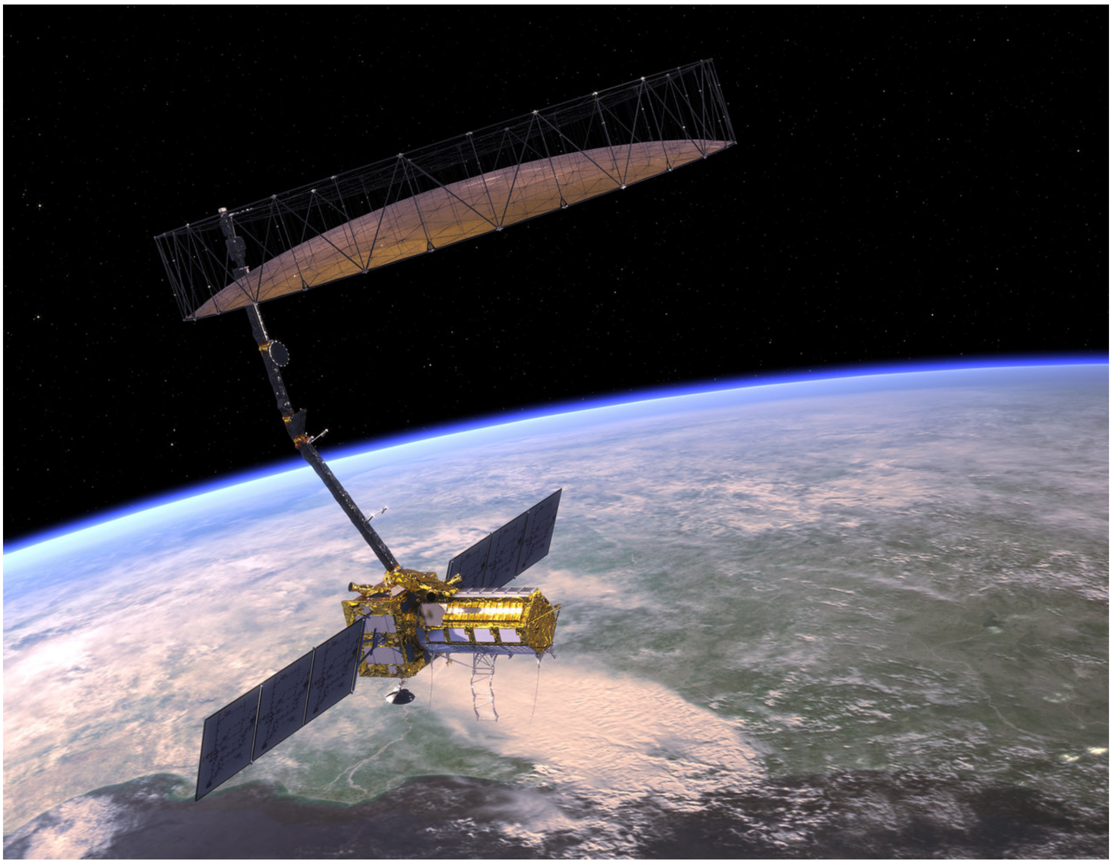
SATELLITE DYNAMICS AND ATTITUDE CONTROL

Kusal Uprety, Harry Zhao

24 April 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

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.

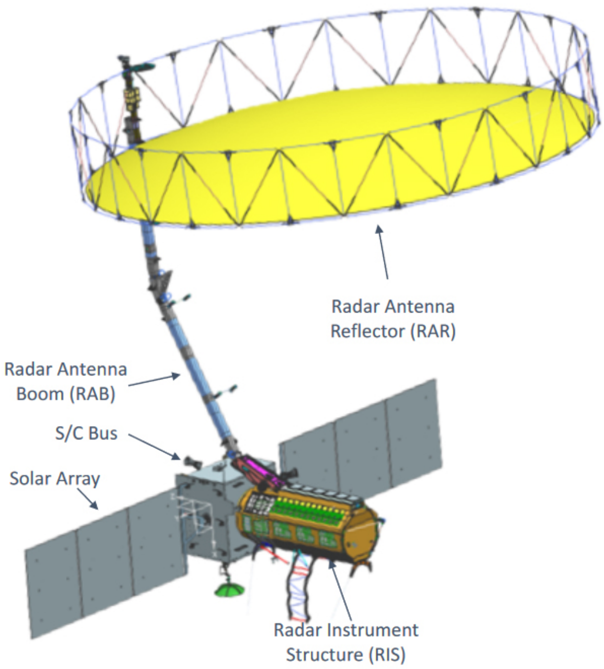


Figure 1: The basic components of the NISAR satellite

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 $[1.046, 0, 0.683]\,\mathrm{m}$ 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.

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

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:

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

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.

```
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)
    data = readmatrix(filename);
    barycenter = data(:,1:3);
    normal = 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.

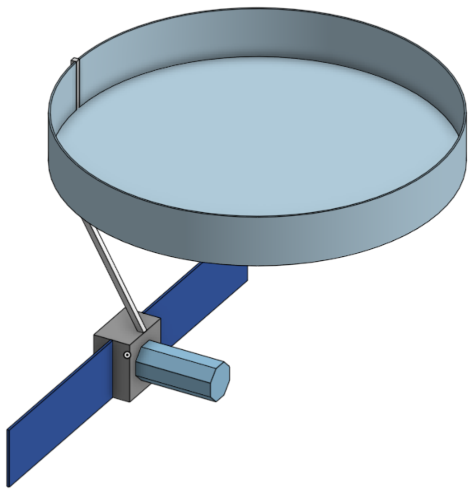


Figure 2: A 3D model of the satellite

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.

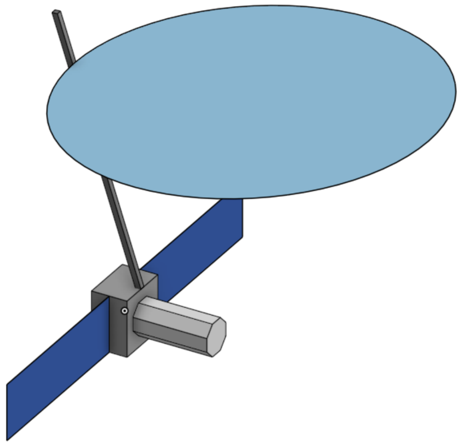


Figure 3: A 3D model of the simplified satellite geometry

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.

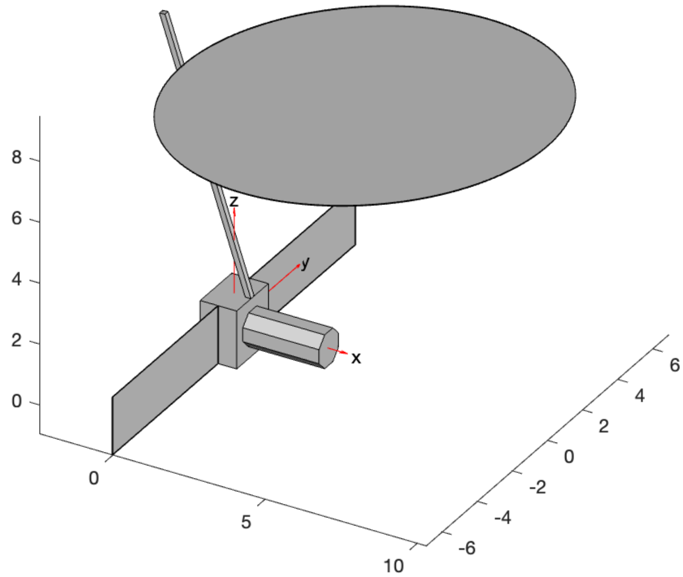


Figure 4: Satellite model in MATLAB with body axes shown

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.

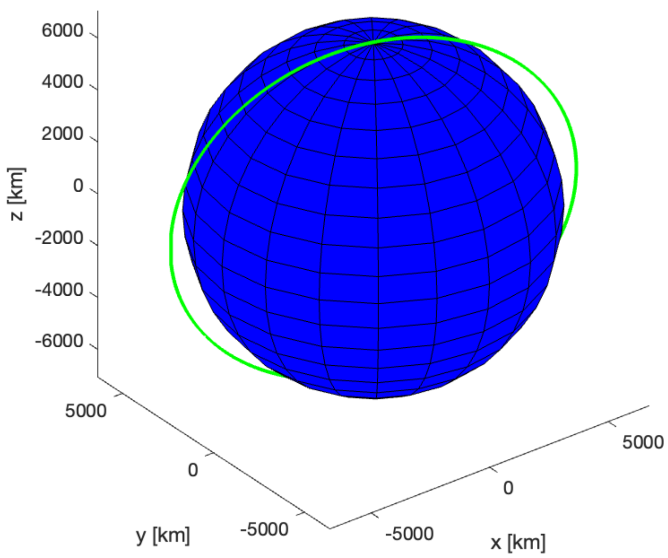


Figure 5: A single orbit for NISAR in ECI coordinates (no perturbations)

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.

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

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

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.

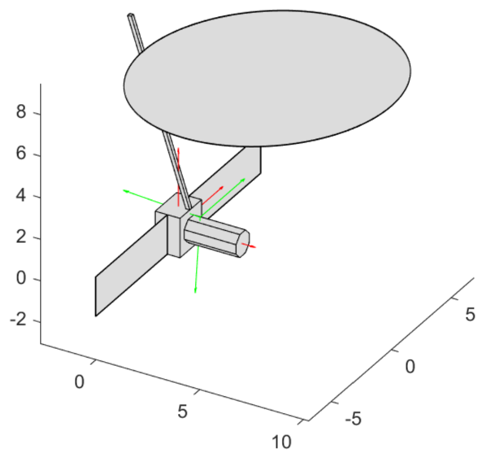


Figure 6: Principal axes at center of mass (green) and body axes at origin (red)

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

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

```
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 ($\omega < 10\,^\circ/\mathrm{s}$, $\omega_i \neq 0$). Multiple attitude revolutions.

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

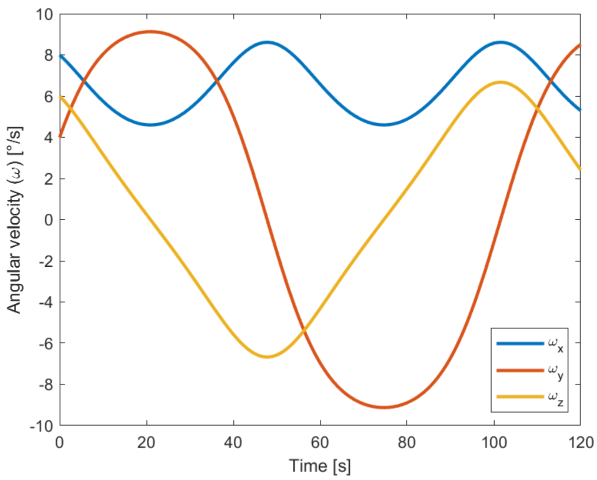


Figure 7: Results from numerical integration of Euler equations

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.

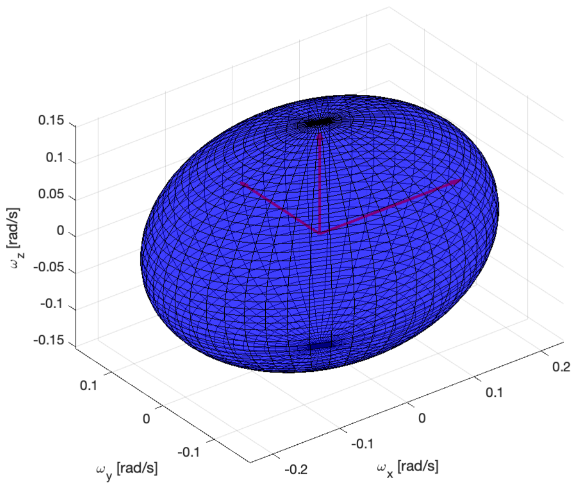


Figure 8: Energy ellipsoid with axes in red

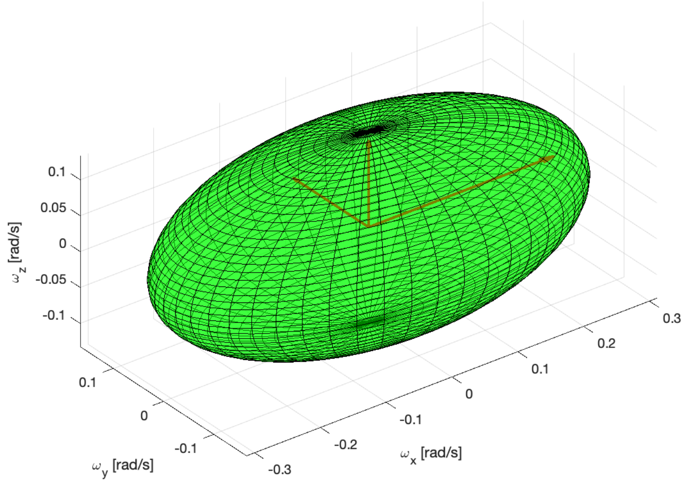


Figure 9: Momentum ellipsoid with axes in red

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.

