECI(a,e,i,O,w_deg,nu,tspan);
close all
format long

% Initialize angular velocity aligned with normal
r0 = y(1,1:3);
v0 = y(1,4:6);
h = cross(r0,v0);
radial = r0 / norm(r0);
normal = h / norm(h);
tangential = cross(normal,radial);
A_RTN = [radial' tangential' normal']';
w0_RTN = [0; 0; 0.1];
euler0_RTN = A2e(A_RTN);
state0 = [euler0_RTN; w0_RTN];
options = odeset('RelTol',1e-6,'AbsTol',1e-9);
[t,state] = ode113(@(t,state) kinEulerAngle(t,state,Ix,Iy,Iz), ...
    tspan,state0,options);

w_RTN = nan(size(state(:,1:3)));
euler_RTN = nan(size(state(:,1:3)));

for n = 1:length(t)
    pos = y(n,1:3);
    vel = y(n,4:6);
    h = cross(pos,vel);
    radial = pos / norm(pos);
    normal = h / norm(h);
    tangential = cross(normal,radial);
    tangential = tangential / norm(tangential);
    A_RTN = [radial' tangential' normal']';

    % Get rotation matrixes (to ECI)
    euler = state(n,1:3);
    w_principal = state(n,4:6)';
    A_principal = e2A(euler);
    A_P2R = A_RTN * A_principal';
    w_RTN(n,:) = A_P2R*w_principal;
    euler_RTN(n,:) = A2e(A_P2R');
end

figure()
plot(t, rad2deg(w_RTN))
xlabel('Time [s]')
ylabel(['Angular Velocity, RTN Frame [' char(176) '/s]'])
legend('\omega_{R}','\omega_{T}','\omega_{N}')
if savePlot == true
```

```
    saveas(gcf,'Images/ps4_problem1b_angvel.png')
end

figure()
plot(t(1:nTrunc), wrapTo180(rad2deg(euler_RTN(1:nTrunc,:))))
xlabel('Time [s]')
ylabel(['Euler Angle, RTN Frame [' char(176) ']'])
legend('\phi','\theta','\psi')
if savePlot == true
    saveas(gcf,'Images/ps4_problem1b_euler.png')
end

%% Problem 2
% Initial conditions
perturbation = 0.001;
w0x = [1; perturbation; perturbation];
w0y = [perturbation; 1; perturbation];
w0z = [perturbation; perturbation; 1];

w0Mat = {w0x, w0y, w0z};
eulerAngle0 = [0; 0; 0];
tStep = 0.01;
tFinal = 60;

for n = 1:3
    w0 = w0Mat{n};
    state0 = [eulerAngle0;w0];

    tspan = 0:tStep:tFinal;
    options = odeset('RelTol',1e-6,'AbsTol',1e-9);
    [t,state] = ode113(@(t,state) kinEulerAngle(t,state,Ix,Iy,Iz), ...
        tspan,state0,options);

    eulerAngle = wrapTo180(rad2deg(state(:,1:3)));
    w = rad2deg(state(:,4:6));

    figure()
    subplot(2,1,1)
    plot(t,w)
    xlabel('Time [s]')
    ylabel(['Angular Velocity [' char(176) '/s]'])
    legend('\omega_{x}','\omega_{y}','\omega_{z}', ...
        'Location','Southeast')

    subplot(2,1,2)
    plot(t,eulerAngle)
    xlabel('Time [s]')
    ylabel(['Euler Angle [' char(176) ']'])
    legend('\phi','\theta','\psi', 'Location','Southeast')
    if savePlot
        saveas(gcf,['Images/ps4_problem2a_' sprintf('%i',n) '.png'])
    end
end

%% Momentum wheel setup
% Based on RSI 68
mr = 8.9; % kg
r = 0.347 / 2; % m
```

```
Ir = mr * r^2;
wrRPM = 2500; % RPM
wr = wrRPM * 0.1047198;

% Initial Euler angle
eulerAngle0 = [0; 0; 0];

% No external torques
M = [0; 0; 0; 0];

% Time
tFinal = 300;
tStep = 0.1;

%% Problem 3(b)
w0 = [0.01; 0.01; 0.01; wr];
r = [0; 0; 1];

namePlot = 'Images/ps4_problem3b.png';
plotPS4Problem3(eulerAngle0,w0,tStep,tFinal, ...
                M,r,Ix,Iy,Iz,Ir,namePlot,savePlot);

%% Problem 3(c)
w0 = [0.1; 0.001; 0.001; wr + 0.001 * rand()];
r = [1; 0; 0];
namePlot = 'Images/ps4_problem3c_x.png';
plotPS4Problem3(eulerAngle0,w0,tStep,tFinal, ...
                M,r,Ix,Iy,Iz,Ir,namePlot,savePlot);

w0 = [0.001; 0.1; 0.001; wr + 0.001 * rand()];
r = [0; 1; 0];
namePlot = 'Images/ps4_problem3c_y.png';
plotPS4Problem3(eulerAngle0,w0,tStep,tFinal, ...
                M,r,Ix,Iy,Iz,Ir,namePlot,savePlot);

w0 = [0.001; 0.001; 0.1; wr + 0.001 * rand()];
r = [0; 0; 1];
namePlot = 'Images/ps4_problem3c_z.png';
plotPS4Problem3(eulerAngle0,w0,tStep,tFinal, ...
                M,r,Ix,Iy,Iz,Ir,namePlot,savePlot);

%% Problem 3(d)
w0 = [0.001; 0.1; 0.001; wr * 10];
r = [0; 1; 0];

namePlot = 'Images/ps4_problem3d.png';
plotPS4Problem3(eulerAngle0,w0,tStep,tFinal, ...
                M,r,Ix,Iy,Iz,Ir,namePlot,savePlot);

%% Problem 3(e)
w0 = [rot' * [0.1; 0.001; 0.001]; 0];
r = rot' * [1; 0; 0];

namePlot = 'Images/ps4_problem3e_unstable.png';
plotPS4Problem3(eulerAngle0,w0,tStep,tFinal, ...
                M,r,Ix,Iy,Iz,Ir,namePlot,savePlot);

w0 = [rot' * [0.1; 0.001; 0.001]; wr * 10];
```

```
namePlot = 'Images/ps4_problem3e_stable.png';
plotPS4Problem3(eulerAngle0,w0,tStep,tFinal, ...
                M,r,Ix,Iy,Iz,Ir,namePlot,savePlot);

%% Problem 4(d)
% Should put this into a function and call it for (d-e)
tFinal = 6000;
tStep = 1;
tspan = 0:tStep:tFinal;

a = 7125.48662; % km
e = 0;
i = 98.40508; % degree
O = -19.61601; % degree
w = 89.99764; % degree
nu = -89.99818; % degree
muE = 3.986 * 10^5;
n = sqrt(muE / a^3);

y = oe2eci(a,e,i,O,w,nu);
r0 = y(1:3);
v0 = y(4:6);
h = cross(r0,v0);
radial = r0 / norm(r0);
normal = h / norm(h);
tangential = cross(normal,radial);
A_RTN = [radial tangential normal]';

state0 = zeros(12,1);
state0(1:6) = y;
state0(7:9) = [0; 0; n];
state0(10:12) = A2e(A_RTN);

options = odeset('RelTol',1e-6,'AbsTol',1e-9);
[t,state] = ode113(@(t,state) gravGrad(t,state,Ix,Iy,Iz,n), ...
        tspan,state0,options);

c = zeros(size(state(:,1:3)));
M = zeros(size(state(:,1:3)));
for i = 1:length(t)
    r = state(i,1:3);
    radial = r / norm(r);
    A_ECI2P = e2A(state(i,10:12));
    c(i,1:3) = A_ECI2P * radial';
    M(i,1:3) = gravGradTorque(Ix,Iy,Iz,n,c(i,1:3));
end

figure()
plot(t,M)
xlabel('Time [s]')
ylabel('Torque in Principal Axes [Nm]')
legend('M_{x}','M_{y}','M_{z}')
ylim([-1e-5 1e-5])
if savePlot == true
    saveas(gcf,'Images/ps4_problem4d_torque.png')
end
```

```
figure()
plot(t,state(:,7:9))
xlabel('Time [s]')
ylabel('Angular Velocity in Principal Axes [rad/s]')
legend('\omega_{x}','\omega_{y}','\omega_{z}')
if savePlot == true
    saveas(gcf,'Images/ps4_problem4d_angvel.png')
end