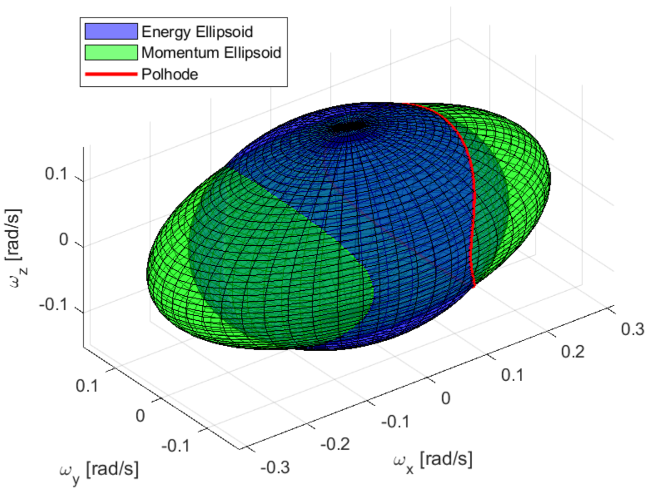


Figure 10: Energy and momentum ellipsoids with polhode

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.

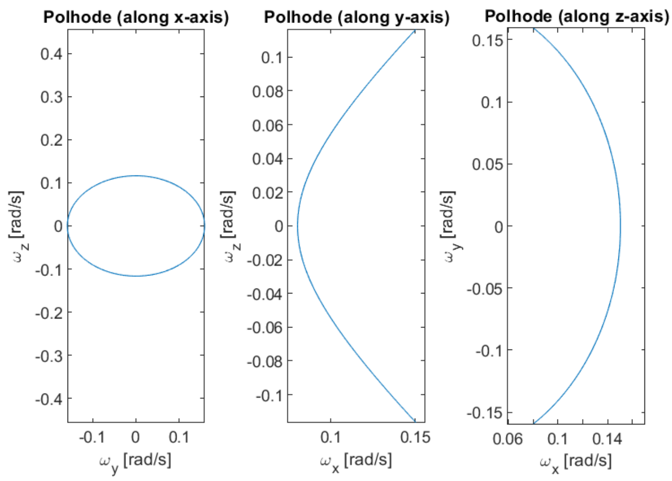


Figure 11: 2D views of polhode

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.