%% Problem 4(e)
tFinal = 6000;
tStep = 1;
tspan = 0:tStep:tFinal;

a = 7125.48662; % km
e = 0;
i = 98.40508; % degree
O = -19.61601; % degree
w = 89.99764; % degree
nu = -89.99818; % degree
muE = 3.986 * 10^5;
n = sqrt(muE / a^3);

y = oe2eci(a,e,i,O,w,nu);
r0 = y(1:3);
v0 = y(4:6);
h = cross(r0,v0);
radial = r0 / norm(r0);
normal = h / norm(h);
tangential = cross(normal,radial);
A_RTN = [radial tangential normal]';
A_Body = rot' * A_RTN;

state0 = zeros(12,1);
state0(1:6) = y;
state0(7:9) = [0; 0; n];
state0(10:12) = A2e(A_Body);

options = odeset('RelTol',1e-6,'AbsTol',1e-9);
[t,state] = ode113(@(t,state) gravGrad(t,state,Ix,Iy,Iz,n), ...
        tspan,state0,options);

c = zeros(size(state(:,1:3)));
M = zeros(size(state(:,1:3)));
for i = 1:length(t)
    r = state(i,1:3);
    radial = r / norm(r);
    A_ECI2P = e2A(state(i,10:12));
    c(i,1:3) = A_ECI2P * radial';
    M(i,1:3) = gravGradTorque(Ix,Iy,Iz,n,c(i,1:3));
end

figure()
plot(t,M)
xlabel('Time [s]')
ylabel('Torque in Principal Axes [Nm]')
legend('M_{x}','M_{y}','M_{z}')
if savePlot == true
```

```
    saveas(gcf,'Images/ps4_problem4e_torque.png')
end

figure()
plot(t,state(:,7:9))
xlabel('Time [s]')
ylabel('Angular Velocity in Principal Axes [rad/s]')
legend('\omega_{x}','\omega_{y}','\omega_{z}')
if savePlot == true
    saveas(gcf,'Images/ps4_problem4e_angvel.png')
end

figure()
plot(t,wrapTo180(rad2deg(state(:,10:12))))
xlabel('Time [s]')
ylabel(['Euler Angles [' char(176) ']'])
legend('\phi','\theta','\psi')
if savePlot == true
    saveas(gcf,'Images/ps4_problem4e_angle.png')
end
```

A.5 Problem Set 5

```
close all; clear; clc
savePlot = true;

%% Import mass properties
cm = computeCM('res/mass.csv');
I = computeMOI('res/mass.csv',cm);

[rot,IPrincipal] = eig(I);
Ix = IPrincipal(1,1);
Iy = IPrincipal(2,2);
Iz = IPrincipal(3,3);

%% Problem 1(a)
IR = Ix;
IT = Iz;
IN = Iy;

kT = (IN - IR) / IT;
kR = (IN - IT) / IR;

plotGravGradStability(kR,kT,'Nominal','Images/ps5_problem1a.png');

%% Problem 1(b) (Unstable, Unperturbed)
tFinal = 6000 * 5; % 5 orbits
tStep = 1;
tspan = 0:tStep:tFinal;

a = 7125.48662; % km
e = 0;
i = 98.40508; % degree
O = -19.61601; % degree
w = 89.99764; % degree
nu = -89.99818; % degree
muE = 3.986 * 10^5;
n = sqrt(muE / a^3);

y = oe2eci(a,e,i,O,w,nu);
r0 = y(1:3);
v0 = y(4:6);
h = cross(r0,v0);
radial = r0 / norm(r0);
normal = h / norm(h);
tangential = cross(normal,radial);
A_Nominal = [-radial -normal -tangential]';

state0 = zeros(12,1);
state0(1:6) = y;
state0(7:9) = [0; -n; 0];
state0(10:12) = A2e(A_Nominal);

options = odeset('RelTol',1e-6,'AbsTol',1e-9);
[t,state] = ode113(@(t,state) gravGrad(t,state,Ix,Iy,Iz,n), ...
        tspan,state0,options);

c = zeros(size(state(:,1:3)));
```

```
M = zeros(size(state(:,1:3)));
for i = 1:length(t)
    r = state(i,1:3);
    radial = r / norm(r);
    A_ECI2P = e2A(state(i,10:12));
    c(i,1:3) = A_ECI2P * radial';
    M(i,1:3) = gravGradTorque(Ix,Iy,Iz,n,c(i,1:3));
end

figure()
plot(t / 3600,state(:,7:9))
xlabel('Time [h]')
ylabel('Angular Velocity in Principal Axes [rad/s]')
legend('\omega_{x}','\omega_{y}','\omega_{z}')
if savePlot == true
    saveas(gcf,'Images/ps5_problem1b_angvel_unperturbed.png')
end

figure()
plot(t / 3600,wrapToPi(state(:,10:12)))
xlabel('Time [h]')
ylabel('Euler Angles in Principal Axes [rad]')
legend('\phi','\theta','\psi')
if savePlot == true
    saveas(gcf,'Images/ps5_problem1b_angle_unperturbed.png')
end

%% Problem 1(b) (Unstable, Perturbed)
tFinal = 6000 * 3; % 2 orbits
tStep = 1;
tspan = 0:tStep:tFinal;

a = 7125.48662; % km
e = 0;
i = 98.40508; % degree
O = -19.61601; % degree
w = 89.99764; % degree
nu = -89.99818; % degree
muE = 3.986 * 10^5;
n = sqrt(muE / a^3);

y = oe2eci(a,e,i,O,w,nu);
r0 = y(1:3);
v0 = y(4:6);
h = cross(r0,v0);
radial = r0 / norm(r0);
normal = h / norm(h);
tangential = cross(normal,radial);
A_Nominal = [-radial -normal -tangential]';

state0 = zeros(12,1);
state0(1:6) = y;
state0(7:9) = [0; -n; 0] * 1.01;
state0(10:12) = A2e(A_Nominal) + pi * [0.01; 0.01; 0.01];

options = odeset('RelTol',1e-6,'AbsTol',1e-9);
[t,state] = ode113(@(t,state) gravGrad(t,state,Ix,Iy,Iz,n), ...
        tspan,state0,options);
```

```
c = zeros(size(state(:,1:3)));
M = zeros(size(state(:,1:3)));
for i = 1:length(t)
    r = state(i,1:3);
    radial = r / norm(r);
    A_ECI2P = e2A(state(i,10:12));
    c(i,1:3) = A_ECI2P * radial';
    M(i,1:3) = gravGradTorque(Ix,Iy,Iz,n,c(i,1:3));
end

figure()
plot(t / 3600,state(:,7:9))
xlabel('Time [h]')
ylabel('Angular Velocity in Principal Axes [rad/s]')
legend('\omega_{x}','\omega_{y}','\omega_{z}')
if savePlot == true
    saveas(gcf,'Images/ps5_problem1b_angvel.png')
end

figure()
plot(t / 3600,wrapToPi(state(:,10:12)))
xlabel('Time [h]')
ylabel('Euler Angles in Principal Axes [rad]')
legend('\phi','\theta','\psi')
if savePlot == true
    saveas(gcf,'Images/ps5_problem1b_angle.png')
end

%% Problem 1(c)
IR = Ix;
IT = Iy;
IN = Iz;
kT = (IN - IR) / IT;
kR = (IN - IT) / IR;

plotGravGradStability(kR,kT, ...
    'Principal XYZ aligned with RTN', ...
    'Images/ps5_problem1c.png');

%% Problem 1(c) (Stable, Perturbed)
tFinal = 6000 * 10; % 10 orbits
tStep = 1;
tspan = 0:tStep:tFinal;

a = 7125.48662; % km
e = 0;
i = 98.40508; % degree
O = -19.61601; % degree
w = 89.99764; % degree
nu = -89.99818; % degree
muE = 3.986 * 10^5;
n = sqrt(muE / a^3);

y = oe2eci(a,e,i,O,w,nu);
r0 = y(1:3);
v0 = y(4:6);
h = cross(r0,v0);
```

```
radial = r0 / norm(r0);
normal = h / norm(h);
tangential = cross(normal,radial);
A_RTN = [radial tangential normal]';

state0 = zeros(12,1);
state0(1:6) = y;
state0(7:9) = [0; 0; n] * 1.01;
state0(10:12) = A2e(A_RTN) + pi * [0.01; 0.01; 0.01];

options = odeset('RelTol',1e-6,'AbsTol',1e-9);
[t,state] = ode113(@(t,state) gravGrad(t,state,Ix,Iy,Iz,n), ...
        tspan,state0,options);

c = zeros(size(state(:,1:3)));
M = zeros(size(state(:,1:3)));
for i = 1:length(t)
    r = state(i,1:3);
    radial = r / norm(r);
    A_ECI2P = e2A(state(i,10:12));
    c(i,1:3) = A_ECI2P * radial';
    M(i,1:3) = gravGradTorque(Ix,Iy,Iz,n,c(i,1:3));
end

figure()
plot(t / 3600,state(:,7:9))
xlabel('Time [h]')
ylabel('Angular Velocity in Principal Axes [rad/s]')
legend('\omega_{x}','\omega_{y}','\omega_{z}')
if savePlot == true
    saveas(gcf,'Images/ps5_problem1c_angvel.png')
end

figure()
plot(t / 3600,wrapToPi(state(:,10:12)))
xlabel('Time [h]')
ylabel('Euler Angles in Principal Axes [rad]')
legend('\phi','\theta','\psi')
if savePlot == true
    saveas(gcf,'Images/ps5_problem1c_angle.png')
end