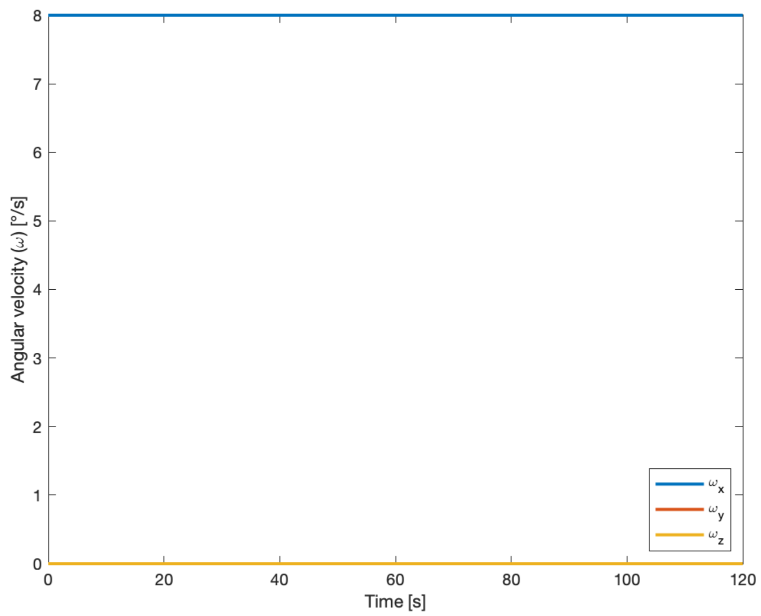


Figure 12: Angular velocity evolution for angular velocity vector for Case 1

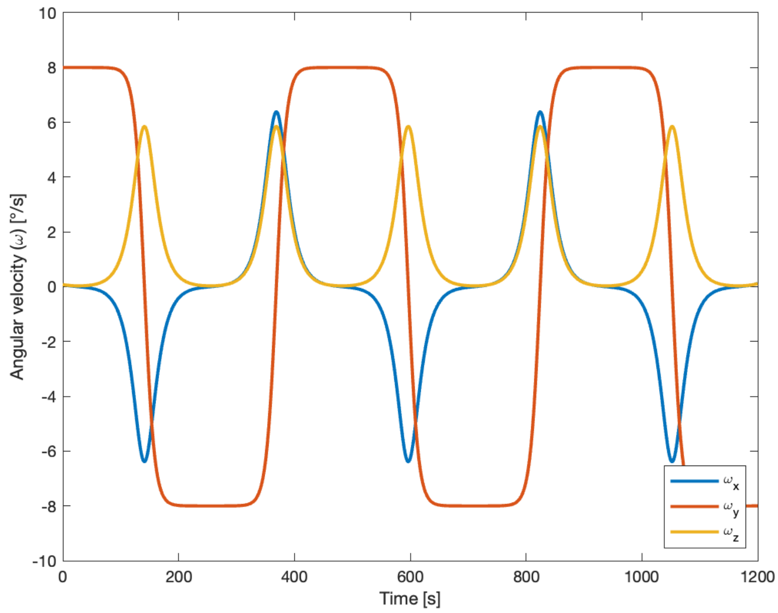


Figure 13: Angular velocity evolution for angular velocity vector for Case 2

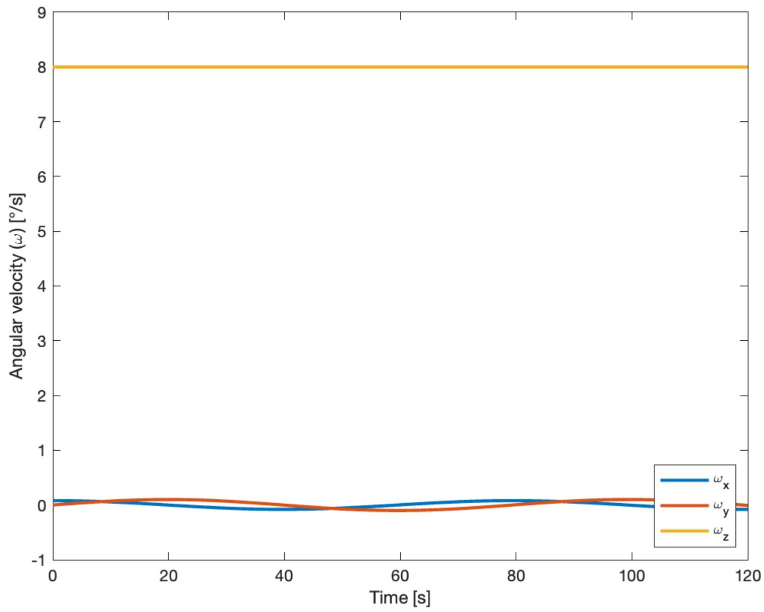


Figure 14: Angular velocity evolution for angular velocity vector for Case 3

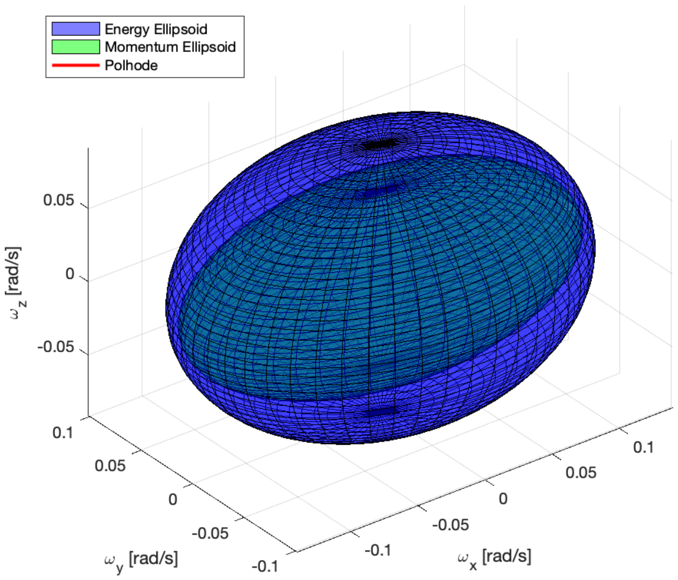


Figure 15: Polhode and ellipsoids for angular velocity vector for Case 1

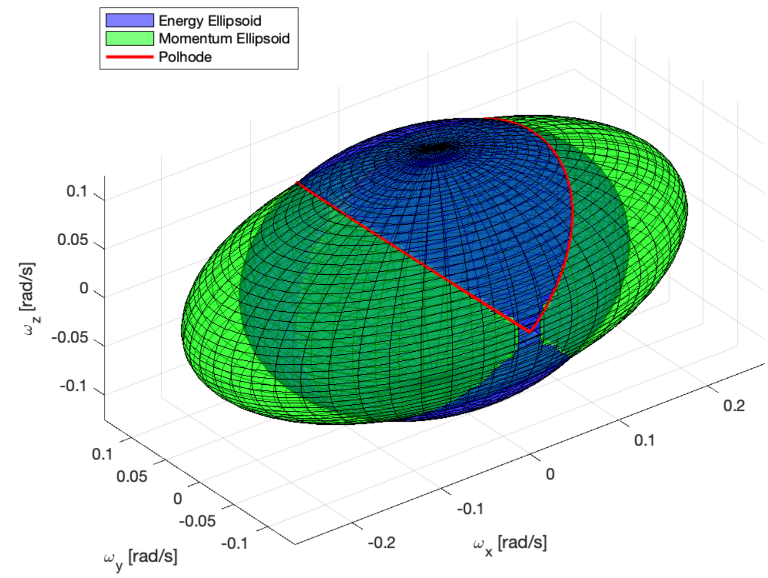


Figure 16: Polhode and ellipsoids for angular velocity vector for Case 2

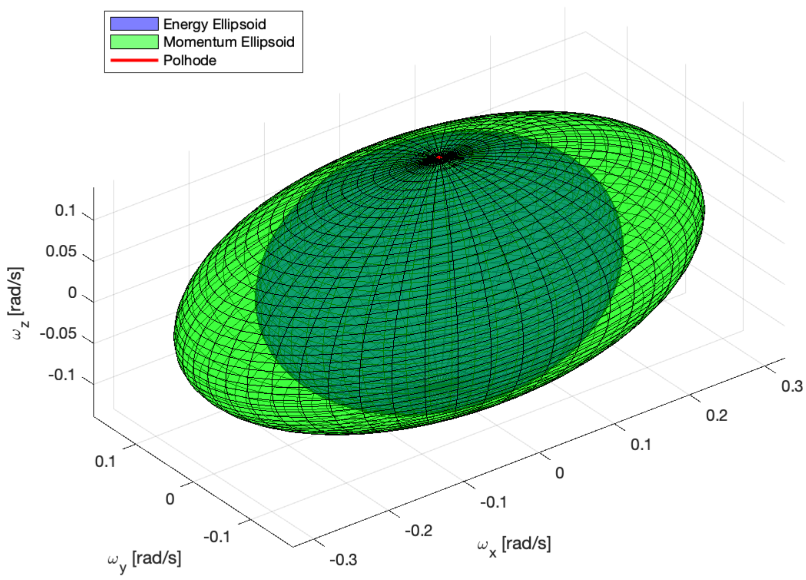