%% Problem 3
tFinal = 6000;
tStep = 1;
tspan = 0:tStep:tFinal;

% Satellite orbit initial conditions
a = 7125.48662; % km
e = 0;
i = 98.40508; % degree
O = -19.61601; % degree
w = 89.99764; % degree
nu = -89.99818; % degree
muE = 3.986 * 10^5; % km^3 / s^2
n = sqrt(muE / a^3);

% Compute initial position and attitude
```

```
y = oe2eci(a,e,i,O,w,nu);
r0 = y(1:3);
v0 = y(4:6);
h = cross(r0,v0);
radial = r0 / norm(r0);
normal = h / norm(h);
tangential = cross(normal,radial);
A_RTN = [radial tangential normal]';

% Earth orbit initial conditions
aE = 149.60E6; % km
eE = 0.0167086;
iE = 7.155; % degree
OE = 174.9; % degree
wE = 288.1; % degree
nuE = 0;
muSun = 1.327E11; % km^3 / s^2
nE = sqrt(muSun / aE^3);
ySun = oe2eci(aE,eE,iE,OE,wE,nuE);

% Initial conditions
state0 = zeros(12,1);
state0(1:6) = y;
state0(7:9) = [0; 0; n];
state0(10:12) = A2e(A_RTN);
state0(13:18) = ySun;

% Properties
[barycenter,normal,area] = surfaces('res/area.csv',rot');
cm = computeCM('res/mass.csv');
I = computeMOI('res/mass.csv',cm);
[rot,~] = eig(I);
cmP = rot' * cm;

% Parameters
CD = 2;
Cd = 0; Cs = 0.9;
P = 1358/3E8;
S_sat = 24.92;
m_max = 4*pi*1e-7 * S_sat * 0.1;
m_direction_body = [1; 0; 0];
m_direction = rot * m_direction_body;
m = m_max*m_direction/norm(m_direction); % Arbitrarily defined ...
    satellite dipole for now
UT1 = [2024 1 1];

% Run numerical method
options = odeset('RelTol',1e-6,'AbsTol',1e-9);
[t,state] = ode113(@(t,state) orbitTorque(t,state,Ix,Iy,Iz, ...
    CD,Cd,Cs,P,m,UT1, ...
    barycenter,normal,area,cmP,n), ...
    tspan,state0,options);

% Compute torques (since ode113 does not allow returning these)
Rx = [1 0 0; 0 cosd(23.5) -sind(23.5); 0 sind(23.5) cosd(23.5)];
c = zeros(size(state(:,1:3)));
Mgg = zeros(size(state(:,1:3)));
Md = zeros(size(state(:,1:3)));
```

```
Msrp = zeros(size(state(:,1:3)));
Mm = zeros(size(state(:,1:3)));

for i = 1:length(t)
    r = state(i,1:3)';
    v = state(i,1:3)';
    radial = r / norm(r);
    rEarth = state(i,13:15)';
    A_ECI2P = e2A(state(i,10:12));

    c(i,1:3) = A_ECI2P * radial;
    Mgg(i,1:3) = gravGradTorque(Ix,Iy,Iz,n,c(i,1:3));

    [~,density] = atmosnrlmsise00(1000 * (norm(r) - ...
        6378.1),0,0,2000,1,0);
    rho = density(6);
    vPrincipal = A_ECI2P * (v + cross([0; 0; 7.2921159E-5],r));
    [~,M] = drag(vPrincipal,rho,CD,barycenter,normal,area,cmP);
    Md(i,1:3) = M;

    s = A_ECI2P * (-Rx * rEarth - r);
    [~,M] = srp(s,P,Cd,Cs,barycenter,normal,area,cmP);
    Msrp(i,1:3) = M;

    M = magFieldTorque(m,r,state(i,10:12),t(i),6378.1,UT1);
    Mm(i,1:3) = M;
end

figure()
plot(t / 3600,state(:,7:9))
xlabel('Time [h]')
ylabel('Angular Velocity in Principal Axes [rad/s]')
legend('\omega_{x}','\omega_{y}','\omega_{z}')
if savePlot == true
    saveas(gcf,'Images/ps5_problem3_angvel.png')
end

figure()
plot(t / 3600,Mgg)
xlabel('Time [h]')
ylabel('Gravity Gradient Torque in Principal Axes [Nm]')
legend('M_{x}','M_{y}','M_{z}')
if savePlot == true
    saveas(gcf,'Images/ps5_problem3_grav.png')
end

figure()
plot(t / 3600,Md)
xlabel('Time [h]')
ylabel('Drag Torque in Principal Axes [Nm]')
legend('M_{x}','M_{y}','M_{z}')
if savePlot == true
    saveas(gcf,'Images/ps5_problem3_drag.png')
end

figure()
plot(t / 3600,Msrp)
xlabel('Time [h]')
```

```
ylabel('Solar Radiation Pressure Torque in Principal Axes [Nm]')
legend('M_{x}','M_{y}','M_{z}')
if savePlot == true
    saveas(gcf,'Images/ps5_problem3_srp.png')
end

figure()
plot(t / 3600,Mm)
xlabel('Time [h]')
ylabel('Magnetic Field Torque in Principal Axes [Nm]')
legend('M_{x}','M_{y}','M_{z}')
if savePlot == true
    saveas(gcf,'Images/ps5_problem3_mag.png')
end

%% Problem 3 Maximum Torques
% Parameters
CD = 2;
Cd = 0; Cs = 0.9;
q = Cd + Cs;
P = 1358/3E8;
S_sat = 24.92;
m_max = 4*pi*1e-7 * S_sat * 0.1;
UT1 = [2024 1 1];
rE3_B0 = 7.943e15; %Wb*km

r_norm = norm(state(1,1:3));
Mgg_max = 3/2 * muE/(r_norm^3) * abs(max([Ix Iy Iz]) - min([Ix Iy Iz]));
Mm_max = 2*m_max*rE3_B0/((r_norm*1e3)^3);
Msrp_max = 0;
Md_max = 0;

vMax = max(vecnorm(state(:,4:6)')) * 1e3;

for n = 1:length(area)
    Msrp_max = Msrp_max + P*area(n)*(1+q) * norm(barycenter(:,n) - cmP);
    Md_max = Md_max + 0.5*rho*CD*vMax^2*area(n) * ...
        norm(barycenter(:,n) - cmP);
end

fprintf("Maximum expected values: \n" + ...
        "M_gg: %d Nm \n" + ...
        "M_m: %d Nm \n" + ...
        "M_srp: %d Nm \n" + ...
        "M_d: %d Nm\n", Mgg_max, Mm_max, Msrp_max, Md_max);
```

```
function [kR,kT] = plotGravGradStability(kR,kT,nameText,namePlot)
    % Gather points
    fimplicit(@(x,y) 1 + 3 * x + y * x + 4 * sqrt(y * x),[-1,1,-1,1])
    leftBranch = findobj(gcf,'Type','ImplicitFunctionLine');
    xLeft = leftBranch.XData;
    yLeft = leftBranch.YData;
    close()
    fimplicit(@(x,y) 1 + 3 * x + y * x - 4 * sqrt(y * x),[-1,1,-1,1])
    rightBranch = findobj(gcf,'Type','ImplicitFunctionLine');
    xRight = rightBranch.XData;
```

```
    yRight = rightBranch.YData;
    close()
    hold on

    % Plot yaw, roll unstable
    plot(polyshape([0 0 1 1],[-1 0 0 -1]), ...
        'FaceAlpha',1, ...
        'FaceColor','b', ...
        'DisplayName','Unstable yaw, roll')
    plot(polyshape([xRight(27:end) -1],[yRight(27:end) -1]), ...
        'FaceAlpha',1, ...
        'FaceColor','b', ...
        'HandleVisibility','off')

    % Plot pitch unstable
    plot(polyshape([0 1 0],[0 1 1]), ...
        'FaceAlpha',1, ...
        'FaceColor','y', ...
        'DisplayName','Unstable pitch')
    plot(polyshape([xLeft -1],[yLeft 0]), ...
        'FaceAlpha',1, ...
        'FaceColor','y', ...
        'HandleVisibility','off')
    plot(polyshape([xRight(1:27) 0],[yRight(1:27) 0]), ...
        'FaceAlpha',1, ...
        'FaceColor','y', ...
        'HandleVisibility','off')