Figure 17: Polhode and ellipsoids for angular velocity vector for Case 3

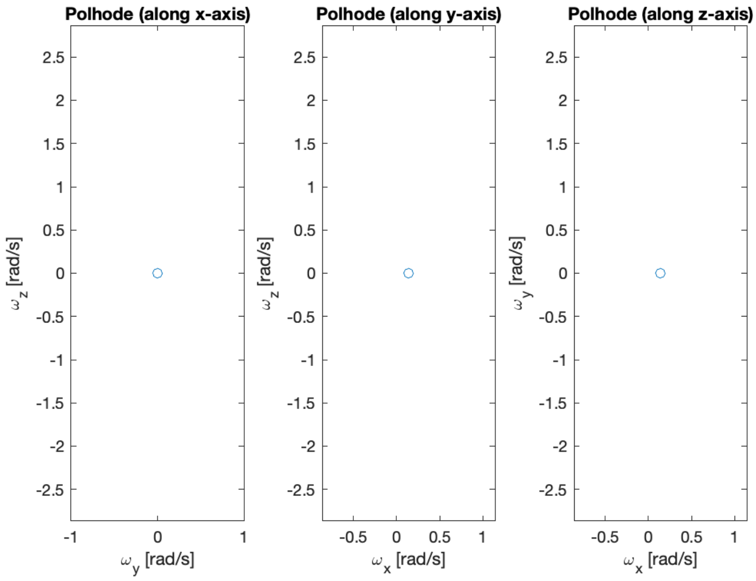


Figure 18: 2D views of polhode for angular velocity vector for Case 1

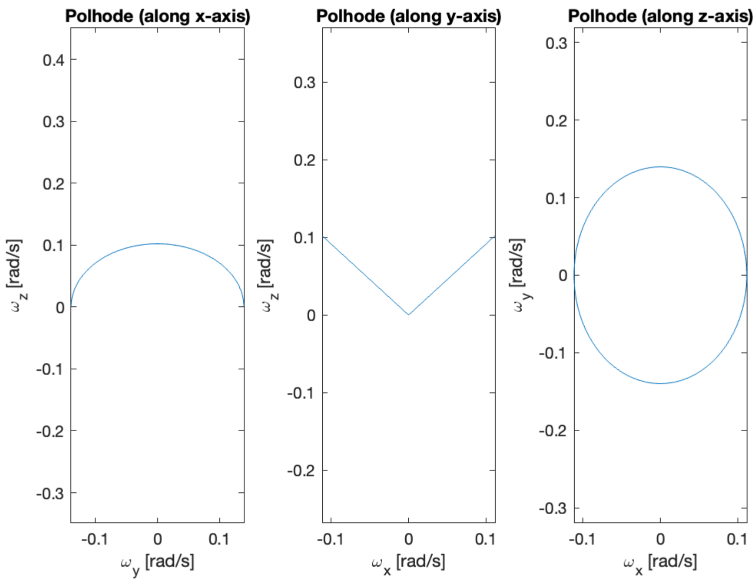


Figure 19: 2D views of polhode for angular velocity vector for Case 1

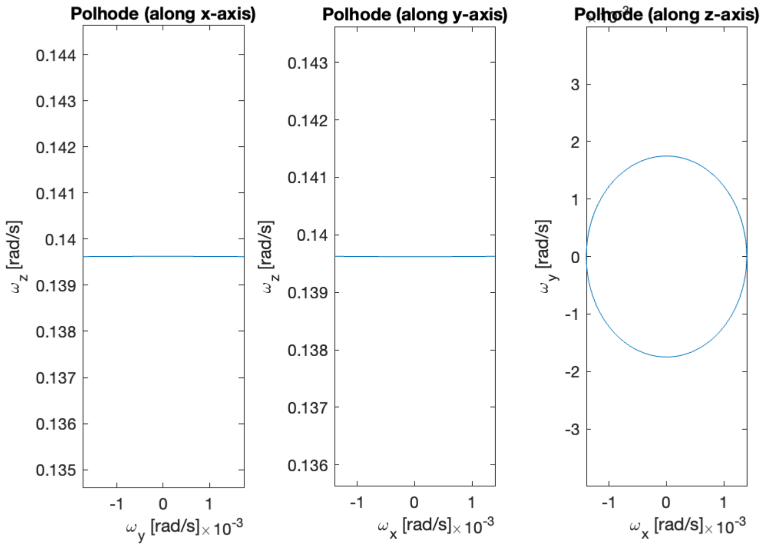


Figure 20: 2D views of polhode for angular velocity vector for Case 1

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

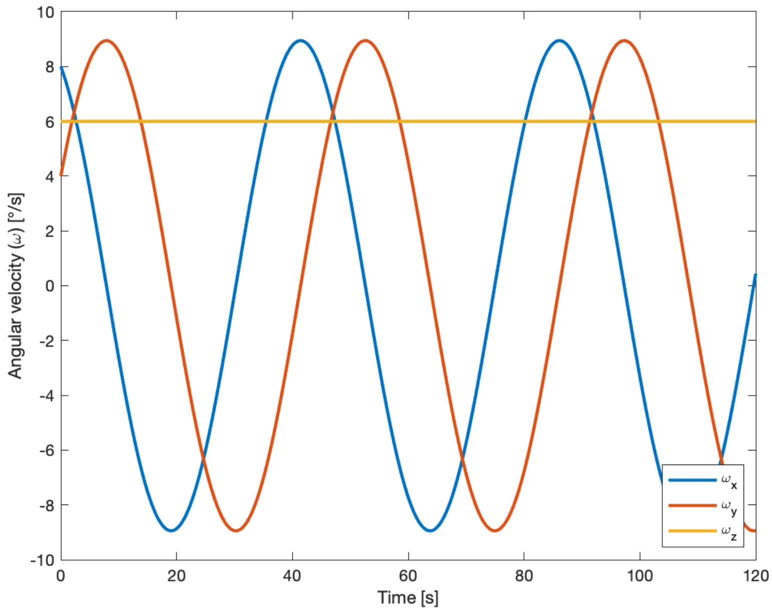


Figure 21: Numerical solution results

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.