    % Plot yaw, roll, pitch unstable
    plot(polyshape([-1 -1 0 0],[0 1 1 0]), ...
        'FaceAlpha',1, ...
        'FaceColor','g', ...
        'DisplayName', ...
        'Unstable yaw, roll, pitch')
    plot(polyshape([flip(xRight(1:27)) xLeft -1], ...
        [flip(yRight(1:27)) yLeft -1]), ...
        'FaceAlpha',1, ...
        'FaceColor','g', ...
        'HandleVisibility','off')

    % Plot spacecraft location
    plot(kT,kR,'x','Color','k','LineWidth',2,'DisplayName',nameText)

    axis equal
    legend()
    xlabel('k_{T}')
    ylabel('k_{R}')
    xlim([-1 1])
    ylim([-1 1])
    hold off
    saveas(gcf,namePlot)
end
```

```
function [B_R,B_theta,B_phi] = magFieldEarth(R,phi,theta,RE)
    % NOTE: slides calls phi lambda (not sure if it's the same thing ...
       or not)
```

```

    % Make sure that R is normalized
    if length(R) == 3
        R = norm(R);
    end

    % For g & h matrix (row = n, col = m + 1)
    g = [-30186 -2036 0 0 0; ...
        -1898 2997 1551 0 0; ...
        1299 -2144 1296 805 0; ...
        951 807 462 -393 235] * 1e-9; % T
    h = [0 5735 0 0 0; ...
        0 -2124 -37 0 0; ...
        0 -361 249 -253 0; ...
        0 148 -264 37 -307] * 1e-9; % T

    B_R = 0;
    B_theta = 0;
    B_phi = 0;
    for n = 1:4
        BR_temp = 0;
        BTheta_temp = 0;
        BPhi_temp = 0;

        for mInd = 1:n
            m = mInd - 1;

            P_nm = getPnm(theta,n,m);
            dPnm_dtheta = getdPdTheta(theta,n,m);

            BR_temp = BR_temp + ...
                (g(n,mInd) * cos(m * phi) + h(n,mInd) * sin(m * ...
                   phi)) * ...
                P_nm;
            BTheta_temp = BTheta_temp + ...
                (g(n,mInd) * cos(m * phi)+ h(n,mInd) * sin(m * phi)) ...
                   * ...
                dPnm_dtheta;
            BPhi_temp = BPhi_temp + ...
                (-g(n,mInd) * sin(m * phi) + h(n,mInd) * cos(m * ...
                   phi)) * ...
                m * P_nm;
        end

        B_R = B_R + (RE / R)^(n + 2) * (n + 1) * BR_temp;
        B_theta = B_theta + (RE / R)^(n + 2) * BTheta_temp;
        B_phi = B_phi + (RE / R)^(n + 2) * BPhi_temp;
    end

    B_theta = -B_theta;
    B_phi = -1 / sin(theta) * B_phi;

end
```

```
function dPdTheta = getdPdTheta(theta,n,m)
    if n < m
```

```
        error("n >= m for Legendre functions")
    end
    if n == m && m == 0
        dPdTheta = 0;
    elseif n == m
        dPdTheta = sin(theta) * getdPdTheta(theta,n-1,n-1) + ...
            cos(theta) * getPnm(theta,n-1,n-1);
    else
        K = getKnm(n,m);
        if K == 0
            dPdTheta = cos(theta) * getdPdTheta(theta,n-1,m) - ...
                sin(theta) * getPnm(theta,n-1,m);
        else
            dPdTheta = cos(theta) * getdPdTheta(theta,n-1,m) - ...
                sin(theta) * getPnm(theta,n-1,m) - ...
                K * getdPdTheta(theta,n-2,m);
        end
    end
end
```

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

```
function P = getPnm(theta,n,m)
    if n == 0 && m == 0
        P = 1;
    elseif n == m
        P = sin(theta) * getPnm(theta,n-1,n-1);
    else
        K = getKnm(n,m);
        if K == 0
            P = cos(theta) * getPnm(theta,n-1,m);
        else
            P = cos(theta) * getPnm(theta,n-1,m) - K * ...
               getPnm(theta,n-2,m);
        end
    end
end
```