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

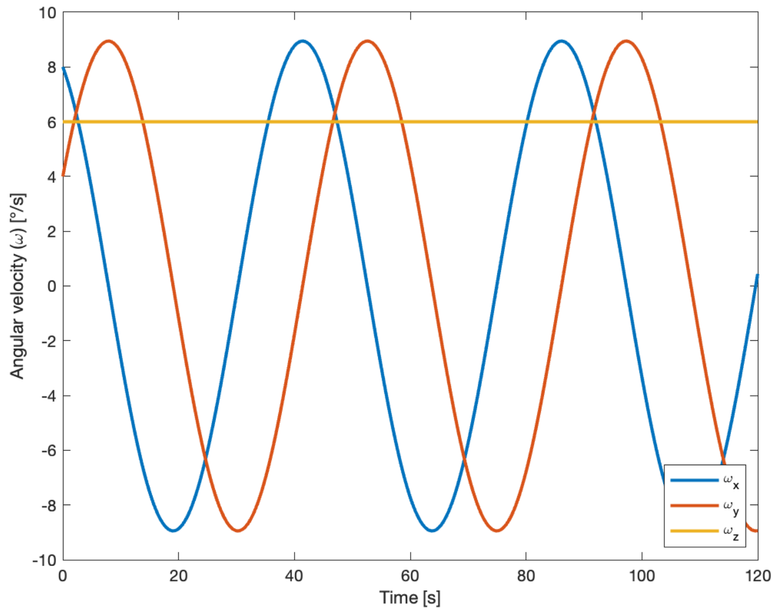


Figure 22: Analytical solution results

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.

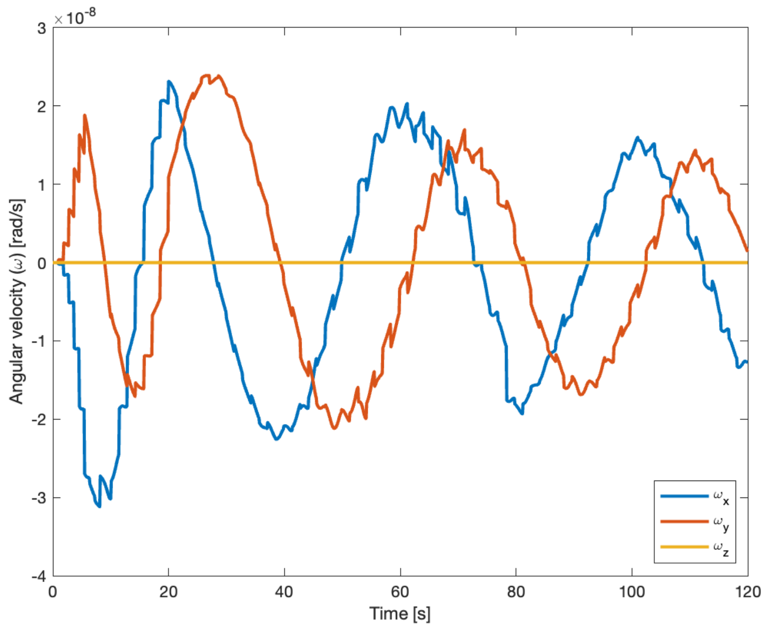


Figure 23: Error between numerical and analytical solutions

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.

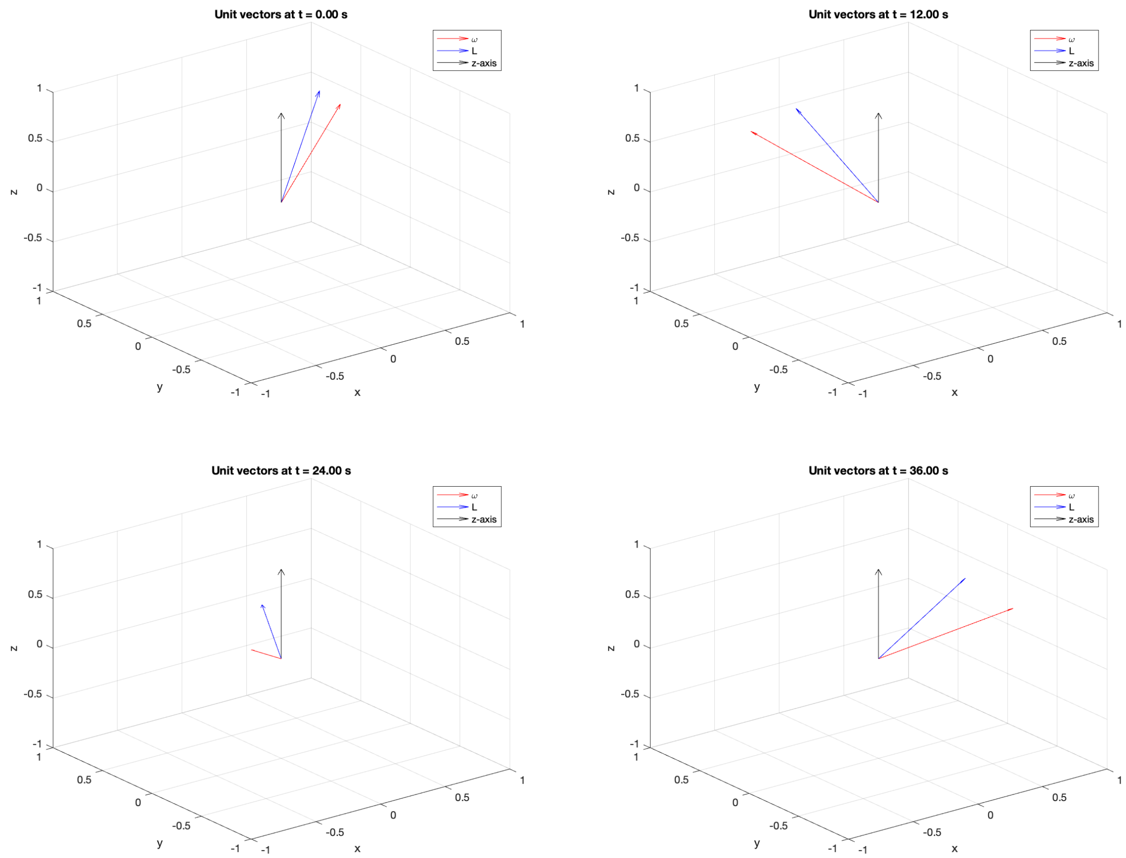


Figure 24: Angular velocity (red) and angular momentum (blue) unit vectors over time.

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
    theta = state(2);
    psi = state(3);
    wx = state(4);
    wy = state(5);
    wz = state(6);

    stateDot = zeros(6,1);
    % Angular velocity time derivatives
    stateDot(4) = (Iy - Iz) / Ix * wy * wz;
    stateDot(5) = (Iz - Ix) / Iy * wz * wx;
    stateDot(6) = (Ix - Iy) / Iz * wx * wy;
    % Euler angle time derivatives
    stateDot(1) = (wx*sin(psi) + wy*cos(psi))/sin(theta);
    stateDot(2) = wx*cos(psi) - wy*sin(psi);
    stateDot(3) = wz - (wx*sin(psi) + wy*cos(psi))*cot(theta);
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).

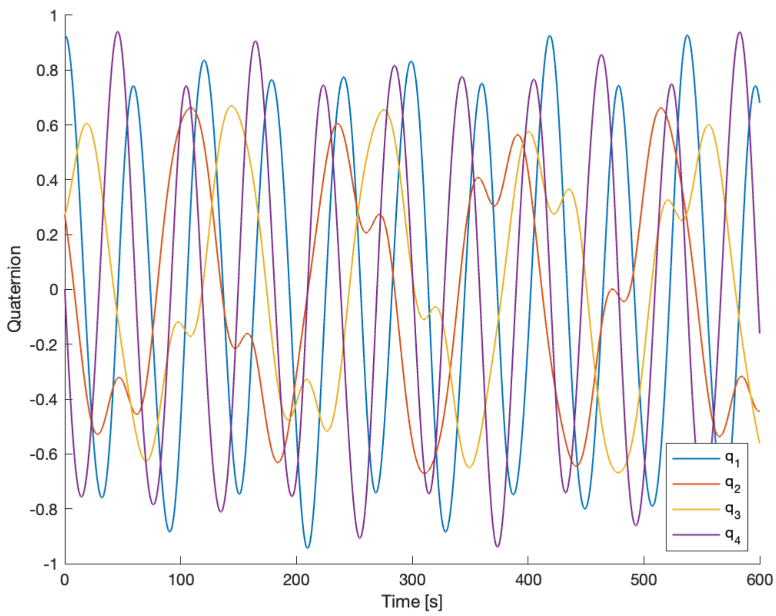


Figure 25: Evolution of quaternions

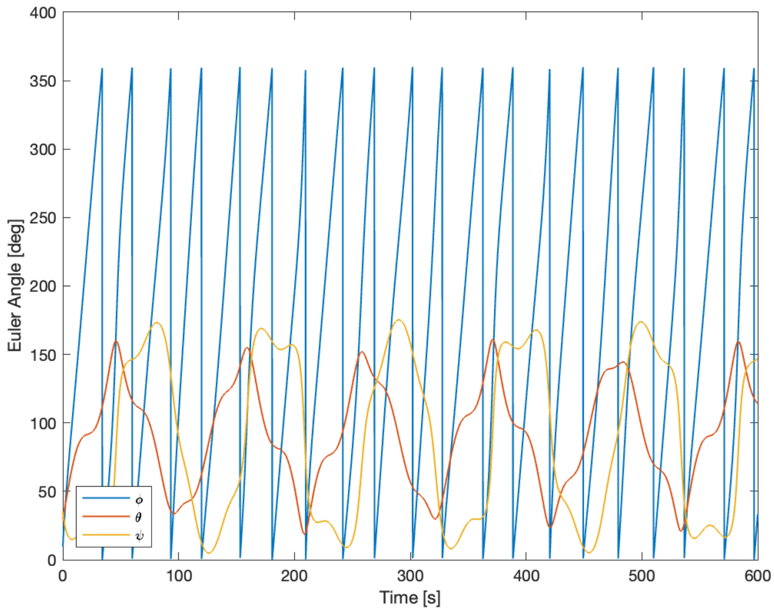


Figure 26: Evolution of Euler angles

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.

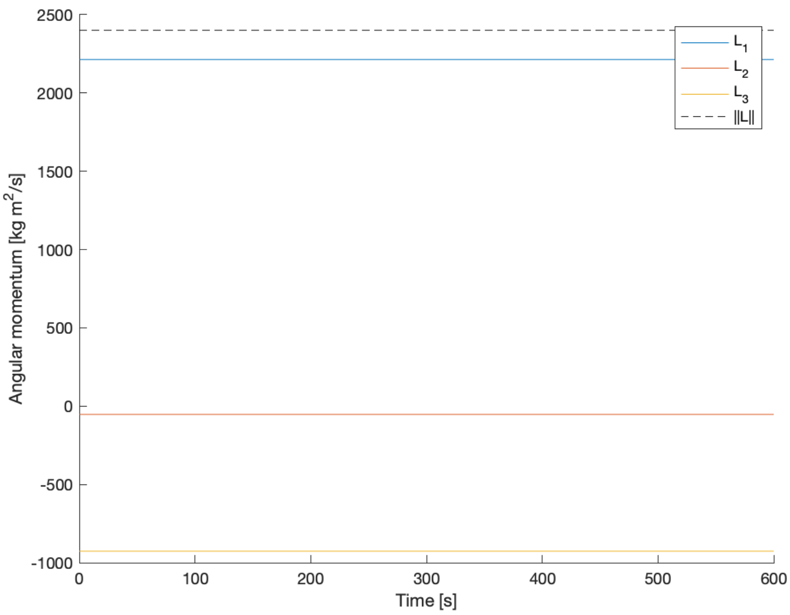


Figure 27: Angular momentum in inertial coordinates is constant

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.

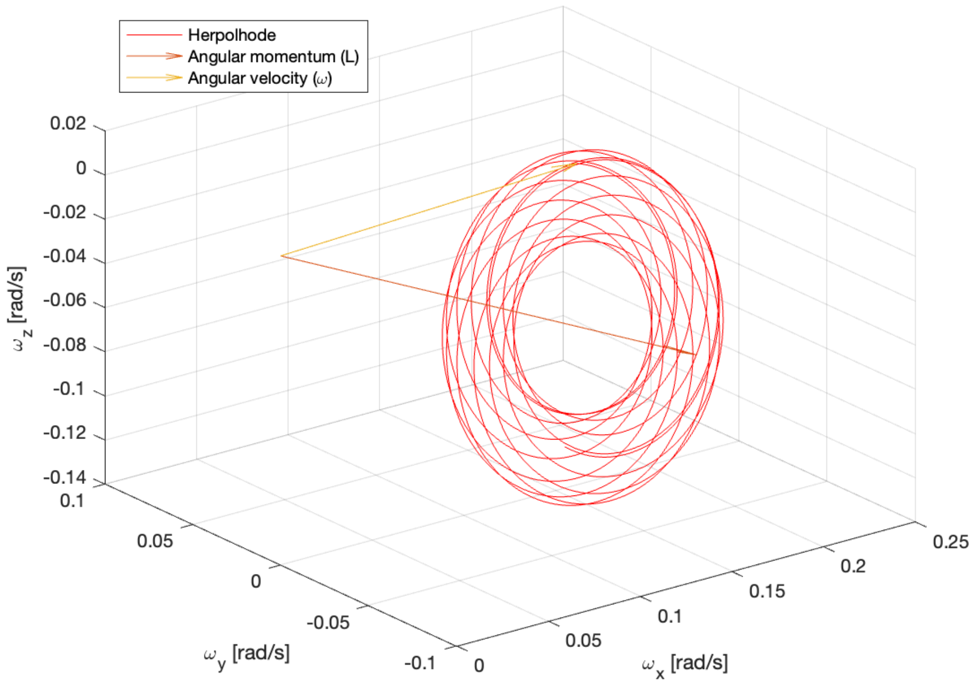


Figure 28: Herpolhode (Animated: https://tinyurl.com/herpolhode)

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.

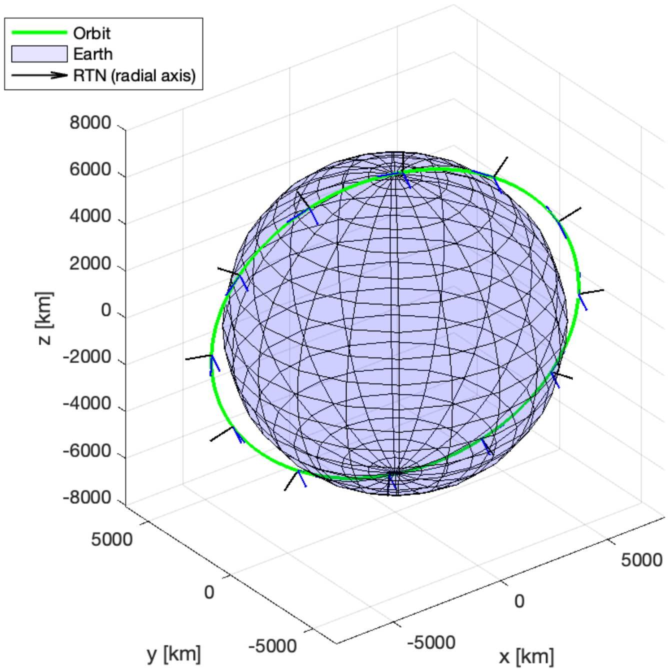


Figure 29: Propagation of RTN frame

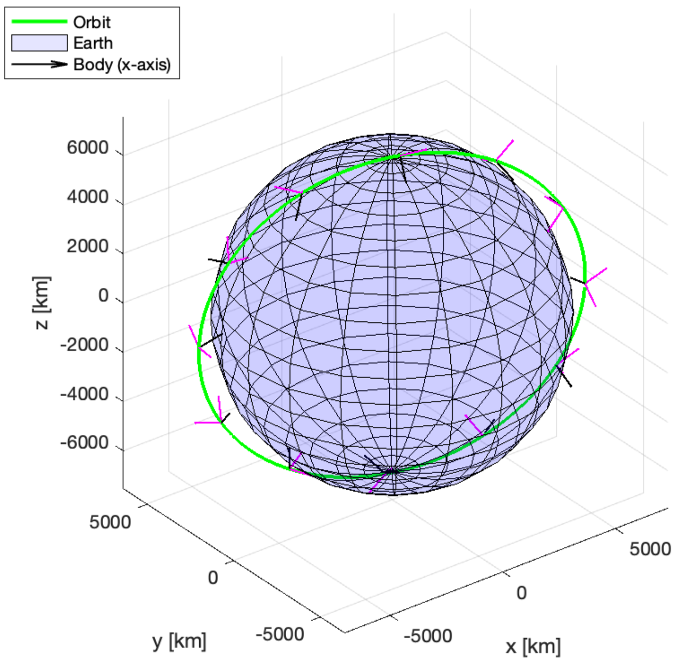


Figure 30: Propagation of body axes

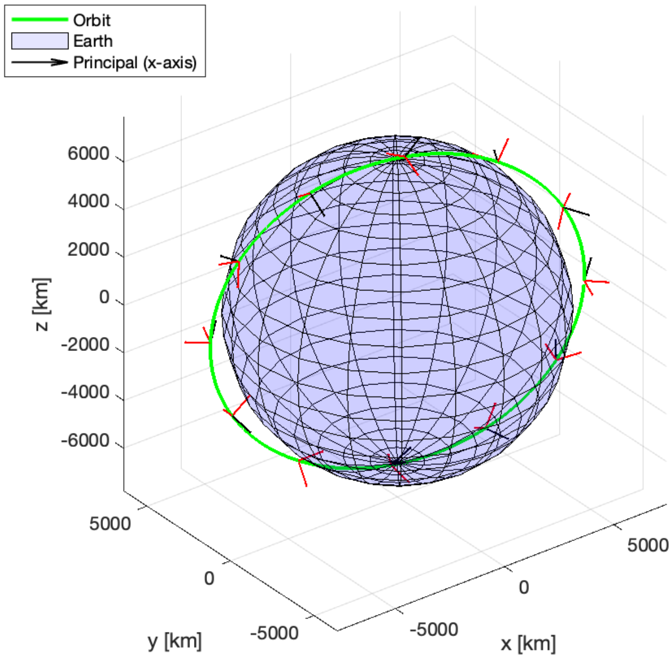


Figure 31: Propagation of principal axes

4 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.

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 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(@propagator,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(@propagator,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

```
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 A = q2A(q)
    qv = q(1:3)';
    q4 = q(4);
    qx = [0, -qv(3), qv(2); ...
        qv(3), 0, -qv(1); ...
        -qv(2), qv(1), 0];
    A = (q4^2 - norm(qv)^2) * eye(3) + 2 * (qv * qv') - 2 * q4 * qx;
end
```

```
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
```